A Additional Details on Point-Based Value Iteration

Point based value iteration (PBVI) is an algorithm for efficiently solving POMDPs (Pineau et al., 2003). See Shani et al. (2013) for a thorough survey of related algorithms and extensions in this area.

A.1 Background

As first observed in Sondik (1978), the value function for a POMDP can be approximated arbitrarily closely as the upper envelope of a finite set of linear functions of the belief, commonly referred to as α-vectors. Representing the value function as a collection of linear functions, we can write the value of an arbitrary belief $b \in \Delta^K$ in the probability simplex as:

$$V(b) = \max_{\alpha} \; b \cdot \alpha. \tag{7}$$

Each α-vector is associated with a corresponding optimal action a_α, so the value function can be represented as a set of pairs $\{(\alpha, a_\alpha)\}_{\alpha \in V}$. To act according to this value function representation, the action a_{α^*} associated with α^*, the maximizing α-vector, is taken given the current belief b at each time point. The task of solving a POMDP is then to compute this set of α-vectors. Unfortunately, exactly representing the true value function (via e.g. exact value iteration) requires exponentially many α-vectors, and this becomes computationally intractable for even small problems. Early techniques for efficiently solving POMDPs often involved iteratively pruning redundant α-vectors at each iteration of the solver, but these approaches also did not scale well. See Shani et al. (2013) for more details.

A.2 PBVI Overview

In PBVI, unlike in exact value iteration, we do not perform full Bellman backups over the space of all possible belief points, as this is typically intractable. Instead, we will only perform backups at a fixed set of belief points, which we denote by $\mathcal{B} \triangleq \{b_i\}_{i=1}^B$, with $B = |\mathcal{B}|$ and $b_i \in \Delta^K$. We will return later to how this set is chosen.

We first highlight the computation for the value at a belief b after a Bellman backup over V, where we let r_a denote the vector $R(\cdot, a)$:

$$V'(b) = \max_{a \in A} r_a \cdot b + \gamma \sum_o p(o|b,a) V(b^{a,o}), \tag{8}$$

where

$$\alpha^{a,o}(s) = \sum_{s'} \alpha(s') p(o|s',a) p(s'|s,a), \tag{9}$$

and $b^{a,o}(s') = p(s'|b,a,o)$ denotes our new belief that we are in state s' having started from belief vector b, taken a, and seen o.

See Shani et al. (2013) for the full derivation, but some algebra eventually reduces this expression to:

$$V'(b) = \max_{a \in A} r_a \cdot b + \gamma \sum_o \max_{\alpha \in V} b \cdot \alpha^{a,o}. \tag{10}$$

The two maxes in this equation implicitly prunes dominated α-vectors twice, which is more efficient than previous approaches that would first enumerate the (massive) space of all α-vectors and then prune afterwards.

We can use the value function computation in Eq.10 to efficiently compute the new α-vector that would have been optimal for b, had we ran the complete Bellman backup:

$$backup(V, b) = \operatorname{argmax}_{\alpha_a^b : a \in A, \alpha \in V} b \cdot \alpha_a^b \tag{11}$$

$$\alpha_a^b = r_a + \gamma \sum_o \operatorname{argmax}_{\alpha^{a,o} : \alpha \in V} b \cdot \alpha^{a,o} \tag{12}$$

During the backup, the action associated with the new α-vector is also cached. Importantly, these point-based updates are substantially more efficient than an exact update, as they are quadratic rather than exponential. In addition, for problems with finite horizons, the error between the PBVI approximate value function and the true value function decreases to 0 as we more densely sample the belief simplex and we take $B \to \infty$.

Choose the Belief Points. We now briefly discuss how the set $\mathcal{B}$ is chosen. There are many different implementation choices that can be made; see Shani et al. (2013) for a comprehensive list of previous works of approaches made in different algorithms. A naive approach is to randomly the simplex or choose beliefs evenly spaced on a grid, but both are usually inefficient and may include many beliefs that, in practice, would rarely be reached by actual trajectories.

We use the strategy used in the original PBVI paper (Pineau et al., 2003). Start with an initial set of beliefs $\mathcal{B}_0$. In our work, we initialize this to be the uniform belief vector $\frac{1}{K}\vec{1}$ along with beliefs that place a large amount of mass (e.g. 99%) on a single state. In the end, our initial set $\mathcal{B}_0$ contains $K+1$ belief points. To add a new belief point b to an existing set $\mathcal{B}$, we find a successor belief b' that is most distant from our current set. We do this by using a distance metric (in practice, we use standard L_2 Euclidean distance), and let

$$|b' - \mathcal{B}| = \min_{b \in \mathcal{B}} |b - b'| \tag{13}$$

be the distance from a new belief b' to the set $\mathcal{B}$. We focus on new candidate beliefs that can be reached from the current set. For discrete observations, we can enumerate all possible $b^{a,o}$ that are reachable given a starting belief b, if we take each possible action a and then see observation o. For continuous observations, we of course cannot enumerate all possible $b^{a,o}$ but can instead just draw samples from our observation model.

We can then add to $\mathcal{B}$ an additional new belief b' that is farthest from $\mathcal{B}$. Or, we can add a set of new beliefs that are all "far" from the current set $\mathcal{B}$, e.g. greater than some pre-specified distance ϵ. The high-level idea for this belief set expansion is for the set $\mathcal{B}$ to be spread out relatively evenly across the *reachable* parts of belief space.

Implementation Details. In practice, we may interleave belief expansion steps where we increase the size of $\mathcal{B}$ with a large number of backups, repeatedly running Eq 11 for all current beliefs in $\mathcal{B}$.

Having the entire PBVI algorithm as a subroutine in the larger optimization pipeline for POPCORN is a challenging task. One rather expensive and inefficient design choice would be to entirely rerun PBVI, *from scratch*, at each iteration of gradient descent. Instead, we cache the intermediate value function and the current set of belief points during optimization. During one gradient update, we then choose to only run a small number of PBVI backups (in practice, between 1-5), where we run backups of our current beliefs given the new model parameters θ at this iteration of training.

As we are learning both the policy and the model online during training for POPCORN, we empirically found that it is helpful to occasionally do a hard reset of both $\mathcal{B}$ and our value function. In the planning community (e.g. the original PBVI paper), it is typically assumed that the ground truth model θ is known, whereas in real-world settings, the model must be learned from data. This means that during training, our estimate of θ constantly changes, and over time our value function and belief set may have been largely determined from very stale previous values of θ. In practice, we do these hard resets very infrequently, e.g. only once every 250 or 500 gradient updates.

A.3 PBVI: Sampling Approximation to Deal with Complex Observation Models

In normal PBVI, we are limited by how complex our observation space is. The PBVI backup crucially depends on a summation over observation space (or integration, for continuous observations). Dealing with multi-dimensional, non-discrete observations is generally intractable to compute exactly.

Instead, we will utilize ideas from Hoey and Poupart (2005) to circumvent this issue. The main idea is to learn a partition of observation space, where we group together various observations that, conditional on a given belief b and taking an action a, would have the same maximizing α-vector. That is, we want to learn $\mathcal{O}_\alpha = \{o | v = \text{argmax}_{\alpha \in V} \alpha \cdot b^{a,o}\}$. We can then treat this collection of $\mathcal{O}_\alpha$ as a set of "meta-observations", which will allow us to replace the intractable sum/integral over observation space into a sum over the number of α-vectors, by swapping out the $p(o|b,a)$ term in Equation 8 with $p(\mathcal{O}_\alpha|b,a)$, the (approximate) aggregate probability mass over all observations in the "meta-observation". In particular, we can express the value of a belief by:

$$V(b) = \max_a r_a \cdot b + \gamma \sum_\alpha p(\mathcal{O}_\alpha|b,a) V(b^{a,\mathcal{O}_\alpha}) \tag{14}$$

$$p(\mathcal{O}_\alpha|b,a) = \sum_s b(s) \sum_{s'} p(s'|a,s) p(\mathcal{O}_\alpha|s',a) \tag{15}$$

$$b^{a,\mathcal{O}_\alpha} \propto p(\mathcal{O}_\alpha|a,s') \sum_s b(s) p(s'|s,a) \tag{16}$$

$$p(\mathcal{O}_\alpha|a,s') = \sum_{o \in \mathcal{O}_\alpha} p(o|a,s'). \tag{17}$$

We will make use of a sampling approximation that admits arbitrary observation functions in order to approximate the $\mathcal{O}_\alpha$ and $p(\mathcal{O}_\alpha|s',a)$, the aggregate probability of each "meta-observation".

To do this, first we sample k observations $o_k \sim p(o|s',a)$, for each pair of states and actions. Then, we can approximate $p(\mathcal{O}_\alpha|a,s')$ by the fraction of sampled o_k where α was the optimal α-vector, ie

$$p(\mathcal{O}_\alpha|a,s') \approx \frac{|\{o_k : \alpha = \text{argmax}_{\alpha \in V} \alpha \cdot b^{a,o_k}\}|}{k}, \tag{18}$$

where ties are broken by e.g. favoring the α-vector with lowest index. Using this approximate discrete observation function, we can perform point-based backups for V at a set of beliefs $\mathcal{B}$ as before. Our backup operation is now:

$$backup(V,b) = \text{argmax}_{\alpha_a^b : a \in A, \alpha \in V} b \cdot \alpha_a^b \tag{19}$$

$$\alpha_a^b = r_a + \gamma \sum_{\alpha'} \underset{\alpha^{a,\mathcal{O}_{\alpha'}}}{\text{argmax}}\ b \cdot \alpha^{a,\mathcal{O}_{\alpha'}} \tag{20}$$

$$\alpha^{a,\mathcal{O}_{\alpha'}}(s) = \sum_{s'} \alpha(s') p(s'|s,a) p(\mathcal{O}_{\alpha'}|a,s'). \tag{21}$$

The previous sum/integral over observations has now been replaced by a sum over α-vectors, which is generally more tractable.

A.4 Softmax Relaxation to Make PBVI Differentiable

In order to be able to differentiate through the entire PBVI backups and allow gradient-based optimization for POPCORN, we relax the original argmax operations involved in PBVI backups and running a PBVI policy to softmaxes. There are 2 argmax operations in the original PBVI backups, in Eqs. 11 and 12. For PBVI with continuous observations, there is an additional argmax associated with the probability of "meta-observations" in Eq. 18. Lastly, there is a fourth argmax associated with actually running a policy, as we need to determine

which α-vector is the maximizing one, and we take its corresponding action.

We relax all 4 of these argmaxes to softmaxes. Previously, there were operations such as $\text{argmax}_i\ x_i$ to select a maximal index; we can view these as returning a delta function at the maximizing index, or a probability mass function with all mass on one element. We instead relax this to a softmax, now returning $p \triangleq \frac{e^{x_i}}{\sum_i e^{x_i}}$, a distribution over all elements. Where before we might have taken, e.g., α_j if j was the maximizing index of x, now we instead take a *soft* mean using the softmax probability distribution p, i.e. we instead would take $A \cdot p$ where $A \in \mathbb{R}^{K \times N}$ is a matrix with all N vectors $\alpha \in \mathbb{R}^K$ stacked up, and p is a probability vector over the N choices.

Last, we further modified these softmaxes by using an additional temperature parameter τ, which lets us control how close to deterministic the softmax is. That is, we redefine the softmax as $p \triangleq \frac{e^{x_i/\tau}}{\sum_i e^{x_i/\tau}}$. As $\tau \to 0$, the softmax p approaches the deterministic argmax, while $\tau \to \infty$ approaches a uniform distribution. In experiments, throughout we used a fixed $\tau = 0.01$ for all environments. In initial tests on the tiger environment, we tried starting with larger temperatures and slowly annealing them to smaller values, but found this only added noise and slowed overall convergence. For relatively small temperatures, we confirmed that the softmax-relaxed PBVI solutions were comparable to the original deterministic ones.

Note that in this relaxation, each α-vector is now associated with a *distribution* over actions, rather than a single action as before. Likewise, as we now learn stochastic policies, to run a soft-PBVI policy, we take a soft mean of the softmax distribution over actions associated with each α-vector; contrast this with the deterministic solution where we'd simply choose the action associated with the maximizing α-vector.

It is also worth noting that in simulated environments where we can actually run a policy, we can always run a deterministic version of a softmax policy by simply selecting the most likely action, rather than probabilistically choosing an action from a policy's distribution over actions. Our main motivation for using softmax policies is that it makes OPE easier, as otherwise for deterministic policies we may run into severe problems if the support of our deterministic policy has little in common with the behavior policy.

B Additional Details on Learning Rewards

We noted in the main text that learning rewards is explicitly not part of the main optimization procedure in POPCORN. This is because we expressly do *not* want to compute gradients for the estimated policy value term with respect to the reward function parameters. If we did so, there would be nothing stopping the optimization procedure to "hallucinate" that the best way to learn a high-value policy is to simply increase all values of the reward function to be large. The policy induced by such a model with incorrectly high rewards would then appear to be very good, with respect to the model, but when run in the real world or real environment, it would perform terribly.

Instead, we simply learn the rewards a separate EM step, that may be performed alongside each gradient update to τ, μ, σ (in practice this is what we do), or may be done only periodically. In the E-step, we compute the relevant summary statistics from the forward and backward pass through the IO-HMM. Importantly, the E-step does *not* depend on the reward function R at all; the forward and backward pass only use the transition and emission distributions to update relevant state probabilities. Then, in the M-step, we update *only* the reward function, using the observed reward values from the trajectories in our dataset. This is equivalent to minimizing the sum of squared errors between our reward function and the observed reward values. It is also nearly identical to what the M-step looks like for μ, but the other observations are assumed to be Gaussian and hence also have the variance σ parameters.

C Additional Setup Details and Results for Tiger Domains

We show a few extra results from the synthetic tiger domains, and provide a bit more detail for the setup for the third environment involving misspecification in the emission distribution itself.

First, in Figure 8 we show results from the first Tiger with Irrelevant Noise environment, where we vary the total dimensionality of observation space from 1 (model is properly specified) to 16, with the results in the main paper only showing 2. Throughout, 2-stage always learns models with the highest likelihood but fails completely at the downstream decision-making task. The difference in likelihood between POPCORN and value-only becomes more muted for larger numbers of dimensions, as the models are more and more misspecified, and the likelihood metric collapses over all dimensions. The differences would remain more apparent if we instead showed the associated likelihood metric for each individual dimension of observation space.

Next, in Figure 10 we show results for the Tiger with Missing Data environment as a function of the fraction of missingness in the relevant dimension needed to make

decisions. We vary this amount in the following range: $\{10\%, 30\%, 50\%, 70\%, 80\%, 90\%, 95\%\}$. As the overall amount of missingness increases, likelihood values and policy values generally degrade, which is to be expected as less and less total information is contained in a single dataset. Notably, 2-stage always learns much better models but performs terribly in its policy, while the converse is true for value-only.

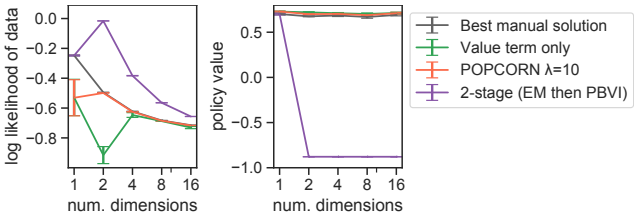

Figure 8: Results from the Tiger with Irrelevant Noise environment, where we now vary the overall observation dimension size. Throughout, the first dimension is the only relevant dimension with any information about the decision-making task ($\sigma = 0.3$ for this dimension) while the rest all contain irrelevant observations with lower $\sigma = 0.1$.

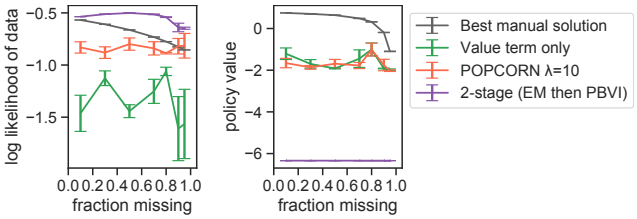

Figure 9: Additional results from the Tiger with Missing Data. These results show how performance varies as a function of the amount of missing data. In the main text, we only show results from the case where 80% of the relevant observation variable is missing.

Last, we provide qualitative probing of the models learned in the last tiger environment, Tiger with Wrong Likelihood. The true emission distribution is given by a truncated Gaussian Mixture Model (GMM) with equal weights, and the truncation depends on the true unknown latent state value. However, we have setup the emissions by choosing an appropriate prior distribution over the latent states so that marginally, the observations look like they come from a normal GMM and so a pure likelihood-based approach would try to fit a GMM rather than the true truncated GMM. See Figure 10 for the models from the manual solution, 2-stage, and POPCORN. The manual solution used oracle knowledge of the true underlying states which other methods did not have access to, and simply moment matched by taking empirical means and variances. Since the results of value-only and POPCORN were near identical for this environment, we do not show it. For this environment, POPCORN does even better than the manual solution. While 2-stage learns a slightly high likelihood model, its policy is substantially worse.

In the figure, the histograms show observed observations in green and blue, and model densities in purple and red. The green and blue histograms show observation values colored by their true state. Green bars correspond to state "1", so that observations are drawn from the GMM but truncated to be positive. Likewise, blue bars correspond to state "0", and observations from this state are drawn from the GMM and then truncated to be negative. The numbers in each subplot denote the learned mean and variance parameter for the 2 states for the emission model for each method (conditioned on the last action being listen). Note that the ground truth means of the truncated GMM were 0 and 1, and the ground truth standard deviations were 0.1 and 1. 2-stage correctly recovers these, but it is tricked into learning a GMM, rather than the true underlying *truncated* GMM. Histograms of the 2 emission distributions learned by each approach are shown in red and purple. 2-stage learns the true parameters of the GMM, whereas POPCORN learns state emissions that are more spread out so that it can correctly differentiate between the two true underlying states. The true underlying states can be perfectly identified by whether or not they are positive or negative; which of the two GMM mixture components they came from does not always identify them correctly.

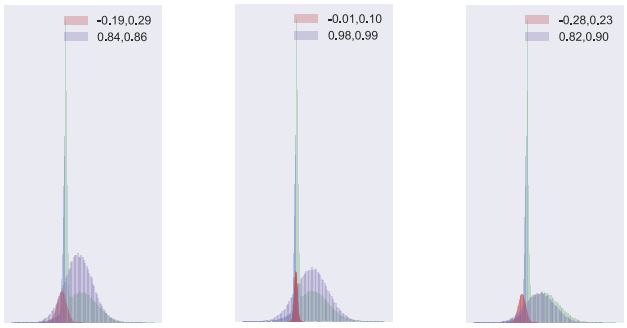

Figure 10: Qualitative results from the Tiger with Wrong Likelihood environment. *Left*: Manual solution, log marginal likelihood -0.95, policy value 0.20. *Middle*: 2-stage EM solution, log marginal likelihood -0.76, policy value -0.57. *Right*: POPCORN with $\lambda = 1$, log marginal likelihood -0.92, policy value 0.50. See text for details.

D Additional Setup Details for Sepsis Domain

The original sepsis environment in Oberst and Sontag (2019) consists of $D = 5$ discrete observation dimensions. Four are vitals and naturally ordinal (e.g. "low", "high"),

while the last is binary.

We encode each ordinal discrete observation with C categories as an integer in $\{0, \ldots, C-1\}$, then add independent Gaussian noise to this integer, with $\sigma = 0.3$. Adding noise to the 4 ordinal-valued vitals is reasonable and can be viewed as approximating measurement error, in some sense, if we pretend that the original discrete variables were obtained by dichotomizing some "true" unknown continuous value. This is not strictly true in practice, as the environment is hard-coded and not actually based on some sort of underlying dynamical system. However, adding noise to the diabetes indicator is just a convenient way to make it continuous-valued.

As noted in the main text, we use this environment simply as a slightly more challenging medically-inspired simulator. This differs substantially from its original use in Oberst and Sontag (2019) where they used it to introduce strong hidden confounding with known structure by masking 2 state variables from their methods.

In our work we used $K = 5$ somewhat arbitrarily. The main purpose of this environment was to illustrate the tradeoff POPCORN makes between likelihood and policy value, and not to try to actually solve the environment or learn a policy that is near-optimal. Given the partially observed nature of our alteration to the environment, and the high noise level with our ϵ-greedy behavior environment, it's not immediately obvious what the best achievable policy that can be learned in the batch setting is. This will be less than the value of the true optimal policy for the original MDP, which is what we showed in the results figure in the main text.

E Additional Setup Details and Results from MIMIC ICU Hypotension Domain

E.1 Data Preprocessing

We did very little filtering to the initial raw dataset. We started with only patients who had data from the MIMIC-III MetaVision database, as this more recent data has better metadata around treatment timing. We only filtered by the first ICU stay for hospital stays that had multiple ICU admits, and then filtered to ICU stays with 3 or more MAP measurements less than 65mmHg. To discretize time, we started 1 hour into ICU admit and used time points at hour 2, etc. up until hour 72, so that at most our trajectories contain 71 actions. We leave to future work to come up with improved, potentially data-driven methods for more realistic time discretization.

Since an IO-HMM generative model can easily handle missing data, we do not impute missing values for time points when a measurement is missing. In the event that multiple measurements were taken in the span of a single hour, we take the most recent value. This is extremely uncommon for lab values, and only really applied to vitals such as MAP or heart rate. Even then, in MIMIC-III most of the time vitals are logged only once an hour.

Before modeling the physiological values, we log transformed them (after adding 1 to avoid numeric issues). After log transforms, the population distributions for each variable looked reasonably close to normally distributed. This step was necessary as many clinical values have a heavy right tail, and would be inappropriately to model with Gaussians. After the log-transform, we further standardized each variable to have zero mean and unit variance.

E.2 Action Space Construction

IV fluids somewhat naturally cluster into discrete bins, so this action variable was easier to discretize. We used 4 bins by amount: $\{0, [200, 500), [500, 1000), [1000, 2000]\}$. Figure 11 shows the distribution of raw fluid amounts in the data.

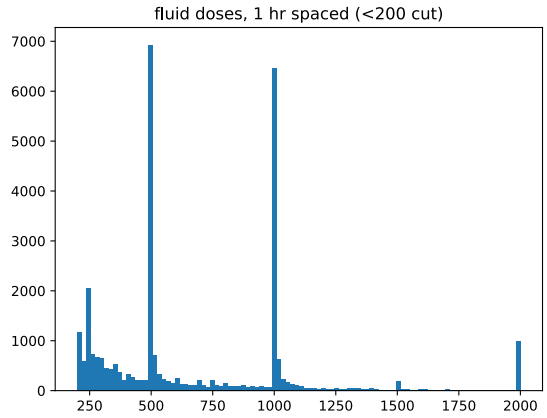

Figure 11: Distribution of raw IV Fluid doses in the original dataset prior to discretization, in mL.

We normalized across vasopressor drug types following the logic in Komorowski et al. (2018) in order to arrive at equivalent rates across drugs. Then we took the total amount of vasopressor administered within each hour-long decision window given our time discretization. We eventually then grouped into 5 bins by total drug amount given each hour: $\{0, (0, 5), [5, 15), [15, 40), [40, 150]\}$ with units of mcg/kg/hr. Figure 12 shows the distribution of raw fluid amounts in the data.

Lastly, Figure 13 shows the frequency by time point for

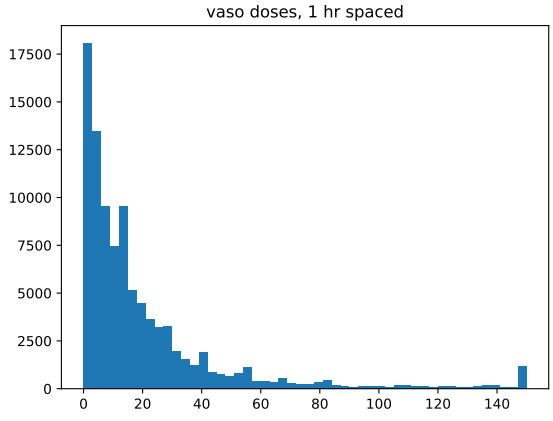

Figure 12: Distribution of raw vasopressor amounts administered per hour in the original dataset prior to discretization, in (normalized) mcg/kg/hr.

how often each of the 20 different types of actions was administered. Roughly 85% of all time points had no treatment administered.

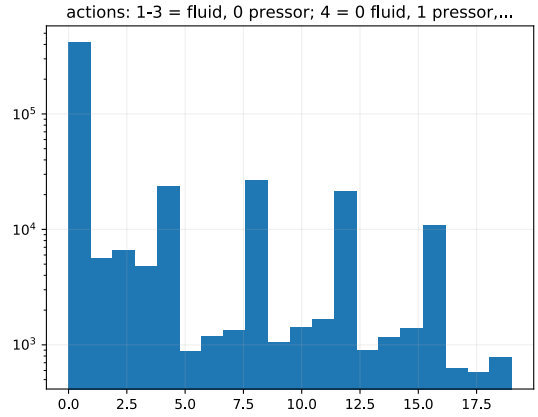

Figure 13: Overall frequency of each action type in our dataset.

E.3 Reward Function Construction

We show reward function plots for the two reward functions used in this paper. Figure 14 shows the MAP-based reward used in most of the work. Note the inflection points at 55 and 60 mmHg values. Also, patients who had adequate urine output had a lower threshold for MAP values that start to yield worse rewards, as clinically a modest urine output means the clinician is less worried about the precise MAP value unless it becomes very low.

Figure 15 shows the reward used for the reward re-

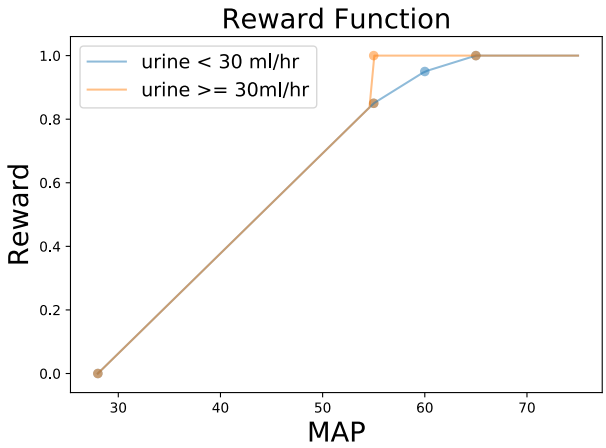

Figure 14: The true reward function used in hypotension experiments. The algorithm is rewarded for keeping the the Mean Arterial blood Pressure (MAP) 65mmHg, a common target value in critical care.

specification experiment in the main paper, when we tested to what extent a model learned to yield good policies with respect to the MAP reward might generalize to this new reward. Clinically, higher lactate values indicate possible organ damage and are a sign of worsening illness.

E.4 Learning the Behavior Policy

We use the approach of Raghu et al. (2018) to learn our behavior policy, and use a k-nearest-neighbors approach. Their work found that the calibration of the behavior policy is crucial for accurate OPE and that more complex models such as neural networks were often poorly calibrated. In consultation with our intensivist collaborator, we hand-constructed a distance function based on our observed variables, and used this to do kNN.

We used a simple weighted Euclidean distance between observations, with a weight of 3 on creatinine, 2 on FiO_2, 3 on heart rate, 4 on lactate, 1 on platelets, 5 on urine output, 2.5 on total bilirubin, 5 on MAP, and 5 on GCS. Although not actually used as features in our models, we also considered 4 additional binary features that indicate the discrete vasopressor amount (if any) given at the last time point, and 3 binary features that indicate the discrete fluid amount administered at the last time point. All of these extra features received a weight of 5. We lastly added features that added the total raw amount of vasopressor and fluid given thus far in a trajectory, as well as in the past 8 hours; these 4 features also had a weight of 5. Concretely, we used $d(o, o') = \sum w_i (o_i - o'_i)^2$, with the weights w_i listed previously and i indexing observation variables.

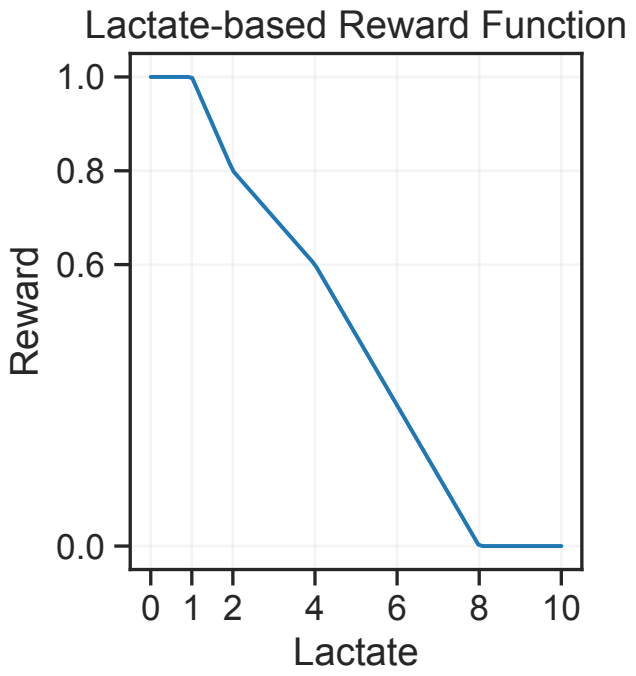

Figure 15: Reward function based on lactate used for the reward re-specification experiment in the main text.

For performing kNN, we learned a behavior policy based on an observation's 100 nearest neighbors using our hand-crafted distance metric, and simply count up the actions performed by those neighbors to use as our estimate of behavior action probabilities. In rare cases where none of the 100 nearest neighbors correctly predicted the true next action taken, we reset the behavior policy to assign 3% probability to the actual action that was taken.

We learn a different estimate of the behavior policy for each of the 5 folds of cross validation, using the same splits that were used by each of the later methods considered.

E.5 Additional Qualitative Results

Figures 16, 17, and 18 show additional qualitative results about the learned models for POPCORN, 2-stage EM, and value-only, for the heart rate, lactate, and urine output variables. As in Figure 5 in the main text, we again see that the 2-stage approach largely learns states that exhibit very high overlap. Likewise, the value-only baseline learns states that are much more spread apart, and even occasionally learn bizarre distributions that are close to a point mass at one value. As expected, POPCORN learns models in between these two extremes, with diverse enough states to learn a good policy while also fitting the data decently well.

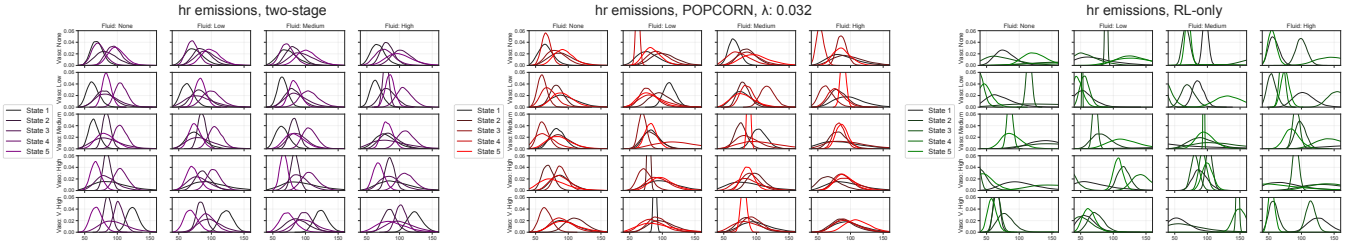

Figure 16: Visualization of learned heart rate distributions. *Left:* 2-stage EM. *Middle:* POPCORN, $\lambda = 0.032$. *Right:* Value-only. Each subplot visualizes all 100 learned distributions of heart rate values for a given method across the 20 actions and $K = 5$ states. Each pane in a subplot corresponds to a different action, and shows distributions across the 5 states. Vasopressor actions vary across rows, and IV fluid actions vary across columns.

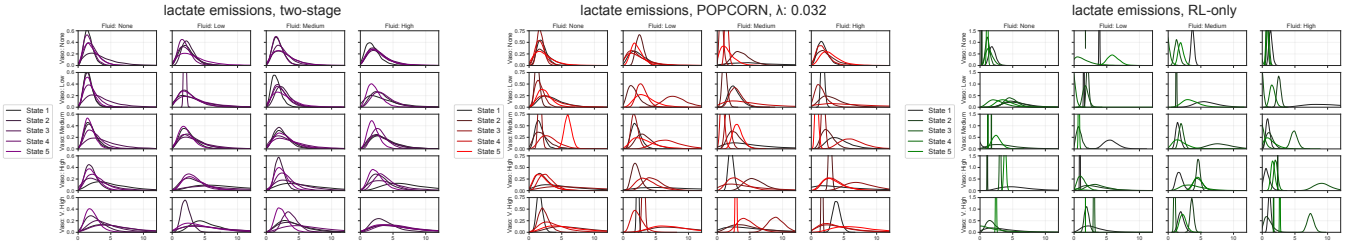

Figure 17: Visualization of learned lactate distributions. *Left:* 2-stage EM. *Middle:* POPCORN, $\lambda = 0.032$. *Right:* Value-only. Each subplot visualizes all 100 learned distributions of lactate values for a given method across the 20 actions and $K = 5$ states. Each pane in a subplot corresponds to a different action, and shows distributions across the 5 states. Vasopressor actions vary across rows, and IV fluid actions vary across columns.

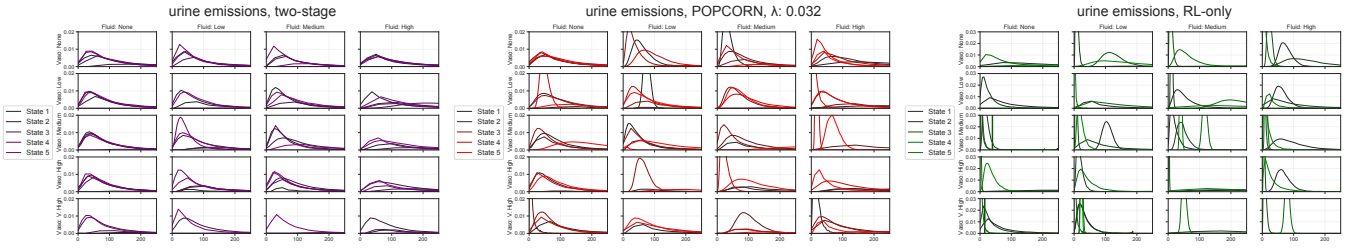

Figure 18: Visualization of learned urine output distributions. *Left:* 2-stage EM. *Middle:* POPCORN, $\lambda = 0.032$. *Right:* Value-only. Each subplot visualizes all 100 learned distributions of urine output values for a given method across the 20 actions and $K = 5$ states. Each pane in a subplot corresponds to a different action, and shows distributions across the 5 states. Vasopressor actions vary across rows, and IV fluid actions vary across columns.