Uncertainty Representations in State-Space Layers for Deep Reinforcement Learning under Partial Observability

Anonymous authors
Paper under double-blind review

Abstract

Optimal decision-making under partial observability requires reasoning about the uncertainty of the environment's hidden state. However, most reinforcement learning architectures handle partial observability with sequence models that have no internal mechanism to incorporate uncertainty in their hidden state representation, such as recurrent neural networks, deterministic state-space models and transformers. Inspired by advances in probabilistic world models for reinforcement learning, we propose a standalone Kalman filter layer that performs closed-form Gaussian inference in linear state-space models and train it end-to-end within a model-free architecture to maximize returns. Similar to efficient linear recurrent layers, the Kalman filter layer processes sequential data using a parallel scan, which scales logarithmically with the sequence length. By design, Kalman filter layers are a drop-in replacement for other recurrent layers in standard model-free architectures, but importantly they include an explicit mechanism for probabilistic filtering of the latent state representation. Experiments in a wide variety of tasks with partial observability show that Kalman filter layers excel in problems where uncertainty reasoning is key for decision-making, outperforming other stateful models.

1 Introduction

The classical reinforcement learning (RL) formulation tackles optimal decision-making in a fully observable Markov Decision Process (MDP) (Sutton and Barto, 2018). However, many real-world problems are *partially* observable, since we only have access to observations that hide information about the state, e.g., due to noisy measurements. Learning in partially observable MDPs (POMDPs) is statistically and computationally intractable in general (Papadimitriou and Tsitsiklis, 1987), but in many practical scenarios it is theoretically viable (Liu et al., 2022) and has lead to successful applications in complex domains like robotics (Zhu et al., 2017), poker (Brown and Sandholm, 2019), real-time strategy games (Vinyals et al., 2019) and recommendation systems (Li et al., 2010).

Practical algorithms for RL in POMDPs employ sequence models that encode the history of observations and actions into a latent state representation amenable for policy optimization. Besides extracting task-relevant information from the history, probabilistic inference over the latent state is also crucial under partial observability (Kaelbling et al., 1998). As a motivating example, consider an AI chatbot that gives restaurant recommendations to users. Since the user's taste (i.e., the state) is unknown, the agent must ask questions before ultimately making its recommendation. Reasoning over the latent state uncertainty is crucial to decide whether to continue probing the user or end the interaction with a final recommendation. An optimal agent would gather enough information to recommend a restaurant with a high likelihood of user satisfaction. In Section 5.2, we evaluate performance of our proposed approach in a simplified version of this problem.

A standard recipe for model-free RL in POMDPs is to combine a sequence model (e.g., LSTM (Hochreiter and Schmidhuber, 1997), GRU (Cho et al., 2014)) with a policy optimizer (e.g., PPO (Schulman et al., 2017) or SAC (Haarnoja et al., 2018)), which has shown strong performance in a wide variety of POMDPs (Ni et al., 2022). More recently, transformers (Vaswani et al., 2017) have also been adopted as sequence models in RL showing improved memory capabilities (Ni et al., 2023). However, their inference runtime scales quadratically with the sequence length, which makes them unsuitable for online learning in physical systems (Parisotto and Salakhutdinov, 2020). Instead, recent deterministic state-space models (SSMs) (Gu et al., 2022a; Smith et al., 2023; Gu and Dao, 2023) maintain the constant-time inference of stateful models, while achieving logarithmic runtime during training thanks to efficient parallel scans (Smith et al., 2023). Moreover, SSMs have shown improved long-term memory, in-context learning and generalization in RL (Lu et al., 2023). Yet, in problems where reasoning over latent state uncertainty is crucial, it remains unclear whether such methods can learn the required probabilistic inference mechanisms for decision making.

While model-free architectures focus on *deterministic* sequence models, in model-based RL *probabilistic* sequence models are a widespread tool to model uncertainty in environment dynamics (Hafner et al., 2019; 2020; Becker and Neumann, 2022). Considering these two sequence modelling approaches, the purpose of this work is to investigate the following questions:

> *Can we leverage the same inference methods developed for model-based RL as general-purpose sequence models in model-free architectures? If so, does it bring any benefits compared to deterministic models?*

Our core hypothesis is that explicit probabilistic inference in sequence models may serve as an inductive bias to learn in tasks where uncertainty over the latent state is crucial for decision making, as our previous example on the restaurant recommendation chatbot.

Our Contributions. Inspired by the simple inference scheme in the Recurrent Kalman Network (RKN) (Becker et al., 2019) architecture for world models, we embed closed-form Gaussian inference in linear SSMs as a standalone recurrent layer — denoted a Kalman filter (KF) layer — and train it end-to-end within a model-free architecture (Ni et al., 2022) to maximize returns. Since our KF layers are designed to be a drop-in replacement for standard recurrent layers, they can also be stacked together and combined with other components (e.g., residual connections, normalization, etc.) to build more complex sequence models. To the best of our knowledge, this is the first work that empirically evaluates the performance of such probabilistic inference layers in a model-free RL architecture. We conduct extensive experiments with a variety of sequence models over a wide range of POMDPs and show that KF layers excel in tasks where probabilistic inference is key for decision-making, with significant improvements over deterministic stateful models.

2 Related Work

RL architectures for POMDPs. In order to deal with partial observability, RL agents are typically equipped with some form of *memory* system, e.g., based on human psycology (Fortunato et al., 2019) or context-dependent retrieval (Oh et al., 2016). The most widespread memory system in the RL literature is through *sequence models*, also known as *history encoders* (Ni et al., 2024), whose purpose is to encode the past history into a state representation useful for RL. These sequence models can augment policies (Wierstra et al., 2007), value functions (Schmidhuber, 1990; Bakker, 2001) and/or world models (Schmidhuber, 1991; Becker et al., 2019; Shaj et al., 2021a;b; 2023), which allows handling of partial observability in previous RL algorithms such as DQN (Hausknecht and Stone, 2015), SAC (Ni et al., 2022), PPO (Kostrikov, 2018; Ni et al., 2023; Lu et al., 2023), DPG (Heess et al., 2015) and Dyna (Hafner et al., 2020; Becker and Neumann, 2022). In this work, we adopt an off-policy model-free architecture similar to Ni et al. (2022), which showed strong performance in various POMDPs.

Sequence models in RL. Frame-stacking is one of the earliest sequence models used in RL (Lin and Mitchell, 1993) and it is still a common tool to convey velocity information from image-based observations, like in the Atari benchmark (Bellemare et al., 2013; Mnih et al., 2013). However, it fails to model long-range dependencies in more complex POMDPs. Stateful recurrent models are the most widespread sequence models in RL to extract relevant information from arbitrarily long contexts, for instance RNNs (Lin and Mitchell, 1993; Schmidhuber, 1990) LSTMs (Bakker, 2001) and GRUs (Kostrikov, 2018). More recently, the transformer architecture (Vaswani et al., 2017) has shown promising results in improving the long-term memory of RL agents (Ni et al., 2023). However, their slow inference and large memory requirements reduce their practicality as real-time control systems (Parisotto and Salakhutdinov, 2020).

Deterministic SSMs. Advances in SSMs are of particular interest to the RL community, since they maintain the fast inference of RNNs but scale better during training via parallel scans (Smith et al., 2023), can circumvent vanishing/exploding gradients with proper initialization (Gu et al., 2020) and match (or even exceed) the performance of transformers in long-range sequence modelling tasks (Lu et al., 2023). In particular, *structured* state space models such as S4 (Gu et al., 2022a), S5 (Smith et al., 2023) and S6 / Mamba (Gu and Dao, 2023) have emerged as a strong competitor to transformers in general sequence modelling problems like language (Fu et al., 2023), audio (Goel et al., 2022) and video (Nguyen et al., 2022). The adoption of these models in RL is still in its infancy, however. Morad et al. (2023) report bad performance of a variant of S4 (Gu et al., 2022b) in various POMDPs. Lu et al. (2023) show strong performance in long-term memory and in-context learning by combining S5 with PPO. Besides these contradictory results, it remains unclear how these models perform in tasks where uncertainty of the latent state plays a vital role in decision-making since they are not equipped with explicit probabilistic inference mechanisms.

Probabilistic SSMs. Probabilistic stateful models are the backbone of world models (Kalweit and Boedecker, 2017; Ha and Schmidhuber, 2018), trained to predict forward dynamics and rewards which can then be used for: (i) planning (Hafner et al., 2019) or policy optimization (Becker and Neumann, 2022; Hafner et al., 2020) via latent imagination (i.e., generate imaginary policy rollouts auto-regressively), or (ii) policy optimization on the learned latent representation (Becker et al., 2024). A prominent approach is the Recurrent State Space Model (RSSM) proposed by Hafner et al. (2019), which divides the latent state into deterministic and stochastic components and uses a GRU for propagating forward the deterministic part. More recently, GRUs have been replaced by transformers (Chen et al., 2021) and S4 (Samsami et al., 2024) models, albeit in a simplified inference scheme that only conditions on the current observation rather than the history. A common concern with these model-based approaches is the objective mismatch (Lambert et al., 2020) between learning accurate world models and training performant control policies.

Inference in Linear SSMs. Closest to our approach are models that perform closed-form probabilistic filtering in *linear* SSMs. Inference in linear SSMs is a well-studied problem in the dynamical systems community, dating back to the seminal work by Kalman (1960). While Kalman filters and smoothers offer tractable, closed-form inference, the linear-Gaussian assumption is typically not suited for high-dimensional, multi-modal data (Murphy, 2012; Bishop, 2006). Recent methods leverage neural networks to project data into learned latent spaces where the Kalman filter assumptions are less restrictive, leading to expressive probabilistic models of high-dimensional data such as images (Haarnoja et al., 2016; Becker et al., 2019; Shaj et al., 2021a;b; 2023). The Recurrent Kalman Network (RKN) proposed by Becker et al. (2019) is an encoder-decoder architecture that employs Kalman filtering using locally linear models and structured (non-diagonal) covariance matrices. Follow-up work has extended the RKN framework in various ways: Shaj et al. (2021a) include action conditioning, Shaj et al. (2021b) consider a multi-task setting with hidden task parameters, Shaj et al. (2023) propose a hierarchical, multi-timescale architecture and Becker and Neumann (2022) replace the closed-form filtering with a variational approach that also performs smoothing, leading to a tight variational bound. Concurrent work by Becker et al. (2024) proposes a world model that leverages a Mamba (Gu and Dao, 2023) backbone to learn a linear dynamics model and a time-parallel Kalman smoother trained on a variational inference loss; the learned state representation is then used with a SAC policy optimizer.

In this work, we propose a model-free RL architecture for POMDPs, similar to Ni et al. (2022), where deterministic recurrent layers (e.g., RNNs or transformers) are replaced with KF layers that leverage the simplified filtering scheme from RKNs. In contrast to prior work proposing RKN-based architectures for model-based RL (Becker and Neumann, 2022; Becker et al., 2024), in our model-free RL architecture KF layers are trained end-to-end to maximize returns instead of auxiliary world model objectives. Consequently, our training paradigm erases concerns of objective mismatch during representation learning (Lambert et al., 2020). Moreover, we design KF layers as standalone recurrent layers that can be stacked and combined with other operations (e.g., residual connections and normalization) to build more complex architectures. Similar to Becker et al. (2024), the associative property of the Kalman filter operations allows efficient training of KF layers via parallel scans, which scale logarithmically with the sequence length provided sufficient parallel GPU cores.

3 Background

In this section, we provide the relevant background and introduce core notation used throughout the paper. We use bold upper case letters ($\mathbf{A}$) to denote matrices and calligraphic letters ($\mathcal{X}$) to denote sets. The notation $\mathcal{P}(\mathcal{X})$ referes to the space of probability distributions over $\mathcal{X}$.

3.1 Reinforcement Learning in Partially Observable Markov Decision Processes

We consider an agent that acts in a finite-horizon partially observable Markov decision process (POMDP) $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{O}, T, p, O, r, \gamma\}$ with state space $\mathcal{S}$, action space $\mathcal{A}$, observation space $\mathcal{O}$, horizon $T \in \mathbb{N}$, transition function $p : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ that maps states and actions to a probability distribution over $\mathcal{S}$, an emission function $O : \mathcal{S} \to \mathcal{P}(\mathcal{O})$ that maps states to a probability distribution over observations, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and a discount factor $\gamma \in [0, 1)$.

At time step t of an episode in $\mathcal{M}$, the agent observes $o_t \sim O(\cdot \mid s_t)$ and selects an action $a_t \in \mathcal{A}$ based on the observed history $h_{:t} = (o_{:t}, a_{:t-1}) \in \mathcal{H}_t$, then receives a reward $r_t = r(s_t, a_t)$ and the next observation $o_{t+1} \sim O(\cdot \mid s_{t+1})$ with $s_{t+1} \sim p(\cdot \mid s_t, a_t)$.

We adopt the general setting by Ni et al. (2023; 2024), where the RL agent is equipped with: (i) a stochastic policy $\pi : \mathcal{H}_t \to \mathcal{P}(\mathcal{A})$ that maps from observed history to distribution over actions, and (ii) a value function $Q^\pi : \mathcal{H}_t \times \mathcal{A} \to \mathbb{R}$ that maps from history and action to the expected return under the policy, defined as $Q^\pi(h_{:t}, a_t) = \mathbb{E}_\pi\left[\sum_{h=t}^{T} \gamma^{h-t} r_t \mid h_{:t}, a_t\right]$. The objective of the agent is to find the optimal policy that maximizes the value starting from some initial state s_0, $\pi^\star = \operatorname{argmax}_\pi \mathbb{E}_\pi\left[\sum_{t=0}^{T-1} \gamma^t r_t \mid s_0\right]$.

3.2 History Representations

A weakness of the general formulation of RL in POMDPs is the dependence of both the policy and the value function on the ever-growing history. Instead, practical algorithms fight this curse of dimensionality by *compressing* the history into a compact representation. Ni et al. (2024) propose to learn such representations via *history encoders*, defined by a mapping $\phi : \mathcal{H}_t \to \mathcal{Z}$ from observed history to some latent representation $z_t := \phi(h_{:t}) \in \mathcal{Z}$. With slight abuse of notation, we denote $\pi(a_t \mid z_t)$ and $Q^\pi(z_t, a_t)$ as the policy and values under this latent representation, respectively. In this paper, we propose a history encoder implemented via Kalman filtering layers that perform simple probabilistic inference on the latent state.

3.3 Structured State Space Models

We consider time-varying linear SSMs defined by

$$\dot{x}(t) = \mathbf{A}_t x(t) + \mathbf{B}_t u(t), \quad y(t) = \mathbf{C}_t z(t) + \mathbf{D}_t u(t), \tag{1}$$

where $t > 0 \in \mathbb{R}$, $x(t) \in \mathbb{R}^N$ is the hidden or latent state, $u(t) \in \mathbb{R}^P$ is the input, $y(t) \in \mathbb{R}^M$ is the output and $(\mathbf{A}_t, \mathbf{B}_t, \mathbf{C}_t, \mathbf{D}_t)$ are matrices of appropriate size. Such a continuous-time system can be discretized (e.g., using zero-order hold) for some step size Δ, which results in a linear recurrent model

$$x_k = \bar{\mathbf{A}}_k x_{k-1} + \bar{\mathbf{B}}_k u_{k-1}, \quad y_k = \bar{\mathbf{C}}_k x_k + \bar{\mathbf{D}}_k u_{k-1}. \tag{2}$$

As it is common in practice, we set $\bar{\mathbf{D}}_k \equiv \mathbf{0}$. In this paper, we consider *structured* SSMs, which simply means special structure is imposed into the learnable matrices $(\bar{\mathbf{A}}_k, \bar{\mathbf{B}}_k, \bar{\mathbf{C}}_k)$. In particular, we consider a diagonal structure with the HiPPO initialization proposed in Gu et al. (2020), which induces stability in the recurrence for handling long sequences.

3.4 Probabilistic Inference on Linear SSMs

To introduce uncertainty into state-space models, we consider a standard linear-Gaussian SSM

$$x_k = \bar{\mathbf{A}}_k x_{k-1} + \bar{\mathbf{B}}_k u_{k-1} + \varepsilon_k, \quad y_k = \bar{\mathbf{C}}_k x_k + \nu_k, \tag{3}$$

where $\varepsilon_k \sim \mathcal{N}(0, \mathbf{\Sigma}_k^{\mathrm{p}})$ and $\nu_k \sim \mathcal{N}(0, \mathbf{\Sigma}_k^{\mathrm{o}})$ are zero-mean process and observation noise variables with their covariance matrices $\mathbf{\Sigma}_k^{\mathrm{p}}$ and $\mathbf{\Sigma}_k^{\mathrm{o}}$, respectively. The latent state probabilistic model is then $p(x_k \mid x_{k-1}, u_{k-1}) = \mathcal{N}(\bar{\mathbf{A}}_k x_{k-1} + \bar{\mathbf{B}}_k u_{k-1}, \mathbf{\Sigma}_k^{\mathrm{p}})$ and the observation model is $p(y_k \mid x_k) = \mathcal{N}(\bar{\mathbf{C}}_k x_k, \mathbf{\Sigma}_k^{\mathrm{o}})$. Inference in such a model has a closed-form solution, which is equivalent to the well-studied Kalman filter (Kalman, 1960).

Predict. The first stage of the Kalman filter propagates forward the *posterior* belief of the latent state at step $k-1$, given by $\mathcal{N}(x_{k-1}^{+}, \mathbf{\Sigma}_{k-1}^{+})$, to obtain a *prior* belief at step k, $\mathcal{N}(x_k^{-}, \mathbf{\Sigma}_k^{-})$, given by

$$x_k^{-} = \bar{\mathbf{A}}_k x_{k-1}^{+} + \bar{\mathbf{B}}_k u_{k-1}, \quad \mathbf{\Sigma}_k^{-} = \bar{\mathbf{A}}_k \mathbf{\Sigma}_{k-1}^{+} \bar{\mathbf{A}}_k^{\top} + \mathbf{\Sigma}_k^{\mathrm{p}}. \tag{4}$$

Update. The second stage updates the prior belief at step k given some observation w_k, to obtain the posterior $p(x_k \mid x_{k-1}, w_k) = \mathcal{N}(x_k^{+}, \Sigma_k^{+})$ given by

$$x_k^{+} = x_k^{-} + \mathbf{K}_k(w_k - \bar{\mathbf{C}}_k x_k^{-}), \quad \mathbf{\Sigma}_k^{+} = (\mathbf{I} - \mathbf{K}_k \bar{\mathbf{C}}_k)\mathbf{\Sigma}_k^{-}, \tag{5}$$

where $\mathbf{K}_k = \mathbf{\Sigma}_k^{-} \bar{\mathbf{C}}_k^{\top} (\bar{\mathbf{C}}_k \mathbf{\Sigma}_k^{-} \bar{\mathbf{C}}_k^{\top} + \mathbf{\Sigma}_k^{\mathrm{o}})^{-1}$ is known as the Kalman gain. The predict and update steps are interleaved to process sequences of input and observations $\{u_k, w_k\}_{k=0}^{K-1}$ of length K, starting from some initial belief $\mathcal{N}(x_{-1}^{+}, \mathbf{\Sigma}_{-1}^{+})$. While (4) and (5) include expensive matrix operations, they simplify to cheap element-wise operations under structured SSMs (e.g., diagonal), as done in prior work (Becker et al., 2019; Becker and Neumann, 2022).

3.5 Parallel Scans

Efficient implementation of state-space models and Kalman filters employ *parallel scans* to achieve logarithmic runtime scaling with the sequence length (Smith et al., 2023; Sarkka and Garcia-Fernandez, 2021). Given a sequence of elements $(a_0, a_1, \dots, a_{K-1})$ and an *associative*[1] binary operator $\bullet$, the parallel scan algorithm outputs all the prefix-sums $(a_0, a_0 \bullet a_1, \dots, a_0 \bullet \dots \bullet a_{K-1})$ in $\mathcal{O}(\log K)$ runtime, given sufficient parallel processors.

4 Method: Off-Policy Recurrent Actor-Critic with Kalman filter Layers

In this section, we describe our method that implements Kalman filtering as a recurrent layer within a standard actor-critic architecture.

4.1 General Architecture

In Figure 1 we present our Recurrent Actor-Critic (RAC) architecture inspired by Ni et al. (2022), where we replace the RNN blocks with general history encoders. We will use this architecture in the following to test the capabilities of different history encoders in various POMDPs.

For both actor and critic, we embed the sequence of observations and actions into a single representation $h_{:t}^{*}$ which is then passed into the history encoders. We use a single linear layer as embedder, which we found worked as reliably as more complex non-linear embedders used in similar RAC architectures by Morad et al. (2023); Ni et al. (2022). We also include the skip connections from current observations and actions into the actor-critic heads, as proposed in previous memory-based architectures (Zintgraf et al., 2021; Ni et al., 2022).

4.2 Kalman Filter Layers

Our main hypothesis is that principled probabilistic filtering within history encoders boosts performance in POMDPs, especially those where reasoning about uncertainty is key for decision-making. To test this hypothesis, we introduce KF layers, as shown in Figure 2. The layer receives as input a history embedding sequence $h_{:t}$ which is then projected into the input $u_{:t}$, observation $w_{:t}$ and observation noise $\mathbf{\Sigma}_{:t}^{\mathrm{o}}$ sequences. These three signals serve as input to the standard KF predict-update equations (4) and (5), which output a posterior (filtered) latent state $x_{:t}^{+}$. Finally, the posterior sequence is projected back to the history embedding space to produce the compressed history representation $z_{:t}$.

---

[1] A binary operator $\bullet$ is associative if $(a \bullet b) \bullet c = a \bullet (b \bullet c)$ for any triplet of elements (a, b, c)

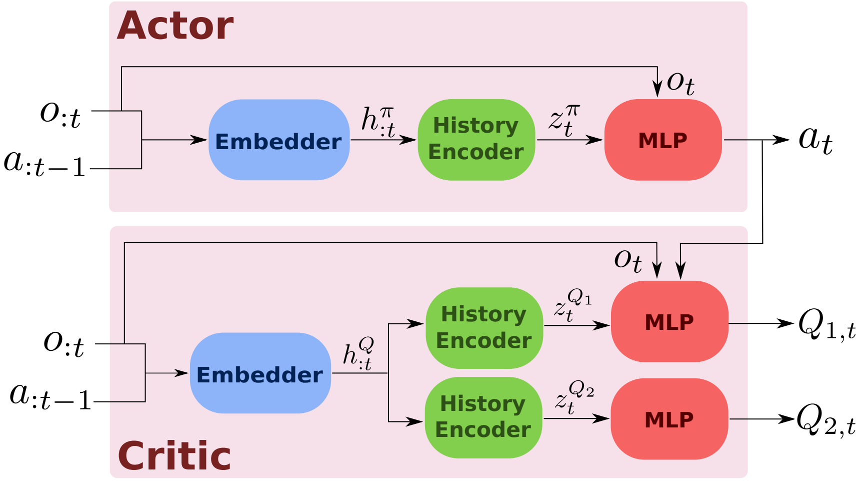

Figure 1: General Recurrent Actor-Critic (RAC) architecture. The components are trained end-to-end with the Soft Actor-Critic (SAC) loss function (Haarnoja et al., 2018). To handle discrete action spaces, we use the discrete version of SAC by Christodoulou (2019).

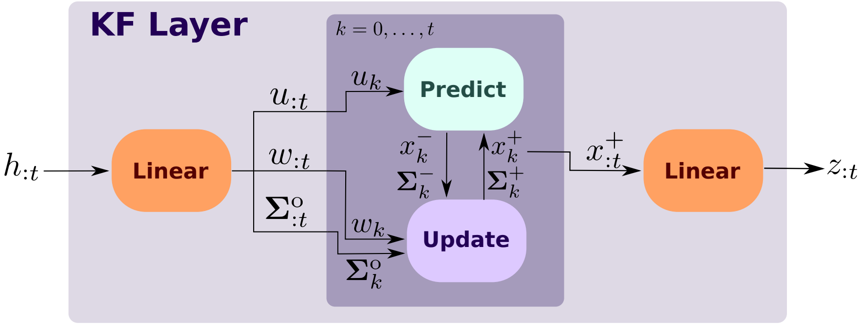
KF Layer
$k = 0, \ldots, t$
$h_{:t}$
Linear
$u_{:t}$
$w_{:t}$
$\boldsymbol{\Sigma}^{\mathrm{o}}_{:t}$
u_k
Predict
x^-_k
$\boldsymbol{\Sigma}^-_k$
x^+_k
$\boldsymbol{\Sigma}^+_k$
w_k
$\boldsymbol{\Sigma}^{\mathrm{o}}_k$
Update
$x^+_{:t}$
Linear
$z_{:t}$

Figure 2: Our proposed Kalman filter layer to build history encoders. The KF layer receives a history sequence $h_{:t}$ and projects it into three separate signals in latent space: the input $u_{:t}$, the observation $w_{:t}$ and the observation noise (diagonal) covariance $\boldsymbol{\Sigma}^{\mathrm{o}}_{:t}$. These sequences are processed using the standard Kalman filtering equations, which scale logarithmically with the sequence length using parallel scans. Lastly, the posterior mean latent state $x^+_{:t}$ is projected from the latent space back into the history space to obtain the compressed representation $z_{:t}$.

History encoders with KF layers. Similar to recent SSM layers such as S5 (Smith et al., 2023) and S6 (Gu and Dao, 2023), these KF layers can be stacked and combined with other operations such as residual connections, gating mechanisms, convolutions and normalization to compose a history encoder block in the RAC architecture. In favor of simplicity, our history encoders are only composed of KF layers and (optionally) an RMS normalization (Zhang and Sennrich, 2019) output block for improved stability.

Filtering as a gating mechanism. We can draw interesting comparisons between KF layers and other recurrent layers from the perspective of gating mechanisms. It was shown in Theorem 1 of (Gu and Dao, 2023) that selective SSMs (S6) behave as generalized RNN gates through an input-dependent step size Δ. In this case, the gate depends on the SSM input and controls how much the input influences the next hidden state. Similarly, as hinted by Becker et al. (2019), during the update step the Kalman gain is effectively an uncertainty-controlled gate depending on the observation noise which regulates how much the observation influences the posterior belief over the latent state. Our experiments in Section 5 shed some light on the strengths and weaknesses of these approaches for RL under partial observability.

Implementation details. We simplify the implementation of KF layers and eliminate expensive matrix operations via a diagonal-structured SSM with matching input-latent-observation dimensions, i.e., $N = P = M$.

In this setting, we directly learn the diagonal components of time-independent matrices $\mathbf{A}$ and $\mathbf{B}$ (no diagonalization step as done in the S5 layer (Smith et al., 2023)), while we fix $\mathbf{C}$ to be the identity matrix, as it is common in prior work (Becker and Neumann, 2022). Second, we consider a time-invariant process noise with learnable diagonal covariance matrix $\mathbf{\Sigma}^{\mathrm{p}}$. The SSM is discretized using the zero-order hold method and consider a learnable scalar step size $\Delta > 0$ (Smith et al., 2023).

Design decisions. We want to highlight two considerations that went into the design of our KF layers. First, we could generalize the architecture to support time-varying process noise by including one extra output channel (alongside the input, observation and observation noise channels) in the history linear projection. Conceptually, such an input-dependent process noise adds more flexibility to the gating mechanism implemented within the KF layer, which would be controlled both by the observation and the process noise signals. Second, we could include the posterior covariance $\mathbf{\Sigma}_{:t}^{+}$ as an additional feature for the output linear projection, alongside the posterior mean $x_{:t}^{+}$. We conduct an ablation study over these two choices in several continuous control tasks subject to observation noise and report the results in Appendix E. The best aggregated performance in this ablation was obtained with time-invariant process noise and only using the posterior mean as a feature for the output projection, which empirically justifies our final design.

4.3 Masked Associative Operators for Variable Sequence Lengths

In off-policy RAC architectures, the agent is typically trained with batches of (sub-)trajectories of possibly different length, sampled from an experience replay buffer. Thus, history encoders must be able to process batches of variable sequence length during training.

A common approach is to right-pad the batch of sequences up to a common length and ensure the model's output is independent of the padding values. For transformer models, this can be achieved by using the padding mask as a self-attention mask. For stateful models like RNNs and SSMs, it is imperative to also output the correct final latent state for each sequence in the batch. This typically requires a post-processing step that individually selects for each sequence in the batch the last state before padding. It turns out that for any recurrent model expressed with an associative operator (e.g., SSMs and KFs), we can obtain the correct final state from a batch of padded sequences *without* additional post-processing by using a parallel scan routine with a *Masked Associative Operator* (MAO).

Definition 1 (Masked Associative Operator)**.** Let $\bullet$ be an associative operator acting on elements $e \in \mathcal{E}$, such that for any $a, b, c \in \mathcal{E}$, it holds that $(a \bullet b) \bullet c = a \bullet (b \bullet c)$. Then, the MAO associated with $\bullet$, denoted $\tilde{\bullet}$, acts on elements $\tilde{e} \in \mathcal{E} \times \{0,1\} = (e, m)$, where $m \in \{0,1\}$ is a binary mask. Then, for $\tilde{a} = (a, m_a)$ and $\tilde{b} = (b, m_b)$, we have:

$$\tilde{a} \,\tilde{\bullet}\, \tilde{b} = \begin{cases} (a \bullet b, m_a) & \text{if} \quad m_b = 0 \\ \tilde{a} & \text{if} \quad m_b = 1 \end{cases} \tag{6}$$

In Appendix A, we show that any MAO is itself *associative* as long as we apply a right-padding mask[2], thus fulfilling the requirement for parallel scans. In practice, augmenting existing SSM and KF operators with their MAO counterpart is a minor code change. MAOs act as a pass-through of the hidden state when padding is applied, thus yielding the correct state at every time step of the padded sequence for each element of the batch without additional indexing or bookkeeping. Due to their pass-through nature, MAOs require strictly equal or less evaluations of the underlying associative operator, which may yield faster runtimes if the operator is expensive to evaluate and/or many elements of the input sequence are masked.

MAOs for SSMs and KFs. As a concrete example, the associative operators for SSMs and KFs involve matrix product and addition. A compute-efficient implementation of MAOs for such operators involves sparse matrix operations, where the sparsity is dictated by the padding mask. However, sparse matrix operations are only expected to yield better runtime than their dense counterparts for large matrices with sufficient levels of sparsity, which are not typical in our application. Thus, no speed-up is expected from using MAOs in the context of this work.

---

[2] A right-padding mask is a sequence $\{m_0, m_1, \dots\}$ with $m_i \in \{0,1\}$ such that if $m_i = 1$ then $m_j = 1$ for all $j > i$.

MAOs are similar to the custom operator proposed by Lu et al. (2023), but their effect is fundamentally different: Lu et al. (2023) considers on-policy RL, where the goal is to handle multi-episode sequences, thus their custom operator resets the hidden state at episode boundaries. Instead, in our off-policy RAC architecture, MAOs act as pass-through of the hidden state for padded inputs.

5 Experiments

In this section, we evaluate the RAC architecture under different history encoders in various POMDPs. Implementation details and hyperparameters are included in Appendices B and C, respectively.

5.1 Baselines

We consider the following implementation of history encoders within the RAC architecture.

`vSSM`. Vanilla, real-valued SSM with diagonal matrices. It is equivalent to a KF layer with infinite observation noise, i.e., the update step has no influence on the output. It can also be seen as a simplification of the S4D model (Gu et al., 2022b), where states are real-valued rather than complex (as in Mega (Ma et al., 2023), such that the recurrence can be interpreted as an exponential moving average) and the recurrence is implemented with a parallel scan rather than a convolution (as in (Smith et al., 2023)).

`vSSM+KF`. Probabilistic SSM via the KF layers described in Figure 2, which is equivalent as adding Kalman filtering on top of `vSSM`.

`vSSM+KF-u`. Equivalent to `vSSM+KF` *without* the input signal $u_{:t}$. It maintains the uncertainty-based gating from the KF layer, but looses flexibility in the KF predict step to influence the prior belief via the input.

`Mamba` (Gu and Dao, 2023). Selective state-space model with input-dependent state transition matrices.

`GRU` (Cho et al., 2014). Stateful model with a gating mechanism and non-linear state transitions.

`vTransformer` (Vaswani et al., 2017). Vanilla encoder-only transformer model with sinusoidal positional encoding and causal self-attention.

All SSM-based approaches are implemented using MAOs and parallel scans. Besides these memory-based agents, we include two additional memoryless agents that implement the same RAC architecure but without embedders or history encoders.

`Oracle`. It has access to the underlying state of the environment, effectively removing the partial observability aspect of the problem. This method should upper-bound the performance of history encoders.

`Memoryless`. Unlike `Oracle`, it does not have access to the underlying state of the environment. This method should lower-bound the performance of history encoders.

All the baselines share a common codebase and hyperparameters. For all stateful models, we use the same latent state dimension N such that parameter count falls within a 10% tolerance range except for `GRU`, which naturally has more parameters due to its gating mechanism (roughly 40% increase). For `vTransformer` we choose the dimension of the feed-forward blocks such that the total parameter count is also within 10% of the SSM methods. With this controlled experimental setup, we aim to evaluate strengths and weaknesses of the different mechanisms for sequence modelling (gating, input-selectivity, probabilistic filtering, self-attention) in a wide variety of partially observable environments.

5.2 Probabilistic Reasoning - Adaptation and Generalization

We evaluate probabilistic reasoning capabilities with a carefully designed POMDP that simplifies our running example from Section 1, where an AI chatbot probes a user in order to recommend a restaurant. Given noisy observations (user's answers) sampled from a bandit with distribution $\mathcal{N}(\mu_b, \sigma_b)$ (user's preference), the task is to infer whether the mean μ_b lies above or below zero (binary decision between two restaurants A and B). At the start of each episode, μ_b and σ_b (the latent parameters) are sampled from some given distribution (i.e., we have a different user every episode). Then, at each step of an episode, the RL agent has three choices:

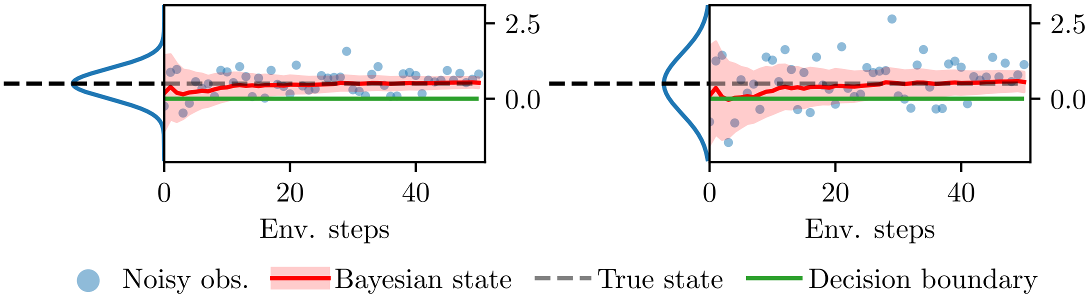

Figure 3: Two example episodes of the Best Arm Identification task of Section 5.2, with $\mu_b = 0.5$ and two different noise scales. **(Left)** Narrow noise distribution with $\sigma_b = 0.5$. **(Right)** Wide noise distribution with $\sigma_b = 1.0$. In red, we visualize the Bayesian posterior mean and 3σ confidence interval around μ_b, obtained via Bayesian linear regression using all prior observations in the episode.

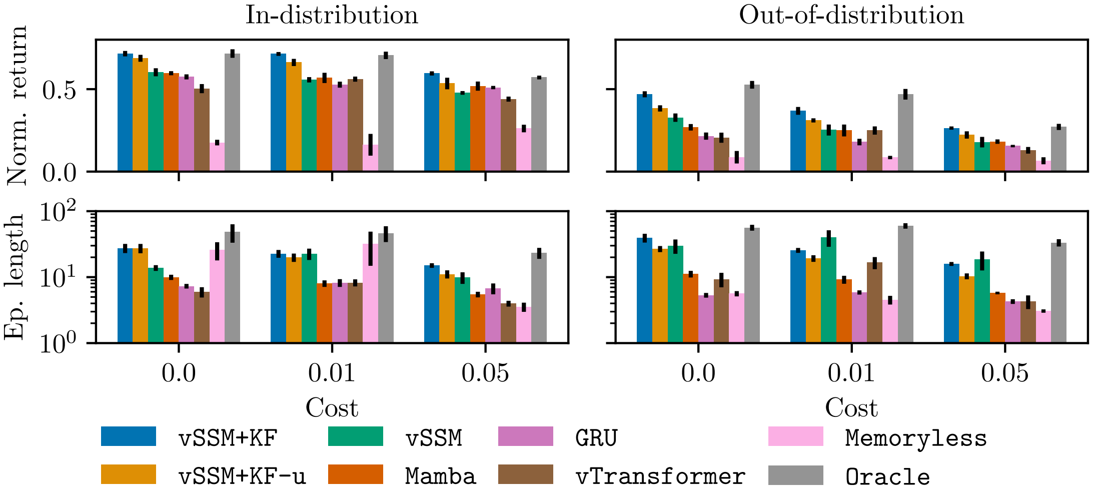

Figure 4: Performance of sequence models in the Best Arm Identification problem after 500K environment steps. We conduct experiments for increasing cost of requesting new observations and evaluate performance both in and out of distribution, averaged over 100 episodes, and report the mean and standard error over 5 random seeds. **(Top row)** Normalized return, obtained by dividing returns by the reward given after winning (10 in our case). **(Bottom row)** Length of episodes.

(1) request a new observation from the bandit (ask a new question to the user), which incurs a cost ρ, (2) decide the arm has mean above zero (recommend restaurant A) or (3) decide the arm has mean below zero (recommend restaurant B), both of which immediately end the episode and provide a positive reward if the decision was correct, or a negative reward if the decision was incorrect (i.e., reward is based on whether the recommendation matches the user's preference). We set a maximum episode length of 1000 steps; if the agent does not issue a decision by then, it receives the negative reward. Example rollouts for this environment are provided in Figure 3. Given the Bayesian state from Figure 3, an optimal agent must strike a balance between requesting new information (which reduces uncertainty about the estimated mean) and minimizing costs. Effective history encoders for this problem should similarly produce a state representation that encodes uncertainty about the latent parameters.

We evaluate two core capabilities: adaptation and generalization. Intuitively, an optimal policy for this problem must be *adaptive* depending on the latent parameters. For example, if μ_b is close to zero the agent might need many observations to make an informed decision, whereas with a large $|\mu_b|$ the correct decision can be made with few observations. Moreover, we can also evaluate *generalization* of the learned policy by

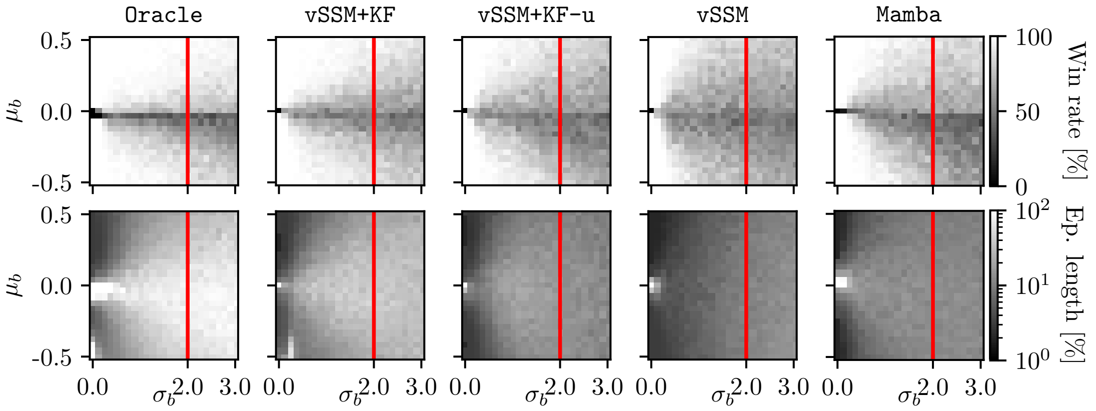

Figure 5: Performance heatmap on Best Arm Identification problem ($\rho = 0$). We generate a grid of noise parameters (μ_b, σ_b) for a total of 625 unique combinations. The red vertical line separates training (to the left) from out-of-distribution (to the right) latent parameters. For each pair of latent parameters, we evaluate performance on five independently trained agents over 100 episodes and report the average win rate and episode lengths.

testing on latent parameters not seen during training. Our hypothesis is that an agent that learns proper probabilistic reasoning (e.g., Bayes' rule) should generalize reasonably well in this task.

We conduct experiments for all baselines under increasing cost ρ. Instead of providing the latent parameters directly to the `Oracle` baseline, we provide the Bayesian posterior mean and standard deviation around the latent parameter μ_b, as shown in Figure 3. The agents are trained under the latent parameter distribution given by $\mu_b \sim \text{Unif}(-0.5, 0.5)$ and $\sigma_b \sim \text{Unif}(0.0, 2.0)$. We additionally evaluate out-of-distribution generalization by using $\sigma_b^{\text{OOD}} \sim \text{Unif}(2.0, 3.0)$, i.e., we test how the agent generalizes to bandits with higher variance. In Figure 4 we report the normalized return and average episode length for both the training and out-of-distribution latent parameters. Full training curves are included in Appendix D. `vSSM+KF` achieves the highest return out of the memory-based agents, both in and out-of-distribution, while matching the performance of `Oracle` in-distribution. The better performance of `vSSM+KF` correlates with longer episodes: compared to the other baselines, `vSSM+KF` learns to request more observations in order to issue a more informed decision.

`vSSM+KF` improves adaptation and generalization. To gain further insights on the results, we do a post-training evaluation on a subset of the agents across the entire latent parameter space, as shown in Figure 5. `vSSM+KF` learns adaptation patterns similar to `Oracle`: the length of episodes increase as the noise scale σ_b increases and decrease as $|\mu_b|$ increases, as it is intuitively expected. Such adaptation is less pronounced in `vSSM`, `vSSM+KF-u` and `Mamba`, where episodes are shorter and ultimately results in lower win rates. While `vSSM+KF` does not match the generalization performance of `Oracle`, it remains the best amongst the history encoder baselines. Given our controlled experimental setup, we attribute the enhanced adaptation and generalization of `vSSM+KF` to the internal probabilistic filtering implemented in the KF layer. Moreover, comparing `vSSM+KF` and `vSSM+KF-u` highlights that including the input signal in the KF layer leads to improved performance in this task.

`vSSM+KF` can handle adversarial episodes. In Figure 6 we compare latent space rollouts[3] from `vSSM+KF` and `vSSM` in an adversarial episode: μ_b is negative, but the first two observations are positive and of relatively large magnitude. After only four observations, `vSSM` is mislead by the positive observations and issues the wrong decision, as visualized in Figure 6 (middle) where we show the policy's output across latent space, overlaid with the rollout trajectory. Instead, `vSSM+KF` remains in the region where the policy requests more observations before it navigates to the correct region of latent space, as shown in Figure 6 (right). While this example was hand-picked, it is consistent with the adaptation patterns from Figure 5.

---

[3] We use a latent state dimension $N = 2$ in order to plot the policy decision boundary in latent space. This results in slightly worse performance than the results reported in Figure 4, where we use $N = 128$.

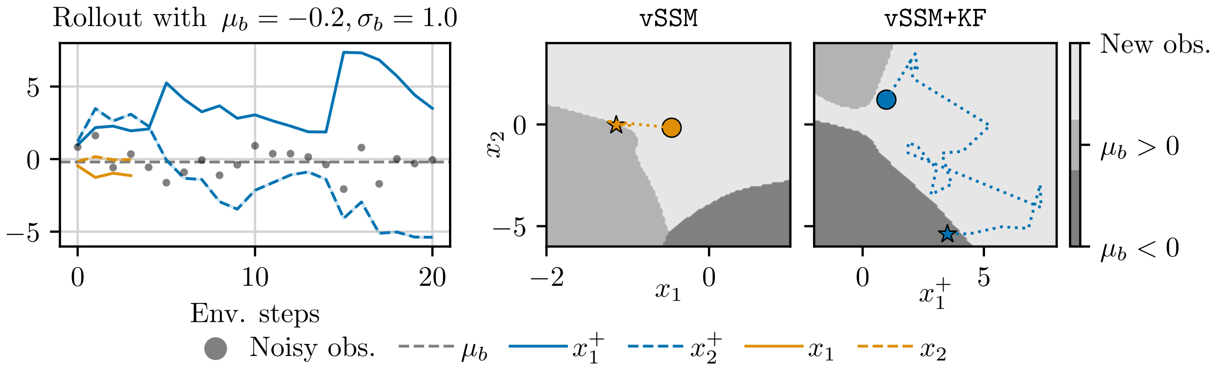

Figure 6: Latent space rollouts in adversarial Best Arm Identification episode. **(Left)** Rollout in latent space ($N = 2$) for `vSSM+KF` and `vSSM` after training. **(Middle-Right)** Policy decision boundaries overlaid with the latent space trajectory. Circles and stars denote the beginning and end of trajectories, respectively.

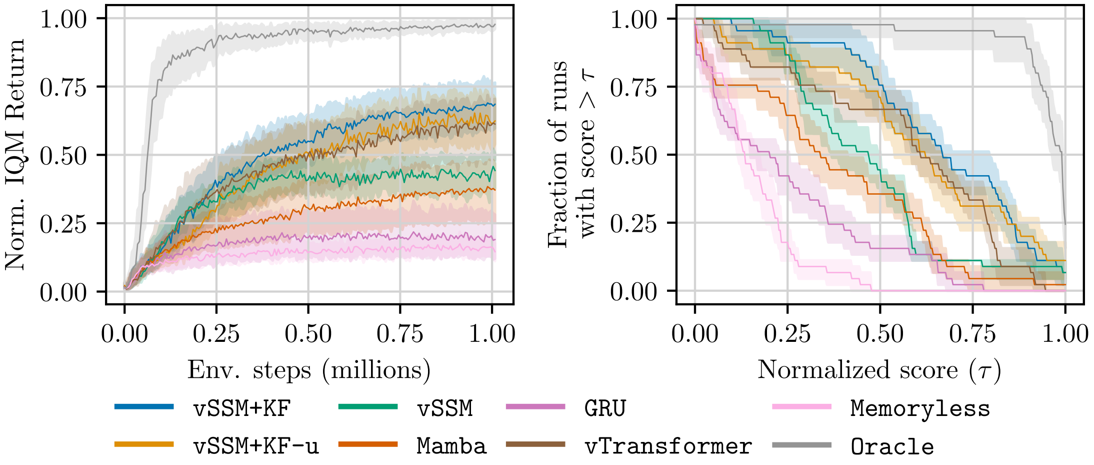

Figure 7: Aggregated performance in noisy DMC benchmark (9 tasks) with 95% bootstrap confidence intervals over five random seeds. **(Left)** Inter-quartile mean returns normalized by the score of `Oracle`. **(Right)** Performance profile after 1M environment steps. Higher curves correspond to better performance.

5.3 Probabilistic Filtering - Continuous Control under Observation Noise

In this experiment, we evaluate the ability to learn control policies subject to observation noise. Effective history encoders must learn to aggregate observations over multiple time steps to produce a filtered state representation amenable for control. Our hypothesis is that internal probabilistic filtering provides an inductive bias for learning such a filtered representation. To test our hypothesis, we conduct evaluations across nine environments from the DeepMind Control (DMC) suite (Tunyasuvunakool et al., 2020) with zero-mean Gaussian noise added to the observations, as done by Becker and Neumann (2022); Becker et al. (2024). We present aggregated performance in Figure 7 following the recommendations from (Agarwal et al., 2021). Detailed training curves are included in Appendix F. We now discuss the main insights from this experiment.

`vSSM+KF` improves performance of stateful models. The KF layer is the only evaluated add-on for stateful models that significantly improves performance over the baseline model `vSSM`. This suggests that the uncertainty-based gating in Kalman filters is more effective at handling noisy data compared to the gating mechanism implemented by `GRU` and `Mamba`. This observation matches the results in the Best Arm

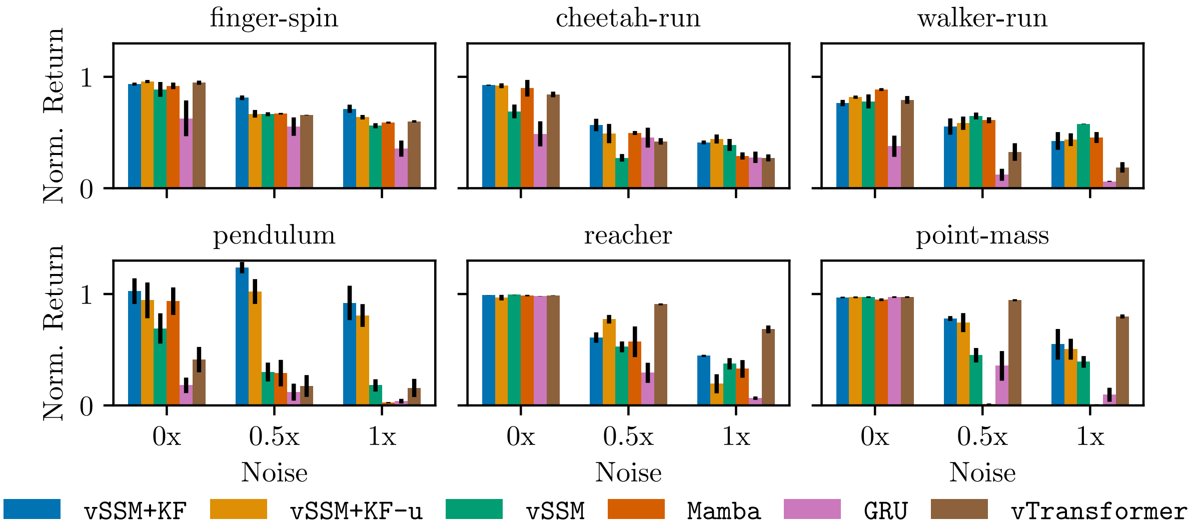

Figure 8: Final performance comparison of recurrent models in six tasks over increasing noise levels. We report the mean and standard error over five random seeds (ten for `pendulum` due to large variance) of the return after 1M environment steps, normalized by the score of `Oracle`.

Identification problem from Section 5.2. Comparing `vSSM+KF` and `vSSM+KF-u`, there is a slight improvement in performance from using an input signal in the KF layer, but it is not statistically significant.

`vSSM+KF` learns consistently across environments. From the detailed results in Figure 14, we observe that `vSSM+KF` consistently improves performance over the `Memoryless` lower-bound and achieves the best or comparable final performance in five out of nine tasks. Instead, `GRU`, `Mamba` and `vTransformer` completely fail to learn in some tasks, barely matching the performance of `Memoryless`.

We conduct an additional ablation over increasing noise levels in six representative tasks from the DMC suite, as shown in Figure 8. Training curves are included in Appendix G

`vSSM+KF` performs close to `Oracle` under full observability. We observe `vSSM+KF` generally matches the performance of `Oracle` in the absence of noise (normalized score close to 1.0), whereas `vSSM` and `vTransformer` significantly underperform in some tasks. This suggests that the added probabilistic filtering in `vSSM+KF` is a general-purpose strategy even under full observability.

`vSSM+KF`'s robustness to noise is environment dependent. Figure 8 suggests that robustness to noise depends generally on the environment, without any clear patterns related to task specifics. `vSSM+KF` is more robust in `finger-spin`, `cheetah-run` and `pendulum`[4], `vSSM` is more robust in `walker-run` but significantly underperforms in other environments, and `vTransformer` is more robuts in `reacher` and `point-mass` but fails to learn in `pendulum`. Overall, `vSSM+KF` shows the most consistent performance across environments and noise levels.

5.4 General Memory Capabilities

So far the evaluations were conducted in tasks where probabilistic filtering was intuitively expected to excel. In this experiment, we evaluate performance in a wider variety of POMDPs from the POPGym (Morad et al., 2023) benchmark. We select a subset of 12 tasks that probe models for long-term memory, compression, recall, control under noise and reasoning. The aggregated results are shown in Figure 9 and full training curves are also included in Appendix H. Below we discuss the main insights.

---

[4] We found that `Oracle` underperforms in the noiseless `pendulum-swingup`, similarly reported in (Luis et al., 2023), which is why the normalized score in this task is larger than 1.0 in some cases. Moreover, performance does not strictly decrease under higher noise levels, perhaps because noise may actually help avoid early convergence under sparse rewards.

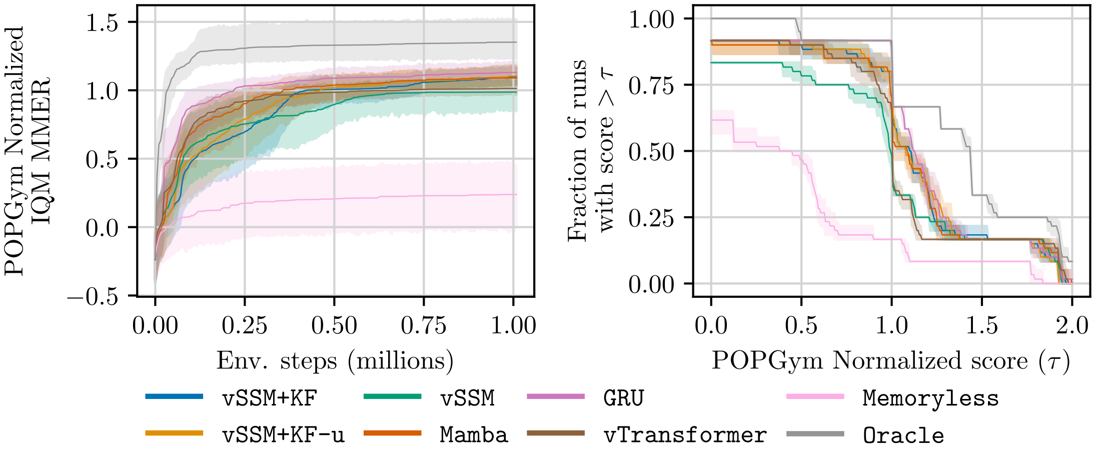

Figure 9: Aggregated performance in POPGym selected environments (12 tasks) with 95% bootstrap confidence intervals over five random seeds. We normalize the maximum-mean episodic return (MMER) by the best reported MMER in (Morad et al., 2023)**(Left)** Normalized IQM MMER **(Right)** Performance profile after 1M environment steps. Higher curves correspond to better performance and a score of 1.0 means equivalent performance as the best baseline (per environment) reported in POPGym.

KF layers can be generally helpful in POMDPs. From the performance profile in Figure 9 we observe a statistically significant gap between `vSSM` and `vSSM+KF`. Interestingly, the largest improvements in sample-efficiency (`RepeatPreviousEasy`) and final performance (`MineSweeperEasy`) correspond to tasks that probe for memory duration and recall, respectively. The parameter count difference between `vSSM` and `vSSM+KF` in these problems is less than 6%, so we believe model capacity is unlikely the reason behind the large performance difference. We hypothesize that, while probabilistic filtering is not required to solve these tasks, the KF layer has extra flexibility via the latent observation and noise signals to accelerate the learning process. We also highlight that `vSSM+KF` and `vSSM+KF-u` show comparable performance in this benchmark, suggesting the input signal to be less critical in general memory tasks.

`vSSM+KF` is less sample-efficient in pure-memory tasks. In particular, we observe that `Mamba`'s input-selectivity is the best-suited mechanism for SSM agents to solve long-term memory problems, matching the performace of `GRU` and `vTransformer`. This is an expected result based on the associative recall performance of Mamba reported in its original paper (Gu and Dao, 2023).

Linear SSMs can have strong performance. Morad et al. (2023) report poor performance when combining PPO with the S4D (Gu et al., 2022b) model. While we do not evaluate the S4D model and use an off-policy algorithm in our RAC architecture, our evaluation shows various linear SSMs have strong performance, often surpassing the best reported scores in Morad et al. (2023). Our observation is consistent with the strong performance of PPO with the S5 model reported by Lu et al. (2023).

5.5 Ablation

We conduct an ablation on `vSSM+KF` where we vary two hyperparameters: the latent state size N and the number of stacked KF layers L[5]. We select four representative tasks from POPGym that test different memory capabilities. The final scores are presented in Figure 10 and the full training curves are included in Appendix J. Performance is most sensitive to these hyperparameters in the `RepeatFirstMedium` task, where the agent must recall information from the first observation over several steps. The general trend is that using more than one layer improves final performance and increases sample-efficiency (see the training curves in Figure 18). Our results are aligned with the good performance of stacked S5 layers reported by Lu et al. (2023), but differ from the observations in (Ni et al., 2023), where both LSTM and transformer models performed best with a single layer in a similar long-term memory task (T-maze passive). From these

---

[5] We use an RMSNorm output block in `vSSM+KF` since it was critical to ensure stable learning when $L > 1$.

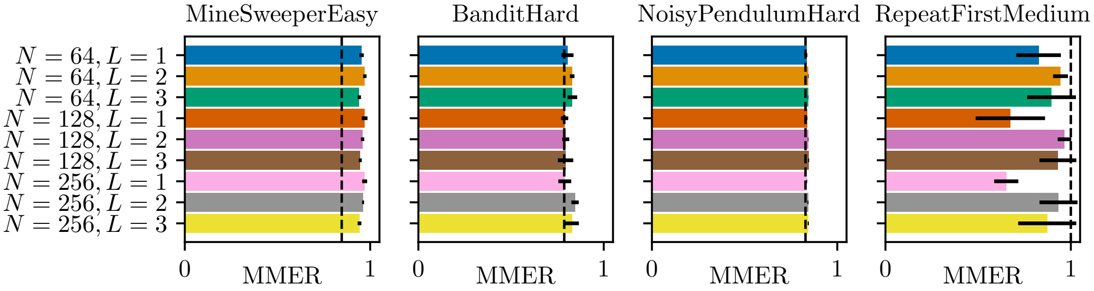

Figure 10: POPGym ablation for `vSSM+KF` over the latent state size N and the number of layers L. We report the mean and standard error over five random seeds of the MMER score after 1M environment steps. The MMER score is shifted from $[-1, 1]$ to $[0, 1]$ for easier visualization. The vertical line represents the best score reported by Morad et al. (2023).

observations, we believe an interesting avenue for future work is to study what mechanisms enable effective stacking and combination of multiple recurrent layers.

6 Conclusion

We investigated the use of Kalman filter (KF) layers as sequence models in a recurrent actor-critic architecture. These layers perform closed-form Gaussian inference in latent space and output a *filtered* state representation for downstream RL components, such as value functions and policies. Thanks to the associative nature of the Kalman filter equations, the KF layers process sequential data efficiently via parallel scans, whose runtime scales logarithmically with the sequence length. To handle trajectories with variable length in off-policy RL, we introduced Masked Associative Operators (MAOs), a general-purpose method that augments any associative operator to recover the correct hidden state when processing padded input data. The KF layers are used as a drop-in replacement for RNNs and SSMs in recurrent architectures, and thus can be trained similarly in an end-to-end, model-free fashion for return maximization.

We evaluated and analysed the strengths and weaknesses of several sequence models in a wide range of POMDPs. KF layers excel in tasks where uncertainty reasoning is key for decision-making, such as the Best Arm Identification task and control under observation noise, significantly improving performance over stateful models like RNNs and deterministic SSMs. In more general tasks, including long-term memory and associative recall, KF layers typically match the performance of transformers and other stateful sequence models, albeit with a lower sample-efficiency.

Limitations and Future Work. We highlight notable limitations of our methodology and suggest avenues for future work. First, we investigated two design decisions in KF layers related to time-varying process noise and posterior covariance as output features. While they resulted in worse performance (see Appendix E), in principle they generalize KF layers and may bring benefits in other tasks or contexts, so we believe it is worth further investigation. Second, we use models with relatively low parameter count ($<$ 1M) which is standard in RL but not on other supervised learning tasks. It may be possible that deeper models with larger parameter counts enable new capabilities, e.g., probabilistic reasoning, without explicit probabilistic filtering mechanisms. Third, `vSSM+KF` uses KF layers as standalone history encoders, but more complex architectures may be needed to stabilize training at larger parameter counts. Typical strategies found in models like `Mamba` include residual connections, layer normalization, convolutions and non-linearities. Fourth, our evaluations were limited to POMDPS with relatively low-dimensional observation and action spaces, where small models have enough capacity for learning. Future work could further evaluate performance in more complex POMDPs and compare with our findings.

References

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep Reinforcement Learning at the Edge of the Statistical Precipice. In *Advances in Neural Information Processing Systems*, volume 34, pages 29304–29320. Curran Associates, Inc., 2021.

Bram Bakker. Reinforcement Learning with Long Short-Term Memory. In *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.

Philipp Becker and Gerhard Neumann. On Uncertainty in Deep State Space Models for Model-Based Reinforcement Learning. *Transactions on Machine Learning Research*, July 2022.

Philipp Becker, Harit Pandya, Gregor Gebhardt, Cheng Zhao, C. James Taylor, and Gerhard Neumann. Recurrent Kalman Networks: Factorized Inference in High-Dimensional Deep Feature Spaces. In *International Conference on Machine Learning*, volume 97, pages 544–552. PMLR, June 2019.

Philipp Becker, Niklas Freymuth, and Gerhard Neumann. KalMamba: Towards Efficient Probabilistic State Space Models for RL under Uncertainty. In *ICML Workshop on Aligning Reinforcement Learning Experimentalists and Theorists (ARLET)*, June 2024.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279, June 2013.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*, volume 29. Springer, 2006.

Noam Brown and Tuomas Sandholm. Superhuman AI for Multiplayer Poker. *Science*, 365(6456):885–890, 2019.

Chang Chen, Yi-Fu Wu, Jaesik Yoon, and Sungjin Ahn. TransDreamer: Reinforcement Learning with Transformer World Models. In *DeepRL Workshop NeurIPS*. arXiv, 2021.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In *8th Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111. Association for Computational Linguistics (ACL), 2014.

Petros Christodoulou. Soft Actor-Critic for Discrete Action Settings. *arXiv:1910.07207*, October 2019.

Meire Fortunato, Melissa Tan, Ryan Faulkner, Steven Hansen, Adrià Puigdomènech Badia, Gavin Buttimore, Charles Deck, Joel Z Leibo, and Charles Blundell. Generalization of Reinforcement Learners with Working and Episodic Memory. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Daniel Y. Fu, Tri Dao, Khaled K. Saab, Armin W. Thomas, Atri Rudra, and Christopher Ré. Hungry Hungry Hippos: Towards Language Modeling with State Space Models. In *International Conference on Learning Representations*, 2023.

Karan Goel, Albert Gu, Chris Donahue, and Christopher Re. It's Raw! Audio Generation with State-Space Models. In *International Conference on Machine Learning*, volume 162, pages 7616–7633. PMLR, July 2022.

Albert Gu and Tri Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. *arXiv:2312.00752*, 2023.

Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. HiPPO: Recurrent Memory with Optimal Polynomial Projections. In *Advances in Neural Information Processing Systems*, volume 33, pages 1474–1487. Curran Associates, Inc., 2020.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently Modeling Long Sequences with Structured State Spaces. In *International Conference on Learning Representations*, 2022a.

Albert Gu, Ankit Gupta, Karan Goel, and Christopher Ré. On the Parameterization and Initialization of Diagonal State Space Models. In *Advances in Neural Information Processing Systems*, volume 35, 2022b.

David Ha and Jürgen Schmidhuber. Recurrent World Models Facilitate Policy Evolution. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Tuomas Haarnoja, Anurag Ajay, Sergey Levine, and Pieter Abbeel. Backprop KF: Learning Discriminative Deterministic State Estimators. In *Advances in Neural Information Processing Systems*, pages 4383–4391, 2016. Curran Associates Inc.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *International Conference on Machine Learning*, volume 80, pages 1861–1870. PMLR, July 2018.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic Algorithms and Applications. *arXiv:1812.05905*, January 2019.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning Latent Dynamics for Planning from Pixels. In *International Conference on Machine Learning*, volume 97, pages 2555–2565. PMLR, June 2019.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to Control: Learning Behaviors by Latent Imagination. In *International Conference on Learning Representations*, 2020.

Matthew Hausknecht and Peter Stone. Deep Recurrent Q-Learning for Partially Observable MDPs. In *AAAI Fall Symposium Series*, 2015.

Nicolas Heess, Jonathan J. Hunt, Timothy P. Lillicrap, and David Silver. Memory-Based Control with Recurrent Neural Networks. In *NIPS Deep Reinforcement Learning Workshop*, December 2015.

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

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, 101(1):99–134, 1998.

R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, March 1960.

Gabriel Kalweit and Joschka Boedecker. Uncertainty-driven Imagination for Continuous Deep Reinforcement Learning. In *Conference on Robot Learning*, pages 195–206. PMLR, October 2017.

Ilya Kostrikov. PyTorch Implementations of Reinforcement Learning Algorithms, 2018. URL `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail`.

Nathan Lambert, Brandon Amos, Omry Yadan, and Roberto Calandra. Objective Mismatch in Model-based Reinforcement Learning. In *Learning for Dynamics and Control*, pages 761–770. PMLR, 2020.

Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A Contextual-Bandit Approach to Personalized News Article Recommendation. In *International Conference on World Wide Web*, pages 661–670, 2010. Association for Computing Machinery.

Long-Ji Lin and Tom M Mitchell. Reinforcement Learning with Hidden States. In *International Conference on Simulation of Adaptive Behavior*, pages 271–280. MIT Press, 1993.

Qinghua Liu, Alan Chung, Csaba Szepesvari, and Chi Jin. When Is Partially Observable Reinforcement Learning Not Scary? In *Conference on Learning Theory*, pages 5175–5220. PMLR, June 2022.

Chris Lu, Yannick Schroecker, Albert Gu, Emilio Parisotto, Jakob Foerster, Satinder Singh, and Feryal Behbahani. Structured State Space Models for In-Context Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 47016–47031. Curran Associates, Inc., 2023.

Carlos E. Luis, Alessandro G. Bottero, Julia Vinogradska, Felix Berkenkamp, and Jan Peters. Value-Distributional Model-Based Reinforcement Learning. *arXiv:2308.06590*, August 2023.

Xuezhe Ma, Chunting Zhou, Xiang Kong, Junxian He, Liangke Gui, Graham Neubig, Jonathan May, and Luke Zettlemoyer. Mega: Moving Average Equipped Gated Attention. In *International Conference on Learning Representations*, 2023.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. In *NIPS Deep Learning Workshop*, December 2013.

Steven Morad, Ryan Kortvelesy, Matteo Bettini, Stephan Liwicki, and Amanda Prorok. POPGym: Benchmarking Partially Observable Reinforcement Learning. In *International Conference on Learning Representations*, 2023.

Kevin Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

Eric Nguyen, Karan Goel, Albert Gu, Gordon Downs, Preey Shah, Tri Dao, Stephen Baccus, and Christopher Ré. S4ND: Modeling Images and Videos as Multidimensional Signals with State Spaces. In *Advances in Neural Information Processing Systems*, volume 35, pages 2846–2861. Curran Associates, Inc., 2022.

Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. Recurrent Model-Free RL Can Be a Strong Baseline for Many POMDPs. In *International Conference on Machine Learning*, pages 16691–16723. PMLR, June 2022.

Tianwei Ni, Michel Ma, Benjamin Eysenbach, and Pierre-Luc Bacon. When Do Transformers Shine in RL? Decoupling Memory from Credit Assignment. In *Advances in Neural Information Processing Systems*, volume 36, pages 50429–50452. Curran Associates, Inc., 2023.

Tianwei Ni, Benjamin Eysenbach, Erfan Seyedsalehi, Michel Ma, Clement Gehring, Aditya Mahajan, and Pierre-Luc Bacon. Bridging State and History Representations: Understanding Self-Predictive RL. In *International Conference on Learning Representations*, 2024.

Junhyuk Oh, Valliappa Chockalingam, Satinder, and Honglak Lee. Control of Memory, Active Perception, and Action in Minecraft. In *International Conference on Machine Learning*, volume 48, pages 2790–2799, June 2016. PMLR.

Christos H. Papadimitriou and John N. Tsitsiklis. The Complexity of Markov Decision Processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

Emilio Parisotto and Russ Salakhutdinov. Efficient Transformers in Reinforcement Learning using Actor-Learner Distillation. In *International Conference on Learning Representations*, October 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Mohammad Reza Samsami, Artem Zholus, Janarthanan Rajendran, and Sarath Chandar. Mastering Memory Tasks with World Models. In *International Conference on Learning Representations*, 2024.

Simo Sarkka and Angel F. Garcia-Fernandez. Temporal Parallelization of Bayesian Smoothers. *IEEE Transactions on Automatic Control*, 66(1):299–306, January 2021.

J. Schmidhuber. Curious Model-Building Control Systems. In *IEEE International Joint Conference on Neural Networks*, pages 1458–1463 vol.2, 1991. IEEE.

Jurgen Schmidhuber. Networks Adjusting Networks. In *Distributed Adaptive Neural Information Processing*, pages 197–208, 1990.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347*, August 2017.

Vaisakh Shaj, Philipp Becker, Dieter Büchler, Harit Pandya, Niels van Duijkeren, C. James Taylor, Marc Hanheide, and Gerhard Neumann. Action-Conditional Recurrent Kalman Networks For Forward and Inverse Dynamics Learning. In *Conference on Robot Learning*, volume 155, pages 765–781. PMLR, November 2021a.

Vaisakh Shaj, Dieter Büchler, Rohit Sonker, Philipp Becker, and Gerhard Neumann. Hidden Parameter Recurrent State Space Models For Changing Dynamics Scenarios. In *International Conference on Learning Representations*, October 2021b.

Vaisakh Shaj, Saleh Gholam Zadeh, Ozan Demir, Luiz Ricardo Douat, and Gerhard Neumann. Multi Time Scale World Models. In *Advances in Neural Information Processing Systems*, November 2023.

Jimmy T. H. Smith, Andrew Warrington, and Scott Linderman. Simplified State Space Layers for Sequence Modeling. In *International Conference on Learning Representations*, 2023.

Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*, volume 7. MIT Press, 2018.

Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and Tasks for Continuous Control. *Software Impacts*, 6:100022, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, and others. Grandmaster Level in StarCraft II using Multi-Agent Reinforcement Learning. *Nature*, 575(7782):350–354, 2019.

Daan Wierstra, Alexander Foerster, Jan Peters, and Jürgen Schmidhuber. Solving Deep Memory POMDPs with Recurrent Policy Gradients. In *Artificial Neural Networks*, pages 697–706, 2007. Springer Berlin Heidelberg.

Biao Zhang and Rico Sennrich. Root Mean Square Layer Normalization. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning. In *International Conference on Robotics and Automation*, 2017.

Luisa Zintgraf, Sebastian Schulze, Cong Lu, Leo Feng, Maximilian Igl, Kyriacos Shiarlis, Yarin Gal, Katja Hofmann, and Shimon Whiteson. VariBAD: Variational Bayes-Adaptive Deep RL via Meta-Learning. *Journal of Machine Learning Research*, 22(289):1–39, 2021.

A Associativity of Masked Associative Operators

Let $\tilde{a}, \tilde{b}, \tilde{c} \in \tilde{\mathcal{E}}$ refer to elements in the space of the MAO $\tilde{\bullet}$, as in Definition 1, with $\tilde{a} = (a, m_a)$, $\tilde{b} = (b, m_b)$, $\tilde{c} = (c, m_c)$. We show that if the sequence $\{m_a, m_b, m_c\}$, is a right-padding mask, that is: $m_a = 1 \implies m_b = m_c = 1$, and $m_b = 1 \implies m_c = 1$, then it holds that $(\tilde{a} \,\tilde{\bullet}\, \tilde{b}) \,\tilde{\bullet}\, \tilde{c} = \tilde{a} \,\tilde{\bullet}\, (\tilde{b} \,\tilde{\bullet}\, \tilde{c})$, i.e., the MAO is associative. Similar to the proof in Lu et al. (2023) we consider all possible values for $\{m_a, m_b, m_c\}$.

Case 1: $m_b = 1$ and $m_c = 1$. The binary masks of b and c are on, so $\tilde{b} \,\tilde{\bullet}\, \tilde{c} = \tilde{b}$, $\tilde{a} \,\tilde{\bullet}\, \tilde{b} = \tilde{a}$ and $\tilde{a} \,\tilde{\bullet}\, \tilde{c} = \tilde{a}$. Then,

$$
\begin{aligned}
(\tilde{a} \,\tilde{\bullet}\, \tilde{b}) \,\tilde{\bullet}\, \tilde{c} &= \tilde{a} && (7)\\
&= \tilde{a} \,\tilde{\bullet}\, (\tilde{b} \,\tilde{\bullet}\, \tilde{c}) && (8)
\end{aligned}
$$

Case 2: $m_b = 0$ and $m_c = 1$. The binary mask of b if off while that of c is on, so $\tilde{b} \,\tilde{\bullet}\, \tilde{c} = \tilde{b}$, then:

$$
\begin{aligned}
(\tilde{a} \,\tilde{\bullet}\, \tilde{b}) \,\tilde{\bullet}\, \tilde{c} &= \tilde{a} \,\tilde{\bullet}\, \tilde{b} && (9)\\
&= \tilde{a} \,\tilde{\bullet}\, (\tilde{b} \,\tilde{\bullet}\, \tilde{c}) && (10)
\end{aligned}
$$

Case 3: $m_b = 0$ and $m_c = 0$. No mask is applied, then the MAO is equivalent to the underlying operator $\bullet$, which is associative by Definition 1.

Note the case $m_b = 1$ and $m_c = 0$ violates associativity, but it is impossible under our initial assumption of a right-padding mask sequence $\{m_a, m_b, m_c\}$.

B Implementation Details

In this section we provide details of various components of the RAC architecture and the specific implementations of history encoders. All methods are implemented in a common codebase written in the Pytorch framework (Paszke et al., 2019).

Embedder. We embed the concatenated observation-action history with a simple linear layer mapping from the combined observation-action dimension to the embedding dimension E.

Soft Actor-Critic. We use a standard SAC implementation with optional automatic entropy tuning (Haarnoja et al., 2019). For discrete action spaces, we use the discrete version of SAC by (Christodoulou, 2019) and one-hot encode the actions.

`vSSM`, `vSSM+KF` & `vSSM+KF-u`. These methods share a similar implementation, with an input linear layer, a linear recurrence and an output linear layer. `vSSM` is equivalent to only using the "Predict" block from the KF layer, while `vSSM+KF-u` removes the input signal $u_{:t}$. For all methods, we discretize the SSM using the zero-order hold method and a learnable scalar step size Δ. In practice we use an auxiliary learnable parameter $\tilde{\Delta}$ and define $\Delta = \texttt{softplus}(\tilde{\Delta})$ to ensure a positive step size. as similarly done in Mamba. We initialize $\tilde{\Delta}$ with a negative value such that after passing through the softplus and after ZOH discretization, the SSM is initialized with eigenvalues close to 1 (i.e., slow decay of state information over time).

`Mamba`. Standard Mamba model from Gu and Dao (2023). We use a reference open-source implementation[6] and modify the parallel scan to use the associated MAO.

`GRU`. Standard implementation included in Pytorch.

`vTransformer`. Default implementation of a causal transformer encoder from Pytorch. We additionally include a sinusoidal positional encoding, as done in prior work using transformers for RL (Ni et al., 2023).

---

[6] `https://github.com/johnma2006/mamba-minimal/tree/03de542a36d873f6e6c4057ad687278cc6ae944d`

C Hyperparameters

Table 1: Hyperparameters used for Section 5. For the `Mamba` parameters, we use the notation from the code by Gu and Dao (2023) and select parameters to match a effective state size $N = 128$. `GRU` and `vTransformer` use default parameters from Pytorch unless noted otherwise.

<table>
<tr><th>Parameter</th><th>BestArm</th><th>DMC</th><th>POPGym</th></tr>
<tr><td colspan="4"><b>Training</b></td></tr>
<tr><td>Buffer size</td><td colspan="3">∞</td></tr>
<tr><td>Adam learning rate</td><td colspan="3">3e-4</td></tr>
<tr><td>Env. steps</td><td>500K</td><td colspan="2">1M</td></tr>
<tr><td>Batch size</td><td>64</td><td colspan="2">32</td></tr>
<tr><td>Update-to-data (UTD) ratio</td><td>0.25</td><td colspan="2">1.0</td></tr>
<tr><td># Eval episodes</td><td>100</td><td colspan="2">16</td></tr>
<tr><td colspan="4"><b>RAC</b></td></tr>
<tr><td>Embedding size (E)</td><td colspan="3">16</td></tr>
<tr><td>Latent size (N)</td><td colspan="3">128</td></tr>
<tr><td>Activations</td><td colspan="3">ReLU</td></tr>
<tr><td>Context length</td><td>256</td><td colspan="2">64</td></tr>
<tr><td>Actor MLP</td><td>[128]</td><td colspan="2">[256, 256]</td></tr>
<tr><td>Critic MLP</td><td>[256]</td><td colspan="2">[256, 256]</td></tr>
<tr><td colspan="4"><b>SAC</b></td></tr>
<tr><td>Discount factor γ</td><td colspan="3">0.99</td></tr>
<tr><td>Entropy temp. α</td><td>0.1</td><td colspan="2">Auto</td></tr>
<tr><td>Target entropy (continuous)</td><td>N/A</td><td colspan="2">-dim($\mathcal{A}$)</td></tr>
<tr><td>Target entropy (discrete)</td><td colspan="2">N/A</td><td>$-0.7\log\big(1/\dim(\mathcal{A})\big)$[7]</td></tr>
<tr><td colspan="4"><b>History Encoders (common)</b></td></tr>
<tr><td>Latent size N</td><td colspan="3">128</td></tr>
<tr><td># layers</td><td colspan="3">1</td></tr>
<tr><td colspan="4"><b><code>vSSM</code>, <code>vSSM+KF</code> & <code>vSSM+KF-u</code></b></td></tr>
<tr><td>$\tilde{\Delta}$ init</td><td colspan="3">-7</td></tr>
<tr><td>$\mathbf{A}$ init</td><td colspan="3">HiPPO (diagonal)</td></tr>
<tr><td>$\mathbf{B}$ init</td><td colspan="3">$\mathbf{I}$</td></tr>
<tr><td>$\mathbf{\Sigma}^{\mathrm{p}}$ init</td><td colspan="3">$\mathbf{I}$</td></tr>
<tr><td>Inital state belief</td><td colspan="3">$\mathcal{N}(\mathbf{0}, \mathbf{I})$</td></tr>
<tr><td>RMSNorm output?</td><td>No</td><td>Yes</td><td>No</td></tr>
<tr><td colspan="4"><b><code>Mamba</code></b></td></tr>
<tr><td>$\mathbf{A}$ init</td><td colspan="3">HiPPO (diagonal)</td></tr>
<tr><td><code>d_model</code> (embedding size)</td><td colspan="3">16</td></tr>
<tr><td><code>d_state</code> (per-channel hidden size)</td><td colspan="3">4</td></tr>
<tr><td>Expand factor E</td><td colspan="3">2</td></tr>
<tr><td>Size of Δ projection</td><td colspan="3">1</td></tr>
<tr><td>1D Conv kernel size</td><td colspan="3">4</td></tr>
<tr><td colspan="4"><b><code>vTransformer</code></b></td></tr>
<tr><td># heads</td><td colspan="3">1</td></tr>
<tr><td>Feedforward size</td><td>128</td><td colspan="2">256</td></tr>
</table>

---

[7] We use a lower value of $-0.35\log\big(1/\dim(\mathcal{A})\big)$ in the `MineSweeper` environment from POPGym, as the default value resulted in divergence during training.

D Best Arm Identification Training Curves

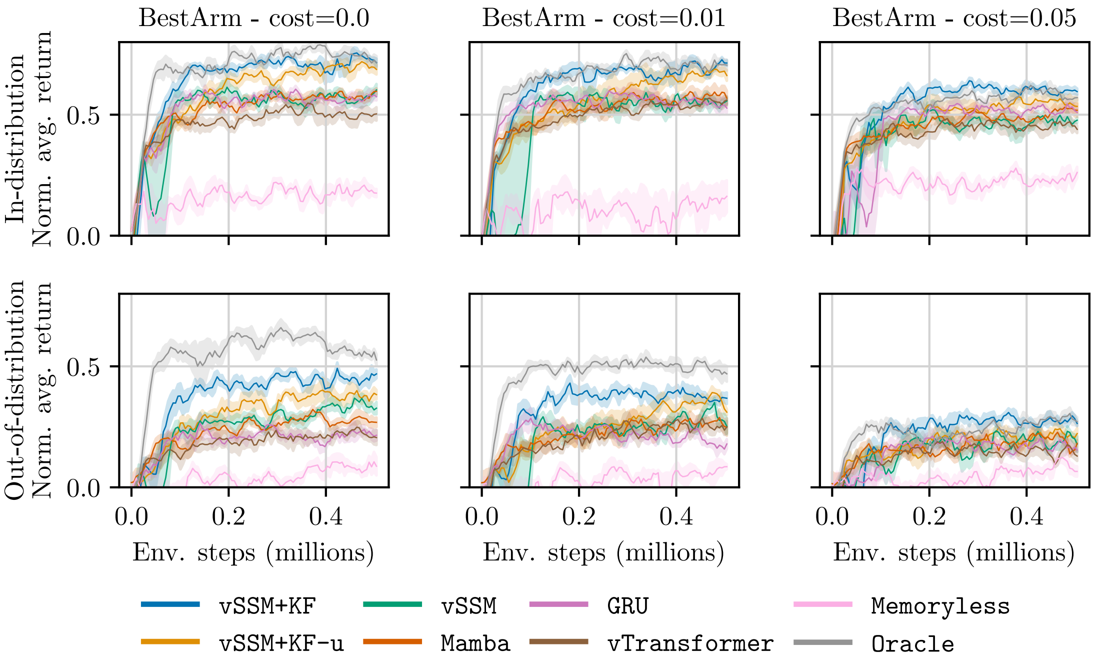

Figure 11: Normalized average return over 100 episodes in and out of distribution, for increasing costs. We report the mean and standard error over 5 random seeds.

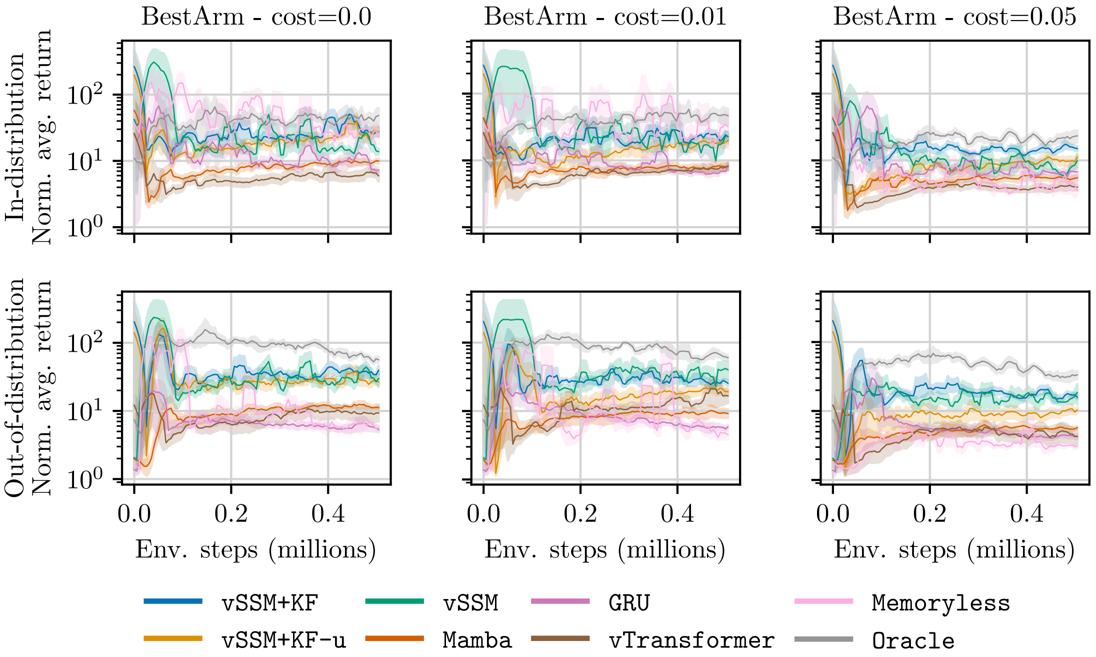

Figure 12: Average (log) episode length over 100 episodes in and out of distribution, for increasing costs. We report the mean and standard error over 5 random seeds.

E KF Layer Design Ablation

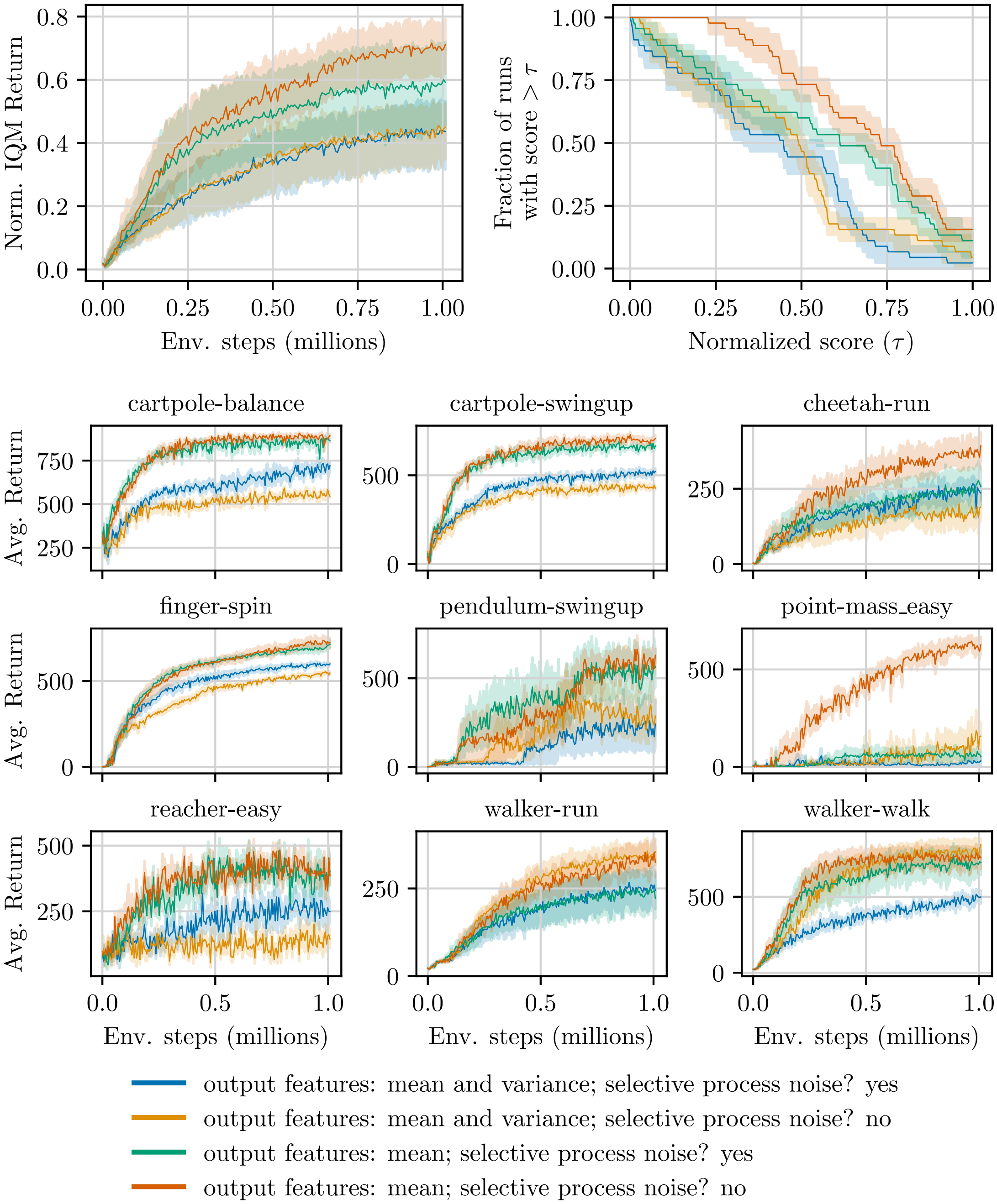

Figure 13: Ablation on design considerations for KF layers. **(Top)** Aggregated performance in noisy DMC benchmark (9 tasks) with 95% bootstrap confidence intervals over five random seeds. **(Top-Left)** Inter-quartile mean returns normalized by the score of `Oracle`. **(Top-Right)** Performance profile after 1M environment steps. **(Bottom)** Training curves. We show mean and standard error over five random seeds. Based on these results, our final design for the KF layer uses only the posterior mean state as the output feature and a time-invariant process noise.

F DMC Training Curves

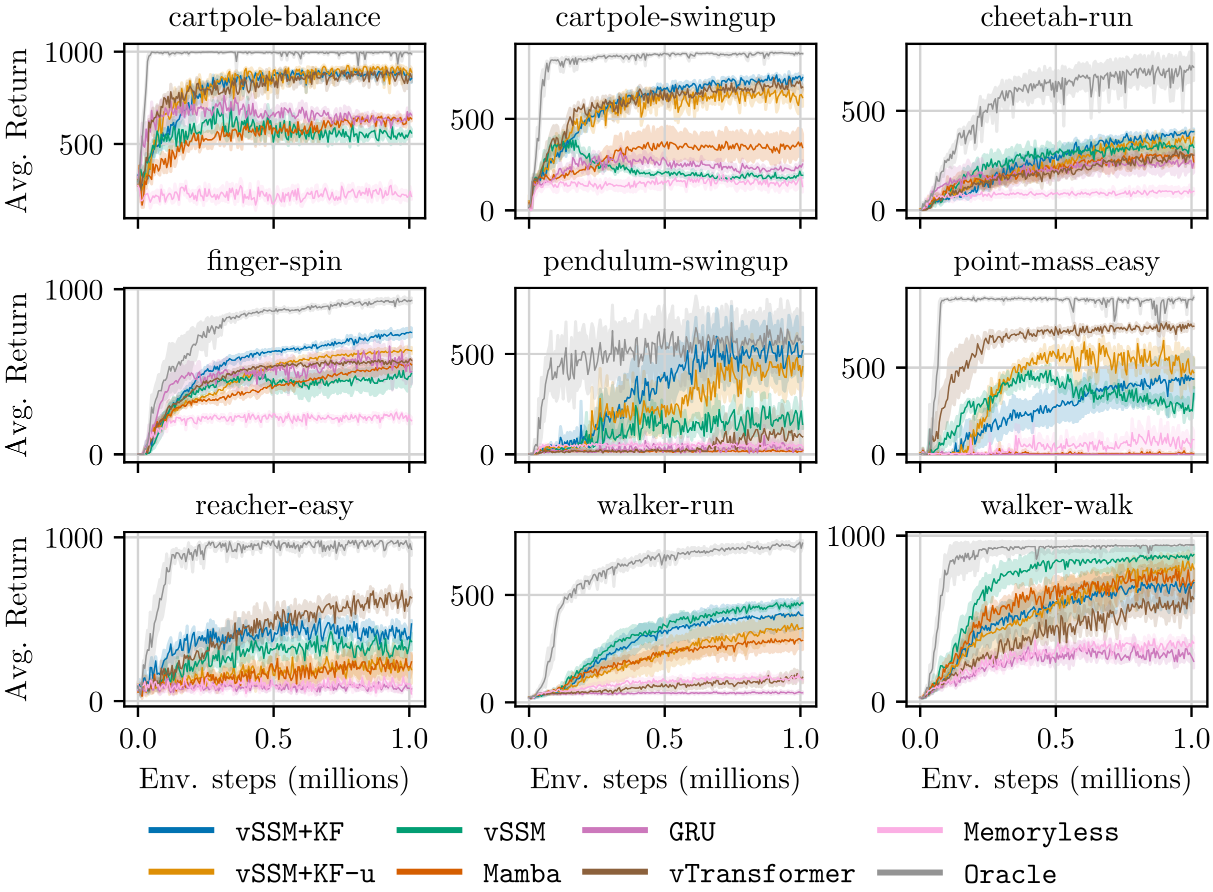

Figure 14: Training curves for the noisy DMC benchmark. We show mean and standard error over five random seeds. For all tasks, we add zero-mean Gaussian noise to the observations with a scale of 0.3, except the `pendulum-swingup` and `point-mass` where the scale is 0.1.

G DMC Noise Ablation

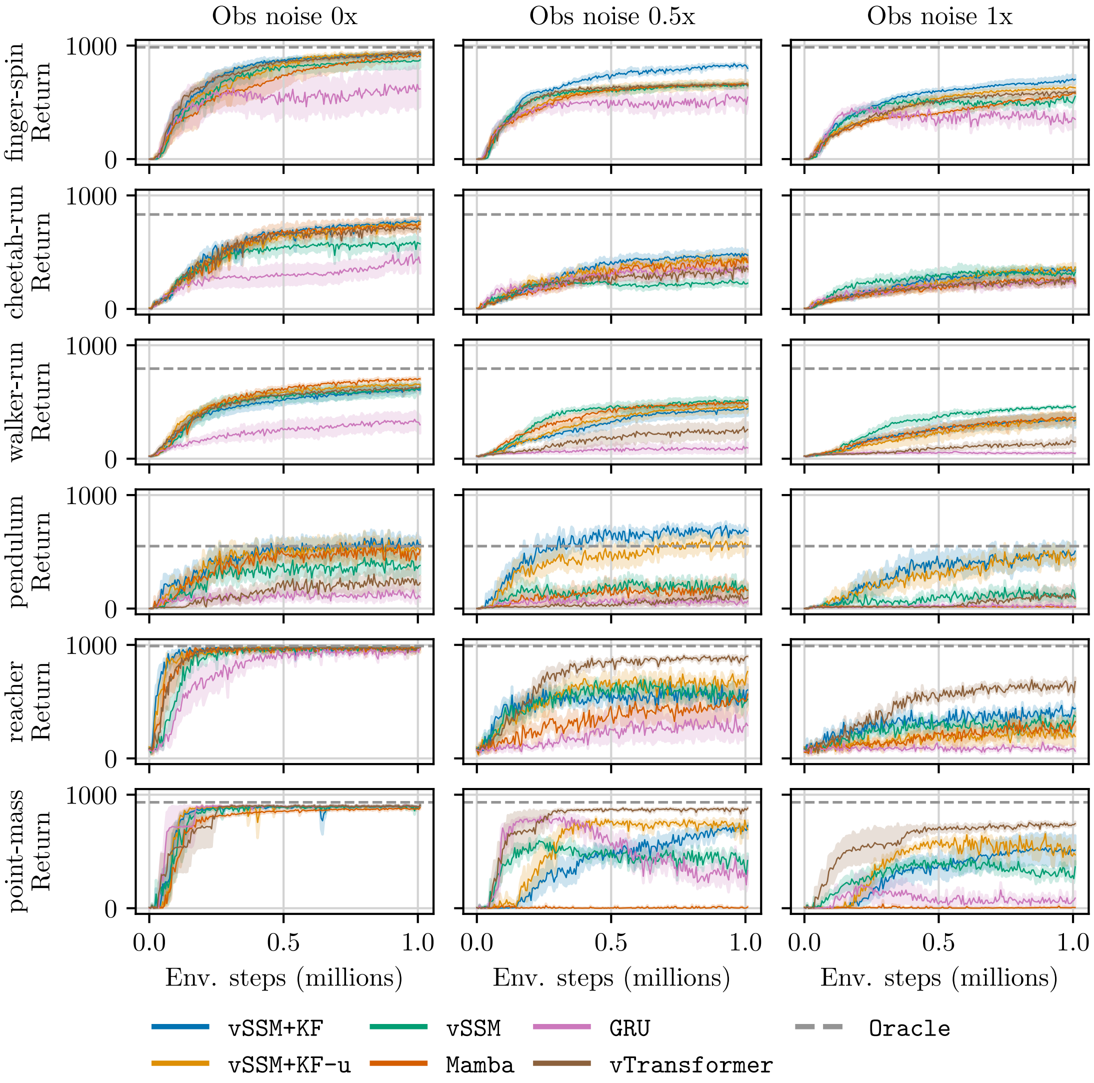

Figure 15: Training curves in six environments from the DMC benchmark with increasing levels of noise. We show mean and standard error over five random seeds (ten for `pendulum`). The base noise scale for all tasks is 0.3, except the `pendulum-swingup` and `point-mass` environments where the scale is 0.1

H POPGym Training Curves

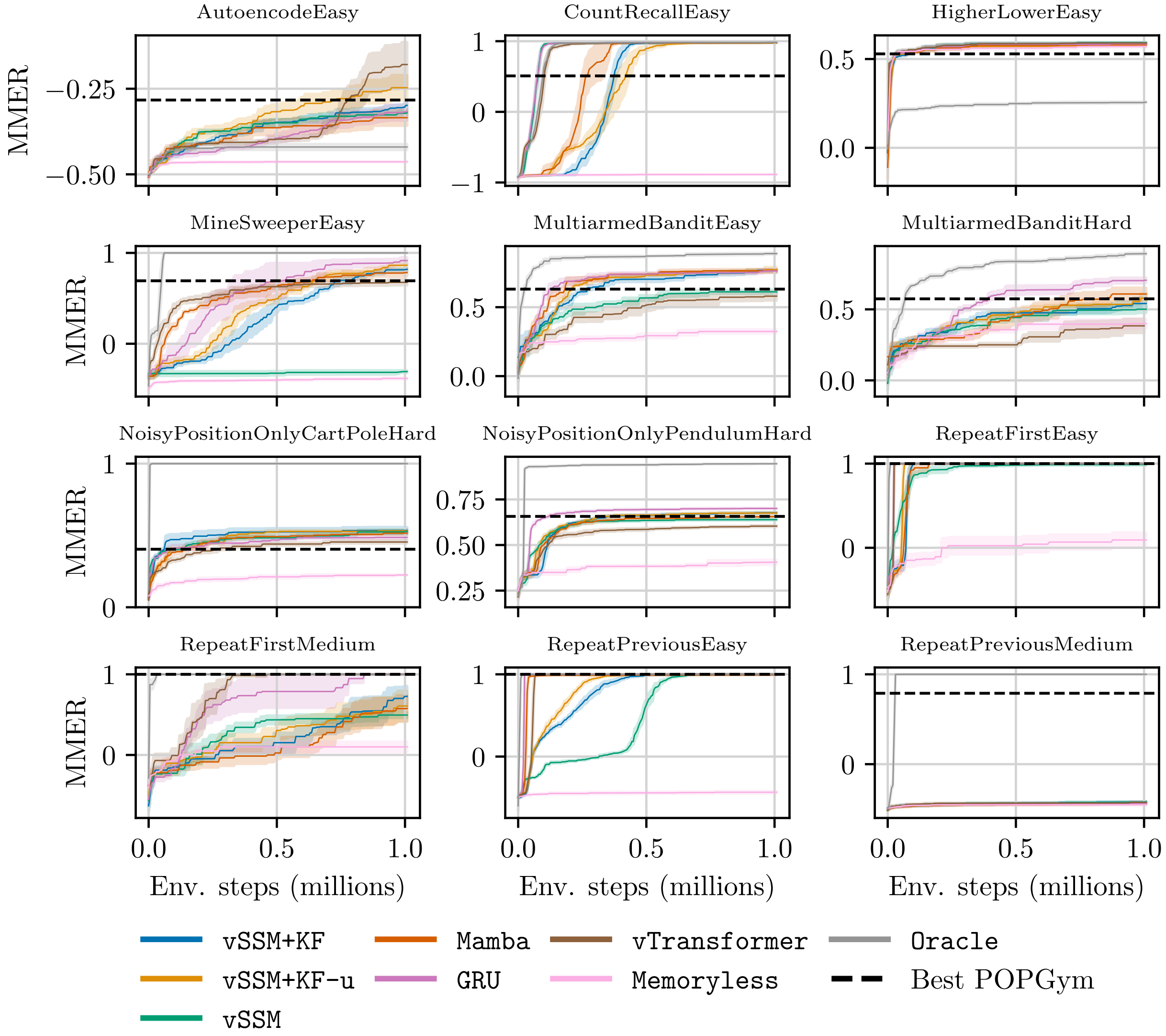

Figure 16: POPGym training curves. We show mean and standard error over five random seeds.

I POPGym Scores

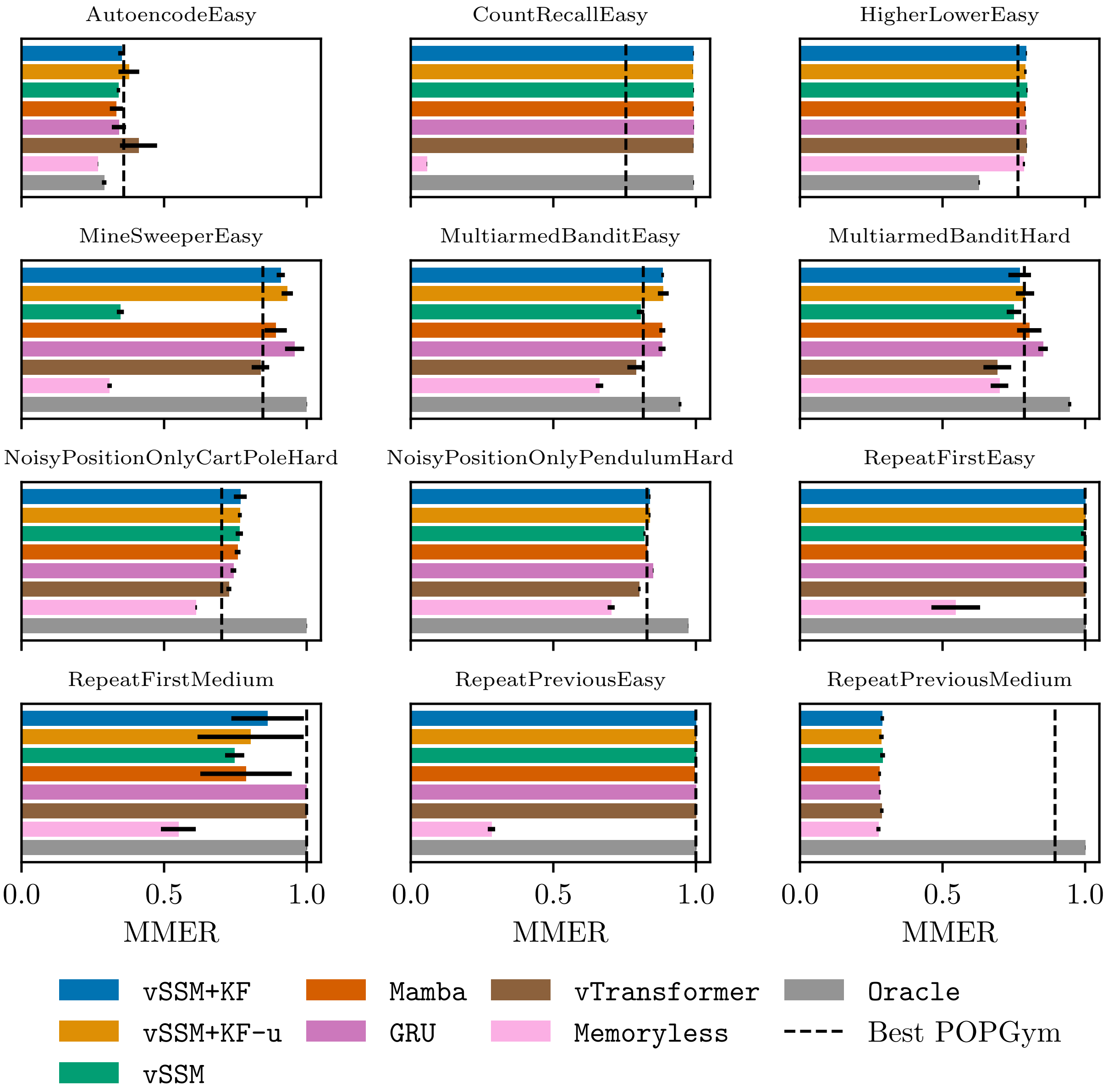

Figure 17: POPGym final MMER after 1M training steps. We show mean and standard error over five random seeds.

Table 2: Scores on POPGym tasks after 1M environment steps. For each environment, we report the MMER mean and standard error over 5 random seeds after 1M steps of training. The MMER is calculated from 16 test episodes. For reference, we include the best MMER score reported by Morad et al. (2023) (mean and standard deviation over three random seeds). We **bold** the highest score(s) per environment obtained by a sequence model.

	AutoencodeEasy	CountRecallEasy	HigherLowerEasy	MineSweeperEasy
`vSSM+KF`	-0.299 ± 0.012	$\mathbf{0.983} \pm 0.002$	0.588 ± 0.002	0.818 ± 0.014
`vSSM+KF-u`	-0.247 ± 0.036	$\mathbf{0.978} \pm 0.001$	0.581 ± 0.004	0.864 ± 0.020
`vSSM`	-0.320 ± 0.006	$\mathbf{0.984} \pm 0.002$	$\mathbf{0.592} \pm 0.003$	-0.307 ± 0.012
`Mamba`	-0.335 ± 0.023	0.982 ± 0.002	0.580 ± 0.003	0.783 ± 0.039
`GRU`	-0.317 ± 0.025	$\mathbf{0.984} \pm 0.001$	0.586 ± 0.002	$\mathbf{0.916} \pm 0.034$
`vTransformer`	$-\mathbf{0.179} \pm 0.065$	0.982 ± 0.001	$\mathbf{0.590} \pm 0.001$	0.676 ± 0.031
`Oracle`	-0.420 ± 0.008	0.984 ± 0.002	0.257 ± 0.003	1.000 ± 0.000
`Memoryless`	-0.463 ± 0.001	-0.887 ± 0.001	0.571 ± 0.004	-0.382 ± 0.008
Best POPGym	-0.283 ± 0.029	0.509 ± 0.062	0.529 ± 0.002	0.693 ± 0.009

	BanditEasy	BanditHard	NoisyCartPoleHard	NoisyPendulumHard
`vSSM+KF`	$\mathbf{0.766} \pm 0.005$	0.541 ± 0.040	$\mathbf{0.535} \pm 0.023$	0.677 ± 0.003
`vSSM+KF-u`	$\mathbf{0.771} \pm 0.019$	0.579 ± 0.032	$\mathbf{0.531} \pm 0.007$	0.675 ± 0.003
`vSSM`	0.612 ± 0.013	0.501 ± 0.025	0.528 ± 0.013	0.639 ± 0.004
`Mamba`	$\mathbf{0.764} \pm 0.011$	0.608 ± 0.043	$\mathbf{0.516} \pm 0.010$	0.658 ± 0.009
`GRU`	$\mathbf{0.763} \pm 0.012$	$\mathbf{0.705} \pm 0.017$	0.486 ± 0.010	$\mathbf{0.701} \pm 0.001$
`vTransformer`	0.580 ± 0.030	0.384 ± 0.049	0.454 ± 0.009	0.604 ± 0.004
`Oracle`	0.889 ± 0.005	0.892 ± 0.006	1.000 ± 0.000	0.946 ± 0.001
`Memoryless`	0.324 ± 0.013	0.399 ± 0.031	0.225 ± 0.003	0.406 ± 0.012
Best POPGym	0.631 ± 0.014	0.574 ± 0.049	0.404 ± 0.005	0.657 ± 0.002

	RepeatFirstEasy	RepeatFirstMedium	RepeatPreviousEasy	RepeatPreviousMedium
`vSSM+KF`	$\mathbf{1.000} \pm 0.000$	0.726 ± 0.127	$\mathbf{1.000} \pm 0.000$	$-\mathbf{0.423} \pm 0.006$
`vSSM+KF-u`	$\mathbf{1.000} \pm 0.000$	0.607 ± 0.186	$\mathbf{1.000} \pm 0.000$	$-\mathbf{0.429} \pm 0.008$
`vSSM`	0.989 ± 0.009	0.495 ± 0.034	$\mathbf{1.000} \pm 0.000$	$-\mathbf{0.420} \pm 0.008$
`Mamba`	$\mathbf{1.000} \pm 0.000$	0.575 ± 0.160	0.993 ± 0.001	-0.441 ± 0.005
`GRU`	$\mathbf{1.000} \pm 0.000$	$\mathbf{1.000} \pm 0.000$	$\mathbf{1.000} \pm 0.000$	-0.440 ± 0.004
`vTransformer`	$\mathbf{1.000} \pm 0.000$	$\mathbf{1.000} \pm 0.000$	$\mathbf{1.000} \pm 0.000$	$-\mathbf{0.426} \pm 0.006$
`Oracle`	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
`Memoryless`	0.093 ± 0.085	0.100 ± 0.061	-0.434 ± 0.013	-0.450 ± 0.007
Best POPGym	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	0.789 ± 0.288

J POPGym Ablation

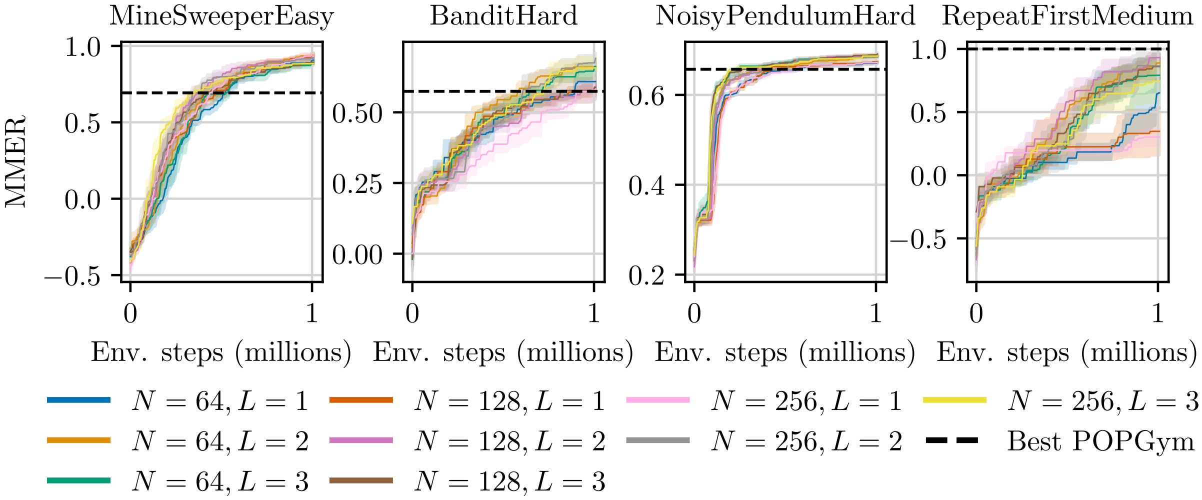

Figure 18: POPGym training curves for `vSSM+KF` ablation over latent state size N and number of KF layers L. We show mean and standard error over five random seeds. For this experiment, `vSSM+KF` uses an RMSNorm output block to ensure stability for $L > 1$.