BELIEF EMBEDDING TREE SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Beliefs in a Partially Observable Markov Decision Process (POMDP) are posterior distributions over the hidden state given the agent's history of actions and observations. They summarize everything the history says about the state, so planning in a POMDP is planning in a Markov decision process (MDP) over beliefs. Belief MDPs have continuous state spaces even for finite POMDPs, and exact planning in them is intractable in general. Planners must therefore approximate beliefs and estimate values. Monte Carlo planners do both inside a search tree, with an approximate belief at every node and a value estimate at every leaf. Approximations usually consist of sets of sampled states, with leaf values coming from rollouts, hand-designed bounds, or learned networks. Policies and values computed from these approximations are limited by approximation quality, and prior methods with learned networks rely on hand-chosen summaries of the sampled states to compute them. A belief representation should be computable from the parent node's representation using the POMDP's dynamics and it should let the search sample from it, but it should also be learned so that it provides the information needed to condition policies and values functions. We present Belief Embedding Tree Search (BETS), a Monte Carlo planner that meets these requirements by representing each node's belief with a generative distribution embedding (GDE) while updating it through the POMDP dynamics. BETS trains its belief model alongside its policy and value networks through experience, with the realized hidden state of the POMDP as the belief model's training target. We evaluate BETS on six configurations of RockSample, LightDark, and PocMan against BetaZero and three planners without learned components. BETS is competitive or leading on all six, while every baseline falls far behind on at least one. Where beliefs are highest-dimensional, it outperforms them by a large margin, even without search.

1 INTRODUCTION

Agents planning in a Partially Observable Markov Decision Process (POMDP) must act on a *belief*, the posterior distribution over hidden states given their history of actions and observations. The action an agent takes should depend on the belief rather than on any single state, so policies map beliefs to actions and value functions map beliefs to expected returns. Computing them exactly is intractable in general, even when the state, action, and observation spaces are finite (Papadimitriou & Tsitsiklis, 1987). Monte Carlo planners instead search forward from the agent's current history by simulating the POMDP's dynamics and observation model (Silver & Veness, 2010; Somani et al., 2013). These simulations draw states from a representation of the belief at every node of the search tree and stop at leaves that need value estimates.

Monte Carlo planners typically represent each node's belief with a set of sampled states and differ in how they maintain that set and how they estimate leaf values. At each node POMCP and POMCPOW keep the states that simulations carried into it and estimate leaf values with rollouts of a default policy (Silver & Veness, 2010; Sunberg & Kochenderfer, 2018). DESPOT and AdaOPS search a fixed set of sampled scenarios and estimate leaf values with upper and lower bounds (Somani et al., 2013; Wu et al., 2021). The accuracy of these leaf values depends on the rollout policies and bounds, which are either designed for each domain or loose. Maintaining a set of sampled states also carries the weaknesses of particle filters. The number of states needed for an accurate approximation grows rapidly with the size of the state space (Thrun, 2002; Snyder et al., 2008), and repeated resampling also collapses the set onto a few distinct states, a failure called *impoverishment* (Sokota et al., 2022).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Policy and value networks trained on experience generated by the planner can replace rollouts and bounds (Silver et al., 2018). The networks of BetaZero (Moss et al., 2024) condition on a hand-chosen summary of the particle set, the mean and standard deviation of its features. This summary discards much of what distinguishes one belief from another, such as its support and the dependence between its variables. BetaZero also keeps a particle set at every node of its tree, so it carries the same weaknesses of particle filters. A learned belief representation that does not collapse under repeated updates would address both, provided updating it and sampling from it remain tractable.

Solinas et al. (2025) proposed Neural Bayesian Filtering (NBF), which represents beliefs as generative distribution embeddings (GDEs) (Fishman et al., 2025) and approximates their Bayesian update after actions and observations with a forward pass through the GDE's generator and encoder. At each step, NBF generates particles from the previous belief's embedding, propagates and reweights them under the POMDP's dynamics and likelihood, and embeds the weighted set as the next belief. Every update therefore draws a fresh particle set, so impoverishment from resampling does not accumulate across steps. NBF's conditional generator is trained to sample from the belief the embedding represents, so the search can generate the states it needs. If the embedding is accurate it is also a sufficient statistic of the history (Fishman et al., 2025), so policies and value functions can condition on it in place of the belief.

We introduce Belief Embedding Tree Search (BETS), a Monte Carlo planner that searches over a tree of embeddings. Every node stores one embedding, expanding a leaf is one NBF update, and the policy and value networks that guide the search condition on the embedding, the same representation of the belief that the generator samples from. Hidden states in many discrete POMDPs are configurations of coupled variables, so BETS adds a *variable-coupling* stream to the GDE that pools a joint feature of every pair of variables over the particle set. The belief model and both networks train together on episodes collected by running BETS, and the belief model's target is the hidden state the environment realizes at each step (Solinas et al., 2025).

We evaluate BETS against BetaZero, POMCPOW, DESPOT, and AdaOPS on six configurations of RockSample, LightDark, and PocMan (Smith & Simmons, 2004; Platt et al., 2010; Silver & Veness, 2010). Each baseline falls far below the best return in at least one setting, while BETS is competitive or leading on all six. On RockSample(20,20) and PocMan, the configurations with the highest-dimensional beliefs, BETS outperforms every baseline by a large margin, even when it acts with its policy network alone. On PocMan, value networks predict the return more accurately from embeddings than from summary statistics of a particle filter with 16× as many particles. BETS also achieves a higher return with this embedding than with a pooled embedding twice as large.

2 Background

BETS plans in POMDPs whose dynamics and observation model are given as input to the problem. A **POMDP** is a tuple $G = (X, A, Y, T, \mathcal{Y}, R, \gamma, b_0)$ of a set of states X, a finite set of actions A, a set of observations Y, a transition function $T : X \times A \to \Delta X$, an observation function $\mathcal{Y} : X \times A \times X \to \Delta Y$, a reward function $R : X \times A \times X \to \mathbb{R}$, a discount factor γ, and an initial state distribution $b_0 \in \Delta X$. An episode starts from $x^{(0)} \sim b_0$, and at each step t the agent takes an action $a^{(t)}$, the environment moves to $x^{(t+1)} \sim T(\cdot \mid x^{(t)}, a^{(t)})$, and the agent receives the observation $y^{(t+1)} \sim \mathcal{Y}(\cdot \mid x^{(t)}, a^{(t)}, x^{(t+1)})$ and the reward $R(x^{(t)}, a^{(t)}, x^{(t+1)})$. The agent observes its own actions, so what it knows at time t is the **history** $h_t \stackrel{\text{def}}{=} (a^{(0)}, y^{(1)}, \ldots, a^{(t-1)}, y^{(t)})$, and it acts according to a **policy** $\pi : H \to \Delta A$, a mapping from the set H of histories to distributions over actions. The objective is a policy that maximizes the expected discounted return $\mathbb{E}_{G,\pi}[\sum_{t=0}^{t_{\max}-1} \gamma^t R(x^{(t)}, a^{(t)}, x^{(t+1)})]$ over episodes of $t_{\max}$ steps, and the **action value** $Q^\pi(h_t, a) \stackrel{\text{def}}{=} \mathbb{E}_{G,\pi}[\sum_{t'=t}^{t_{\max}-1} \gamma^{t'-t} R(x^{(t')}, a^{(t')}, x^{(t'+1)}) \mid h_t, a^{(t)} = a]$ is the expected return from taking a after h_t and following π thereafter.

The initial state is random, transitions can be stochastic, and distinct states can emit the same observation, so histories do not identify states; they induce a distribution over X. This distribution is the **belief** b_t, which depends on h_t. Given $a^{(t)}$ and $y^{(t+1)}$, Bayes' rule gives the posterior belief,

$$b_{t+1}(x') \propto \sum_{x \in X} b_t(x)\, T(x' \mid x, a^{(t)})\, \mathcal{Y}(y^{(t+1)} \mid x, a^{(t)}, x'). \tag{1}$$

The update, with an integral in place of the sum when X is continuous, is exact but intractable when the state space is large or continuous. The belief is a sufficient statistic of the history (Kaelbling et al., 1998), so an optimal policy and its action values depend on h_t only through b_t. Planning in a POMDP is therefore planning in a **belief MDP**, whose states are beliefs, whose transitions apply Eq. 1 to the next observation, and whose values are expected returns under b_t. Its state space is continuous even when X is finite, and the planners of Section 2.2 search with approximations of b_t.

2.1 Approximating Beliefs

Particle filters (Gordon et al., 1993; Doucet & Johansen, 2009) approximate b_t by a set of weighted states, or *particles*, $\{x_i, w_i\}_{i=1}^n$. Given $a^{(t)}$ and $y^{(t+1)}$, the update simulates a successor $x_i' \sim T(\cdot \mid x_i, a^{(t)})$ for every particle and multiplies its weight by the likelihood $\mathcal{Y}(y^{(t+1)} \mid x_i, a^{(t)}, x_i')$. When many weights become small, the filter *resamples* by duplicating high-weight particles and discarding the rest. Resampling never introduces a state the set did not already contain, so repeated resampling concentrates the set on a few distinct states, a failure called **impoverishment**. The approximation of b_t then collapses, and the value estimates a planner computes from it inherit the error (Poupart & Boutilier, 2000). Regularization (Musso et al., 2001), MCMC (Gilks & Berzuini, 2001), and locally optimal proposals (Doucet & Johansen, 2009) slow the collapse, but every new particle still descends from the surviving set, so loss compounds over the episode. Preventing weights degeneration takes exponentially many particles in the dimension of the state (Snyder et al., 2008).

Neural Bayesian Filtering (NBF) (Solinas et al., 2025) approximates the Bayes update of Eq. 1 in the latent space of a **generative distribution embedding** (GDE) (Fishman et al., 2025). A GDE consists of an **encoder** $\mathcal{E}_\phi : \bigcup_{n\geq 1} X^n \times \mathbb{R}^n \to \mathbb{R}^{d_{\text{emb}}}$ that maps a weighted set of states to a vector θ, and a conditional **generator** $\mathcal{G}_\psi(\theta)$ that maps the vector back to sample states. The encoder is *distributionally invariant* (Fishman et al., 2025): it depends on a weighted set only through the set's weighted empirical measure, so reordering the set or duplicating every sample leaves its output unchanged. The NBF **update** $\theta' \leftarrow \mathcal{U}_{\phi,\psi}(\theta, a, y)$ generates n particles from $\mathcal{G}_\psi(\theta)$, propagates each through T, weights it by $\mathcal{Y}$, and embeds the weighted set with $\mathcal{E}_\phi$. Each update regenerates its particle set from the embedding, so the collapse that resampling causes in a particle filter does not accumulate across steps. If the particle set loses most or all of the belief's support in a single update, the approximation can collapse irrecoverably. In practice, this happens only in extreme cases, such as a sharp likelihood on a belief far from those BETS visits during training.

2.2 Planning with Partial Observability

Full-width planners expand every branch with exact belief updates (Ross et al., 2008), which limits them to small state and observation spaces. Monte Carlo planners (Kocsis & Szepesvári, 2006; Browne et al., 2012) scale better by simulating states drawn from an approximate belief through the environment dynamics. POMCP (Silver & Veness, 2010) runs UCT over histories with a particle set at each node, and POMCPOW (Sunberg & Kochenderfer, 2018) adds progressive widening for large observation spaces. DESPOT (Somani et al., 2013) and AdaOPS (Wu et al., 2021) search a sparse tree over a fixed set of sampled *scenarios* and prune it with upper and lower bounds on the value. These planners have no learned components, so their search is only as informed as their rollout policies and bounds, but they need no training phase. BetaZero (Moss et al., 2024) approximates each node's belief with a particle filter, and uses learned policy and value networks to guide search. These networks condition on the mean and standard deviation of features of the particle set and are trained on targets from BetaZero's own searches (Silver et al., 2018).

Summary statistics of particle features can discard the structure a decision depends on. Consider two cells, each of which may hold a hidden object, so the state is $x \in \{0,1\}^2$ with $x_j = 1$ when cell j is occupied. The belief b_{excl} is uniform over $\{01, 10\}$, where exactly one cell is occupied, and b_{co} is uniform over $\{00, 11\}$, where both or neither are. Both beliefs give each cell probability $\frac{1}{2}$ of holding the object, so their per-cell means and standard deviations are identical, yet their supports are disjoint. An action rewarded only when both cells are occupied has zero expected reward under b_{excl} and positive expected reward under b_{co}. A network that conditions on per-variable statistics must assign the two beliefs the same policy and value. A generator conditioned on per-variable statistics fails in the same way, since it must produce one distribution for both beliefs and so places at least half of its mass on configurations that one of them rules out. A planner that simulates those

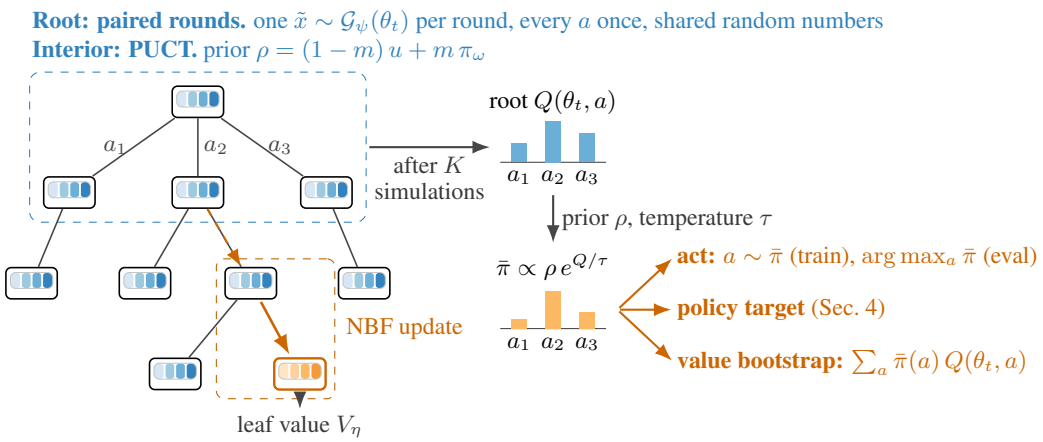

Figure 1: **BETS.** Each round starts every root action from the same sampled state and interior actions are selected via PUCT. Expansion costs one NBF update.

configurations estimates values on states the belief rules out. The next section builds a planner whose networks and generator condition on an embedding that represents this dependence explicitly.

3 Belief Embedding Tree Search

Belief Embedding Tree Search (BETS) is a decision-time planner for POMDPs that searches over a tree of embeddings (Figure 1). At each step, BETS roots the tree at the root's embedding and returns a policy over actions that determines the agent's next action. Every node in the tree stores one embedding θ, and expanding a leaf is exactly one NBF update. A policy network $\pi_\omega : \mathbb{R}^{d_{\text{emb}}} \times \mathbb{R}^{d_\xi} \to \Delta A$ and a value network $V_\eta : \mathbb{R}^{d_{\text{emb}}} \times \mathbb{R}^{d_\xi} \to \mathbb{R}$ guide the search. Both networks condition on the embedding and a small vector ξ of features of the observed part of the state, such as the fraction of the horizon elapsed (Appendix A). Conditioning on θ rather than on per-variable statistics lets the networks assign different policies and values to beliefs that share those statistics.

3.1 Search

At decision time, BETS follows the standard Monte Carlo tree search procedure of selection, expansion, evaluation, and backup (Kocsis & Szepesvári, 2006; Browne et al., 2012). It runs K simulations of this procedure through a tree rooted at the belief embedding θ_t.

Root. BETS groups the K simulations into $K/|A|$ rounds, and each round selects every root action exactly once. At the start of a round, BETS samples a state $\tilde{x} \sim \mathcal{G}_\psi(\theta_t)$, and every action in the round starts from $\tilde{x}$ with the same random numbers, so the actions' returns are *paired* (Veness et al., 2011). The decision depends only on the differences between root action values, and pairing reduces the variance that the sampled state adds to those differences. When UCT (Kocsis & Szepesvári, 2006) or PUCT (Rosin, 2011; Silver et al., 2017) selects the root action, each action is evaluated on its own sampled states, and that state sampling variance can influence the relative value estimates of actions for finite K. Selecting every action once per round leaves root visit counts uniform, so the decision and the policy target are computed from the root action values instead (Section 3.2).

Interior. Below the root, the traversal applies the selected action to $\tilde{x}$, samples the next state and observation from T and $\mathcal{Y}$, and descends to the child node for that action and observation if one exists. At an interior node with embedding θ, the traversal selects the action maximizing PUCT,

$$\arg\max_{a \in A} \; \bar{Q}(\theta, a) + c\,\rho(a \mid \theta, \xi)\,\frac{\sqrt{N(\theta)}}{1 + N(\theta, a)},$$

where $N(\theta)$ and $N(\theta, a)$ are the node and edge visit counts, $\bar{Q}$ is the action value estimate normalized to $[-1, 1]$ within the node (Schrittwieser et al., 2020), and c is the exploration constant. The

prior ρ mixes the policy network with the uniform distribution u over legal actions,

$$\rho(a \mid \theta, \xi) \stackrel{\text{def}}{=} (1 - m)\, u(a) + m\, \pi_\omega(a \mid \theta, \xi),$$

so no action's prior reaches zero and BETS keeps exploring regardless of policy network bias.

Expansion, evaluation, and backup. When the current node has no child for the action a and observation y of the traversal's last step, BETS expands by adding a new leaf. If the current node lies d levels below the root with embedding θ_{t+d}, the leaf's embedding is computed with an NBF update, $\theta_{t+d+1} \leftarrow \mathcal{U}_{\phi,\psi}(\theta_{t+d}, a, y)$. The value network evaluates the leaf, $V_\eta(\theta_{t+d+1}, \xi_{t+d+1})$, with no rollout. The value backed up through the traversal is the discounted sum of the rewards generated below it, plus the leaf value. The visit counts along the path are then incremented. Each simulation applies the NBF update, the policy network, and the value network once, and every node stores one embedding in place of a particle set.

3.2 Action Selection

After K simulations, the root node holds a paired estimate $Q(\theta_t, a)$ for every action. BETS turns these estimates into a distribution over actions with one step of KL-regularized policy improvement (Abdolmaleki et al., 2018),

$$\bar{\pi} \stackrel{\text{def}}{=} \arg\max_{\pi' \in \Delta A} \sum_a \pi'(a)\, Q(\theta_t, a) - \tau\, \mathrm{KL}(\pi' \,\|\, \rho(\cdot \mid \theta_t, \xi_t))\,,$$

where the temperature τ sets how far the search can move the policy away from the prior. The maximizer has the closed form $\bar{\pi}(a) \propto \rho(a \mid \theta_t, \xi_t) \exp(Q(\theta_t, a)/\tau)$, which is one mirror descent step from the prior with step size $1/\tau$ (Beck & Teboulle, 2003). AlphaZero's visit-count distribution approximates the solution of this problem with the KL taken in the other direction (Grill et al., 2020), and Danihelka et al. (2022) use this closed form. BETS evaluates this closed form exactly.

During training, the agent acts by sampling $a \sim \bar{\pi}$; at evaluation, it takes $\arg\max_a \bar{\pi}(a)$. The same $\bar{\pi}$ is the policy network's training target, and its expected root value $\sum_a \bar{\pi}(a)\, Q(\theta_t, a)$ is the bootstrap for the value network's target (Section 4.2). After BETS acts, one more NBF update computes the next root embedding from the action taken and the observation received.

3.3 Variable-Coupling Embeddings

Standard GDE encoders compute a feature vector for each sample and pool the per-sample features over the set (Fishman et al., 2025). Each per-sample feature depends on every variable of its sample, so in principle the pooled features can capture how the variables of a sample depend on one another. The two beliefs in the example of Section 2.2 give each cell the same probability of holding the object, so the dependence between the two cells is all that separates them. In practice, the encoder has to discover the dependence inside generic per-sample features, and if the dependence does not survive the pooling, the two beliefs receive the same embedding. We extend GDEs by giving the encoder and the generator an explicit representation of the co-occurrence between pairs of variables.

A discrete state is a vector $x \in \{0, \ldots, q-1\}^D$ of D variables. The *pooled stream* is a set transformer (Lee et al., 2019) whose per-sample features are averaged over the set to give the pooled embedding μ. A second stream, which we call the **variable-coupling stream**, averages a joint feature of each pair of variables over the same samples. From these pairwise averages it forms a coupling summary for each variable that describes how the variable relates to the others (Figure 2). The D coupling summaries are combined with μ to form θ. The policy and value networks take all of θ as input, so they condition on every coupling summary as well as on μ. Both streams average over samples, so θ remains distributionally invariant, and both are trained jointly with the generator.

BETS pairs the variable-coupling stream with an autoregressive generator that treats each variable as a token and factorizes the belief over them, $p_\psi(x \mid \theta) = \prod_{j=1}^{D} p_\psi(x_j \mid x_{<j}, \theta)$, where $p_\psi(\cdot \mid \theta)$ is the density of $\mathcal{G}_\psi(\theta)$. The generator conditions factor j on the preceding variables $x_{<j}$, the shared pooled embedding μ, and the coupling summary of variable j (Appendix C.1). Each factor is a normalized categorical, so the product is an exact joint likelihood computable in one forward pass.

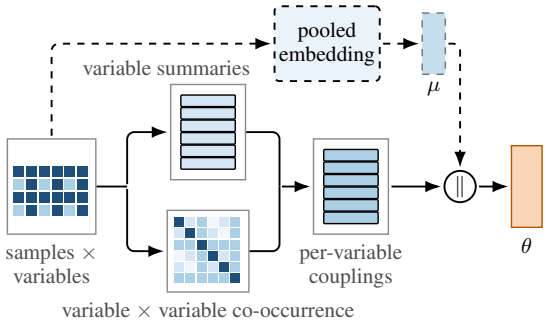

Figure 2: **The variable-coupling stream.** Pooling over samples gives one summary per variable and a variable-by-variable co-occurrence matrix. Attention over the variable summaries, biased by the co-occurrence matrix, produces coupling summaries, which are concatenated with μ to form θ.

4 Training

BETS learns three models: the belief model, which is the GDE encoder and generator that NBF uses in its update, and the policy and value networks that guide the search. The beliefs BETS visits depend on its policy, so the beliefs the model must represent change as the networks train and the search policy improves. All three models train together on episodes that BETS collects.

Training alternates between gradient steps and collection rounds. In each *collection round*, a set of collectors runs BETS with the parameters from the most recent gradient step, and each collector takes a fixed number of environment steps. The consecutive transitions that one collector records in one round form a *fragment*, and an episode that ends inside a fragment is reset, so the fragment continues with the next episode. Before the first gradient step, a fixed number of collection rounds run with the initial model parameters. Collection rounds repeat at a fixed interval of gradient steps, and every gradient step draws one batch for each of the three losses from the fragments of the most recent rounds. The policy and value networks are trained on the embeddings θ_t stored during collection, so their losses never reach the encoder and the GDE is trained on the belief loss alone. The exploration constant c, the mix weight m, and the temperature τ are annealed from exploratory to greedy values over the first part of training (Appendix B).

4.1 Belief Model

Each training example for the GDE is a tuple $(\theta_t, a^{(t)}, y^{(t+1)}, x^{(t+1)})$ stored from a collected episode. For each example, the NBF update generates particles from $\mathcal{G}_\psi(\theta_t)$, propagates them under $a^{(t)}$, and weights them by the likelihood of $y^{(t+1)}$, which yields the weighted set S_{t+1} and its embedding $\theta_{t+1} = \mathcal{E}_\phi(S_{t+1})$. The update runs at training time, so the particle sets the belief model embeds are regenerated from the generator being trained rather than loaded from storage. At the start of an episode, the particles are drawn from b_0 instead.

The target is the hidden state $x^{(t+1)}$ that the environment realized after $a^{(t)}$. Though the agent never observes $x^{(t+1)}$, it is the ground truth state for simulation, so it is available to compute the loss at training time. The belief model minimizes the realized-state objective of Solinas et al. (2025),

$$\mathcal{L}(\phi, \psi) \stackrel{\text{def}}{=} \mathbb{E}\big[\ell_\psi\big(x^{(t+1)};\, \mathcal{E}_\phi(S_{t+1})\big)\big], \tag{2}$$

where the expectation is over the collected transitions and over the particle sets the NBF update generates from them. $\ell_\psi(x; \theta) = -\log p_\psi(x \mid \theta)$ is the GDE's negative log-likelihood of the single target x given the embedding $\theta = \mathcal{E}_\phi(S_{t+1})$.[1] Gradients pass through the embedding of the set and through the generator, but not through the sampling that produced the set.

BETS's actions depend on the hidden state only through the history, so each $x^{(t+1)}$ is a sample from the true posterior b_{t+1} given the history h_{t+1}. The particle set S_{t+1} is computed from that history and the filter's own random draws, and the **set posterior** $p(\cdot \mid S_{t+1})$ is the distribution of $x^{(t+1)}$

[1] On LightDark, ℓ_ψ is instead the conditional flow-matching regression at $x^{(t+1)}$ (Appendix C.2).

given the set alone. It equals b_{t+1} when the set retains all the information that the history carries about the state, and in general it is the belief with the lowest expected log loss among all beliefs computed from the set. By the decomposition of Solinas et al. (2025, Proposition 4.1), minimizing $\mathcal{L}(\phi, \psi)$ for fixed sets is equivalent to minimizing $\mathbb{E}\big[\mathrm{KL}\big(p(\cdot \mid S_{t+1}) \,\|\, p_\psi(\cdot \mid \theta_{t+1})\big)\big]$, the expected KL divergence from the set posterior to the GDE's belief $p_\psi(\cdot \mid \theta_{t+1})$.

4.2 Policy and Value Targets

The policy and value networks are trained on targets derived from the search (Anthony et al., 2017; Silver et al., 2017). At step t, the root gives the improved policy $\bar{\pi}_t$ and its value $\bar{v}_t \stackrel{\text{def}}{=} \sum_a \bar{\pi}_t(a)\, Q(\theta_t, a)$, and the transition $(\theta_t, \xi_t, a^{(t)}, r_t, \bar{\pi}_t, \bar{v}_t)$ is stored, where r_t is the reward of step t. The **policy target** is $\bar{\pi}_t$, and the policy network minimizes the cross-entropy $\mathrm{H}(\bar{\pi}_t, \pi_\omega(\cdot \mid \theta_t, \xi_t))$.

The **value target** is the λ-return bootstrapped from the root values $\bar{v}$ (Sutton & Barto, 1998),

$$G_t^\lambda \stackrel{\text{def}}{=} r_t + \gamma\big[(1-\lambda)\,\bar{v}_{t+1} + \lambda\, G_{t+1}^\lambda\big],$$

computed backward over each fragment, with $G_{t+1}^\lambda \stackrel{\text{def}}{=} \bar{v}_{t+1}$ past the end of a fragment and $\bar{v}_{t+1} = G_{t+1}^\lambda = 0$ at terminal states. The value network is trained as a classifier over a fixed grid of bins. Each λ-return is squashed by $f(u) = \mathrm{sign}(u)\big(\sqrt{|u|+1} - 1\big) + \epsilon u$ (Pohlen et al., 2018), and $f(G_t^\lambda)$ is encoded as a two-hot distribution over the bins (Schrittwieser et al., 2020). Training minimizes the cross-entropy between the value network's prediction and this two-hot target. During search, a leaf's value is f^{-1} applied to the mean of the predicted distribution over the bins.

5 Experiments

We evaluate BETS on six configurations of three POMDP domains: RockSample(15,15) and RockSample(20,20) (Smith & Simmons, 2004), LightDark(5) and LightDark(10) (Platt et al., 2010), and PocMan on a medium maze (M) and the large maze (L) of Silver & Veness (2010) (Appendix A). The domains differ in belief structure. In RockSample, the agent's position is known, and the belief is a product of 15 or 20 independent Bernoulli distributions, one per rock, over 2^{15} or 2^{20} joint configurations. In LightDark, the agent moves along a line, and the belief is a one-dimensional density over its position that stays broad in the dark and sharpens near the light. In PocMan, it is a joint distribution over the positions and headings of three or four ghosts, which move through the same maze, chasing or running away from the agent when it is nearby.

Baselines. We compare against **BetaZero** (Moss et al., 2024) and three planners without learned components: **AdaOPS** (Wu et al., 2021), **DESPOT** (Somani et al., 2013), and **POMCPOW** (Sunberg & Kochenderfer, 2018). All four use the same number of simulations and particles as BETS on PocMan, and the settings of Moss et al. (2024) on RockSample and LightDark (Appendix B).

Protocol. BETS is evaluated in two ways on the same model: with **search**, acting by $\arg\max_a \bar{\pi}(a)$ with the search settings training ends with, and with the **policy** network alone, acting by $\arg\max_a \pi_\omega(a \mid \theta, \xi)$. We tuned the training hyperparameters of BETS on every domain and those of BetaZero on PocMan, where Moss et al. (2024) give no settings. BETS and BetaZero are trained at three seeds, and each trained model is evaluated on 300 episodes at each of three evaluation seeds. AdaOPS, DESPOT, and POMCPOW have no learned components, so each runs three evaluation seeds with 300 episodes each.

5.1 Results

Table 1 shows that BETS earns the highest return on RockSample(20,20) and on both PocMan mazes, where baseline search heuristics or belief approximations fail. It trails AdaOPS on RockSample(15,15), where AdaOPS's heuristic bounds are feasible, and BetaZero on LightDark(10). On RockSample(20,20) computing the heuristic upper bounds of AdaOPS and DESPOT is infeasible, so both run on a fixed upper bound, which reduces discounted return.

Table 1: Mean discounted return with the SE across training seeds (BETS, BetaZero) or evaluation seeds (AdaOPS, DESPOT, POMCPOW). AdaOPS has no bounds implementation for PocMan.

	BETS					
Domain	search	policy	BetaZero	AdaOPS	DESPOT	POMCPOW
RockSample(15,15)	20.78 ± 0.09	20.65 ± 0.10	19.79 ± 0.18	$\mathbf{21.45 \pm 0.10}$	19.45 ± 0.10	11.45 ± 0.08
RockSample(20,20)	$\mathbf{14.97 \pm 0.13}$	13.91 ± 0.16	11.30 ± 0.15	$10.77 \pm 0.20^{\dagger}$	$0.28 \pm 0.02^{\dagger}$	$4.33 \pm 0.18^{\dagger}$
LightDark(5)	3.61 ± 0.10	3.77 ± 0.12	3.76 ± 0.15	$3.08 \pm 0.03^{\dagger}$	2.29 ± 0.09	2.82 ± 0.25
LightDark(10)	13.01 ± 0.41	13.88 ± 0.28	$\mathbf{14.43 \pm 0.33}$	$3.18 \pm 0.40^{\dagger}$	0.33 ± 0.05	0.98 ± 0.13
PocMan(M)	$\mathbf{78.08 \pm 0.54}$	72.42 ± 2.05	40.87 ± 3.26	–	54.20 ± 0.34	36.34 ± 1.93
PocMan(L)	$\mathbf{78.95 \pm 0.23}$	77.47 ± 0.46	69.87 ± 1.64	–	60.49 ± 0.71	39.91 ± 1.41

[†] Without domain heuristics: exit-policy lower and fixed upper bounds for AdaOPS and DESPOT on RockSample(20,20), fixed bounds for AdaOPS on LightDark, and no rollout heuristic for POMCPOW.

BetaZero also learns to guide its search, so we attribute most of the difference from BetaZero to what its networks condition on: the mean and standard deviation of each state variable over its particle set. Either the particle set or its summary statistics can make a poor input to the policy and value networks. On RockSample(20,20) the belief is a product of Bernoulli distributions, so the per-rock means are a sufficient statistic for it. We attribute the gap there to BetaZero's particle set, which covers a small fraction of the 2^{20} configurations. Each resampling step of the bootstrap filter duplicates the surviving particles, and with deterministic transitions the duplicates stay identical, so the per-rock means rest on ever fewer distinct configurations that carry disproportionate weight, whereas BETS regenerates its particles from the embedding at every update. A resampler that redraws each rock from its per-rock mean would restore distinct configurations, but it relies on the factorization, which is a domain-specific property.

On PocMan we attribute the gap to the summary statistics, which lose two kinds of structure. A ghost that may be in any of several corridors has a multimodal belief, and the mean and standard deviation of its position cannot distinguish those corridors from the region between them. These statistics also discard the dependence between state variables, such as ghost positions and headings, which is what the variable-coupling stream was designed to model.

BetaZero matches BETS on LightDark(5) and leads on LightDark(10). With a one-dimensional state and Gaussian observation noise, every particle keeps a nonzero weight and the effective sample size does not collapse, so the particle set approximates the belief well and its mean and standard deviation are close to a sufficient statistic for it. The policy network alone matches search on LightDark(5) and exceeds it on LightDark(10). We hypothesize that, because every sampled observation is new, the tree stays shallow and a single search compares actions mostly through the value network's estimates at a few sampled leaves, whose noise the policy network averages out by training on $\bar{\pi}$ from many searches. On RockSample(20,20) and both PocMan mazes, the policy network alone earns more return than every baseline does with search.

Effect of Variable Coupling on PocMan. **VC** is the BETS configuration of Table 1, and **Pooled** removes its variable-coupling stream and doubles the pooled embedding to 128 dimensions, which more than doubles the model's capacity. Everything else is held fixed, and both are trained with three seeds on PocMan(M). Table 2a shows that Pooled earns less return than VC with both search and the policy network, so the added capacity does not replace the variable-coupling stream.

Pooled has to discover the dependence between ghost positions and headings inside generic per-sample features. With search, Pooled also clears the maze in fewer than half as many episodes, and removing the stream costs more discounted return. Search scores its leaves with the value network and simulates states drawn from the belief, so the larger difference under search suggests that the coupling summaries improve both the value estimates and the simulated states.

Value Prediction from Belief Representations. We collect 50000 PocMan(M) episodes with the policy network from one training seed, while a bootstrap particle filter with the true model and 8192 particles tracks the belief on the same actions and observations. Value networks with the same architecture as those of Tables 1 and 2a are trained by squared error to predict the discounted return from

Table 2: Variable coupling and value prediction on PocMan(M).

	VC	Pooled
Discounted return (policy)	**72.42 ± 2.05**	65.39 ± 0.72
Discounted return (search)	**78.08 ± 0.54**	68.20 ± 0.48
Maze clears (search)	**14.1%**	6.7%

(a) Mean discounted return over three training seeds.

Episodes Collected	5000	10000	25000	50000
VC	**0.256**	**0.280**	**0.315**	**0.341**
Pooled	0.245	0.276	0.311	0.330
Moments	0.218	0.250	0.285	0.311

(b) Held-out R^2 of value networks.

each step, conditioned on either the mean and variance of the filter's particles (**Moments**) or the model's embedding (**VC / Pooled**). We report the held-out R^2 under five-fold cross-validation over episodes, where each network trains on episodes from three folds and early stops with the fourth. Table 2b shows that, for both models and at every training-set size, value networks conditioned on the embedding predict the return significantly more accurately than those conditioned on the moments (paired t-test over held-out episodes). The moment estimates use 16 times as many particles, so a shortage of particles does not explain the gap. This supports the attribution of BetaZero's gap on PocMan to conditioning on particle summary statistics, and conditioning on the combination of the pooled embedding and coupling summaries leads to better value estimates than conditioning on the pooled embedding alone.

6 Limitations

Paired simulations at the root spend the budget evenly among all actions, so each action's share shrinks as the action space grows. Sequential halving discards half of the remaining root actions after each phase and assigns each unvisited action an estimate of the root's value, so $\bar{\pi}$ remains defined over every root action (Danihelka et al., 2022). Pairing the actions within each phase would keep the variance reduction of common sampled states and extend BETS to large action spaces.

BETS must be trained on each domain before it can search effectively, while AdaOPS, DESPOT, and POMCPOW have no training phase. Against BetaZero, training improves returns where a particle set and its summary statistics represent the belief poorly, as on RockSample(20,20) and PocMan, but not on LightDark, where it adds computational overhead with no benefit over simpler belief models. Each simulation applies the NBF update, the policy network, and the value network once, so search takes roughly K times as long per decision as the policy network alone (Appendix D). The policy network alone earns more return than every baseline does with search on RockSample(20,20) and both PocMan mazes (Table 1), so BETS can act without search where decisions must be fast. Training also requires a simulator that exposes the realized hidden state, which might not always be available in practice.

7 Conclusion

This paper introduced Belief Embedding Tree Search (BETS), a Monte Carlo planner that represents the belief at search tree nodes with a generative distribution embedding. The embedding lets the search sample states, and it gives the policy and value networks a learned compact representation of the belief to condition on. We show that these function approximators benefit from conditioning on embeddings, rather than summaries of particle sets. BETS is competitive or leading on every configuration, and even without search it outperforms every baseline by a large margin on RockSample(20,20) and PocMan, the configurations with the highest-dimensional beliefs. Our experiments also suggest that adding per-variable coupling summaries to a GDE improves the model's representation on PocMan.

BETS shows that a learned belief embedding can take the place of a particle set inside a planner, and the same substitution is open to any algorithm that learns or plans on beliefs. Embedding and updating beliefs is not specific to tree search, so the same ideas extend to other learning and planning algorithms for partially observable environments. In games, player policies would enter the NBF update through the same particle propagation as the dynamics, so an embedding could represent public beliefs needed for many algorithms (Moravčík et al., 2017; Brown et al., 2020).

AI USE STATEMENT

Large language models assisted the authors with writing and coding. They were used to help the authors refine writing and help implement and test the experiment pipeline. The authors revised and verified all LLM work carefully.

REPRODUCIBILITY STATEMENT

All source code for BETS is provided as supplementary material and will be made publicly available on GitHub upon publication. Appendix B gives the hyperparameter settings of BETS and of every baseline.

REFERENCES

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Rémi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. In *International Conference on Learning Representations*, 2018.

Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Amir Beck and Marc Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175, 2003.

Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. Combining deep reinforcement learning and search for imperfect-information games. *Advances in Neural Information Processing Systems*, 33:17057–17069, 2020.

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

Ivo Danihelka, Arthur Guez, Julian Schrittwieser, and David Silver. Policy improvement by planning with Gumbel. In *International Conference on Learning Representations*, 2022.

Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. In *Handbook of Nonlinear Filtering*, pp. 656–704. Oxford University Press, 2009.

Nic Fishman, Gokul Gowri, Peng Yin, Jonathan Gootenberg, and Omar Abudayyeh. Generative distribution embeddings. *arXiv preprint arXiv:2505.18150*, 2025.

Walter R Gilks and Carlo Berzuini. Following a moving target: Monte Carlo inference for dynamic Bayesian models. *Journal of the Royal Statistical Society: Series B*, 63(1):127–146, 2001.

Neil J Gordon, David J Salmond, and Adrian FM Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings F (Radar and Signal Processing)*, 140(2):107–113, 1993.

Jean-Bastien Grill, Florent Altché, Yunhao Tang, Thomas Hubert, Michal Valko, Ioannis Antonoglou, and Rémi Munos. Monte-Carlo tree search as regularized policy optimization. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 3769–3778. PMLR, 2020.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.

Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, pp. 282–293. Springer, 2006.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, 2019.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023.

Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

Robert J Moss, Anthony Corso, Jef Caers, and Mykel J Kochenderfer. BetaZero: Belief-state planning for long-horizon POMDPs using learned approximations. *Reinforcement Learning Journal*, 1:158–181, 2024.

Christian Musso, Nadia Oudjane, and François Le Gland. Improving regularised particle filters. In Arnaud Doucet, Nando de Freitas, and Neil Gordon (eds.), *Sequential Monte Carlo Methods in Practice*, pp. 247–271. Springer, 2001.

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

Robert Platt, Russ Tedrake, Leslie Kaelbling, and Tomás Lozano-Pérez. Belief space planning assuming maximum likelihood observations. In *Robotics: Science and Systems*, 2010.

Tobias Pohlen, Bilal Piot, Todd Hester, Mohammad Gheshlaghi Azar, Dan Horgan, David Budden, Gabriel Barth-Maron, Hado van Hasselt, John Quan, Mel Večerík, Matteo Hessel, Rémi Munos, and Olivier Pietquin. Observe and look further: Achieving consistent performance on Atari. *arXiv preprint arXiv:1805.11593*, 2018.

Pascal Poupart and Craig Boutilier. Value-directed belief state approximation for POMDPs. In *Conference on Uncertainty in Artificial Intelligence*, pp. 497–506, 2000.

Christopher D Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, 2011.

Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-Draa. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 32:663–704, 2008.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

David Silver and Joel Veness. Monte-Carlo planning in large POMDPs. *Advances in Neural Information Processing Systems*, 23, 2010.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

Trey Smith and Reid Simmons. Heuristic search value iteration for POMDPs. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pp. 520–527, 2004.

Chris Snyder, Thomas Bengtsson, Peter Bickel, and Jeff Anderson. Obstacles to high-dimensional particle filtering. *Monthly Weather Review*, 136(12):4629–4640, 2008.

Samuel Sokota, Hengyuan Hu, David J Wu, J Zico Kolter, Jakob Nicolaus Foerster, and Noam Brown. A fine-tuning approach to belief state modeling. In *International Conference on Learning Representations*, 2022.

Christopher Solinas, Radovan Haluska, David Sychrovsky, Finbarr Timbers, Nolan Bard, Michael Buro, Martin Schmid, Nathan R Sturtevant, and Michael Bowling. Neural Bayesian filtering. *arXiv preprint arXiv:2510.03614*, 2025.

Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. DESPOT: Online POMDP planning with regularization. *Advances in Neural Information Processing Systems*, 26, 2013.

Zachary Sunberg and Mykel Kochenderfer. Online algorithms for POMDPs with continuous state, action, and observation spaces. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, pp. 259–263, 2018.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.

Sebastian Thrun. Particle filters in robotics. In *Conference on Uncertainty in Artificial Intelligence*, pp. 511–518, 2002.

Alexander Tong, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Kilian Fatras, Guy Wolf, and Yoshua Bengio. Conditional flow matching: Simulation-free dynamic optimal transport. *arXiv preprint arXiv:2302.00482*, 2023.

Joel Veness, Marc Lanctot, and Michael Bowling. Variance reduction in Monte-Carlo tree search. *Advances in Neural Information Processing Systems*, 24, 2011.

Chenyang Wu, Guoyu Yang, Zongzhang Zhang, Yang Yu, Dong Li, Wulong Liu, and Jianye Hao. Adaptive online packing-guided search for POMDPs. *Advances in Neural Information Processing Systems*, 34:28419–28430, 2021.

A Domains

RockSample. RockSample (Smith & Simmons, 2004) places a rover on a square grid containing k rocks at known positions, each independently good or bad. The rover moves deterministically in the four cardinal directions, samples the rock at its current cell, or checks any rock with a noisy long-range sensor. A check returns the rock's quality corrupted by noise whose accuracy decays exponentially with the distance to the rock. Sampling a good rock earns a positive reward and turns the rock bad, sampling a bad rock is penalized, and the episode ends with a positive reward when the rover exits the east edge of the grid. The rover's position is fully observable and transitions are deterministic. We use the two configurations of Moss et al. (2024): **RockSample(15,15)** is a 15×15 grid with 15 rocks and 20 actions, and **RockSample(20,20)** is a 20×20 grid with 20 rocks and 25 actions. Both use a fixed rock layout, sensor efficiency 20, discount 0.95, and horizon 100.

LightDark. LightDark (Platt et al., 2010) is a continuous one-dimensional localization domain. The agent's position is a real-valued state that is never observed directly. Each step yields an observation of the position corrupted by Gaussian noise whose standard deviation grows with the distance from a light. The actions move the agent one unit left or right or stop, and stopping ends the episode with a positive reward if the agent's true position is within distance 1 of the origin and a negative reward otherwise. Both configurations draw the agent's start position from $\mathcal{N}(2, 3^2)$, and use a discount factor 0.9 and horizon 100. **LightDark(5)** places the light at position 5. The standard deviation of the noise is $|y-5|/\sqrt{2}+0.01$ at position y, and stopping earns $+10$ or -10. **LightDark(10)** places the light at position 10 with noise standard deviation $|y - 10| + 10^{-4}$, and stopping earns $+100$ or -100.

PocMan. PocMan (Silver & Veness, 2010) is a partially observable adaptation of the arcade game Pac-Man. The agent navigates a maze, collecting food pellets while avoiding ghosts that chase it when nearby, and eating a power pill briefly lets the agent eat ghosts for a bonus. The observation is 11 bits: sight of a ghost along each of the four cardinal directions, contact with walls in each direction, hearing a ghost within Manhattan distance 2, and smelling a pellet and smelling a power pill in the eight surrounding cells. Rewards are -1 per step, $+10$ for a pellet or a power pill, $+25$ for eating a ghost, -100 for being caught, and $+1000$ for clearing the maze; the discount is 0.95. **PocMan(M)** is a 9×9 maze with 59 reachable cells, three ghosts, three power pills, and horizon 100. **PocMan(L)** is the 17×19 maze of Silver & Veness (2010) with 189 reachable cells, four ghosts, four power pills, and horizon 250. In both mazes, the agent, the ghosts, and the power pills start on the same cells in every episode. The hidden state is the ghosts' positions and headings together with the pellet field, in which each reachable cell holds a pellet independently with probability 0.5

at the start of an episode. The simulator forbids a ghost from reversing its direction, so its heading constrains its next move. The pellet field is not modelled by the generator and an observed pellet mask is given to all filters.

Policy and Value Network Inputs. The policy and value networks condition on the embedding θ together with a vector ξ of features of the observed part of the state. In every domain, ξ contains the fraction of the horizon elapsed. On PocMan it also contains the fraction of pellets remaining, the power-pill timer, and the observed pellet mask over the reachable cells.

B HYPERPARAMETERS

BETS. Search settings are shared between training and evaluation, and an annealed setting moves linearly from its first to its second value across the anneal window and keeps the second value afterwards. An action node keeps at most 16 observation children, and once it is full, a traversal whose observation matches none of them descends to an existing child sampled in proportion to its visit count plus one. On LightDark every sampled observation is new, so observation progressive widening lowers this limit to $\lceil k\, N(\theta, a)^{\alpha} \rceil$ (Table 3).

Table 3: BETS settings per configuration. A range is annealed over the anneal window.

	RockSample		LightDark		PocMan
	(15,15)	(20,20)	(5)	(10)	(M), (L)
Search					
Simulations per decision K	600	600	99	99	100
Particles per node n	256	256	256	256	512
Exploration constant c			20 to 3		
Prior mix m			0 to 0.75		
Temperature τ	2 to 0.5	2 to 0.5	2 to 0.5	20 to 5	10 to 2.5
Observation widening (k, α)	–	–	(4, 0.1)	(4, 0.1)	–
Training					
Gradient steps	150k	300k	75k	150k	60k
Anneal window (gradient steps)	10k to 40k	15k to 60k	7.5k to 30k	15k to 60k	4k to 16k
Collectors × fragment length	84 × 100	84 × 100	84 × 100	84 × 100	84 × 25
Collection interval (gradient steps)	400	400	400	400	100
Initial collection rounds	8	8	5	5	5
Value target λ	1	1	1	1	0.95
Optimizer	Adam, learning rate 10^{-3}, batch 32, gradient clip 1.0				
Networks					
Embedding dimension d_{emb}	32	32	32	32	64
VC stream (width × layers)	16 × 3	16 × 3	–	–	32 × 4
Set transformer width	128	128	128	128	256
Set transformer blocks, inducing points			3, 16		
Attention heads			4		
Generator (layers × width)	AR, 3 × 48	AR, 3 × 48	FM, 3 × 64	FM, 3 × 64	AR, 3 × 48
Value bins on $[f_{\min}, f_{\max}]$	66, ±8.1	66, ±8.1	66, ±3	66, ±9.2	128, $[-14, 38]$
Policy and value networks	MLP, 3 hidden layers of 128				

BetaZero. BetaZero runs with the same number of simulations per decision and particles as BETS in every domain, and otherwise with its default settings on RockSample and LightDark. On PocMan, BetaZero tracks the belief with a bootstrap particle filter. Its networks condition on the mean and standard deviation of the player and ghost positions and the mean pellet and power-pill grids over its particles. During policy iteration, BetaZero samples the root action from the visit counts weighted by the root action values ($z_q = z_n = 1$) at temperature τ, where $\tau = 0$ takes the argmax, and at evaluation it takes the argmax under the weights (z_q, z_n). A dash for widening means it is disabled, so every action is expanded at once and each node keeps only the first belief it is reached with (Table 4).

Table 4: BetaZero settings per configuration. A cell with two values gives the value during policy iteration, then the value at evaluation.

	RockSample		LightDark		PocMan
	(15,15)	(20,20)	(5)	(10)	(M), (L)
Policy iteration					
Iterations × episodes	50 × 500	50 × 500	30 × 500	30 × 500	40 × 250
Epochs per iteration	10	10	50	50	10
Optimizer	RMSProp	RMSProp	Adam	Adam	RMSProp
Learning rate	10^{-3}	10^{-3}	10^{-4}	10^{-4}	10^{-4}
Value loss	MSE	MSE	MSE	MSE	MAE
Dropout	0.5	0.5	0.2	0.2	0.5
Hidden layers × width	4 × 128	4 × 128	4 × 64	4 × 64	4 × 256
Batch, L_2 penalty, samples	1024, 10^{-5}, 10^5 from the latest iteration				
Search					
Simulations per decision	100 / 600	100 / 600	100 / 99	100 / 99	100
Particles	1000 / 256	1000 / 256	500 / 256	500 / 256	512
Exploration constant c	50	50	1	1	50
Maximum depth	15	4	10	10	10
Action widening (k_a, α_a)	– / (5, 0.9)	–	(2, 0.25)	(2, 0.25)	–
Belief widening (k_b, α_b)	– / (1, 0)	– / (1, 0)	(2, 0.1)	(2, 0.1)	–
Temperature τ	1 / 0	1.5 / 0	0	0	1 / 0
Evaluation weights (z_q, z_n)	(0.4, 0.9)	(0.5, 0.8)	(1, 1)	(1, 1)	(0, 1)
Bootstrapped Q_0	no / yes	no / yes	no	no	no

Online planners. AdaOPS, DESPOT, and POMCPOW run with the same number of simulations per decision and particles as BETS in every domain, and otherwise with the settings and heuristics from Moss et al. (2024) on RockSample and LightDark. They use the preferred-actions rollout policy of Silver & Veness (2010) on PocMan. The exit policy moves the rover east to the exit without sampling any rock, and on LightDark r is the reward for a correct stop (Table 5).

Table 5: AdaOPS, DESPOT, and POMCPOW settings per domain.

	RockSample	LightDark	PocMan
AdaOPS (Wu et al., 2021)			
Lower, upper bound	exit policy, QMDP†	$-r$, r	–
$m_{\min}$, δ	100, 0.1	10, 1.0	–
Maximum depth	90	90	–
DESPOT (Somani et al., 2013)			
Lower, upper bound	exit policy, QMDP†	random policy, r	preferred actions, fixed
Scenarios K, regularization λ	100, 0	30, 0.1	30, 0.01
Maximum depth		90	
POMCPOW (Sunberg & Kochenderfer, 2018)			
Leaf value	exit policy‡	MDP value	preferred-actions rollout
UCB constant	10	10	$\sqrt{2}$
Action widening (k_a, α_a)	–	(10, 0.5)	–
Observation widening (k_o, α_o)	(1, 1)	(4, 0.03)	(1, 1)
Maximum depth	90	20	100

† Fixed upper bound $V_{\max} = 138.3$ on RockSample(20,20).

‡ Random rollouts on RockSample(20,20).

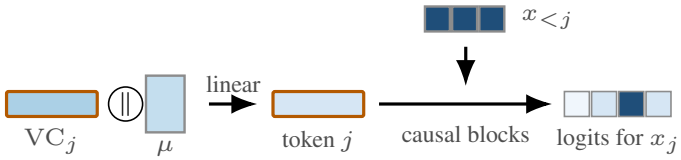

Figure 3: **Per-variable conditioning of the autoregressive generator.** Token j combines the coupling summary VC_j of variable j with the pooled embedding μ shared by every token. The causal blocks attend to the preceding values $x_{<j}$, and the output is the logits of the factor $p_\psi(x_j \mid x_{<j}, \theta)$.

C Generator Architectures

C.1 Autoregressive Transformer

The generator of Section 3.3 is a causal transformer that keeps one token per variable of the sample, and token j emits the logits of the factor $p_\psi(x_j \mid x_{<j}, \theta)$. Each token's input is the sum of three terms: a learned positional embedding for token j, an embedding of the preceding value x_{j-1} with a learned start token in place of x_0 at $j = 1$, and a conditioning vector computed from θ.

Pooled conditions the generator on one vector. A single linear map sends θ to the token dimension and the result is added to every token identically. The conditioning is the same at every token. VC instead splits θ into the pooled embedding μ and the D per-variable coupling summaries, and applies one linear map, shared by every token, to the concatenation $[\mu\,;\,\mathrm{VC}_j]$ at token j (Figure 3). Token j and summary j index the same variable, so factor j sees the shared pooled statistics and the summary for the variable it predicts.

C.2 Flow-Matching

LightDark's hidden state is a real-valued position, so its generator is a conditional flow-matching model (Lipman et al., 2023; Tong et al., 2023). A velocity field, an MLP with three hidden layers of width 64 that takes the current sample, the flow time, and θ as input, transports noise to a sample of the belief in 50 integration steps. Its belief loss is the conditional flow-matching regression at the realized state, and its embedding uses the pooled stream.

D Wall-Clock Time

We measure the wall clock time used by BETS, averaged over 100 episodes. Standard errors are below 3% of the mean. Each simulation applies the NBF update, the policy network, and the value network once, so search takes roughly K times as long per decision as the policy network alone (Table 6). The policy network alone, which earns more return than every baseline on RockSample(20,20) and both PocMan mazes (Table 1), takes under 25 ms per decision on every configuration.

Table 6: Milliseconds per decision of BETS on one NVIDIA GeForce RTX 5080.

		search			
Domain	n	$K = 50$	$K = 100$	$K = 200$	policy
RockSample(15,15)	256	140	229	460	2.8
	512	248	410	812	4.5
RockSample(20,20)	256	176	349	688	4.0
	512	323	641	1289	6.6
LightDark(5)	256	237	454	901	5.6
	512	452	866	1722	9.7
LightDark(10)	256	238	455	902	5.6
	512	452	867	1724	9.7
PocMan(M)	256	253	517	1042	5.6
	512	425	879	1729	9.2
PocMan(L)	256	894	1876	3722	19.6
	512	1107	2320	4692	24.1