Appendices

This document provides more details of our method and experimental details, which are organized as follows:

- *Appendix A: Implementation details of our model;*
- *Appendix B: Real world deployment.*

A. More Implementation Details of Our Model

In this section, we provide more details of our model. Specifically, we discuss how an autoencoder is employed to significantly reduce computational costs and GPU memory consumption. Additionally, we provide comprehensive details regarding data collection, model specifications, and training procedures for both our Implicit Partial Completion(IPC) module and the BEV-based Waypoint Predictor.

A.1. AutoEncoder for Computation Reduction

We introduce an autoencoder to reduce feature dimension. The encoder part is used followed by the MobileSAM[7] to reduce channel size from 256 to 32. The decoder part is used before the Implicit Partial Completion(IPC) Module to recover the feature dimension back to 256. The primary advantages of this autoencoder lie in its ability to reduce feature dimensionality, lower computational costs, and decrease memory consumption.

The autoencoder architecture is based on a convolutional neural network (CNN). The encoder consists of two convolutional layers, each employing the GELU activation function. Both convolutional layers use a kernel size of 3 with padding set to 1. The first layer increases the channel size from 256 to 512, while the second reduces it to 32. The decoder mirrors this structure, initially expanding the channel size from 32 to 512 and then reducing it back to 256 to reconstruct the original feature dimensions.

This autoencoder is trained using images sourced from the MP3D[1] and HM3D[4] simulators. Images are rendered at specific points within these environments. For MP3D, these points are derived from the connectivity graph provided by VLN-CE [3], with 12 views captured per point at 30-degree intervals. To ensure there is no data leakage, only images from the training split are utilized for training purposes. For HM3D, where no such graph exists, navigable points are randomly sampled using API provided by Habitat[5]. Points closer than 3 meters are filtered out to ensure diversity in the dataset. A maximum of 10 points are sampled per scan in the HM3D training split. Ultimately, a total of 320,000 images are randomly selected to form the training set, complemented by an additional 32,000 images collected similarly from the MP3D validation unseen scans for evaluation purposes.

Training proceeded as follows: for each batch, feature map m are extracted using MobileSAM[7], followed by processing through the encoder and decoder to reconstruct the feature map. We denote the reconstructed feature map as $\hat{m}$. During this process, MobileSAM remains fixed while the autoencoder is trained. The objective function combined cosine similarity maximization with L1 and L2 loss minimization:

$$\mathcal{L}_{ae} = 1 - \sum_{i,j} \frac{\hat{m}_{i,j} \cdot m_{i,j}}{\|\hat{m}_{i,j}\| \|m_{i,j}\|} + \|\hat{m} - m\|_2^2 + \|\hat{m} - m\|_1 \quad (1)$$

The training process uses a batch size of 32 and a learning rate of 0.0001, following a cosine scheduler for training 10,000 iterations.

A.2. Implicit Partial Completion Module Details

A.2.1. Data Collection

We detail the process of collecting datasets for training the IPC module. To gather partially incomplete feature maps, we conduct random navigation in both MP3D and HM3D environments by following specific trajectories. For MP3D, navigation is performed along trajectories provided by R2R-CE. In the case of HM3D, where no predefined trajectories are available, we first randomly sample up to 20 points using the Habitat API, ensuring that each point is at least 3 meters apart to maintain dataset diversity. These sampled points serve as starting points, while the farthest corresponding points within the environment are designated as endpoints. For each scan, we collect up to 20 trajectories, navigating from the starting points to their respective endpoints within the HM3D environment.

Prior to navigation, the agent performs a 360-degree rotation to capture surrounding information. During navigation, we construct the feature field and generate corresponding incomplete panoramic feature maps. Simultaneously, we collect complete panoramic images to provide supervision for the IPC module. Each panoramic image comprises 12 monocular images. As a result, we obtain 12 samples at each step, with each sample consisting of a complete monocular image paired with its corresponding incomplete feature map. From the training scans of MP3D and HM3D, we collected 600,408 training samples, and from the MP3D validation scans, we gathered 42,444 evaluation samples.

During the training phase, we extracted feature maps from complete images to establish ground truth for training the completion module. The incomplete feature maps were processed through the IPC module to reconstruct the ground truth. This procedure is elaborated in the main body. Additionally, for CLIP feature alignment, we extracted CLIP features from the images and computed the associated loss.

A.2.2. Model Details

As illustrated in the main body, the IPC module take render feature maps as input and extract features for then. Besides, an decoder are used to reconstruct complete feature map according to extracted features. The IPC module is a 4 layer

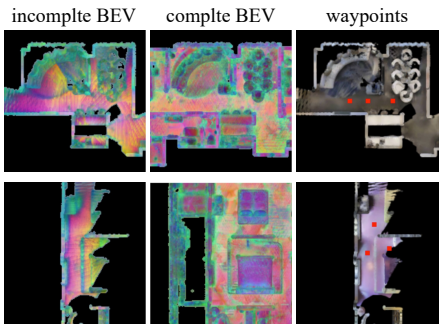

Figure 1. Samples for training our BEV-based waypoint predictor.

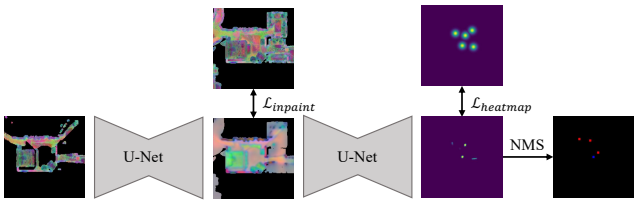

Figure 2. The pipeline of waypoint predictor. The input BEV map is first fed input a UNet for inpainting, then forward into another UNet to predict a heatmap. After NMS on the heatmap, waypoints are obtained(red points).

transformer. Hidden dimension is 512 and head number is 8. The input feature map size is 32× 32 with channel size 256. We patchify it with patch size 2, yielding 256 tokens. Together with a learnable [CLS] token, they are input to the view encoder. To align the [CLS] token with CLIP feature, we use a linear layer convert 512 dimension to 768 for the loss calculation.

The decoder follows the same architecture. The only difference is the inclusion of an additional linear layer that increases the token dimension from 512 to 1024. This layer helps in recovering these tokens back to the shape of the input feature map for loss computation.

A.3. Waypoint Predictor

A.3.1. Data Collection

We collect data from MP3D simulator by navigating in MP3D. Specifically, we extracted the start and end points of from trajectories in R2R-CE training set and navigated between them using the shortest path computed on a high-quality topological connectivity graph provided in [2]. This graph ensures precise point-to-point navigation while enabling us to gather essential training data, such as complete and incomplete top-down view feature maps generated through feature rendering, coordinates of nearby waypoints derived from the topological graph, and other relevant information.

Since a complete bird's-eye-view(BEV) map is required for training, we first reconstructed the entire feature field for each scan before collecting data for waypoint predictor. During navigation, we rendered the complete BEV maps based on this whole feature field. Through this process, we collect 19,273 training samples from the MP3D training scans and 3,531 evaluation samples from the validation unseen scans. Some samples are shown in Figure 1.

A.3.2. Model Details

We present the architecture of our waypoint predictor, which adopts a cascaded UNet framework following the 3DFF approach [6]. As illustrated in Fig. 2, the system comprises two sequential UNets: the first UNet performs inpainting tasks, while the second generates a heatmap that indicates the probability distribution of potential waypoints. The input to this architecture is a BEV (Bird's Eye View) map rendered from the feature field. Size of the BEV map is 192×192, with each grid of size $5cm \times 5cm$.

The processing pipeline begins with the BEV map being fed into the first UNet for inpainting, which produces an enhanced feature map $\hat{x}$. This inpainted feature map is subsequently processed by the second UNet to generate a probabilistic heatmap $\hat{h}$, representing potential navigable locations. The inpainting ground truth is obtained from the complete BEV map, as detailed in Section A.3.1. For the heatmap generation, the ground truth is formulated as a combined Gaussian distribution centered on the actual waypoints, with a standard deviation $\sigma = 5$, consistent with the 3DFF methodology.

To extract discrete waypoints from the predicted heatmap, we employ non-maximum suppression as 3DFF, with a threshold of 0.01, limiting the maximum number of waypoints to 10. These selected waypoints are then transformed into world coordinates for subsequent navigation tasks.

A.3.3. Training

We train the waypoint predictor by combining two loss functions: $\mathcal{L}_{inpaint}$ and $\mathcal{L}_{heatmap}$. The total loss function is defined as:

$$\mathcal{L} = \mathcal{L}_{inpaint} + \mathcal{L}_{heatmap} \tag{2}$$

The inpainting loss, $\mathcal{L}_{inpaint}$, is formulated as:

$$\mathcal{L}_{inpaint} = 1 - \sum_{i,j} \frac{\hat{x}_{i,j} \cdot x_{i,j}}{\|\hat{x}_{i,j}\| \cdot \|x_{i,j}\|} + \|\hat{x} - x\|_2^2 + \|\hat{x} - x\|_1 \tag{3}$$

Here, $\hat{x}_{i,j}$ represents the inpainted feature, and $x_{i,j}$ denotes the ground truth feature at pixel location (i, j). This loss incorporates both cosine similarity and L1/L2 distance to ensure robust predictions.

The heatmap loss, $\mathcal{L}_{heatmap}$, addresses the class imbalance issue in heatmap prediction. It is the loss function pro-

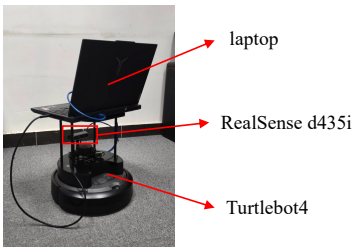

Figure 3. Hardware platform.

posed in CenterNet [8] and is defined as:

$$\mathcal{L}_{heatmap} = \begin{cases} -(1-\hat{h}_{i,j})^2 \log(\hat{h}_{i,j}) & \text{if } h_{i,j} = 1 \\ -(1-h_{i,j})^4 \hat{h}_{i,j}^2 \log(1-\hat{h}_{i,j}) & \text{otherwise.} \end{cases} \tag{4}$$

In this formulation, $\hat{h}_{i,j}$ is the predicted heatmap value, and $h_{i,j}$ is the corresponding ground truth. The loss penalizes false negatives more heavily when $h_{i,j} = 1$, while reducing the penalty for false positives when $h_{i,j} \neq 1$.

For training, we set the batch size to 8 and use a learning rate of 0.0001 together with cosine scheduler. The model is trained for 10,000 iterations to optimize the combined loss function. We assess the model's performance using a metric that calculates the sum of the minimum distances between the predicted waypoints and the ground truth waypoints, as well as the minimum distances from the ground truth waypoints to the predicted waypoints. This bidirectional evaluation ensures recall and precision of predicted waypoints. During the training process, we evaluate the model every 1000 iterations. We select the best model according to this metric for navigation.

B. Real World Deployment

B.1. Deploy Details

We provide details on deploying our model in a real-world robotic setup. Our platform consists of a Turtlebot4 equipped with a RealSense D435i camera for capturing monocular RGB-D observations, as shown in Figure 3. The Turtlebot4 is controlled using ROS2 humble, and we obtain the robot's pose from its own odom data. The entire model runs on a laptop equipped with an RTX 4060 GPU, which is placed on the robot. The RealSense camera is directly connected to this laptop, the captured RGB image and depth image are resized to 224×224 to match the common image size.

To facilitate the adaptation of simulation code for use with a real-world robot while minimizing code modifications, we encapsulate the robot's control functions within an API that closely mirrors the simulator's interface. Given that the RealSense d435i camera on the Turtlebot4 has a narrower field of view(FOV) and is mounted at a lower height compared to its configuration in the simulator, we trained a new model tailored to these specific conditions.

Methods	initial rotation	NE↓	OSR↑	SR↑	SPL↑	num steps↓
ours		5.43	56.5	48.7	35.4	153
ours(fix bias)		4.10	63.0	54.4	45.9	99
ours	✓	4.61	62.4	54.8	44.4	128
ours(fix bias)	✓	3.48	67.7	60.7	52.9	101

Table 1. Impact of the dataset bias.

B.2. Dataset Bias

During the deployment phase, we encounter an issue where the model is likely to turn back at the first step. This problem is traced back to dataset bias of R2R-CE, specifically concerning the initial orientation at the start of each episode, they are random and do not align well with the navigation instructions provided. To mitigate this, we adjust the initial heading to align it with the ground truth trajectory. As shown in Table 1, this adjustment significantly improves model performance. After fixing the bias, performance is increased by approximately 6% both with and without initial rotation. These results indicate that current VLN models are significantly constrained by dataset bias. Addressing such issues can lead to notable improvements in model performance.

References

[1] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D data in indoor environments. *3DV*, 2017. 1

[2] Yicong Hong, Zun Wang, Qi Wu, and Stephen Gould. Bridging the gap between learning in discrete and continuous environments for vision-and-language navigation. In *CVPR*, 2022. 2

[3] Jacob Krantz, Erik Wijmans, Arjun Majumdar, Dhruv Batra, and Stefan Lee. Beyond the nav-graph: Vision-and-language navigation in continuous environments. In *ECCV*, 2020. 1

[4] Santhosh Kumar Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alexander Clegg, John M. Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X. Chang, Manolis Savva, Yili Zhao, and Dhruv Batra. Habitat-matterport 3d dataset (HM3D): 1000 large-scale 3d environments for embodied AI. In *NeurIPS*, 2021. 1

[5] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A Platform for Embodied AI Research. In *ICCV*, 2019. 1

[6] Zihan Wang, Xiangyang Li, Jiahao Yang, Yeqi Liu, and Shuqiang Jiang. Sim-to-real transfer via 3d feature fields for vision-and-language navigation. In *CoRL*, 2024. 2

[7] Chaoning Zhang, Dongshen Han, Yu Qiao, Jung Uk Kim, Sung-Ho Bae, Seungkyu Lee, and Choong Seon Hong. Faster segment anything: Towards lightweight sam for mobile applications. *arXiv preprint arXiv:2306.14289*, 2023. 1

[8] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *CoRR*, 2019. 3