Versatile Navigation under Partial Observability via Value-guided Diffusion Policy

Supplementary Material

6. Network Architectural and Experimental Specification

Expert trajectory padding. To ensure alignment among different trajectories within expert demonstrations for closed-loop plan generation, we employ a padding strategy. Specifically, for the action trajectory, $\boldsymbol{\tau}_a$, we append T_h steps of the termination action, *Done*, to its end, aligning with the prediction horizon defined in Sec. 3.2. This trajectory is conditioned on the partial environmental map $\boldsymbol{e}$, corresponding to the timestep of the sequence's initial action. Thus, we pad at the end to match the prediction horizon's length. This approach, also functioning as a form of data augmentation, effectively addresses the sparsity of the *Done* action in expert demonstrations, essential for training the policy to execute *Done* accurately and timely. These adjustments ensure consistent alignment across $\boldsymbol{\tau}_e$ and $\boldsymbol{\tau}_a$ in training data, with the same approach adopted during evaluation to condition the diffusion models properly.

Partial maps. Each partial map comprises three channels. The first marks observed obstacles as 1s while marking free spaces and unobserved areas as 0s. The second inversely marks observed free spaces as 1s and the others as 0s. The third marks the goal as 1 if having observed it; otherwise, the channel is all-zero. We follow Wilson's methodology [35] to generate these maps. The starting point and target are selected uniformly from free cells, always ensuring a clear path between them. Under partial observability, the initial map presents minimal environmental information akin to the agent's first observation. As the exploration proceeds, the agent uncovers more areas and incorporates them into the progressively more complete map.

Bit encoding. We can represent discrete data from an alphabet of size K variables with $n = \lceil \log_2 K \rceil$ bits, in the form of $\{0, 1\}^n$. This discretization drives previous work to remodel continuous diffusion models to accommodate discrete data and state spaces. Nonetheless, Chen et al. put forward an alternative strategy, where binary bits, $\{0, 1\}^n$, are converted to real numbers, $\mathbb{R}^n$, thus making them suitable for continuous diffusion models. They refer to these real numbers as "analog bits", which are processed as real numbers despite mimicking the dual nature of binary bits. For sample generation, the approach remains the same as in continuous diffusion models but incorporates an additional step of quantization at the end, through which the resultant analog bits are thresholded. Consequently, this transforms the analog bits back into binary bits, ready to be turned into their original discrete or categorical variables.

Plan generation module. Our plan generation module employs a diffusion model anchored on a U-Net architecture. Each section of the U-Net, including the downward, middle, and upward modules, contains residual blocks merged with one-dimensional convolution neural networks (CNNs). Tab. 4 showcases the specific layout of the plan generation module. The module takes in an expert action trajectory, with channels equal to the number of bits used to encode the action sequence (set to 4 for both domains) and length equal to T_h, serving as the input for the initial residual block. The hidden layers progressively widen the feature map channels to 256, 1024, and 2048 along the downsampling path. This map then navigates a bottleneck stage without altering its shape. In the upsampling path, the feature map is first concatenated with its counterpart stored during the downsampling path. Consequently, the input dimension for each stage of the upward module is effectively doubled. The final convolutional block of the upward module restores the output from a 256-channel feature map to a valid action trajectory. Note that no downsampling or upsampling is involved in the last stage of both paths.

An integral part of the plan generation module is an environment encoder, which converts a partial environment map at the current timestep, $\boldsymbol{e}_{(t)}$, into a low-dimensional embedding. This encoder comprises three convolutional blocks, succeeded by a global average pooling layer and two fully-connected layers. Each convolutional layer is configured with a kernel size of 3, a stride and padding size of 1, and output channels set to 128, 256, and 256, respectively. We apply group normalization, with eight groups, when conducting the convolutions. The global average pooling layer flattens the result into a one-dimensional vector of length 256. The two fully connected layers, with a dimension of 256×256, encode the features extracted from the map into an embedding as the condition for conditional diffusion. This embedding is subsequently incorporated into the one-dimensional diffusion model.

We leverage Feature-wise Linear Modulation (FiLM) to model the conditional distribution $p(\boldsymbol{\tau}_{a,(t)}|\boldsymbol{e}_{(t)})$. To this end, we set the output channels as twice the output of the residual block. This configuration allows us to treat half of the conditional embedding as scale and the other half as bias, facilitating a linear operation on the output of the first convolutional block within the residual block and then going through the rest. Adopting FiLM enables the diffusion policy to adjust its behavior based on specific features of the input partial environmental map, facilitating more effective learning of the action trajectory and consequently boosting its performance.

Value function. Within the value function module detailed

	Down	Middle	Up
Input	8×4	2×2048	2×4096
Stage 1	Conv1d: [3, 1, 256]×2 Res1d: [3, 1, 256]×1 Downsample: [3, 2, 1024]×1	Conv1d: [3, 1, 2048]×2 Res1d: [3, 1, 2048]×1	Conv1d: [3, 1, 1024]×2 Res1d: [3, 1, 1024]×1 Upsample: [4, 2, 1024]×1
Stage 2	Conv1d: [3, 1, 1024]×2 Res1d: [3, 1, 1024]×1 Downsample: [3, 2, 2048]×1	Conv1d: [3, 1, 2048]×2 Res1d: [3, 1, 2048]×1	Conv1d: [3, 1, 256]×2 Res1d: [3, 1, 256]×1 Upsample: [4, 2, 256]×1
Stage 3	Conv1d: [3, 1, 2048]×2 Res1d: [3, 1, 2048]×1		Conv1d: [3, 1, 256]×1 Conv1d: [1, 1, 4]×1
Output	2×2048	2×2048	8×4

Table 4. U-Net constitution of the plan generation module. Suppose the input action sequence length is 8 and has 4 channels. 4 is the number of bits used to encode each action. All our network's foundational components except upsampling are constructed using one-dimensional convolutional layers. Upsampling utilizes one-dimensional transposed convolutional layers. Within a convolutional layer, we designate the kernel size K, stride S, and target channels C in the form of $[K, S, C]$. Additionally, we specify the input and output channels corresponding to each convolutional layer, which denotes the number of filters used. Note that the final convolutional block is regular instead of residual.

Hyperparameter	Value
Dataset (vary with env size)	
Minimal episode length (15×15)	16
Maxmal path length (15×15)	50
Diffusion model	
Observation horizon (T_o)	2
Prediction horizon (T_h)	4
Execution horizon (T_a)	2
Timestep embedding dimension	256
Diffusion steps	32
EMA decay	0.995
Number of plan candidates	24
Value function	
Value iteration rounds (K)	60
Discount factor (γ)	0.99
Training	
Batch size	32
Gradient accumulation steps	2
Initial learning rate (diffusion policy)	2e-4
Initial Learning rate (value function)	0.005
Optimizer	RAdam
Total training epochs	60
Training steps per epoch	10000

Table 5. Hyperparameters for training the diffusion-based planner and the value function estimator, respectively.

in Sec. 3.3, the motion transition function $\hat{T}_m$, valid action reward function $\hat{R}_m$, and invalid action reward function $\hat{R}_f$ each consist of a convolutional kernel with A filters. Notably, $\hat{T}_m$ is initialized as a Gaussian distribution with a Softmax function applied over the height and width, i.e., the action space S, to ensure a valid probability distribution. To enhance learning flexibility and representation, $\hat{T}_m$ has independent weights for belief update and value iteration. Both $\hat{R}_m$ and $\hat{R}_f$ are initialized to zero, and dot products in Eq. (2) and Eq. (3) are implemented as convolutions using these kernels. The action embeddings $\hat{A}_{\text{logit}}$ and $\hat{A}_{\text{thresh}}$ are concurrently learned through CNNs. Specifically, the output action embedding comprises $(|A| + 1)$ channels, with the first $|A|$ channels serving as $\hat{A}_{\text{logit}}$ and the last one acting as $\hat{A}_{\text{thresh}}$. We train the model to minimize the discrepancy between $\hat{A}_{\text{logit}}$ and the expert demonstration, a^*, to reliably estimate the logarithmic probabilities associated with each action being taken.

Parallel plan candidate generation. One of our goals in adopting trajectory-level behavior synthesis for multi-step decision-making is to improve planning efficiency. However, introducing sampling variability, where the diffusion policy produces multiple plans for selection based on value function, ironically reintroduces inefficiency. This is because, at each planning step, the model must iteratively generate various plans. To address this, we employ multi-processing for plan generation, aligning the number of plans with the count of CPU cores. In our experiments, this number is set to 24 (refer to Tab. 5), effectively matching the time needed for generating multiple plans in a multi-processing setup to producing a single plan in a single-processing framework. This approach successfully preserves the high efficiency we initially sought.

Hyperparameters. We divide all the hyperparameters listed in Tab. 5 into four parts, each corresponding to expert demonstration and dataset, diffusion policy module, value function module, and general training configurations.

Transforming FPV RGB-D observations into 3D point clouds. Our framework for 3D navigation first processes each depth image by creating a mesh grid based on the camera's intrinsic parameters, representing the camera's 2D image space pixel coordinates. Integrating this mesh grid with depths facilitates a 2D-to-3D mapping for each pixel. By employing the depth values and the camera mesh grid,

the system calculates the 3D position of each pixel relative to the camera, a process known as unprojection. This method converts 2D pixel coordinates into 3D coordinates relative to the camera. Subsequently, these 3D coordinates are transformed into world coordinates using rotation and translation matrices, adjusting for the camera's position and orientation within the scene. This conversion process is systematically applied to each specified image. For every image processed, a 3D point cloud in world coordinates is generated and aggregated into a unified data structure. The culmination of this process is an extensive point cloud that encapsulates the entire scene as captured by the FPV images, where each point in the cloud corresponds to a pixel in the original depth images and is positioned according to its real-world location.

Generating BEV environmental maps from 3D point cloud. In Sec. 3.4, we detail our use of Swin3D for semantic segmentation of 3D point clouds. The pre-trained Swin3D classifies points in unseen scenes into 13 categories: ceiling, floor, wall, beam, column, window, door, table, chair, sofa, bookcase, board, and clutter. Points unassigned to the first 12 categories default to clutter. After segmenting the point cloud (represented as (X, Y, Z, R, G, B)) with Swin3D, each point's class prediction is appended to its original tuple, creating a seven-element tuple. These tuples are then projected onto the X-Z plane, aligning with the BEV perspective, as the Y-axis (height in FPV RGB-D input) is perpendicular to the BEV plane. The Z-axis, indicating depth in FPV images, corresponds to one of the BEV dimensions. In our experiment, we map these points onto a grid determined by a given map resolution of 50×50. For each grid cell, we classify it as an obstacle or free space based on the majority of points it contains—cells primarily with ceiling and floor points are marked as free space; others are deemed obstacles. This process yields a binary BEV map analogous to those in GridMaze2D, directly applying to our diffusion policy for planning.

7. Additional Qualitative Results

In Fig. 7, we present three cases of employing our method for navigation in GridMaze2D. The first two subfigures showcase successful navigation, where the agent following our policy explores the environment effectively, safely backtracks from dead ends, and finally reaches the target. Such cases account for the majority of all test cases. The third subfigure illustrates a failed scenario where the agent ends up in an indefinite loop. Unlike CALVIN's failure, where the agent repeats reverse actions consecutively and thus gets stuck in some small space, our policy leads to larger navigation path cycles, which are more unlikely to occur.

8. Additional Quantitative Results

We evaluate the robustness against sensor noise in AVD. Specifically, we add noise sampled from $\mathcal{N}(0, \mathbf{I})$ to depth

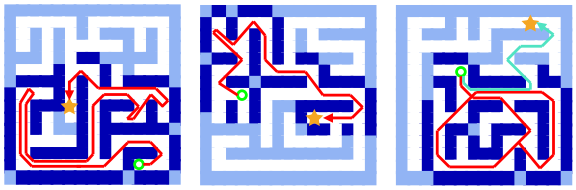

Figure 7. Qualitative results in GridMaze2D. The first two subfigures depict successful cases that use our approach. The last one illustrates a failed case, where the agent gets stuck in an indefinite loop. Refer to Fig. 6 for the denotation of markings.

images. Numerical results in Tab. 6 reveal a nuanced landscape of performance resilience. Both CALVIN variants exhibit a moderate decline in effectiveness under noisy conditions, with CALVIN-3D displaying slightly better resilience than its 2D counterpart, suggesting that 3D models possess inherent robustness to sensor inaccuracies in its trained 3D-to-2D projector. The zero-shot version of our approach stands out for its remarkable stability, experiencing negligible performance degradation despite the introduction of noise, underscoring its exceptional capability for zero-shot policy transfer in adverse environments. However, our policy retrained with RGB-D inputs consistently outperforms the others across a spectrum of scenarios and demonstrates extraordinary robustness. It suffers only minimal setbacks in the face of sensor noise. Thus, while all methods show some susceptibility to sensor noise, our approach emerges as the most robust, underlining the efficacy of incorporating additional sensory data through retraining in enhancing noise immunity. This finding could be pivotal for real-world applications where conditions are far from ideal.

Scene	CALVIN-2D	CALVIN-3D	Ours (Zero-shot)	Ours (Retrain)
Home_001_1	0.692±0.037	0.720±0.052	0.769±0.038	**0.776±0.028**
Home_001_1 (noisy)	0.674±0.048	0.714±0.059	0.767±0.043	**0.772±0.033**
Home_001_2	0.627±0.037	0.640±0.048	0.655±0.033	**0.732±0.030**
Home_001_2 (noisy)	0.604±0.042	0.638±0.058	0.652±0.036	**0.730±0.031**
Home_002_1	0.735±0.035	0.740±0.048	0.728±0.034	**0.755±0.027**
Home_002_1 (noisy)	0.696±0.045	0.735±0.050	0.727±0.033	**0.754±0.030**
Home_003_1	0.606±0.042	0.642±0.060	0.638±0.041	**0.686±0.031**
Home_003_1 (noisy)	0.613±0.050	0.636±0.059	0.634±0.043	**0.681±0.038**
Home_003_2	0.558±0.033	0.590±0.043	0.603±0.033	**0.622±0.030**
Home_003_2 (noisy)	0.552±0.036	0.589±0.048	0.602±0.034	**0.620±0.035**
Home_004_1	0.647±0.040	0.680±0.050	0.684±0.042	**0.695±0.036**
Home_004_1 (noisy)	0.634±0.043	0.674±0.057	0.681±0.042	**0.690±0.043**
Home_007_1	0.587±0.038	**0.610±0.045**	0.584±0.039	0.601±0.035
Home_007_2 (noisy)	0.580±0.041	**0.602±0.049**	0.584±0.040	0.598±0.038
Home_010_1	0.728±0.033	0.736±0.043	0.769±0.032	**0.781±0.028**
Home_010_1 (noisy)	0.717±0.030	0.731±0.046	0.766±0.033	**0.780±0.032**
Mean succ. rate	0.635±0.032	0.682±0.047	0.679±0.040	**0.706±0.032**
Mean succ. rate (noisy)	0.626±0.037	0.670±0.052	0.675±0.042	**0.700±0.040**

Table 6. Performance of CALVIN and our method in AVD's embodied navigation and object searching tasks, where the goal is to locate a Coca-Cola glass bottle in an indoor scene. It presents the agent's success rates across various scenes. Our method, which achieves comparable results to CALVIN in zero-shot policy transfer from the 2D domain, surpasses CALVIN in scenarios retrained with additional RGB inputs, with an exception in one scene.

9. Limitations and Future Work

This work has two main limitations—high reliance on precise point cloud semantic segmentation for the 3D domain and potential suboptimal long-term decision-making embedded in the value guidance due to QMDP's strong assumption of full observability in future timesteps.

Reliance on the performance of segmentation model. For effective semantic segmentation of point clouds, it is imperative that the chosen model reliably identifies key objects like floors, ceilings, and walls across diverse indoor settings. Mislabeling can lead to incorrect BEV map projections and cause catastrophic planning errors. Continuously improving the segmentation model's quality is essential to ensure accuracy. Thus, future work may involve fine-tuning the model on a small, labeled dataset from the target domain, if available, to boost performance significantly. Alternatively, for domains where labeled data is unavailable, unsupervised adaptation methods like adversarial training or self-ensembling could be employed. However, developing an entirely new semantic segmentation framework specifically for point cloud data is beyond the scope of our discussion, which focuses on decision-making in navigation planning.

Potential suboptimal long-term planning. As we introduce in Sec. 2, the QMDP heuristic simplifies the value iteration of POMDP by taking into account partial observability only at the current timestep but assuming full observability on subsequent timesteps. Doing so can make optimal decisions in the immediate sense but be suboptimal when considering a longer horizon. In our work, unlike autoregressive approaches, we need not only one optimal step of action but a sequence of steps that accounts for the predicted action trajectory, which might amplify the impact of suboptimality of QMDP in farther timesteps. To address this issue, we can consider point-wise value iteration in future work.