A NBF Pseudocode

We present the full pseudocode for Neural Bayesian Filtering. The algorithm updates the input belief embedding by generating and then simulating n particles for a single step. These particles are weighed according to the probability of input observation y given the environment dynamics G and control variable π.

Algorithm 1: Neural Bayesian Filtering

input : θ — Belief Embedding, y — Observation, G — Environment, π — Control, (ψ, ϕ) — Model Parameters, n — Number of Particles, φ — function to estimate $\mathbb{E}_p[\varphi]$

output : θ' — Updated Belief Embedding

1 $z_{1:n} \overset{\text{iid}}{\sim} \mathcal{N}(0_d, I_d)$ `// n samples of d-dimensional Gaussian noise`

2 $x_{1:n} \leftarrow f_\psi(z_{1:n}; \theta)$ `// Generate particles from belief embedding`

3 **for** $i \leftarrow 1 \dots n$ **do**

4 $\quad x'_i \sim T_G(x_i, \pi)$

5 $\quad w_i \leftarrow T_G(x_i, \pi)[x'_i] \cdot H_G(x_i, x'_i)[y]$

6 **end**

7 $\hat{\mu}^{(n)}(\varphi) \leftarrow \frac{\sum_{i=1}^n w_i \varphi(x_i)}{\sum_{i=1}^n w_i}$ `// Estimate target expectation`

8 **return** $\mathcal{E}_\phi(x'_{1:n}, \texttt{normalize}(w_{1:n})), \hat{\mu}^{(n)}(\varphi)$ `// Output embedding and estimate` $\mathbb{E}_p[\varphi]$

The algorithm can optionally estimate the expectation (Line 7) of a target function φ over the belief state. Next, we show theoretical results about the convergence of the estimate when NBF has access to a perfect belief embedding model.

B Proof of Theorem 4.1

The theorem requires two main assumptions: a perfect belief embedding model and ϵ-global observation positivity. A *perfect* embedding model has of two properties. First, for every $p(x) \in \mathcal{P}_G^\Pi$, there exists a $\theta_p^* \in \mathbb{R}^m$ such that $p_{\theta_p^*}(x) = p(x)$. Second, for any finite collection of samples $x_{1:n} \in X^n$ and weights $w_{1:n} \in \mathbb{R}^n, \sum_{i=1}^n w_i = 1$, if the weighted empirical distribution induced by $(x_{1:n}, w_{1:n})$, $\hat{p}(x)$ is equal to $p(x)$, then $\mathcal{E}_\phi(x_{1:n}, w_{1:n}) = \theta_p^*$.

Define the successor pairs of state space X given (G, π) as $\text{succ}_{G,\pi}(X) \overset{\text{def}}{=} \{(x, x') : x, x' \in X, T_G(x, \pi)[x'] > 0\}$. *$\epsilon$-global observation positivity* of (G, π) states that there exists an $\epsilon \in (0, 1]$ such that for all $y \in Y$ and $(x, x') \in \text{succ}_{G,\pi}(X)$:

$$T(x, \pi)[x'] \cdot H_G(x, x')[y] \geq \epsilon$$

This means that for G and π, every transition has some probability of generating any observation y. In practice, a small ϵ is sufficient to guarantee NBF's consistency. Without this simplifying assumption, there can be a non-zero (but vanishing) chance that the sample weights in the estimator $\hat{\mu}^{(n)}(\varphi)$ are all equal to zero. In a practical setting where this happens, one could repeat lines 2-6 of Algorithm 1 until some $w_i > 0$. With finite state and observation spaces and these assumptions, we can prove the almost-sure convergence of NBF to the target posterior. First, we provide a lemma for the strong law of large numbers for self-normalizing importance samplers. The proof is an adaptation of the one found in Owen [2013].

Lemma B.1 (Strong Law for Self-Normalized Importance Sampling Estimators)**.** *Given a finite sample space X, let $p(x)$ be a target distribution and $q(x)$ be a proposal distribution such that $q(x) > 0$ whenever $p(x) > 0$. Let $W : X \to (0, 1]$ be a weight function such that $p(x) = c \cdot W(x)q(x)$ for normalization constant c. Finally, let $\varphi : X \to \mathbb{R}$ be any bounded function.*

Draw samples $x_1, \dots, x_n \overset{iid}{\sim} q$. Let $w_i = W(x_i)$ and

$$\hat{\mu}^{(n)}(\varphi) = \frac{\sum_{i=1}^n w_i \varphi(x_i)}{\sum_{i=1}^n w_i}$$

be the self-normalized importance sampling estimate of $\mathbb{E}_p[\varphi]$. *Then,*

$$\hat{\mu}^{(n)}(\varphi) \xrightarrow{a.s.} \mathbb{E}_p[\varphi]$$

as $n \to \infty$.

Proof. Since the pairs (w_i, x_i) are i.i.d., the numerator $S_n \overset{\text{def}}{=} \sum_{i=1}^n w_i \varphi(x_i)$ and denominator $B_n \overset{\text{def}}{=} \sum_{i=1}^n w_i$ of the estimate are i.i.d sums. With bounded φ and W, we can apply Kolmogorov's Strong Law of Large Numbers to S_n and B_n, which gives

$$\frac{S_n}{n} \xrightarrow{\text{a.s.}} \mathbb{E}_q[\varphi W], \qquad \frac{B_n}{n} \xrightarrow{\text{a.s.}} \mathbb{E}_q[W]$$

as $n \to \infty$.

By the definition of q and W, $1 = \sum_x p(x) = \sum_x cW(x)q(x) = c\mathbb{E}_q[W]$. Thus, $\mathbb{E}_q[W] = c^{-1}$. Since $\mathbb{E}_q[\varphi W] = c^{-1}\mathbb{E}_q[\varphi W c] = c^{-1}\mathbb{E}_p[\varphi]$.

Thus, by the Continuous Mapping Theorem [Mann and Wald, 1943], we have

$$\hat{\mu}^{(n)}(\varphi) = \frac{S_n/n}{B_n/n} \xrightarrow{\text{a.s.}} \frac{c^{-1}\mathbb{E}_p[\varphi]}{c^{-1}} = \mathbb{E}_p[\varphi]$$

as $n \to \infty$. □

Applying this Lemma to the one-step particle update of NBF lets us show that if the current estimate θ is consistent, then θ is also consistent in the limit.

Theorem B.2 (NBF Consistency)**.** *Assume* ϵ*-global observation positivity of* (G, π) *and a finite* X *and* Y. *For any finite horizon* t_{max}, *belief state* $p_t(x), t \leq t_{max}$, *and any bounded function* $\varphi : X \to \mathbb{R}$, *let*

$$\hat{\mu}_t^{(n)}(\varphi) = \frac{\sum_{i=1}^n w_i \varphi(x_i)}{\sum_{i=1}^n w_i}$$

be the estimate of $\mathbb{E}_{p_t}[\varphi]$ *computed by NBF with a perfect embedding model and* n *particles. Then,*

$$\sup_{0 \leq t \leq t_{max}} |\hat{\mu}_t^{(n)}(\varphi) - \mathbb{E}_{p_t}[\varphi]| \xrightarrow{a.s.} 0$$

as $n \to \infty$.

Proof. According to Algorithm 1, the weight for transitioning from x_i to x_i' is $w_i = T_G(x_i, \pi)[x_i'] \cdot H_G(x_i, x_i')[y]$. We start by proving almost sure convergence for any fixed $t \leq t_{\text{max}} : t, t_{\text{max}} \in \mathbb{N}$ by induction.

Base case. $t = 0$ Here $p(x) = p_0$, so a perfect embedding model implies we can compute θ_p^* by embedding samples from p_0 and then generate $x_{1:n} \overset{\text{iid}}{\sim} p(x)$. φ is bounded, so we can apply the Strong Law of Large Numbers to get the result.

Inductive step. Assume for some $t < t_{\text{max}}$, generated particles $x_{1:n}$ are i.i.d. according to $p_t(x)$.

After the loop in Algorithm 1, we have $x'_{1:n}$ distributed according to the proposal $q(x) = \sum_{x'} p_t(x')T_G(x', \pi)[x]$ with weight function $W = H_G(x', x)[y]$. Note that the exact posterior at time $t+1$ can be written as

$$p_{t+1}(x) = \frac{\sum_{x'} p_t(x')T_G(x', \pi)[x] \cdot H_G(x', x)[y]}{c} = q(x)W(x)/c \tag{1}$$

for some normalization constant c.

ϵ-global observation positivity implies that for all $x \in X$ both $p_{t+1}(x) > 0 \implies q(x) > 0$ and $W(x) > 0$ (see Algorithm 1, Line 5). Since T_G and H_G output probability mass functions, $W(x) \leq 1$ for all $x \in X$. As a result, we can apply Lemma B.1 and get that $\hat{\mu}_{t+1}^{(n)}(\varphi) \xrightarrow{\text{a.s.}} \mathbb{E}_{p_{t+1}}[f]$ as $n \to \infty$. This implies that embedding $\theta^*_{p_{t+1}} = \mathcal{E}_\phi(x'_{1:n}, w_{1:n})$ and regenerating $x''_{1:n} \overset{\text{iid}}{\sim} p_{\theta^*_{t+1}}(x) = p_{t+1}(x)$ using a perfect model gives the desired result.

Almost sure convergence of the sequence. Now that we have shown almost sure convergence for any $t \leq t_{\max}$, we can complete the proof of the theorem. For any $\delta > 0$, there exists a random integer $N_t = \min\{n : \forall m \geq n, |\hat{\mu}_t^{(n)}(\varphi) - \mathbb{E}_{p_t}[\varphi]| \leq \delta\}$. Let $N = \sup_{0 \leq t \leq t_{\max}}\{N_t : t \leq t_{\max}\}$, then for all $n \geq N$ we have $|\hat{\mu}_t^{(n)}(\varphi) - \mathbb{E}_{p_t}[\varphi]| \leq \delta$ for every $t \leq t_{\max}$.

□

Corollary B.3. *Under the same conditions as Theorem 4.1, as* $n \to \infty$,

$$\sup_{0 \leq t \leq t_{max}} |\hat{\mu}_t^{(n)}(\varphi) - \mathbb{E}_{p_t}[\varphi]| = O_p(n^{-1/2})$$

Proof. From Equation 1, let $\tilde{p}_t(x) \stackrel{\text{def}}{=} cp_t(x) = W(x)q(x)$ and denote $w_i \stackrel{\text{def}}{=} W(x_i) = \frac{\tilde{p}_t(x_i)}{q(x_i)}$ for $1 \leq i \leq n$. Let $Z_i \stackrel{\text{def}}{=} w_i(\varphi(x) - \mathbb{E}_p[\varphi])$. Then,

$$\begin{aligned}
\mathbb{E}_q[Z_i] &= \sum_x q(x) w_i(\varphi(x) - \mathbb{E}_p[\varphi]) \\
&= \sum_x \tilde{p}_t(\varphi(x) - \mathbb{E}_p[\varphi]) \\
&= c\mathbb{E}_p[\varphi] - c\mathbb{E}_p[\varphi] = 0
\end{aligned}$$

So Z_i have mean zero and are independent.

Now take the denominator of the estimate $B_n = \sum_{i=1}^n w_i$. By ϵ-global observation positivity, $w_i \geq \epsilon$, so $B_n \geq n\epsilon \implies B_n^{-2} \leq (\epsilon^2 n^2)^{-1}$.

Since,

$$\hat{\mu}_t^{(n)}(\varphi) - \mathbb{E}_{p_t}[\varphi] = \frac{\sum_{i=1}^n Z_i}{B_n},$$

it follows that, by independence and zero mean of Z_i,

$$\mathbb{E}[(\hat{\mu}_t^{(n)}(\varphi) - \mathbb{E}_{p_t}[\varphi])^2] = \mathbb{E}[B_n^{-2}(\sum_{i=1}^n Z_i)^2] \leq \frac{1}{\epsilon^2 n^2} \sum_{i=1}^n \text{Var}[Z_i].$$

φ is bounded, so $\text{Var}[Z_i] \leq \text{Var}_{p_t}[\varphi] \leq ||\varphi||_\infty^2 < \infty$. Define $\sigma_t^2 \stackrel{\text{def}}{=} \text{Var}_{p_t}[\varphi]$, so $\sum_{i=1}^n \text{Var}[Z_i] \leq n\sigma^2$. Plugging this into the previous bound and taking the supremum over t gives

$$\sup_{0 \leq t \leq t_{\max}} \mathbb{E}[(\hat{\mu}_t^{(n)}(\varphi) - \mathbb{E}_{p_t}[\varphi])^2] \leq \sup_{0 \leq t \leq t_{\max}} \frac{\sigma_t^2}{\epsilon^2 n} = \frac{\sigma^2}{\epsilon^2 n}$$

for $\sigma^2 \stackrel{\text{def}}{=} \sup_{0 \leq t \leq t_{\max}} \sigma_t^2$.

Applying Chebyshev's inequality gives, for any $\delta > 0$

$$\Pr[|\hat{\mu}_t^{(n)}(\varphi) - \mathbb{E}_{p_t}[\varphi]| \geq \delta] \leq \frac{\sigma_t^2}{\epsilon^2 n \delta^2}$$

Therefore,

$$\Pr[\sup_{0 \leq t \leq t_{\max}} |\hat{\mu}_t^{(n)}(\varphi) - \mathbb{E}_{p_t}[\varphi]| \geq \delta] \leq \frac{(t_{\max} + 1)\sigma^2}{\epsilon^2 n \delta^2}$$

Setting $\delta = M/\sqrt{n}$ for $M > 0$ gives:

$$\Pr[\sqrt{n} \sup_{0 \leq t \leq t_{\max}} |\hat{\mu}_t^{(n)}(\varphi) - \mathbb{E}_{p_t}[\varphi]| \geq M] \leq \frac{(t_{\max} + 1)\sigma^2}{\epsilon^2 M^2}$$

which lets us apply the definition of stochastic boundedness [Van der Vaart, 2000] to finish the proof. □

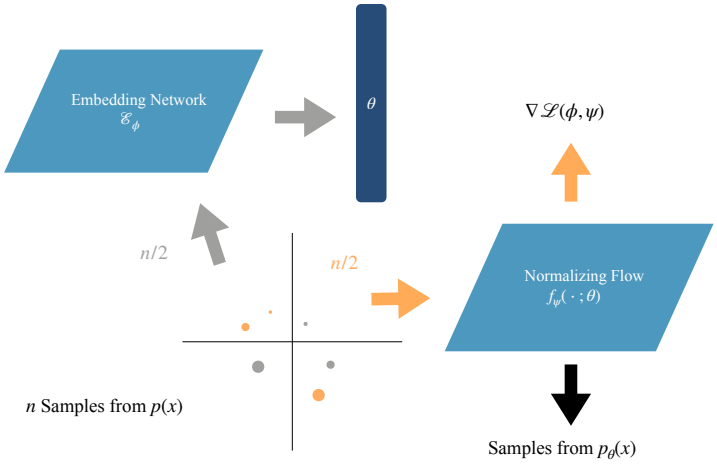


Figure 7: Training belief embedding models. Colored arrows show the flow of training sample points from the target distribution.

C Additional Experimental Details

Figure C outlines the training process for the belief models used to embed distributions in all of our experiments. Belief embedding models are trained by generating n samples from a belief state instance $p(x)$ from the set of target distributions ($\mathcal{P}_G^\Pi$ in Goofspiel and Gridworld). Half of the samples are used to compute the embedding θ that conditions the generative model $f_\psi(z; \theta)$. The rest of the samples are used to approximate the gradient of $\mathcal{L}(\phi, \psi)$.

All experiments were implemented in Jax using standard libraries from the Jax ecosystem. We used custom implementations for Normalizing Flows and both environments. All source code is available as part of the supplementary material. Next, we provide domain-specific details about the models used in our experiments.

C.1 Gridworld

Parameters for the set of grids used for training and evaluation are shown in Table 2.

Belief Embedding Model. The embedding function $\mathcal{E}_\phi$ consisted of a standard MLP with 3 layers of 128 units each and ReLU activations. The generative model $f_\psi(z; \theta)$ used a uniform prior and 5 coupling layers [Dinh *et al.*, 2016] with masked inputs. Variational dequantization [Ho *et al.*, 2019] was performed to smooth discrete grid locations. More hyperparameters are shown in Table 1.

Recurrent Model. The recurrent baseline learns a mapping from a sequence of observations about movement in the grid to an approximation of $p(x)$. These observations are the same as those used to define the posteriors in the filtering tasks. Since belief states in Gridworlds of these sizes are small, the model outputs a softmax distribution over potential grid locations. JS divergence between model output and belief state instances was minimized directly. More hyperparameters are shown in Table 1.

Computational Resources. For each random seed, model training required roughly 2 CPU-hours on commodity consumer hardware. Each evaluation (consisting of 500 episodes) took at most 3 CPU-hours. Since every experiment was repeated for 100 seeds, the end-to-end compute requirements were roughly 500 CPU-hours for each of the 8 grid configurations. We used cluster resources provided by a source that will be revealed upon publication.

C.2 Goofspiel

Goofspiel Policy Generation. We generated a sequence of policies by independent self-play using PPO to simulate the effect of changing policies during learning, as in classical self-play settings. We

Table 1: Gridworld model and training hyperparameters.

	Belief Embedding Model	**Recurrent Baseline**
Embedding size	32	–
Embedding network hidden units	128	–
Embedding network hidden layers	3	–
Dequantization hidden units	32	–
Dequantization hidden layers	2	–
Normalizing Flow / RNN hidden units	32	32
Normalizing Flow / RNN hidden layers	5	2
Normalizing Flow coupling layers	5	–
Batch size	32	32
Training steps	100 000	100 000
Training samples (per $p(x)$)	64	–
Optimizer	AdaGrad	AdaGrad
Learning rate	0.10	0.10

Table 2: GridWorld environment parameters.

Parameter	**5 × 5**	**8 × 8**
Obstacle cubes	1	2
Cube width	2	3
Softmax temp. (for random policies)	1×10^{-5}	1×10^{-5}

used the Jax version of StableBaselines 3 (SBX) [Raffin *et al.* 2021]—modified to support action masking. In self-play, we trained a policy against its previous checkpoint for 524 288 timesteps and saved a checkpoint every 4096 timesteps. We repeated this self-play loop four times, producing a sequence of 512 policies in total.

Belief Embedding Model. The belief embedding model for Goofspiel uses a Standard Normal prior, and consists of a variational dequantization layer parameterized by a single coupling layer, followed by a series of coupling layers. The dequantization coupling layer uses an affine transformation, and each of the following layers use one-dimensional non-linear squared (NLSq) transformations. All coupling layers transform masked inputs. After dequantization, each coupling layer is further parameterized by θ. Concrete hyperparameters used in our experiments are listed in Table 3.

Recurrent Model. Observations in Goofspiel consist of features such as the player's hand, the prize deck, the current one-hot encoded prize card, and the winnings. The recurrent baseline learns a mapping from a sequence of these observations to an embedding. This embedding conditions a Normalizing Flow with the same architecture as described above. The key difference is that the recurrent model maps the observation sequence to an embedding directly, whereas the belief embedding model embeds sample sets from $p(x)$.

Computational Resources. Goofspiel experiments were run on computing resources provided by a source that will be revealed upon publication. For each game size and each random seed, belief model training required approximately 12 hours and recurrent model training approximately 3 hours on 32 CPU cores. Afterwards, each model was evaluated for 500 episodes, which took between several minutes and three hours on 12 CPU cores, depending on the size of the game. The evaluation was repeated 10 times.

C.3 Donuts

For our illustrative example from Section 3, we used a scaled-down version of the belief embedding model used in the main experiments. Model and training hyperparameters are shown in Table 4. Donuts models train in several minutes on a laptop.

Table 3: Goofspiel model and training hyperparameters.

	Belief Embedding Model	**Recurrent Baseline**
Embedding size	48	48
Embedding network LSTM hidden units	–	64
Embedding network LSTM hidden layers	–	2
Embedding network MLP hidden units	128	64
Embedding network MLP hidden layers	3	2
Dequantization hidden units	48	48
Dequantization hidden layers	2	2
Normalizing Flow MLP hidden units	128	64
Normalizing Flow MLP hidden layers	4	4
Normalizing Flow coupling layers	8	8
Batch size	64	64
Training steps	150 000	16 000
Training samples (per $p(x)$)	64	32
Optimizer	Nadam	Nadam
Learning rate	0.001	0.001

Table 4: Donuts model and training hyperparameters.

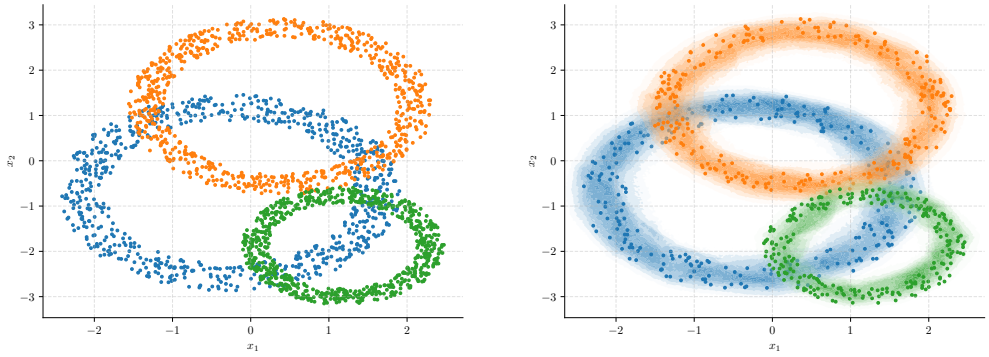
	Normalizing Flow Model	**Cond. FM Model**
Embedding size	8	8
Embedding network hidden units	64	64
Embedding network hidden layers	3	3
Dequantization hidden units	–	–
Dequantization hidden layers	–	–
Normalizing Flow MLP hidden units	32	64
Normalizing Flow MLP hidden layers	3	4
Normalizing Flow coupling layers	8	–
Batch size	32	32
Training steps	30 000	30 000
Training samples (per $p(x)$)	128	128
Optimizer	Adam	Adam
Learning rate	0.001	0.001

D Illustrative Example using Conditional Flow Matching

To highlight the versatility of our proposed approach, we applied a Conditional Flow Matching (CFM) model [Lipman *et al.*, 2022, Tong *et al.*, 2023] to the donut-shaped distributions introduced as a toy domain described in Section 3. We experimented with both independent coupling and optimal transport (OT) coupling [Lipman *et al.*, 2022] to define the target vector fields that generate the conditional probability paths in our flow matching models. An example of a CFM model trained with optimal transport coupling predicting the density for a randomly sampled set of donuts is shown in Figure 8(b). The model was conditioned using 256 samples sampled from the true distribution and asked to predict the density of each point in a 512×512 grid. Model and training hyperparameters are shown in Table 4.

E Code

We include all of our training and evaluation code as a ZIP archive. It will also be publicly available on GitHub upon publication.



(a) Sample donut distributions. Each distribution has three parameters: mean, radius, and width.

(b) Learned densities after conditioning the model on 256 samples from the target distribution.

Figure 8: Embedding the set of donut distributions in $\mathbb{R}^2$ using Conditional Flow Matching