Proceedings of the Thirty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2026)

GammaZero: Learning to Guide Belief-Space Search for Long-Horizon POMDPs with Generalizable Graph Representations

Rajesh Mangannavar, Prasad Tadepalli
Oregon State University, Corvallis, OR 97330, USA
{mangannr, prasad.tadepalli}@oregonstate.edu

Abstract

We introduce an uncertainty-aware graph representation framework for learning to guide planning in Partially Observable Markov Decision Processes (POMDPs). Unlike existing approaches that require domain or problem size specific neural architectures, GammaZero leverages a unified graph-based belief representation that enables generalization across problem sizes within a domain. Our key insight is that belief states can be systematically transformed into uncertainty-aware graphs where structural patterns learned on small problems transfer to larger instances. We employ a graph neural network with a decoder architecture to learn value functions and policies from expert demonstrations on computationally tractable problems, then apply these learned heuristics to guide Monte Carlo tree search on larger problems. Experimental results on standard POMDP benchmarks demonstrate that GammaZero achieves comparable performance to BetaZero when trained and tested on the same-sized problems, while enabling zero-shot generalization to problems 2-6× larger than those seen during training.

Code — https://tinyurl.com/GammaZero
Extended version — https://arxiv.org/abs/2510.14035

Introduction

Partially Observable Markov Decision Processes (POMDPs) provide a principled framework for sequential decision-making under uncertainty, where agents must act based on incomplete information about the true state of the environment (Kaelbling, Littman, and Cassandra 1998). This partial observability arises naturally in many real-world applications, from autonomous driving where sensors provide limited field-of-view (Hoel et al. 2019), to robotic manipulation where object properties must be inferred through interaction (Lauri, Hsu, and Pajarinen 2022), to subsurface exploration where underground structures can only be observed at sparse drilling locations (Mern and Caers 2023). The ability to reason explicitly about uncertainty while planning makes POMDPs particularly well-suited for safety-critical applications where robust decision-making is essential.

Despite their advantages, solving POMDPs exactly becomes computationally intractable for all but the smallest problems due to the curse of dimensionality in belief space (Shani, Pineau, and Kaplow 2013). Online planning algorithms such as POMCP (Silver and Veness 2010) and POMCPOW (Sunberg and Kochenderfer 2018) have made significant progress by using Monte Carlo tree search to focus computational effort on reachable belief states. However, these methods face fundamental limitations when dealing with long planning horizons and high-dimensional state spaces. Without effective heuristics to guide search, online planners struggle to look sufficiently far ahead to discover rewarding action sequences that may require extended information gathering (Ye et al. 2017).

The success of AlphaZero in fully observable domains demonstrates that learned approximations can effectively replace hand-crafted heuristics (Silver et al. 2018). Algorithms like BetaZero (Moss et al. 2024a) have extended this to partially observable domains by training neural networks to predict values and policies from belief states. However, reliance on fixed-size inputs creates a representational bottleneck, preventing generalization to larger problem sizes.

Graph Neural Networks (GNNs) offer a powerful alternative to address this scalability gap. While prior work has successfully leveraged GNNs to generalize policies in fully observable planning domains (Mangannavar et al. 2025), their application to the probabilistic belief spaces of POMDPs remains largely unexplored. In this work, we present GammaZero, a novel framework for learning to guide belief space search in POMDPs using uncertainty-aware graph representations, enabling a model trained on computationally tractable "toy" problems to generalize to large-scale instances that are computationally expensive for traditional online planners. Our contributions are:

1. **A graph-based belief representation for POMDPs** that transforms belief states into action-centric graphs encoding relationships between objects, their attributes, and actions. This enables learning from small problems to generalize to larger instances.
2. **Experimental validation:** Integration of learned value function and policy with MCTS showing GammaZero achieves comparable performance to BetaZero on same-sized problems, while enabling zero-shot generalization to problems 2-6× larger than training instances.

These contributions address long-horizon planning under uncertainty by combining graph neural networks' general-

ization capabilities with POMDP belief-space search.

Related Work

Monte Carlo Tree Search: Monte Carlo Tree Search (MCTS) is a best-first search algorithm that builds a search tree incrementally through repeated simulations (Browne et al. 2012). Each iteration consists of four phases: traversing the tree using a selection policy, adding new nodes to the leafs, simulating a rollout policy until terminal states, and updating statistics along the traversed path. The UCB1 formula commonly guides selection by balancing average rewards with exploration bonuses based on visit counts (Kocsis and Szepesvári 2006). While MCTS has achieved remarkable success in games and planning (Silver et al. 2016, 2017, 2018), applying it to POMDPs requires careful handling of belief states and typically demands substantial computational resources for reliable value estimates.

Online POMDP Planning: Classical online POMDP planning algorithms employ tree search methods to determine optimal actions through forward simulation. POMCP (Silver and Veness 2010) extends Monte Carlo tree search to POMDPs by maintaining particle beliefs at tree nodes, enabling planning in large state spaces. POMCPOW (Sunberg and Kochenderfer 2018) further extends this to continuous observation spaces through progressive widening. DESPOT (Ye et al. 2017) regularizes the search tree to focus on high-probability scenarios, while AdaOPS (Wu et al. 2021) adaptively adjusts particle beliefs to maintain value function bounds. While these methods rely heavily on domain-specific heuristics for value estimation and action selection, our approach learns generalizable graph-based representations that eliminate the need for hand-crafted heuristics.

Learning for Online POMDP Planning: Recent work has explored combining offline learning with online planning to reduce reliance on heuristics. BetaZero (Moss et al. 2024a) and ConstrainedZero (Moss et al. 2024b) learn neural network approximations of optimal policies and value functions offline, then use these to guide online MCTS. LeTS-Drive (Cai and Hsu 2022) similarly combines offline learning with online HyP-DESPOT planning for autonomous driving domains. These approaches fundamentally rely on fixed-size belief representations that must be predetermined for each domain. Our work differs by introducing a graph-based belief representation that naturally handles variable-sized problems and explicitly captures action-state relationships, enabling zero-shot generalization to problems significantly larger than those in training.

Learning for Classical Planning : The planning community has developed several approaches for learning generalized policies that transfer across problem sizes. GPL (Rivlin, Hazan, and Karpas 2020) learns value functions over relational state representations using GNNs, selecting actions by evaluating successor states. However, maintaining globally consistent value estimates becomes increasingly challenging as problems scale. ASNets (Toyer et al. 2020) employs alternating action and proposition layers with weight sharing, but its fixed-depth architecture limits reasoning about long dependency chains. GRAPL (Chrestien et al. 2024) learns to rank actions using canonical abstractions but lacks explicit modeling of action-object relationships and parameter dependencies.

Graph Representations for Planning: Graph neural networks have shown promise for learning generalizable planning policies due to their ability to handle variable-sized inputs and capture relational structure (Scarselli et al. 2008; Hamilton 2020). Existing work primarily focuses on deterministic, fully observable domains where states map directly to graph structures (Shen, Trevizan, and Thiébaux 2020; Karia and Srivastava 2021). Most recently, GABAR (Mangannavar et al. 2025) introduces an action-centric graph representation for deterministic planning, using a GNN encoder and a GRU-based decoder to incrementally construct grounded actions. While GABAR demonstrates strong generalization in fully observable domains, it cannot handle the belief uncertainty inherent in POMDPs. The key challenge addressed in the current paper is extending the graph representations to POMDPs by encoding belief uncertainty while preserving the structural patterns that enable generalization.

Problem Formulation

We formulate the problem of learning to guide belief space search as a supervised learning task. Given a POMDP $\mathcal{M} = \langle S, A, T, R, O, Z, \gamma, b_0 \rangle$ with states S, actions A, transition function T, reward function R, observations O, observation function Z, discount factor γ, and initial belief b_0, our goal is to learn both a value function $V_\theta : \mathcal{B} \to \mathbb{R}$ and a policy $\pi_\phi : \mathcal{B} \times A \to [0, 1]$ over the belief space $\mathcal{B}$ (Kaelbling, Littman, and Cassandra 1998).

The training data consists of tuples (b_i, v_i^*, π_i^*) where b_i is a belief state encountered during expert planning, v_i^* is the optimal value-to-go from that belief, and π_i^* is the optimal action distribution. These targets are obtained by running optimal or near-optimal planners on small problem instances where exact solutions are computationally feasible. The learned value function and policy are then used to guide online search on larger problem instances.

GammaZero

GammaZero operates in two phases: an offline learning phase where we train graph neural network approximations from expert-generated data on small problem instances, and an online planning phase where these learned models guide MCTS in belief space. The key innovation is our graph-based belief representation that captures relationships between actions, objects, and their associated uncertainties, enabling zero-shot generalization to larger problem instances without retraining. Figure 1 provides an overview of the framework. We detail each component below.

Belief State as Uncertainty-Aware Graph

Graph Construction Principles Our graph representation transforms particle-based belief states into structured graphs that capture both the uncertainty inherent in partial observability and the action-centric relationships necessary for decision-making. The core principle is to create a sparse yet expressive representation where the graph topology itself encodes belief uncertainty, attribute instance nodes exist

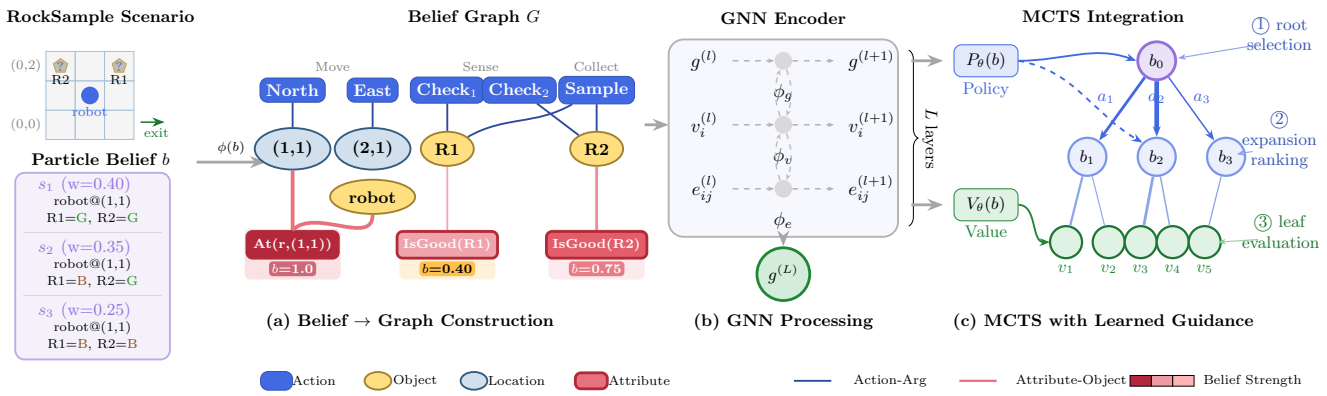

Figure 1: GammaZero framework. (a) Particle beliefs are transformed into uncertainty-aware graphs encoding objects, their attributes and actions. (b) A GNN processes the graph through L message-passing layers, updating node, edge, and global features. (c) The learned policy P_θ guides MCTS action selection while V_θ evaluates leaf nodes, replacing expensive rollouts.

Algorithm 1: GammaZero MCTS Planning

Require: Belief b, GNN $f_\theta = (V_\theta, P_\theta)$, simulations n_{sim}, depth d
Ensure: Selected action a^*

$\mathcal{T} \leftarrow \textsc{InitTree}(b)$
$G \leftarrow \phi(b)$ {Convert belief to graph}
for $i = 1$ to n_{sim} **do**
  $\textsc{Simulate}(\mathcal{T}, b, G, d)$
end for
for each action $a \in \textsc{Children}(\mathcal{T}.\text{root})$ **do**
  $\pi(a) \propto N(b,a)^{z_n} \cdot \exp(z_q \cdot Q(b,a))$
end for
$a^* \leftarrow \arg\max_a \pi(a)$
return a^*

Algorithm 2: GammaZero MCTS Simulation

Require: Tree $\mathcal{T}$, belief b, graph G, depth d
Ensure: Value estimate q

if $d = 0$ or $\textsc{IsTerminal}(b)$ **then**
  return $\mathbf{V}_\theta(\mathbf{G})$ {Leaf evaluation}
end if
if $\textsc{IsLeaf}(\mathcal{T}, b)$ **then**
  $\textsc{Expand}(\mathcal{T}, b, \mathbf{P}_\theta(\mathbf{G}))$ {Policy-guided expansion}
  return $\mathbf{V}_\theta(\mathbf{G})$
end if
$a \leftarrow \arg\max_a \left[Q(b,a) + c \cdot \mathbf{P}_\theta(\mathbf{a} \mid \mathbf{G}) \cdot \frac{\sqrt{\sum_{a'} N(b,a')}}{1+N(b,a)} \right]$
$s \sim b$
$s', o, r \leftarrow \textsc{Step}(s, a)$
$b' \leftarrow \textsc{UpdateBelief}(b, a, o)$
if $b' \notin \mathcal{T}$ **then**
  $\textsc{AddNode}(\mathcal{T}, b, a, b')$
end if
$G' \leftarrow \phi(b')$
$q \leftarrow r + \gamma \cdot \textsc{Simulate}(\mathcal{T}, b', G', d-1)$
$N(b,a) \leftarrow N(b,a) + 1$
$Q(b,a) \leftarrow Q(b,a) + \frac{q - Q(b,a)}{N(b,a)}$
return q

only when sufficient particle support justifies their inclusion, naturally encoding the multimodal nature of beliefs through the presence and absence of nodes. The graph schema, i.e., the object types, attribute types, and action types, is defined once per domain; given this schema, graphs are constructed automatically from any belief state.

Given a belief state b represented as a set of weighted particles $\{(s_i, w_i)\}_{i=1}^n$, where each particle s_i is a complete world state assigning values to all object attributes, we construct a graph $G = (V, E, X^V, X^E)$ where V is the node set, E is the edge set, X^V maps nodes to feature vectors, and X^E maps edges to feature vectors. We assume an object-centric state representation: each state decomposes into a fixed set of objects, each with typed attributes (e.g., `At(robot)`, `IsGood(rock1)`). To build the graph, we compute the probability of each attribute-value pair by aggregating particle weights where that assignment holds. This graph representation approximates the joint belief through independent per-attribute distributions; inter-attribute correlations (e.g., "all objects are co-located") present in the particles are not preserved in the graph (see Limitations). Figure 2 illustrates this construction on a RockSample domain, showing how particle beliefs are aggregated into per-attribute probabilities and selectively instantiated as graph nodes based on the threshold τ.

The node set consists of four distinct types:

$$V = V_{\text{obj}} \cup V_{\text{attr}} \cup V_{\text{act}} \cup \{v_{\text{global}}\} \tag{1}$$

where:

- V_{obj} contains **object nodes** representing all distinct entities in the environment. This unifies movable entities (e.g., robot, box, package) and spatial entities (e.g., rooms, hallways, distinct waypoints) into a single set. These nodes persist across all beliefs and serve as the valid arguments over which actions and attributes are

defined. They encode entity-specific properties, such as type (e.g., `is_location`, `is_item`) and static attributes.

- V_{attr} contains **attribute instance nodes** representing object attributes that hold with sufficient probability in the belief. Each attribute node encodes a specific attribute-value assignment for an object. For example, `At(robot)= (kitchen)` represents that the `At` attribute of `robot` has value `kitchen`. These nodes are created selectively: an attribute instance is instantiated only when $\sum_i w_i \cdot \mathbf{1}[\text{attr}(obj) = val \text{ in } s_i] \geq \tau$. This selective instantiation allows the graph structure to directly encode which attribute-value assignments are plausible under the current belief.
- V_{act} contains **action nodes** representing parameterized actions available in the domain (e.g., `move(?from, ?to)`, `pick(?object)`). These nodes enable the model to reason about action applicability by examining their connections to the relevant entities in V_{obj} and condition nodes in V_{attr}.
- v_{global} is a **global aggregation node** that maintains a holistic representation of the belief state and propagates information across distant nodes in the graph.

This structure serves multiple purposes. First, it separates persistent structural elements (entities and actions) from belief-dependent elements (attribute instances), allowing the model to distinguish between static domain knowledge and dynamic uncertainty. Second, the unification of locations and objects into V_{obj} simplifies the topology, treating spatial navigation and object manipulation as fundamentally similar operations defined by relationships between entities. Third, the selective creation of attribute nodes naturally handles multimodal distributions - if the belief assigns significant probability to the robot being in either the kitchen or hallway, both `At(robot, kitchen)` and `At(robot, hallway)` nodes will exist, with edge weights encoding their respective probabilities.

Belief-Driven Sparsity A key innovation is the belief-driven creation of attribute instance nodes. Rather than instantiating all possible attribute groundings, we create nodes only when the aggregated particle support exceeds a threshold τ:

$$\text{CreateNode}(\text{attr}(args)) \text{ iff } \sum_{i=1}^{n} w_i \cdot \mathbf{1}[\text{attr}(args) \in s_i] \geq \tau \tag{2}$$

This threshold-based construction serves multiple purposes: it naturally encodes the belief distribution through the graph topology (existence implies plausibility), reduces computational complexity by avoiding nodes for unlikely hypotheses, and enables the model to learn patterns from structural presence/absence rather than just numerical features.

Edge Construction and Belief Encoding Edges in our graph serve as the primary mechanism for encoding relationships between entities and, crucially, how belief uncertainty affects these relationships. The edge set consists of three primary categories:

$$E = E_{\text{attr-obj}} \cup E_{\text{act-obj}} \cup E_{\text{attr-act}} \tag{3}$$

where:

- $E_{\text{attr-obj}}$ contains **attribute-object edges** connecting attribute instance nodes to the relevant object nodes. Each attribute node connects to the object it describes and to the node representing its value. For instance, the node `At(robot) = (kitchen)` has edges to `robot` (the object whose `At` attribute is described) and to `kitchen` (the value of the attribute). Edge features distinguish between these two roles, and edge weights encode the belief probability of this attribute-value assignment.
- $E_{\text{act-obj}}$ contains **action-object edges** linking actions to objects that can serve as their parameters. These edges capture action applicability constraints and expected outcomes. For example, an edge from `check(?rock)` to `rock_3` encodes not only that rock_3 can be checked, but also the observation accuracy (decreasing with distance) and expected information gain.
- $E_{\text{attr-act}}$ contains **attribute-action edges** connecting attribute instances to actions that either require them as preconditions or produce them as effects. These edges encode how current beliefs constrain future actions and their expected outcomes.

Edge features capture multiple layers of information:

$$X_{ij}^{E} = [\phi_{\text{type}}(e_{ij}), \phi_{\text{role}}(e_{ij}), \phi_{\text{belief}}(e_{ij}), \phi_{\text{support}}(e_{ij})] \tag{4}$$

where:

- $\phi_{\text{type}}(e_{ij}) \in \{0,1\}^{10}$ is a 10-dimensional one-hot vector encoding the edge type (action-object, object-action, attribute-object, object-attribute, attribute-action, action-attribute, etc.), with bidirectional edges having distinct types to capture directionality.
- $\phi_{\text{role}}(e_{ij}) \in \{0,1\}^{2}$ is a 2-dimensional one-hot vector encoding the edge role for attribute edges: whether the edge connects to the object whose attribute is described (owner) or to the node representing the attribute's value.
- $\phi_{\text{belief}}(e_{ij}) \in [0,1]$ is the continuous belief probability for this relationship, computed as $\sum_{i=1}^{n} w_i \cdot \mathbf{1}[\text{relationship holds in } s_i]$.
- $\phi_{\text{support}}(e_{ij})$ is a categorical discretization of ϕ_{belief} into consensus levels: *unanimous* ($> 95\%$), *strong* (70–95%), *weak* (30–70%), or *split* ($< 30\%$). Both features derive from the same aggregated particle support, but the categorical version serves as an inductive bias that provides the network with explicit signal for patterns such as "split support suggests information-gathering actions are valuable." For example, if $\phi_{\text{belief}} = 0.72$ for IsGood(rock1), then ϕ_{support} is *strong*; a nearby value of 0.68 would instead be *weak*, making the transition salient even though the continuous values are close. A sufficiently expressive GNN could learn such boundaries from ϕ_{belief} alone, but the categorical feature makes these patterns easier to learn from limited training data.

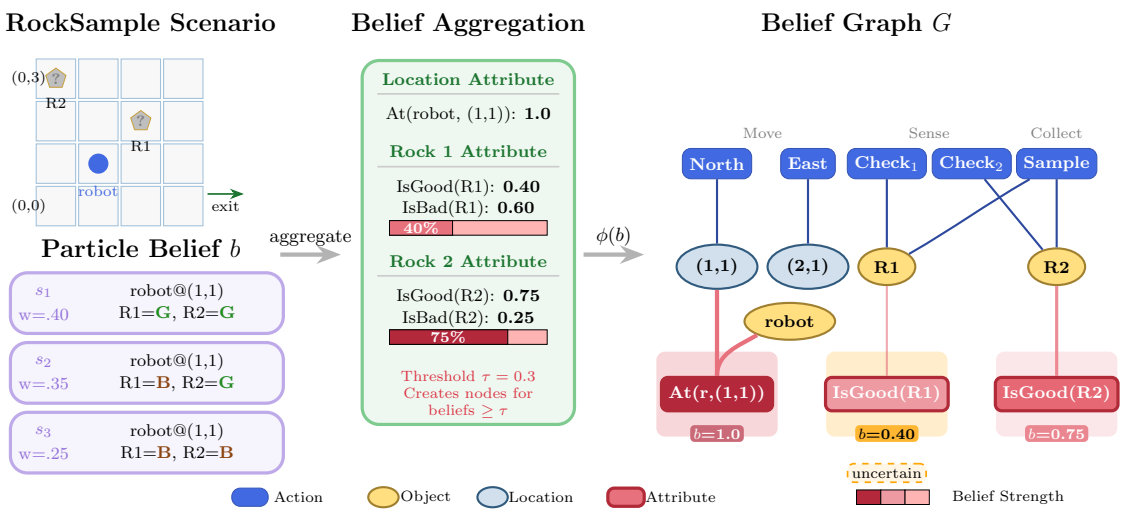

Figure 2: Belief graph construction pipeline. A particle belief from RockSample is aggregated into attribute probabilities (e.g., IsGood(R1)= 0.40, IsGood(R2)= 0.75), with attributes exceeding threshold τ instantiated as nodes. The resulting graph encodes objects, actions, and belief-weighted attributes, where edge thickness and node color intensity reflect uncertainty. Note: locations are visually distinguished for clarity but are treated as object nodes in V_{obj}.

This rich edge representation serves three critical functions. First, it encodes **belief-weighted relationships**—the edge weight between an attribute and its value directly represents how strongly this relationship is believed to hold. Second, it captures **action-specific context**—edges from check actions to objects encode observation accuracy based on distance, while edges from sample actions encode co-location requirements. Third, it provides **particle support metadata** that distinguishes between confident beliefs (high support from many particles) and uncertain hypotheses (support from few particles), enabling the model to reason about both the magnitude and confidence of beliefs.

This graph construction applies broadly to object-centric POMDPs domains where the state can be decomposed into discrete entities with attributes and relational structure between them (e.g., spatial adjacency, prerequisite dependencies). The model learns to interpret structural patterns (e.g., "high entropy on attribute nodes connected to an action indicates information-gathering value") rather than domain-specific features, enabling transfer learning across problem instances of vastly different scales within the same domain.

Graph Neural Network Architecture

We employ a graph neural network that processes the belief graph through L rounds of message passing:

$$e_{ij}^{(l+1)} = \phi_e([e_{ij}^{(l)}, v_i^{(l)}, v_j^{(l)}, g^{(l)}]), \tag{5}$$

$$v_i^{(l+1)} = \phi_v([v_i^{(l)}, \text{AGG}(\{e_{ij}^{(l+1)} : j \in \mathcal{N}(i)\}), g^{(l)}]), \tag{6}$$

$$g^{(l+1)} = \phi_g([g^{(l)}, \text{AGG}(\{v_i^{(l+1)}\}), \text{AGG}(\{e_{ij}^{(l+1)}\})]) \tag{7}$$

where ϕ_e, ϕ_v, and ϕ_g are learned update functions, and AGG represents an attention-weighted aggregation. The global node g enables rapid information propagation across the graph, crucial for maintaining performance as problem size increases.

The network outputs both a value estimate and action probabilities:

$$V_\theta(G) = \text{MLP}_v(g^{(L)}), \tag{8}$$

$$P_\theta(a|G) = \text{softmax}(\text{MLP}_p([g^{(L)}, v_a^{(L)}])). \tag{9}$$

Data Collection and Training

Expert Data Generation We collect training data by running optimal or near-optimal planners on small problem instances where exact solutions are computationally feasible. Algorithm 3 details our trajectory collection procedure. For each belief state encountered, we query the expert planner for the optimal action and Q-values (line 4), execute the action in the environment (line 7), and update the belief based on the received observation (line 9). After the episode terminates, we compute discounted returns via backward induction (lines 14–17), associating each visited belief with its value-to-go. This supervised approach leverages existing planning algorithms to generate high-quality training targets.

Loss Functions The network is trained using a combination of mean squared error for value prediction and cross-entropy for action classification:

$$\mathcal{L} = \lambda_v \|V_\theta(G) - v^*\|^2 + \lambda_p \mathcal{L}_{\text{CE}}(P_\theta(\cdot|G), a^*) \tag{10}$$

where $\mathcal{L}_{\text{CE}}$ is the cross-entropy loss and λ_v, λ_p are weighting coefficients. We use a shared GNN encoder with separate MLP heads rather than independent networks, as this avoids redundant feature learning and halves the inference cost during MCTS.

Algorithm 3: Collect Expert Trajectory

Require: POMDP $\mathcal{M}$, expert planner π^*, initial belief b_0, max steps T
Ensure: Dataset $\mathcal{D}$ of (belief, value, action) tuples
1: $\mathcal{D} \leftarrow \emptyset$, $b \leftarrow b_0$, $t \leftarrow 0$
2: $\mathcal{H} \leftarrow [(b_0, \cdot, \cdot)]$ {History buffer for return computation}
3: **while** $t < T$ and not IsTerminal(b) **do**
4: $a^*, Q^* \leftarrow \pi^*(b)$ {Expert action and Q-values}
5: $\mathcal{H}[t]$.action $\leftarrow a^*$
6: $s \sim b$ {Sample state from belief}
7: $s', o, r \leftarrow$ Step$(\mathcal{M}, s, a^*)$
8: $\mathcal{H}[t]$.reward $\leftarrow r$
9: $b \leftarrow$ UpdateBelief(b, a^*, o)
10: $\mathcal{H}$.append$((b, \cdot, \cdot))$
11: $t \leftarrow t + 1$
12: **end while**
13: {Compute discounted returns from end of episode}
14: $G \leftarrow 0$
15: **for** $\tau = t$ down to 0 **do**
16: $G \leftarrow \mathcal{H}[\tau]$.reward $+ \gamma \cdot G$
17: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathcal{H}[\tau]$.belief$, G, \mathcal{H}[\tau]$.action$)\}$
18: **end for**
19: **return** $\mathcal{D}$

Integration with Monte Carlo Tree Search

During online execution, the learned approximations enhance MCTS as detailed in Algorithms 1 and 2. Algorithm 1 initializes the search tree and converts the current belief to a graph representation (lines 1–2), then iteratively runs simulations to build the tree (lines 3–5). After all simulations complete, action selection combines visit counts and Q-values (lines 6–9).

Algorithm 2 details the recursive simulation process. The learned approximations enhance MCTS in three critical ways:

Action Prioritization: When expanding nodes, we sample actions from P_θ rather than uniformly, focusing search on promising actions. This policy-guided expansion occurs at line 5 of Algorithm 2, where the learned policy $P_\theta(G)$ initializes action priors for the newly expanded node:

$$a \sim P_\theta(\cdot|\phi(b)) \tag{11}$$

Value Estimation: At leaf nodes, we replace expensive rollouts with a value network lookup. This occurs at lines 2 and 6 of Algorithm 2, where terminal or newly expanded nodes are evaluated using $V_\theta(G)$ instead of Monte Carlo rollouts:

$$v = V_\theta(\phi(b)) \tag{12}$$

Root Action Selection: We combine visit counts with Q-values for robust action selection, as shown in lines 6–9 of Algorithm 1:

$$\pi(a|b) \propto N(b,a)^{z_n} \cdot \exp(Q(b,a))^{z_q} \tag{13}$$

The UCB-style action selection during tree traversal (line 8 of Algorithm 2) balances exploitation of high-value actions with exploration guided by the learned policy prior $P_\theta(a \mid G)$, adapting the PUCT formula from AlphaZero to our belief-space setting.

After selecting an action, the algorithm samples a state from the current belief and simulates a transition to obtain the next state, observation, and reward (lines 9–10). Then, weighted particle belief update is performed using a domain-specific transition function to obtain the new particle belief state (line 11). If the successor belief b' is not yet in the tree, a new node is added (lines 12–13). The successor belief is converted to its graph representation $G' \leftarrow \phi(b')$ (line 15), and the value estimate is computed recursively (line 16). Finally, backpropagation updates the visit count $N(b,a)$ and running Q-value estimate $Q(b,a)$ (lines 17–18), refining action valuations over successive simulations.

Experiments

We evaluate GammaZero's ability to learn generalizable value functions and policies for guiding belief-space search in POMDPs. Our experiments are designed to demonstrate three key capabilities: (1) comparable performance to existing methods when trained and tested on same-sized problems, (2) zero-shot generalization to problems significantly larger than those seen during training, and (3) computational efficiency gains through learned approximations. We compare against BetaZero, the current state-of-the-art method for learning-guided POMDP planning, along with classical online planners.

Experimental Setup

Domains We evaluate GammaZero on on four POMDP benchmark domains that exhibit different characteristics:

LightDark (LD) (Sunberg and Kochenderfer 2018) : A 1D localization problem where an agent navigates to a goal under state uncertainty, receiving noisy observations that improve near a "light" region at distance d. We test LightDark(5) and LightDark(10), where larger d requires longer-horizon information gathering.

RockSample(n,k)/RS(n,k) (Smith and Simmons 2004): An information-gathering problem on an $n \times n$ grid with k rocks of unknown quality. The agent must use a distance-dependent noisy sensor to assess rocks before sampling. State space: $O(n^2 \cdot 2^k)$, ranging from 12,800 states for (7,8) to over 400 million for (20,20).

MultiObjectSearch(n,k)/MOS(n,k) (Wandzel et al. 2019): A target localization problem where a robot must find and declare k hidden objects on an $n \times n$ grid using a range-limited sensor with detection probability ϵ. State space: $O(n^{2(k+1)} \cdot 2^k)$. We test MOS(6,4) and MOS(8,6).

Rearrangement(n,k)/RG(n,k): A mobile manipulation domain we introduce for this evaluation, where a robot must locate k objects with unknown positions and transport them to goal locations on an $n \times n$ grid. State space: $O(n^{2(k+1)} \cdot 4 \cdot 2^k)$, combining perceptual uncertainty with multi-step planning. We test (6,4) to (8,5).

Baselines We compare GammaZero against the following baselines:

BetaZero (Moss et al. 2024a): The state-of-the-art learning-based POMDP planner that combines offline neu-

ral network training with online MCTS. BetaZero learns policy and value networks from expert demonstrations but requires separate training for each problem size due to its fixed-dimensional belief representation.

POMCPOW (Sunberg and Kochenderfer 2018): A model-based online planner that extends POMCP to continuous observation spaces through progressive widening. We test POMCPOW both with domain-specific heuristics.

AdaOPS (Wu et al. 2021): An adaptive online planner that maintains value function bounds through particle filtering. We test AdaOPS with fixed bounds for problems where QMDP is intractable.

DESPOT (Ye et al. 2017): An online planner that uses scenario sampling to construct a sparse belief tree, with regularization to balance policy size and estimated value.

Evaluation Metric We evaluate all methods using average return: the expected cumulative discounted reward achieved by the policy, averaged over 100 episodes with different random seeds. For GammaZero and BetaZero, we train 5 runs per configuration and report results from the best-performing setting. We report the mean and standard error.

Experimental Design

Our experiments are structured to answer two questions:

RQ1 (Table 1): How does GammaZero compare to BetaZero and other classical planners when trained and tested on the same problem size? This establishes a performance baseline and validates that our graph-based approach achieves comparable results to the state-of-the-art.

RQ2 (Table 2): Can GammaZero generalize to problems larger than those seen during training? We train on small instances (e.g., RockSample(5,5) to RockSample(10,10)) and test on problems 2-6$\times$ larger. BetaZero cannot perform this zero-shot generalization due to its fixed input dimensions, requiring expensive retraining for each problem size.

Implementation Details

GammaZero uses a graph neural network with 3 hidden layers. During online planning, we use PUCT exploration with $c = 50$ for action selection within MCTS, with progressive widening parameters $k_a = 2.0$, $\alpha_a = 0.9$. It takes 2-4 hours to train a model for each domain on an RTX 3080.

Results and Discussion

Same-Size Performance (RQ1)

Table 1 presents performance comparisons when all methods are trained and tested on identical problem sizes. GammaZero consistently matches or outperforms BetaZero across all domains where direct comparison is possible.

LightDark. On LightDark(10), GammaZero achieves 17.5 ± 1.2, outperforming BetaZero's 16.17 ± 1.58. This improvement is notable because LightDark requires extended information-gathering trajectories before committing to the goal, validating that our approach effectively captures the relationship between uncertainty reduction and future value.

RockSample. On RockSample(15,15), GammaZero achieves 20.5 ± 0.8 compared to BetaZero's 19.87 ± 0.91. Both learning-based methods substantially outperform POMCPOW (11.01 ± 0.67) and match AdaOPS (20.53 ± 0.81), demonstrating that learned heuristics dominate expensive hand-crafted value bounds.

Extended Domains. GammaZero extends to domains that BetaZero does not support due to its fixed-dimensional architecture. On both MultiObjectSearch(5,3) and Rearrangement(5,2), GammaZero significantly outperforms all classical baselines. These domains require coordinating perception actions (look) with commitment actions (find/pick), a pattern that our action-centric graph representation explicitly captures through attribute-action edges.

Ablation Analysis. The "Raw P_θ" and "Raw V_θ" columns show the contribution of each component. The policy network alone (without MCTS) achieves $60 - 80\%$ of full performance, while one-step lookahead with the value network performs slightly worse. The combination through MCTS consistently yields the best results, confirming that both networks provide complementary guidance.

Generalization (RQ2)

Table 2 demonstrates GammaZero's unique capability: generalizing to problems significantly larger than those seen during training. BetaZero requires retraining for each problem size due to its fixed input dimensions.

Training Protocol. For LightDark, we train on size 5 and test on size 10. For RockSample, we train on instances ranging from 5×5 to 10×10 grids with 5-10 rocks, then test on $(15, 15)$, $(20, 20)$, and $(25, 25)$. For MultiObjectSearch and Rearrangement, we train on grid sizes 3-4 with 2–3 objects, then generalize to grid sizes 5-8 with 3-6 objects.

LightDark Generalization. Training on LightDark(5), GammaZero achieves 15.2 ± 1.5 on LightDark(10), dramatically outperforming all classical baselines including AdaOPS (6.28 ± 2.03). This demonstrates effective transfer of the light-seeking localization strategy to larger state spaces.

RockSample Generalization. GammaZero exhibits graceful degradation across increasing problem scales. On RockSample(15,15), it achieves 17.8 ± 1.2, competitive with DESPOT (18.83 ± 0.81). Performance remains strong on $(20, 20)$ at 10.2 ± 1.8, approaching AdaOPS (10.96 ± 0.78) - despite never seeing problems larger than 10×10 during training. On the extreme $(25, 25)$ configuration, where all methods struggle with timeouts, the raw policy network achieves best performance (4.8 ± 1.2), demonstrating that structural patterns learned on small instances transfer even when search becomes intractable.

MultiObjectSearch Generalization. Training on MOS instances with 2–3 objects on 3×3 to 4×4 grids, GammaZero generalizes across three scales: $(6, 4)$ achieving 14.5 ± 1.8, $(7, 5)$ at 11.2 ± 2.0, and $(8, 6)$ at 8.0 ± 2.2. These results significantly exceed classical baselines, which increasingly suffer from timeouts at larger scales. The model successfully transfers the coordination pattern between all actions to scenarios with twice the number of target objects.

Rearrangement Generalization. Similar patterns emerge in Rearrangement, where training on $(3, 2)$ to $(4, 3)$ configurations enables generalization to $(6, 4)$, $(7, 4)$, and

	GammaZero			BetaZero†			Classical Baselines		
Domain	Full	Raw P_θ	Raw V_θ^*	Full	Raw P_θ	Raw V_θ^*	POMCPOW	DESPOT	AdaOPS
LD(10)	**17.5**±1.2	14.4±1.3	13.3±1.4	16.17±1.58	13.98±1.08	12.45±1.13	1.08±0.53	0.73±0.44	6.28±2.03
RS(15,15)	**20.5**±0.8	11.1±2.0	9.1±2.2	**19.87**±0.91	11.04±0.88	9.44±0.55	11.01±0.67	18.83±0.81	**20.53**±0.81
MOS(5,3)	**18.0**±1.5	10.8±1.8	9.9±2.0	—	—	—	7.5±1.5	6.4±1.8	15.5±2.0
RG(5,2)	**12.5**±2.0	5.6±2.2	6.3±2.0	—	—	—	4.3±1.5	3.4±1.5	7.7±2.0

Table 1. Same-size performance comparison (Returns ± SE). All methods are trained and tested on the same problem size. Bold indicates best mean return; methods within one standard error of the best are also bolded. *One-step look-ahead using only the value network. †BetaZero only supports LightDark and RockSample. "–" indicates unsupported domain.

	GammaZero (zero-shot transfer)			Classical Baselines (per-size)		
Test Domain	Full	Raw P_θ	Raw V_θ^*	POMCPOW	DESPOT	AdaOPS
LightDark(10)	**15.2**±1.5	12.1±1.6	11.2±1.7	1.08±0.53	0.73±0.44	6.28±2.03
RockSample(15,15)	17.8±1.2	11.1±2.0	9.1±2.2	11.01±0.67	18.83±0.81	**20.53**±0.81
RockSample(20,20)	**10.2**±1.8	5.4±1.0	4.4±2.0	9.92±0.67	0.0±0.0†	**10.96**±0.78
RockSample(25,25)	3.5±2.0	**4.8**±1.2	**3.9**±1.5	2.1±0.8	0.0±0.0†	**4.2**±1.0
MOS(6,4)	**14.5**±1.8	8.8±2.0	8.1±2.2	5.5±1.6	4.8±1.8	12.2±2.0
MOS(7,5)	**11.2**±2.0	6.5±2.2	6.0±2.3	3.8±1.8	3.2±2.0	9.0±2.2
MOS(8,6)	**8.0**±2.2	4.8±2.5	4.5±2.5	0.0±0.0†	0.0±0.0†	**5.8**±2.5
Rearrange(6,4)	**9.2**±2.0	4.5±2.3	5.0±2.2	3.0±1.6	2.4±1.8	5.8±2.0
Rearrange(7,4)	**6.8**±2.2	3.2±2.5	3.8±2.3	0.0±0.0†	0.0±0.0†	4.0±2.2
Rearrange(8,5)	**4.5**±1.8	2.2±1.6	2.8±1.5	0.0±0.0†	0.0±0.0†	**2.9**±1.7

Table 2. Zero-shot generalization (Returns ± SE). GammaZero trained on small problems, tested on larger sizes. Classical baselines trained per-size for reference. *One-step look-ahead. †Search timeout/failure. Bold indicates best mean return; methods within one standard error of the best are also bolded.

$(8, 5)$. GammaZero achieves 9.2 ± 2.0, 6.8 ± 2.2, and 4.5 ± 1.8 respectively, substantially outperforming all baselines across scales. This domain combines perceptual uncertainty with multi-step manipulation planning.

Graceful Degradation. Across all domains, performance degrades gradually rather than catastrophically as problem size increases beyond training distribution. This contrasts with fixed-dimensional approaches that cannot process out-of-distribution inputs at all.

Importantly, the graph construction principles remain consistent regardless of problem size. For example, adding more rocks to RockSample simply adds more object nodes and associated attribute instances; the graph topology and edge types remain unchanged. This allows the same GNN weights to process problems of arbitrary scale.

Limitations: The scope of this work is object-centric POMDPs where the state decomposes into discrete entities with attributes and inter-entity relations. The graph construction assumes V_{obj} is known at planning time; domains where objects appear, disappear, or have unknown cardinality would require dynamic graph construction. Additionally, particle beliefs are aggregated into independent per-attribute probabilities, so two beliefs with identical marginals but different correlations produce the same graph; capturing joint structure would require higher-order representations such as hyperedges. Finally, the current formulation also assumes discrete action spaces; extending to continuous actions requires modifications to the action-node representation.

Conclusions

Planning under partial observability remains challenging for long-horizon tasks where traditional online methods struggle to search deeply enough to discover rewarding action sequences. We presented GammaZero, a framework that learns to guide belief-space search in POMDPs by transforming particle beliefs into action-centric graph representations. Our key insight is that approximating the belief state as a graph where nodes represent actions and discrete distributions over object properties enables GNNs to learn transferable knowledge. Experiments on standard POMDP benchmarks demonstrate that GammaZero achieves competitive performance with BetaZero on same-sized problems while uniquely enabling zero-shot generalization to instances that are both spatially larger (2-6× grid area) and more complex (up to twice as many objects) than training instances, enabling scalable deployment without retraining. Future directions include hierarchical graph representations for extreme-scale problems, extension to continuous state and action spaces, and self-supervised learning approaches to reduce dependence on expert demonstrations.

Acknowledgements

The authors acknowledge the support of Army Research Office under grant W911NF2210251 and the support of Defense Advanced Research Projects Agency under grant HR0011-24-9-0423.

References

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1): 1–43.

Cai, P.; and Hsu, D. 2022. Closing the Planning–Learning Loop With Application to Autonomous Driving. *IEEE Transactions on Robotics*, 39(2): 998–1011.

Chrestien, L.; Edelkamp, S.; Komenda, A.; and Pevny, T. 2024. Optimize planning heuristics to rank, not to estimate cost-to-goal. *Advances in Neural Information Processing Systems*, 36.

Hamilton, W. L. 2020. *Graph representation learning*. Morgan & Claypool Publishers.

Hoel, C.-J.; Driggs-Campbell, K.; Wolff, K.; Laine, L.; and Kochenderfer, M. J. 2019. Combining Planning and Deep Reinforcement Learning in Tactical Decision Making for Autonomous Driving. *IEEE Transactions on Intelligent Vehicles*, 5(2): 294–305.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2): 99–134.

Karia, R.; and Srivastava, S. 2021. Learning generalized relational heuristic networks for model-agnostic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 8064–8073.

Kocsis, L.; and Szepesvári, C. 2006. Bandit Based Monte-Carlo Planning. In *European Conference on Machine Learning*, 282–293. Springer.

Lauri, M.; Hsu, D.; and Pajarinen, J. 2022. Partially Observable Markov Decision Processes in Robotics: A Survey. *IEEE Transactions on Robotics*.

Mangannavar, R. D.; Lee, S.; Fern, A.; and Tadepalli, P. 2025. Graph Neural Network Based Action Ranking for Planning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Mern, J.; and Caers, J. 2023. The Intelligent Prospector v1.0: Geoscientific Model Development and Prediction by Sequential Data Acquisition Planning with Application to Mineral Exploration. *Geoscientific Model Development*, 16(1): 289–313.

Moss, R. J.; Corso, A.; Caers, J.; and Kochenderfer, M. J. 2024a. BetaZero: Belief-State Planning for Long-Horizon POMDPs using Learned Approximations. arXiv:2306.00249.

Moss, R. J.; Jamgochian, A.; Fischer, J.; Corso, A.; and Kochenderfer, M. J. 2024b. ConstrainedZero: Chance-Constrained POMDP Planning using Learned Probabilistic Failure Surrogates and Adaptive Safety Constraints. arXiv:2405.00644.

Rivlin, O.; Hazan, T.; and Karpas, E. 2020. Generalized planning with deep reinforcement learning. *arXiv preprint arXiv:2005.02305*.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2008. The graph neural network model. *IEEE transactions on neural networks*, 20(1): 61–80.

Shani, G.; Pineau, J.; and Kaplow, R. 2013. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems*, 27: 1–51.

Shen, W.; Trevizan, F.; and Thiébaux, S. 2020. Learning domain-independent planning heuristics with hypergraph networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 574–584.

Silver, D.; and Veness, J. 2010. Monte-Carlo Planning in Large POMDPs. *Advances in Neural Information Processing Systems (NIPS)*, 23.

Silver, D.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587).

Silver, D.; et al. 2017. Mastering the game of Go without human knowledge. *Nature*, 550(7676).

Silver, D.; et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419).

Smith, T.; and Simmons, R. 2004. Heuristic Search Value Iteration for POMDPs. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, 520–527.

Sunberg, Z. N.; and Kochenderfer, M. J. 2018. Online Algorithms for POMDPs with Continuous State, Action, and Observation Spaces. In *International Conference on Automated Planning and Scheduling (ICAPS)*, volume 28.

Toyer, S.; Thiébaux, S.; Trevizan, F.; and Xie, L. 2020. Asnets: Deep learning for generalised planning. *Journal of Artificial Intelligence Research*, 68: 1–68.

Wandzel, A.; Oh, Y.; Fishman, M.; Kumar, N.; Wong, L. L.; and Tellex, S. 2019. Multi-object search using object-oriented pomdps. In *2019 International Conference on Robotics and Automation (ICRA)*, 7194–7200. IEEE.

Wu, C.; Yang, G.; Zhang, Z.; Yu, Y.; Li, D.; Liu, W.; and Hao, J. 2021. Adaptive Online Packing-guided Search for POMDPs. *Advances in Neural Information Processing Systems (NeurIPS)*, 34: 28419–28430.

Ye, N.; Somani, A.; Hsu, D.; and Lee, W. S. 2017. DESPOT: Online POMDP Planning with Regularization. *Journal of Artificial Intelligence Research*, 58: 231–266.