Pessimistic Iterative Planning for Robust POMDPs

Maris F. L. Galesloot^1 **Marnix Suilen**^{1} **Thiago D. Simão**^{2} **Steven Carr**^{3}
Matthijs T. J. Spaan^4 **Ufuk Topcu**^{3} **Nils Jansen**^{1,5}
^{1}Radboud University Nijmegen ^{2}Eindhoven University of Technology
^{3}University of Texas at Austin ^{4}Delft University of Technology ^{5}Ruhr-University Bochum

Abstract

Robust partially observable Markov decision processes (robust POMDPs) extend classical POMDPs to handle additional uncertainty on the transition and observation probabilities via so-called *uncertainty sets*. Policies for robust POMDPs must not only be memory-based to account for partial observability but also robust against model uncertainty to account for the worst-case instances from the uncertainty sets. We propose the *pessimistic iterative planning* (PIP) framework, which finds such robust memory-based policies for robust POMDPs. PIP alternates between two main steps: (1) selecting an adversarial (non-robust) POMDP via worst-case probability instances from the uncertainty sets; and (2) computing a finite-state controller (FSC) for this adversarial POMDP. We evaluate the performance of this FSC on the original robust POMDP and use this evaluation in step (1) to select the next adversarial POMDP. Within PIP, we propose the RFSCNET algorithm. In each iteration, RFSCNET finds an FSC through a recurrent neural network by using *supervision policies* optimized for the adversarial POMDP. The empirical evaluation in four benchmark environments showcases improved robustness against several baseline methods and competitive performance compared to a state-of-the-art robust POMDP solver.

1 Introduction

Partially observable Markov decision processes (POMDPs; Kaelbling et al., 1998) are the standard model for decision-making under uncertainty. *Policies* select actions based on limited state information towards some objective, *e.g.*, minimizing the expected cost. The standard assumption is that probabilities are precisely known for the transition and observation functions of a POMDP. This assumption is unrealistic, for example, when probabilities are derived from historical data or sensors with limited precision (Thrun et al., 2005), or when the uncertainty is expressed by domain experts.

Robust POMDPs (RPOMDPs; Osogami, 2015) overcome this assumption by introducing *uncertainty sets*, *i.e.*, sets of probabilistic transition and observation functions. *Robust* policies for RPOMDPs account for this uncertainty by optimizing against the *worst-case* instances within the uncertainty sets. Consequently, a robust policy exhibits a lower bound on its actual performance.

Finding optimal policies for (non-robust) POMDPs is an extensively studied problem (Hansen, 1997; Aberdeen and Baxter, 2002; Poupart and Boutilier, 2003; Smith and Simmons, 2004; Kurniawati et al., 2008). The policies inherently depend on the sequences of past actions and observations (the *history*), *i.e.*, they require *memory*. Nowadays, recurrent neural networks (RNNs) are a common formalism to represent such memory-based policies (Ni et al., 2022) thanks to their ability to learn sufficient statistics of the history (Lambrechts et al., 2022). As such, RNNs have successfully been used in POMDPs within reinforcement learning (Bakker, 2001; Hausknecht and Stone, 2015; Heess et al., 2015) and planning settings (Carr et al., 2021).

Alternatively, finite-state controllers (FSCs, Meuleau et al., 1999) offer a more structured way to represent policies and, given the model, allow for a precise and effective numerical evaluation of their performance. Finding FSCs often relies on selecting a predetermined memory size and structure (Junges et al., 2018). An exhaustive search for the optimal size and specific structures may, in general, be computationally intractable. Therefore, one may decide to cluster any history-based policy into a finite-memory policy, for instance, by expanding all possible histories and minimizing the resulting so-called *policy tree* (Grzes et al., 2015). However, such an approach suffers from the typical exponential blow-up in the size of the history that is inherent to POMDPs. Instead, Carr et al. (2021) learn the memory structures for FSCs by training an RNN on data collected from the POMDP.

Challenges in RPOMDPs. Finding robust policies for RPOMDPs has its own set of challenges. As robust policies must be optimized against the worst-case instance of the uncertainty sets, determining this worst-case is paramount. Furthermore, a policy's performance in an RPOMDP cannot be evaluated through simulations, as it is unclear which instance from the uncertainty sets to sample. This last challenge is detrimental to the use of RNNs in planning settings where guarantees on a policy's worst-case performance are required. FSCs, on the other hand, can be evaluated exactly in RPOMDPs through *robust dynamic programming* (Iyengar, 2005), which finds and uses the exact worst-case instance to evaluate the FSC's performance. Finding robust policies represented as FSCs, however, suffers from the same problems as in POMDPs, *e.g.*, selecting a predetermined memory size and structure, as done in *sequential convex programming* (SCP, Cubuktepe et al., 2021). As our empirical evaluation shows, wrongly or over-specifying the required memory may lead to worse results for SCP in RPOMDPs.

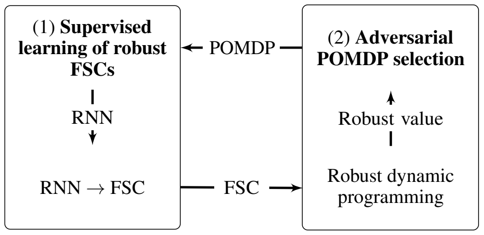
(1) Supervised learning of robust FSCs
RNN
RNN → FSC
POMDP
FSC
(2) Adversarial POMDP selection
Robust value
Robust dynamic programming

Figure 1: Overview of the PIP framework. The steps on the left-hand side are specific to rFSCNet.

Contributions. Observing how RNNs and FSCs complement each other for RPOMDPs, we combine the representational power of RNNs and the exact robust evaluation of FSCs, allowing for the best of both worlds by leveraging prior work to derive an FSC representation of an RNN policy (Koul et al., 2019). Figure 1 outlines our iterative pessimistic planning framework as the main contribution.

Pessimistic iterative planning for RPOMDPs (Section 4)**.** We propose the *pessimistic iterative planning* (PIP) framework to find a robust policy by iterating over and (approximately) solving POMDPs within the uncertainty set of the RPOMDP that are *adversarial* instances to the current policy, *i.e.*, a worst-case instance given the current policy. We implement PIP in our RNN-based algorithm, which we name rFSCNet, consisting of the following two main parts:

(1) **Supervised learning of robust FSCs** (Section 5)**.** We train an RNN based on data collected from *supervision policies* that we compute on the adversarial POMDPs. We explicitly construct the RNN to enable the extraction of FSCs. At each iteration, the FSC serves in the robust evaluation and adversarial POMDP selection. The adversarial instances are employed to further train the RNN by refining the collected histories and associated supervision policy, guiding the RNN towards a better-performing, robust policy.

(2) **Adversarial POMDP selection** (Section 6)**.** First, we compute the worst-case performance of the FSC on the RPOMDP via robust dynamic programming, thereby producing an exact guarantee. Then, we introduce a linear program that efficiently finds a POMDP instance within the uncertainty sets that is adversarial, *i.e.*, a worst-case instance, to the current FSC.

In our experimental evaluation on four benchmarks, we (1) showcase that the FSCs found by our method are competitive with and outperform the state-of-the-art FSC-based solver for RPOMDPs by Cubuktepe et al. (2021), and, (2) conduct an ablation study to show the impact of our contributions on robustness, namely iteratively finding and re-training on adversarial POMDPs, as opposed to baselines that train on either fixed or random POMDPs within the uncertainty set of the RPOMDP.

2 Preliminaries

The set of all distributions over X is denoted by $\Delta(X)$. A probability distribution $\mu \in \Delta(X)$ is called *Dirac* if $\mu(x) = 1$ for precisely one $x \in X$ and zero otherwise. The number of elements in a set X is denoted by $|X|$. *Iverson brackets* return $[P] = 1$ if P is true and 0 otherwise. Finally, the set of *probability intervals* with lower bounds strictly greater than zero is $\mathbb{I} = \{[i, j] \mid 0 < i \leq j \leq 1\}$.

Definition 1 (POMDP). *A* partially observable Markov decision process *is a tuple $M = \langle S, A, T, Z, O, C\rangle$, where S, A, Z are finite sets of states, actions, and observations, $T\colon S \times A \to \Delta(S)$ is the probabilistic transition function, $O\colon S \to Z$ is the deterministic state-based observation function, and $C\colon S \times A \to \mathbb{R}_{\geq 0}$ is the cost (or reward) function.*

For simplicity and without loss of generality, we consider POMDPs with *deterministic* observations, where the observation function maps only to Dirac distributions. This assumption is non-restrictive as every POMDP can be transformed into one with deterministic observations (Chatterjee et al., 2015, Remark 1). A *trajectory* in a POMDP is a finite sequence of states and actions: $\omega_t = s_1 a_1 s_2 \dots s_t \in (S \times A)^{t-1} \times S$, such that $T(s_{i+1} \mid s_i, a_i) > 0$ for $1 \leq i < t$. Due to partial observability, states are not observable to the agent. A *history* is the observable part of a trajectory: $h_t = O(s_1) a_1 O(s_2) \dots O(s_t) = z_1 a_1 z_2 \dots z_t \in (Z \times A)^{t-1} \times Z$. The set of all (finite) trajectories and associated histories are Ω and $\mathcal{H}$, respectively. Histories can be compressed into sufficient statistics known as *beliefs*, that is, probability distributions over states (Kaelbling et al., 1998). The set of all belief states in a POMDP is $\mathcal{B} \subseteq \Delta(S)$. The initial state distribution (belief) is $b_0 \in \mathcal{B}$. A belief $b \in \mathcal{B}$ can be computed from a history $h \in \mathcal{H}$ using Bayes' rule (Spaan, 2012). A *Markov decision process* (MDP) is a POMDP where each state is fully observable, and a *Markov chain* (MC) is an MDP with a single action (which can be omitted).

2.1 Policies and Expected Costs

A *policy* resolves the action choices in a POMDP and is a function $\pi\colon \mathcal{H} \to \Delta(A)$ mapping histories to distributions over actions. Correspondingly, policies may also be belief-based $\pi\colon \mathcal{B} \to \Delta(A)$. The set of all history-based policies is Π. We seek to find a policy $\pi \in \Pi$ that minimizes the expected cost of reaching goal states $G \subseteq S$, also known as the stochastic shortest path (SSP) problem (Bertsekas, 2005). For any trajectory ω, the cumulative cost $\rho_{\Diamond G}\colon \Omega \to \mathbb{R}_{\geq 0} \cup \{+\infty\}$ is (Forejt et al., 2011):

$$\rho_{\Diamond G}(\omega) = \begin{cases} \infty & \forall t \in \mathbb{N}, s_t \notin G, \\ \sum_{t=0}^{\min\{t \mid s_t \in G\}-1} C(s_t, a_t) & \text{otherwise.} \end{cases} \tag{1}$$

The SSP objective is to find an optimal policy $\pi \in \Pi$ that minimizes the expected cumulative cost J_T^π of the trajectories generated by playing π under the transition function T:

$$\pi^* \in \operatorname*{arginf}_{\pi \in \Pi} J_T^\pi, \quad J_T^\pi = \mathbb{E}_{\pi, T}\Big[\rho_{\Diamond G} \mid s_0 \sim b_0\Big]. \tag{2}$$

In the fully observable setting of MDPs, dynamic programming can compute an optimal policy and the associated cost for the SSP problem. For POMDPs, finding optimal policies for the SSP problem is undecidable (Madani et al., 2003). Therefore, we approximate optimal policies with finite memory.

Finite-state controllers. A policy has *finite-memory* if it can be represented by a finite-state controller.

Definition 2 (FSC). *An FSC is a tuple $\pi_f = \langle N, n_0, \delta, \eta\rangle$ where N is a finite set of memory nodes and $n_0 \in N$ the initial node, $\delta\colon N \times Z \to \Delta(A)$ is the stochastic* action function*, and $\eta\colon N \times Z \to N$ is the deterministic* memory update. *When $|N| = k$, the FSC is also called a k-FSC.*

The *performance* of an FSC π_f on a POMDP M can be evaluated by computing the state-values $V^{\pi_f}\colon S \times N \to \mathbb{R}$ on the product Markov chain (Meuleau et al., 1999) via dynamic programming.

Recurrent neural networks. RNNs are (infinite) state machines parameterized by differential functions, which we use to learn information about the history in $\hat{\mathcal{H}} \subseteq \mathbb{R}^d$. The *hidden size d* defines the length of the latent history vector. Analogously to an FSC, the RNN consists of a parameterized internal memory update $\hat{\eta}_\phi\colon \hat{\mathcal{H}} \times Z \to \hat{\mathcal{H}}$ that recurrently computes the new latent memory vector $\hat{h} \in \hat{\mathcal{H}}$ from the observations in a history $h \in \mathcal{H}$. Thus, the RNN represents a function $\text{RNN}\colon \mathcal{H} \to \hat{\mathcal{H}}$.

When we append a fully connected layer with a $\mathrm{softmax}$ activation function $\sigma^{\phi}\colon \hat{\mathcal{H}} \to \Delta(A)$ to the RNN, it yields an RNN policy network $\pi^{\phi}\colon \mathcal{H} \to \Delta(A)$ from histories to distributions over actions.

3 Robust Markov Models

A *robust* MDP (RMDP) is an MDP where the transition probabilities are not precisely given but belong to a predefined set of distributions known as the uncertainty set (Nilim and Ghaoui, 2005). We fix a specific type of RMDP where the transition function maps to a probability interval or zero.

Definition 3 (RMDP)**.** *An RMDP with interval uncertainty is a tuple $\mathcal{U} = \langle S, A, \mathcal{T}, C\rangle$, where $\mathcal{T}\colon S \times A \to (S \to \mathbb{I} \cup \{0\})$ is the uncertain transition function that maps each transition to either a probability interval in $\mathbb{I}$, or probability 0 whenever the transition does not exist.*

The uncertainty is resolved in a game-like manner where the agent (choosing the actions) and nature (choosing a probability distribution from the uncertainty set) alternate turns (Iyengar, 2005). We assume independence between all state-action pairs, an assumption known as (s, a)*-rectangularity*, which means that nature's choices are not restricted by any previous choice (Iyengar, 2005; Wiesemann et al., 2013). Thus, the uncertainty set $\mathcal{T}$ is convex and factorizes over state-action pairs:

$$\mathcal{T} = \bigotimes_{(s,a)\in S\times A} \mathcal{T}(s,a), \quad \mathcal{T}(s,a) = \left\{ T(s,a) \in \Delta(S) \mid \forall_{s'\in S}\colon T(s' \mid s,a) \in \mathbb{I} \cup \{0\} \right\}. \tag{3}$$

Furthermore, we assume that nature's resolution of uncertainty is *static*. This means that, if the model contains a cyclic structure, nature's choice does not change when state-action pairs are revisited.

RMDPs have two optimal value functions and associated optimal policies: one where the agent and nature play *adversarially*, and one where they play cooperatively. The former is known as the robust setting, and the latter as optimistic. For ease of presentation, we focus on the robust setting for the remainder of the paper. Optimal robust $\underline{\mathcal{Q}}$ and $\underline{\mathcal{V}}$ values can be computed by extending dynamic programming to *robust dynamic programming* (Iyengar, 2005; Nilim and Ghaoui, 2005). The optimal robust $\underline{\mathcal{V}}$ and $\underline{\mathcal{Q}}$ values are the least fixed points $\underline{\mathcal{V}}^{*}$ and $\underline{\mathcal{Q}}^{*}$ of the following recursive equations[*]:

$$\underline{\mathcal{V}}_n(s) = \min_{a\in A} \underline{\mathcal{Q}}_n(s,a), \quad \underline{\mathcal{Q}}_{n+1}(s,a) = C(s,a) + \sup_{T(s,a)\in\mathcal{T}(s,a)} \left\{ \sum_{s'\in S} T(s' \mid s,a)\underline{\mathcal{V}}_n(s') \right\}. \tag{4}$$

Under (s, a)-rectangularity and the interval uncertainty model, the inner optimization problem can be efficiently solved via a *bisection algorithm* (Nilim and Ghaoui, 2005, Section 7.2). The robust state-values $\underline{\mathcal{V}}$ and optimal policy, which is memoryless deterministic under (s, a)-rectangularity, can be derived from the robust action values $\underline{\mathcal{Q}}^{*}$ in the same way as for standard MDPs (Iyengar, 2005)

Robust POMDPs (RPOMDPs; Osogami, 2015) extend POMDPs in a similar way that RMDPs extend MDPs by accounting for uncertainty in the transition and observation functions. Without loss of generality, we consider only uncertainty in the transition function, as any RPOMDP with an uncertain observation function can be transformed into an equivalent RPOMDP with deterministic observations (Bovy et al., 2024, Appendix B). Again, we focus on uncertainty given by intervals.

Definition 4 (RPOMDP)**.** *An RPOMDP with interval uncertainty is a tuple $\mathcal{M} = \langle S, A, \mathcal{T}, C, Z, O\rangle$, where $S, A, Z, O,$ and C are the sets of states, actions, and observations, observation function and the cost function from standard POMDPs, respectively. The uncertain transition function $\mathcal{T}\colon S \times A \to (S \to \mathbb{I} \cup \{0\})$ is the same as in RMDPs.*

Again, we assume (s, a)-rectangularity and static uncertainty resolution. An *instance* of an RPOMDP $\mathcal{M}$ is a POMDP M such that every transition probability of T lies within its respective interval in $\mathcal{T}$. We also write $T \in \mathcal{T}$ and $M \in \mathcal{M}$ in that case. In RPOMDPs, the trajectories generated by the policy $\pi \in \Pi$ depend on the transition function $T \in \mathcal{T}$. Hence, the robust SSP objective becomes:

$$\pi^{*} \in \operatorname*{arginf}_{\pi\in\Pi} \mathcal{J}^{\pi}_{\mathcal{T}}, \quad \mathcal{J}^{\pi}_{\mathcal{T}} = \sup_{T\in\mathcal{T}} J^{\pi}_{T}. \tag{5}$$

Goal. Given an RPOMDP $\mathcal{M} = \langle S, A, \mathcal{T}, C, Z, O\rangle$ with a set of goal states $G \subseteq S$, compute a policy $\pi \in \Pi$ in the form of a finite-state controller π_f to minimize the worst-case expected cost $\mathcal{J}^{\pi}_{\mathcal{T}}$.

[*]To compute optimistic policies and values $\overline{\mathcal{V}}$ and $\overline{\mathcal{Q}}$, we only need to replace the inner supremum with an infimum. All our following results automatically extend to the optimistic case by substituting the $\sup$ for an $\inf$.

4 Pessimistic Iterative Planning

We now present our main contributions. We outline the two main parts of the *pessimistic iterative planning* (PIP) framework and, subsequently, give an overview of our implementation of PIP, named RFSCNET, to compute robust policies for RPOMDPs.

The framework. Analogously to a two-player game, PIP consists of iteratively executing two parts:

(1) Compute an FSC policy π_f for a given POMDP $M \in \mathcal{M}$.
(2) Select an *adversarial* POMDP $M' \in \mathcal{M}$ with respect to this FSC policy π_f.

The adversarial POMDP M' then is used as the next input POMDP in part (1). These steps are repeated until we reach a termination criterion.

The implementation. Next, we detail the steps of our algorithm RFSCNET, which implements the two parts of PIP. Note that, unlike our implementation of the first part, the implementation of part two applies to any implementation of PIP and is not specific to RFSCNET.

Part one corresponds with Section 5 and the left-hand side in Figure 1. In this part, we compute FSCs for adversarial POMDPs $M \in \mathcal{M}$ through the use of an RNN, which is specific to RFSCNET:

i. Compute a *supervision policy* π_M for the POMDP M (Section 5.1) and simulate π_M on M to collect the histories and the action distributions of π_M into a data set $\mathcal{D}$ (Section 5.2).
ii. Train the RNN policy π^ϕ on the data $\mathcal{D}$ (Section 5.3) and extract an FSC π_f (Section 5.4).

Part two corresponds to Section 6 and the right-hand side of Figure 1. Here, we implement the robust evaluation of a computed FSC and, subsequently, the selection of the adversarial POMDP:

iii. Evaluate the FSC π_f exactly through robust dynamic programming (Section 6.1).
iv. Compute the adversarial POMDP $M' \in \mathcal{M}$ based on the FSC π_f (Section 6.2).

After step (iv.), if we have not hit the termination criteria, we start a new iteration at step (i.). We determine to stop the algorithm at step (iii.) based on the FSC's evaluation when the worst-case expected cost of the FSC satisfies a target threshold, or we may set a maximum number of iterations.

5 Supervised Learning of Robust FSCs

We present our methods of collecting a data set to train the RNN, the policies used to gather this data, and how to extract a finite-memory policy from an RNN. We initialize with an arbitrary POMDP instance $M \in \mathcal{M}$ as input, which adapts at each iteration to be adversarial towards the current policy.

5.1 Supervision Policies

Recall from Section 2 that computing an optimal policy in POMDPs for our objective is undecidable. Furthermore, our iterative procedure relies on computing several policies for different POMDPs. Thus, computational efficiency is a concern, and fast approximate methods are preferred. Therefore, we compute a *supervision policy* $\pi_M : \mathcal{B} \to \Delta(A)$ that approximates the optimal policy π^* for the POMDP M. Thus, we compute an approximation $\hat{Q}_M$ of the belief-based values $Q_M : \mathcal{B} \times A \to \mathbb{R}$ for $M \in \mathcal{M}$. The supervision policy is then derived by taking the minimizing action, *i.e.*, $\pi_M(b) = \operatorname{argmin}_{a \in A} \hat{Q}_M(b, a)$. For training, we interpret these deterministic policies as stochastic by taking Dirac distributions. We consider the algorithms Q_{MDP} (Littman et al., 1995) and the fast-informed bound (FIB; Hauskrecht, 2000) to compute the approximations Q_{MDP} and Q_{FIB} respectively.

Q_{MDP} may compute sub-optimal policies as it assumes full state observability after a single step, neglecting information-gathering (Littman et al., 1995). The values Q_{FIB} are tighter than the one given by Q_{MDP} since it factors in the observation of the next state. Although Q_{FIB} is computationally more expensive than Q_{MDP}, Q_{FIB} updates are still of polynomial complexity, allowing us to compute it for each $M \in \mathcal{M}$ efficiently, see Appendix B for more details.

Alternatives. We opt for Q_{MDP} and Q_{FIB} because of their computational efficiency. Nonetheless, other POMDP solution methods may also be used, such as *Monte Carlo planning* (Silver and Veness, 2010) or variants of *heuristic-search value iteration* (Smith and Simmons, 2004; Horák et al., 2018).

5.2 Data Generation

To train the RNN, we generate a data set $\mathcal{D}$ by simulating the supervision policy π_M on the current (adversarial) POMDP $M \in \mathcal{M}$. We aggregate a batch of $I \in \mathbb{N}$ finite-horizon histories and associated action distributions of the supervision policy π_M up to length $H \in \mathbb{N}$.

During simulation for $i = 1, \dots, I$, we compute the beliefs $b_t^{(i)}$ from history $h_t^{(i)}$ derived from the probabilities of M at each time step $1 \leq t < H$, with b_0 from $\mathcal{M}$. Then, $\mu_t^{(i)}{=}\pi_M(b_t^{(i)})$ represents the action distributions of the supervision policy during the simulation, and $a_t^{(i)} \sim \mu_t^{(i)}$ is the action played in simulation i at time t. Then, $\mathcal{D} = \{\{h_t^{(i)}, \mu_t^{(i)}\}_{t=1}^{H}\}_{i=1}^{I}$ consists of I simulations of H histories and their associated action distributions. In the next step, we employ $\mathcal{D}$ to train the RNN.

5.3 RNN Policy Training

We train the RNN policy π^ϕ on the data set $\mathcal{D}$ collected from $M \in \mathcal{M}$. The training objective Φ of the RNN is to minimize the distance between the distributions of the RNN policy π^ϕ and the distributions μ of the supervision policy π_M: $\Phi = \min_\phi \frac{1}{|\mathcal{D}|H} \sum_{i=0}^{|\mathcal{D}|} \sum_{t=0}^{H} D_{\text{KL}}\big(\pi^\phi(h_t^{(i)}) \parallel \mu_t^{(i)}\big)$, where D_{KL} is the Kullback-Leibler divergence, the histories h are inputs to the RNN, and the action distributions μ are labels. We optimize the parameters ϕ for this objective by calculating the gradient via backpropagation through time.

We opt for a model-based approach, using approximate solvers to compute the supervision policy π_M for $M \in \mathcal{M}$ and train the RNN to imitate π_M in a supervised manner. Thus, we know we can stop training once the loss for Φ is sufficiently low. Alternatively, one could employ a model-free objective such as *recurrent policy gradients* (Wierstra et al., 2007). However, the latter neglects the available information from the model and, thus, requires a large number of samples to converge (Peters and Schaal, 2006; Moerland et al., 2020). Furthermore, it is unclear when training should be stopped.

Recall that in our approach, we change the POMDP at each iteration to be adversarial against the current policy. To find the new adversarial POMDP $M' \in \mathcal{M}$, we need to find a finite-memory representation of the policy, which we outline in the next subsection.

5.4 Extracting an FSC from an RNN

We cluster the hidden memory states of the RNN (Zeng et al., 1993; Omlin and Giles, 1996) to find a finite-memory policy in the form of an FSC. There are multiple ways to achieve such a clustering. Prior work (Carr et al., 2021) uses a quantized bottleneck network (QBN; Koul et al., 2019) to reduce the possible memory states to a finite set. They train the QBN *post hoc* by minimizing the mean-squared error on the hidden states generated by the RNN. Alternatively, it can be trained *end-to-end* by updating the parameters with the gradient calculated with Φ, which we name QRNN for quantized RNN. Moreover, similar to *post hoc* training of the QBN, we can run a clustering algorithm such as `k-means++` (Arthur and Vassilvitskii, 2007) to minimize the in-cluster variance of the hidden states. For *post hoc* training, we employ the histories in $\mathcal{D}$ to generate the RNN hidden states. We consider all three methods in this paper. For more details on the QBN, we refer to Appendix D.1.

Instead of through simulations, as done in Koul et al. (2019), we utilize the model to construct the FSC. The clustering determines a finite set N from the RNN's hidden states in $\mathcal{H}$. We find the FSC's memory update η by executing a forward pass of the RNN's memory update $\hat{\eta}_\phi$ for each reconstruction of $n \in N$, which produces the next memory nodes $n' \in N$ and RNN hidden state $\hat{h} \in \mathcal{H}$ for each $z \in Z$ by exploiting the batch dimension. Then, the action mapping δ for n and z is given by the distribution of the RNN policy network $\sigma^\phi(\hat{h})$ for the next memory state. The initial node n_0 is determined by the RNN's initial state and we prune any unreachable nodes from the FSC.

6 Robust Policy Evaluation and Adversarial Selection of POMDPs

For the FSCs found by the approach explained in the previous section, we present our methods for robust policy evaluation of the FSC and, subsequently, for selecting the worst-case adversarial POMDP $M' \in \mathcal{M}$ from the uncertainty set.

6.1 Robust Policy Evaluation

The performance of a finite-memory policy represented by $\pi_f = \langle N, n_0, \eta, \delta \rangle$ in an (s,a)-rectangular RPOMDP, such as we have, is given by the following robust Bellman equation, for all $\langle s, n \rangle \in S \times N$:

$$\underline{\mathcal{V}}_\star^{\pi_f}(\langle s, n \rangle) = \sum_{a \in A} \Big(\delta(a \mid n, O(s)) C(s,a) \\ + \sup_{T(s,a) \in \mathcal{T}(s,a)} \Big\{ \sum_{s' \in S} \sum_{n' \in N} T(s' \mid s, a) \delta(a \mid n, O(s)) [n' = \eta(n, O(s))] \underline{\mathcal{V}}_\star^{\pi_f}(\langle s', n' \rangle) \Big\} \Big). \quad (6)$$

The inner optimization problem is convex under (s,a)-rectangular and convex uncertainty sets, but solving it at each dynamic programming iteration requires $|S||A|$ linear programs. For computational tractability, we instead opt to evaluate the FSC on the RPOMDP via a product construction of a robust Markov chain, similar to the one used for evaluating FSCs in POMDPs (Meuleau et al., 1999).

Definition 5 (Robust policy evaluation)**.** *Given an RPOMDP $\mathcal{M} = \langle S, A, \mathcal{T}, C, Z, O \rangle$ with initial belief b_0 and an FSC $\pi_f = \langle N, n_0, \eta, \delta \rangle$, the robust state-values of $\mathcal{M}$ under F are given by the state-values in the robust Markov chain $\langle S \times N, b_0^{\pi_f}, \mathcal{T}^{\pi_f}, C^{\pi_f} \rangle$ where the state-space is the product of RPOMDP states S and FSC memory nodes N, the initial state distribution is $b_0^{\pi_f}(\langle s, n \rangle) = b_0(s)[n = n_0]$, and the robust transition and cost functions are:*

$$\mathcal{T}^{\pi_f}(\langle s', n' \rangle \mid \langle s, n \rangle) = \sum_{a \in A} \mathcal{T}(s,a)(s') \delta(a \mid n, O(s)) [n' = \eta(n, O(s))],$$
$$C^{\pi_f}(\langle s, n \rangle) = \sum_{a \in A} \delta(a \mid n, O(s)) C(s,a),$$

where addition and multiplication over intervals follow the standard rules for interval arithmetic (Hickey et al., 2001). The robust value function $\underline{\mathcal{V}}^{\pi_f} \colon S \times N \to \mathbb{R}_{\geq 0} \cup \{+\infty\}$, follows directly from robust dynamic programming on this robust Markov chain with the bisection algorithm, see Equation (4)*. Then, the* robust value *in the RPOMDP is given by $\sum_{\langle s,n \rangle \in S \times N} b_0^{\pi_f}(\langle s, n \rangle) \underline{\mathcal{V}}^{\pi_f}(\langle s, n \rangle)$.*

We again assume full (s,a)-rectangularity on this product state-space, effectively leading to $(\langle s, n \rangle, a)$-rectangularity. That is, nature's choices are independent of the agent's current memory, yielding a value $\underline{\mathcal{V}}^{\pi_f}$ that is a conservative bound compared to the value $\underline{\mathcal{V}}_\star^{\pi_f}$ where nature's choices must be consistent with the agent's memory. Appendix A contains the formal statement and proof. Using Definition 5, we compute the performance of any FSC extracted from the RNN and store the best FSC over the iterations based on its performance. Furthermore, the robust state-values $\underline{\mathcal{V}}^{\pi_f}$ enable us to find the new associated adversarial POMDP $M' \in \mathcal{M}$ for the next iteration of PIP.

6.2 Finding Adversarial POMDP Instances

We now construct a heuristic to find a new POMDP instance $M' \in \mathcal{M}$ that constitutes the local worst-case instance for the current policy under (s,a)-rectangularity of the RPOMDP. Let $\pi_f = \langle N, n_0, \delta, \eta \rangle$ be the current FSC. Given the robust value function $\underline{\mathcal{V}}^{\pi_f}$ computed at the previous step, we aim to find a POMDP $M' \in \mathcal{M}$ that is adversarial to the FSC π_f and, therefore, induces its worst-case value. We compute the following transition probability under the additional constraint that probabilities are dependent on the memory nodes $n \in N$, *i.e.*, under (s,a)-rectangularity:

$$\forall (s,a) \in S \times A \colon \quad \underline{T}(s,a) = \operatorname*{argsup}_{T(s,a) \in \mathcal{T}(s,a)} \sum_{n \in N} \sum_{s' \in S} \sum_{n' \in N} \mathcal{T}^{\pi_f}(\langle s', n' \rangle \mid \langle s, n \rangle) \underline{\mathcal{V}}^{\pi_f}(\langle s', n' \rangle). \quad (7)$$

We construct a linear program (LP) that precisely encodes our requirements. Let $\hat{T}_{s,a,s'}$ be the optimization variables representing the transition function probabilities $\Pr(\cdot \mid s, a)$ for each state-action pair $(s,a) \in S \times A$. The LP is then given by:

$$\forall (s,a) \in S \times A \colon \quad \max_{\hat{T}_{s,a,s'}} \quad \sum_{n \in N} \sum_{s' \in S} \sum_{n' \in N} [n' = \eta(n, O(s))] \, \delta(a \mid n, O(s)) \hat{T}_{s,a,s'} \, \underline{\mathcal{V}}^{\pi_f}(\langle s', n' \rangle)$$
$$\text{s.t.} \quad \sum_{s' \in S} \hat{T}_{s,a,s'} = 1, \text{ and, } \forall s' \in S \colon \hat{T}_{s,a,s'} \in \mathcal{T}(s,a)(s'). \quad (8)$$

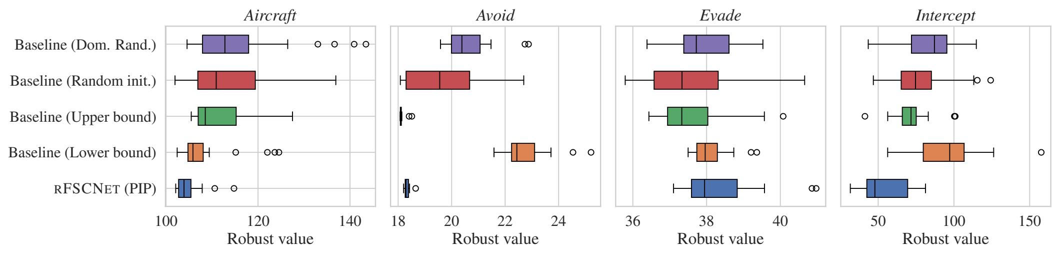

Figure 2: Boxplots depicting the minimum (*i.e.*, best) robust value of the extracted FSC policies for both rFSCNet and the baselines across 20 seeds, all configured with Q_{MDP} and `k-means++`.

Solving this LP yields assignments for the variables $\hat{T}_{s,a,s'}$ that determine the *adversarial* transition function $\hat{T}\colon S \times A \to \Delta(S)$ that satisfies $\hat{T}(s' \mid s,a) \in \mathcal{T}(s,a)(s')$ for all $(s,a) \in S \times A$. By construction, the assignments are valid probability distributions inside each respective interval and yield a heuristic for the worst possible value for the given FSC under (s,a)-rectangularity. Thus, the variables $\hat{T}_{s,a,s'}$ constitute the probabilities of the transition function $\hat{T}$ for the *adversarial* POMDP $M' = \langle S, A, \hat{T}, C, Z, O\rangle$. In this step, by computing M', we have closed the loop. With this adversarial POMDP instance $M' \in \mathcal{M}$, we resume execution from the first step in Section 5 of our framework until the FSC satisfies a target value or we reach the maximum number of iterations.

7 Experimental Evaluation

We empirically assess different aspects of rFSCNet to address the following questions:

- **(Q1) Robustness and baseline comparison.** Does rFSCNet provide robustness against model uncertainty? How does it compare to various baselines not utilizing the PIP framework?
- **(Q2) Comparison with the state-of-the-art.** How does rFSCNet's performance compare to SCP (Cubuktepe et al., 2021), a state-of-the-art method to compute policies for RPOMDPs?
- **(Q3) Memory size sensitivity.** How does the FSC size affect rFSCNet and SCP's performance?
- **(Q4) Configuration sensitivity.** How do different configurations of supervision policies and clustering methods affect the performance of rFSCNet?

Environments. We extend four POMDP environments to RPOMDPs: an *Aircraft* collision avoidance problem (Kochenderfer et al., 2015; Cubuktepe et al., 2021), and three grid-worlds with adversaries (Junges et al., 2021) named *Avoid*, *Evade*, and *Intercept*. On *Aircraft*, the agent is tasked to avoid a collision with an intruder aircraft but has to account for uncertainty in the probabilities of the pilot's responsiveness and of the intruder changing direction, both mapping to a $[0.6, 0.8]$ interval. The grid-world environments model the probability of taking multiple steps instead of a single one for each possible moving action, given by the interval $[0.1, 0.4]$. We provide environment descriptions and dimensions, run times, and tools and hyperparameters in Appendices C, E and G, respectively.

Baselines. We consider baselines to evaluate the impact of the adversarial selection of POMDPs of the PIP framework on robust performance. The baselines train either on a fixed POMDP throughout the iteration, initialized randomly or by the lower/upper bound of the uncertainty set, or on random POMDPs at each iteration, resembling the practice of *domain randomization* (Tobin et al., 2017).

Metrics. For performance, we compare the *robust values* of the computed FSCs, as in Definition 5. For rFSCNet and the baselines, we consider the best robust value found across the iterations. As these methods exhibit randomness due to sampling and initialization, we report statistics of the robust value across 20 seeds. SCP is not dependent on randomness, and we only report a single value.

7.1 Analysis

Figure 2 compares the performance of rFSCNet to the aforementioned baselines. Figure 3 shows rFSCNet's median and minimum performance when configured with a maximal memory size of $k = 9$, compared to the SCP method with two different sizes $k \in \{3, 9\}$, see Definition 2.

Robust values		*Aircraft*	*Avoid*	*Evade*	*Intercept*
RMDP (LB)	$\underline{Q}^*$	94.24	18.05	31.19	16.99
SCP	$k = 3$	116.03	20.07	37.97	**31.57**
	$k = 9$	116.58	29.51	39.78	101.12
rFSCNet	med.	**103.30**	**18.51**	**37.61**	56.20
	min.	102.03	18.16	36.65	34.95

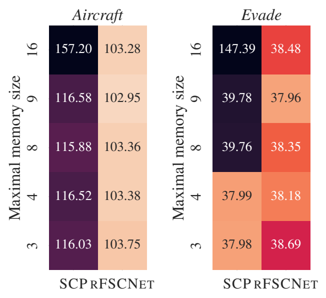

Figure 3: The robust values (lower is better) as computed by rFSCNet when compared to SCP. *Right:* heatmaps comparing the median value of rFSCNet to SCP under various memory sizes. *Left:* median (med.) and minimum (min.) of the best values found across 20 seeds compared to SCP. The lower bound (LB) is the optimal robust value $\underline{Q}^*$ of the underlying RMDP, ignoring partial observability. We highlight the best value between SCP's value and the median value of rFSCNet.

Additionally, the heatmaps showcase the effect of various memory sizes k on the performance of both rFSCNet and SCP in *Aircraft* and *Evade*. Appendix F provides the complete set of results. In particular, we include confidence estimates in the memory comparison in Appendix F.2, Figure 4 in Appendix F.1 provides results under various additional configurations, and Figure 6 in Appendix F.3 compares the training performance of rFSCNet to a baseline trained on a fixed POMDP.

(Q1) Comparison to baselines. As seen in Figure 2, on *Aircraft* and *Intercept*, rFSCNet outperforms all baselines, reaching a median value across the seeds lower than the first quartile of all baselines. Therefore, on average, rFSCNet incurs lower expected costs than the baselines. On *Evade*, the results are more ambiguous, and the baselines can perform better, demonstrating that in this environment, it can suffice to ignore the model uncertainty. However, we still observe that rFSCNet achieves at least the same robust performance as the baselines.

On *Avoid*, rFSCNet performs slightly worse than the baseline that is trained on the upper bound of the uncertainty set, while the remaining baselines clearly perform very poorly. By coincidence, the upper bounds appear to result in a good approximation of the worst-case probabilities. rFSCNet, without this lucky initialization, still achieves comparable performance to this baseline. Good performance of the baselines is not guaranteed, and the baselines may find much worse policies, as evidenced by the results obtained when trained on the lower bound or through domain randomization. Therefore, we conclude that the baselines are unreliable as they are sensitive to the POMDPs used throughout training. In contrast, rFSCNet performs reliably across environments.

(Q2) Comparison with the state-of-the-art. As seen in Figure 3, in comparison to rFSCNet, SCP performs comparably on *Evade* and best on *Intercept* when $k = 3$. rFSCNet outperforms SCP on *Aircraft*, *Avoid*, and *Evade* by both the median and minimum performance across the 20 seeds. When the memory size in SCP is set to $k = 9$, which is also the memory size we set for rFSCNet in these results, rFSCNet outperforms SCP across all benchmarks, both in median and minimum performance. Therefore, we can conclude that rFSCNet improves over the state-of-the-art.

(Q3) Memory size sensitivity. Figure 3 with SCP's results across $k \in \{3, 9\}$ indicates that SCP performs worse with more memory, especially on *Avoid* and *Intercept*. The heatmaps of Figure 3 show that SCP also performs much worse on *Aircraft* and *Evade* when more memory is specified, as indicated by the high values. In contrast, rFSCNet is not sensitive to memory over-specification, exhibiting consistent performance across the memory sizes. These results demonstrate the benefit of learning the memory structure instead of specifying it beforehand, as done in SCP.

(Q4) Configuration sensitivity. In Appendix F.1, Figure 4 depicts the performance of rFSCNet across various configurations. From our results, we did not observe a major difference between using Q_{MDP} and Q_{FIB} as supervision policies. Training the QBN *end-to-end* proves less stable than *post hoc*, and `k-means++` produces the best results. The results demonstrate that, while the configuration does impact performance, rFSCNet performs consistently across configurations.

7.2 Discussion

The results show that rFSCNet finds better robust policies than both SCP and the baseline methods, positively answering research questions (1) and (2). To rFSCNet's merit, our results also show that a learning-based approach through the RNN has the advantage of flexibility in determining the memory structure and sizes of the FSCs, positively answering research question (3). Lastly, while rFSCNet allows for various configurations, it performs consistently, positively answering question (4).

The performance of rFSCNet is dependent on a few factors. In Appendix F.3, Figure 6 shows evidence that training on adversarial POMDPs, instead of a single fixed POMDP, is a more difficult learning target. Even though this makes the training task more challenging, rFSCNet still performs well. Secondly, the performance of the FSCs is impacted by the quality of discretization of the hidden states of the RNN, and faulty extraction of the FSC from the RNN leads to finding a worst-case that is not informative. How to optimally extract finite-state representations of RNNs is still an open problem. In this paper, we tested multiple options based on clustering. Ultimately, the PIP framework is modular and allows for other methods that compute FSCs for POMDPs to be used instead.

Lastly, we note that the performance of rFSCNet is limited by the quality of the supervision policies we compute during training. For instance, this limitation could explain why rFSCNet performs worst on *Intercept*, as this benchmark relies on information gathering, an aspect on which Q_{MDP} is known to perform poorly. Nonetheless, rFSCNet's modularity allows for any POMDP policy to be applied as a supervision policy in the RNN's training procedure, allowing for trade-offs between quality and computational efficiency, depending on the task at hand.

8 Related Work

Computing optimal robust policies for RMDPs has been studied extensively (Nilim and Ghaoui, 2005; Iyengar, 2005; Wiesemann et al., 2013). RMDPs are also used in model-based (robust) reinforcement learning (RL) to explicitly account for model uncertainty and enable efficient exploration (Jaksch et al., 2010; Petrik and Subramanian, 2014; Suilen et al., 2022; Moos et al., 2022).

For RPOMDPs, early works extend value iteration and point-based methods to account for the additional uncertainty (Itoh and Nakamura, 2007; Ni and Liu, 2013; Osogami, 2015), or use sampling over the uncertainty sets (Burns and Brock, 2007). Ni and Liu (2008) introduce a policy iteration algorithm for optimistic policies. Chamie and Mostafa (2018) consider robustifying a given POMDP policy to combat sensor imprecision. Nakao et al. (2021) extend value iteration for *distributionally robust* POMDPs, where the agent receives side information after a decision period, which is a less conservative setting. Extensions to value iteration for RPOMDPs typically do not scale well to the large state spaces (up to 13000+) we consider in this paper. More recent methods compute FSCs for RPOMDPs via quadratic (Suilen et al., 2020) or sequential (Cubuktepe et al., 2021) convex programming. In contrast to our work, the convex optimization methods compute FSCs of a predefined size and structure. Additionally, they cannot compute FSCs in the optimistic case.

In RL, policy gradient algorithms incorporate memory using RNNs (Wierstra et al., 2007). Thereafter, RNNs serve as the baseline neural architecture for reinforcement learning in POMDPs (Ni et al., 2022). Recently, Wang et al. (2023) learned a stochastic recurrent memory representation for (non-robust) POMDPs using information from the model during training. RNNs have also successfully been used in a planning setting to compute FSCs for POMDPs (Carr et al., 2019, 2020, 2021).

9 Conclusion

We presented PIP, a novel planning framework for robust POMDPs. Our framework implementation, rFSCNet, utilizes RNNs to compute FSCs for the adversarial POMDPs selected by PIP, allowing for the memory structure to be learned from data. Our experiments show that our approach yields more robust policies than the baseline approaches. Additionally, rFSCNet is less sensitive to over-parameterization of memory size than SCP, and outperforms the state-of-the-art SCP solver in three of the four benchmarks considered in this paper. Future work may investigate alleviating the limitation of the supervision policies by optimizing the RNN with a more complicated training objective or by considering more advanced supervision policies.

Impact Statement

This paper presents work whose goal is to advance the field of machine learning and, more specifically, robust sequential decision-making under partial observability. There are many potential societal consequences of our work due to the generality of RPOMDPs.

Acknowledgments and Disclosure of Funding

We would like to thank the anonymous reviewers for their valuable feedback, Eline M. Bovy and Merlijn Krale for their helpful discussions on this work, and Niels Neerhoff for allowing us to use the code of his master's thesis. This research has been partially funded by the NWO grant NWA.1160.18.238 (PrimaVera) and the ERC Starting Grant 101077178 (DEUCE).

References

Abadi, M., Barham, P., Chen, J., Chen, Z., et al. (2016). Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283.

Aberdeen, D. and Baxter, J. (2002). Scalable internal-state policy-gradient methods for pomdps. In *ICML*, pages 3–10. Morgan Kaufmann.

Arthur, D. and Vassilvitskii, S. (2007). k-means++: the advantages of careful seeding. In *SODA*, pages 1027–1035. SIAM.

Bakker, B. (2001). Reinforcement learning with long short-term memory. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1475–1482. MIT Press.

Bertsekas, D. P. (2005). *Dynamic programming and optimal control, 3rd Edition*. Athena Scientific.

Bovy, E. M., Suilen, M., Junges, S., and Jansen, N. (2024). Imprecise probabilities meet partial observability: Game semantics for robust pomdps. In *IJCAI*, pages 6697–6706. ijcai.org.

Burns, B. and Brock, O. (2007). Sampling-based motion planning with sensing uncertainty. In *ICRA*, pages 3313–3318. IEEE.

Carr, S., Jansen, N., Junges, S., and Topcu, U. (2023). Safe reinforcement learning via shielding under partial observability. In *AAAI*, pages 14748–14756. AAAI Press.

Carr, S., Jansen, N., and Topcu, U. (2020). Verifiable rnn-based policies for pomdps under temporal logic constraints. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4121–4127. IJCAI.org.

Carr, S., Jansen, N., and Topcu, U. (2021). Task-aware verifiable RNN-based policies for partially observable Markov decision processes. *J. Artif. Intell. Res.*, 72:819–847.

Carr, S., Jansen, N., Wimmer, R., Serban, A. C., Becker, B., and Topcu, U. (2019). Counterexample-guided strategy improvement for pomdps using recurrent neural networks. In *IJCAI*, pages 5532–5539. ijcai.org.

Chamie, M. E. and Mostafa, H. (2018). Robust action selection in partially observable Markov decision processes with model uncertainty. In *CDC*, pages 5586–5591. IEEE.

Chatterjee, K., Chmelik, M., Gupta, R., and Kanodia, A. (2015). Qualitative analysis of pomdps with temporal logic specifications for robotics applications. In *ICRA*, pages 325–330. IEEE.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, pages 1724–1734. ACL.

Cubuktepe, M., Jansen, N., Junges, S., Marandi, A., Suilen, M., and Topcu, U. (2021). Robust finite-state controllers for uncertain pomdps. In *AAAI*, pages 11792–11800. AAAI Press.

Forejt, V., Kwiatkowska, M. Z., Norman, G., and Parker, D. (2011). Automated verification techniques for probabilistic systems. In *SFM*, volume 6659 of *Lecture Notes in Computer Science*, pages 53–113. Springer.

Geiger, L. and Team, P. (2020). Larq: An open-source library for training binarized neural networks. *Journal of Open Source Software*, 5(45):1746.

Grzes, M., Poupart, P., Yang, X., and Hoey, J. (2015). Energy efficient execution of POMDP policies. *IEEE Trans. Cybern.*, 45(11):2484–2497.

Gurobi Optimization, LLC (2023). Gurobi Optimizer Reference Manual.

Hansen, E. A. (1997). An improved policy iteration algorithm for partially observable mdps. In *NIPS*, pages 1015–1021. The MIT Press.

Hausknecht, M. J. and Stone, P. (2015). Deep recurrent Q-learning for partially observable MDPs. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 29–37. AAAI Press.

Hauskrecht, M. (2000). Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research (JAIR)*, 13:33–94.

Heess, N., Wayne, G., Silver, D., Lillicrap, T. P., Erez, T., and Tassa, Y. (2015). Learning continuous control policies by stochastic value gradients. In *NIPS*, pages 2944–2952.

Hensel, C., Junges, S., Katoen, J., Quatmann, T., and Volk, M. (2022). The probabilistic model checker storm. *Int. J. Softw. Tools Technol. Transf.*, 24(4):589–610.

Hickey, T. J., Ju, Q., and van Emden, M. H. (2001). Interval arithmetic: From principles to implementation. *J. ACM*, 48(5):1038–1068.

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

Horák, K., Bosanský, B., and Chatterjee, K. (2018). Goal-hsvi: Heuristic search value iteration for goal pomdps. In *IJCAI*, pages 4764–4770. ijcai.org.

Itoh, H. and Nakamura, K. (2007). Partially observable Markov decision processes with imprecise parameters. *Artif. Intell.*, 171(8-9):453–490.

Iyengar, G. N. (2005). Robust dynamic programming. *Math. Oper. Res.*, 30(2):257–280.

Jaksch, T., Ortner, R., and Auer, P. (2010). Near-optimal regret bounds for reinforcement learning. *J. Mach. Learn. Res.*, 11:1563–1600.

Junges, S., Jansen, N., and Seshia, S. A. (2021). Enforcing almost-sure reachability in pomdps. In *Computer Aided Verification (CAV)*, volume 12760 of *Lecture Notes in Computer Science*, pages 602–625. Springer.

Junges, S., Jansen, N., Wimmer, R., Quatmann, T., Winterer, L., Katoen, J., and Becker, B. (2018). Finite-state controllers of POMDPs using parameter synthesis. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 519–529. AUAI Press.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

Kochenderfer, M. J., Amato, C., Chowdhary, G., How, J. P., Reynolds, H. J. D., Thornton, J. R., Torres-Carrasquillo, P. A., Ure, N. K., and Vian, J. (2015). *Optimized Airborne Collision Avoidance*, pages 249–276.

Koul, A., Fern, A., and Greydanus, S. (2019). Learning finite state representations of recurrent policy networks. In *ICLR (Poster)*. OpenReview.net.

Kurniawati, H., Hsu, D., and Lee, W. S. (2008). SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems (RSS)*. MIT Press.

Kwiatkowska, M., Norman, G., and Parker, D. (2011). PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification (CAV)*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer.

Lambrechts, G., Bolland, A., and Ernst, D. (2022). Recurrent networks, hidden states and beliefs in partially observable environments. *Trans. Mach. Learn. Res.*, 2022.

Li, F. and Liu, B. (2016). Ternary weight networks. *CoRR*, abs/1605.04711.

Littman, M. L., Cassandra, A. R., and Kaelbling, L. P. (1995). Learning Policies for Partially Observable Environments: Scaling Up. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 362–370. Morgan Kaufmann.

Madani, O., Hanks, S., and Condon, A. (2003). On the undecidability of probabilistic planning and related stochastic optimization problems. *Artif. Intell.*, 147(1-2):5–34.

Meuleau, N., Kim, K., Kaelbling, L. P., and Cassandra, A. R. (1999). Solving pomdps by searching the space of finite policies. In *UAI*, pages 417–426. Morgan Kaufmann.

Moerland, T. M., Broekens, J., and Jonker, C. M. (2020). Model-based reinforcement learning: A survey. *CoRR*, abs/2006.16712.

Moos, J., Hansel, K., Abdulsamad, H., Stark, S., Clever, D., and Peters, J. (2022). Robust reinforcement learning: A review of foundations and recent advances. *Mach. Learn. Knowl. Extr.*, 4(1):276–315.

Nakao, H., Jiang, R., and Shen, S. (2021). Distributionally robust partially observable markov decision process with moment-based ambiguity. *SIAM J. Optim.*, 31(1):461–488.

Ni, T., Eysenbach, B., and Salakhutdinov, R. (2022). Recurrent model-free RL can be a strong baseline for many pomdps. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pages 16691–16723. PMLR.

Ni, Y. and Liu, Z. (2008). Bounded-Parameter Partially Observable Markov Decision Processes. In *ICAPS*, pages 240–247. AAAI.

Ni, Y. and Liu, Z. (2013). Policy iteration for bounded-parameter POMDPs. *Soft Comput.*, 17(4):537–548.

Nilim, A. and Ghaoui, L. E. (2005). Robust control of Markov decision processes with uncertain transition matrices. *Oper. Res.*, 53(5):780–798.

Omlin, C. W. and Giles, C. L. (1996). Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9(1):41–52.

Osogami, T. (2015). Robust partially observable Markov decision process. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 106–115. JMLR.org.

Peters, J. and Schaal, S. (2006). Policy gradient methods for robotics. In *IROS*, pages 2219–2225. IEEE.

Petrik, M. and Subramanian, D. (2014). RAAM: the benefits of robustness in approximating aggregated mdps in reinforcement learning. In *NIPS*, pages 1979–1987.

Poupart, P. and Boutilier, C. (2003). Bounded finite state controllers. In *Advances in Neural Information Processing Systems (NIPS)*, pages 823–830. MIT Press.

Saxe, A. M., McClelland, J. L., and Ganguli, S. (2014). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *ICLR*.

Silver, D. and Veness, J. (2010). Monte-carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2164–2172. MIT Press.

Smith, T. and Simmons, R. (2004). Heuristic search value iteration for POMDPs. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 520–527. AUAI Press.

Spaan, M. T. J. (2012). Partially Observable Markov Decision Processes. In Wiering, M. and van Otterlo, M., editors, *Reinforcement Learning: State-of-the-Art*, pages 387–414. Springer Berlin Heidelberg, Berlin, Heidelberg.

Suilen, M., Jansen, N., Cubuktepe, M., and Topcu, U. (2020). Robust policy synthesis for uncertain pomdps via convex optimization. In *IJCAI*, pages 4113–4120. ijcai.org.

Suilen, M., Simão, T. D., Parker, D., and Jansen, N. (2022). Robust anytime learning of Markov decision processes. In *NeurIPS*.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic robotics*. Intelligent robotics and autonomous agents. MIT Press.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS*, pages 23–30. IEEE.

Wang, A., Li, A. C., Klassen, T. Q., Icarte, R. T., and McIlraith, S. A. (2023). Learning belief representations for partially observable deep RL. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 35970–35988. PMLR.

Wierstra, D., Förster, A., Peters, J., and Schmidhuber, J. (2007). Solving deep memory pomdps with recurrent policy gradients. In *ICANN (1)*, volume 4668 of *Lecture Notes in Computer Science*, pages 697–706. Springer.

Wiesemann, W., Kuhn, D., and Rustem, B. (2013). Robust Markov decision processes. *Math. Oper. Res.*, 38(1):153–183.

Zeng, Z., Goodman, R. M., and Smyth, P. (1993). Learning finite state machines with self-clustering recurrent networks. *Neural Comput.*, 5(6):976–990.

Appendix

A Robust Policy Evaluation

In this appendix, we show that robust policy evaluation (Definition 6) indeed provides a conservative upper bound to the actual robust Bellman equation Equation (6):

$$\begin{aligned}\underline{\mathcal{V}}^{\pi_f}_{\star}(\langle s,n\rangle) = \sum_{a\in A}\Big(&\delta(a\,|\,n,O(s))C(s,a)\\ &+\sup_{T(s,a)\in\mathcal{T}(s,a)}\Big\{\sum_{s'\in S}\sum_{n'\in N}T(s'\mid s,a)\delta(a\mid n,O(s))[n'=\eta(n,O(s))]\underline{\mathcal{V}}^{\pi_f}_{\star}(\langle s',n'\rangle)\Big\}\Big).\end{aligned}$$

Definition 6 (Robust policy evaluation (extended))**.** *Given an RPOMDP $\mathcal{M}=\langle S,A,\mathcal{T},C,Z,O\rangle$ with initial state distribution (belief) b_0 and an FSC $\pi_f=\langle N,n_0,\eta,\delta\rangle$, the robust state-values of $\mathcal{M}$ under F, $\underline{\mathcal{V}}^{\pi_f}\colon S\times N\to\mathbb{R}$, are given by the state-values in the robust Markov chain $\langle S\times N,\mathcal{T}^{\pi_f},C^{\pi_f}\rangle$ where the states are the product of RPOMDP states S and FSC memory nodes N, and the robust transition and cost functions are:*

$$\begin{aligned}\mathcal{T}^{\pi_f}(\langle s',n'\rangle\mid\langle s,n\rangle,a)&=\mathcal{T}(s'\,|\,s,a)\delta(a\,|\,n,O(s))[n'=\eta(n,O(s))]\\ \mathcal{T}^{\pi_f}(\langle s',n'\rangle\mid\langle s,n\rangle)&=\sum_{a\in A}\mathcal{T}^{\pi_f}(\langle s',n'\rangle\mid\langle s,n\rangle,a)\\ C^{\pi_f}(\langle s,n\rangle)&=\sum_{a\in A}\delta(a\,|\,n,O(s))C(s,a),\end{aligned}$$

where addition and multiplication over intervals follow the standard rules for interval arithmetic (Hickey et al., 2001). Then, the robust state-values are defined as:

$$\underline{\mathcal{V}}^{\pi_f}(\langle s,n\rangle)=C^{\pi_f}(\langle s,n\rangle)+\sup_{T^{\pi_f}(\langle s,n\rangle)\in\mathcal{T}^{\pi_f}(\langle s,n\rangle)}\sum_{s'\in S}\sum_{n'\in N}T^{\pi_f}(\langle s',n'\rangle\mid\langle s,n\rangle)\underline{\mathcal{V}}^{\pi_f}(\langle s',n'\rangle).\tag{9}$$

For an initial belief b_0 and initial memory node n_0, both state-based value functions can be extended to the value in the initial belief:

$$\begin{aligned}\underline{\mathcal{V}}^{\pi_f}_{\star}(\langle b_0,n_0\rangle)&=\sum_{s\in S}b_0(s)\underline{\mathcal{V}}^{\pi_f}_{\star}(\langle s,n_0\rangle),\\ \underline{\mathcal{V}}^{\pi_f}(\langle b_0,n_0\rangle)&=\sum_{s\in S}b_0(s)\underline{\mathcal{V}}^{\pi_f}(\langle s,n_0\rangle).\end{aligned}$$

Theorem 1. *Given the FSC π_f, the robust state-values $\underline{\mathcal{V}}^{\pi_f}$ of the robust Markov chain $\langle S\times N,b_0^{\pi_f},T^{\pi_f},C^{\pi_f}\rangle$ provide a (conservative) upper bound on the value $\underline{\mathcal{V}}^{\pi_f}_{\star}$ of F under (s,a)-rectangularity in the RPOMDP with initial belief b_0. That is, $\underline{\mathcal{V}}^{\pi_f}_{\star}(\langle s,n\rangle)\leq\underline{\mathcal{V}}^{\pi_f}(\langle s,n\rangle)$, and consequently $\underline{\mathcal{V}}^{\pi_f}_{\star}(\langle b_0,n_0\rangle)\leq\underline{\mathcal{V}}^{\pi_f}(\langle b_0,n_0\rangle)$.*

Proof. We show that $\underline{\mathcal{V}}^{\pi_f}(\langle s,n\rangle)\geq\underline{\mathcal{V}}^{\pi_f}_{\star}(\langle s,n\rangle)$ for all $(\langle s,n\rangle)\in S\times N$. Recall Equation (9). Omitting the constant $C^{\pi_f}(\langle s,n\rangle)$, we rewrite the supremum as follows:

$$\begin{aligned}&\sup_{T^{\pi_f}(\langle s,n\rangle)\in\mathcal{T}^{\pi_f}(\langle s,n\rangle)}\sum_{s'\in S}\sum_{n'\in N}T^{\pi_f}(\langle s',n'\rangle\mid\langle s,n\rangle)\underline{\mathcal{V}}^{\pi_f}(\langle s',n'\rangle)\\ &=\sup_{T^{\pi_f}(\langle s,n\rangle)\in\mathcal{T}^{\pi_f}(\langle s,n\rangle)}\sum_{s'\in S}\sum_{n'\in N}\sum_{a\in A}T^{\pi_f}(\langle s',n'\rangle\mid\langle s,n\rangle,a)\underline{\mathcal{V}}^{\pi_f}(\langle s',n'\rangle)\\ &=\sup_{T^{\pi_f}(\langle s,n\rangle)\in\mathcal{T}^{\pi_f}(\langle s,n\rangle)}\sum_{s'\in S}\sum_{n'\in N}\sum_{a\in A}T(s'\,|\,s,a)\delta(a\,|\,n,O(s))[n'=\eta(n,O(s))]\underline{\mathcal{V}}^{\pi_f}(\langle s',n'\rangle)\\ &=\sup_{T^{\pi_f}(\langle s,n\rangle)\in\mathcal{T}^{\pi_f}(\langle s,n\rangle)}\sum_{a\in A}\sum_{s'\in S}\sum_{n'\in N}T(s'\,|\,s,a)\delta(a\,|\,n,O(s))[n'=\eta(n,O(s))]\underline{\mathcal{V}}^{\pi_f}(\langle s',n'\rangle).\end{aligned}$$

Under (s, a)-rectangularity of the RPOMDP, we can continue rewriting this supremum as:

$$\begin{aligned}
&\sup_{T^{\pi_f}(\langle s,n\rangle)\in\mathcal{T}^{\pi_f}(\langle s,n\rangle)} \sum_{a\in A}\sum_{s'\in S}\sum_{n'\in N} T(s' \,|\, s,a)\delta(a \,|\, n, O(s))[n' = \eta(n, O(s))]\underline{\mathcal{V}}^{\pi_f}(\langle s', n'\rangle) \\
&= \sum_{a} \sup_{T^{\pi_f}(\langle s,n\rangle,a)\in\mathcal{T}^{\pi_f}(\langle s,n\rangle,a)} \sum_{s'\in S}\sum_{n'\in N} T(s' \,|\, s,a)\delta(a \,|\, n, O(s))[n' = \eta(n, O(s))]\underline{\mathcal{V}}^{\pi_f}(\langle s', n'\rangle) \\
&\geq \left(\sum_{a} \sup_{T(s,a)\in\mathcal{T}(s,a)} \sum_{s'\in S}\sum_{n'\in N} T(s' \,|\, s,a)\delta(a \,|\, n, O(s))[n' = \eta(n, O(s))]\underline{\mathcal{V}}^{\pi_f}(\langle s', n'\rangle\right).
\end{aligned}$$

Inserting the constant $C^{\pi_f}(\langle s,n\rangle)$ again, we derive:

$$\begin{aligned}
&\sum_{a}\Bigg(\delta(a \mid n, O(s))C(s,a) \; + \\
&\sup_{T(s,a)\in\mathcal{T}(s,a)} \sum_{s'\in S}\sum_{n'\in N} T(s' \,|\, s,a)\delta(a \,|\, n, O(s))[n' = \eta(n, O(s))]\underline{\mathcal{V}}^{\pi_f}(\langle s', n'\rangle\Bigg) = \underline{\mathcal{V}}^{\pi_f}_{\star}(\langle s,n\rangle).
\end{aligned}$$

Since the inequality holds for each state-memory node pair, we also have for some initial belief b_0 and initial memory node n_0 that $\underline{\mathcal{V}}^{\pi_f}_{\star}(\langle b_0, n_0\rangle) \leq \underline{\mathcal{V}}^{\pi_f}(\langle b_0, n_0\rangle)$. □

Intuitively, the robust Markov chain, and thus its value function $\underline{\mathcal{V}}^{\pi_f}$, operates under $(\langle s,n\rangle, a)$-rectangularity, meaning nature may choose a probability distribution for each state $s \in S$, memory node $n \in N$, and action $a \in A$ independently. In the RPOMDP, and thus the associated value function $\underline{\mathcal{V}}^{\pi_f}_{\star}$, nature operates under (s, a)-rectangularity, meaning it chooses probability distributions independently of the state s and action a but is restricted to choose the same probability distribution for each memory node n. The latter is more restrictive to nature, hence nature has fewer options to adversarially play against the agent. As a result, the agent's cost may be lower than when nature's choices depend on the agent's memory.

This difference in semantics may also be explicitly encoded in a partially observable stochastic game by making the agent's memory either observable or unobservable to nature (Bovy et al., 2024).

B Supervision Policies

This section elaborates on the POMDP approximations used for computing the supervision policies.

QMDP. The Q_{MDP} algorithm (Littman et al., 1995) is an effective method to transform an optimal MDP policy to a POMDP policy by weighting the (optimal) action values Q^* of the MDP to the current belief $b \in \mathcal{B}$ in the POMDP $M \in \mathcal{M}$:

$$Q_{\text{MDP}}(b,a) = \sum_{s\in S} b(s)Q^*(s,a) = \sum_{s\in S} b(s)\left(C(s,a) + \sum_{s'\in S} T(s' \mid s,a)V^*_{\text{MDP}}(s')\right),$$

where V^*_{MDP} is the optimal value of the MDP underlying the POMDP M.

Fast-informed bound. The fast-informed bound (FIB; Hauskrecht, 2000) is an approximation of the optimal value of the POMDP. It is tighter than the one given by Q_{MDP} since it includes a sum over the observation of the next state. The Q values of FIB are defined as:

$$Q_{\text{FIB}}(b,a) = \sum_{s\in S} b(s)\alpha^a(s) = \sum_{s\in S} b(s)\left(C(s,a) + \sum_{z\in Z}\min_{a'\in A}\sum_{s'\in S} T(s' \mid s,a)[z = O(s')]\alpha^{a'}(s')\right),$$

where $\alpha^a : S \to \mathbb{R}$ for each $a \in A$ is a linear function, or *alpha-vector*, updated via:

$$\alpha^a_{i+1}(s) = C(s,a) + \sum_{z\in Z}\max_{a'}\sum_{s\in S} T(s' \mid s,a)[z = O(s')]\alpha^{a'}_i(s').$$

C Tools and Hyperparameters

We use the tools Storm Hensel et al. (2022) for parsing the models and PRISM Kwiatkowska et al. (2011) to compute the RMDP values for the lower bounds in Figure 3 and for robust policy evaluation. We build and train the RNN and the QBN using TensorFlow Abadi et al. (2016). The RNN cell is a gated recurrent unit (GRU, Cho et al., 2014). Unless specified otherwise, we initialize with a concrete POMDP instance $M \in \mathcal{M}$, where the intervals of the uncertainty sets are resolved to a value in the middle of the interval $[i, j]$ given by ${}^{i+j}/_{2}$, taking into account that transitions must sum to 1. For all the experiments, the simulation batch size is set to $I = 256$, the maximum simulation length is set to $H = 200$, and we run for a maximum of 50 iterations. The RNN and QBN use an Adam optimizer (Kingma and Ba, 2015) with a learning rate of $1 \cdot 10^{-3}$. The hidden size of the RNNs was set to $d = 16$. The experiments ran inside a Docker container on a Debian 12 machine. Our infrastructure includes an AMD Ryzen Threadripper PRO 5965WX machine with 512 GB of RAM. We train the neural networks on the CPU. The different seeds for the RNN-based methods were executed in parallel, each running on a single core. Multi-threading in the Gurobi LP solver (Gurobi Optimization, LLC, 2023) used by SCP was enabled. In our initial tests, we considered hidden sizes $d \in \{3, 16, 64\}$, batch sizes $I \in \{128, 256, 512\}$, learning rates in the range of $[1 \cdot 10^{-2}, 1 \cdot 10^{-4}]$, and different number of iterations before arriving at our final values. We used the same infrastructure and experimental setup across methods.

D Network Architectures

In this section, we provide more details on the neural network architectures. Our *post hoc* QBN approach largely follows Koul et al. (2019) and Carr et al. (2021), apart from differences mentioned in Section 5.4. We used a batch size of 32 for both networks during stochastic gradient descent.

D.1 QBN

Similar to prior work (Carr et al., 2021), we employ a quantized bottleneck network (QBN; Koul et al., 2019). It consists of an encoder $E\colon \hat{\mathcal{H}} \to [-1, 1]^l$ that maps the output of the RNN to a latent encoding with $\tanh$ activation, where l is the latent encoding dimension. The latent encoding is then quantized by a function $q\colon [-1, 1]^l \to \beta^{B_h}$, where β is the finite set of possible discrete values, for instance, $\beta = \{-1, 0, 1\}$ for three-level quantization. The *bottleneck dimension* B_h is the number of quantized neurons. Lastly, there is a decoder $D\colon \beta^{B_h} \to \hat{\mathcal{H}}$ to reconstruct the input given the quantized encoding. The QBN represents a function $\mathcal{Q}\colon \hat{\mathcal{H}} \to \hat{\mathcal{H}}$ where $\mathcal{Q}(\hat{h}) = D(q(E(\hat{h})))$ for all $\hat{h} \in \hat{\mathcal{H}}$. We train the QBN to minimize the reconstruction loss, *i.e.*, mean-squared error, on the RNN's memory representations derived from the histories in $\mathcal{D}$. The finite set of memory nodes extracted is formed by the Cartesian product $N = \times_{B_h} \beta$, and $n = q(E(\hat{h})) \in N$ is the discrete memory representation. Therefore, the extracted FSC's memory size $|N| = |\beta|^{B_h}$ is directly controlled by B_h and the quantization level, *i.e.*, size of the set β. Note that the quantization level can be changed to be 2-level, *i.e.*, with $B = \{-1, 1\}$ using the sign function as q, resulting in different controller sizes.

To ensure the encoder E maps to $[-1, 1]$ we use $\tanh$ activation. The gradient of this activation function is close to one around the zero input. Thus, for the 3-level quantization, we use a version $\tanh_{flat}$ of the $\tanh$ function in the encoder that is flatter around the zero input to allow for easier learning of quantization level 0, given by (Koul et al., 2019):

$$\tanh_{flat}(x) = 1.5 \tanh(x) + 0.5 \tanh(-3x).$$

To allow the gradient to pass through the quantization layer, we employ a simple straight-trough estimator that treats the quantization as an identity function during back-propagation (Li and Liu, 2016). The quantization activation function was provided by the Larq library (Geiger and Team, 2020). The encoder and decoder use a symmetrical architecture with $\tanh$ activation. The networks were quite small. The input and output sizes of the encoder and decoder were set to the hidden size d of the RNN, with intermediate layers of sizes $8 \cdot B_h$ and $4 \cdot B_h$.

D.2 RNN

We use a Gated Recurrent Unit (GRU, Cho et al. (2014)) as the RNN architecture. Although there is no clear consensus between the Long Short-Term Memory (LSTM, Hochreiter and Schmidhuber

Table 1: Dimensions of each benchmark environment.

Instances	*Aircraft*	*Avoid*	*Evade*	*Intercept*
$\lvert S\rvert$	13552	10225	4261	4802
$\lvert Z\rvert$	37	6968	2202	2062
$\lvert A\rvert$	5	4	5	4

(1997)) architecture and the GRU, the latter has fewer parameters than the LSTM but does have the ability to learn long-term dependencies due to the *forget gate*. The forget gate is known to combat *vanishing gradients* that occur through the variant of stochastic gradient descent employed for sequential models, known as backpropagation through time. The inputs to the RNN were put through a learnable embedding layer. We trained the RNN policy using the method in section 5.3 with a categorical cross-entropy loss implemented in TensorFlow. To prevent exploding gradients in the RNN, we use a norm-clipped gradient and orthogonal weight initialization (Saxe et al., 2014) in the recurrent layer of the GRU, as recommended by Ni et al. (2022).

E Benchmark Descriptions

In this section of the Appendix, we describe the benchmarks studied in the paper. All environments are adapted with uncertain transition functions. The grid-world environments model the probability of taking multiple steps instead of a single one for each possible moving action, to which we assign the interval $[0.1, 0.4]$. In *Aircraft*, we have two uncertainties: the probabilities of the pilot's responsiveness and of the adversary changing direction, both mapping to the same $[0.6, 0.8]$ interval. The dimensions of the benchmarks are given in Table 1. We specify the dimensions of the grid-worlds to the same sizes as set in Carr et al. (2023).

E.1 Aircraft Collision Avoidance

We consider a discretized and model-uncertain version of the aircraft collision avoidance problem (Kochenderfer et al., 2015) as introduced in Cubuktepe et al. (2021).

Aircraft. We follow the discretization procedure exactly and base our model on Cubuktepe et al. (2021), but slightly adapt it for our expected cost formulation. The objective is to minimize the expected cost, which models avoiding a collision with an intruder aircraft while taking into account partial observability (sensor errors) and uncertainty with respect to future paths. Crashes incur an additional cost of 100 over the usual cost incurred of 1 for each action. Furthermore, the only sink states are the goal states $G \subseteq S$.

E.2 Grid-worlds

We consider the grid-worlds introduced by Junges et al. (2021) but reformulate them as an expected cost (SSP) objective. All actions incur a cost of $c = 1$, with an additional penalty of $c = 100$ when in a bad state. Once again, the only sink states of the RPOMDP are the goal states $G \subseteq S$.

Avoid. The *Avoid* benchmark models a scenario where a moving agent must keep a distance from patrolling adversaries that move with uncertain speed. Additionally, its sensor yields partial information about the position of the patrolling adversaries. The agent may exploit knowledge over the predefined routes of the patrolling adversaries.

Evade. *Evade* is a scenario where a robot needs to reach a destination and evade a faster adversary. The agent has a limited range of vision but may scan the whole grid instead of moving, incurring the same cost as moving. A certain safe area is only accessible by the agent.

Intercept. *Intercept* is the opposite of Evade because the agent aims to meet another robot before it leaves the grid via one of two available exits. Once the target robot has exited, the agent incurs an

Clustering Supervision		RFSCNET QRNN Q_{MDP}	RFSCNET `k-means++` Q_{MDP}	RFSCNET QBN Q_{MDP}	RFSCNET QBN Q_{FIB}	baseline QBN Q_{MDP}	baseline QBN Q_{FIB}
Aircraft	med	$\times$	103.30	**102.95**	103.41	105.91	105.83
	min	$\times$	102.03	101.91	**101.88**	104.66	104.60
Avoid	med	19.90	**18.51**	20.19	19.43	18.83	18.70
	min	18.62	**18.16**	18.57	18.53	18.39	18.35
Evade	med	$\times$	37.61	37.96	38.20	**36.64**	36.67
	min	$\times$	36.65	36.98	37.07	**36.06**	36.11
Intercept	med	66.00	**56.20**	79.71	79.51	110.47	109.36
	min	42.92	**34.95**	45.37	48.16	82.30	81.66

Table 2: Evaluation across multiple configurations for RFSCNET and a baseline trained on the (nominal) POMDP that resides in the middle of the uncertainty set. The values represent median (med.) and minimum (min.) robust values from the best FSCs computed of each run across 20 seeds. QRNN represents training the QBN *end-to-end*, see Section 5.4. $\times$ indicates a run failed. **Bold** indicates the best (med/min) performance for each environment, *i.e.*, across the rows.

additional penalty of $c = 100$ for each step before reaching a goal state. On top of the view radius, the agent observes a corridor in the center of the grid.

F Extended Experimental Evaluation

Below, the is trained in the middle of the uncertainty set, as specified in Appendix C.

F.1 Configuration Study

Due to its modularity, RFSCNET allows for different configurations that may have an effect on its performance. In Table 2, we collect median and minimum results across different configurations of RFSCNET. The combination of Q_{MDP} and `k-means++` proves best, which is what is shown in the table of Figure 3 in the paper. QRNN, the method that uses a QBN trained *end-to-end*, did not perform successfully on all environments. This is due to instability during training, caused by updating the QBN's parameters with the gradients calculated from training the RNN, see Section 5.3. However, by directly encoding the clustering of the QBN into the RNN architecture during training, we observe an improvement in the median and minimum performance on the two successful runs, *Avoid* and *Intercept*, over training the QBN *post hoc*. We also show results for the baseline when trained with the two different supervision policies.

The full results in the form of boxplots are depicted in Figure 4.

F.2 Extended Analysis on Various Memory Sizes

In this subsection, we study the memory comparison between RFSCNET and SCP in more detail. We chose *Aircraft* and *Evade*, as SCP appeared most consistent on these benchmarks. In this study, RFSCNET ran with a *post hoc* QBN and Q_{MDP} as supervision policy. We run for the maximal memory settings that we can restrict RFSCNET to when using the QBN, namely the values in the set $k \in \{3, 4, 8, 9, 16\}$, which is the size of the sets that can be found through binary $k \leq 2^{B_h}$ or ternary quantization $k \leq 3^{B_h}$, see also Appendix D.1. Figure 5 extends the right-side plot in Figure 3 with statistical details. Namely, we plot the standard deviation around the median values in the heatmap and show the global min and max of each method. We observe very stable performance for our method across the various memory sizes. Both on *Aircraft* and *Evade*, SCP shows relatively stable performance across memory sizes up to $k = 9$. However, also on these benchmarks the performance drops when the required memory is set to a high level. Evidently, RFSCNET does not suffer from the same phenomenon. Furthermore, RFSCNET outperforms *Aircraft* on all memory settings and performs similarly or better than SCP on *Evade*.

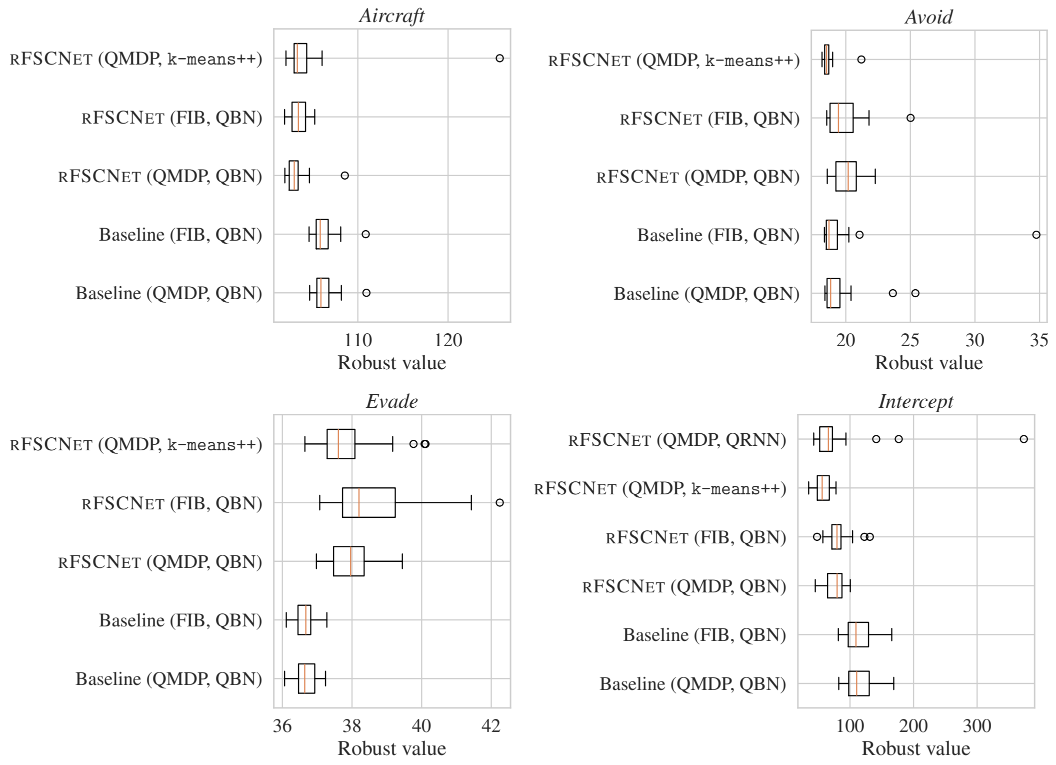

Figure 4: Comparison of the robust values between rFSCNet and a baseline trained across configurations. For *Avoid*, we plot without the run with QRNN, as it produces large outliers.

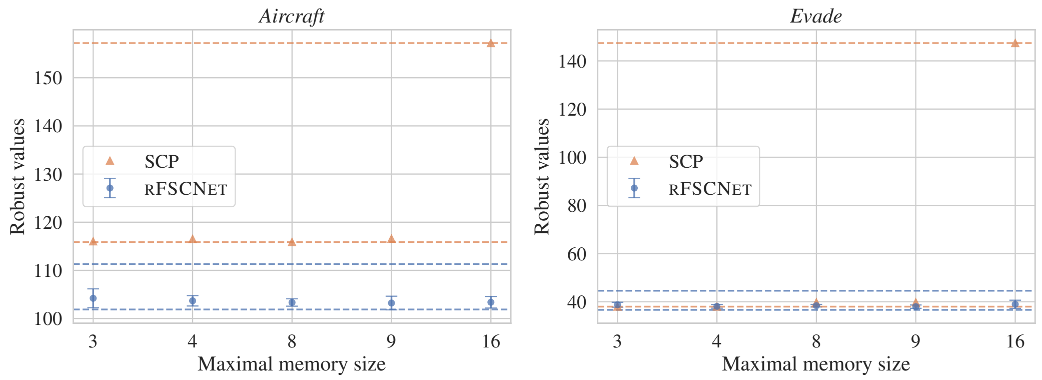

Figure 5: Comparison of the robust values between rFSCNet and SCP. For rFSCNet, the error bars depict the standard deviation, and the dotted line shows for each method the global minimum and global maximum.

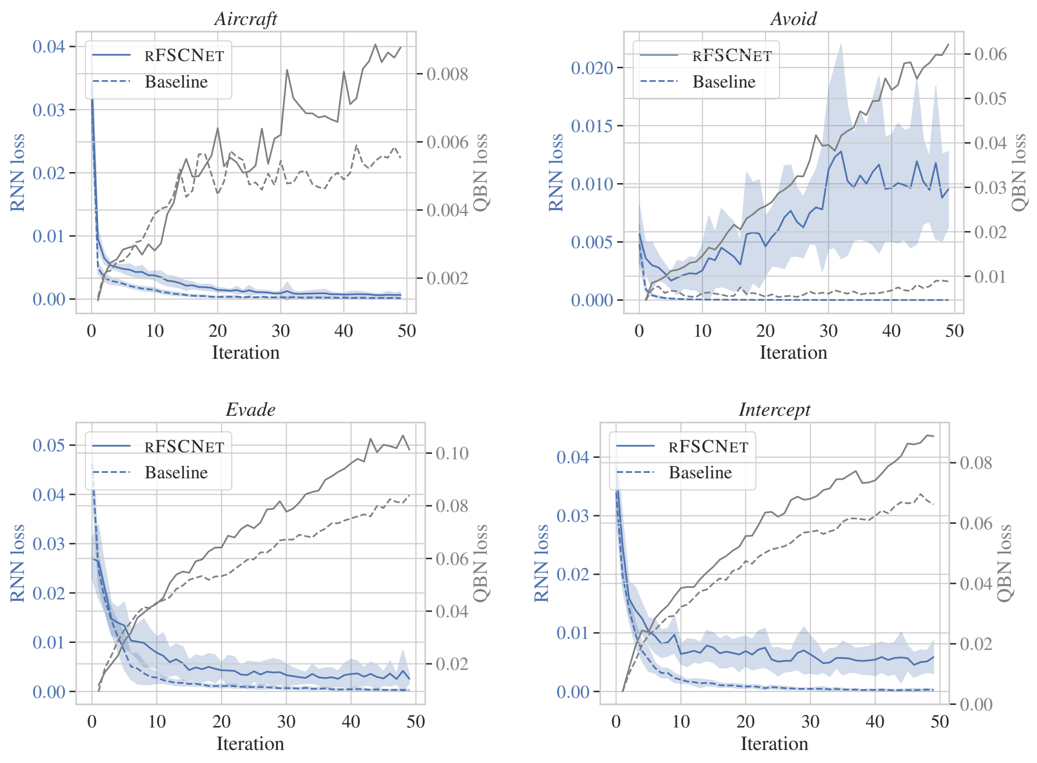

Figure 6: Comparison of RNN and QBN losses between rFSCNet and a baseline over the iterations. The line shows the mean over 20 seeds, and we plot the standard deviation around the mean for the RNN loss. On *Aircraft* and *Evade*, there is only a slight difference between the losses of the baseline and rFSCNet. On the right, on *Avoid*, a big difference is visible.

F.3 Loss Comparison

Figure 6 shows the RNN and QBN losses of the baseline and rFSCNet on *Aircraft* and *Avoid*. Both runs employ a QBN and use FIB as the supervision policy. The QBN is trained individually from the RNN, *i.e.*, *post hoc*. The results show that as the RNN loss decreases, the QBN reconstruction loss increases. This tells us that it gets increasingly hard to compress the hidden states of the RNN as they get more refined. An intuition is that the RNN learns to use a larger part of $\hat{\mathcal{H}}$ to represent the hidden states as training progresses, therefore making it harder to cluster the hidden states. Alternatively, one could train the QBN *end-to-end*. However, as we elaborate in Appendix F.1, this approach suffers from instability during training and, therefore, did not successfully run on all environments.

G Run Times

For completeness, we report the run times of each procedure for every environment in Table 3. We report the run times using a QBN trained *post hoc* and FIB, which is the most expensive configuration in terms of computations. The RNN-based run times are averaged over the 20 seeds. We would like to note that the times given for SCP are *user time* and do not account for the total *CPU time* incurred by multi-threading. For the RNN-based approach, we see that the baseline is slightly faster in every environment except for *Avoid*, as it does not execute Step 4 of our method from Section 6.2, and does not need to recompute the supervision policies as the POMDP is fixed. The run times naturally increase for larger FSCs because the Markov chain used for robust policy evaluation grows larger when the FSC has more memory nodes. Our method spends the majority of its time in its robust evaluation, executed by robust value iteration (robust dynamic programming) in PRISM. Additionally, extraction from the QBN can take longer, as $|N|$ forward passes of the RNN are required. Typically, the worse the policies found, the longer it takes to perform robust dynamic programming. By

Algorithm 1 PIP algorithmic set-up

```
1: repeat
2:     Compute $\pi_f$ for POMDP $M$.
3:     Compute $\underline{\mathcal{V}}^{\pi_f}$ and find $M'$, through Equation (10), and set $M \leftarrow M'$.
4: until convergence
```

comparing the run times of the baseline to our robust method, we see that the heuristic for finding worst-case instances does increase execution time. Finally, we would like to point out that the run times for SCP could be summed for a fair comparison, as running SCP for only $k = 9$ yields much worse results than for $k = 3$.

Table 3: Average run times in seconds across 20 runs for rFSCNet and the baseline, and the *user time* of the SCP method on each environment. Thus, both run-times represent a form of user time.

	Robust ($k \leq 9$)	Baseline ($k \leq 9$)	SCP ($k = 3$)	SCP ($k = 9$)
Aircraft	10562.51 ± 156.03	2518.39 ± 158.76	1133.8	2169.3
Avoid	9987.82 ± 1209.82	12778.36 ± 1164.16	2167.9	6217.9
Evade	5157.85 ± 131.58	1281.85 ± 58.73	872.7	3674.1
Intercept	2501.95 ± 16.00	1624.33 ± 12.49	1884.9	3243.5

H Convergence of PIP (WIP)

Definition 7 (Adversarial POMDP)**.** *Given an FSC policy π_f, the* adversarial POMDP $M' \in \mathcal{M}$ *is the POMDP $M' = \langle S, A, \underline{T}, C, Z, O\rangle$ with the worst-case transition function $\underline{T} \in \mathcal{T}$ with respect to the robust value function $\underline{\mathcal{V}}^{\pi_f}$:*

$$\underline{T} \in \operatorname*{argmax}_{T \in \mathcal{T}} \mathcal{T}^{\pi_f} \underline{\mathcal{V}}^{\pi_f} \tag{10}$$

$$\begin{aligned}
&\inf_{\pi_f \in \Pi} \sup_{T \in \mathcal{T}} J_T^{\pi} \\
&= \inf_{\pi_f \in \Pi} \sup_{T \in \mathcal{T}} \mathbb{E}_{\pi, T}\left[\rho_{\Diamond G}\right] \\
&= \min_{\pi_f \in \Pi} \max_{T \in \mathcal{T}} \left[C^{\pi_f} + T^{\pi_f} V_T^{\pi_f}\right] \\
&\leq \min_{\pi_f \in \Pi} \left[C^{\pi_f} + \max_{T^{\pi_f} \in \mathcal{T}^{\pi_f}} \left\{T^{\pi_f} V_T^{\pi_f}\right\}\right]
\end{aligned}$$

Then, the left-hand side of PIP computes a new FSC π_f' for the given adversarial POMDP $M' \in \mathcal{M}$. Therefore, under our objective of minimizing the worst-case expected costs, we are interested to know that there is a (monotonic) improvement between $\underline{\mathcal{V}}^{\pi_f}$ and $\underline{\mathcal{V}}^{\pi_f'}$, e.g., that π_f' is a better robust policy than π_f. That is, that $\underline{\mathcal{V}}^{\pi_f} \succeq \underline{\mathcal{V}}^{\pi_f'}$, where $\succeq$ denotes the entry-wise comparison operator for all states and memory nodes, such that $\forall \langle s, n\rangle \in S \times N \colon \underline{\mathcal{V}}^{\pi_f}(\langle s, n\rangle) \geq \underline{\mathcal{V}}^{\pi_f'}(\langle s, n\rangle)$. From this, it would follow that $\sum_{s \in S} b_0(s) \underline{\mathcal{V}}^{\pi_f}(\langle s, n_0\rangle) \geq \sum_{s \in S} b_0(s) \underline{\mathcal{V}}^{\pi_f'}(\langle s, n_0\rangle)$.

Policy improvement with respect to $\underline{T}$ will improve

Algorithm 2 rFSCNet's algorithmic set-up

```
1: Input: Robust POMDP $\mathcal{M}$, stopping criterion $c$
2: repeat
3:     Simulate $\pi_M$ using Section 5.1 on $M$ and store histories and action distributions in $\mathcal{D}$.
4:     Train $\pi^\phi$ on $\mathcal{D}$ to match $\pi_M$ via BPTT.
5:     Discretize $\pi^\phi$ into $\pi_f$.
6:     Compute $\underline{\mathcal{V}}_f^\pi$ through Definition 5.
7:     Solve LP from Equation (8) to find $T \in \mathcal{T}$ adversarial to $\underline{\mathcal{V}}_f^\pi$ and instantiate a new $M' \in \mathcal{M}$.
8: until $c$ is satisfied
```