Department of Computer Science

BSCCS Final Year Project Report
2024-2025

24CS053

Embodied AI for Navigation with Quadruped Robots

(Volume 1 of 1)

Student Name : **WONG Lik Hang Kenny**

Student No. : **57166465**

Programme Code : **BSCEGU4**

For Official Use Only

Supervisor : **Prof HOU, Junhui David**

1st Reader :

2nd Reader :

Student Final Year Project Declaration

I have read the project guidelines and I understand the meaning of academic dishonesty, in particular plagiarism and collusion. I hereby declare that the work I submitted for my final year project, entitled:

Embodied AI for Navigation with Quadruped Robots

does not involve academic dishonesty. I give permission for my final year project work to be electronically scanned and if found to involve academic dishonesty, I am aware of the consequences as stated in the Project Guidelines.

Student Name: WONG Lik Hang Kenny Signature: 

Student ID: 57166465 Date: February 27, 2025

1 Abstract

Embodied Artificial Intelligence (Embodied AI) enables robots to perceive and interact with their environments, with Vision-Language Navigation (VLN) as one of the most critical downstream tasks where robots are required to follow language instructions to reach target locations. Current VLN research mainly uses physically simplified simulators and low-mobility wheeled robots. These approaches fail to match the complex real-world conditions such as uneven terrains. This project aims to advance VLN by introducing **Recurrent-VLN-Bert-Isaac**, a novel imitation learning-based model architecture tailored for quadruped robots, and the **VLN-Go2-Matterport** dataset, a novel VLN dataset for navigation in high-fidelity indoor environments. This dataset enhances accessibility for researchers in Computer Vision and Natural Language Processing, and to bridge the gap between research and real-world deployment. Our promising experimental results demonstrate that the model effectively learns navigation policies from the dataset that are transferable to simulators. Additionally, We also explored LLM-based approaches, detailed in the Appendix. Together, this work contributes important and meaningful tools and insights to Embodied AI.

Code: https://github.com/Kenn3o3/FYP-Navigator (Actively Maintained)

2 Acknowledgments

I would like to express my sincere gratitude to everyone who has supported this project.

I am deeply thankful to my supervisor, Prof. Hou, Junhui David, for his unwavering support. He provided me with the resources and freedom to explore this topic in depth, and his encouragement and valuable feedback through our email correspondences were very important in shaping this work. This project could not have been successfully initiated without his guidance and mentorship.

Contents

3 Introduction

3.1 Embodied AI

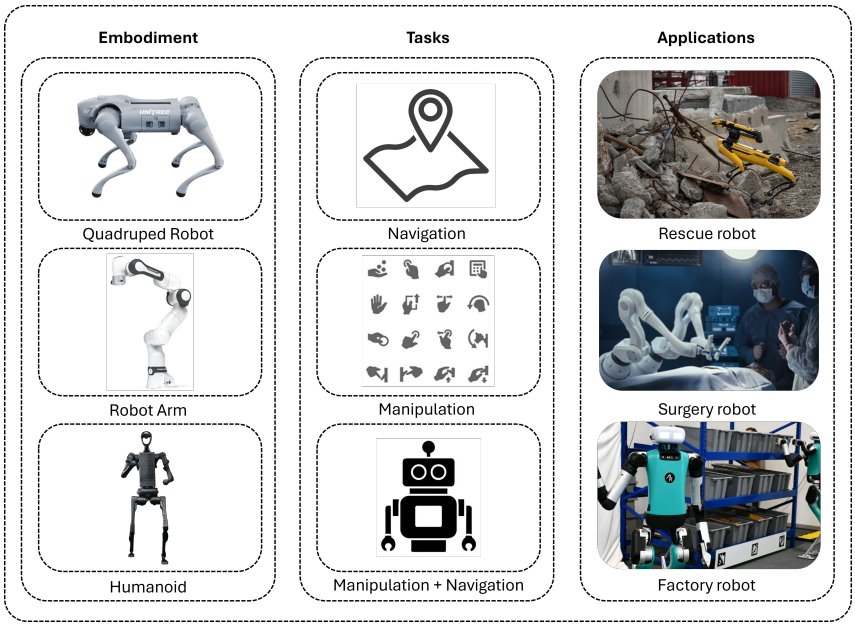

Figure 3.1: Overview of Embodied AI: Embodiment Types, Associated Tasks, and Real-World Applications

Imagine you are in a new house, and you are asked to find the oven. Most of us would explore the house and navigate to the kitchen, where ovens are often located. We can easily perform this task using common sense. Robots, on the other hand, do not have common sense. Instead, they perceive the their environment through numerical data—for example, images from onboard cameras represented as 2D matrices or point clouds from lidar scanners expressed as sets of coordinate points. Embodied Artificial Intelligent(EAI) Agent aims to interpret these perception into environmental understanding, and then embody the understanding on robot agents to perform actions to accomplish specific tasks. As illustrated in Figure 3.1, different types of embodiment allow robots to tackle various tasks and support a range of real-world applications.

3.2 Existing Problems and Project goal

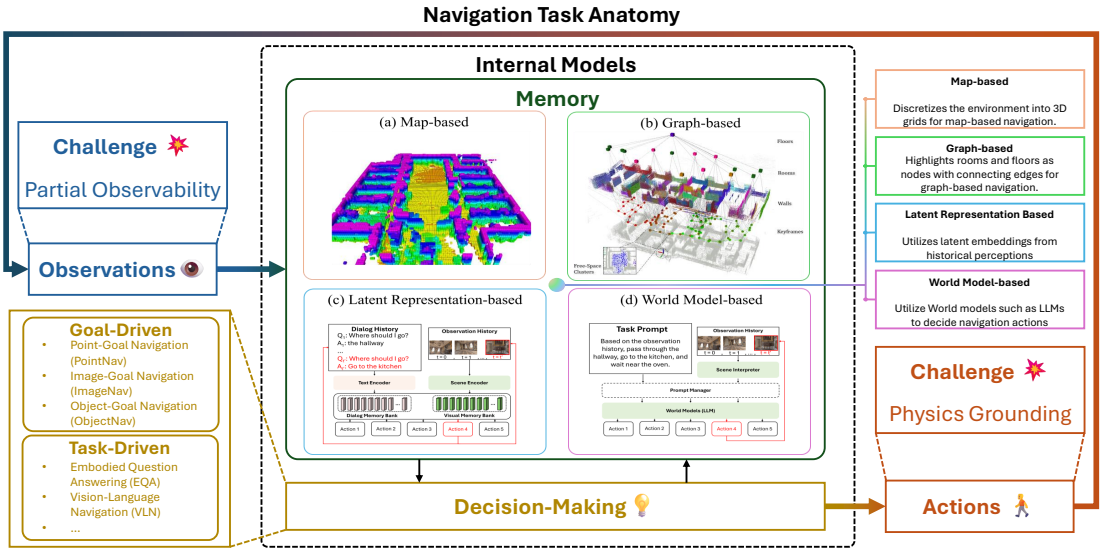

Figure 3.2: Overview of Navigation Task.

This project focuses on studying Vision-Language Navigation (VLN) algorithms within the context of Embodied AI. There are two reasons for selecting this task. First, VLN covers core components of Embodied AI, including perception, memory, planning, and multi-modality reasoning, as shown in Figure 3.2, making it an ideal task for exploring the foundational elements of this field. Second, VLN stands out as one of the most powerful and important downstream tasks in EAI. A VLN agent can be deployed in diverse real-world applications requiring navigating and commands ordering, such as autonomous vehicles [1], personal assistants [2], rescue robots [3], and beyond.

With the rise of foundation models, ongoing research explores training or fine-tuning large foundation models for navigation tasks [4, 5]. However, many of these approaches test foundation models on low mobility (low degree-of-freedom, or DoF) wheeled robots within simulator platforms such as Habitat [6], Matterport3D Simulator [7] and AI2THOR [8], which often simplify physics to speed up training. These methods, while efficient, overlook real-world physics and complex terrains. To develop a general embodied agent capable of working in complex real-world environments, it is essential to evaluate its per-

formance on agile robots—such as quadrupeds—in high-fidelity simulators with realistic physics, like Nvidia Isaac Sim [9]. To this end, we propose a novel architectures for a quadruped VLN agent on the Isaac Sim platform: an Imitation Learning based architecture called Recurrent-VLN-Bert-Isaac.

Concurrently, there is growing interest among researchers from Computer Vision (CV) and Natural Language Processing (NLP) in studying VLN. However, our extensive research reveals that existing resources are not easily accessible or well-suited for newcomers to the field. Thus, a key mission of this project is to bridge the gap between Embodied AI and CV research by introducing a novel dataset called VLN-Go2-Matterport. These contributions aim to facilitate easier entry into VLN and Embodied AI research for both robotics and CV researchers.

Our Contributions: First, we formulated Vision-Language Navigation (VLN) into a Computer Vision-like problem and released the VLN-Go2-Matterport dataset to the public together with a customizable collection pipeline. **Second**, we propose a novel latent representation-based architecture called Recurrent-VLN-Bert-Isaac for the problem. **Lastly**, we even conducted experiments on an LLM-based architecture for VLN, analyzing its performance and sharing our insights into the future directions.

4 Literature Review

4.1 Task-Driven Navigation Tasks

Task-driven navigation tasks typically require the robotic agent to understand language questions (e.g., Is there a water bottle on my desk in my Bedroom) or language instruction (e.g., go to my bedroom) and navigate to a place (e.g., Bedroom) to accomplish an instruction (e.g., successfully navigate to the bedroom), or provide the solution to the task (e.g., answer "yes"). They can be mainly categorized into language Instruction-based and question-answering-based.

4.1.1 Language Instruction Based Navigation

Vision-and-Language Navigation (VLN), along with the Room-to-Room (R2R) benchmark proposed in the same paper [10], is the first task requiring a robotic agent to navigate in an unknown environment by following linguistic navigation instructions, figure 4.1(b) shows different types of instructions defined by the paper. Extended from this task, subsequent tasks, including Cooperative Vision-and-Dialog Navigation (CVDN) [11], are proposed to tackle language ambiguity issues. Specifically, CVDN tasks allow the navigator to ask the oracle (humans) for clues during navigation. However, both VLN and CVDN chose wheeled robots as the navigation agents and evaluated results on a non-physic-based simulator.

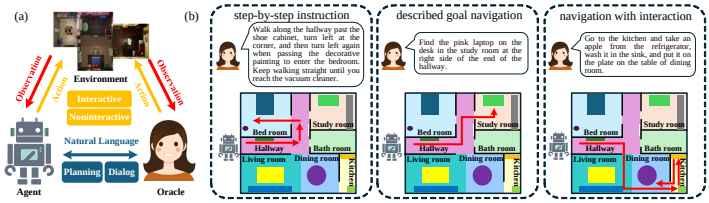

Figure 4.1: Figure from [12] showing an overview of Vision Language Navigation. (a) Interaction between agent, human and environment. (b) Types of language instructions.

4.1.2 Question Answering Based Navigation Tasks

Unlike VLN, the Embodied Question Answering (EQA) task [13] does not directly require a navigation agent to follow a linguistic instruction. Instead, a question is provided, and the agent should navigate around the environment and find clues to answer the question. Different types of questions are illustrated in figure 4.2. This task can better evaluate the language understanding ability of embodied agents and thus has gained significant research attention since the breakthrough of LLMs; for example, NaviLLM [4] achieved state-of-the-art on the CVDN benchmark by selecting Vicuna-7B-v0 [14], a GPT-4 [15] based instruction-tuned model as the base model to infer exploration actions.

Figure 4.2: Figure from [12] showing an overview of Embodied Question Answering. The blocks at the bottom show different types of questions that can be asked.

4.2 LLM/VLM-Driven Navigation Methods

Numerous methods use LLMs or VLMs to predict navigation actions by reasoning over textual or multimodal representations of the environment. They internalize environmental knowledge through pretraining on internet-scale data and infer actions via in-context learning. Figure 4.3 shows an overview of the architecture of an LLM-based navigation system.

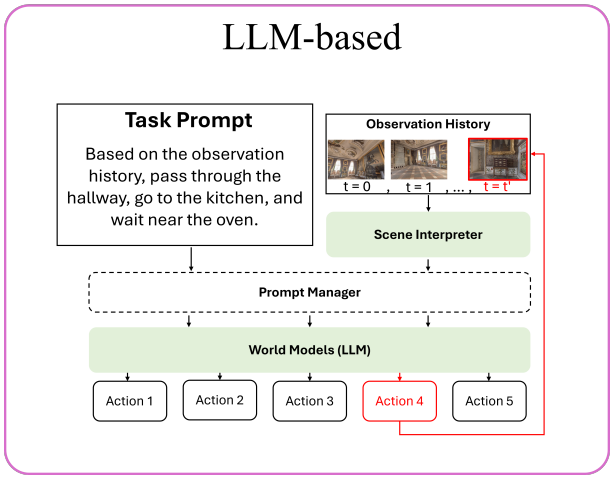

Figure 4.3: Figure showing an overview of an LLM/VLM-based navigation system.

For example, NavGPT [16] converts historical observation during navigation and instructions into a text prompt, enabling an LLM to autoregressively predict the next action. Similarly, Zheng et al. [17] fuse visual features with textual prompts (e.g., task instructions) in a VLM to generate actions for tasks such as vision-language navigation (VLN) and Embodied Question Answering (EQA).

These approaches avoid explicit spatial mapping by maintaining state information in textual context tokens or hidden embeddings. Partial observability is mitigated by dynamically updating prompts with new observations (e.g., camera frames), while unobserved environmental aspects (e.g., occluded rooms) are inferred through the pre-trained world knowledge and commonsense reasoning of LLMs and VLMs.

4.3 Latent Representation-based Navigation Methods

In dynamic environments, Implicit latent memory offers an efficient solution by encoding historical observations in compact and latent representations. Figure 4.4 shows an overview architecture of such a system.

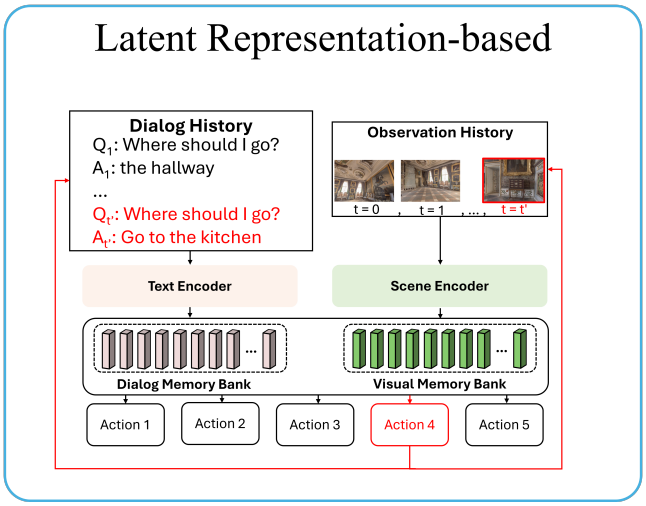

Figure 4.4: Figure showing an overview of a Latent representation-based navigation system.

For instance, Zhu et al. [18] learn a joint visual-textual embedding for dialog-based navigation. By applying cross-modal attention to both image features and language inputs and processing the features through a Long Short-Term Memory(LSTM) module [19], their system infers navigation actions directly from historical perceptual and textual cues rather than relying on an explicit map. Furthermore, Bono et al. [20] train an agent to learn a task-agnostic, actionable representation of scenes. Similarly, Moghaddam et al. [21] introduce the Foresight Imagination (ForeSIT) module, which enables an agent to imagine a latent representation of a future state that leads to success. By conditioning the policy on this imagined future state during training, the agent learns to navigate more effectively toward the goals.

These methods rely on recurrent or transformer-based networks to encode sequences of inputs into latent states, handling partial observability in a compact form. However, as we dive deep into the experiments, challenges arise when long-term consistency is required.

5 VLN-Go2-Matterport Dataset

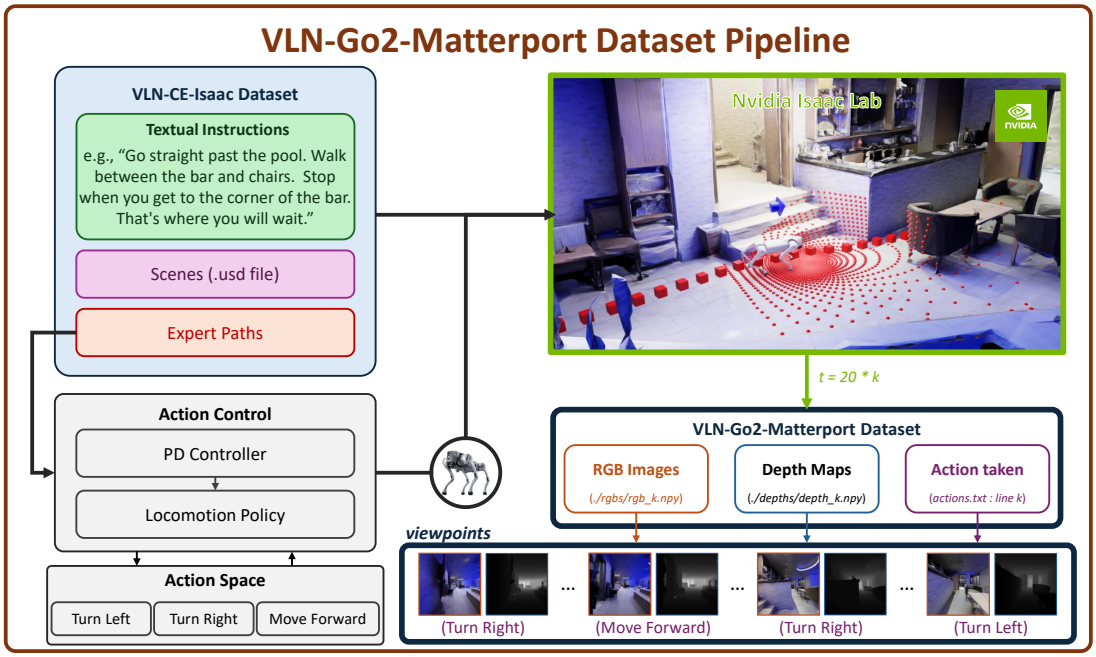

Figure 5.1: Figure showing the collection pipeline for the VLN-Go2-Matterport Dataset

Existing VLN datasets can be broadly classified into two types—image-based and 3D-based. **Image-based** datasets represent environments as topological graphs of pre-captured images. They restrict agents to perform translative actions (e.g., turn left, turn right, move forward) and perceive fixed viewpoints. Due to the discreteness in scenes and actions, they lack adjustability for embodied agents like quadruped robots, which require continuous motion and camera perspectives (e.g., tilted views) that reflect their limb-based dynamics. Therefore, these datasets are typically used only for learning high-level planning and are infeasible for training end-to-end policies for agile robots.

On the other hand, **3D-based** datasets provide continuous navigation environments in simulators, but training end-to-end policies that account for environmental dynamics is still computationally impractical. To address these limitations, we propose the **VLN-Go2-Matterport dataset**, which discretizes continuous environments while capturing

visual observations from a quadruped robot's perspective, offering a feasible platform for training end-to-end imitation learning based VLN policies for agile robots in a continuous environment. The dataset collection pipeline is illustrated in figure 5.1 and will be detailed in the next section.

5.1 Dataset Description

The VLN-Go2-Matterport dataset is designed for training and evaluating models on Vision Language Navigation (VLN) tasks involving quadruped robots. It collects actions and perspectives of an expert **Unitree Go2 robot** when it navigates indoor environments reconstructed from the Matterport dataset [7], rendered in Isaac Lab. Each episode in the dataset corresponds to a VLN navigation task where the robot follows an expert path, capturing visual observations and discrete actions. A pre-trained low-level policy manages the locomotion part of the expert robot, while high-level navigation decisions are recorded as discrete actions, making the dataset suitable for end-to-end imitation learning in VLN. Specifically, each episode contains:

- A natural language instruction (e.g., "Go to the kitchen and find the red chair"), sourced from the VLN-CE-Isaac benchmark.
- A sequence of RGB images captured from the Go2's camera.
- Corresponding depth maps, with the properties shown in table 5.1.
- Discrete actions taken by the Go2 ("Move forward", "Turn left", "Turn right").

Table 5.1: Summary statistics for depth maps. All values are in meters.

Statistic	Value
Minimum	0.04
Maximum	39.36
Mean	2.02
Standard Deviation	1.68

5.2 Collection Pipeline

Figure 5.1 illustrates the collection pipeline. The dataset is collected via Isaac Lab, using environments from the VLN-CE-Isaac benchmark dataset [22]. A Proportional-Derivative (PD) controller guides the Unitree Go2 robot along expert paths while a pre-trained low-level policy executes velocity commands. The process is outlined below:

1. **PD Controller:**

   - Selects discrete actions to follow the expert path.
   - For a robot at position $\mathbf{r} = [x_r, y_r, z_r]$ with orientation quaternion $\mathbf{q} = [q_w, q_x, q_y, q_z]$, and a target point $\mathbf{t} = [x_t, y_t, z_t]$ ahead on the expert path (look-ahead distance = 2 points), compute the relative position in the world frame: $\Delta\mathbf{p} = \mathbf{t} - \mathbf{r}$.
   - Convert the quaternion $\mathbf{q}$ to a rotation matrix $\mathbf{R}$ (Mathematical details of how it is done are listed in Appendix 9.1).
   - Transform to the robot's body frame using the rotation matrix $\mathbf{R}$: $\Delta\mathbf{p}_{\text{body}} = \mathbf{R}^T\Delta\mathbf{p}$.
   - Calculate the desired yaw: $\theta_d = \tan^{-1}\left(\frac{\Delta\mathbf{p}_{\text{body}}[1]}{\Delta\mathbf{p}_{\text{body}}[0]}\right)$.
   - Compute the desired yaw rate: $\omega_d = K_p \cdot \theta_d$, where $K_p = \frac{0.5}{\pi}$.
   - Select an action based on ω_d:
     - If $|\omega_d| \leq w_{\text{threshold}}$ (default 0.05 rad/s): "Move forward".
     - If $\omega_d > w_{\text{threshold}}$: "Turn left".
     - If $\omega_d < -w_{\text{threshold}}$: "Turn right".

2. **Action Execution:**

   - Map actions to velocity commands: "Move forward" $\rightarrow (0.6, 0, 0)$, "Turn left" $\rightarrow (0, 0, 0.5)$, "Turn right" $\rightarrow (0, 0, -0.5)$ (linear x, y, angular z in m/s or rad/s). Use a pre-trained locomotion policy to make Go2 follow the velocity commands.

- Repeat each action for 20 simulation steps, executed by the pre-trained low-level locomotion policy.

3. **Data Collection:**

   - Every 20 simulation steps, collect:
     - RGB image from the camera (saved as PNG).
     - Depth map (saved as NumPy array).
     - The discrete action taken.

4. **Episode Termination:**

   - End the episode when the robot is within 1 meter of the goal (Euclidean distance in the x-y plane) or after 3200 steps.
   - Save data only for successful episodes to ensure valid navigation trajectories.

869 episodes are collected from this pipeline for this project.

6 Methods

IMPORTANT: The entire navigation system is complex, and showing how to set it up is not the primary focus of this project. Therefore, we have released the full code on GitHub, with detailed instructions to guide users in configuring the system and environment for their research, and replicating the training and testing processes. The repository will be continuously maintained and improved, ensuring it remains a valuable community resource beyond this project.

6.1 Problem Setup and Notations

This work develops a quadruped robot agent to address the Vision-Language Navigation (VLN) task. In each navigation episode, the agent receives a language instruction I specifying a target location. At every simulation time step $t = 0, 1, 2, \ldots$, the agent observes an RGB image O_t from its onboard camera and executes an action $a_t \in \mathcal{A}$, where $\mathcal{A} =$ "Move Forward","Turn Left","Turn Right". The objective is for the sequence of actions to guide the agent to a position sufficiently close to the target location, at which point the episode is deemed successful. Examples of language instructions I include:

- "Walk across the room and wait at the far end of the bar."
- "Walk out of the bedroom through the open door into the hallway. Turn the corner and walk into the dining area. Pass the dining table and walk into the living room area towards the television. Stop near the chair and open sliding doors to outside."
- "Go out of the room you're in. Turn left. Go past the dining room and into the living room. Wait right by the coffee table."

6.2 Recurrent-VLN-BERT-Isaac

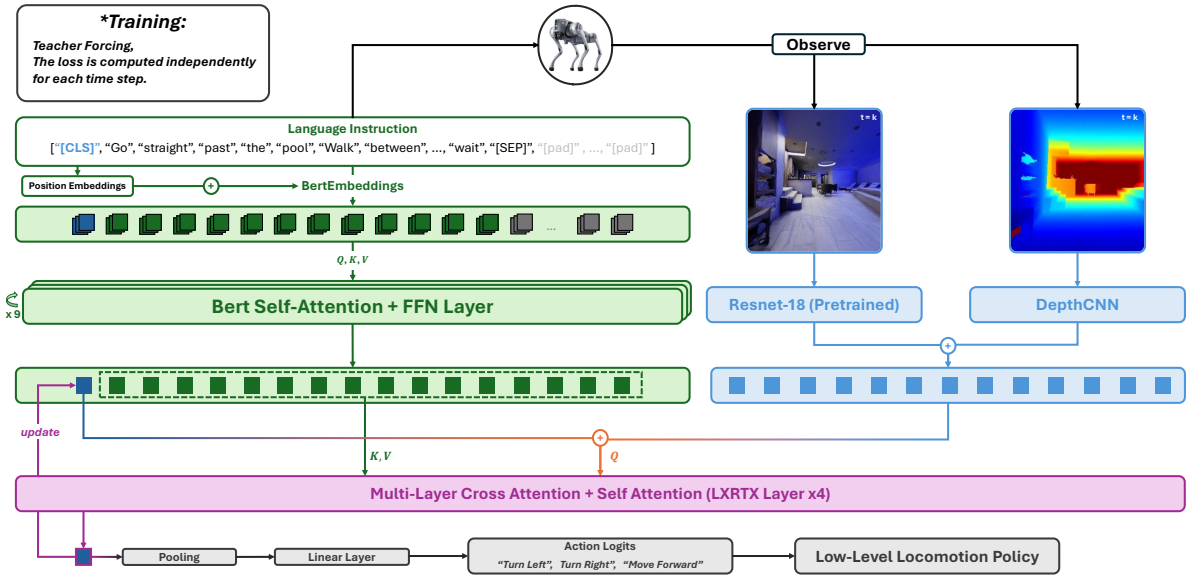

Figure 6.1: Architecture of Recurrent-VLN-BERT-Isaac

The Recurrent-VLN-BERT-Isaac model is inspired by the Recurrent-VLN-BERT [23] and PREVALENT [24] frameworks, originally developed for Vision-and-Language Navigation (VLN) tasks with reinforcement learning in image-based matterport environments [25]. We used a similar architecture but for imitation learning, where the agent learns to predict expert actions directly from RGB and depth observations at each navigation step. Unlike Recurrent-VLN-BERT, which processes multiple viewpoints to inform navigation decisions, Recurrent-VLN-BERT-Isaac focuses on a single ego-centric observation per step, making this model more adaptable to various sensor hardware.

6.2.1 Architecture

The architecture of Recurrent-VLN-BERT-Isaac, illustrated in Figure 6.1, integrates language and visual inputs through a transformer-based network.

Initialization Step: The language instruction (e.g., "Go to the kitchen") is processed only once at the beginning of each episode. Firstly, the instruction is tokenized using BertTokenizer from HuggingFace, padded to a fixed length (e.g., 80 tokens), and then

transformed into BertEmbeddings, which are concatenated with position embeddings, and passed through the BertLayer module to generate 768-dimensional language embeddings.

$$\mathbf{E}_{\text{lang}} = \text{BertLayer}(\text{BertEmbeddings}(\text{tokenized\_instruction})),$$

These embeddings include a special [CLS] token that summarizes the instruction. The embeddings except for the [CLS]'s one are computed just once at the start and serve as the foundation for the entire navigation episode. Following [23], the [CLS] token, derived from the initial language embeddings, acts as the state (a dynamic representation of the instruction with respect to observation over steps). the embedding for the [CLS] token does not remain static; rather, it is updated at each subsequent simulation inference step to incorporate new information (by attending to visual information) from the environment.

Visual Processing

Visual inputs are processed using two specialized feature extractors:

- **RGB Feature Extractor**: A pre-trained ResNet-18 model, extracting 512-dimensional features from RGB images. Adaptive average pooling is used to ensure spatial invariance. The equation is represented as:

  $$\mathbf{f}_{\text{RGB}} = \text{ResNet18}(\text{RGB\_image}),$$

  where $\mathbf{f}_{\text{RGB}} \in \mathbb{R}^{512}$.

- **Depth Feature Extractor**: A `CNN` with three convolutional layers, followed by group normalization and ReLU activations, extracts 512-dimensional features from depth maps. Notably, Depth inputs are standardized via bilinear interpolation to improve the training stability, The equation is represented as:

  $$\mathbf{f}_{\text{depth}} = \text{DepthCNN}(\text{standardized\_depth\_map}),$$

where $\mathbf{f}_{\text{depth}} \in \mathbb{R}^{512}$.

The RGB and depth features are concatenated into a 1024-dimensional visual feature:

$$\mathbf{f}_{\text{vis}} = [\mathbf{f}_{\text{RGB}}, \mathbf{f}_{\text{depth}}] \in \mathbb{R}^{1024}.$$

This vector is then projected to $\mathbf{E}_{\text{vis}}$ via:

$$\mathbf{E}_{\text{vis}} = \text{LayerNorm}(\mathbf{W}_{\text{vis}}\mathbf{f}_{\text{vis}} + \mathbf{b}_{\text{vis}}),$$

where $\mathbf{E}_{\text{vis}} \in \mathbb{R}^{1\times1\times768}$, and $\mathbf{W}_{\text{vis}} \in \mathbb{R}^{1024\times768}$, $\mathbf{b}_{\text{vis}} \in \mathbb{R}^{768}$ are learnable parameters.

By adding the depth information as input, the model can identify spatial patterns, enhancing navigation robustness and generalization in environments, even in novel settings, since depth is a more general expression of space than rgb.

Fusion and State Update

Multimodal fusion is performed within the `LXRTXLayer` [26], which implement cross-attention between language and visual features. Following [23], the state is initialized with the [CLS] token from the language embeddings and updated recurrently at each step. The update process involves:

1. Concatenating the previous state with the current visual features:

$$\mathbf{S}_t = [\mathbf{E}_{\text{lang}}^{[\text{CLS}]}, \mathbf{E}_{\text{vis},t}],$$

where $\mathbf{S}_t \in \mathbb{R}^{1\times2\times768}$.

2. Applying cross-attention to integrate visual representation:

$$\mathbf{S}'_t = \text{CrossAttention}(\mathbf{S}_t, \mathbf{E}_{\text{lang}}).$$

3. Performing self-attention and feed-forward operations to further refine the state:

$$\mathbf{S}_t'' = \text{SelfAttention}(\mathbf{S}_t'), \quad \mathbf{S}_t''' = \text{FeedForward}(\mathbf{S}_t'').$$

4. Performing Pooling on the state:

$$\mathbf{p}_t = \tanh(\mathbf{W}_{\texttt{pool}} \cdot \mathbf{S}_t''' + \mathbf{b}_{\texttt{pool}})$$

This process iterates until the simulation ends.

Action Prediction

We use a linear layer as classification head to map the 768-dimensional pooled state obtained from the [CLS] token to a 3-dimensional logit vector, representing probabilities over the action space: "Move forward" (0), "Turn left" (1), and "Turn right" (2). Action a_t at step t is computed as:

$$\mathbf{a}_t = \text{argmax}(\text{softmax}(\mathbf{W}_{\text{action}}\mathbf{p}_t + \mathbf{b}_{\text{action}})),$$

where $\mathbf{W}_{\text{action}} \in \mathbb{R}^{768\times 3}$, and $\mathbf{b}_{\text{action}} \in \mathbb{R}^3$ are the parameters of the action head.

6.2.2 Training

The training process follows an imitation learning paradigm.

Data Preparation

We use our previously proposed `VLN-Go2-Matterport` to train the model, It comprises 869 episodes, each contains:

- An instruction (e.g., "Go to the kitchen") in `instructions.txt`.
- A sequence of RGB images in `rgbs/`.

- Corresponding depth maps in `depths/`.
- Expert actions seperated by 'new lines' in `actions.txt`.

Episodes are split into training (80%) and validation (20%) sets.

Batch Generation

Batches are constructed with a default size of 2 (due to limited GPU resources). RGB and depth images are loaded dynamically instead of pre-loaded, they are also transformed (normalized for RGB, standardized for depth) to optimize memory efficiency.

Loss Calculation

The model is trained using a cross-entropy loss between predicted action logits and ground truth actions. Since each episodes have different sequence length, we add masking to exclude padded steps:

$$L = -\frac{1}{\sum_{i,t} M_{i,t}} \sum_{i,t} M_{i,t} \log P(a_{i,t}|s_{i,t}),$$

where $M_{i,t}$ is the mask for batch i at step t, $P(a_{i,t})$ is the predicted probability of the correct action, and $s_{i,t}$ is the state. Episodes are padded and the loss is averaged over only valid steps via the mask, ensuring robustness to different sequence lengths.

Optimization

We use the AdamW optimizer (learning rate 10^{-5}) with a cosine annealing scheduler (100 steps, minimum learning rate 0) to minimize the loss. To stabilize training, we perform Gradient clipping with a norm of 40; checkpoints are saved every 100 iterations, and we select the best model based on validation loss. A teacher-forcing strategy is employed, where the model uses the ground truth observations from the dataset at every step and compares the action predictions. This approach potentially results in compounding errors, which is one of the limitations of this architecture.

Inference

For each step:

1. Visual features are fused with the previous state.

2. The updated state is pooled, and action logits are computed.

3. The action with the highest probability is selected via argmax.

6.2.3 Dataset Evaluation

We train with 10,000 epochs and get the model with the lowest validation loss. Figure 6.3 and figure 6.2 show the training loss and validation loss of the first 1800 epochs.

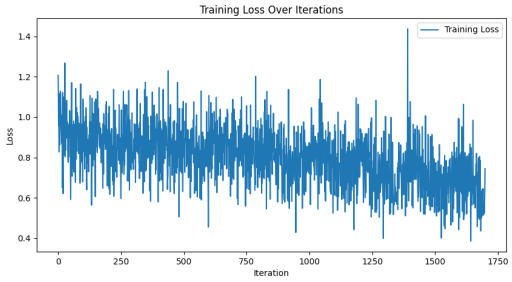

Figure 6.2: Training loss in the first 1800 epochs

Figure 6.3: Validation loss in the first 1800 epochs

The model that yields the highest validation has an accuracy of 70.38% across all 869 episodes.

In-depth Evaluation

We evaluate the behavior and performance of Recurrent-VLN-Bert-Isaac at various stages of training, specifically focusing on attention and action predictions. To ensure consistency across different model checkpoints, we evaluate results from a single episode.

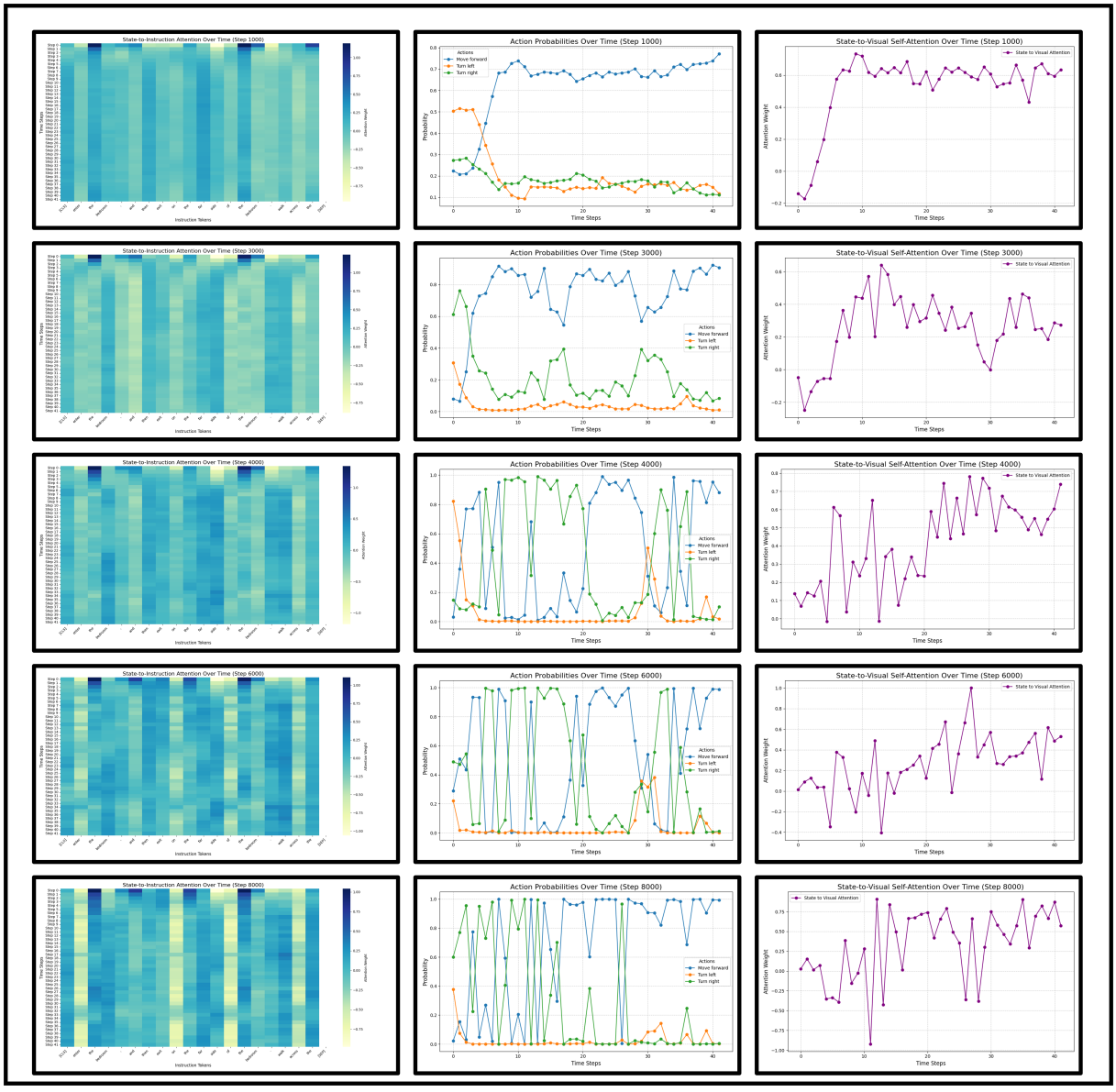

Figure 6.4: Model in-depth analysis, as the model is trained progressively (from top to bottom). It shows state-to-instruction attention score(Left), action logits (Middle) and state-to-vision attention score (right)

In figure 6.4, we examine how the model's attention to the instruction and visual inputs evolves over time during training, and how these attention patterns influence its action

predictions. Specifically in the figure, the **State-to-Instruction Cross-Attention** captures how the state token attends to different parts of the instruction over time. It is computed by averaging attention scores over transformer heads from the last LXRTXLayer's visual attention module. On the other hand, the **State-to-Visual Self-Attention** measures the attention weight from the state token to the visual token within the visual self-attention layer.

We observe that in the initial stage of training (1000 epochs), "Move Forward" is the dominant action to pick. After analyzing the dataset, we determined that this is because "Move Forward" is the dominant labeled action. Therefore, picking "Move Forward" will result in a higher chance of being correct. In the later stage (after 3000 epochs), figure 6.5 illustrates the action distribution across the episodes, validating the statement. Spikes are observed in state-visual attention around the timesteps where changes of actions are required, indicating that the model is relying on visual information to determine its actions. Finally, at the late stage of training (8000 epochs), serious overfitting can be observed from the action probabilities graph. Another observation is that state-to-instruction attention over training attends less to words such as "on", "of", "across", "enter" in the instructions, and attends more to semantically rich words such as "bedroom".

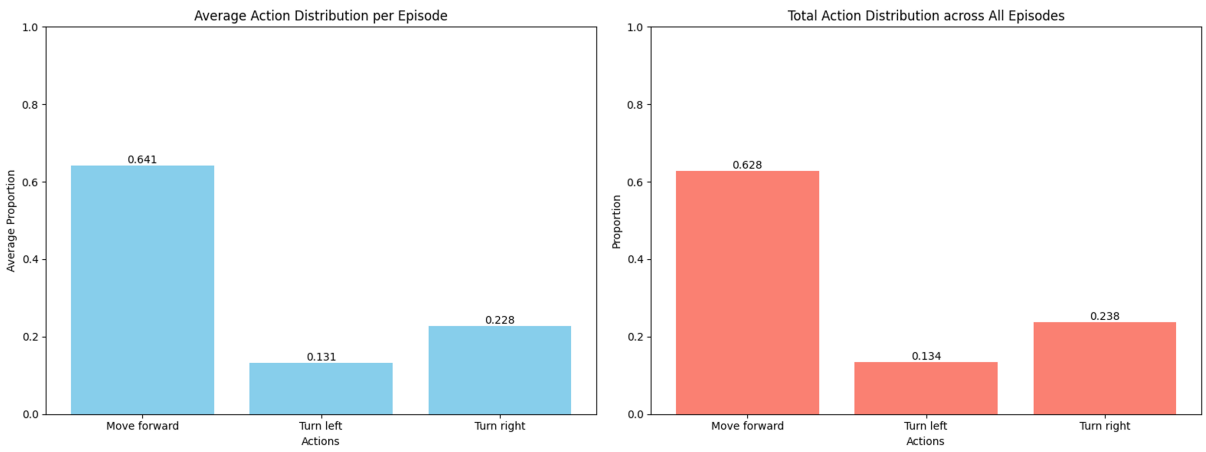

Figure 6.5: Dataset Action Distribution

6.2.4 Simulator Evaluation

Most importantly, we need to test how the model performs in the simulator on an actual robot. So, following how we collect the data in VLN-Go2-Matterport, we replace the Ground-Truth PD-Controller model with the Recurrent-VLN-Bert-Isaac. We set the inference period to be 3Hz (once per 20 simulation steps) to reduce the computation overhead and to be compatible with the period set to obtain the training dataset. We expect that the result will be worse than the dataset result since in the actual simulation, all visual observations across simulation steps are novel despite being trained with the same scenes since the angles and position PD controller captures would not be completely identical every time. We tested the model in the first 10 environments and obtained test results in table 6.1.

Episode	Outcome	Progress
0	Failure	0.16
1	Failure	0.00
2	Failure	0.18
3	Success	1.00
4	Success	1.00
5	Failure	0.47
6	Failure	0.22
7	Failure	0.22
8	Success	1.00
9	Success	1.00
Overall Success Rate		0.40
Average Progress		0.53
SPL (Success weighted by (normalized inverse) Path Length)		0.40

Table 6.1: Test Results on the first 10 episodes

Please refer to the figure 6.6. We have released a simulation test demo on YouTube (click here) for a couple of interesting episodes. In general, Recurrent-VLN-Bert-Isaac succeeds in picking the right actions in the first few time steps. However, It often fails in environments that require multiple 90-degree turns, showing that it cannot memorize well and robustly for long navigation paths. One problem, as mentioned, is compounding errors. Since the training dataset size is small, it does not generalize well enough to

completely novel settings. For example, when we simulate the same episode in Isaac Sim multiple times, each time the periodic depth images and RGB images differ, making each simulation run a novel test.

However, from the demo of episode 10, it appears that the model is capable of performing actions that align with the target doorway or hallway. One possible reason is the addition of depth features. For example, it may discover a pattern from the training dataset that some actions are to align with the deepest part of the depth map.

It tells a very promising possibility. The model is trained with very limited (869) episodes and is able to identify paths from novel views according to the instructions and the images. With a larger training dataset, the model can generalize better to novel settings and be more robust. However, as we will mention in the next section, data scaling may not be the right path to a true generalist navigation model.

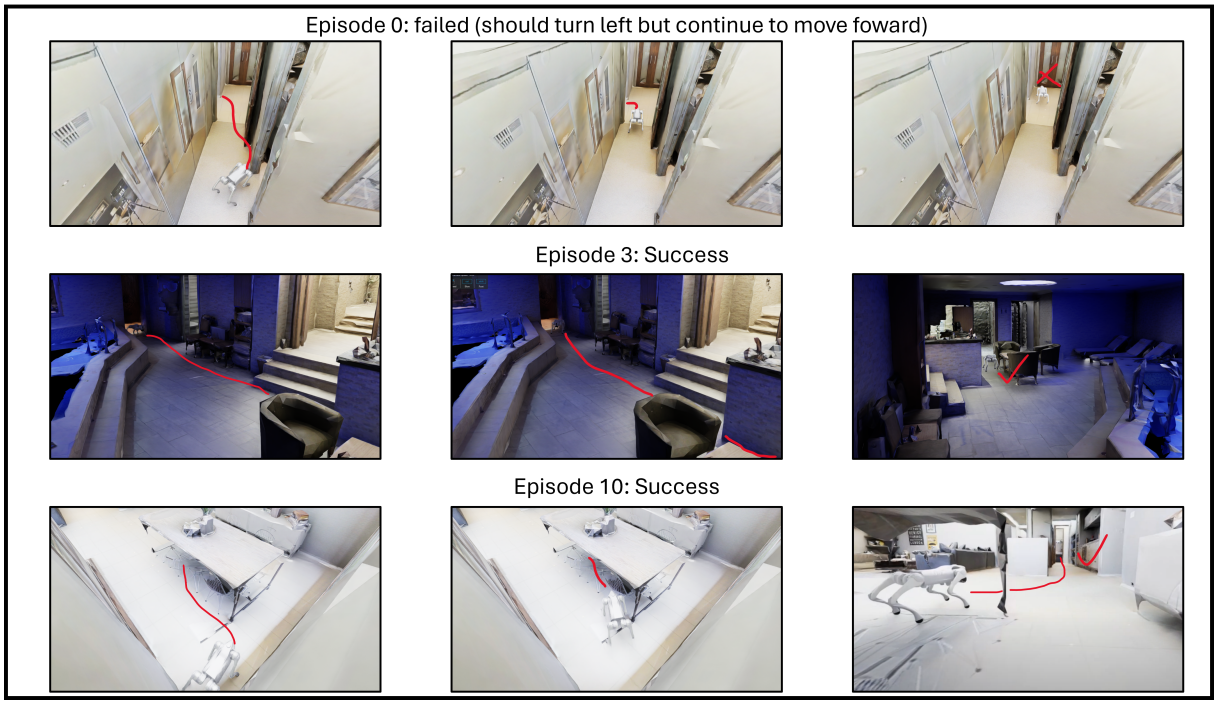

Figure 6.6: Demonstration of trajectories taken on different episodes. YouTube video for episode 10: link

6.2.5 Failure Analysis

In Episode 0, the agent was supposed to:

1. Turn left and move forward until reaching the doorway on the left.
2. Turn left continuously by 90 degrees to align itself with the doorway.
3. Continue the navigation procedure through the doorway.

The model successfully navigated to the doorway on the left. However, instead of executing a full 90-degree left turn to align with the doorway, it only turned left slightly and continued moving forward. This resulted in the agent hitting the door in front of it rather than passing through the doorway.

As noted in section 6.2.3 and illustrated in Figure 6.5, the training dataset has a significant imbalance favoring the action “Move Forward”. This bias could be a cause of such error.

Moreover, the model uses a single [CLS] token-based state feature, updated at each step, as the local memory. This approach may not effectively capture the global features, such as the topological structure of the environments, making it more difficult to localize itself, especially in environments like episode 0, which requires multiple turnings.

6.2.6 Limitations and Challenges

Overfitting. We used a very small dataset to train the model, which potentially resulted in overfitting for the same or similar scenes within the dataset. This can, on the other hand, show that the model can learn more generalized strategies once we obtain a much larger dataset.

Compounding Errors. Since the model adopts a teacher-forcing imitation learning approach at every single step of learning, it results in compounding errors, which happen in the inference time when the model makes the first mistake and fails to correct itself on the right track. This limitation happens because we only store a "local memory,"

which is a single CLS token-based state feature updated at every step, without a global memory such as a topological graph. However, as we tested with a topological graph implementation (Appendix 9.3), the fps in the Isaac Sim Simulator instantly dropped a lot. This could be solved with better hardware or a more efficient way to implement topological graph data structure, such as the one used in MemoNav [27]; we leave this to the future.

Sim-to-real gap exists. Despite using real-world reconstructed environments for this project, the environments still show faulty and still exist a really large difference from the environments in the real world, making sim-to-real transfer challenging.

Action Bias. The actions may be biased toward "Move Forward" due to dataset action imbalance.

Attention Bottleneck. There may exist an attention bottleneck since we assumed a maximum length for all language instructions (80 tokens).

Hardware Constraints. To save time for training and inference, the model infers at 3Hz action frequency, making it less precise in highly dynamic environments.

7 Future

7.1 LLMs for VLN

Many works already incorporate LLMs for VLN tasks, as mentioned in the literature review. Although LLMs contain web-scale knowledge and could be promising for this task, we provided a case study in Appendix 9.2 showing that hallucination in LLMs is still a severe limitation. One way to make LLMs more robust is to allow them to incorporate robust spatial information, such as depth maps. We have also experimented with this by directly feeding depth maps to LLMs. Yet, the results are still very disappointing.

7.2 Spatial-Temporal Reasoning

In this work, we try to add the spatial component by adding a depth map CNN feature to the transformer-based Recurrent-VLN-Bert-Isaac model. Currently, spatial reasoning is a topic of significant interest and shows promise, yet much remains to be explored.

7.3 World Models and VLA Models

Recent studies focus on two directions: state-centric world models and vision language action (VLA) models. However, training these models requires massive datasets and GPU resources.

7.4 Are pre-trained models the future of Embodied AI?

The real world is highly dynamic, and current approaches mainly rely on scaling models with training data. However, the problem is: can we capture enough data for a generalist

embodied agent? If there is not enough training data, there is always something new to learn. Could there be another learning paradigm, such as continual learning?

8 Conclusion

This project significantly enhances Vision-Language Navigation (VLN) by introducing **Recurrent-VLN-Bert-Isaac**, a novel model designed for quadruped robots, and the **VLN-Go2-Matterport** dataset for navigation in high-fidelity indoor environments. Experimental results confirm that the model is able to learn navigation policies transferrable to high physics fidelity simulators, while our exploration of **LLMs-based approaches** highlights their potentials and challenges.

Bibliography

[1] A. Hu, L. Russell, H. Yeo, Z. Murez, G. Fedoseev, A. Kendall, J. Shotton, and G. Corrado, "Gaia-1: A generative world model for autonomous driving," 2023. [Online]. Available: https://arxiv.org/abs/2309.17080

[2] Y. Dai, R. Peng, S. Li, and J. Chai, "Think, act, and ask: Open-world interactive personalized robot navigation," 2024. [Online]. Available: https://arxiv.org/abs/2310.07968

[3] Q. Zhou, S. Chen, Y. Wang, H. Xu, W. Du, H. Zhang, Y. Du, J. B. Tenenbaum, and C. Gan, "Hazard challenge: Embodied decision making in dynamically changing environments," 2024. [Online]. Available: https://arxiv.org/abs/2401.12975

[4] D. Zheng, S. Huang, L. Zhao, Y. Zhong, and L. Wang, "Towards learning a generalist model for embodied navigation," 2024. [Online]. Available: https://arxiv.org/abs/2312.02010

[5] X. Zhao, W. Cai, L. Tang, and T. Wang, "Imaginenav: Prompting vision-language models as embodied navigator through scene imagination," 2024. [Online]. Available: https://arxiv.org/abs/2410.09874

[6] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, "Habitat: A platform for embodied ai research," 2019. [Online]. Available: https://arxiv.org/abs/1904.01201

[7] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. van den Hengel, "Vision-and-Language Navigation: Interpreting visually-grounded navigation instructions in real environments," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[8] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, M. Deitke, K. Ehsani, D. Gordon, Y. Zhu, A. Kembhavi, A. Gupta, and A. Farhadi, "Ai2-thor: An interactive 3d environment for visual ai," 2022. [Online]. Available: https://arxiv.org/abs/1712.05474

[9] Nvidia, "Nvidia isaac sim: Robotics simulation and synthetic data." [Online]. Available: https://developer.nvidia.com/isaac-sim

[10] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. van den Hengel, "Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[11] J. Thomason, M. Murray, M. Cakmak, and L. Zettlemoyer, "Vision-and-dialog navigation," 2019. [Online]. Available: https://arxiv.org/abs/1907.04957

[12] Y. Liu, W. Chen, Y. Bai, X. Liang, G. Li, W. Gao, and L. Lin, "Aligning cyber space with physical world: A comprehensive survey on embodied ai," 2024. [Online]. Available: https://arxiv.org/abs/2407.06886

[13] A. Das, S. Datta, G. Gkioxari, S. Lee, D. Parikh, and D. Batra, "Embodied question answering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[14] B. Peng, C. Li, P. He, M. Galley, and J. Gao, "Instruction tuning with gpt-4," 2023. [Online]. Available: https://arxiv.org/abs/2304.03277

[15] OpenAI, "Gpt-4 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2303.08774

[16] G. Zhou, Y. Hong, and Q. Wu, "Navgpt: Explicit reasoning in vision-and-language navigation with large language models," *arXiv preprint arXiv:2305.16986*, 2023.

[17] D. Zheng, S. Huang, L. Zhao, Y. Zhong, and L. Wang, "Towards learning a generalist model for embodied navigation," 2023.

[18] Y. Zhu, F. Zhu, Z. Zhan, B. Lin, J. Jiao, X. Chang, and X. Liang, "Vision-dialog navigation by exploring cross-modal memory," 2020. [Online]. Available: https://arxiv.org/abs/2003.06745

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[20] G. Bono, L. Antsfeld, A. Sadek, G. Monaci, and C. Wolf, "Learning with a mole: Transferable latent spatial representations for navigation without reconstruction," 2023. [Online]. Available: https://arxiv.org/abs/2306.03857

[21] M. K. Moghaddam, E. Abbasnejad, Q. Wu, J. Shi, and A. V. D. Hengel, "Learning for visual navigation by imagining the success," 2021. [Online]. Available: https://arxiv.org/abs/2103.00446

[22] A.-C. Cheng, Y. Ji, Z. Yang, Z. Gongye, X. Zou, J. Kautz, E. Bıyık, H. Yin, S. Liu, and X. Wang, "Navila: Legged robot vision-language-action model for navigation," 2025. [Online]. Available: https://arxiv.org/abs/2412.04453

[23] Y. Hong, Q. Wu, Y. Qi, C. Rodriguez-Opazo, and S. Gould, "A recurrent vision-and-language bert for navigation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 1643–1653.

[24] W. Hao, C. Li, X. Li, L. Carin, and J. Gao, "Towards learning a generic agent for vision-and-language navigation via pre-training," *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[25] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang, "Matterport3d: Learning from RGB-D data in indoor environments," *International Conference on 3D Vision (3DV)*, 2017.

[26] H. Tan and M. Bansal, "Lxmert: Learning cross-modality encoder representations from transformers," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019.

[27] H. Li, Z. Wang, X. Yang, Y. Yang, S. Mei, and Z. Zhang, "Memonav: Working memory model for visual navigation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

9 Appendix

9.1 Quaternion to Rotation Matrix Conversion

Please refer to The SciPy Documentation and How to Convert a Quaternion to a Rotation Matrix for more details.

A unit quaternion $\mathbf{q} = [q_w, q_x, q_y, q_z]$ can be converted into a 3x3 rotation matrix $\mathbf{R}$ that performs the same rotation on 3D vectors. The conversion formula is:

$$\mathbf{R} = \begin{bmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_x q_y - q_z q_w) & 2(q_x q_z + q_y q_w) \\ 2(q_x q_y + q_z q_w) & 1 - 2(q_x^2 + q_z^2) & 2(q_y q_z - q_x q_w) \\ 2(q_x q_z - q_y q_w) & 2(q_y q_z + q_x q_w) & 1 - 2(q_x^2 + q_y^2) \end{bmatrix}$$

This matrix $\mathbf{R}$ is orthogonal ($\mathbf{R}^T\mathbf{R} = \mathbf{I}$) with a determinant of 1.

The formula arises from the quaternion rotation operation:

$$\mathbf{v}' = \mathbf{q} \cdot \mathbf{v} \cdot \mathbf{q}^*$$

where:

- $\mathbf{v} = [0, v_x, v_y, v_z]$ is the vector to be rotated, represented as a pure quaternion,
- $\mathbf{q}^*$ is the conjugate of $\mathbf{q}$, given by $\mathbf{q}^* = [q_w, -q_x, -q_y, -q_z]$,
- $\cdot$ denotes quaternion multiplication.

The resulting matrix form is more computationally efficient for transforming vectors via $\mathbf{Rv}$, making it ideal for real-time applications.

In Python, the respective function call with `scipy.spatial.transform.Rotation`:

```
from scipy.spatial.transform import Rotation
rot = Rotation.from_quat([
    robot_quat[1],   # q_x
    robot_quat[2],   # q_y
    robot_quat[3],   # q_z
    robot_quat[0]    # q_w
])
rot_matrix = rot.as_matrix()
```

9.2 LLM case study

Apart from the Latent Representation-based Recurrent-VLN-Bert-Isaac, we have also created an LLMs-based architecture with Qwen VL and Qwen Max for this task. However, it has failed with all 10 episodes we tested. below, we have attached a fail case showing the prompt (automatically generated), response, and action taken for the LLM Architecture for further investigation. The code is also released in Github for study.

Since the case study involves AI-generated navigation reasoning text, to avoid AI detection in this report, we put a link **here** which re-directs to the pdf file of the LLM case study. The code that runs the LLM architecture in the simulator and generates the report is in here.

9.3 Topological Graph

The implementation of the topological graph. The code was designed for a panoramic camera view; here, we adapted this for an ego-centric camera view from Isaac Sim.

Algorithm 1 Topological Graph Construction

Initialize empty graph $\mathcal{G}$ with capacity $M = 100$
while agent explores environment **do**
  Extract visual embedding $f_t \leftarrow \text{CNN}(I_t, D_t)$
  Get current pose (p_t, q_t) from simulator
  if first frame **then**
    $\mathcal{G}$.add_node$(0, f_t, p_t, q_t)$ {Initialize root node}
  else
    Compute similarity $\alpha \leftarrow \text{cosine}(f_t, F_{\text{prev}})$
    if $\alpha < T_{\text{img}}$ (e.g., 0.8) **then**
      $\nu_{\text{new}} \leftarrow \mathcal{G}$.add_node$(t, f_t, p_t, q_t)$
      $\mathcal{G}$.add_edge$(\nu_{\text{prev}}, \nu_{\text{new}})$
    else
      $\mathcal{G}$.update_node(ν_{prev}, f_t)
    end if
  end if
  $\nu_{\text{prev}} \leftarrow \nu_{\text{current}}$ {Track last localized node}
end while