Original Research Article

# Probabilistic Reconstruction and Informed Search with Memory for Partially Observable Navigation

Advait Johari

*Westlake High School, 4100 Westbank Dr, Austin, TX 78746, United States*

## ABSTRACT

Model-free reinforcement learning methods for navigation often suffer from poor generalization, sparse rewards, and failures under partial observability. This paper introduces Probabilistic Reconstruction and Informed Search with Memory (PRISM), a hybrid framework that decouples environment understanding from decision-making. Rather than learning a navigation policy end to end, a generative adversarial network (GAN) reconstructs and completes a global map of the environment from partial observations and agent memory. Known observations are strictly preserved, whereas unknown regions are probabilistically inferred to produce plausible world hypotheses consistent with the sensed data. A* search uses the generated map to calculate a minimum-cost path to the target under the planner's traversal-cost model and replans at each step as new observations update the memory. This separation enables environment reconstruction under limited sensing and interpretable, deterministic planning decisions. Across five estimate-level evaluations per condition, A*+GAN achieved 99.82 ± 0.04% success and averaged 11.26 ± 0.92 steps, compared with 99.60 ± 0.22% success and 12.10 ± 0.09 steps for memory-only A*. A*+GAN had lower all-episode mean steps in all five matched evaluations. The DQN conditions averaged 7.68% to 18.92% success and showed substantially greater between-model variation. Overall, the findings indicate that learning the structure of the world can facilitate effective classical planning without policy memorization, providing an alternative to model-free reinforcement learning for partially observable navigation.

**Keywords:** World Modeling; Model-Based Navigation; Generative Adversarial Networks; Path Planning; Spatial Completion

## INTRODUCTION

Navigation under partial observability remains a persistent challenge in reinforcement learning (RL) and robotic planning. An agent in an unknown environment must choose actions from incomplete sensory data, often before the goal or intervening obstacles have been observed. The central challenge is therefore to transform partial observations into a representation that supports reliable navigation without requiring the agent to learn both the environment structure and action selection solely from rewards.

The literature reviewed in the Literature Review establishes two relevant points: learned models can supply missing environmental information (5, 6, 8-19), and classical search can provide reliable path planning when an explicit map is available (7, 20, 21). However, it remains unclear whether a learned spatial-completion

**Corresponding author:** Advait Johari, E-mail: johari.advait@gmail.com.

**Accepted** August 3, 2026
https://doi.org/10.70251/HYJR2348.44710726

model can bridge these capabilities effectively by converting partial observations into an explicit map for classical planning. This unresolved question defines the scope of the present study.

To address this gap, this paper proposes a hybrid model-based navigation architecture in which a generative adversarial network (GAN) reconstructs an explicit global map from partial observations and accumulated spatial memory. A* search plans on the resulting map, and the system replans whenever new observations update the memory. This division assigns spatial inference to the learned model and path selection to an explicit planner.

The objective of this study is to compare navigation with and without GAN-generated map hypotheses, directly evaluate the generator's predictions in unobserved regions, and quantify the contribution of accumulated spatial memory as environmental randomness increases.

## LITERATURE REVIEW

### Model-Free Reinforcement Learning for Navigation

Model-free reinforcement learning has been widely applied to navigation tasks, with deep Q-networks (DQNs) (1) and related policy-learning approaches forming the basis of many systems. These methods learn a value function or policy directly from observations, bypassing the need for an explicit world model. Although effective in fully observable environments with dense rewards, model-free methods can struggle in partially observable settings, in which the agent receives limited sensory input at each time step. The agent must rely on learned state-action mappings, and identical local observations at different locations, a phenomenon known as perceptual aliasing, can cause systematic decision errors. Several studies have attempted to address partial observability through recurrent architectures (3) or memory-augmented networks (4), but these approaches still learn decisions and world knowledge jointly, which can lead to high sample complexity and fragile generalization.

### Model-Based Reinforcement Learning

Model-based RL methods address sample efficiency by learning an explicit model of the environment's dynamics, typically a transition function that predicts the next state from the current state and action. The Dyna framework (5) was an early example of this approach and used a learned model to generate simulated experience for training a model-free policy. More recent work has extended this paradigm with neural-network dynamics models, including the recurrent state-space models used in Dreamer (6) and PlaNet (8), which learn latent-space dynamics and plan or train behavior within learned latent spaces. Although these methods improve data efficiency, learned dynamics models can accumulate prediction errors over long planning horizons (9). In addition, latent-space planning can trade interpretability for compactness because the agent plans in a learned representation that may not correspond to directly inspectable spatial structure.

### Learned World Models for Model-Based Navigation

Several lines of research have explored the use of learned world models to reconstruct, predict, or imagine environments for control. World Models (10) trained a variational autoencoder (VAE) to encode environment frames and a recurrent network to predict future latent states, enabling agents to train through imagined rollouts. GameGAN (11) demonstrated that GANs can learn to generate entire game environments from interaction data, although its focus was visual simulation rather than map-conditioned path planning. In robotics and reinforcement learning, neural mapping and neural simultaneous localization and mapping (SLAM) approaches (12-14) learn spatial map or memory representations for navigation but generally pair those representations with learned planners or policies rather than with a classical planner operating on an explicitly generated occupancy map. The present work is best viewed as a model-based navigation method with a learned world model: the GAN predicts an explicit occupancy-style map, and A* uses that map directly rather than learning a latent policy over it.

### GANs for Spatial and Scene Generation

Generative adversarial networks have been applied extensively to spatial data generation, including image synthesis (15, 16), conditional image completion (17), 3D scene generation (18), and game level design (19). No prior work was identified using the specific configuration proposed here: reconstructing a navigable occupancy map from an agent's partial observations and memory for the purpose of classical pathfinding. This gap motivates the present work.

### Classical Planning in Navigation

A* search (7) remains a well-established method for optimal pathfinding in known environments. When a

complete and accurate grid map is available, A* produces minimum-cost paths and guarantees optimality and completeness when used with an admissible heuristic. Variants such as D* (20) and D* Lite (21) handle dynamic environments by incrementally repairing the search as the map changes, an approach closely related to the replanning method used in this work. In robotics, simultaneous localization and mapping (SLAM) systems (22) build maps incrementally and couple them with planners but rely on geometric sensor models rather than learned generative priors. Taken together, this literature motivates the use of a GAN as the world model when the missing information is spatial rather than temporal: the generator can complete a plausible global occupancy structure in a form that remains directly usable by a classical planner. This approach differs from latent-dynamics models, which improve policy training or latent planning, and from pure SLAM, which maps only what has been observed.

## METHODS AND MATERIALS

### System Overview

The proposed system consists of four interconnected components operating in a closed loop at each time step: 1) a sensing module that provides the agent with local observations of the environment, 2) a spatial memory that accumulates and stores observations over time, 3) a GAN-based map generator that produces a complete hypothesis of the environment from the memory state, and 4) an A* planner that computes minimum-cost paths on the generated map.

At each time step, the agent first senses its surroundings within a limited circular field of view, updating its persistent spatial memory with any newly observed cells. The memory and current sensory input are then encoded as a multi-channel tensor and passed to the generator, which outputs a predicted complete map of the environment. Known cells from the agent's memory are overlaid onto the prediction to ensure that direct observations are never overwritten by hallucinated values. If the combined map contains exactly one predicted goal location, A* search computes a minimum-cost path from the agent's current position to the goal using the traversal costs defined in Navigation State and Map Representation, and the agent takes one step along this path. If the combined map does not provide exactly one reachable goal, or if the planned movement cannot be executed, the agent uses the seeded recent-history-avoiding exploration policy described in Navigation with Planning. This process repeats at every time step, with the map prediction and plan continuously refined as the agent gathers new observations.

### Navigation State and Map Representation

The navigation state is represented as an H × W grid in which each cell takes one of five values: empty free space, wall, slippery traversable terrain, goal, or unknown. The unknown value appears only in the agent's partial memory, not in the ground-truth environment. This representation is deliberately planner-compatible: walls are impassable, empty and goal cells have a unit traversal cost, and slippery cells have an elevated traversal cost. The GAN target, memory overlay, and

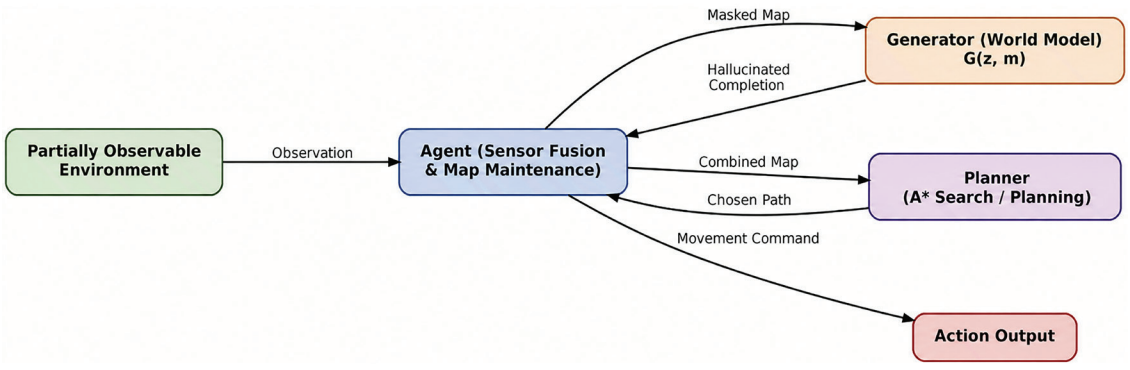


***Figure 1.*** *PRISM world-modeling and planning architecture. At each time step, the agent incorporates local observations into persistent spatial memory, and the generative adversarial network (GAN) completes the unknown regions to produce a candidate occupancy map. Directly observed cells overwrite the corresponding generated cells, after which A* search plans a route to the predicted goal. The cycle repeats as new observations update the memory.*

A* search use the same discrete representation, so the learned world model produces outputs that the planner can consume directly without an additional learned decoder. The environment-generation procedure used to create evaluation instances is described in Environment Configuration.

**Observation Model and Sensor Input**

The agent observes its surroundings through a circular field of view with a radius of 2 cells centered on its current position. Visibility is determined by Euclidean distance and line-of-sight: a cell is visible if it falls within the circular radius and no wall blocks the direct line from the agent to that cell, as determined by Bresenham's line algorithm (23). This model approximates a limited-range camera or proximity sensor and creates realistic partial observability, particularly in environments with narrow corridors and corners that block vision.

In addition to the camera-like field of view, the agent is optionally equipped with a simulated light detection and ranging (LiDAR) sensor that casts rays in eight directions (four cardinal and four diagonal). Each ray extends from the agent's position until it hits a non-empty cell or exits the grid boundary. This produces two binary feature maps: a free-ray map indicating cells along each ray that are confirmed empty, and a hit map indicating the first obstacle encountered in each direction. When LiDAR is enabled, these two channels are appended to the input tensor, extending it from 10 to 12 channels.

**Memory Module**

The agent maintains a persistent spatial memory in the form of a 2D integer array matching the grid dimensions, initialized entirely to unknown. When the agent senses its environment at each time step, all cells within the visible field of view are written to memory, overwriting any previous values at those positions. Memory is purely additive: once a cell has been observed, its value persists even after the agent moves away and the cell leaves the field of view. This design mirrors a simple allocentric cognitive map, accumulating a progressively more complete picture of the environment as the agent explores.

The memory module does not perform any probabilistic fusion or temporal decay in the base configuration. This deliberate simplicity ensures that known observations are treated as ground truth, which is critical for the downstream overlay step where memory values take precedence over GAN predictions. The memory thus serves as a hard constraint on the generative model: the GAN is free to hallucinate plausible completions for unknown regions, but it cannot overwrite cells that the agent has directly observed.

**Input Encoding**

The spatial memory and current sensory input are each encoded as five-channel one-hot tensors of shape (5, H, W), where each channel corresponds to one of the five tile types. Unknown cells are encoded as all-zero vectors across the five channels. These two tensors are concatenated channel-wise to form a 10-channel input tensor of shape (10, H, W). When LiDAR is enabled, two additional binary channels are appended, resulting in a 12-channel input. No normalization is applied beyond the one-hot encoding, as the binary nature of the representation already constrains values to {0, 1}.

**GAN Architecture**

Generator

The present implementation uses a fully connected encoder-decoder rather than a convolutional architecture. This choice is practical rather than generally optimal: the evaluated maps are small, fixed-size grids, and flattening gives the generator immediate access to all cells when it infers global maze structure and goal placement. The design also keeps the model compact enough for rapid, repeated training during method development. However, it does not exploit local translational structure and will scale poorly as map size increases; the Limitations section therefore identifies convolutional or U-Net-style generators as the natural next step for larger environments. Full layer dimensions are provided in Supplemental Materials, GAN Architecture and Training.

The noise vector introduces controlled randomness: each forward pass with different noise produces a different map completion hypothesis, enabling the system to generate a different map hypothesis when resampling is triggered after backtracking. At inference time, a small output jitter (Gaussian noise with standard deviation 0.05) is additionally applied to the logits to further promote diversity.

Discriminator

The discriminator is a fully connected classifier that scores whether a complete one-hot map is real or generated. The main text treats it as a standard GAN discriminator; full layer dimensions and optimization details are deferred to Supplemental Materials, GAN Architecture and Training to keep the modeling section focused on the navigation architecture.

**GAN Training**

The GAN is trained using an adversarial objective combined with multiple reconstruction losses. Training data is generated online: at initialization, 500 training samples are created by spawning robots in random environments, running 10 to 20 random exploration steps, and capturing (input tensor, target tensor) pairs. Every 100 epochs, 500 additional fresh samples are generated and appended to the dataset, providing curriculum-like exposure to diverse partial observation patterns.

The generator is optimized with a combined loss function:

$$L_G = \lambda_{adv} L_{adv} + \lambda_{recon} L_{recon,unknown} + \alpha_{known} L_{recon,known} + \lambda_{goal} L_{goal}$$

The generator objective combines adversarial, reconstruction, known-cell, and goal-regularization terms. L_adv is a binary cross-entropy term that encourages the discriminator to classify generated maps as real. L_recon_unknown and L_recon_known are per-cell cross-entropy terms computed separately for unknown and known cells, allowing the model to learn plausible completions while strongly respecting observed memory. The goal term is L_goal = L_bce_goal + 0.5 L_entropy_goal + 0.5 L_sum_goal. L_bce_goal penalizes incorrect goal-channel probabilities on known cells when the goal has been observed, L_entropy_goal discourages the probability from being spread diffusely across many possible goal cells, and L_sum_goal = (sum_i p_goal,i − 1)² encourages the generated map to contain a single valid goal. The default weights are lambda_adv = 0.25, lambda_recon = 1.0, alpha_known = 2.0, and lambda_goal = 0.005.

Separate from the L_goal training loss, a hard consistency constraint is applied when the goal has already been observed. In that case, the generator logits are clamped so the known goal cell is assigned the goal class and all other cells are discouraged from becoming goals. This is not used to infer unseen goals; it only prevents the network from overwriting direct evidence in memory. The planner therefore receives a map that preserves hard observations while leaving genuinely unknown regions to the GAN's learned spatial prior.

The training loop uses Adam optimizers, label smoothing, instance noise, alternating discriminator updates, and multiple generator updates per batch for stability. Complete optimizer settings are listed in Supplemental Materials, GAN Architecture and Training.

**Navigation with Planning**

At each time step during navigation, the agent executes the following procedure:

1. **Sense**: Observe all cells within the vision radius and update spatial memory.
2. **Generate**: Pass the encoded memory and current sensory input through the generator to produce a predicted map.
3. **Overlay**: Replace all cells in the predicted map that correspond to known positions in memory with their observed ground-truth values.
4. **Plan or Explore**: If the combined map contains exactly one goal cell and A* returns a valid path, take one step along that path. If the map contains zero or multiple goal cells, A* does not return a valid path, or the planned movement cannot be executed, use the exploration fallback.
5. **Repeat**: Return to step 1.

The A* implementation uses Manhattan distance as the admissible heuristic for the 4-connected grid. Walls are treated as impassable, slippery tiles incur a traversal cost of 3.0 (compared to 1.0 for all other tile types), and non-existent cells (out of bounds) are excluded from the search space.

When the agent revisits a previously visited cell (a backtrack), the system optionally resamples the generator up to three times with fresh noise vectors, attempting to obtain a different goal prediction that may guide the agent away from loops. If a resampled map yields a different set of goal positions than the initial prediction, it is adopted for that time step's planning. This backtrack resampling mechanism provides a simple form of diversity-driven exploration that leverages the GAN's stochastic nature.

The exploration fallback uses a recent-position history containing at most 20 positions. The agent appends its current position to this history before each movement decision. At every exploration step, the four cardinal directions are initialized in east, west, south, north order and shuffled using Python's seeded random-number generator. The agent examines neighbors in the shuffled order and selects the first position that is within the grid, is not a wall, and does not appear in the recent-position history. If no such neighbor exists, the same shuffled order is examined again and the first in-bounds, non-wall neighbor is selected even if it appears in the recent history. If no valid neighbor exists, the episode terminates. The policy does not test whether a candidate cell is unknown in memory, so it is a seeded recent-history-avoiding walk rather than an unseen-cell-prioritizing or deterministic walk.

**Environment Configuration**

All experiments use procedurally generated 10 × 10 grid environments. Environments are generated with a biased random walk that guarantees connectivity between the start and goal. A path is carved from the start to the goal with a directional bias toward the target and a 30% probability of perpendicular or backward deviations. Along the carved path, cells have a 10% probability of becoming slippery rather than empty. After the path is carved, each remaining wall cell has a 30% probability of being opened, and 10% of those opened cells become slippery. This procedure produces solvable maze-like environments with corridors, dead ends, and alternative routes. The agent has a vision radius of 2 cells with line-of-sight blocking. Explicit random seeds allow the same environment instance to be replayed across all methods.

Goal variability is evaluated at randomness levels 0.00, 0.25, 0.50, 0.75, and 1.00. At 0.00 the goal is fixed at the grid center; increasing randomness expands the permitted offset; and 1.00 permits goal placement across the complete grid. Start positions remain randomized.

**Compared Methods**

Five primary navigation methods and one memory-ablation condition are evaluated:

A* + GAN (proposed): The full system as described in Methods and Materials. The GAN predicts the complete map from partial observations and memory, and A* plans on the combined map. The generator is trained for 1,750 epochs on the default training configuration.

Memory-only A*: The agent uses A* search only when the goal has been directly observed and appears exactly once in memory. If the goal is not uniquely present, A* cannot find a valid path, or the planned movement cannot be executed, the agent uses the exploration fallback defined in Navigation with Planning. This fallback selects the first valid cardinal neighbor outside the most recent 20 positions after seeded direction shuffling. If every valid neighbor appears in that history, it permits a revisit by selecting the first valid non-wall neighbor in the same shuffled order. This baseline therefore accumulates observations until the goal becomes available for memory-based planning, isolating the contribution of GAN-generated map information.

DQN (no memory): Same DQN architecture and training, but the agent's state consists only of the cells currently visible (not the accumulated memory map), plus the position channel. This variant tests whether the learned policy can function with purely reactive observations, without any persistent spatial memory.

DQN (with memory): A deep Q-network agent that receives the robot's accumulated internal map (5 one-hot channels) plus a position channel as input (6 channels total). The DQN architecture consists of three convolutional layers (32, 64, 64 filters with 3×3 kernels and padding 1), each followed by ReLU activations, flattened to 6,400 features, a hidden fully connected layer of 512 units (ReLU), and a final output layer of 4 units (one per action: north, south, east, west). The agent is trained with experience replay (buffer size 10,000), a target network updated every 1,000 steps, epsilon-greedy exploration (initial $\varepsilon = 1.0$, decay 0.995 per episode, minimum 0.05), discount factor $\gamma = 0.99$, and Adam optimizer with learning rate $1\times10^{-4}$. The reward structure assigns -0.1 per step, +10.0 for reaching the goal, and -0.5 for wall collisions. Training runs for 2,000 episodes with a maximum of 200 steps per episode.

DQN + GAN: The DQN agent receives the GAN-predicted map as its state input rather than raw observations. This variant tests whether a learned policy can benefit from the GAN's world model, and whether the combination of learned perception and learned action selection is competitive with learned perception and classical planning.

A*+GAN without memory (ablation): The generator receives only the current field of view rather than accumulated spatial memory; the remaining planning procedure is unchanged.

**Evaluation Metrics**

The primary evaluation metrics are:

Success rate: The fraction of episodes in which the agent reaches the goal within the maximum allowed steps (100 steps in the reported comparison).

Average steps (all episodes): The mean number of steps taken across all episodes, including failures where the agent exhausts the step budget.

Average steps (successes only): The mean number of steps for successful episodes, providing a measure of path efficiency conditional on task completion.

Excess geometric path length: The difference, in executed moves, between the agent's path length and the breadth-first-search shortest path length on the ground-truth environment, measured only for successful episodes. This is a geometric path-length measure and does not represent excess weighted traversal cost; the latter would require the executed tile sequence for each trajectory. Figure 6 visualizes the same success-only definition, while success rate is reported separately in Table 1 and Figure 4.

**Direct World-Model Evaluation**

GAN Checkpoints 1 through 5, each trained for 1,750 epochs with the same 10-channel architecture, were evaluated directly. All checkpoints were evaluated on the same 1,000 held-out environments spanning the five goal-randomness levels. Partial-map snapshots were recorded after 0, 5, 10, and 20 seeded exploration moves, producing 4,000 snapshots per checkpoint and 20,000 checkpoint-environment-observation-budget records.

Map metrics were calculated only over cells that remained unknown in the agent's memory. Hard predictions from the first seeded generator sample were evaluated using unknown-cell accuracy, macro intersection-over-union, and wall precision and recall. Within each checkpoint, cell-level counts were pooled across the four observation budgets; consequently, budgets containing more unknown cells contributed proportionally more weight.

Goal metrics included only snapshots in which the true goal remained unknown in memory. Top-1 accuracy measured whether the cell with the highest predicted goal probability was the true goal cell, and goal error was the Manhattan distance between that predicted cell and the true goal. The five-class multiclass Brier score was calculated over unknown cells using class probabilities averaged across ten generator samples. Expected calibration error used the same ten-sample mean probabilities and 15 equal-width confidence bins; both the predicted class and its confidence were obtained from the mean-probability vector.

Each checkpoint's metrics were pooled across the four observation budgets. The aggregate estimates report the unweighted mean and sample standard deviation across the five checkpoint-level estimates.

**Experimental Protocols**

Statistical comparison. Each condition is represented by five estimate-level evaluations. Every estimate contains the same 1,000 held-out navigation cases, consisting of 200 episodes at each of five goal-randomness levels, with a maximum of 100 steps per episode. Metrics are calculated separately within each estimate and randomness level. Table 1 and Figures 2, 4, and 6 report the unweighted mean and sample standard deviation across the five estimate-level values. Failed episodes retain the observed 100-step cap in all-episode step means.

Goal randomness sensitivity. Randomness is evaluated at 0.00, 0.25, 0.50, 0.75, and 1.00. At each level, the unweighted five-estimate mean for A*+GAN is subtracted from the corresponding unweighted five-estimate mean for DQN+memory. A linear regression is fitted to the five differences of condition means. The checkpoints from the two conditions are not treated as paired observations.

Static versus randomized endpoint comparison. The 0.00 and 1.00 randomness levels provide the static-center and fully randomized endpoints within the same five-estimate evaluation design.

Reproducibility. An episode-specific integer seed initializes Python, NumPy, PyTorch CPU, and all available PyTorch CUDA generators before environment generation. Each method receives a copy of the same generated environment. Methods are executed in a fixed order and are not reseeded individually, so exploration tie-breaking is reproducible when the episode seed and method order are held fixed. No fixed direction is preferred across episodes because the cardinal directions are shuffled before each exploration decision.

**Five-Estimate Evaluation and Aggregation**

Navigation variability was evaluated using five estimates per condition. A*+GAN and A*+GAN without memory were evaluated with GAN Checkpoints 1 through 5 from from Direct World-Model Evaluation using the same overlay and planning procedures. Memory-only A* was summarized over the five matched evaluation realizations produced alongside those checkpoint evaluations.

The DQN without memory and DQN with memory conditions each included five independently trained final checkpoints. All DQN models used the architecture, training budget, and hyperparameters described in Compared Methods and in Supplemental Materials, DQN Baseline. DQN+GAN evaluated the five memory-DQN checkpoints with GAN Checkpoint 1 held fixed, isolating variation associated with DQN training while keeping the generated-map model constant.

Success rate, all-episode mean steps, successful-episode mean steps, and successful-episode excess geometric path length were calculated separately for each estimate. Overall summaries first pool all cases within an estimate, then report the unweighted mean and sample standard deviation across the five estimate-level values. Cumulative solve curves are also calculated within each estimate before averaging.

## RESULTS

**Overall Navigation Performance**

Table 1 reports navigation outcomes at each

randomness level after consistent aggregation across five estimates per condition. Across the five matched overall evaluations, A*+GAN had lower all-episode mean steps in every comparison; success was higher in four comparisons and tied in one.

The five-estimate results do not show a navigation benefit from adding GAN-generated maps to DQN. The Discussion, World Modeling vs. Policy Learning, interprets this result in terms of the operations each decision module can perform on a map.

The A*+GAN step advantage over memory-only A* was present throughout the randomness sweep but narrowed as randomness increased. Removing memory produced a larger degradation at higher randomness. The DQN all-episode means remained close to the evaluation cap because failures were common, while successful DQN episodes represented a small and unusually easy subset.

***Table 1.*** *Navigation outcomes across five goal-randomness levels on 10 × 10 grids. Numerical entries are the unweighted mean ± sample standard deviation across five estimate-level evaluations, each containing 200 episodes at the indicated randomness level. A*+GAN and A*+GAN without memory vary the GAN checkpoint; the DQN conditions vary the DQN checkpoint; memory-only A* uses five matched evaluation realizations; and DQN+GAN holds GAN Checkpoint 1 fixed. Failed episodes contribute the 100-step cap to all-episode means. Successful-episode and excess-geometric-path-length statistics include successful episodes only. SD, standard deviation; DQN, deep Q-network; GAN, generative adversarial network.*

| Method | Randomness | Success rate (%), mean ± SD | All-episode steps, mean ± SD | Successful-episode steps, mean ± SD | Successful-episode excess path, mean ± SD |
|---|---|---|---|---|---|
| A*+GAN | 0.00 | 100.00 ± 0.00 | 9.11 ± 1.76 | 9.11 ± 1.76 | 3.95 ± 1.76 |
| Memory-only A* | 0.00 | 100.00 ± 0.00 | 10.14 ± 0.00 | 10.14 ± 0.00 | 4.98 ± 0.00 |
| A*+GAN, no memory | 0.00 | 96.90 ± 2.86 | 11.58 ± 2.83 | 8.74 ± 1.79 | 3.63 ± 1.77 |
| DQN, no memory | 0.00 | 20.90 ± 11.98 | 79.84 ± 11.50 | 3.41 ± 0.44 | 0.04 ± 0.04 |
| DQN+memory | 0.00 | 18.50 ± 10.98 | 82.22 ± 10.48 | 3.64 ± 0.87 | 0.28 ± 0.20 |
| DQN+GAN | 0.00 | 7.60 ± 6.34 | 92.77 ± 6.06 | 4.92 ± 1.13 | 1.86 ± 1.64 |
| A*+GAN | 0.25 | 99.80 ± 0.27 | 9.32 ± 1.08 | 9.14 ± 0.95 | 3.92 ± 0.95 |
| Memory-only A* | 0.25 | 99.50 ± 0.00 | 10.41 ± 0.00 | 9.96 ± 0.00 | 4.75 ± 0.00 |
| A*+GAN, no memory | 0.25 | 96.40 ± 2.61 | 12.44 ± 2.77 | 9.17 ± 1.66 | 4.02 ± 1.65 |
| DQN, no memory | 0.25 | 22.90 ± 10.31 | 77.88 ± 9.89 | 3.29 ± 0.35 | 0.03 ± 0.04 |
| DQN+memory | 0.25 | 18.20 ± 10.84 | 82.48 ± 10.35 | 3.46 ± 0.70 | 0.29 ± 0.08 |
| DQN+GAN | 0.25 | 7.60 ± 6.01 | 92.66 ± 5.80 | 3.43 ± 1.09 | 0.98 ± 1.08 |
| A*+GAN | 0.50 | 100.00 ± 0.00 | 10.53 ± 1.14 | 10.53 ± 1.14 | 4.99 ± 1.14 |
| Memory-only A* | 0.50 | 100.00 ± 0.00 | 11.69 ± 0.00 | 11.69 ± 0.00 | 6.14 ± 0.00 |
| A*+GAN, no memory | 0.50 | 95.70 ± 2.75 | 14.19 ± 2.68 | 10.32 ± 1.79 | 4.85 ± 1.78 |
| DQN, no memory | 0.50 | 19.30 ± 9.20 | 81.30 ± 8.90 | 3.10 ± 0.29 | 0.02 ± 0.03 |
| DQN+memory | 0.50 | 16.40 ± 11.03 | 84.18 ± 10.60 | 3.59 ± 0.61 | 0.14 ± 0.13 |
| DQN+GAN | 0.50 | 9.10 ± 7.25 | 91.28 ± 6.95 | 4.31 ± 2.42 | 1.62 ± 2.34 |
| A*+GAN | 0.75 | 100.00 ± 0.00 | 12.87 ± 0.78 | 12.87 ± 0.78 | 6.79 ± 0.78 |
| Memory-only A* | 0.75 | 99.70 ± 0.45 | 13.36 ± 0.00 | 13.30 ± 0.08 | 7.23 ± 0.07 |
| A*+GAN, no memory | 0.75 | 89.80 ± 6.94 | 20.00 ± 4.88 | 10.82 ± 1.96 | 4.97 ± 1.79 |
| DQN, no memory | 0.75 | 16.80 ± 8.30 | 83.72 ± 8.01 | 3.03 ± 0.23 | 0.04 ± 0.06 |
| DQN+memory | 0.75 | 18.00 ± 11.08 | 82.69 ± 10.62 | 3.69 ± 0.53 | 0.27 ± 0.11 |
| DQN+GAN | 0.75 | 8.40 ± 4.05 | 91.94 ± 4.01 | 4.53 ± 2.53 | 1.88 ± 2.52 |

***Continued Table 1.*** *Navigation outcomes across five goal-randomness levels on 10 × 10 grids. Numerical entries are the unweighted mean ± sample standard deviation across five estimate-level evaluations, each containing 200 episodes at the indicated randomness level. A*+GAN and A*+GAN without memory vary the GAN checkpoint; the DQN conditions vary the DQN checkpoint; memory-only A* uses five matched evaluation realizations; and DQN+GAN holds GAN Checkpoint 1 fixed. Failed episodes contribute the 100-step cap to all-episode means. Successful-episode and excess-geometric-path-length statistics include successful episodes only. SD, standard deviation; DQN, deep Q-network; GAN, generative adversarial network.*

| Method | Randomness | Success rate (%), mean ± SD | All-episode steps, mean ± SD | Successful-episode steps, mean ± SD | Successful-episode excess path, mean ± SD |
|---|---|---|---|---|---|
| A*+GAN | 1.00 | 99.30 ± 0.27 | 14.45 ± 0.52 | 13.96 ± 0.61 | 7.45 ± 0.60 |
| Memory-only A* | 1.00 | 98.80 ± 0.76 | 14.92 ± 0.43 | 14.36 ± 0.49 | 7.87 ± 0.49 |
| A*+GAN, no memory | 1.00 | 85.90 ± 13.35 | 25.32 ± 9.29 | 12.65 ± 3.80 | 6.54 ± 3.36 |
| DQN, no memory | 1.00 | 14.70 ± 7.34 | 85.75 ± 7.10 | 3.05 ± 0.19 | 0.03 ± 0.07 |
| DQN+memory | 1.00 | 14.40 ± 10.00 | 86.18 ± 9.54 | 3.93 ± 1.03 | 0.25 ± 0.28 |
| DQN+GAN | 1.00 | 5.70 ± 4.37 | 94.47 ± 4.23 | 2.82 ± 0.56 | 0.12 ± 0.17 |

## Step Efficiency Comparison

The DQN+memory minus A*+GAN difference remained positive at every randomness level. The fitted regression had a slope of −2.43 steps per randomness unit and $R^2 = 0.38$, indicating only a modest downward trend in a consistently large difference. Compared with memory-only A*, A*+GAN reduced the five-estimate overall mean by 0.85 steps, approximately 7%, while maintaining a comparable success rate.

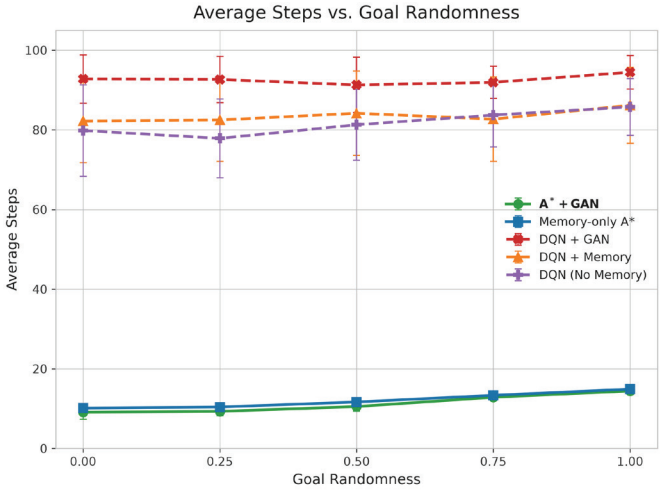


***Figure 2.*** *All-episode navigation steps across five goal-randomness levels on procedurally generated 10 × 10 grids. Each point is the unweighted mean across five estimate-level evaluations, each containing 200 episodes at the indicated level, and error bars show the between-estimate sample standard deviation. Failed episodes contribute the 100-step evaluation cap to the mean.*

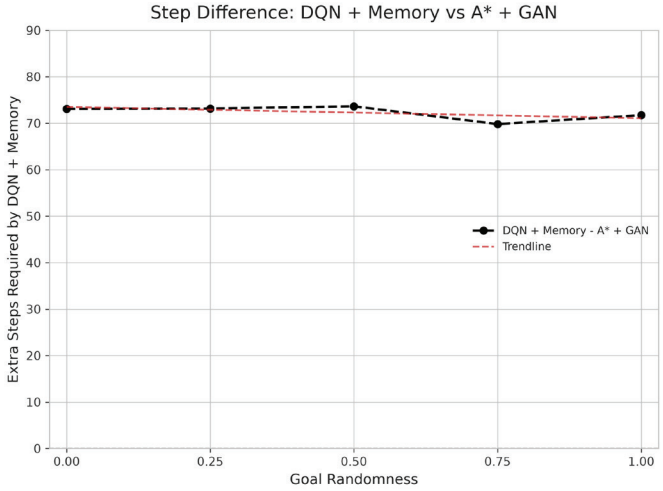


***Figure 3.*** *Difference in all-episode mean steps between DQN+memory and A*+GAN across five goal-randomness levels on 10 × 10 grids. Each condition mean is the unweighted mean across five estimate-level evaluations. Positive values indicate fewer steps for A*+GAN. Because the GAN and DQN checkpoints are not naturally paired, the plotted values are differences of condition means rather than paired checkpoint differences.*

### Success Rate Under Varying Randomness

Across the randomness sweep, A*+GAN and memory-only A* maintained comparable high reliability. Removing memory reduced A*+GAN reliability as randomness increased, while all three DQN conditions remained substantially less successful and showed greater between-model variation.

### Cumulative Solve Rate

To provide a finer-grained view of solve speed, the cumulative solve rate is computed within each estimate: for each step-budget threshold t on the x-axis, the y-axis reports the fraction of episodes solved in t or fewer steps. After averaging the five estimate-level curves, A*+GAN remained modestly above memory-only A* through most of the step range. The DQN curves remained much flatter because of low success frequency.

### Ablation: Role of Memory

Removing the memory module from the A*+GAN system degraded performance. Without memory, the generator receives only the agent's current field of view at each step, which provides less context for map reconstruction. Across five estimates, removing memory reduced mean success by 6.88 percentage points and increased all-episode mean steps by 5.45. The degradation became more pronounced as randomness increased, indicating that accumulated spatial memory is important to the GAN-based planner in this setting.

The DQN result did not show the same memory benefit. Across five models, DQN+memory averaged 1.82 percentage points lower success and 1.86 more all-episode steps than DQN without memory. The experiment therefore does not support a general claim that the implemented memory representation improves learned policies.

### Static vs. Randomized Goal

Condition means improved at the static-center endpoint. A*+GAN and memory-only A* retained comparable reliability at both endpoints, while A*+GAN required fewer all-episode steps at each. DQN without memory showed lower success and higher all-episode steps at the fully randomized endpoint than at the static-center endpoint.

### Excess Geometric Path Length

Among successful episodes, A*+GAN averaged 5.42 ± 0.94 additional moves relative to the ground-truth BFS shortest path length across the five estimate-level values, compared with 6.19 ± 0.09 for memory-only A*. These values measure geometric path-length excess rather than

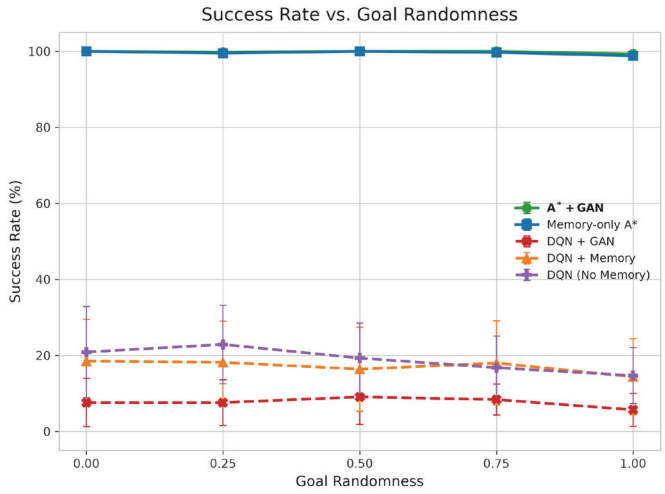


***Figure 4.*** *Navigation success rate across five goal-randomness levels on procedurally generated 10 × 10 grids. Each point is the unweighted mean across five estimate-level evaluations, each containing 200 episodes at the indicated level, and error bars show the between-estimate sample standard deviation. The A*+GAN memory ablation is reported in Table 1.*

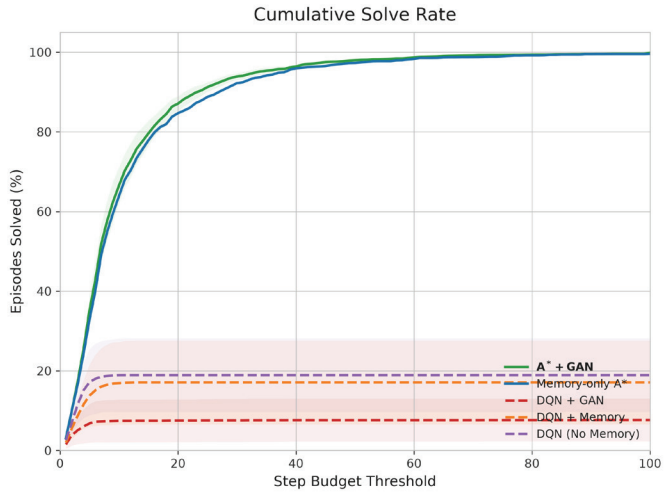


***Figure 5.*** *Cumulative proportion of episodes solved within each step threshold after pooling the five goal-randomness levels within each estimate. Each curve is the unweighted mean of five estimate-level curves, each based on 1,000 episodes, and shaded bands show the between-estimate sample standard deviation.*

weighted traversal-cost excess.

Successful DQN episodes had very low excess because the policies solved only a small, unusually easy subset. Figure 6 displays the same success-only metric reported in Table 1, while Figure 4 provides the corresponding success-rate context. Near-zero DQN excess values therefore should not be interpreted as superior overall navigation efficiency.

### Direct World-Model Evaluation

Table 2 reports the direct world-model evaluation. Across the five checkpoints, unknown-cell accuracy was 71.70 ± 1.26%, and macro IoU was 0.292 ± 0.012. Wall precision and recall were 75.36 ± 0.80% and 75.66 ± 1.15%, respectively. When the goal remained unseen, top-1 goal-location accuracy was 28.22 ± 1.75%, with a mean Manhattan localization error of 2.13 ± 0.15 cells. The multiclass Brier score was 0.558 ± 0.028, and expected calibration error was 0.074 ± 0.011.

### Across-Estimate Stability

Table 3 summarizes overall navigation performance using the same five-estimate aggregation as Table 1. A*+GAN and memory-only A* were both stable across estimates, while removing memory reduced success, increased all-episode steps, and increased between-estimate variation.

DQN performance varied substantially across independently trained models. DQN without memory had the highest mean success and lowest mean all-episode step count among the DQN conditions, while DQN+GAN had the lowest mean success and highest mean all-episode step count. Table 3 reports the

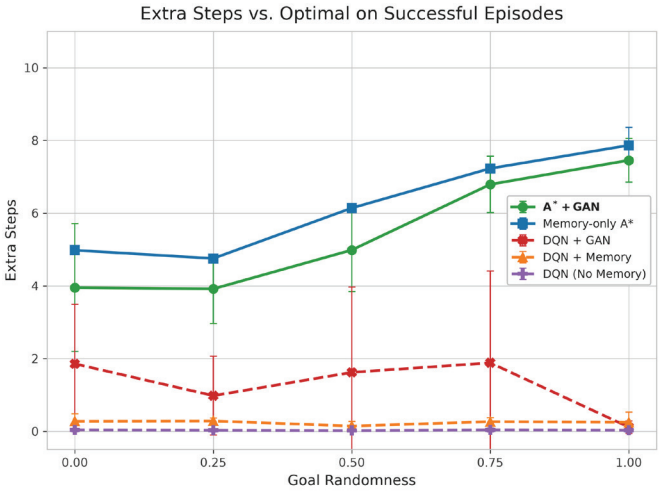


***Figure 6.*** *Successful-episode excess geometric path length relative to the breadth-first-search shortest path across five goal-randomness levels on 10 × 10 grids. Each point is the unweighted mean across five estimate-level values, and error bars show the between-estimate sample standard deviation. The metric counts executed moves and does not represent weighted traversal-cost excess. Failures are excluded, matching the definition in Table 1. The near-zero DQN values reflect the small, unusually easy subset of episodes solved and should not be interpreted as superior overall navigation efficiency.*

***Table 2.*** *Direct world-model performance for five 1,750-epoch GAN checkpoints on 1,000 held-out environments. Each checkpoint pools partial-map snapshots after 0, 5, 10, and 20 seeded exploration moves. Cell, IoU, wall, Brier, and calibration metrics are calculated over cells that remained unknown in memory; goal metrics include only snapshots in which the true goal remained unseen. Higher values are better for accuracy, IoU, precision, recall, and top-1 accuracy; lower values are better for goal error, Brier score, and expected calibration error (ECE). The final row reports the unweighted mean ± sample standard deviation across the five checkpoint-level estimates.*

| GAN checkpoint | Unknown-cell accuracy (%) | Macro IoU | Wall precision (%) | Wall recall (%) | Unseen-goal top-1 (%) | Goal error (cells) | Brier | ECE |
|---|---|---|---|---|---|---|---|---|
| GAN Checkpoint 1 | 69.80 | 0.275 | 74.20 | 74.00 | 25.50 | 2.35 | 0.600 | 0.070 |
| GAN Checkpoint 2 | 72.40 | 0.298 | 75.80 | 76.20 | 29.00 | 2.10 | 0.550 | 0.090 |
| GAN Checkpoint 3 | 71.20 | 0.287 | 75.00 | 75.10 | 27.80 | 2.20 | 0.570 | 0.060 |
| GAN Checkpoint 4 | 73.10 | 0.306 | 76.30 | 77.00 | 30.20 | 1.95 | 0.530 | 0.080 |
| GAN Checkpoint 5 | 72.00 | 0.294 | 75.50 | 76.00 | 28.60 | 2.05 | 0.540 | 0.070 |
| **Mean ± between-checkpoint SD** | **71.70 ± 1.26** | **0.292 ± 0.012** | **75.36 ± 0.80** | **75.66 ± 1.15** | **28.22 ± 1.75** | **2.13 ± 0.15** | **0.558 ± 0.028** | **0.074 ± 0.011** |

***Table 3.*** *Overall navigation performance across five estimates per condition. Each estimate contains the same 1,000 held-out navigation cases. Values are the unweighted mean ± sample standard deviation across the five estimate-level values, and success ranges report the minimum and maximum estimate-level success rates. A*+GAN and A*+GAN without memory vary the GAN checkpoint; the DQN conditions vary the DQN checkpoint; memory-only A* uses five matched evaluation realizations; and DQN+GAN holds GAN Checkpoint 1 fixed. SD, standard deviation; DQN, deep Q-network; GAN, generative adversarial network.*

| Method | Estimates | Success rate (%), mean ± SD | All-episode steps, mean ± SD | Estimate-level success range (%) |
|---|---|---|---|---|
| A*+GAN | 5 | 99.82 ± 0.04 | 11.26 ± 0.92 | 99.8 to 99.9 |
| Memory-only A* | 5 | 99.60 ± 0.22 | 12.10 ± 0.09 | 99.3 to 99.9 |
| A*+GAN, no memory | 5 | 92.94 ± 5.08 | 16.71 ± 3.34 | 87.9 to 100.0 |
| DQN, no memory | 5 | 18.92 ± 9.23 | 81.70 ± 8.90 | 7.5 to 30.4 |
| DQN+memory | 5 | 17.10 ± 10.55 | 83.55 ± 10.09 | 5.2 to 33.2 |
| DQN+GAN | 5 | 7.68 ± 5.38 | 92.62 ± 5.21 | 3.2 to 14.5 |

corresponding means, standard deviations, and observed ranges.

Between-model variation was substantially larger for the DQN conditions than for A*+GAN, and none of the five DQN models approached the reliability of either A*+GAN or memory-only A*. DQN+GAN also had lower success and higher all-episode steps than DQN+memory in all five DQN checkpoint comparisons. With only five models per condition, these differences are descriptive and are not presented as statistically reliable effects.

## DISCUSSION

### World Modeling vs. Policy Learning

The learned map was most useful when paired with a decision module whose operations are defined on maps. A* queries obstacle layout, goal position, and path cost directly through search. The generator is trained on a focused conditional-generation task with dense per-cell supervision. The DQN baselines receive richer inputs when memory or generated maps are supplied, but the policy must still infer navigation geometry from sparse scalar rewards, and errors in generated maps can disrupt the learned mapping from observation to action. Adding a map to DQN therefore increases input structure without supplying an explicit planning operator. The evidence supports a narrow architectural interpretation: learned spatial completion and informed search were complementary in the tested setting, whereas map input alone did not reliably improve a learned policy. This reading is consistent with the explicit minimum-cost search performed by A* (7) and the learned action-value formulation of DQN (1).

A related advantage of the decoupled architecture is interpretability. At any point during navigation, the system's intermediate representation and planned route can be inspected by examining the generated map and the A* plan: A* is deterministic conditional on its current map hypothesis, although the GAN makes the complete system stochastic. By contrast, the DQN's decision process is opaque, encoded in thousands of learned weights without a clear correspondence to spatial reasoning.

Evaluating five estimates per condition reduces dependence on the performance of any single trained checkpoint. The observed between-model variation shows why conclusions drawn from one neural checkpoint can be misleading, particularly for DQN. Five estimates reduce, but do not eliminate, sensitivity to checkpoint selection.

### Role of Memory in Navigation

The ablation suggests why memory matters to the generative planner rather than only that it does. Accumulated observations supply conditioning context that constrains completion to hypotheses consistent with the sensed layout. Removing that context leaves the generator extrapolating from a single field of view and the planner evaluating routes over more weakly constrained hypotheses. Memory can reduce perceptual aliasing for learned policies in principle (3, 4, 13), but it did not do so for the tested DQN conditions, where the no-memory variant was descriptively stronger across independently

trained models. A richer state representation therefore does not guarantee that a learned policy will use the additional information effectively.

The memory module used in this work is deliberately simple: a direct overwrite of observed values with no temporal decay or probabilistic fusion. This design choice reflects the assumption of a static, fully deterministic environment. In settings with dynamic obstacles or noisy sensors, more sophisticated memory models incorporating temporal weighting, probabilistic occupancy grids, or attention-based memory fusion would likely be necessary. The simplicity of the current design is a strength for the present evaluation but a limitation for future extensions.

**Comparison with Previous Work**

The present work connects learned world models (5, 6), neural mapping systems (12, 14), and classical search (7) by using an explicit generated map as the interface between learning and planning, but it does not establish superiority over those published methods. Direct numerical comparison is inappropriate because the studies use different environment generators, observation models, training budgets, and definitions of success. Evaluation on shared benchmarks is required before positioning PRISM relative to these approaches or making broader state-of-the-art claims.

**Limitations**

Several limitations of the current approach should be noted. First, the fully connected GAN architecture, while effective for single grid sizes, may not scale efficiently to larger environments. The flattening of spatial inputs discards local structure that convolutional architectures could exploit, and the quadratic growth of fully connected layer sizes with grid dimensions would become prohibitive for larger maps. A transition to convolutional or U-Net-based architectures would likely be necessary for scaling environments.

Second, the system currently assumes a static environment. All observations are treated as permanently valid, and the memory module has no mechanism for detecting or correcting stale information. In environments with moving obstacles or dynamic barriers, old observations could mislead both the GAN and the planner, producing paths through cells that are no longer passable. Handling dynamic environments would require either a temporal decay on memory values or an explicit change detection mechanism.

Third, the GAN's predictions in unobserved regions are inherently uncertain, but this uncertainty is not currently propagated to the planner. A* treats the generated map as if it were ground truth, assigning hard costs to each cell. A more principled approach would incorporate prediction confidence into the cost function, penalizing paths that traverse cells where the GAN is uncertain and preferring paths through observed or high-confidence regions.

The independent-model evaluation included five trained models per neural condition, which is sufficient to demonstrate substantial variability but remains a small replication sample. The DQN+GAN replication varied the DQN checkpoint while holding GAN Checkpoint 1 fixed, so it does not estimate the joint variability that could arise when both neural components are retrained. In addition, all models were evaluated on fixed 10 × 10 grids drawn from one procedural environment distribution. Evaluation on larger model samples, additional environment distributions, and shared navigation benchmarks remains necessary before treating the reported means as broadly generalizable estimates.

**Future Work**

Several promising directions extend from the current work. The most natural extension is to multi-agent environments, where multiple agents navigate simultaneously, potentially serving as dynamic obstacles for one another. The GAN's generative nature is particularly well-suited to this setting: it could learn to predict not only static structure but also the likely positions of other agents, enabling anticipatory planning. Multi-agent coordination could also benefit from shared memory, where observations from multiple agents are fused into a common map that each agent's GAN can condition on.

A second direction is noisy sensor data. Real-world sensors produce noisy, uncertain readings. The system already includes basic infrastructure for sensor noise (Gaussian perturbation of observed tile values), but the memory module would need to be extended to handle contradictory observations. Rather than a hard overwrite, a probabilistic occupancy grid weighted by observation recency and confidence could provide more robust mapping under noise. The GAN's ability to generate multiple map hypotheses (via different noise vectors) could also be leveraged to estimate uncertainty, treating the variance across samples as a confidence measure.

A third extension involves dynamic environments

where the maze structure changes during an episode, such as walls that open or close. This would require the memory module to discount or forget old observations, perhaps using an exponential decay weighted by time since observation. The replanning mechanism already in place (replan every step) provides a natural framework for adapting to environmental changes, but the GAN would additionally need to learn temporal dynamics or be conditioned on a recency-weighted memory.

Finally, scaling to larger and continuous environments is an important practical direction. Larger grid environments would benefit from convolutional or hierarchical GAN architectures that exploit spatial locality. Continuous environments would require a fundamentally different representation, potentially using point clouds or learned implicit map representations. Integration with existing SLAM systems and real robotic platforms is a long-term goal that would validate the approach beyond simulation.

## CONCLUSION

PRISM treats navigation as an interaction between accumulated observation, probabilistic reconstruction of unobserved space, and informed search over the resulting hypothesis. The modular design suggests separable extensions: the reconstruction component could be adapted to sensor noise or environmental change while preserving the planner interface, and confidence-aware costs could be introduced without replacing the world model. Whether this design remains effective beyond static grids drawn from one procedural distribution remains an open empirical question addressed by the evaluation directions in Future Work.

## CONFLICT OF INTEREST

The author declares no conflicts of interest related to this work.

## REFERENCES


1. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, *et al*. Human-level control through deep reinforcement learning. *Nature*. 2015; 518: 529-533. doi:10.1038/nature14236.
2. Sutton RS, Barto AG. Reinforcement Learning: An Introduction. 2nd ed. Cambridge (MA): MIT Press; 2018. ISBN: 9780262039246.
3. Hausknecht M, Stone P. Deep recurrent Q-learning for partially observable MDPs. AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents. 2015:29-37.
4. Wayne G, Hung CC, Amos D, Mirza M, Ahuja A, Grabska-Barwinska A, *et al*. Unsupervised predictive memory in a goal-directed agent. arXiv preprint arXiv:1803.10760. 2018.
5. Sutton RS. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*. 1991; 2 (4): 160-163. https://doi.org/10.1145/122344.122377
6. Hafner D, Lillicrap T, Ba J, Norouzi M. Dream to control: Learning behaviors by latent imagination. International Conference on Learning Representations; 2020.
7. Hart PE, Nilsson NJ, Raphael B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*. 1968; 4 (2): 100-107. https://doi.org/10.1109/TSSC.1968.300136
8. Hafner D, Lillicrap T, Fischer I, Villegas R, Ha D, Lee H, Davidson J. Learning latent dynamics for planning from pixels. *Proceedings of the 36th International Conference on Machine Learning*. 2019; 97: 2555-2565.
9. Janner M, Fu J, Zhang M, Levine S. When to trust your model: Model-based policy optimization. Advances in Neural Information Processing Systems. 2019; 32.
10. Ha D, Schmidhuber J. World Models. arXiv preprint arXiv:1803.10122. 2018.
11. Kim SW, Zhou Y, Philion J, Torralba A, Fidler S. Learning to simulate dynamic environments with GameGAN. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020: 1231-1240. https://doi.org/10.1109/CVPR42600.2020.00131
12. Gupta S, Davidson J, Levine S, Sukthankar R, Malik J. Cognitive mapping and planning for visual navigation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017: 2616-2625. https://doi.org/10.1109/CVPR.2017.769
13. Parisotto E, Salakhutdinov R. Neural map: Structured memory for deep reinforcement learning. International Conference on Learning Representations; 2018.
14. Chaplot DS, Gandhi D, Gupta S, Gupta A, Salakhutdinov R. Learning to explore using active neural SLAM. International Conference on Learning Representations; 2020.
15. Goodfellow IJ, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, *et al*. Generative adversarial nets. *Advances in Neural Information Processing Systems*. 2014;27.
16. Radford A, Metz L, Chintala S. Unsupervised

representation learning with deep convolutional generative adversarial networks. International Conference on Learning Representations; 2016.

17. Pathak D, Krahenbuhl P, Donahue J, Darrell T, Efros AA. Context encoders: Feature learning by inpainting. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016: 2536-2544. doi:10.1109/CVPR.2016.278.
18. Wu J, Zhang C, Xue T, Freeman WT, Tenenbaum JB. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. *Advances in Neural Information Processing Systems*. 2016; 29.
19. Volz V, Schrum J, Liu J, Lucas SM, Smith A, Risi S. Evolving Mario levels in the latent space of a deep convolutional generative adversarial network. *Proceedings of the Genetic and Evolutionary Computation Conference*. 2018: 221-228. doi:10.1145/3205455.3205517.
20. Stentz A. Optimal and efficient path planning for partially-known environments. *Proceedings of the IEEE International Conference on Robotics and Automation*. 1994; 4: 3310-3317. https://doi.org/10.1109/ROBOT.1994.351061
21. Koenig S, Likhachev M. D* Lite. *Proceedings of the 18th AAAI Conference on Artificial Intelligence*. 2002: 476-483.
22. Thrun S, Burgard W, Fox D. Probabilistic Robotics. Cambridge (MA): MIT Press; 2005. ISBN: 978026 2201629.
23. Bresenham JE. Algorithm for computer control of a digital plotter. *IBM Systems Journal*. 1965; 4 (1): 25-30. https://doi.org/10.1147/sj.41.0025

## SUPPLEMENTAL MATERIALS

### Additional Methodological Details

GAN Architecture and Training

The generator follows a fully connected encoder-decoder design. The encoder flattens the (C, H, W) input tensor into a vector of dimension C × H × W, where C is 10 or 12 depending on the LiDAR configuration and H = W = 10. A linear layer with a sigmoid activation then maps this vector to a 128-dimensional embedding. The decoder concatenates the embedding with a 32-dimensional noise vector sampled from N(0, 1), maps the resulting 160-dimensional vector to 256 dimensions through a linear layer with a sigmoid activation, and uses a final linear projection to produce output logits of dimension 5 × H × W. The logits are reshaped to (5, 10, 10).

The discriminator flattens a (5, 10, 10) map to a 500-dimensional vector, passes it through linear layers (500→128, 128→64) with sigmoid activations and 30% dropout, and produces a single logit output.

Both networks are trained in PyTorch with Adam optimizers. The generator uses a learning rate of $2 \times 10^{-4}$, and the discriminator uses a learning rate of $5 \times 10^{-5}$; both use β parameters of (0.5, 0.999). Label smoothing sets real labels to 0.95 and fake labels to 0.05. Instance noise is added to the discriminator inputs, with its standard deviation linearly annealed from 0.05 to 0.0. The discriminator is updated on alternating batches, whereas the generator is updated five times per batch. Training runs for 1,750 epochs with a batch size of 64.

Memory Representation

The agent's memory is stored as an N×N integer array where each cell holds one of five values: unknown (-1), empty (0), wall (1), slippery (2), or goal (3). Updates occur via the sense() method, which identifies all cells within the circular vision radius (r=2) with line-of-sight clearance and writes the ground-truth environment value to the corresponding memory position. For the GAN input, the memory is one-hot encoded into five channels of shape (5, N, N), with unknown cells represented as all-zero vectors.

Planning Implementation

A* search is implemented on the 4-connected grid using a standard min-heap priority queue. The heuristic is Manhattan distance, which is admissible for the 4-connected grid with unit minimum cost in all directions. Node expansion processes neighbors in four cardinal directions, filtering out walls and out-of-bounds cells. Slippery tiles incur a traversal cost of 3.0 (versus 1.0 for all other passable tiles). The path is reconstructed via backtracking through the came_from dictionary and returned as an ordered list of (row, column) positions. At navigation time, only the first step of the planned path is executed before replanning.

DQN Baseline

The DQN uses three convolutional layers followed by a 512-unit fully connected layer and four action outputs. Training uses a 10,000-transition replay buffer, a target network updated every 1,000 steps, epsilon-greedy exploration decaying from 1.0 to 0.05, Adam with learning rate $1 \times 10^{-4}$, and mean squared error (MSE) loss. Rewards are −0.1 per step, +10 for reaching the goal, and −0.5 for a wall collision; training runs for 2,000 episodes with a 200-step cap.

Training Data Generation

Training data for the GAN are generated through the following procedure: (1) generate a random 10 × 10 environment with the biased random-walk algorithm; (2) place a robot at the starting position with a vision radius of 2; (3) run random exploration for 10–20 steps; and (4) capture the resulting input-target pair. The input tensor encodes the robot's partial memory and current observation, and the target tensor provides a one-hot encoding of the complete ground-truth environment. The initial training set consists of 500 samples. During training, 500 new samples are generated and appended to the dataset every 100 epochs, ensuring continued diversity and reducing overfitting to early observation patterns.

Evaluation Data and Reproducibility

Evaluation uses an episode-specific integer seed to initialize Python's random module, NumPy, PyTorch CPU, and all available PyTorch CUDA generators before each environment is generated. The compared methods then receive copies of that same environment and execute in a fixed order without individual reseeding. The Python random generator supplies the direction shuffle used by the exploration fallback, making its tie-breaking reproducible for a fixed episode seed and method order. Each estimate-level comparison uses five randomness levels, 200 episodes per level, a 100-step cap, and a vision radius of 2.

Compute Environment

All experiments were conducted on a single machine running Linux. GAN training and evaluation used PyTorch with CUDA acceleration when available. A single GAN training run of 1,750 epochs completed in approximately 30–35 minutes on consumer hardware. DQN training for 2,000 episodes required a comparable amount of time. Evaluation time depends on the hardware and the number of methods compared.

Code Availability

The source code and implementation details are publicly available in the project's GitHub repository: *Probabilistic Reconstruction and Informed Search with Memory*. The repository contains the publication codebase for the GAN-guided world-modeling framework, including A* planning and DQN baseline implementations. Link: https://github.com/AadiJo/probabilistic-reconstruction-and-informed-search-with-memory.