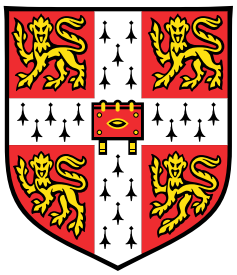
LanGWM: Language Grounded World Model



Nanze Chen

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy

St Edmund's College

September 2024

Declaration

I, Nanze Chen of St Edmund's College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed Nanze Chen Date 14 August, 2024

Word Count: 14998

Software Declaration: Standard Python packages for machine learning applications were employed such as `TensorFlow`, `Keras`, `PyTorch`, `NumPy`, and `Matplotlib`.

Various software libraries and repositories were leveraged in this work:

- **LanGWM**(private code shared by Poudel et al. (2023)) is our baseline. Modification and development are done on top of the original LanGWm pipeline.
- **iGibson 1.0**(https://svl.stanford.edu/igibson/1.0/docs/installation.html - Shen et al. (2021)) is used, without modification, as simulation environment
- **Grounding DINO** (https://github.com/IDEA-Research/GroundingDINO - Liu et al. (2023)) is loaded, without modification, to use its pretrained visual backbone to generate visual-language embedding.
- **DINO** (https://github.com/facebookresearch/dino?tab=readme-ov-file - Caron et al. (2021)) is loaded, without modification, to use its pretraiend visual backbone to generate visual embedding.
- **DreamerV2** (https://github.com/danijar/dreamerv2 - Hafner et al. (2020)) is incorporated into LanGWM, with modification in architecture, as the world model module.
- **BERT** (https://huggingface.co/docs/transformers/en/model_doc/bert - Devlin et al. (2018)) is loaded with pretrained checkpoints, without modification, to generate language embedding.

- **DETR** (https://github.com/Visual-Behavior/detr-tensorflow/tree/main - Carion et al. (2020)) is refered for building our transformer-based detection decoder.
- **ChatGPT** (https://chat.openai.com/ - OpenAI (2024)) is used for checking the grammatical errors and reducing redundancy in Chapter 5. ChatGPT has not created any of the content. It was only asked to delete words and correct grammatical mistakes on top of the original text I wrote.

Nanze Chen
September 2024

Acknowledgements

I would like to express my sincere gratitude to my supervisors at Toshiba Europe Ltd., Dr. Rudra P.K. Poudel and Dr. Svetlana Stoyanchev, for their unwavering support and valuable insights throughout this project. I am particularly grateful to Rudra for his continuous guidance and collaboration in designing the experiments, and for his concern about my well-being. I vividly recall him saying, "I hope to see you happy next week," after one of our meetings, which had a significant positive impact on me. I would also like to thank Svetlana for her critical thinking during our discussions. Her objective evaluations of my experimental designs provided me with the clarity to approach my work with a more balanced perspective, preventing overconfidence and encouraging careful consideration of all aspects.

I am also deeply appreciative of my internal supervisor at the University of Cambridge Department of Engineering, Brian Sun. His valuable suggestions have been instrumental not only in advancing my experiments but also in improving the quality of this dissertation. Reflecting on the journey, I realize that some of his advice, which I did not fully appreciate at the time, could have significantly smoothed the progress of my project and enhanced the significance of my experimental results.

I would like to extend my heartfelt thanks to my girlfriend, Yanyu Zhong, for her invaluable support. Her insights from a software engineer's perspective have been extremely helpful in my cooperating and coding. I am also grateful for her emotional support, her willingness to listen to my challenges, and her understanding of my struggles. Even though we are physically distant, we have worked hard to stay connected and support each other throughout this journey.

A special thank you to my family for their unwavering financial and emotional support. Studying at Cambridge has brought me closer to my dreams, and I could not have achieved this without your generosity and encouragement.

I am also grateful to my study companions, Fengzhe Zhang, Junyi Qian, and Weijia Ai. Your advice and encouragement kept me going even during the most challenging times. I also

extend my thanks to all the friends I have met in the MLMI program, who have made this year a memorable experience. We have endured the most difficult year of my life so far, and I am thankful for the camaraderie we shared.

Finally, I would like to thank Cambridge and all the professors who have imparted valuable knowledge, challenged me through examinations, and placed the pressures that have shaped me into who I am today. Though the journey was arduous, the challenges have spurred my growth, far beyond where I stood a year ago.

Abstract

Navigating to a specific coordinate in an unfamiliar room is a straightforward task for humans, as they can quickly orient themselves by observing their surroundings. However, this task poses a significantly greater challenge for robots. In recent years, reinforcement learning (RL) has made substantial progress, with deep reinforcement learning (DRL) demonstrating considerable potential in addressing complex tasks. A particularly noteworthy advancement is the concept of World Models, which focuses on constructing an internal representation of the environment within the agent's mind, allowing it to predict future and plan actions accordingly. Since their introduction in 2017, World Models have surpassed traditional model-free reinforcement learning approaches in various tasks, often requiring even less training time. Nevertheless, current World Models are still far from achieving a true understanding of their environments. This is evident from experiments on visual control tasks, where even state-of-the-art World Models struggle in unfamiliar settings. If these models can comprehended the semantic meaning of the environment, they should be able to quickly adapt to new surroundings, much like humans do.

This project aims to explore how to enhance World Models in the aspect of understanding of environmental semantics. We hypothesize that language information is crucial for enabling World Models to grasp the contextual meaning of environments, as language provides a powerful means of efficiently describing surroundings. In this context, the existing *LanGWM: Language Grounded World Model* framework has introduced an approach that integrates language and visual inputs, utilizing a masked auto-encoder training method. Building upon this foundation, our project further refines the training methodology of the environment encoder and investigates various components within LanGWM's training pipeline. These efforts represent an advancement in the LanGWM framework and underscore areas for future improvement.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Do machine learning models make decisions based on their knowledge of the world, or do they rely solely on experience? This question has long puzzled machine learning scientists and sparked considerable debate. Some argue that most traditional architectures depend on regression to derive insights from training data, meaning these models lack a true understanding of general concepts within the data. Conversely, other scientists believe that using observed features to make decisions demonstrates a form of understanding, suggesting that models will gradually develop a deeper comprehension of the world as they encounter more data.

One might question the importance of this issue as long as the model makes correct decisions. This concern is negligible for simple tasks, such as text style transfer, where models rely on statistical methods to predict the most likely next word given preceding words. Similarly, in image classification, despite the higher dimensionality of the data, the primary mechanism remains statistical, identifying the most likely class label based on image features. In these scenarios, models do not grasp the meaning behind labels or text; the distinction between "person" and "label 0" is irrelevant to the model.

However, as we assign machine learning models with increasingly complex and realistic tasks, this distinction becomes critical. For instance, in autonomous driving, the model must not only detect the presence of a person but also discern if the person is real or an image on a billboard. It has to predict if the person will obstruct the vehicle's path and determine how to maneuver to avoid a collision. Here, treating "person" as equivalent to "label 0" is inadequate, as the former entails understanding motion, 3D shape, ethical implications, and more.

While larger models and datasets might improve performance on complex tasks, this approach is both costly and unreliable in unforeseen situations or attacks. Unseen scenarios, considered out-of-distribution tasks, challenge models that rely on familiar features. Attacks on machine learning models exploit their dependency on observed features, highlighting the limitations of regression-based approaches. Ultimately, we aim for models to think like humans, comprehending the semantic meaning of every input.

Understanding the global concept and semantic meaning of the objects that might appear in the input aligns with the idea of the World Model. The concept of the World Model was first proposed by Jürgen Schmidhuber in his paper "Recurrent World Models Facilitate Policy Evolution," published in NeurIPS ‘18(Ha and Schmidhuber, 2018). Inspired by the human brain’s mental model, Jürgen defined the World Model as a special kind of model-based RL algorithm. In the architecture of World Model RL, an agent has a model that can simulate the world. In this "simulated world," it can "imagine" how its actions will interact with the surrounding environment and thus plan its actions accordingly.

As a World Model, the ability to imagine is crucial. But what does "imagine" entail in this context? According to Judea Pearl’s book, "The Book of Why," there are different levels of causal inference that shed light on this concept(Pearl and Mackenzie, 2018). At the bottom level is "association," which involves linking causes to effects based on past experiences, akin to what regression models do. The mid-level is "intervention," where one interacts with the environment to understand the outcomes, typical of RL models. The highest level is "counterfactual reasoning," which involves making decisions even under conditions that have not been encountered during training (see Figure 1.1. This highest level, counterfactual reasoning, is the essence of imagination in a World Model.

To enhance a World Model’s ability to truly understand and represent its environment, it must move beyond simple feature matching and association. While many large models today excel at traditional out-of-distribution (OOD) tasks, their success often stems from encountering similar features across datasets, rather than a genuine understanding of novel scenarios. For instance, models trained on the COCO dataset may perform well on the ImageNet dataset, but this is not a true OOD task since both datasets consist of real-world images with overlapping features. The key challenge, therefore, is to develop World Models that can form robust, high-level representations of their environments, allowing them to generalize effectively to entirely new and unseen situations.

Building upon the foundation of the existing LanGWM (Language Grounded World Model) framework, which integrates language and visual inputs using a masked auto-encoder training method, our project re-evaluates the role of semantic understanding in World Models,

particularly in the context of out-of-distribution (OOD) PointGoal navigation tasks. We propose an enhanced training pipeline that moves beyond pixel-wise depth reconstruction, incorporating segmentation and detection tasks to better capture high-level semantic information. By integrating these elements into the LanGWM framework, we aimed to develop a model capable of more robust environment comprehension. Although our experiments revealed limitations in the current architecture, especially with the task observation encoder and multi-modal feature encoder, our findings highlight the potential of language grounding to improve performance in novel environments. These insights pave the way for refining World Models to better mimic the human-like ability to adapt to unfamiliar settings.

1.1 Contribution

- **Implementation of New Decoders and Design New Training Tasks for LanGWM Pipeline**: Implemented segmentation decoder and detection decoder for the LanGWM pipeline. Design the associated training task details, such as loss calculation and data generation.
- **Identification of Limitations in Current Architecture**: Conducted experiments that identified significant limitations in the current LanGWM pipeline.
- **Demonstration of Language Grounding Potential**: Perform experiments that show the potential of language grounding to enhance the performance of World Models in novel environments.

1.2 Roadmap

The thesis is organized as follows:

- In Chapter 2, we offer background knowledge in the field of robotics indoor navigation task, world model, and encoder for feature extraction. These are some basic knowledge necessary for understanding our tasks.
- In Chapter 3, we introduce LanGWM pipeline and our methods of implementing segmentation decoder and detection decoder.
- In Chapter 4, we introduce all the experiments that we have done.
- In Chapter 5, we conclude this project and discuss the future work.

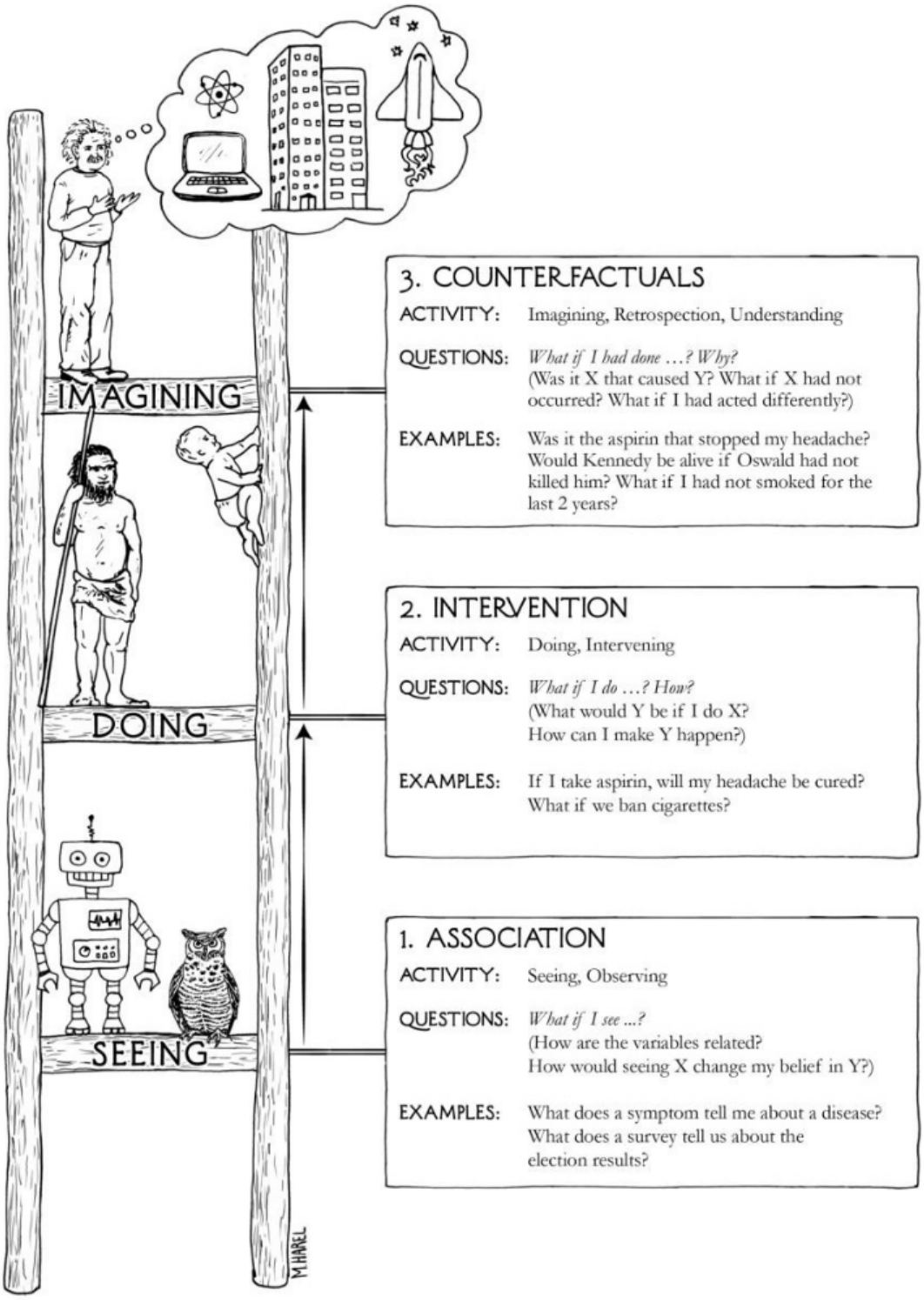


Fig. 1.1 **Levels of causal inference in Judea Pearl's book "The Book of Why" (Pearl and Mackenzie, 2018).** It reveals the essence of "imagination" in a World Model is to do counterfactual reasoning, which, in another word, requires the model to tackle situation not existed in the dataset.

Chapter 2

Background

In this chapter, we begin by discussing the out-of-distribution (OOD) PointGoal Navigation tasks faced by our robotic system, emphasizing the importance of a comprehensive environment embedding for effective task performance. An **environment embedding**, or environment representation, is generally an encoding of all the environment information, such as RGB visual input, radar depth information, robot's orientation, and so on. The ability to navigate accurately in the challenging OOD scenarios hinges not only on the quality of the environment representation but also on the efficiency the environment encoder. This forms the basis for our motivation to explicitly train an environment encoder capable of capturing both the physical and semantic details necessary for successful navigation.

Following this, we introduce the concept of the World Model, tracing its origins in model-based reinforcement learning where it was initially used to simulate and predict environmental dynamics for agents. In the LanGWM pipeline, the World Model plays a crucial role, and our approach focuses on training a more effective environment encoder that can provide a comprehensive representation of the environment, enabling the World Model to better predict environmental dynamics and generalize across diverse and unseen settings.

Finally, we explore how advancements in transformer architectures, particularly transformer encoders, have revolutionized the extraction of global features from input data. Unlike traditional regression methods, which often fall short in capturing complex, high-dimensional relationships, transformer-based encoders excel in producing rich, multi-modal environment embeddings. This capacity to integrate information across different modalities, such as visual and linguistic data, directly informs our decision to employ a multi-modal transformer-based encoder, ensuring that our environment embeddings are both comprehensive and effective for the tasks at hand.

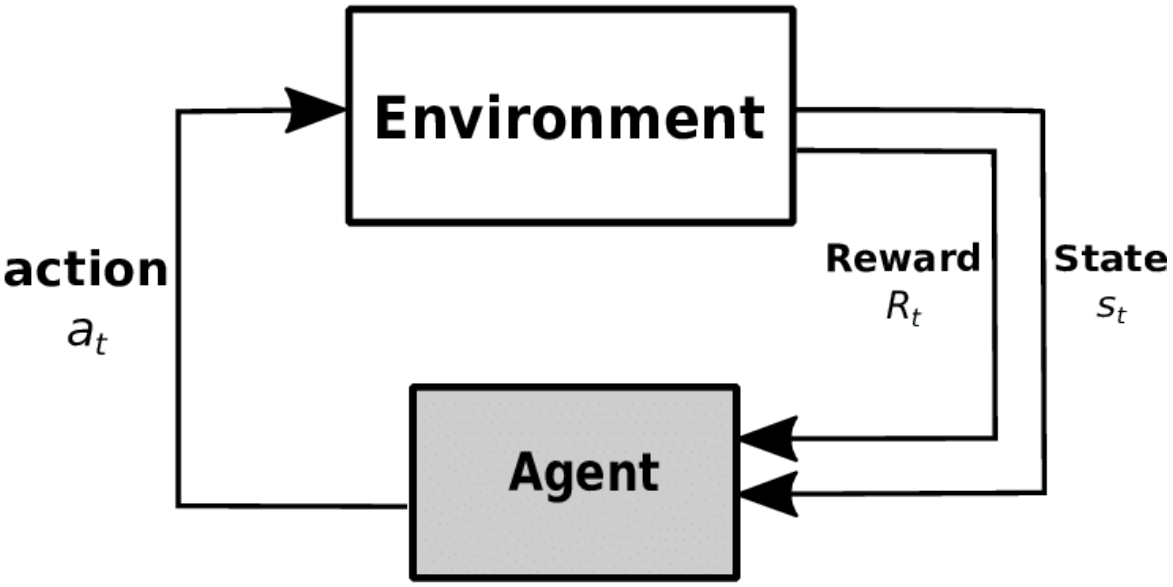


Fig. 2.1 A general flowmap showing the interaction between agent and environment. At time t, the agent will output action command a_t to the robot, which is a part of the environment. Then, the environment will capture reward(R_t) and environment state (s_t) at time t (Amiri et al., 2018).

2.1 Robotics Indoor Navigation Task

2.1.1 Reinforcement Learning

In the context of RL, there are always two essential roles: the agent and the environment. The **agent** usually means the RL model, while the **environment** encompasses everything else, including the robot itself and the physical environment the robot interacts with. During training or inference, the agent will decide what is the next action and then send the commands to the robot's motors. In response, the environment provides state (or observations) in return of the action, such as RGB images from the robot's camera and its current location if provided, as well as rewards that evaluate how effectively the action contributes to completing the task. Through this process, the RL model learns to associate actions with rewards given observations. Since the strategy the RL model learns evolves over time, the agent will perform different actions given even the same observations and therefore receiving different reward. Therefore, RL model training requires dynamically collected data throughout the process (Sutton and Barto, 2018).

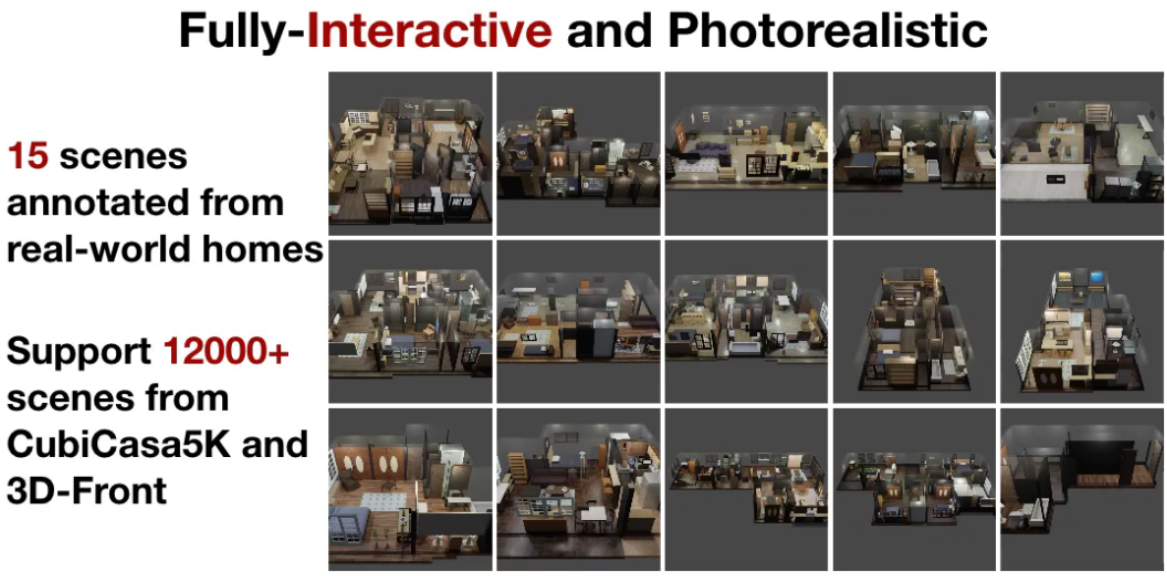


Fig. 2.2 The 3D demo of the 15 simulated room in iGibson 1.0. All the objects in iGibson 1.0 is interactive and reconstructed from real images (Shen et al., 2021).

2.1.2 Experiment Environment

Simulation Environment

However, performing RL training in the real world can be prohibitively costly in terms of resources, which is why simulation environments are widely used. A simulation environment is essentially a virtual environment created through computer programming. Usually, a simulation environment will mimic the environment the agent will interact with after training. For example, RL models trained to play video games often operate in simulated environments that replicate the game but at faster speeds and with parallelism. Similarly, as in many other robotics tasks, the simulation environment mimics the real world.

Reproducing physical laws is a critical aspect of accurately mimicking the real world, a task accomplished by physics simulators, which are a fundamental component of simulation environments. **Physics simulators** model the physical interactions within a simulation environment—including object collisions, fluid dynamics, and force applications—thereby enabling an accurate replication of real-world dynamics (Coumans et al., 2013; Lee et al., 2018; Liang et al., 2018; Todorov et al., 2012). Notable examples of physics simulators are Bullet and MuJoCo, renowned in robotics research for their capability to simulate detailed interactions in real-time (Coumans et al., 2013; Todorov et al., 2012).

Besides physics simulators, simulation environments also consists of virtual sensor simulations, thus providing a comprehensive development framework. These environments are

crucial for training robots in tasks ranging from basic navigation to complex manipulation. AI2Thor and Habitat are well-known examples, especially for object manipulation and navigation within large-scale indoor environments, respectively (Kolve et al., 2017; Savva et al., 2019). However, these environments often face limitations, such as restricted interactivity or reliance on predefined actions that oversimplify the physics of interactions, rendering them less effective for tasks requiring full physical simulation.

In our work, we use iGibson 1.0 (see Figure 2.2), which is an advanced simulation environment that stands out for its ability to simulate large-scale, realistic indoor scenes with full interactivity. It integrates a robust physics engine, high-quality sensor simulations, and domain randomization capabilities. For our project, which involves the OOD PointGoal Navigation task, iGibson 1.0 provides 15 fully interactive scenes with a total of 108 rooms and a large number of objects with various appearances and features. This ensures that LanGWM can be trained on a wide range of scenarios and also can be tested in held-out scenes[1]. Its ability to provide multi-modal information about the scene, including RGB images, depth, and segmentation, is also crucial for training a powerful environment encoder.

Robot

Robots play a crucial role in embodied AI research, serving as the carrier of agents that interact with simulated environments to perform various tasks. In the iGibson 1.0 simulation environment, several robots have been implemented to support diverse research goals, as shown in Table 2.1. These robots range from simple, two-degree-of-freedom (DOF) platforms to more complex systems with multiple actuators and sensors.

In our baseline, the TurtleBot serves as the primary robotic agent (see Figure 2.3). The TurtleBot is a widely-used, open-source mobile robot platform, popular in both education and research for its affordability and versatility. Typically, TurtleBot models, such as the TurtleBot 2 and TurtleBot 4, feature a differential drive system with two wheels, allowing for smooth navigation. The robot has a compact, cylindrical shape, making it well-suited for maneuvering in indoor environments. The TurtleBot is equipped with various sensors that enable it to perform tasks like Simultaneous Localization and Mapping (SLAM), autonomous navigation, and obstacle avoidance. For instance, it often includes an RGB-D camera or LiDAR for capturing detailed 3D representations of its surroundings. These sensors are critical for tasks that require real-time environmental perception and interaction, making the

[1]Held-out scenes represents scenes whose features (e.g. floor plan, furniture, texture, wall papers) are totally different from previous viewed scene, which is fit for evaluating model's ability to handle OOD tasks.

TurtleBot an ideal platform for developing and testing algorithms in autonomous robotics (TurtleBot, 2024).

Agent Name	DOF
Mujoco Ant	8
Mujoco Humanoid	17
Husky Robot	4
Minitaur Robot	8
Quadrotor	6
TurtleBot	**2**
Freight	2
Fetch	10
JackRabbit	2 & 7
LocoBot	2

Table 2.1 **Simulated robots provided by iGibson 1.0 and their DOF.**

Fig. 2.3 **TurtleBot**

2.1.3 PointGoal Navigation Task

The PointGoal navigation task represents a significant benchmark in the field of embodied AI, where an agent is required to navigate to a indoor target location using relative goal coordinates and sensory inputs such as RGB-D images. This task assesses the agent's abilities in perception, planning, and navigation, making it a comprehensive test of an agent's environmental understanding and adaptability. The primary challenge is for the agent to effectively interpret sensory data to perceive its surroundings, devise a route from its current position to the goal, and dynamically adjust its path as new sensory information is received.

The ability to navigate efficiently within an indoor environment is fundamental to the development of personal robots. It has been a focal point of research in computer vision for many years, as noted by (Nilsson et al., 1984). Savva et al. (2017) introduced three primary indoor goal-directed navigation tasks: PointGoal, ObjectGoal, and RoomGoal. Anderson et al. (2018) further defined these tasks, explaining that in **PointGoal** navigation, the agent must navigate to a specific location. While this task is straightforward in an empty environment, it poses significant challenges in cluttered, unexplored spaces. **ObjectGoal** requires the agent to find an object of a specific category, such as a refrigerator or keys, using prior knowledge about the object's appearance and typical location. **AreaGoal** involves

navigating to a specific area type, like a kitchen or garage, also relying on the agent's understanding of typical spatial layouts.

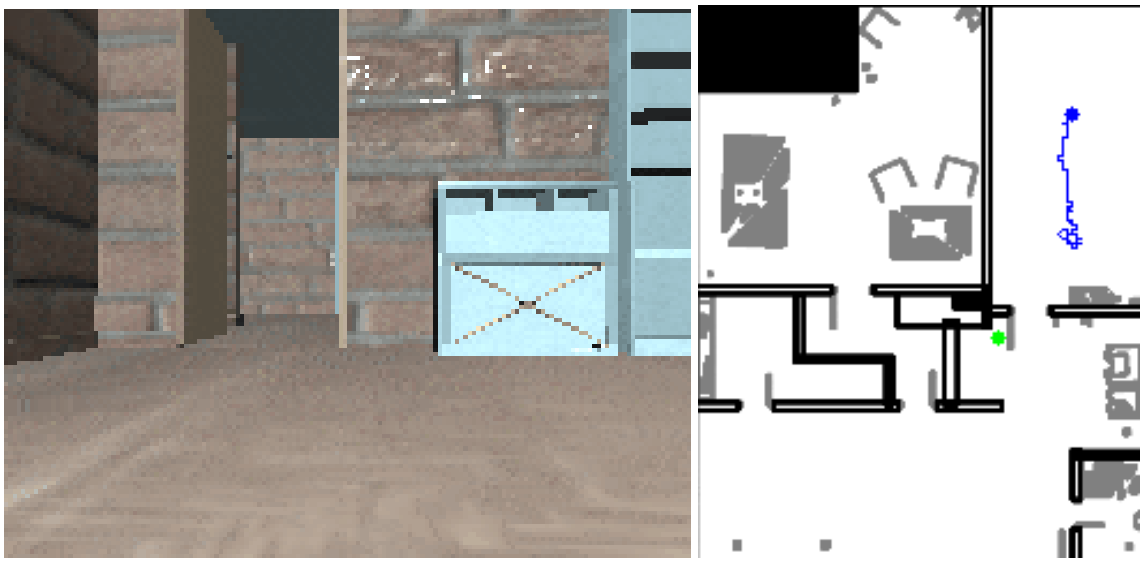


Fig. 2.4 **PointGoal Navigation Task Demonstration.** This is a demonstration of PointGoal navigation task from our later experiment. The right image shows a top view travel map of a robot. The blue point is the starting point and the green point is the destination. The thin blue line is the robot's trajectory. We can see the robot failed to reach its destination as it struggle to go through a door. The left image is one frame of the RGB input that the robot takes during the task.

In this work, we focus on **PointGoal** navigation task (see Figure 2.4). There are two important metrics for this task:

- **Success Rate** (SR): The percentage of test episodes that the agent successfully navigates to the goal.
- **Success weighted by Path Length** (SPL): A success score that weighted the success rate inversely with the path length taken by the robot to reach the goal, calculated by:

$$\frac{1}{N}\sum_{i=1}^{N} S_i \frac{l_i}{\max(p_i, l_i)}, \tag{2.1}$$

  where N is the total number of test episodes; S_i, l_i, and p_i are, respectively, the binary indicator of success, the shortest path length to the goal, and the length of the path actually taken by the agent in episode i. The largest possible SPL is 1.0, meaning all the task is completed with the robot always taking the shortest path to the goal.

Previous Works' Performance

Although tasks similar to PointGoal navigation long exist before Anderson et al. (2018) defines then, previous work mainly focus on navigation in made-up 3D simulated maze. The researchers shows that navigation in real-world environment are much harder than in made-up maze. In 2017, Savva et al. (2017) evaluated the state-of-the-art (SOTA) RL algorithms and discovered that they only performed well in simple, three-dimensional generated mazes (Mnih et al., 2016). To address this limitation, Savva et al. (2017) proposed MINOS (Multimodal Indoor Simulator), which facilitate the development of multi-modal models for navigation in more complex indoor environments. The researchers trained four SOTA RL models[2] in MINOS with training hyperparameters were based on those published by Jaderberg et al. (2016). The final test results indicated that even the best-performing agent achieved only an 80% SR in simple, two-room empty environments, whereas in the more complex 24-room 3D houses, the SR dropped to 20% or lower. In contrast to the nearly 100% SR observed in the made-up mazes (Mnih et al., 2016), these findings underscore the limitations of the RL algorithms of that era for deployment in real-world environments.

Subsequent research has increasingly focused on PointGoal Navigation in real-world environments. In 2019, Wijmans et al. (2019) introduced an parallel RL training method called Decentralized Distributed Proximal Policy Optimization (DD-PPO)[3], which can train RL models with much higher efficiency. Researchers utilize it and trained a navigation model with 2.5 billion steps, which is equivalent to 80 years real-world experience. The navigation models was tested in PointGoal navigation task and achieves near-perfect autonomous navigation in unseen environments (OOD tasks): SR exceeding 98% and SPL above 0.94. Compared to the earlier work by Savva et al. (2017) shown in last paragraph, which also utilized the Matterport3D dataset, this model and its training methods demonstrated significant improvements in navigation performance. Notably, this work leveraged a particularly rich set of multimodal inputs, including RGB-D images, GPS data (agent's current location), compass information (agent's orientation), and goal location data.

However, subsequent research by Zhao et al. (2021) raised concerns about the reliance on perfect environmental information, noting that GPS and compass data are typically unavailable in real indoor environments, and camera inputs are often noisy. Zhao et al. (2021) found that when visual sensor and execution noise were introduced and GPS and Compass

[2]The SOTA RL models they trained include Feedforward A3C, LSTM A3C, UNREAL, and DFP (Dosovitskiy and Koltun, 2016; Gupta et al., 2017; Mnih et al., 2016).

[3]The model architecture employed in this work included a vision encoder (with experiments conducted using three variants: ResNet50, SE-ResNeXt50, and SE-ResNeXt101 (He et al., 2016; Hu et al., 2018)), an MLP goal encoder, and a navigation agent module that integrated LSTM and PPO (Zhao et al., 2021).

Fig. 2.5 **Made-up Scene V.S. Simulated Indoor Scene.** The left image is the training task in *Asynchronous Methods for Deep Reinforcement Learning* (Mnih et al., 2016): An agent navigating in a made-up 3D maze to collect apple and access goal. The right image is a robot performing navigation task in iGibson 1.0. The realistic indoor environments have much more obstacles, thereby is much harder for agents to navigate.

were removed, the SR of the model from Wijmans et al. (2019) dropped to just 0.3%. In response, Zhao et al. (2021) focused on enabling PointGoal Navigation using only RGB-D images in noisy environments. They employed Particle SLAM, which combines particle filtering with SLAM techniques and successfully improved the PointGoal navigation SR to 71.7% under noisy conditions.

The core approach of this work involves using RGB-D images as input and predicts Visual Odometry, which generaly represents the movement of the robot. This method effectively replaces the need for Compass and GPS. In a later work, Partsey et al. (2022) further improve the Visual Odometry Module with larger datasets, data augmentation and better feature labeling. These enhancement raise the SR from 71.7% to 94% and raise SPL from 0.53 to 0.74, which is close to the "near-perfect" state.

The significance of these two studies lies in training a neural network to accurately extract physical information about the robot from visual data, thereby significantly enhancing its navigation capabilities. This approach inspired us to explore training an environment encoder that extracts both environmental and robot-specific physical information from RGB images, with the goal of achieving a more comprehensive environment embedding. The detailed methodology for this approach will be presented in the subsequent section.

2.1.4 Out-of-distribution Tasks for Robotics

The ability to generalize to OOD tasks is crucial for robots to be deployed in real-world environments. As robots increasingly rely on machine learning models to interact with their surroundings, the challenge of ensuring reliable performance in unseen scenarios becomes more pressing. This is especially important in safety-critical applications, such as autonomous vehicles or drones, where failures could cause traffic accidents. Traditional RL models often struggle to adapt to new environments that deviate from the data they were trained on, making OOD generalization a key area of focus in robotics research (Farid et al., 2022).

To test a model in OOD data, we have to first identify what is a OOD dataset. Sinha et al. (2022) define OOD data as instances where the test data does not follow the same distribution as the training data. In typical supervised learning tasks, the training and testing datasets are often a counterexample of OOD dataset: they are split randomly, ensuring that both datasets are representative of the same overall distribution (DeVries and Taylor, 2018).

However, in robotics, training and testing agents in different environments are often counted as a OOD task. For example, consider a drone trained to navigate in one living room on vision. Unlike supervised learning tasks, such as image classification whose dataset always contains diverse features, the vision features in one room is always limited. If this drone is then deployed in another living room where the furniture's types, lighting, and object placements are all unfamiliar, the distribution of the sensory input it receives will differ significantly from what it encountered during training. In a 3D world, changes in object placement, orientation, or even surface textures can result in large shifts in data distribution, making it much harder for the model to generalize effectively (Cai and Koutsoukos, 2020; Farid et al., 2022; Sedlmeier et al., 2019, 2020). On one hand, this again reveals the importance to train robot agent in OOD tasks. On the other hand, this shows the possibility of using iGibson 1.0's capability of controlling detailed feature in the room to easily create OOD test dataset.

We acknowledge that reliance on OOD tasks in robotics could be reduced if we could collect sufficiently diverse data to enable RL models to generalize effectively across varied real-world environments. However, even the most sophisticated simulation environments fall short of this objective. To tackle this, robotics scientists rigorously train and test a model's capability to handle OOD tasks. Meanwhile, they are also exploring strategies to enhance the generalization of RL models. In the following section, we will explore the history of the 'World Model' —how RL researchers aspire to incorporate an imagined world within the agents' cognition.

2.2 World Model

As discussed in the previous sections, researchers are increasingly addressing RL tasks that account for real-world complexities. In real-world indoor navigation tasks, challenges such as sensor limitations—including issues with GPS, compass, and radar—must be considered, alongside unpredictable factors like human movement, arbitrary furniture placement, and out-of-distribution (OOD) scenarios. As the reliability of inputs decreases and task environments become more intricate, researchers have shifted their focus towards reducing models' dependency on specific inputs.

One key solution researchers developed over time is enhancing a model's ability to imagine the consequences of its actions based on prior knowledge rather than solely relying on inputs. This approach has eventually evolved into the concept of the "World Model."

In this section, we will first introduce the precursor to the World Model: Model-based Reinforcement Learning and compare it with Model-free Reinforcement Learning. We will then discuss the concept of the World Model and its development trends in recent years.

2.2.1 Model-based RL v.s. Model-free RL

Model-free Reinforcement Learning

Model-Free Reinforcement Learning (MFRL) is distinguished by its direct approach to learning policies without the need to construct an explicit model of the environment. In MFRL, the agent learns by interacting with the environment, receiving rewards as feedback, and gradually improving its policy purely through trial and error. This approach bypasses the complexities involved in modeling the environment's dynamics, instead focusing on learning a mapping from states to actions that maximize cumulative rewards over time. Techniques such as Q-learning, DQN (Deep Q-Network), and actor critics algorithms fall under this category (Konda and Tsitsiklis, 2000; Mnih et al., 2015; Watkins and Dayan, 1992).

The primary advantage of MFRL is its simplicity and robustness, particularly in environments where modeling the dynamics is challenging. However, this simplicity comes at a cost: MFRL largely relies on data and often requires a large number of interactions with the environment to achieve good performance, which can be inefficient and time-consuming, particularly in complex or high-dimensional environments. Based on their properties, MFRL algorithms are often used in applications that we can build simulation environment easily, such as video game, Robotics, and autonomous systems.

Noteably, all the paper we mentioned in last section uses MFRL method, this also explain why they takes millions or billions of steps to train.

Model-based Reinforcement Learning

In contrast, Model-Based Reinforcement Learning (MBRL) involves building a **environment model** for simulating future environment states and task-related rewards, enabling the agent to plan and optimize its actions more efficiently. By leveraging a learned model, MBRL can reduce the number of real-world interactions required, improving sample efficiency. This is particularly advantageous in scenarios where interactions are costly or limited, such as in energy management or healthcare applications.

Noteably, some model-based RL algorithm can be viewed as an simple intergration of an environment model with model-free RL controller. For example, in the next section, we will talk about Dreamer, which uses an RSSM (Recurrent State-Space Model) as the environment model and an actor-critics controller(Hafner et al., 2019a). In this way, they leverage the advantage of both model-free and model-based reinforcement learning algorithms. However, there are also some other algorithms, such as PILCO, which purely rely on its environment model's prediction about the future state to find the minimal cost next action, which is more like directly predicting the next action(Deisenroth and Rasmussen, 2011). Comparing to other model-free LR or model-based LR that relys on model-free controler, pure model-based RL significantly reduced the training time (see Figure 2.6).

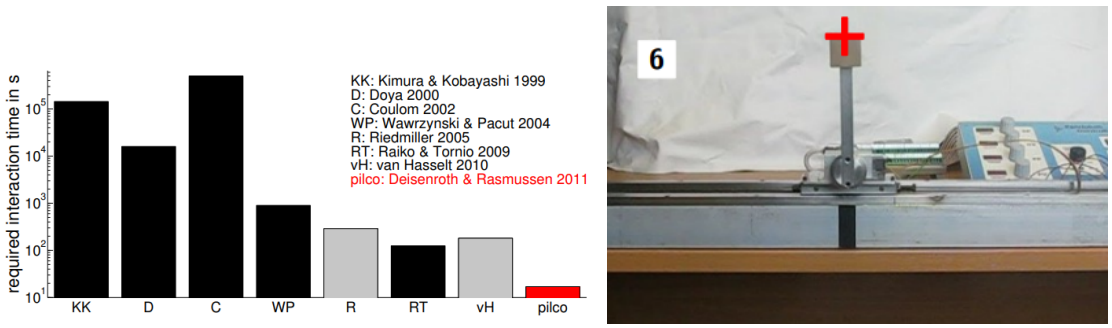


Fig. 2.6 **PILCO is a pure model-based RL algorithm**, which purely rely on its environment model's to predict the minimal cost next action. The right image shows that PILCO is training on a real-world cart-pole balancing task, which require the model to balance a hammer that can freely swing on a cart by controlling the cart's movement. PILCO learns to complete the task in just 20 seconds. The left histogram shows that PILCO's data efficiency comparing to other RL algorithms.

2.2.2 The Development of World Model

If MBRL is so efficient, why isn't it more widely adopted? There are several key reasons:

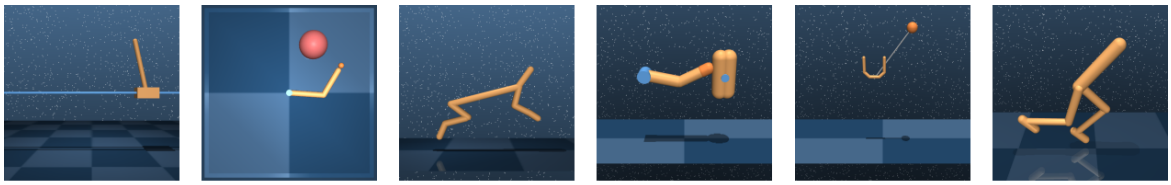
1. **Complexity of Environment Modeling**: Modeling complex, highly dynamic, or high-dimensional environments is inherently challenging. In such scenarios, model-free methods are often preferred because they can directly learn the mapping from states to actions.

2. **Accumulation of Model Errors**: Model-based RL heavily depends on the accuracy of its environment model. When the model fails to accurately represent the environment, the predictions of future states become unreliable. Furthermore, because model-based methods typically involve predicting multiple future steps (e.g. forecasting the next five time steps from the current state), any inaccuracies in the model can compound over time, leading to significant errors in the agent's decision-making process.

3. **Computational Complexity**: The inclusion of an environment model in model-based RL adds to the computational burden. For example, the environment model used in the PILCO algorithm employs Gaussian Processes, which have a computational complexity of $O(n^3)$. This makes model-based RL computationally intensive, often requiring significantly more time and resources to train compared to model-free approaches.

For decades, RL researchers have been dedicated to addressing the challenges of modeling complex environments. As early as the 1980s, foundational studies on feedforward neural networks (FFNs) emerged, demonstrating their powerful capabilities in approximating complex nonlinear relationships, making them effective for modeling dynamic systems (Munro, 1987; Nguyen and Widrow, 1990; Robinson and Fallside, 1989; Werbos, 1987, 1989). Building on this foundation, significant advancements in recurrent neural networks (RNNs) were made in the 1990s. RNNs evolved from FFNs by introducing recurrent connections with hidden states, enabling the network to utilize information from previous time steps. This architecture endowed RNNs with the ability to predict time-series data, laying the groundwork for environment models capable of forecasting future states(Schmidhuber, 1990a,b,c, 1991).

Building on traditional RNNs, Ha and Schmidhuber (2018) later proposed the concept of the **World Model**, which consists of a more sophisticated generative model known as the Mixture Density Network combined with an RNN (MDN-RNN). This model not only remembers and utilizes past observations but also predicts multiple future possibilities through probability distributions, effectively handling uncertainty and environmental randomness. With these advancements, the MDN-RNN surpass simple time-series predictions, leading to more stable and robust policy transfers in real-world scenarios.



(a) Cartpole (b) Reacher (c) Cheetah (d) Finger (e) Cup (f) Walker

Fig. 2.7 **PlaNet's Training Task**: a) balancing a pole on a moving cart, b) moving a robotic arm to reach the ball, c) controlling a simulated cheetah to run as fast as possible, d) rotating a spinning object using a robotic finger, e)catching a moving ball within a cup, and f)controlling a bipedal robot to walk (Hafner et al., 2019b).

After the concept of the World Model is proposed, more and more advance was made in this field. Hafner et al. (2019b) proposed the Deep Planning Network (PlaNet), which leverages a Recurrent State Space Model (RSSM) to predict environment dynamics. This model learns to separately predict the deterministic state and the stochastic state of the whole environment state. The deterministic state captures changes in the environment directly caused by the agent's actions, while the stochastic state accounts for factors beyond the agent's control. This dual structure enables the agent to better understand the relationship between its actions and subsequent environmental changes.

Compared to the previously mentioned MDN-RNN, the RSSM in PlaNet offers enhanced predictive capabilities, allowing for more accurate forecasting of future states over multiple time steps. The PlaNet model was evaluated using six tasks from the DeepMind Control Suite, a benchmark commonly used for assessing continuous control tasks in RL (see Figure 2.7). In these tasks, PlaNet demonstrated significant improvements in data efficiency and reduction of environment interactions compared to traditional model-free RL models. For example, the performance level that PlaNet achieved in 1,000 episodes is comparable with that achieved by D4PG in 100,000 episodes. This demonstrates PlaNet's superiority in efficiently learning policies that generalize well across various tasks with minimal environmental exposure.

DreamerV2

Another significant advancement in the area of World Models is the development of Dreamer V2. Given the ability of World Models to simulate environments and predict future states, researchers have long considered the potential of using these models to replace traditional simulation environments for training controllers, a concept initially proposed by Sutton (1990). Unlike traditional simulation environments, which accurately modeling the environment and precisely calculate the effects of actions, World Models—exemplified by the Dreamer series—leverage a technique known as latent imagination. This approach directly

predicts the latent embedding of the environment based on the agent's actions, bypassing the need for explicit environmental modeling. The potential advantage of latent imagination is the significant acceleration of the training process if a controller can be trained entirely within a World Model. However, before Dreamer V2, the accuracy of World Models in simulating environments was insufficient for standalone training of controllers. As a result, controllers were typically trained using a combination of both World Models and traditional simulation environments (Sutton, 1990).

Dreamer V2 represents a breakthrough in this domain.Through several architectural improvements, such as the introduction of discrete latent variables, improved KL divergence balancing, and other enhancements, Dreamer V2 has achieved the capability to train controllers solely within the World Model, without the need for external simulation environments. This development marks a significant leap forward in the capability of World Models to simulate environments with sufficient accuracy to support independent training of controllers, thereby advancing the field of RL.

2.3 Encoders for Feature Extraction

In the previous chapter, we explored a key strategy to enhance a robotics agent's ability to adapt to real-world applications with limited inputs and complicated environments: improving the model's capacity to predict changes in observations following specific actions. This approach is central to MBRL, which has evolved significantly over the past decades. As these models have grown more proficient at accurately simulating environmental changes, the concept of the 'World Model' has emerged as a pivotal advancement in this domain.

One crucial factor in enhancing the World Model's ability to predict future outcomes is training the environment encoder to extract richer information from limited sensory inputs. By obtaining environment embeddings that contain more comprehensive data, the World Model can better understand the relationship between actions and environmental changes during training. For instance, depth information provides more insights into a robot's movements compared to simple RGB data. In recent years, particularly with the advent of transformers and their associated encoder architectures, substantial progress has been made in this area. Transformer-based models, known for their robust feature representation and generalization capabilities, offer significant advantages over traditional regression models and RNN models.

Given the two primary challenges faced by robotics in real-world tasks—extracting detailed environmental information from minimal inputs and managing out-of-distribution scenarios—

transformer-based encoders are emerging as promising tools for environment encoding in robotic agents. In this chapter, we will begin by discussing the introduction of transformers, their evolution in different modality, and various training methodologies designed to enhance the semantic information extraction capabilities of transformer-based encoders.

2.3.1 Transformer-based Feature Extraction

Before the advent of the Transformer, feature extraction in natural language processing (NLP) and computer vision primarily relied on Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs). RNNs, including their variants such as Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU), are designed to capture temporal dependencies in sequential data through their recurrent structure (Hochreiter and Schmidhuber, 1997). These networks perform particularly well in tasks involving short texts or data with clear temporal dependencies. However, the sequential nature of RNN processing imposes limitations on computational efficiency, and RNNs often struggle with issues such as gradient vanishing when dealing with long-range dependencies.

CNNs, on the other hand, excel in extracting local features from images by progressively applying convolutional layers (LeCun et al., 1998). This hierarchical approach allows CNNs to move from detecting simple edges to complex shapes and eventually to global concepts, making them highly effective in image classification, object detection, and other vision tasks. However, CNNs have inherent limitations in capturing global context, particularly in scenarios that require an understanding of long-range dependencies within images.

The introduction of the Transformer marked a significant shift in feature extraction techniques. The Transformer's unique self-attention and multi-head attention mechanisms enable it to efficiently capture global dependencies and represent multi-dimensional features. The **self-attention mechanism** in the Transformer model processes input sequences by splitting them into tokens and adding positional embeddings to capture the sequential or spacial nature of the data. Self-attention calculates the relevance of each token within itself and across the entire sequence, allowing the model to effectively handle long-range dependencies. Additionally, teniques like layer normalization and residual connection are also utilized in self-attention, effectively preventing vanishing gradients and stabilizing deep network training.The **multi-head attention mechanism** further enhances this process by projecting the input data into multiple subspaces simultaneously. Each attention head focuses on different aspects of the sequence, allowing the model to extract a richer set of features and better capture global contextual information (Vaswani et al., 2017).

These structural advantages have allowed the Transformer to significantly outperform traditional models in various feature extraction tasks, leading to a revolutionary advancement in the fields of NLP and computer vision.

2.3.2 Large Language Model

After the formal introduction of the Transformer architecture, OpenAI released GPT (Generative Pretrained Transformer) in 2018, marking the first Transformer-based large language model. This paper compares GPT with several bi-LSTM-based models, such as ESIM and CAFE (Chen et al., 2016; Tay et al., 2018). Notably, GPT was initially trained on a large-scale corpus using a next-word prediction task for self-supervised pretraining, followed by fine-tuning on smaller datasets for various specific tasks. The GPT model trained in this manner significantly outperformed bi-LSTM models specifically crafted for each task in 9 out of the 12 tasks examined (Radford et al., 2018). The study highlights that the Transformer architecture provides a robust structured memory, which is more effective at handling long-term dependencies in text compared to traditional LSTM-based architectures. This example also underscores the Transformer's strong general feature extraction capability, where the features extracted by the pre-trained Transformer model encompass the information required for a wide range of tasks.

BERT (Bidirectional Encoder Representations from Transformers) is another prominent example of the Transformer model and serves as one of the backbones of our architecture. What distinguishes BERT is its bidirectional encoding and autoencoding training methods. During BERT's training process, the model learns bidirectional contextual information by predicting randomly masked words using a Masked Language Model (MLM). In contrast, GPT employs an autoregressive approach, generating text exclusively in a left-to-right sequence. Additionally, BERT incorporates a Next Sentence Prediction (NSP) task, which is trained in a contrastive learning framework. This task enhances the model's contextual understanding by determining whether two sentences are adjacent. These unique training strategies enable BERT to outperform GPT in many NLP tasks (Devlin et al., 2019).

2.3.3 Vision Transformer

In 2021, researchers introduced the Vision Transformer (ViT), marking a significant innovation in the field of computer vision (CV). ViT applies a pure Transformer architecture to image data, dividing images into patches and processing them as sequences, akin to how words are handled in natural language processing (NLP). The application of this approach was delayed in CV due to the initial dominance of convolutional neural networks (CNNs),

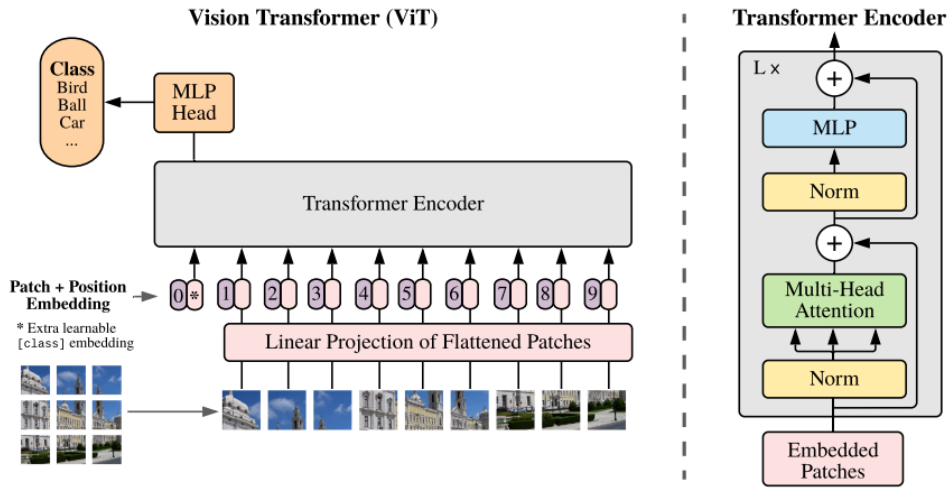


Fig. 2.8 **ViT Model Overview.** ViT will split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. A MLP head is at the end of the pipeline for classification task (Dosovitskiy et al., 2021)

which effectively exploited spatial hierarchies in images. Transformers, initially designed for sequential data, were not straightforward to apply to the inherently two-dimensional nature of images. Additionally, the computational cost of applying attention mechanisms to every pixel in an image posed a significant barrier until advances in hardware and the availability of large-scale datasets made this feasible (Dosovitskiy et al., 2021).

ViTs demonstrate competitive performance compared to traditional CNNs, such as ResNet, and newer architectures like EfficientNet. When pre-trained on large datasets, such as ImageNet-21k or JFT-300M, and fine-tuned on task-specific datasets like CIFAR-10 or Oxford-IIIT Pets, ViTs often surpass state-of-the-art CNNs across various benchmarks, achieving notable results like 88.55% accuracy on ImageNet.

The success of Vision Transformers (ViTs) in computer vision highlights their effectiveness as encoders for feature extraction. ViTs utilize a two-stage training process: pre-training and fine-tuning. In the **pre-training stage**, ViTs process images as sequences of patches, learning to recognize and encode visual structures through supervised learning on diverse datasets. This stage allows the model to develop rich, multi-dimensional visual representations that serve as a robust foundation for subsequent tasks.

During the **fine-tuning stage**, the pre-trained ViT model is adapted to specific tasks using smaller, targeted datasets. This approach enables the model to transfer its broad visual

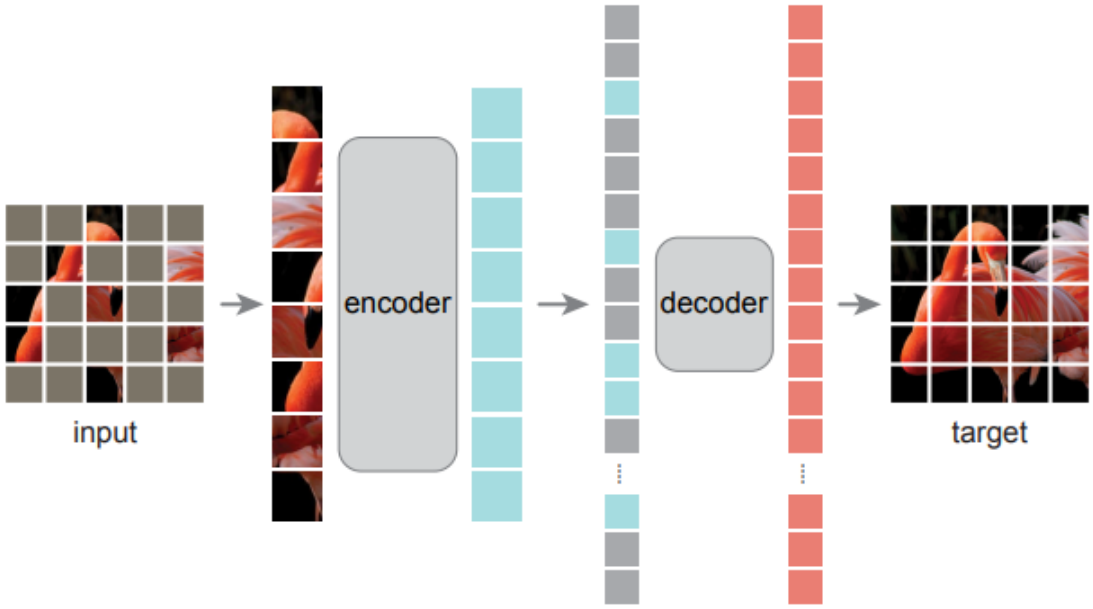


Fig. 2.9 **MAE Architecture.** The masked auto-encoder training method involving masking 75% of the patches at the beginning use the remaining subset of patches to train the ViT model in reconstruction task (He et al., 2022).

knowledge to specialized applications, achieving high performance with minimal additional training. The ability of ViTs to model long-range dependencies within images, combined with their scalable training process, positions them as powerful encoders not only for computer vision but also for other domains requiring robust feature extraction from complex data.

2.3.4 Masked Autoencoders

Before introducing Masked Autoencoders (MAE), we need to introduce auto-encoder first. Auto-encoders are a class of unsupervised learning models that aim to learn efficient representations of data, typically by compressing the input into a latent space and then reconstructing the input from this compact representation. These models are composed of two main parts: an encoder, which compresses the input into a lower-dimensional latent code, and a decoder, which reconstructs the original input from this code. Auto-encoders have been widely used in various applications, including dimensionality reduction, anomaly detection, and generative modeling, demonstrating their versatility in capturing the underlying structure of data (Hinton and Salakhutdinov, 2006).

MAE leverage the Vision Transformer (ViT) architecture and the auto-encoder training method in a novel self-supervised training approach inspired by the BERT model from

natural language processing. In this method, images are divided into patches, and a significant portion of these patches are masked out before being processed by Transformer blocks. The model is then trained to reconstruct the missing patches, effectively learning to predict the missing content based on the context provided by the visible patches. This process compels the model to focus on capturing meaningful, robust features rather than relying on redundant or superficial patterns present in the data.

The MAE training approach results in a model with exceptional semantic feature extraction capabilities. By forcing the model to reconstruct masked portions of the input, MAE develops a deep and holistic understanding of image content, capturing both global context and fine-grained details. This capability is reflected in its superior performance across a range of vision tasks, such as image classification, object detection, and semantic segmentation. For instance, MAE has achieved groundbreaking results on ImageNet-1K, with an accuracy of 87.8%, surpassing previous models trained solely on this dataset. The richness of the features learned through this process makes MAE particularly effective in tasks requiring a comprehensive understanding of the visual data.

Such success in learning robust semantic features can certainly be borrowed to train successful environment encoder. By employing a training methodology akin to MAE's masked reconstruction, our environment encoder can be trained to capture a comprehensive representation of the environment. This enriched environment embedding, in turn, allows the World Model to better predict environmental dynamics and adapt to diverse, unseen scenarios. The methodology presented in the following chapter details how the LanGWM pipeline incorporates these insights, using a multi-modal encoder inspired by MAE's architecture to achieve superior performance in navigation tasks.

2.4 Conclusion

In this background section, we first explored the challenges and advancements in robotic indoor navigation, with a particular focus on out-of-distribution (OOD) PointGoal Navigation tasks. We discussed how RL, particularly DRL, has been applied to these tasks, highlighting the critical need for robust environment embeddings and efficient environment encoders. These elements are essential for enabling robots to navigate unfamiliar environments, aligning directly with the project's goal of improving navigation performance through enhanced environmental understanding.

Next, we delved into the concept of World Models, tracing their evolution from traditional MBRL. We emphasized how World Models enable agents to internally simulate and predict

environmental dynamics, offering a significant advantage over model-free approaches in various tasks. However, we also noted the current limitations of these models, particularly their challenges in adapting to new and unfamiliar settings, which is closely related to our research focus on enhancing the semantic understanding of environments within World Models.

Finally, we examined the role of encoder architectures, particularly transformer-based models and Masked Autoencoders (MAE), in feature extraction. These advancements are crucial for training environment encoders capable of capturing rich, multi-modal information, which is essential for improving the predictive accuracy and adaptability of World Models. By integrating these insights, the background section sets the stage for our project's investigation into refining the LanGWM framework, with the ultimate goal of enabling robots to better understand and navigate complex, real-world environments.

Chapter 3

Methodology

3.1 Language Grounded World Model

Our work extends the baseline LanGWM: Language Grounded World Model (Poudel et al., 2023). The baseline model is based on the assumption that explicitly training the environment encoder to interpret visual observations through language can lead to a more comprehensive environment embedding. To validate this assumption, Poudel et al. (2023) developed the LanGWM pipeline, illustrated in Figure 3.1. This pipeline comprises three major components:

1. a language-grounded representation learning process for training the environment encoder,
2. a world model for action planning,
3. an actor-critic controller for robotic control.

The model is trained on an out-of-distribution PointGoal navigation task within the iGibson 1.0 simulation environment. Essentially, Poudel et al. (2023) investigates a novel training method aimed at enhancing the environment encoder, which in turn improves both the world model and the controller.

Since our work builds directly on the baseline architecture, in the subsequent sections, we will provide a detailed explanation of the pipeline, the training and testing procedures, the results from the baseline model, and its limitations.

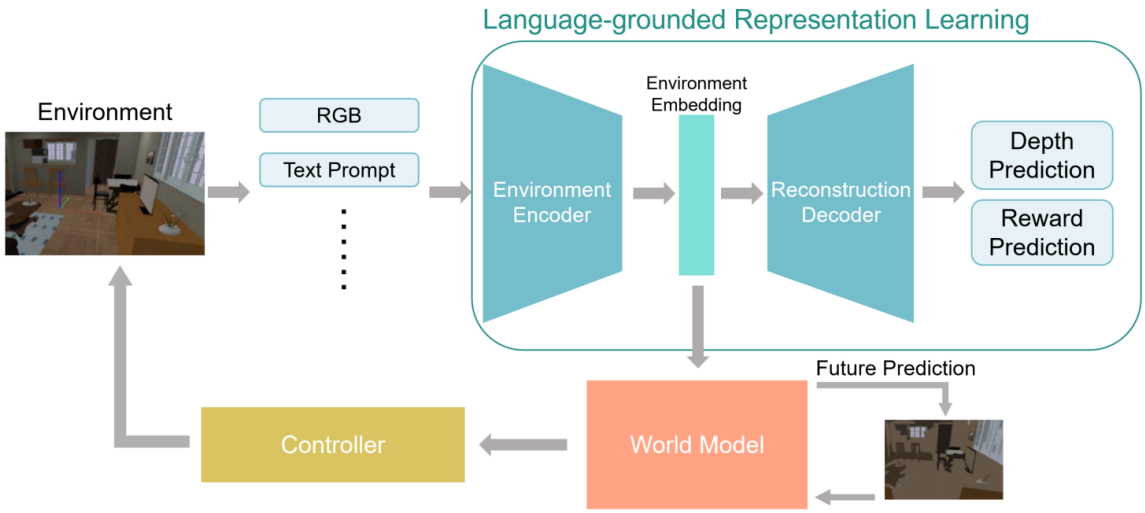


Fig. 3.1 **LanGWM Pipeline.** The LanGWM pipeline consists of three main components: 1) language-grounded representation learning, 2) the world model, and 3) the controller. The simulation environment first provides observations to the environment encoder, which generates an environment embedding and passes it to the world model. The world model then integrates the predicted future state with the current state and outputs the state features to the controller. The controller, in turn, outputs actions to the simulated environment. This process continues iteratively until the robot reaches its goal or a time restriction is triggered.

3.1.1 Architecture

Language-grounded Representation Learning

The detailed pipeline of the language-grounded representation learning is illustrated in Figure 3.3. It consists of two main components: the task observation auto-encoder and the vision encoder auto-encoder. Both auto-encoders are designed to train the environment information encoder.

The task observation encoder is responsible for encoding all task-specific non-sensory observations, including the robot's current location, relative destination, and motor velocities. It first receives `task_obs` from the simulation environment, encodes it using a multi-layer perceptron (MLP) encoder, and generates the task observation embedding. This embedding is then passed through an MLP decoder to reconstruct the original task observation, and a mean square error (MSE) loss is calculated to assess the accuracy of the reconstruction. The primary objective of this task observation encoder is to expand the feature dimension of the task observation so it can be concatenated with the vision embedding in subsequent processes. It is crucial to ensure that, after dimensionality expansion, the task observation embedding still retains the original information. If the MLP decoder can successfully reconstruct the task

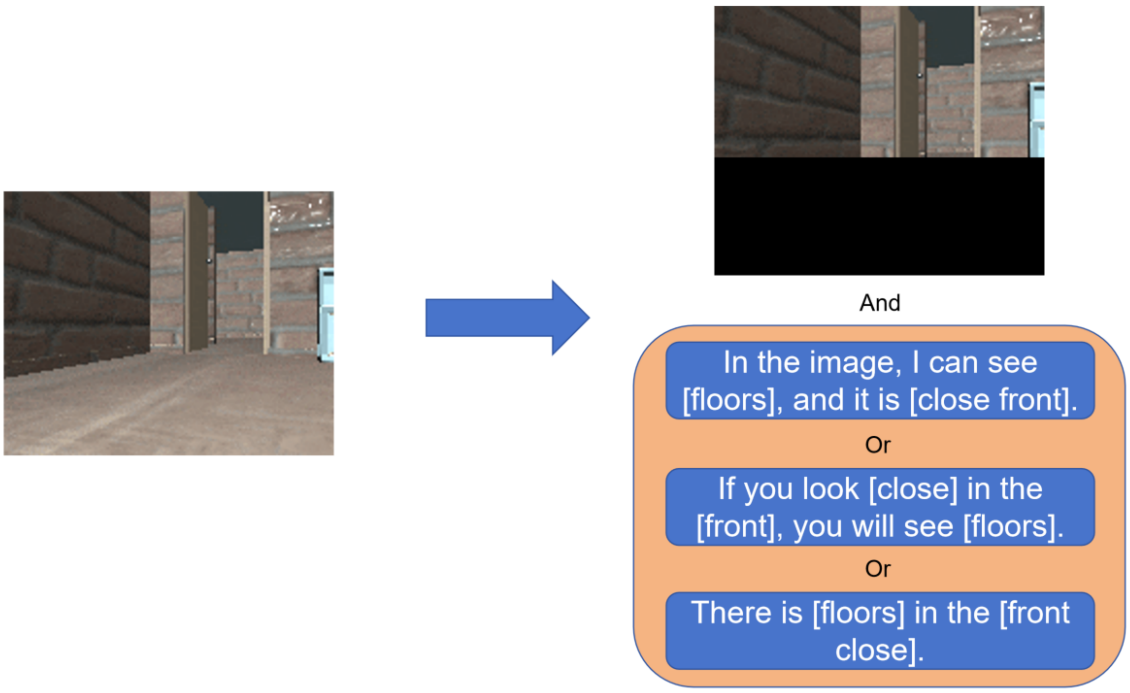


Fig. 3.2 **Object Masking and Language Prompt Generation.** During the object masking stage, a random object within the RGB observation is selected. A rectangular mask is then applied to the object and its surrounding area. Simultaneously, a language prompt is generated using a randomly selected template. The depth information is utilized to determine the object's location, and the object's name and location are inserted into the template to create the complete prompt. This procedure is repeated until the masked area covers more than 25% of the observations.

observation, it indicates that the MLP encoder has effectively captured the task observation's information during encoding. The MLP encoder and MLP decoder are updated solely based on the reconstruction of MSE loss.

The vision encoder forms the other component of the environment encoder and is crucial for validating our assumption. Upon receiving the RGB image and object segmentation map from the simulation environment, an object masking module in the baseline model randomly masks objects and their surrounding areas on the RGB images. This module then generates a paragraph of language description about the masked objects (see Figure 3.2). The primary objective of object masking is to remove specific objects' visual information and replace it with related language information. In subsequent steps, the baseline model is forced to rely on this language information to infer what is missing in the masked RGB images.

Afterwards, the language description (or text prompt in Figure 3.3) is processed by a pre-trained BERT model to generate language embeddings. For vision embeddings, the masked

RGB image is passed through a CNN to produce visual embeddings. Subsequently, an MAE masking is applied, removing 75% of the visual tokens. The remaining 25% of visual tokens are concatenated with the language embeddings and passed through the ViT encoder, where the language and visual information are fused. The LanGWM pipeline then truncates the language tokens from the ViT encoder's output. According to Poudel et al. (2023), this truncation ensures that the subsequent reconstruction does not rely on any language tokens, thereby forcing the ViT encoder to more effectively integrate language information with visual information.

After truncation, the remaining embedding is called the "visual environment embedding," which is then sent to a ViT decoder. Unlike the standard MAE approach, LanGWM does not aim to reconstruct the RGB image. Instead, it focuses on reconstructing the scene's depth and the reward from the last action. By doing this, Poudel et al. (2023) intends for the ViT encoder to learn more task-related semantic representations rather than focusing on RGB details. Finally, the MSE loss is calculated for both the depth and reward reconstructions, and this loss is used to update the CNN, ViT encoder, and ViT decoder.

The final output of the environment encoder, referred to as the environment embedding, is produced by concatenating the task observation embedding and the visual environment embedding.

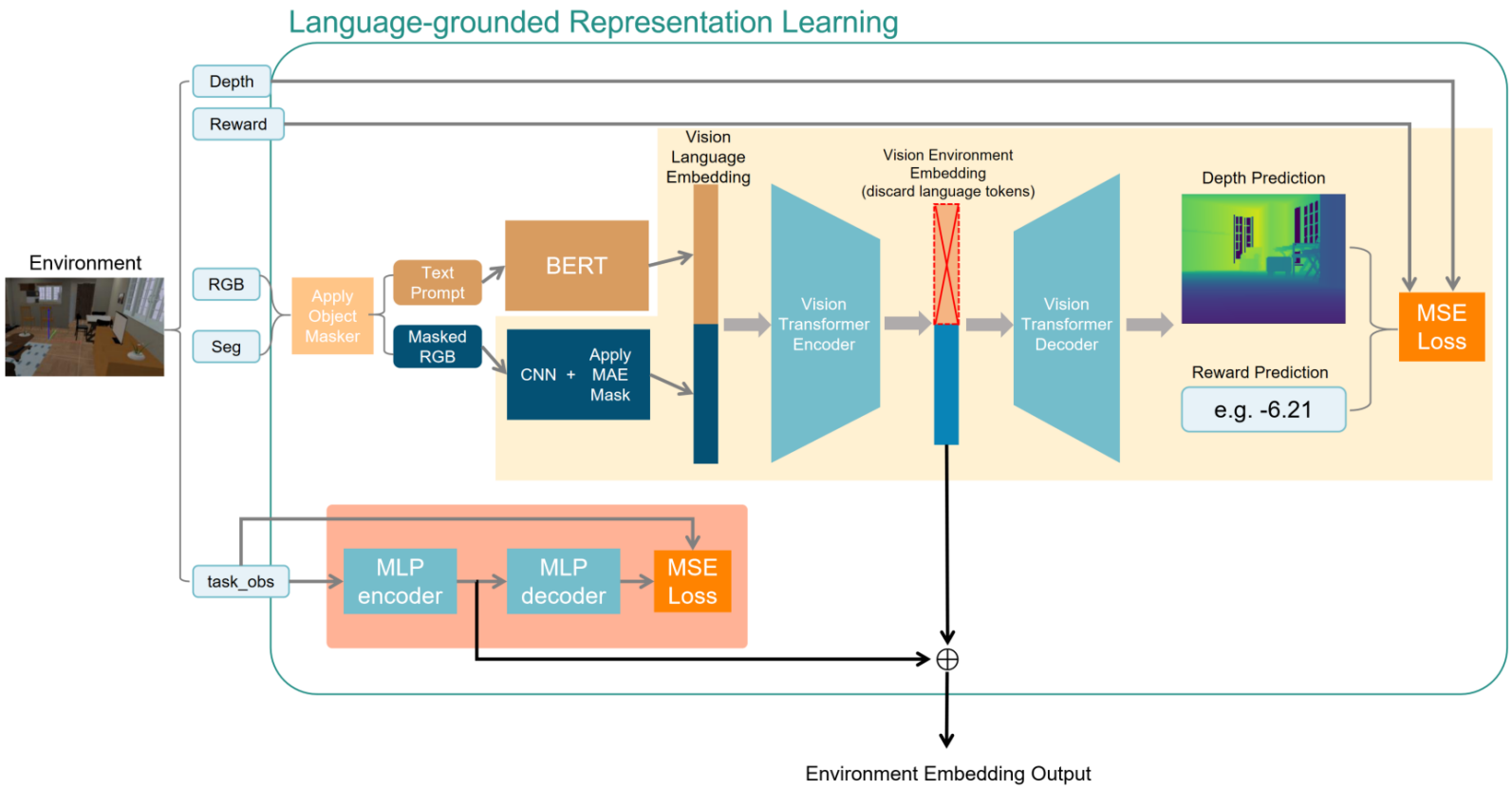


Fig. 3.3 **Language-grounded Representation Learning.** The language-grounded representation learning process consists of two main components: the task observation encoder and the vision encoder. The task observation encoder takes task observations from the simulation environment as input and utilises an auto-encoder approach for training. The network updated by the loss is covered in the red area. For the vision encoder, the segmentation map and RGB image are first preprocessed to generate an object-masked image and the associated language prompt. The language embeddings and the MAE masked vision embedding are then generated. The unmasked visual tokens and language embeddings are concatenated and passed through a ViT encoder. After this, the language tokens are truncated, leaving the visual environment embedding, which is used for depth reconstruction and reward prediction. The auto-encoder loss is then calculated to update all networks within the yellow area. Finally, the visual environment embedding is concatenated with the task observation embedding to form the final environment embedding.

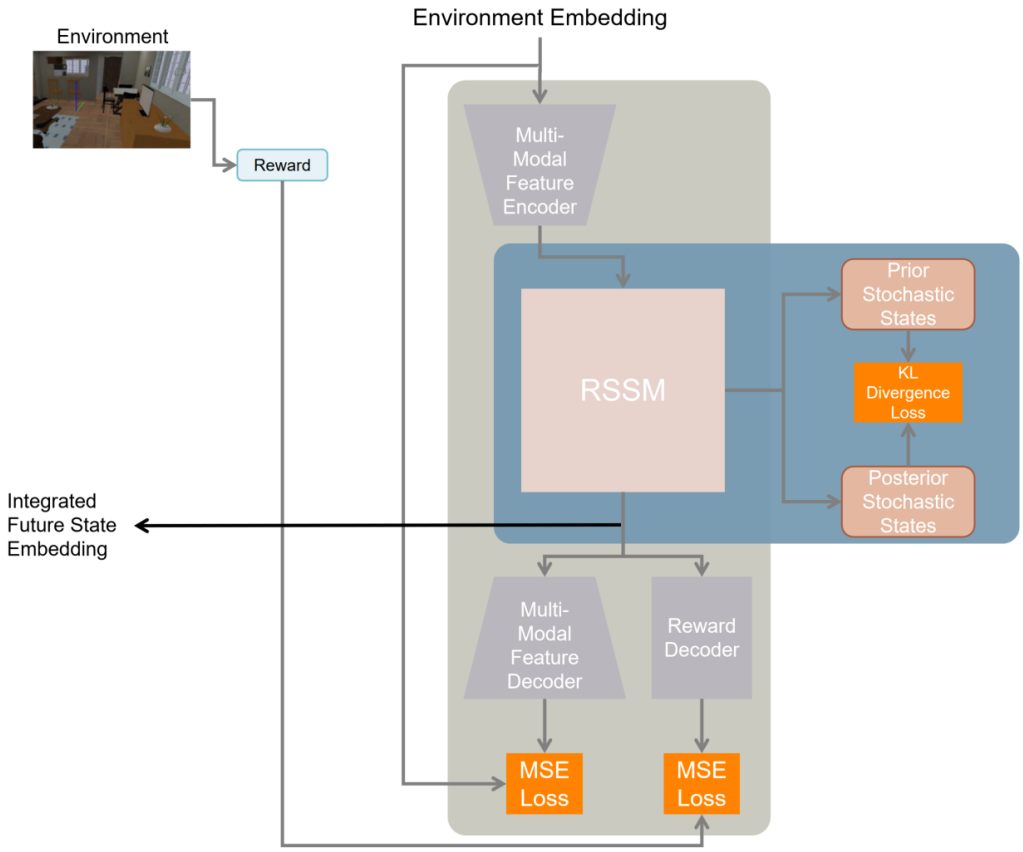


Fig. 3.4 **LanGWM World Model.** The world model in LanGWM is tasked with predicting future states and providing output features to the subsequent controller. Initially, the environment embedding is processed through a multi-modal feature encoder, which compresses the embedding. This compressed embedding is then passed to the RSSM for future state prediction. A KL divergence loss is computed between the prior and posterior stochastic states, which is used to update the RSSM. The RSSM subsequently outputs an integrated future state embedding. This embedding is then utilized by both the multi-modal feature decoder head and the reward head to calculate their respective losses, which are used to update the networks involved in these processes (denoted by the grey area). Finally, the integrated future state embedding is passed to the controller.

World Model

The second component of the LanGWM architecture is the world model, which is responsible for future prediction. It receives the environment embedding and utilizes it to predict future states in the latent space. In the LanGWM, the main component of the world model is RSSM, similar to DreamerV2.

Given that the RSSM has limitations in predicting future states in large latent spaces, the baseline model incorporates a transformer-based multi-modal feature encoder to process the environment embedding, and compressed the embedding through an average pooling module.The RSSM uses these compressed embeddings to estimate prior and posterior stochastic states. The prior stochastic state represents the RSSM's prediction of future state randomness, while the posterior stochastic state integrates information from the actual RGB observation.

To evaluate the RSSM's ability to predict uncontrollable randomness in future states, a KL divergence loss is calculated between the prior and posterior states. This loss guides the update of the RSSM to improve its predictive accuracy. For further details on Dreamer V2, please refer to Section 2.2.2.

To ensure that the RSSM is learning the correct information, several decoder heads have been implemented. The RSSM will then output an integrated future state, which integrate the current environment embedding information and the predicted future information. Base on the integrated future state, LanGWM employs a reward decoder head to predict rewards and a transformer-based multi-modal feature decoder to reconstruct the environment embedding. MSE losses are calculated for these two decoder heads and are used to update the multi-modal feature encoder, RSSM, multi-modal feature decoder, and reward decoder.

The integrated future state embedding is then passed to the third component of the pipeline, the controller, which is responsible for executing actions.

Controller

The third component of the pipeline is an actor-critic controller, an MFRL model. It consists of two primary components: the actor, which determines the actions the robot should take, and the critic, which evaluates those actions by estimating the expected reward. The actor updates its policy based on feedback from the critic, which learns to predict the value of the current state-action pair using the real reward as a reference.

In the PointGoal navigation task, the reward is calculated based on the following principles:

- During each trial within an environment, the robot is initialized at a random starting position, and a destination point is randomly selected. The robot receives positive rewards for moving closer to the destination, with a significant reward granted upon successfully reaching the goal. Conversely, the robot is penalized with negative rewards if it collides with obstacles or moves further away from the goal. This reward structure

is designed to encourage the robot to navigate efficiently while avoiding collisions, thereby testing its ability to adapt its learned behaviours to new, unseen environments.

By balancing exploration and exploitation, the actor-critic framework enables the robot to learn and refine its movements over time, ultimately improving its ability to reach its destination. In this pipeline, LanGWM utilizes a standard actor-critic controller that receives the integrated future state embedding and predicts actions. These predicted actions are then sent to the simulation environment, iGibson 1.0, to generate the next set of observations. As the baseline model employs a conventional actor-critic setup, further details on this part are not discussed.

3.1.2 Training and Testing Skills

As we can see from the previous section, the baseline model is quite complex. Poudel et al. (2023) implemented specific training and testing procedures, along with various techniques, to ensure the effectiveness of the pipeline in supporting their assumptions. These procedures and techniques can be summarized as follows:

Isolating the Environment Encoder's Training

One of the most notable aspects of the baseline model is that most of the networks are updated independently. For instance, the Vision Transformer (ViT) encoder and decoder are not updated using any losses from the world model or controller but solely through their own auto-encoder loss. This approach by Poudel et al. (2023) aims to support the assumption that "explicitly training the environment encoder to interpret visual observations through language can lead to a more comprehensive environment embedding." By isolating the encoder's training, the authors ensure that its quality depends entirely on the training method and architecture.

Explicit Language Training

A key technique employed in this project is language-guided object mask reconstruction, which aims to enable the ViT encoder to efficiently learn the language grounding of objects. In comparison, large language models (e.g., GPT, BERT) and large vision encoders (e.g., Sora, MAE) typically contain millions or even billions of parameters, are trained on vast amounts of raw data, and rely on end-to-end learning. While these large models become very powerful through extensive training, they are not data-efficient and are impractical for deployment on small robots.

To demonstrate the efficiency of explicit training, Poudel et al. (2023) specifically designed the ViT encoder to be smaller than conventional vision encoders, thereby making it more suitable for resource-constrained environments. (i.e. the LanGWM ViT encoder only have 3 layers and 4 heads by default but the smallest MAE encoder has 6 layers and 8 heads)

However, a potential issue with explicit training is the train-test discrepancy: during testing, there will be no object masks or language guidance available. If the model becomes overly dependent on language guidance during training, its performance may degrade in the absence of such guidance. To address this concern, Poudel et al. (2023) included 25% of non-masking examples in the training dataset. This technique is intended to reduce the encoder's reliance on language guidance, improving its robustness during testing.

Out-of-distribution Testing

To evaluate whether the model learns a comprehensive environment embedding, the testing was conducted using OOD scenarios. In the OOD PointGoal Navigation task, our model was trained across five distinct simulation environments and subsequently tested in three entirely different environments. The critical challenge in these OOD tasks is that all elements within the testing environments—such as furniture, wallpapers, floor textures, and floor plans—differ entirely from those in the training environments. This stark contrast in environmental elements rigorously tests the ViT encoder's ability to generalize its encoding capabilities to unseen scenarios.

Given the limited object features present in the training datasets, if the ViT encoder can still generate effective embeddings in the testing environments, it would indicate that the model is leveraging the language grounding of objects to understand and represent the environment in a more generalized manner.

3.1.3 Baseline Performance

Poudel et al. (2023) conducted experiments to compare LanGWM with state-of-the-art (SOTA) RL algorithms (see Table 3.1). Although all models were trained for only 100k steps—a relatively small number for the PointGoal navigation task—LanGWM outperformed all other SOTA RL algorithms. Notably, LanGWM's performance was compared with *CURL*[1], *DreamerV2 + DA*, and *DreamerV2 + Grounding DINO*.

CURL (Contrastive Unsupervised Representations for Reinforcement Learning), a world model algorithm proposed in 2020, leverages contrastive learning to train its CNN-based

[1]

Models		**Ihlen_0_int**		**Ihlen_1_int**		**Rs_int**		**Env Avg**	
	Steps	**SR**	**SPL**	**SR**	**SPL**	**SR**	**SPL**	**SR**	**SPL**
RAD	100k	0.6	0.01	0.1	0.00	0.8	0.01	0.5	0.01
CURL	100k	8.0	**0.07**	0.6	**0.01**	5.4	0.05	4.7	**0.04**
DreamerV2	100k	1.8	0.01	0.6	0.01	1.7	0.01	1.3	0.01
DreamerV2 + DA	100k	7.3	0.05	1.6	0.01	7.7	0.05	5.5	0.04
MWM	100k	1.6	0.01	0.5	0.01	2.9	0.02	1.7	0.01
LanGWM	100k	**8.3**	0.05	**2.1**	0.01	**9.9**	**0.06**	**6.8**	0.04
LanGWM + Obj Mask + Empty Lang	100k	1.6	0.01	0.5	0.01	2.3	0.01	1.5	0.01
LanGWM - Obj Mask + Empty Lang	100k	1.6	0.01	0.5	0.01	3.0	0.02	1.7	0.01
DreamerV2 + Grounding DINO	100k	48.9	0.45	17.0	0.14	45.4	0.38	37.1	0.33

Table 3.1 This table shows the OOD generalization performances on iGibson 1.0 dataset for the PointGoal navigation task. SR and SPL are reported. Models have been trained on five scenes and tested on three held-out scenes. LanGWM outperforms state-of-the-art RL models CURL, RAD and DreamerV2 on 100k interactive steps. Even though data augmentation (DA) improves DreamerV2, the proposed language grounded features learning technique yields even better results (Poudel et al., 2023).

environment encoder, enabling it to extract high-level features from raw pixels. CURL's success supported its hypothesis: "If an agent learns a useful semantic representation from high-dimensional observations, control algorithms built on top of those representations should be significantly more data-efficient" (Laskin et al., 2020). By utilizing a transformer-based environment encoder and incorporating the depth reconstruction training task, LanGWM appears to surpass CURL in feature extraction capability.

For "DreamerV2 + DA"[2], the world model and controller are nearly identical to those used in LanGWM, but LanGWM still surpass "DreamerV2 + DA"'s performance. This controlled variable experiment allows us to observe the positive impact of language-grounded environment representation on model performance.

"DreamerV2 + Grounding DINO" refers to replacing the CNN visual encoder in DreamerV2 with a pre-trained Grounding DINO model. Grounding DINO is a transformer-based, pretrained open-set detection model[3], which also utilizes language grounding techniques, though without explicit training. The performance of this method significantly surpasses that of any previous SOTA RL methods, including LanGWM. As a result, Poudel et al. (2023) has identified making LanGWM comparable to "DreamerV2 + Grounding DINO" as a key area for future work.

[2]DA stands for *data augmentation*, which is used to train models to focus on important features and enhance their robustness. In other words, it improves the model's feature extraction capability

[3]An open-set detection model means that the detectable labels are not restricted to those used during training. In fact, Grounding DINO accepts free-form descriptions of objects, interprets the descriptions, and outputs the most relevant object bounding boxes in the image.

Poudel et al. (2023) have also done ablation studies: "LanGWM + Obj Mask + Empty Lang" and "LanGWM - Obj Mask + Empty Lang". The significant performance drop shows the importance of language guidance.

3.1.4 Limitation

Although LanGWM outperforms all the SOTA RL models, several limitations in the baseline have been identified:

Pixel-wise Reconstruction

Since language represents abstract global concepts, the environment encoder and decoder may struggle to reconstruct detailed depth information using language guidance. For example, a language prompt might describe the presence of a chair in the foreground, but such a description does not convey the detailed shape or specific features of the chair. Training the environment encoder-decoder to reconstruct the pixel-wise depth of a masked chair could cause the model to struggle with recognizing chairs in different styles, hindering its ability to generalize to unseen scenarios. Additionally, reconstructing the pixel-wise depth may not be necessary for the PointGoal Navigation task, but it requires a significant amount of network resources, which could also impede the network's ability to learn the mapping between RGB, language, and depth.

Lack of Ablation Studies on Many Architectural Components

The original LanGWM architecture is highly complex, involving the training of five networks from scratch. The baseline primarily focuses on investigating the impact of the environment encoder while treating other components as black boxes, under the assumption that an RL algorithm built on a more powerful environment encoder will inevitably perform better. However, some parts of the architecture seem to lack clear justification, and the baseline does not evaluate their effectiveness through ablation studies. This omission leaves uncertainty regarding the contribution of each component to the overall performance of the model.

3.2 Methods to Improve

In the previous section, which introduced the baseline, we discussed its strengths as well as some potential limitations. In this section, we will focus on how we plan to further improve LanGWM. We hypothesize that **the pixel-wise reconstruction of depth is the primary**

hindrance to LanGWM's performance. Therefore, we plan to incorporate more abstract semantic decoders into the ViT encoder, including a segmentation decoder and a detection decoder. These decoders focus on extracting classification information and the rough location of objects, which should alleviate the encoder's struggle with pixel-wise reconstruction.

3.2.1 Multi-decoder Implementation

Segmentation

The segmentation decoder in our pipeline consists of two major components: a transformer block and a multilayer perceptron (MLP) head. The transformer block receives embeddings from the encoder and refines the information by attending to various spatial and contextual relationships within the image. The refined tokens from the transformer block are passed to the MLP head, which outputs a tensor of shape [Batch size, Image Height, Image Width, Number of Classes], where in our case, the number of classes is 63. This means that for each input image, the decoder produces 63 single-channel images, with each channel representing the logit for each pixel belonging to one of the 63 possible classes.

The final segmentation prediction is obtained by applying a softmax function across these 63 channels and selecting the most likely class for each pixel. The segmentation task is optimized by calculating a cross-entropy loss between the predicted segmentation maps and the ground truth, which allows for updating both the ViT encoder and the segmentation decoder.

Unlike depth decoding tasks that require highly detailed reconstructions, the segmentation task focuses on identifying regions corresponding to different objects rather than on pixel-level details. This focus aligns more naturally with the level of detail provided by language prompts. By reducing the complexity of the prediction task, the segmentation decoder allows the encoder to concentrate on extracting more abstract, high-level semantic features from the observations. As a result, the encoder can learn a richer and more comprehensive representation of the environment, which is expected to enhance the performance of downstream tasks, such as RL, by providing a more robust and semantically meaningful understanding of the environment.

Detection

The detection decoder implemented in our pipeline refers to the architecture of the DETR (DEtection TRansformers) model, as illustrated in Figure 3.5. In this architecture, it is necessary to predefine the number of bounding boxes to be predicted. For our application,

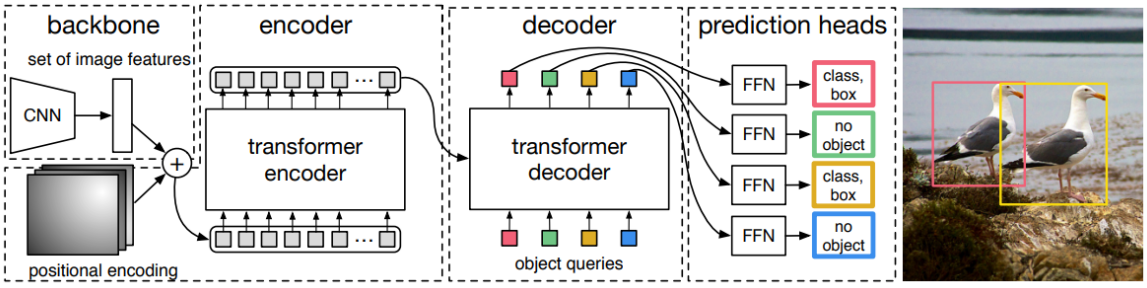


Fig. 3.5 **DETR Architecture.** The image illustrates the unique architecture of DETR. Our implementation adopts DETR's concept of object queries, which are learnable positional embeddings (Carion et al., 2020).

we predefined 30 bounding boxes, considering that the maximum number of objects within the robot camera's view typically does not exceed 20. Consequently, 30 learned positional embeddings, referred to as object queries, are generated. These object queries serve as the initial input for the decoder to attend to the visual environment embeddings generated by the ViT encoder.

The decoder processes these embeddings through multiple transformer layers, ultimately refining them into output embeddings, which are subsequently passed through a shared feed-forward network (FFN). This FFN is responsible for predicting either an object detection, which includes both the class and bounding box of an object, or a "no object" class, indicating the absence of an object for that particular query.

During the loss calculation phase, it is essential to match the predicted bounding boxes with the target boxes, as the predicted boxes are not output sequentially. The DETR model employs the Hungarian matching algorithm, which is an $O(n^3)$ algorithm that enforces permutation invariance and guarantees that each target element has a unique match. However, due to limitations in computational resources, we opted for a Nearest Neighbor Matching algorithm, which is an $O(n^2)$ algorithm that matches the predicted bounding boxes to the closest target bounding boxes.

Once the matching is complete, the label cross-entropy loss, GIoU loss[4], and L1 loss[5] are calculated to update the ViT encoder and detection decoder.

[4]GIoU Loss is an extension of the IoU metric, used to evaluate the overlap between predicted and ground truth bounding boxes in object detection. While IoU is calculated as $\frac{\text{Intersection Area}}{\text{Union Area}}$, GIoU improves upon IoU by considering the area of the smallest enclosing box that covers both the predicted and ground truth boxes. This penalizes cases where the boxes do not overlap, optimizing bounding box predictions more effectively, especially when boxes are not perfectly aligned.

[5]L1 Loss in object detection measures the absolute difference between the predicted and ground truth bounding box coordinates (e.g., centre, width, height). It is used to minimize discrepancies in location and size

By training the encoder to generate embeddings that are specifically useful for object detection, the decoder enhances the model's ability to capture higher-level semantic information, such as the presence and position of objects, rather than merely processing low-level pixel data. This approach ensures that the encoder learns to identify and represent essential features that are critical for comprehending the overall structure and context of the environment.

3.3 Conclusion

In this chapter, we have detailed the methodology behind the LanGWM, a sophisticated approach that leverages language-grounded representation learning to enhance the performance of environment encoders in RL tasks. By integrating language with visual observations, LanGWM creates a comprehensive environment embedding that has demonstrated superior performance in OOD navigation tasks when compared to other SOTA RL models. The methodology outlines the architecture of the model, including the environment encoder, world model, and controller, and highlights the unique training techniques employed to ensure the model's robustness and generalization capabilities.

Building upon the strengths of LanGWM, our approach addresses its limitations by introducing a multi-decoder strategy that includes segmentation and detection decoders. These decoders focus on abstracting semantic information rather than pixel-wise depth reconstruction, which we hypothesize to be a limiting factor in the original model. By refining the encoder's ability to capture high-level semantic features, our enhancements are expected to improve the model's ability to generalize across diverse and unseen environments, ultimately leading to more effective and resource-efficient RL outcomes.

between the predicted and actual bounding boxes. While L1 Loss is simple and effective, it does not directly consider the overlap between boxes.

Chapter 4

Experiments and Results

In this chapter, we will present the experimental section in two parts. The first part focuses on the most critical experiments, including the reproduction of baseline results, the evaluation of the effectiveness of the proposed decoders, and a pipeline analysis of LanGWM. It is important to note that some of our proposed methods yielded insignificant experimental results. However, through the experiments detailed in the first part, we gradually identified the underlying reasons for these outcomes.

The second part primarily addresses ablation studies that were not covered in the first part. These ablation studies were instrumental in examining and eliminating potential issues while highlighting the LanGWM framework's specific properties.

4.1 Experiment Setup

This section will introduce some experiment setups, including dataset, simulated robot, and detection data generation. Please check Section A for the pipeline configuration.

4.1.1 Dataset

This project utilised iGibson 1.0 as the simulation environment for the PointGoal Navigation task. Five interactive 3D scenes built into iGibson 1.0 were used: Beechwood_0_int, Benevolence_0_int, Merom_0_int, Pomaria_0_int, and Wainscott_0_int. Our pipeline controls a simulated robot to navigate within these scenes, collecting observations from the environment at each time step, including:

1. Task Observation: Robot's current location, relative destination, and motor velocity.

2. RGB observation: RGB three-channel image taken by robot's camera.
3. Depth observation: Depth map in the same shape as the RGB image, including the depth information of every pixel of the RGB image.
4. Instance segmentation map: The segmentation map of each object in the observation. Each pixel shows its correlated object ID.
5. Segmentation map: The segmentation map of each class of objects in the observation. Each pixel shows its correlated class ID.
6. Reward: The reward of the last action. The reward will be None if the scene has just been initiated
7. is_first: A boolean value showing whether this observation is the first observation after this run of the experiment is initiated.
8. is_last: A boolean value showing whether this observation is the last observation of this run. This could be caused by finishing the task or reaching the time limit.
9. is_terminated: A boolean value showing whether this observation is terminated because of reaching the time limit.

At each time step, a set of the above data is generated and temporarily stored in memory. Once a run is completed, the observations are transferred and saved to disk. During training, batches are formed by extracting 50 time steps from 16 different runs, which are then used for the training process.

During evaluation and testing, observations are not extracted from the saved data but generated in real time. The evaluation and testing phases utilise three different environments that share no common features with the training environments: Ihlen_0_int, Ihlen_1_int, and Rs_int.

4.1.2 Robot selection

For our experiments, we have kept the baseline selection of the TurtleBot as the agent. This choice is motivated by several factors:

- **Ease of Training**: The TurtleBot has only 2 DOF, making it one of the simplest robots to control and train. Its simplicity reduces the complexity of the learning problem, allowing us to focus more on the development and evaluation of the environment encoder.

- **Compact Size**: With dimensions of 354 x 354 x 420 mm, the TurtleBot is the most miniature robot implemented in iGibson 1.0. This compact size makes it more manoeuvrable in tight spaces, which is advantageous for the PointGoal Navigation task. More giant robots might struggle to navigate through narrow passages or cluttered environments where the destination point could be difficult to reach.
- **Focus on Environment Encoding**: Larger, more complex robots often have arms or other appendages that can appear in the camera's field of view, introducing additional variables into the perception process. While handling such complexities is a common challenge in robotics, for this study, we aim to isolate the performance of the environment encoder and the PointGoal navigation task. By using the TurtleBot, we can avoid the complications that arise from self-occlusions or extraneous visual data, allowing us to maintain a more explicit focus on the core objectives of our research.

4.1.3 Detection Data Generation

As the simulation does not provide the detection-related data, we implemented the following procedures to generate the bounding boxes and label ground truth:

1. Extract all unique object IDs from the instance segmentation map.
2. For each object ID, determine the minimum x, minimum y, maximum x, and maximum y coordinates in the instance segmentation map.
3. Calculate the *x_center*, *y_center*, width, and height of the object, and save these values as the bounding box ground truth.
4. Identify the corresponding area in the segmentation map and extract the associated class label for each object, saving these as the class label ground truth.

These four steps are repeated for each time step to generate the detection data. Figure 4.1 shows a simple visualisation of this process. Saving the bounding box in the format [*x_center*, *y_center*, width, height] facilitates Nearest Neighbor Matching, as the distance between bounding boxes can be easily calculated using the distances between [*x_center*, *y_center*].

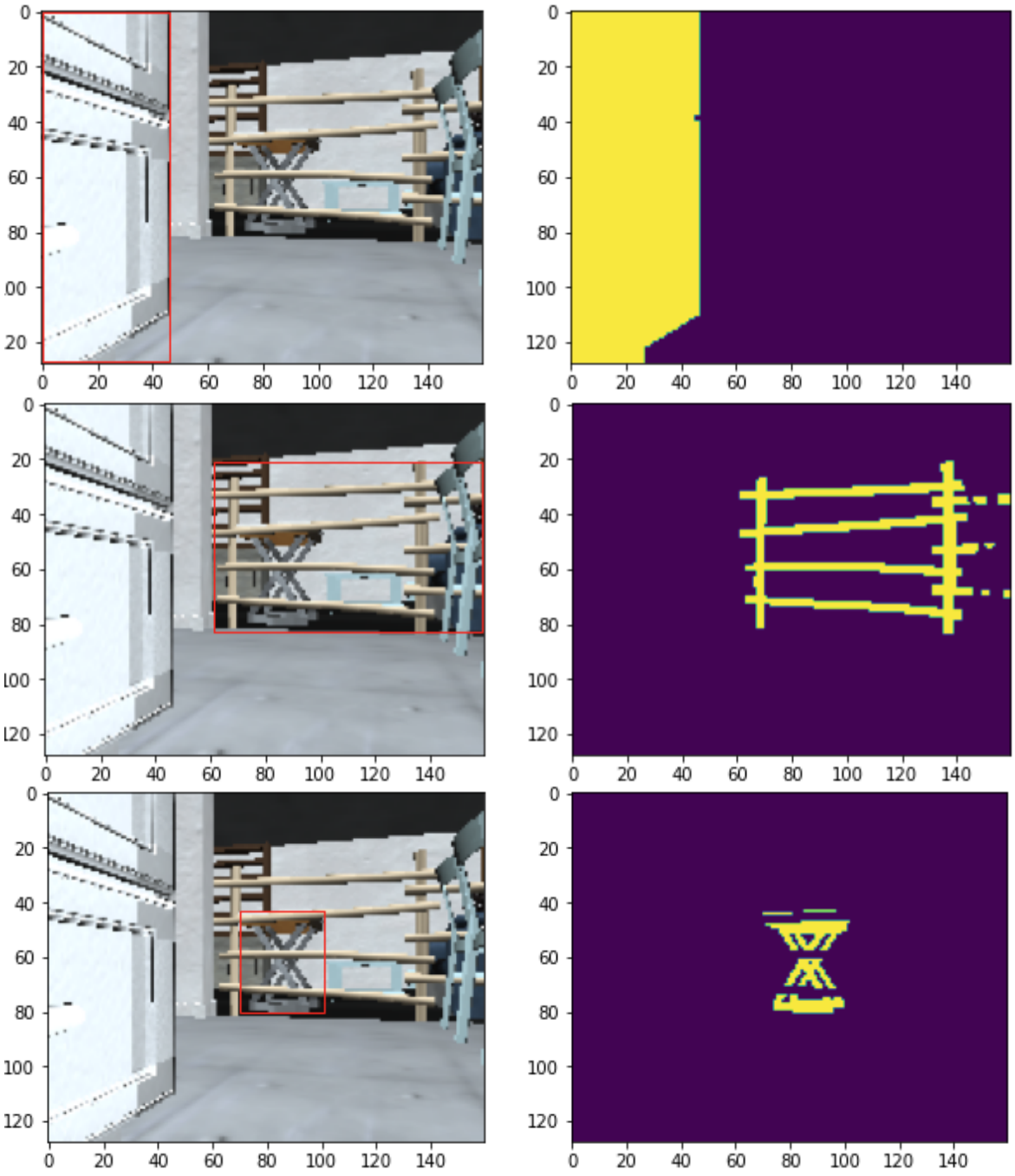


Fig. 4.1 **Detection Data Generation.** Generating detection bounding boxes based on segmentation data

4.2 Main Experiments

4.2.1 Baseline Replication

The first step of the main experiment is to reproduce the baseline result. After receiving the baseline code from Poudel et al. (2023), three reproducing attempts were made with the same setting. Training the baseline model for one time, which contains one depth decoder head and one reward decoder head, requires about 12 hours of training on a single A100 GPU.

As shown in Table 4.1, the first two attempts not only failed to reproduce the baseline result but also achieved a lower success rate than other RL algorithms (see Table 3.1). The third attempt, however, outperformed the baseline result. After consulting Poudel et al. (2023), we received confirmation that such randomness is normal. However, as we don't want our experiment to be affected by such randomness, We trained the baseline model again with the random seed fixed to be 1. Although the model performance after fixing the random seed is lower than the baseline result, we will still treat this result as an experimental baseline.

Models		**Ihlen_0_int**		**Ihlen_1_int**		**Rs_int**		**Env Avg**	
	Steps	**SR**	**SPL**	**SR**	**SPL**	**SR**	**SPL**	**SR**	**SPL**
Baseline Result	100k	**8.3**	**0.05**	2.1	0.01	9.9	**0.06**	6.8	**0.04**
Reproducing Attempt 1	100k	2.0	0.02	3.2	**0.02**	6.4	0.04	3.9	0.03
Reproducing Attempt 2	100k	1.4	0.01	1.1	0.01	2.5	0.02	1.7	0.01
Reproducing Attempt 3	100k	7.9	0.04	**3.3**	0.02	**11.2**	0.05	**7.4**	0.04
Reproducing Attempt 4 (fixed random seed)	100k	2.0	0.02	3.2	0.02	6.4	0.04	3.9	0.03

Table 4.1 **Results of reproducing the baseline model performance as reported in Poudel et al. (2023).** The table compares the SR and SPL across three reproduction attempts and one attempt with a fixed random seed.

4.2.2 Segmentation Decoder Experiment

Although depth information is highly valuable for navigation tasks, there is a significant issue in our pipeline: restoring the depth information of an object from a masked RGB image based on language prompts may be overly challenging, as language does not inherently convey depth information. If the model is forced to predict depth accurately on the training set, it may overfit to the training data and fail to generalise to OOD test sets.

To address this, we propose introducing an additional loss to train the encoder to capture more general environmental information. The first approach we are exploring is segmentation. Compared to depth prediction, segmentation prediction requires only a rough understanding of where an object is located in the RGB input. By training the model on segmentation, we

expect it to better understand the relationship between the location of objects and navigable paths. An environment embedding that includes this information should enable the World Model module to make more effective action planning.

Building on the original pipeline, we implemented a new Segmentation decoder. Like the Depth decoder head, the Segmentation decoder head receives the same masked visual environment embedding and reconstructs the segmentation of the entire image. With the addition of the segmentation decoder, the training process now requires approximately 15 hours on an A100 GPU.

Figure 4.2 shows that all three decoder heads are learning but have not yet converged. Figure 4.3 provides examples of depth and segmentation predictions, including both successful and unsuccessful cases. These examples show that while the depth prediction is not precise, it generally aligns with the ground truth. The segmentation decoder performs well in predicting large objects such as walls, floors, and bookshelves. However, it struggles with interpreting overlapping objects.

Our empirical understanding is that the segmentation decoder is particularly effective at identifying walls and floors because these are the most common objects in the scenes, allowing the decoder to learn these concepts better. Given that the encoder's output now includes information for three different semantic prediction tasks (i.e. depth, segmentation, and reward), we conclude that the addition of the segmentation decoder has made the ViT encoder more comprehensive in its representations.

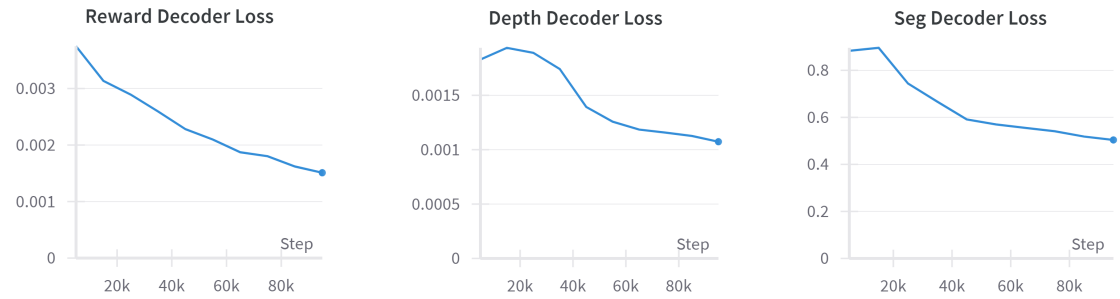


Fig. 4.2 **(Depth+Segmentation) Decoders Training Loss.** The training losses for all three decoders are converging, indicating that the ViT encoder and decoders have successfully learned.

However, as shown in Table 4.2, the model's performance in the PointGoal Navigation task declined after the segmentation decoder was implemented. The consistent performance drop

Models		**Ihlen_0_int**		**Ihlen_1_int**		**Rs_int**		**Env Avg**	
	Steps	**SR**	**SPL**	**SR**	**SPL**	**SR**	**SPL**	**SR**	**SPL**
Experimental Baseline	100k	**2.0**	**0.02**	**3.2**	**0.02**	**6.4**	**0.04**	**3.9**	**0.03**
LanGWM (depth+seg)	100k	1.6	0.01	0.8	0.00	1.6	0.01	1.3	0.01

Table 4.2 **(Depth+Segmentation) Decoder Experiment.** The experimental baseline consistently performs across different environments, while the LanGWM model with added segmentation demonstrates lower SR and SPL values.

across all three testing environments suggests that this is not simply a training failure due to randomness. We propose two empirical assumptions to explain this phenomenon:

1. Although segmentation prediction is more abstract than depth prediction, it is still pixel-wise. Reconstructing masked areas in pixel space may be too complex, especially given the 75% MAE masking area and the difference in input-output modalities. Training the ViT encoder with such a challenging task may increase the convergence difficulty, leading to the observed performance drop.

2. The information provided by segmentation may not benefit the PointGoal Navigation task. If the environment encoder includes irrelevant information in the embedding, it becomes more challenging for the world model and controller to extract the relevant information. Consequently, this could result in worse overall performance.

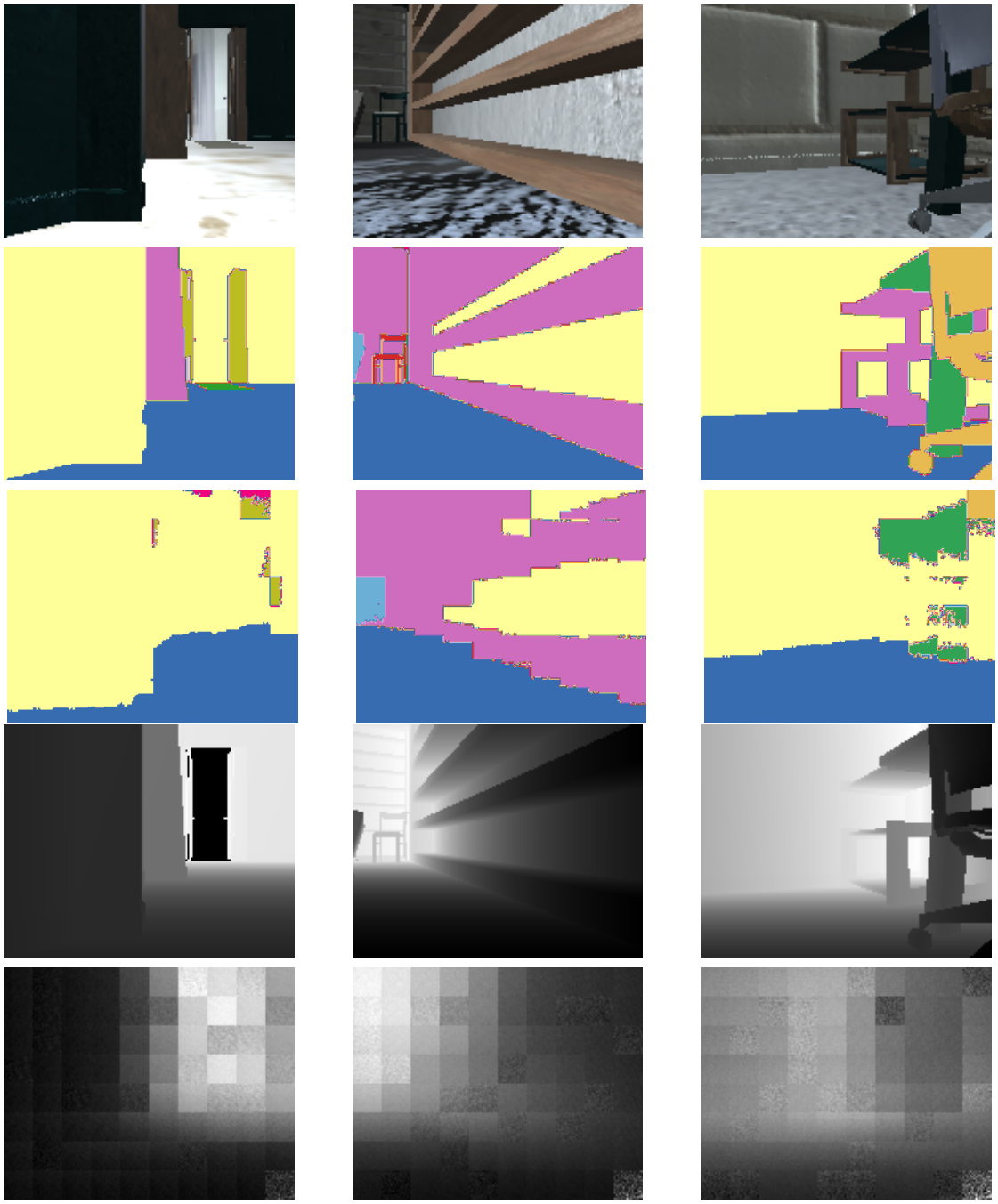


Fig. 4.3 **(Depth+Segmentation) Decoders Training Result Visualization.** The images above show observations at three different time steps. From top to bottom, the five images represent RGB input, observed segmentation, predicted segmentation, observed depth, and predicted depth. We observed the following: 1) While the segmentation decoder performs well predicting large objects, such as floors and walls, it struggles with small, overlapping objects. However, it consistently produces meaningful results. 2) The depth decoder provides results that, while somewhat vague, are still meaningful.

4.2.3 Detection Decoder Experiment

Based on the assumptions mentioned above, we implemented a detection decoder in the pipeline. Since the detection task requires the encoder and decoder to understand spatial relationships strongly and does not predict results in pixel space, we hypothesised that training on the detection task could improve our model's performance in the PointGoal Navigation task. The detection decoder processes the masked visual environment embedding and predicts bounding boxes and labels through two MLP heads.

After adding the detection decoder, the pipeline required 36 hours to complete 80k training steps on an A100 GPU, which reached the Cambridge HPC's running time restriction. Given the limited computational resources and the need for fairness in model comparison, we decided to proceed with the results as they are.

Fig. 4.4 **(Depth+Detection) Decoders Training Loss.** The training losses for the Reward decoder, Depth decoder, and the three training losses for the Detection decoder are converging, indicating that the ViT encoder and decoders have successfully learned.

Figure 4.4 shows a result similar to Figure 4.2, where most of the losses are decreasing but have not yet fully converged. Figure 4.5 presents examples of depth and detection predictions (only bounding boxes with confidence scores above 80% are shown).

We observed that the detection decoder performs well in predicting large, frequently seen objects such as doors, walls, floors, and bookshelves, while often ignoring smaller objects

like chairs. There are instances of overlapping and redundant bounding boxes, a known limitation of the Nearest Neighbor Matching algorithm, which cannot prevent multiple predicted bounding boxes from matching with a single target box.

Despite this, the detection decoder is generating meaningful results, indicating that the ViT encoder has learned to encode the rough locations of objects within the visual environment embedding. However, the depth predictions appear much noisier compared to when the depth decoder is trained alone or in combination with the segmentation decoder.

Models		**Ihlen_0_int**		**Ihlen_1_int**		**Rs_int**		**Env Avg**	
	Steps	**SR**	**SPL**	**SR**	**SPL**	**SR**	**SPL**	**SR**	**SPL**
Experimental Baseline	100k	2.0	**0.02**	**3.2**	**0.02**	**6.4**	**0.04**	**3.9**	**0.03**
LanGWM (depth+detect)	100k	**3.2**	0.02	0.8	0.00	1.6	0.01	1.8	0.01

Table 4.3 **(Depth+Detection) Decoder Experiment.** The results indicate that the addition of the detection decoder, despite increasing the complexity of the model and the training time, did not enhance the model's performance in the PointGoal Navigation task.

However, as shown in Table 4.3, training with the detection decoder did not improve the model's performance in the PointGoal Navigation task. Adding such a complex decoder, coupled with the significantly increased training time, without contributing to the navigation task, highlights a critical issue. Considering all the experimental results and the observed deterioration in the depth decoder's performance when trained alongside the detection decoder, we propose four empirical assumptions to explain this phenomenon:

1. The ViT encoder may be too small to interpret physical environment information effectively across multiple modalities.
2. The ViT encoder may be underfitting and require a longer training schedule to converge fully.
3. Abstract and general object representation might not benefit the PointGoal Navigation task.
4. There may be problematic components within the LanGWM pipeline affecting performance.

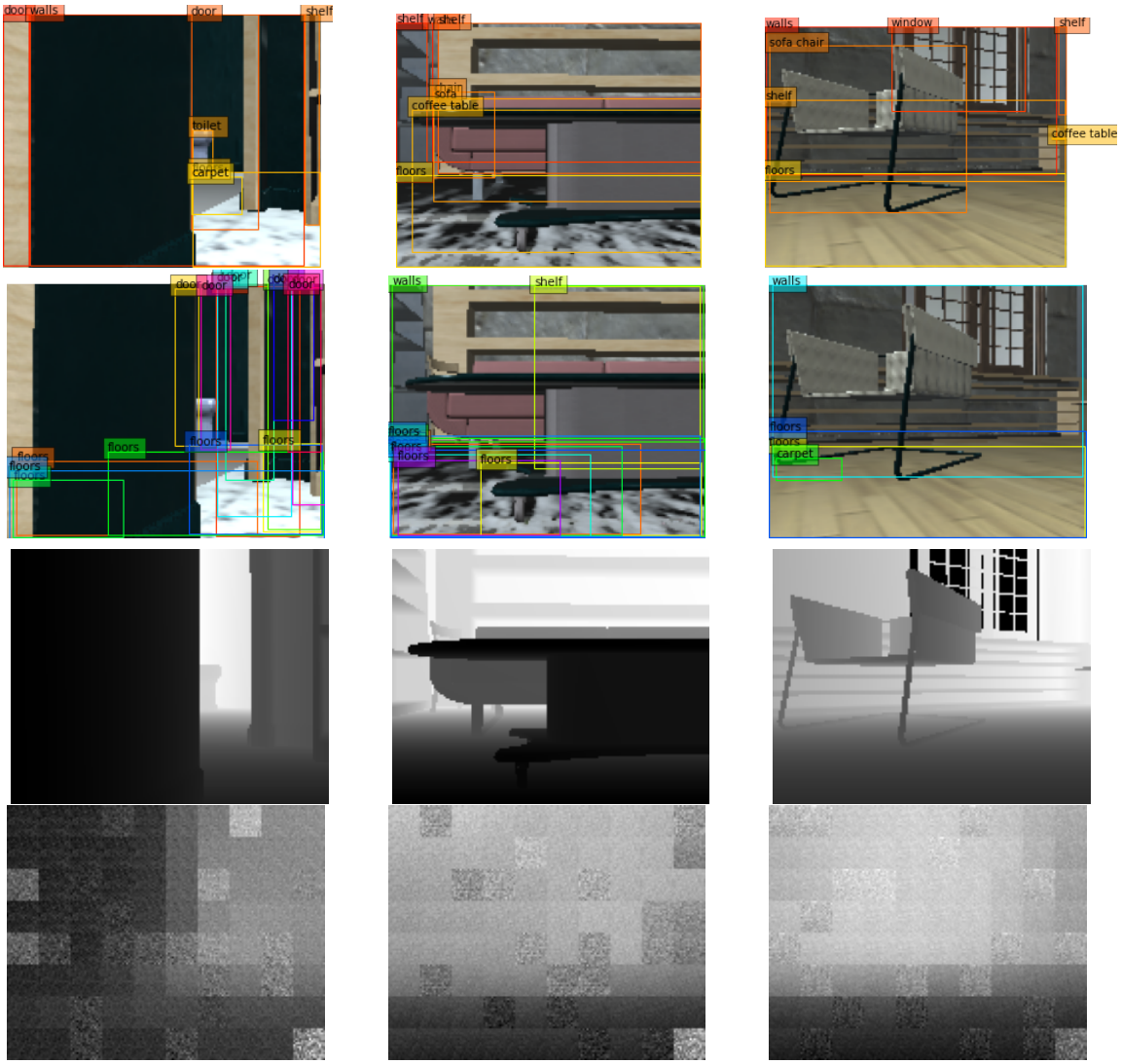


Fig. 4.5 **(Depth+Detection) Decoders Training Result Visualization.** The images above show observations at three different time steps. From top to bottom, the four images represent bounding box ground truth, predicted bounding box, observed depth, and predicted depth. We observed the following: 1) The detection decoder performs well in predicting frequently seen objects such as floors, walls, and doors, consistently producing meaningful results. 2) The depth predictions are noisier than the results shown in Figure 4.3.

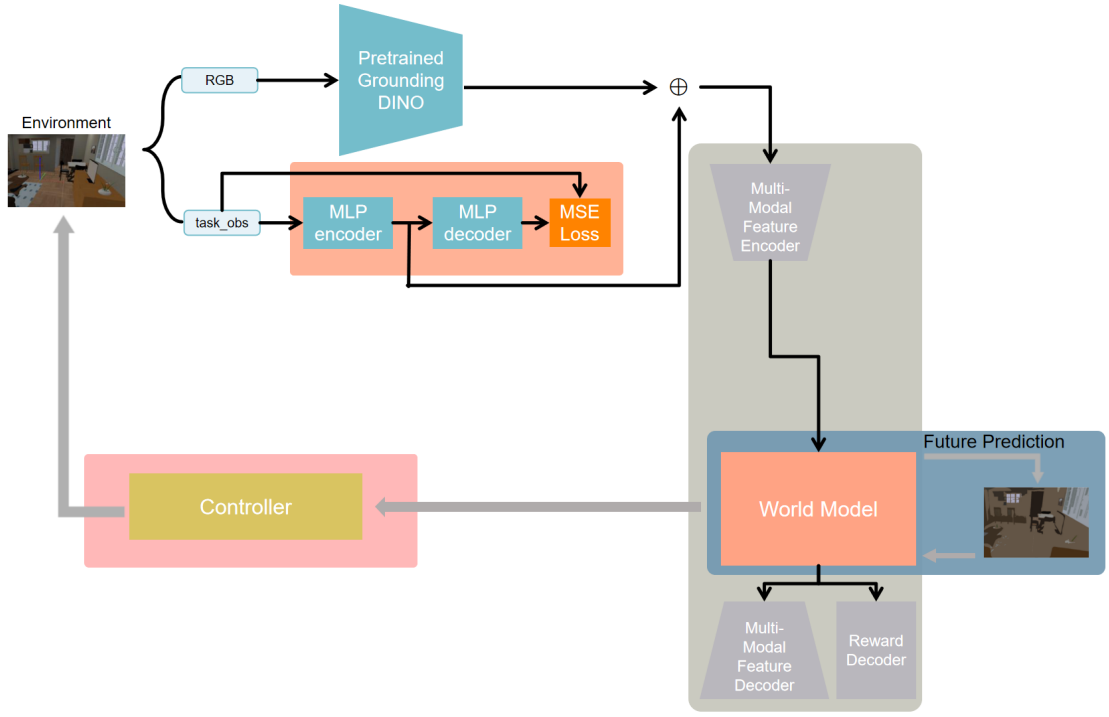


Fig. 4.6 **(LanGWM + Grounding DINO) Pipeline Architecture.** The architecture to test whether there are problematic components in LanGWM's original pipeline. In this architecture, we solely replace the ViT encoder with pre-trained Grounding DINO. If the LanGWm pipeline has no problems, it should show a high SR according to Table 3.1.

4.2.4 Pipeline Analysis with Pretrained Grounding DINO

At this point, we have not only conducted the experiments mentioned above but also performed several ablation studies (see Section 4.3). Unfortunately, none of these experiments yielded satisfactory results. Throughout this process, we focused on improving the ViT encoder's training to enhance the model's performance while treating the other components of the pipeline as a black box. However, given the results of all the experiments conducted, we began to question the reliability of the overall LanGWM pipeline.

To validate this concern, we conducted an experiment using *LanGWM + Pretrained Grounding DINO*. In this experiment, we kept all other components the same but replaced the original ViT encoder with a pre-trained Grounding DINO model. We then trained the remaining components and tested the model on the PointGoal Navigation task. An overview of the modified architecture is shown in Figure 4.6.

As shown in the baseline experiment (see Table 3.1), *DreamerV2 + Grounding DINO* achieve an average accuracy of 37.1%. However, our *LanGWM + Pretrained Grounding DINO*

Models		Ihlen_0_int		Ihlen_1_int		Rs_int		Env Avg	
	Steps	SR	SPL	SR	SPL	SR	SPL	SR	SPL
LanGWM + Grounding DINO	100k	2.4	0.02	0.4	0.00	1.6	0.02	1.47	0.01
LanGWM + Grounding DINO - task_obs encoder	100k	0.0	0.00	0.0	0.00	0.0	0.00	0.0	0.00
LanGWM + Grounding DINO - multi-modal feature encoder	100k	37.6	0.25	13.2	0.08	44.0	0.25	31.6	0.19
LanGWM + Grounding DINO - both encoders	100k	**54.0**	**0.47**	**14.8**	**0.10**	**53.2**	**0.40**	**40.7**	**0.32**

Table 4.4 **SR and SPL for different pipeline configurations in the LanGWM model using a pre-trained Grounding DINO.**

experiment has only achieved an average accuracy of 1.5% (see Table 4.4). Based on that, we conclude that some components in the LanGWM are problematic.

After carefully reviewing the LanGWM pipeline, we concluded two components whose effectiveness seems to be unclear:

1. The task observation MLP auto-encoder module.
2. The multi-modal feature auto-encoder module.

After consulting with the baseline author, we learned that the task observation MLP auto-encoder is intended to isolate the training of the environment encoder from other parts of the model. The purpose of the multi-modal feature auto-encoder is to compress visual and task observation information to a size that is manageable for the RNN in the world model. However, the baseline paper did not include ablation studies to validate the impact of these components on task performance. Therefore, we developed three new pipelines to examine their effects.

Task Observation Auto-Encoder Effectiveness Examination

In the first pipeline, we removed the task observation auto-encoder. In this setup, the task observation MLP encoder is now updated based on the losses from the world model's decoder heads. The detailed pipeline is shown in Figure 4.7. This experiment allows us to assess whether the task observation auto-encoder negatively impacts the pipeline.

The experiment results are presented in Table 4.4. Although the task success rate (SR) decreased slightly after removing the task observation auto-encoder, the change was limited. Therefore, we believe that the impact of the task observation encoder on the model's performance warrants further investigation.

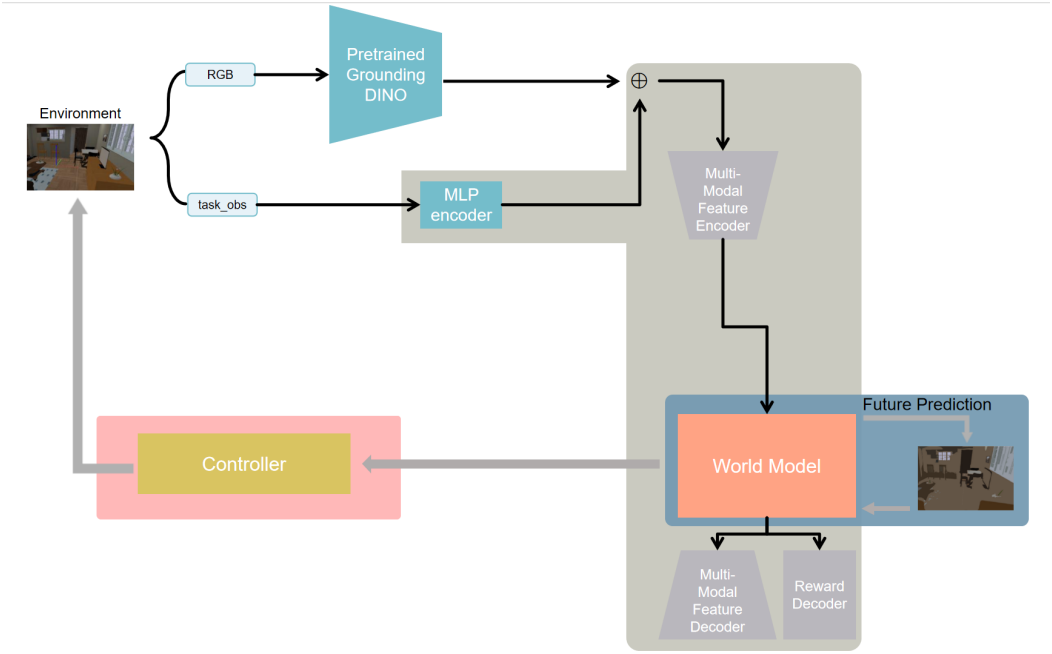


Fig. 4.7 **(LanGWM + Grounding DINO - Task Observation Encoder) Pipeline Architecture.** This architecture is designed to assess whether the Task Observation Encoder is a problematic component of the LanGWM pipeline.

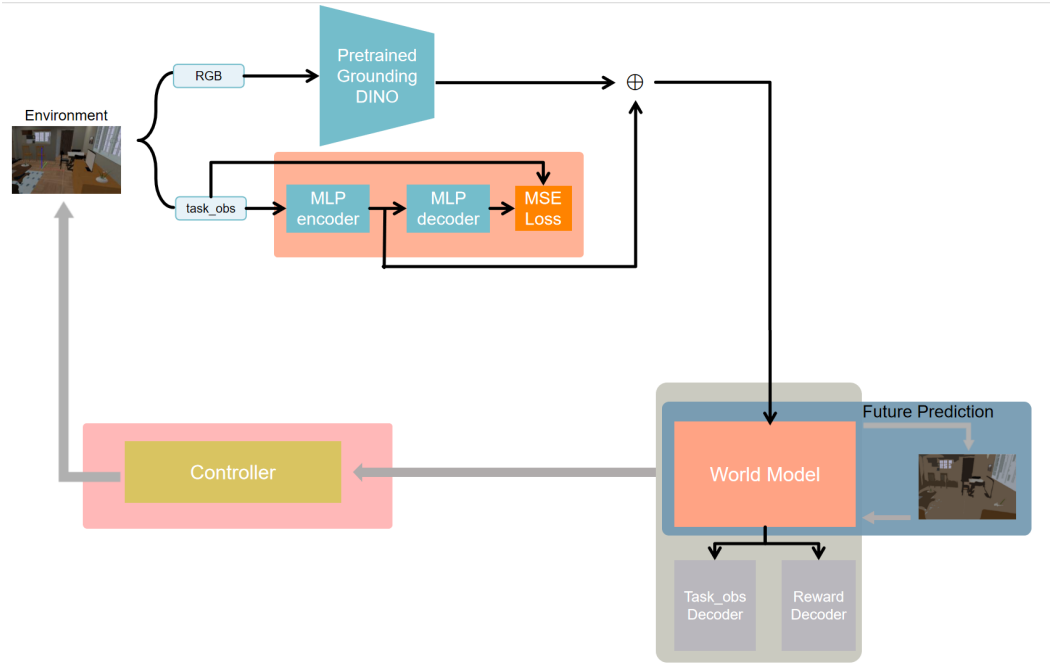


Fig. 4.8 **(LanGWM + Grounding DINO - Multi-modal Feature Encoder) Pipeline Architecture.** This architecture is designed to assess whether the Multi-modal Feature auto-encoder is a problematic component of the LanGWM pipeline.

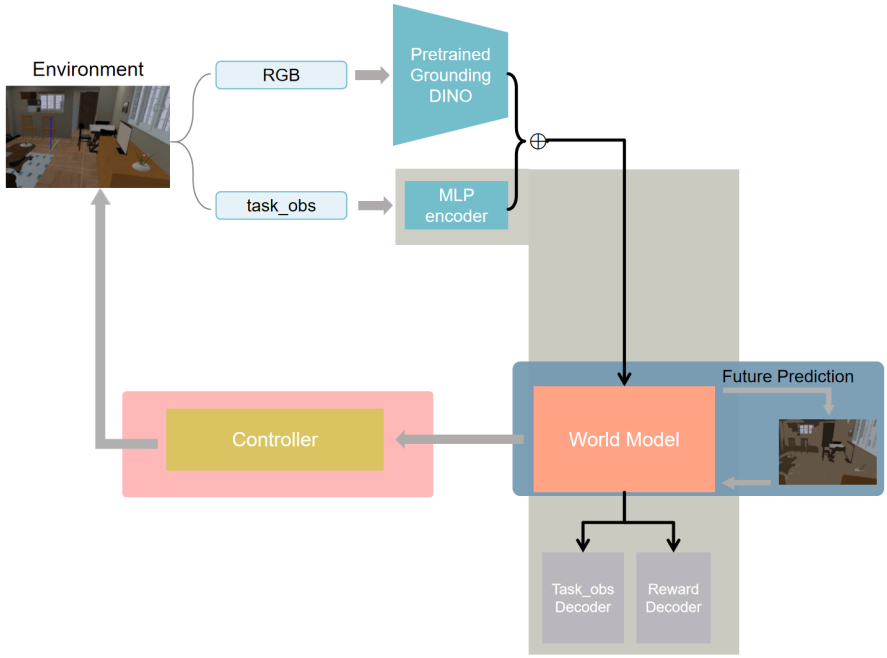


Fig. 4.9 **(LanGWM + Grounding DINO - both Encoder) Pipeline Architecture.** This architecture is designed to assess whether the two auto-encoders are both problematic.

Multi-Modal Feature Auto-Encoder Effectiveness Examination

In the second pipeline, we removed the multi-modal feature auto-encoder while retaining the task observation auto-encoder. Inspired by the architecture of DreamerV2, we also added a task observation decoder head to control the number of decoder heads. The detailed pipeline is shown in Figure 4.8. The experiment results are presented in Table 4.4. This pipeline achieved a success rate (SR) of 31.6%, a significant improvement that suggests the multi-modal feature auto-encoder has a negative impact on performance.

Upon reviewing the multi-modal feature decoder's training loss in the previous experiments, we observed that the loss consistently decreased and often converged. Despite this, the multi-modal feature auto-encoder's negative effect on performance is evident. We hypothesise that this negative impact may be due to the world model's inability to effectively extract useful information from the embeddings produced by the transformer, given that the world model is just an RNN. However, this assumption requires further validation.

Removing Both Auto-Encoder

Finally, we built a pipeline without any auto-encoders. In this setup, the world model decoder heads' losses directly update the task observation MLP encoder. As shown in Table 4.4, this pipeline achieved the highest success rate (SR) of 40.7

When compared to the experiment involving the multi-modal auto-encoder, it is evident that the removal of the task observation auto-encoder further improved performance.

Summary

The series of experiments conducted revealed significant flaws within the LanGWM pipeline, particularly in its multi-modal feature auto-encoder. Initial experiments replacing the ViT encoder with a pre-trained Grounding DINO led to a drastic reduction in performance, indicating potential issues with the pipeline itself rather than just the encoder.

Upon further investigation, it was determined that the task observation MLP auto-encoder was not detrimental to the model's performance, as its removal will not significantly affect the task performance. Conversely, removing the multi-modal feature auto-encoder resulted in a substantial performance improvement (31.6% SR), highlighting its negative impact on the pipeline. This suggests that the multi-modal feature auto-encoder may be hindering the RNN-based world model's ability to effectively extract and utilize information from the transformer-generated embeddings.

A final experiment, removing both auto-encoders, yielded the highest task success rate (40.7%), closely aligning with the baseline performance of DreamerV2 + Grounding DINO. This indicates that the exclusion of these auto-encoders allows the pipeline to function more effectively, particularly by enabling the world model's decoder heads to directly update the task observation MLP encoder.

The findings suggest that the inclusion of the multi-modal feature auto-encoder in the LanGWM pipeline introduces a significant bottleneck. Further research is required to explore potential alternatives for effective multi-modal feature integration within the pipeline. But now, we can conclude that these components were likely the primary reasons for the earlier experiments failing to produce valuable results.

4.2.5 Language Grounding Effectiveness Experiment

Building on the insights from the previous pipeline experiment, where we identified significant performance improvements by removing both auto-encoders, we refined the original

Models		Ihlen_0_int		Ihlen_1_int		Rs_int		Env Avg	
	Steps	SR	SPL	SR	SPL	SR	SPL	SR	SPL
LanGWM + Grounding DINO - both encoders	100k	**54.0**	**0.47**	**14.8**	**0.10**	**53.2**	**0.40**	**40.7**	**0.32**
LanGWM + DINO - both encoders	100k	41.2	0.38	14.0	0.12	38.4	0.34	31.2	0.28

Table 4.5 **Language Grounding Effectiveness Experiment Results.** This table presents the Success Rate (SR) and Success weighted by Path Length (SPL) for the LanGWM pipeline when using two different pre-trained vision transformers: Grounding DINO, which incorporates language information, and DINO, which is purely vision-based.

pipeline. In this experiment, we aimed to investigate the impact of incorporating language information into visual environment embeddings on the performance of an out-of-distribution (OOD) PointGoal navigation task. Specifically, we compared two pre-trained vision transformers (ViTs): Grounding DINO, which is trained with both language and vision, and DINO, which is solely trained on vision data. The goal was to determine whether language-enhanced embeddings, provided by Grounding DINO, offer any advantage over purely vision-based embeddings from DINO in navigating OOD environments.

Both pre-trained models were integrated into the LanGWM pipeline, with their respective encoders replacing the original ViT encoder. To ensure a fair comparison, we maintained consistent architecture and training protocols across both setups, only varying the pre-trained models used. The results are presented in Table 4.5. The *LanGWM + Grounding DINO* configuration achieved an average SR of 40.7% and an SPL of 0.32 across the three environments, outperforming the *LanGWM + DINO* setup, which achieved an average SR of 31.2% and an SPL of 0.28.

The results suggest that incorporating language information into the visual embeddings via Grounding DINO can enhance the model's ability to generalise to OOD environments, as evidenced by the higher success rates and SPL metrics. This outcome aligns with the broader hypothesis that language-grounded embeddings could lead to a more comprehensive understanding of the environment. However, it is important to note that the difference in performance may also be partially attributed to the specific architectures and fine-tuning processes of the pre-trained models.

The sizes of the two models were comparable, with the Grounding DINO using a Swin-T backbone with 29 million parameters and the DINO model utilising a ViT-S/16 backbone with 21 million parameters. This comparability in size suggests that the observed performance differences are likely due to the nature of the pre-trained embeddings rather than discrepancies in model capacity.

While this experiment does not directly prove or disprove our original assumption regarding pixel-wise depth reconstruction as a hindrance, it provides valuable insights into the potential benefits of language grounding in visual embeddings. The fact that the language-vision pre-trained model outperforms the vision-only model in OOD tasks suggests that abstract, language-driven information can contribute to better environmental understanding and task performance. Further research is needed to explore this relationship and refine the methods for incorporating language information into visual models.

4.3 Ablation Studies

In this section, we will present the various attempts we have undertaken to enhance the performance of our pipeline and identify the factors that may be hindering its effectiveness. These efforts encompass multiple aspects, including addressing invalid data issues returned by iGibson 1.0, experimenting with different encoder sizes, and evaluating the effectiveness of MAE masking. Although none of these experiments led to significant breakthroughs, we made several important observations that warrant discussion. We will now outline the motivations, experimental setups, and conclusions drawn from these ablation studies, shedding light on the intricate details of our investigative process.

4.3.1 Depth Loss Masking

The first aspect we aimed to improve is the depth decoder, as depth information is crucial for navigation tasks. Even though we have developed other decoders, we decided not to remove the depth decoder due to its importance. However, if the depth decoder is already struggling to reconstruct masked objects, introducing additional semantic prediction tasks could exacerbate the challenges faced by the ViT encoder. To address this, we investigated and developed two types of depth reconstruction loss masking: Object Mask Loss Masking and Invalid Depth Masking, which will be introduced in the following sections.

Object Mask Loss Masking Methodology

Object masking involves randomly obscuring objects in a room and generating a language prompt that describes the masked objects. This approach encourages the model to rely on language information to reconstruct the masked objects, thereby facilitating language grounding. However, since the language prompts are abstract and lack depth information, forcing the model to reconstruct depth for these masked areas can result in irreducible high loss. To mitigate this, we applied language-guided loss masking by saving the object masks

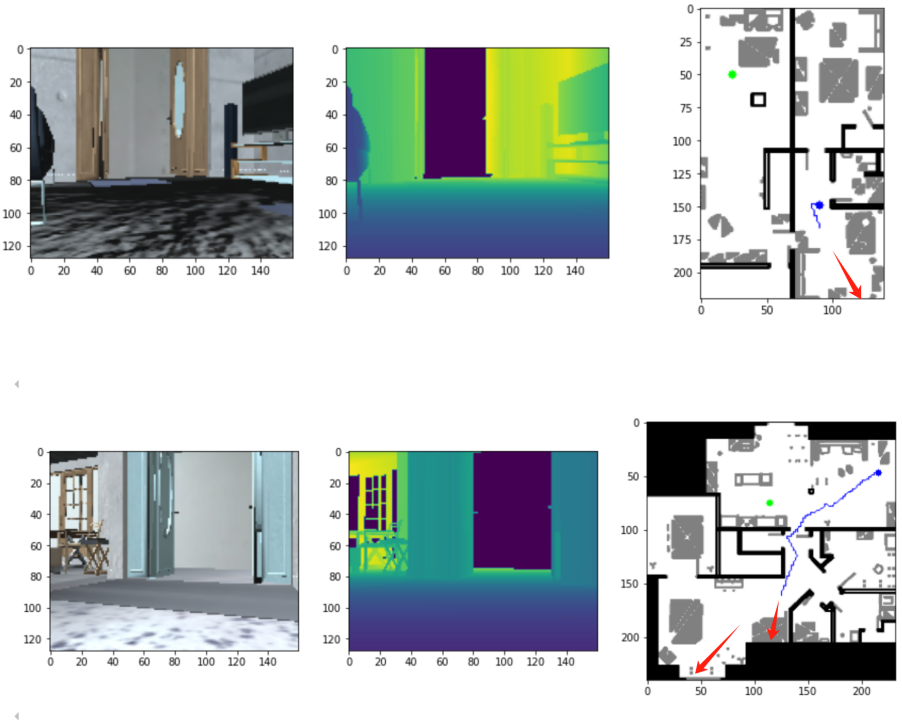


Fig. 4.10 **Invalid Depth Examples in iGibson 1.0.** The upper and lower rows depict two examples of invalid depth returned by iGibson 1.0. In each row, the left image represents the RGB observation, the middle image shows the depth observation, and the right image displays the travel map. Both examples illustrate that unmodeled areas, such as the regions outside of a front door or window, return a depth value of zero. In the depth observation images, it appears as though the highlighted depth suddenly disappears. The locations where invalid depth is observed are marked by red arrows on the travel map.

from the early stages of the pipeline and reapplying them during loss calculation. We expect this will allow the depth decoder to ignore the irreducible loss associated with the masked areas and focus on reconstructing depth from the available RGB information, enhancing overall model performance.

Invalid Depth Loss Masking Methodology

During the investigation of the robot's actions, we observed certain behaviours that were difficult to interpret. For instance, when the robot was navigating toward a destination behind a door, it would repeatedly change direction upon detecting the door in its RGB observation. To understand this behaviour, we manually controlled the robot and analysed its observations in the simulated environment. This investigation revealed that the depth observations contained significant amounts of invalid depth data. Specifically, areas like the

glass sections of windows and regions outside main doors consistently returned a depth value of zero.

We hypothesise that these zero-depth readings occur because iGibson 1.0 returns a depth of zero for unmodeled regions. We also suspect that the robot's tendency to avoid doors stems from the model learning to associate the RGB features of a door with these low (zero) depth values, leading the model to interpret doors as obstacles it needs to avoid. Although sombody could argued that the model might learn to interpret zero depth as invalid, this is unlikely in our context due to the short training schedule and the separation between the vision encoder's training and the subsequent task-related training.

To address this issue, we implemented invalid depth loss masking during the depth reconstruction loss calculation. By masking out all minimum depth values as invalid, we instructed the loss function to ignore these areas, allowing the model to focus on reconstructing valid depth information and thereby improving its overall performance.

Experiments

Models		**Ihlen_0_int**		**Ihlen_1_int**		**Rs_int**		**Env Avg**	
	Steps	**SR**	**SPL**	**SR**	**SPL**	**SR**	**SPL**	**SR**	**SPL**
Experimental Baseline	100k	2.0	0.02	**3.2**	**0.02**	6.4	**0.04**	**3.9**	**0.03**
LanGWM (depth) + Invalid Depth Mask	100k	**5.2**	**0.03**	0.4	0.00	5.6	0.03	3.7	0.02
LanGWM (depth) + Object Mask	100k	4.4	0.03	2.4	0.01	4.0	0.01	3.6	0.02
LanGWM (depth) + both Masks	100k	4.0	0.03	0.8	0.00	**6.8**	0.03	3.9	0.02

Table 4.6 **Effect of Depth Loss Masking Techniques on PointGoal Navigation Performance.** This table presents the SR and SPL for various configurations of the LanGWM model with different depth loss masking strategies: Invalid Depth Masking, Object Mask Loss Masking, and a combination of both.

The experiment results (see Table 4.6)indicate that neither the Invalid Depth Mask nor the Language-Guided Mask triggered significant improvements in performance across the evaluated environments. The Success Rate (SR) and Success per Length (SPL) metrics remained largely unchanged, with slight variations that do not suggest a meaningful impact from the applied masking strategies.

From the current point of view, it is plausible that the lack of significant effect can be attributed to the task observation encoder and multi-modal feature encoder. The presence of these encoders may overshadow the impact of more subtle adjustments, such as depth loss masking. Further investigation is required to explore this hypothesis and identify more effective modifications.

4.3.2 Encoder Size Experiment

Models		Ihlen_0_int		Ihlen_1_int		Rs_int		Env Avg	
	Steps	SR	SPL	SR	SPL	SR	SPL	SR	SPL
LanGWM (depth+seg) (Encoder_S)	100k	1.6	0.01	0.8	0.00	1.6	0.01	1.3	0.01
LanGWM (depth+seg) (Encoder_M)	100k	0.4	0.00	0.0	0.00	0.0	0.00	1.3	0.01
LanGWM (depth+seg) (Encoder_L)	100k	0.0	0.00	0.0	0.00	0.0	0.00	0.0	0.00
LanGWM (depth+detect_S) (Encoder_S)	100k	**3.2**	**0.02**	0.8	0.00	1.6	0.01	1.8	0.01
LanGWM (depth+detect_S) (Encoder_M)	100k	2.8	0.02	**1.2**	**0.01**	**4.4**	**0.03**	**2.8**	**0.02**
LanGWM (depth+detect_S) (Encoder_L)	100k	1.2	0.01	0.4	0.00	0.8	0.01	0.8	0.01

Table 4.7 **Encoder Size Experiment Results.** This table compares the SR and SPL across different encoder configurations (Encoder_S, Encoder_M, and Encoder_L) within the LanGWM pipeline. The models were tested with two decoder setups: depth paired with segmentation and depth paired with detection.

Previously, we expressed concerns that our ViT encoder might be too small to effectively integrate multi-modal information, which typically requires larger encoders to handle the increased complexity and variety of data sources. To address this concern, we conducted an experiment to assess how increasing the encoder's size impacts the model's performance.

In this experiment, we tested three encoder configurations: **Encoder_S** with 3 layers and 4 heads, **Encoder_M** with 6 layers and 8 heads, and **Encoder_L** with 12 layers and 10 heads. The experiments were carried out using pipelines that included both a depth decoder paired with a segmentation decoder and a depth decoder paired with a detection decoder. The result can be seen in Table 4.7.

The results of the encoder size experiment indicate a clear trend: as the encoder size increases, the performance consistently declines. Specifically, the models with larger encoders (Encoder_M and Encoder_L) perform worse compared to those with smaller encoders (Encoder_S).

Regardless of the influence of the task observation encoder and the multi-modal feature encoder, there are two primary explanations for this phenomenon. First, the larger encoders may be underfitting due to the fixed training schedule and the limited amount of training data available. The increased complexity of the larger models could require more extensive training to fully capture the nuances of the data. Second, it is possible that the larger encoders extract features that are too high-level for the downstream world model and actor-critic components to effectively interpret, leading to a mismatch between the encoder output and the requirements of the subsequent processing stages. Further investigation is needed to validate these assumptions and to determine the optimal encoder size for this task.

4.3.3 MAE Effectiveness Experiment

Models		Ihlen_0_int		Ihlen_1_int		Rs_int		Env Avg	
	Steps	SR	SPL	SR	SPL	SR	SPL	SR	SPL
LanGWM (depth+seg)	100k	**1.6**	**0.01**	**0.8**	**0.00**	**1.6**	**0.01**	**1.3**	**0.01**
LanGWM (depth+seg-MAE Masking)	100k	0.0	0.00	0.0	0.00	0.0	0.00	0.0	0.00

Table 4.8 **MAE Effectiveness Experiment.** This table shows the Success Rate (SR) and Success weighted by Path Length (SPL) for the LanGWM model with and without MAE masking.

Previously, concerns were raised about the potential conflict between applying both object masking and MAE masking in our pipeline. Specifically, while we aim to explicitly train the model to reconstruct masked objects using language prompts, the simultaneous application of MAE masking introduces uncertainty: will the model rely on the language prompt to reconstruct the masked object, or will it utilise the tokens discarded by the MAE masking algorithm?

To address this concern, we conducted an experiment to explore the effectiveness of MAE masking. We removed the MAE masking process in a pipeline with a depth decoder and segmentation decoder. In this setup, after the CNN transformed images into image tokens, all tokens were directly passed to the ViT encoder for processing, allowing us to observe how the encoder's performance would change when solely focused on reconstructing the masked object without the interference of MAE masking.

As shown in Table 4.8, the results of this experiment revealed that removing MAE masking completely disabled LanGWM's ability to navigate robots, as indicated by a 0.0% SR and SPL across all environments. However, analysis of the loss curves showed that both the segmentation prediction loss and depth reconstruction loss converged to the same levels as they did in the standard (depth+seg) pipeline with MAE masking. This suggests that, while the quality of segmentation and depth reconstruction tasks remains consistent regardless of MAE masking, training with MAE enables the encoder to learn more meaningful semantic environment representations. These findings underscore the importance of MAE masking in effectively training the LanGWM pipeline.

4.4 Conclusion

The experiments conducted in this study can be broadly categorised into three main areas: baseline reproduction, the proposed methods' experiments, and pipeline analysis using pretrained models. The baseline reproduction focused on replicating the original LanGWM

results to ensure consistency in the experimental setup. This process revealed significant randomness in the model's performance.

In the proposed methods' experiments, we introduced segmentation and detection decoders to investigate their impact on enhancing the model's semantic understanding. The segmentation decoder was expected to help the model capture the spatial relationships between objects, while the detection decoder aimed to improve the model's ability to identify and localise objects within the environment. However, even though the decoders generates some meaningful segmentation image and detection bounding boxes, these experiments showed such training methods do not improve PointGoal Navigation task performance at all.

The pipeline analysis was conducted using pretrained models, specifically Grounding DINO and DINO, to evaluate the impact of various components within the original pipeline on task performance. This analysis was aimed at understanding how effectively the pipeline incorporated language information into the visual environment embeddings. The results highlighted significant flaws in the existing LanGWM pipeline, particularly with the task observation encoder and the multi-modal feature encoder, both of which were found to have a detrimental effect on task performance. This finding suggests that these components were likely the primary reasons for the earlier experiments failing to produce valuable results. Additionally, we compared the contributions of pretrained Grounding DINO and pretrained DINO to task performance, and concluded that incorporating language information can indeed enhance performance in OOD navigation tasks.

Additionally, several ablation studies were performed alongside the main experiments to investigate the effects of training strategies. These studies, which included experiments like depth loss masking, encoder size variation, and the effectiveness of MAE (Masked Autoencoder) masking, provided further insights into the factors influencing model performance and helped identify areas for improvement within the pipeline .

Chapter 5

Conclusion and Future Work

In this dissertation, we systematically examined the LanGWM, focusing on extending its architecture and identifying performance limitations. Our investigation centred on the OOD PointGoal navigation task, where we hypothesized that the original depth reconstruction task used for training the language-grounded representation might be a significant bottleneck. To address this, we introduced new segmentation and detection decoders into the LanGWM pipeline to enhance the model's ability to capture high-level semantic information. These additions aimed to shift the model's focus away from detailed visual reconstruction towards a broader, language-grounded understanding.

Our experiments evaluated the LanGWM pipeline through baseline reproduction, the integration of new decoders, and the analysis of pre-trained models. We identified significant limitations within the current architecture, particularly in the task observation and multi-modal feature encoders, which hindered the model's performance. Additionally, the integration of segmentation and detection tasks did not improve PointGoal navigation performance as anticipated, underscoring the challenges of effectively incorporating high-level semantic information. Despite these obstacles, our findings suggest that language grounding holds promise for enhancing model performance in novel environments, indicating potential avenues for further research and optimization.

In conclusion, while the new decoders and training tasks did not produce the expected performance gains, our research identified critical architectural limitations, highlighted the potential of language grounding, and laid the groundwork for future efforts to refine and optimize the LanGWM pipeline.

5.1 Future Work

Unfortunately, I realized the issues in the original pipeline only in the last two weeks of the project after spending too much time blindly trying to improve the environment encoder. Additionally, I lacked the necessary computing resources and time to conduct further experiments.

From a personal perspective, I believe our project proposal set an ambitious goal. However, our baseline work lacked a series of critical hypothesis validations. For example, our baseline pipeline is a complex architecture involving five nearly independent neural networks. The loss from downstream tasks is not used to update upstream networks, yet all networks require online training. Additionally, the environment encoder we aim to train involves various types of information (e.g., language, images, depth, task observation), but we wanted to prove the high data efficiency of explicit training methods, so we designed a smaller-than-usual transformer block. We also did not know whether the masked auto-encoding approach was suitable for cross-modality tasks. For instance, is it reasonable to ask the decoder to predict object bounding boxes from an image with 75% of its pixels masked out? The baseline pipeline's accuracy on the navigation task was less than 10%, with a high variance in reproduction. Before proposing to add new decoders to the original pipeline, should we not first ensure the reliability of the existing pipeline?

Given these considerations, I believe the most important future work for LanGWM should be divided into the following four steps:

1. Test various pipeline structures using pre-trained Grounding DINO. Select the best-performing and most stable pipeline as the foundation for subsequent experiments.
2. Independently train the environment encoder outside the pipeline using various training methods. Evaluate the encoder's generality using decoder tasks (e.g., depth prediction, segmentation, detection), zero-shot, or few-shot learning.
3. Integrate the pre-trained encoders from the second step into the pipeline and compare their performance on the PointGoal Navigation task against pre-trained Grounding DINO. If the performance does not surpass Grounding DINO, does this suggest that Grounding DINO's training method is better suited for producing a general environment representation?
4. Take the best-performing encoders from the third step and train them from scratch in the pipeline with online training. Compare their performance with other SOTA RL models.

Following these experimental steps will build each phase of our work on a reliable foundation, likely leading to more significant advancements.

References

Amiri, R., Mehrpouyan, H., Fridman, L., Mallik, R. K., Nallanathan, A., and Matolak, D. (2018). A machine learning approach for power allocation in hetnets considering qos. In *2018 IEEE international conference on communications (ICC)*, pages 1–7. IEEE.

Anderson, P., Chang, A., Chaplot, D. S., Dosovitskiy, A., Gupta, S., Koltun, V., Kosecka, J., Malik, J., Mottaghi, R., Savva, M., et al. (2018). On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*.

Cai, F. and Koutsoukos, X. (2020). Real-time out-of-distribution detection in learning-enabled cyber-physical systems. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*, pages 174–183. IEEE.

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. (2020). End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer.

Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., and Joulin, A. (2021). Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660.

Chen, Q., Zhu, X., Ling, Z., Wei, S., Jiang, H., and Inkpen, D. (2016). Enhanced lstm for natural language inference. *arXiv preprint arXiv:1609.06038*.

Coumans, E. et al. (2013). Bullet physics library. *Open source: bulletphysics. org*, 15(49):5.

Deisenroth, M. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

DeVries, T. and Taylor, G. W. (2018). Learning confidence for out-of-distribution detection in neural networks. *arXiv preprint arXiv:1802.04865*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Dosovitskiy, A. and Koltun, V. (2016). Learning to act by predicting the future. *arXiv preprint arXiv:1611.01779*.

Farid, A., Veer, S., and Majumdar, A. (2022). Task-driven out-of-distribution detection with statistical guarantees for robot learning. In *Conference on Robot Learning*, pages 970–980. PMLR.

Gupta, S., Davidson, J., Levine, S., Sukthankar, R., and Malik, J. (2017). Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2616–2625.

Ha, D. and Schmidhuber, J. (2018). Recurrent world models facilitate policy evolution. *Advances in neural information processing systems*, 31.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. (2019a). Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2019b). Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR.

Hafner, D., Lillicrap, T., Norouzi, M., and Ba, J. (2020). Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*.

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. (2022). Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*.

Kolve, E., Mottaghi, R., Han, W., VanderBilt, E., Weihs, L., Herrasti, A., Deitke, M., Ehsani, K., Gordon, D., Zhu, Y., et al. (2017). Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*.

Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014.

Laskin, M., Srinivas, A., and Abbeel, P. (2020). Curl: Contrastive unsupervised representations for reinforcement learning. In *International conference on machine learning*, pages 5639–5650. PMLR.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lee, J., X. Grey, M., Ha, S., Kunz, T., Jain, S., Ye, Y., S. Srinivasa, S., Stilman, M., and Karen Liu, C. (2018). Dart: Dynamic animation and robotics toolkit. *The Journal of Open Source Software*, 3(22):500.

Liang, J., Makoviychuk, V., Handa, A., Chentanez, N., Macklin, M., and Fox, D. (2018). Gpu-accelerated robotic simulation for distributed reinforcement learning. In *Conference on Robot Learning*, pages 270–282. PMLR.

Liu, S., Zeng, Z., Ren, T., Li, F., Zhang, H., Yang, J., Li, C., Yang, J., Su, H., Zhu, J., et al. (2023). Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Munro, P. (1987). A dual back-propagation scheme for scalar reward learning. In *Ninth Annual Conference of the Cognitive Science Society*, pages 165–176. Hillsdale, NJ. Cognitive Science Society Lawrence Erlbaum.

Nguyen, D. and Widrow, B. (1990). The truck backer-upper: An example of self-learning in neural networks. In *Advanced neural computers*, pages 11–19. Elsevier.

Nilsson, N. J. et al. (1984). *Shakey the robot*, volume 323. Sri International Menlo Park, California.

OpenAI (2024). Gpt-4. https://openai.com. Accessed: 2024-09-09.

Partsey, R., Wijmans, E., Yokoyama, N., Dobosevych, O., Batra, D., and Maksymets, O. (2022). Is mapping necessary for realistic pointgoal navigation? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17232–17241.

Pearl, J. and Mackenzie, D. (2018). *The book of why: the new science of cause and effect*. Basic books.

Poudel, R. P., Pandya, H., Zhang, C., and Cipolla, R. (2023). Langwm: Language grounded world model. *arXiv preprint arXiv:2311.17593*.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018). Improving language understanding by generative pre-training.

Robinson, T. and Fallside, F. (1989). Dynamic reinforcement driven error propagation networks with application to game playing. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 11.

Savva, M., Chang, A. X., Dosovitskiy, A., Funkhouser, T., and Koltun, V. (2017). Minos: Multimodal indoor simulator for navigation in complex environments. *arXiv preprint arXiv:1712.03931*.

Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., et al. (2019). Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9339–9347.

Schmidhuber, J. (1990a). Making the world differentiable: On using supervised learning fully recurrent neural networks for dynamic reinforcement learning and planning in non-stationary environments. *Technische Universität München Tech. Report: FKI-126-90*.

Schmidhuber, J. (1990b). An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In *1990 IJCNN international joint conference on neural networks*, pages 253–258. IEEE.

Schmidhuber, J. (1990c). Reinforcement learning in markovian and non-markovian environments. *Advances in neural information processing systems*, 3.

Schmidhuber, J. (1991). Curious model-building control systems. In *Proc. international joint conference on neural networks*, pages 1458–1463.

Sedlmeier, A., Gabor, T., Phan, T., Belzner, L., and Linnhoff-Popien, C. (2019). Uncertainty-based out-of-distribution classification in deep reinforcement learning. *arXiv preprint arXiv:2001.00496*.

Sedlmeier, A., Müller, R., Illium, S., and Linnhoff-Popien, C. (2020). Policy entropy for out-of-distribution classification. In *Artificial Neural Networks and Machine Learning–ICANN 2020: 29th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 15–18, 2020, Proceedings, Part II 29*, pages 420–431. Springer.

Shen, B., Xia, F., Li, C., Martín-Martín, R., Fan, L., Wang, G., Pérez-D'Arpino, C., Buch, S., Srivastava, S., Tchapmi, L., et al. (2021). igibson 1.0: A simulation environment for interactive tasks in large realistic scenes. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7520–7527. IEEE.

Sinha, R., Sharma, A., Banerjee, S., Lew, T., Luo, R., Richards, S. M., Sun, Y., Schmerling, E., and Pavone, M. (2022). A system-level view on out-of-distribution data in robotics. *arXiv preprint arXiv:2212.14020*.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pages 216–224. Elsevier.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

Tay, Y., Tuan, L. A., and Hui, S. C. (2018). Compare, compress and propagate: Enhancing neural architectures with alignment factorization for natural language inference. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1565–1575.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033.

TurtleBot (2024). Turtlebot 2. https://www.turtlebot.com/turtlebot2/. Accessed: July 30, 2024.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Werbos, P. J. (1987). Learning how the world works: Specifications for predictive networks in robots and brains. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics, NY*.

Werbos, P. J. (1989). Neural networks for control and system identification. In *Proceedings of the 28th IEEE Conference on Decision and Control,*, pages 260–265. IEEE.

Wijmans, E., Kadian, A., Morcos, A., Lee, S., Essa, I., Parikh, D., Savva, M., and Batra, D. (2019). Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. *arXiv preprint arXiv:1911.00357*.

Zhao, X., Agrawal, H., Batra, D., and Schwing, A. G. (2021). The surprising effectiveness of visual odometry techniques for embodied pointgoal navigation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16127–16136.

Appendix A

Pipeline Configuration

Here we only listed out the most important configuration to our experiment. For more configuration, please check the Appendix of Poudel et al. (2023).

A.1 Segmentation Decoder

As introduced in the Section 3.2, the segmentation decoder consists a transformer block and an single layer MLP head. The transformer block uses the following configuration:

1. Number of self-attention layer: 3
2. Number of self-attention head: 2
3. Self-attention embedded dimension: 256
4. MLP layer embedded dimension: [512, 256]
5. Layer normalization Epsilon: 10^{-6}
6. Dropout rate: 0.1
7. Activation function: gelu

The single layer MLP head uses the following configuration:

1. MLP head embedded dimension: $256 * 63 = 16128$

The transformer block is small comparing to standard size. The specific reason for this will be analyzed in Section 4.2.3 and Section 4.3.2.

A.2 Detection Decoder

The detection decoder consists a transformer block, a single layer MLP head for predicting object labels, and a three layer MLP head for predicting bounding box. The transformer block uses the following configuration:

1. Number of self-attention layer: 3
2. Number of self-attention head: 4
3. Self-attention embedded dimension: 256
4. MLP layer embedded dimension: [2048, 256]
5. Layer normalization Epsilon: 10^{-6}
6. Dropout rate: 0.1
7. Activation function: relu
8. Number of queries (related to position embedding calculation): 30

The single MLP head for predicting object labels has the following configuration:

- MLP head embedded dimension: 63 (number of class labels)

The three layer MLP head for predicting bounding box:

- MLP head embedded dimension: [256, 256, 4]

Most of the above configuration adopt DETR's original configuration(Carion et al., 2020), except for the number of self-attention layers and heads. Experiment evaluating the effect of the self-attention layer will be discussed in Section 4.2.3.

A.3 Training Configuration

The general training processes are using the following configuration:

1. Training steps: 100k steps
2. Train run time limit: 500 steps
3. Prefill training data: 5000 steps
4. Networks training interval: every 5 steps

A.4 Testing Configuration

1. Number of Evaluation Trails: 250