

Model-based Planning for Efficient Task Execution

by

Wenqi Ding

B.S. Computer Science and Engineering and Mathematics, MIT, 2025

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2025

© 2025 Wenqi Ding. All rights reserved.

The author hereby grants to MIT a nonexclusive, worldwide, irrevocable, royalty-free license to exercise any and all rights under copyright, including to reproduce, preserve, distribute and publicly display copies of the thesis, or release the thesis under an open-access license.

Authored by: Wenqi Ding
Department of Electrical Engineering and Computer Science
May 15, 2025

Certified by: Hamsa Balakrishnan
William E. Leonhard (1940) Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by: Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Model-based Planning for Efficient Task Execution

by

Wenqi Ding

Submitted to the Department of Electrical Engineering and Computer Science
on May 15, 2025 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE

ABSTRACT

Robotic agents navigating 3D environments must continuously decide their next moves by reasoning about both visual observations and high-level language instructions. However, they plan in a high-dimensional latent space, opaque to human collaborators. Hence, it is difficult for humans to understand the agent’s decision-making process. This lack of interpretability hinders effective collaboration between humans and robots. The key question we are trying to answer in this thesis is: Can we build a unified planning framework that fuses visual and language into a single, interpretable representation, so that humans can interpret robots’ decisions?

We propose a model-based planning framework built around pretrained vision-language models (VLMs). We show that VLMs can be used to plan in a unified embedding space, where visual and language representations can be decoded back to human-interpretable forms. Empirical evaluation on vision-language navigation benchmarks demonstrates both improved sample efficiency and transparent decision making, enabling human-in-the-loop planning and more effective human-robot collaboration.

Thesis supervisor: Hamsa Balakrishnan

Title: William E. Leonhard (1940) Professor of Aeronautics and Astronautics

Acknowledgments

I am profoundly thankful to my supervisor Professor Hamsa for giving me the opportunity to work on this project. I am also grateful to my mentor Siddharth for all the support and encouragement he has given me during my research. From the very first UROP I did on the multi-agent project to this MEng project, working with Siddharth has been super rewarding and exciting. I also want to thank my collaborator Aditya Kapoor for his invaluable help and insights throughout the project. Many thanks as well to everyone in the DINaMo lab for creating a wonderful community.

Finally, I want to thank my family and friends for all their care and love. Without their moral support and unwavering faith in my abilities, I would not have had the strength and determination to see this journey through to the end.

Contents

<i>List of Figures</i>	9
<i>List of Tables</i>	11
1 Introduction	13
2 Background and Related Work	15
2.1 Background	15
2.1.1 Markov Decision Process and Reinforcement Learning	15
2.1.2 Language Models	17
2.1.3 Representation Learning	20
2.2 Related Work	21
2.2.1 Model-based Planning with LLMs	22
2.2.2 Model-free planning with LLMs	23
3 System Design	25
3.1 System Overview	25
3.2 Environment	26
3.3 Dataset	27
3.3.1 Expert Trajectories	28
3.3.2 Generating Scene Description	28
3.4 State Abstractor	33
3.4.1 Formal Mapping	34
3.4.2 Training Objective	35
3.4.3 Generating Belief States	36
3.5 Policy Model	37
3.5.1 What the Module Does	37
3.5.2 Policy Training	39
3.6 Dynamics Model	39
3.7 Outcome Evaluator	40
4 Experiments and Results	41
4.1 Training the VLMs	41
4.1.1 Training Setup	41
4.1.2 Performance Bottlenecks	41
4.1.3 Mitigations	42
4.2 Evaluation for Policy Learning	43

4.2.1	Future Improvements	44
5	Conclusions	47
A	Prompt Evaluation	49
A.1	Scene Description From Metadata	49
B	Supplemental Material for Model Training and Evaluation	51
B.1	VLM fine-tuning	51
B.2	Policy Model Evaluation	53
B.2.1	Comparison of Predicted and Ground Truth Action Sequences	53
	<i>References</i>	55

List of Figures

2.1	Model-based RL vs Model-free RL	16
2.2	Vision-Language Model Architecture	19
3.1	System Overview	26
3.2	An example of ProcTHOR environment	27
3.3	An example of a scene image	32
3.4	State abstractor	34
3.5	Policy Model Architecture	38

List of Tables

3.1	Comparison of scene descriptions generated with different prompts.	29
B.1	Hyperparameters for fine-tuning InstructBLIP with LoRA.	51
B.2	Hyperparameters for fine-tuning Qwen2.5-VL with LoRA.	52

Chapter 1

Introduction

Embodied agents are often tasked with navigating complex environments, making decisions based on visual observations, and following high-level language instructions. Examples include vision-and-language navigation (e.g. robots following spoken directions through an indoor environment), object manipulation with natural-language instructions (e.g. “pick up the blue block and place it on the red table”), and assistive robotics in household or industrial settings. In all of these scenarios, the planner must take in both visual and language cues and plan accordingly, and most importantly, allow human collaborators to inspect and understand its reasoning. However, these agents typically operate in high-dimensional latent spaces that are opaque to human collaborators, making it difficult for humans to understand the agent’s decision-making process. Making the agent’s reasoning interpretable is a key challenge for effective human-robot collaboration.

Recent advances in large-scale vision-language foundation models open new possibilities for unified representations that bridge perception, language, and decision-making. A pretrained VLM’s latent space encodes rich semantic and spatial information, suggesting it could serve simultaneously as a perception backbone, a prediction engine, and a natural interface for human-readable explanations. However, leveraging these embeddings for long-horizon planning raises important questions: How can we learn accurate dynamics in latent space?

How do we propose and evaluate actions efficiently? And how can each planning step remain interpretable to human collaborators?

In this thesis, we propose a modular, VLM-centric planning framework that unifies representation, prediction, action selection, and evaluation within a single pretrained embedding space. We develop methods for supervised latent dynamics learning, lightweight policy and value estimation, and natural-language decoding of imagined futures.

The main contributions of this thesis are:

- We design a modular framework that leverages pretrained VLMs and unifies perception, language, and decision-making in the same embedding space, enabling interpretable planning and reasoning.
- We develop a hybrid data-generation pipeline that balances precision and conciseness by distilling short seed captions from GPT-4 and structured scene metadata, mitigating language models' hallucinations while capturing essential spatial relationships in the scene.
- We introduce practical optimizations for training VLMs on large datasets that significantly reduce CPU-GPU stalls and improve training efficiency.

Chapter 2

Background and Related Work

2.1 Background

In this section, we introduce the foundational concepts for understanding the methods and algorithms developed in this thesis. We begin with a formal definition of Markov Decision Processes (MDPs) and Reinforcement Learning (RL) in Section 2.1.1. We then review large language models (LLMs) and vision-language models (VLMs) in Section 2.1.2. Finally, we discuss representation learning techniques in Section 2.1.3 that enable agents to transform high-dimensional perceptual inputs, such as language and vision, into compact, task-relevant representations.

2.1.1 Markov Decision Process and Reinforcement Learning

Reinforcement learning (RL) addresses the problem of an agent making a sequence of decisions in a stochastic environment to maximize cumulative reward. Formally, an RL problem is modeled as a Markov Decision Process (MDP) defined by the tuple (S, A, P, R, γ) , where S is the set of states, A the actions, $P(s' | s, va)$ the transition probabilities, $R(s, a)$ the (possibly stochastic) reward, and $\gamma \in [0, 1]$ a discount factor. At each time step t , the agent observes state s_t , selects action a_t according to its policy π , transitions to $s_{t+1} \sim P(\cdot | s_t, a_t)$,

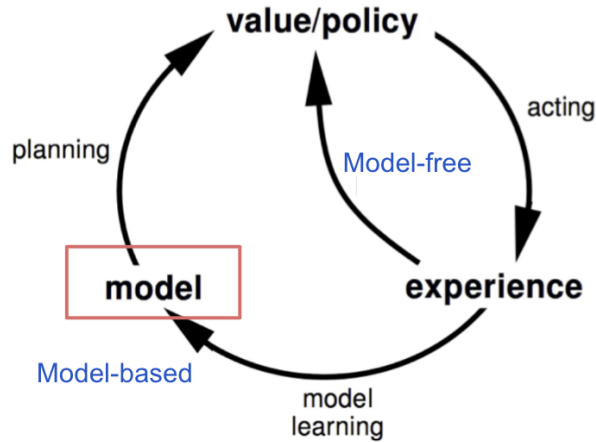


Figure 2.1: Model-based RL vs Model-free RL

and receives reward $r_t = R(s_t, a_t)$. The objective is to find a policy π^* that maximizes the expected return $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$.

There are several classes of methods for solving MDPs, including dynamic programming, model-free methods, and model-based methods. Here we discuss model-based and model-free methods. These two methods differ in how they learn the policy and the value function. Model-based methods learn a model of the environment and use it to plan the best action to take. Model-free methods learn a policy directly from experience without explicitly modeling the environment. The choice between these paradigms has significant implications for sample efficiency, generalization, stability, and applicability to real-world systems. Figure 2.1 shows an illustrative overview of the difference between model-based and model-free reinforcement learning.

Model-based RL

Model-based methods explicitly model the environment’s dynamics and use it to plan and evaluate candidate action sequences through trajectory simulation or tree search. This makes the methods highly sample efficient and interpretable for decision-making. However, this requires accurate modeling which incurs extra computational overhead.

Model-free RL

Model-free methods bypass explicit modeling and directly estimate value functions or policies from interactions. Algorithms such as Q-learning, policy gradients, or actor-critic methods update estimates based solely on observed rewards and transitions. These methods are simpler to implement and more robust to model errors, but generally require more environment samples.

2.1.2 Language Models

Large Language Models (LLMs) such as GPT [1, 2], LLaMA [3], and Mistral [4] have emerged as powerful tools for reasoning, planning, and decision-making in natural language contexts.

LLMs are trained to predict the next token in a sequence given the previous context. These models generate coherent sequences by sampling one token at a time, conditioned on the history. For a sequence of tokens $x = (x_1, x_2, \dots, x_T)$, the probability of the sequence is given by:

$$P(x) = \prod_{t=1}^T P(x_t | x_{<t}),$$

Transformer Architecture

Transformers [5] are the architectural foundation of modern LLMs. They process token sequences through self-attention and feedforward layers. Decoder-only variants (as in GPT models) employ causal masking to ensure each position attends only to previous tokens. The self-attention mechanism computes weighted combinations of value vectors using attention scores derived from query-key interactions:

$$Attention(Q, K, V) = softmax \left(\frac{QK^T}{\sqrt{d_k}} + M \right) V,$$

where $Q, K, V \in \mathbb{R}^{t \times d_k}$ are the query, key, and value matrices, respectively, and M is a mask matrix that enforces causality. The attention scores are scaled by $\sqrt{d_k}$ to stabilize gradients

during training. The final hidden state h_{t-1} is passed through a linear layer and softmax to produce the next-token distribution:

$$P(x_t|x_{<t}) = \text{softmax}(Wh_{t-1} + b)$$

Training and Inference

LLMs are trained to minimize the negative log-likelihood of observed sequences:

$$\mathcal{L}_{NLL} = -\sum_{t=1}^T \log P(x_t|x_{<t}; \theta),$$

where θ denotes model parameters.

At inference time, LLMs generate text by sampling from the learned distribution. Several decoding strategies can be employed with different trade-offs between diversity and coherence of generated text:

- Greedy decoding selects the token with the highest probability at each step.
- Top- k sampling samples from the k -highest probability tokens.
- Top- p sampling samples from the top p -percentile of the distribution.
- Beam search maintains multiple hypotheses in parallel and expands them based on cumulative probabilities.

Scaling Laws and Emergent Behavior

Empirical work [6] has shown that LLM performance follows scaling laws: loss decreases as model size, dataset size, and the amount of compute used for training increase. When models grow beyond a certain scale, they exhibit emergent behaviors that were not explicitly

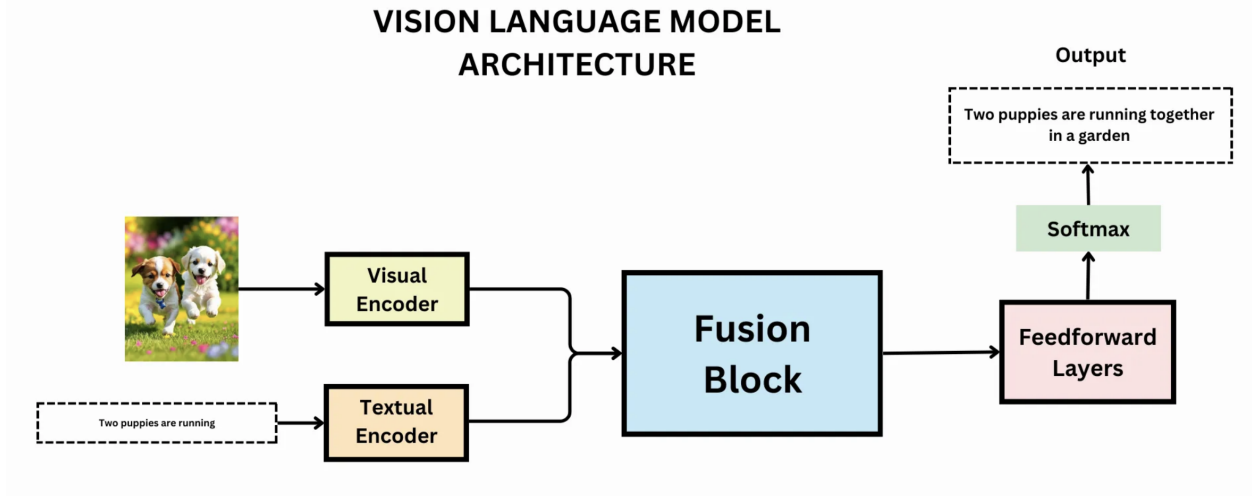


Figure 2.2: Vision-Language Model Architecture

trained for. These behaviors include in-context learning, logical reasoning, and zero-shot generalization. Specifically, LLMs can learn to perform tasks by conditioning on examples provided in the input prompt, even when they have not been explicitly trained on those tasks. This is achieved through the model’s ability to leverage its vast training data and internal representations to generalize from the context provided in the prompt.

Another notable emergent behavior is compositional reasoning, particularly when aided by chain-of-thought (CoT) prompting. CoT is a prompting technique that encourages the model to generate intermediate reasoning steps before arriving at a final answer. This step-by-step reasoning process can lead to more accurate and interpretable outputs, as the model is pushed to simulate internal reasoning processes.

Vision-Language Models

Vision-Language Models (VLM) are a type of LLM that can process both visual and textual information. They are typically pre-trained on a large corpus of text and images, and then fine-tuned for specific tasks.

Vision language models typically consist of four key blocks: vision encoder, language encoder, fusion block, and output block (Figure 2.2). Text features such as word embeddings

and visual features such as image regions or patches are extracted using a text encoder and visual encoder. A multimodal fusion module then combines these independent streams, producing cross-modal representations. A decoder translates these representations into text or other outputs for generation-type tasks, such as captioning, translation, and question answering.

2.1.3 Representation Learning

Representation learning is a critical component of modern RL and planning systems. It enables agents to learn compact, structured representations of high-dimensional observations, such as images or text, that are relevant for decision-making.

Formally, for observation $O \in \mathcal{O}$, the goal of representation learning is to learn a mapping $f_\phi : \mathcal{O} \rightarrow \mathbb{R}^d$ that transforms the raw observation into a lower-dimensional latent representation $z \in \mathbb{R}^d$. This learned representation can then be used for downstream tasks such as policy learning, planning, or value estimation.

An effective representation should capture the essential features of the observation while discarding irrelevant information. It should reduce the dimensionality of the input space, making it easier for the agent to learn and generalize. The representation should also be robust to noise and variations in the input data to ensure the agent can make reliable decisions based on the learned features.

Multimodal Applications

In multimodal models, different inputs, such as vision, language, and audio, must merge into a common latent space. Pretrained models like CLIP [7] learn shared embeddings for text and images through contrastive learning – Given a batch of image-text pairs, CLIP learns to maximize the cosine similarity between the image and text embeddings. These pretrained embeddings can serve as foundations for policies, exploration strategies, or planning algorithms.

Temporal and Relational Structure

Sequential decision-making requires representations that capture temporal and relational structure. Recurrent neural networks (RNNs) and long short-term memory (LSTM) networks are commonly used to model sequences of observations. They maintain a hidden state that encodes information about past observations. Graph neural networks (GNNs) are commonly used to model relational structures, such as social networks or multi-agent systems, where the relationships between entities are important for decision-making.

2.2 Related Work

In this section, we discuss existing works that employ Large Language Models (LLMs) and Vision Language Models (VLMs) for planning in both *model-free* and *model-based* settings, highlighting their respective challenges and the solutions proposed so far.

A pioneering work in this area is SayCan [8]. By structuring the interaction between a user and a robot as a question-answer dialogue, SayCan lets an LLM provide high-level task decompositions while the robot grounds those instructions in the physical world. The study exposes two recurring difficulties for language-driven planning: (i) LLMs can hallucinate impossible actions when geometric or kinematic constraints are not represented in the prompt, and (ii) grounding abstract language onto low-level motor commands requires a reliable perception stack. SayCan mitigates (i) with a probabilistic model that scores how feasible an LLM-suggested action is for the robot, and tackles (ii) through a learned language-to-skill mapping.

Zero-shot and few-shot planners. Huang et al. [9] show that, with carefully designed prompts, GPT-3 and Codex act as zero-shot planners: for complex household goals such as “make breakfast,” the model outputs a sensible sequence of actions. Building on this, LLM-Planner [10] demonstrates that injecting only a handful of in-context examples (few-shot) already surpasses specialized Vision-Language Navigation (VLN) policies that are trained

on orders of magnitude more data, indicating that there are commonsense priors stored in LLMs.

Structured input representations. Several works improve robustness by feeding the LLM structured state descriptions instead of raw text. Examples include tabular representations [11], 3D scene graphs [12], and other symbolic abstractions. While these formats reduce ambiguity and help the model reason about discrete object relations, they incur non-trivial pre-processing costs in unstructured real-world scenes and can bottleneck at perception errors.

Vision input and object grounding. Advances in VLMs enable planners to bypass manual symbol construction and consume pixels directly. [13] confirms that a frozen VLM can retrieve objects from images given free-form natural-language queries (e.g., a cat-shaped mug), opening the door to visually grounded planning. However, most popular VLMs inherit the bag-of-words limitation of CLIP [7] because they treat a scene as an unordered set of tokens and ignore spatial relations. [14] addresses this by prompting the LLM to reason over coarse location cues, reducing spatial hallucinations without expensive fine-tuning.

2.2.1 Model-based Planning with LLMs

LLMs as world-model builders. A complementary line of work keeps a symbolic planner in the loop and asks the LLM to generate the domain description instead. Guan et al.[15] translate natural language task specifications into PDDL domain and solve them with off-the-shelf planners, delivering correctness guarantees that pure language pipelines lack. The same LLM can translate solver feedback back into natural language, closing the teaching loop for non-expert users.

VLM-TAMP [16] combines the semantic richness of VLMs with the geometric exactness of Task-and-Motion Planning (TAMP). The VLM proposes intermediate subgoals or parameterized actions in PDDL, reducing the horizon that TAMP must optimize over. TAMP then rejects infeasible proposals and triggers re-prompting.

Multi-agent extensions such as COMBO [17] rely on a fine-tuned LLaVA *action proposer* to sample candidate joint actions, which are then rolled out inside a learned world model to pick cooperative strategies. It provides a promising template for scaling language guidance to decentralized settings.

Chain-of-Thought (CoT) prompting. A recurring theme across both model-based and model-free pipelines is to ask the LLM to explain its reasoning before committing to an action, especially on long horizons. CoT has been shown to be useful in symbolic planning [9], high-level navigation [10], and even visual decision making when combined with VLMs [18].

2.2.2 Model-free planning with LLMs

LLMs/VLMs as policy encoders. Model-free RL methods commonly freeze the language backbone and learn a lightweight controller on top. RoboFlamingo [19] fine-tunes OpenFlamingo on language-conditioned manipulation clips, then trains a small history encoder that achieves state-of-the-art success on CALVIN while remaining deployable on commodity GPUs. PR2L [18] show that task-specific prompts fed into a frozen VLM yield semantic embeddings that improve both sample efficiency and generalisation in Minecraft and Habitat tasks; incorporating CoT yields further gains.

Even with strong language priors, learning end-to-end policies can overfit to spurious image details. Codebook bottlenecks [20] act as top-down attention filters, discarding irrelevant visual features and delivering smoother trajectories and better domain transfer.

Open challenges. Despite rapid progress, three issues persist: (i) *Reliability*—LLMs still hallucinate unsafe or impossible actions under distribution shift; (ii) *Scalability*—prompt engineering or structured scene extraction can dominate wall-clock time in complex environments; (iii) *Interpretability*—understanding why an LLM chose a specific action remains difficult, hampering debugging and safety analysis. Research on selective attention, CoT explanations, and tighter TAMP integration offers promising avenues for tackling these challenges.

In summary, model-based uses of LLMs excel at reasoning over long horizons but can

bottleneck on inaccurate symbolic translations; model-free uses inherit the robustness of end-to-end learning yet may under-utilize the rich priors stored in the language model. Hybrid systems like VLM-TAMP and COMBO suggest a unifying direction: invoke the LLM only when its abstract reasoning is most valuable—sub-goal discovery, commonsense filtering—while delegating geometry-aware control and gradient-based credit assignment to classical planners or learned policies.

Chapter 3

System Design

3.1 System Overview

We design a model-based planning framework that uses pretrained VLMs to create a unified representation for both visual and language inputs. Figure 3.1 illustrates the overall architecture of our system, which consists of four main components: state abstractor, policy model, dynamics model, and outcome evaluator.

The state abstractor encodes the agent’s current visual observations and task-relevant memory into a compact belief state. This belief state serves as the single source of truth about the world, capturing both visual layout and semantic context without requiring handcrafted features or separate perception modules. The policy model is a lightweight transformer that predicts future belief states under candidate actions. It operates directly based on the VLM latents, so it inherits the foundation model’s prior knowledge. The dynamics model is trained in a supervised fashion to roll out the belief state over time, simulating the effects of taking different actions. This enables the agent to explore potential future states and evaluate the consequences of its decisions. Finally, the outcome evaluator is responsible for scoring imagined futures by generating natural language reasoning about goal achievement. This component provides a human-interpretable summary of the agent’s predictions, which enables

human-in-the-loop inspection and intervention.

During inference, the system uses a Monte Carlo tree search (MCTS) framework, where the policy model suggests candidate actions, the dynamics model rolls each action forward in the latent space, and the outcome evaluator produces a score and reasoning trace for each imagined future.

We explain in detail the design for the state abstractor and policy model in Section 3.4 and Section 3.5, respectively. The dynamics model is discussed in Section 3.6.

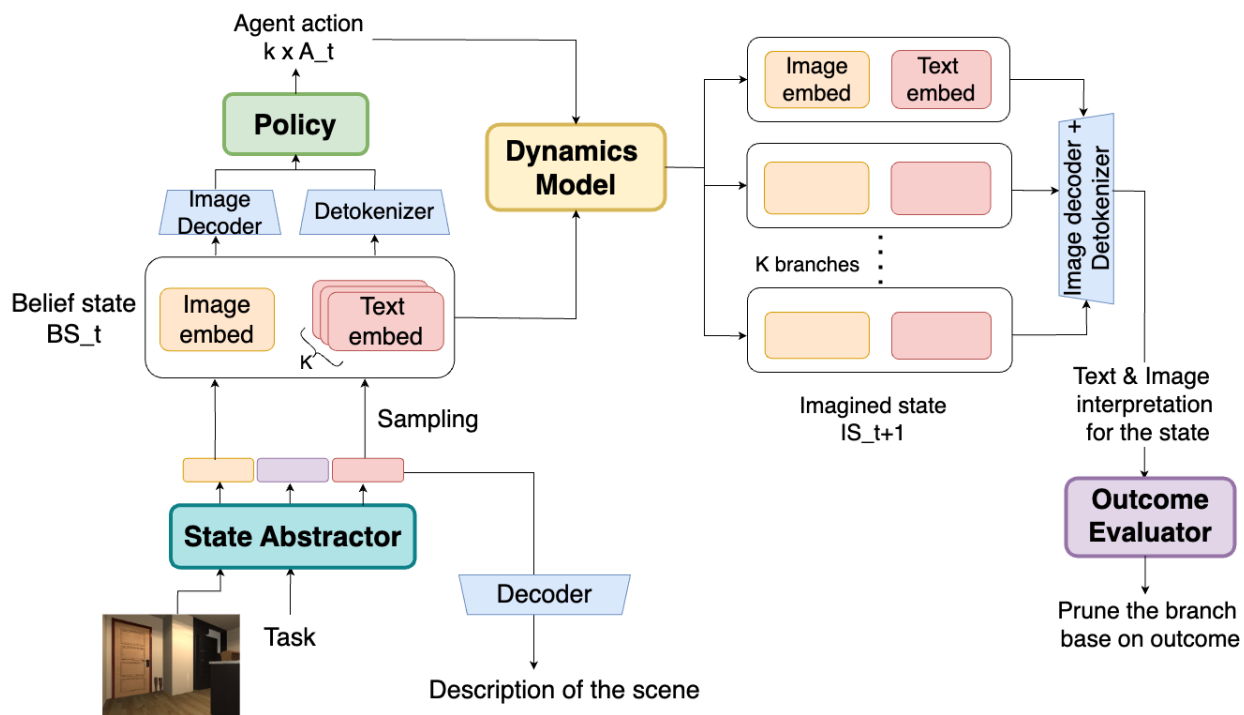


Figure 3.1: System Overview

3.2 Environment

Our experiments are conducted within ProcTHOR [21], an extension of the AI2-THOR simulator [22] that expands the scope and diversity of available indoor environments.

The simulator provides photorealistic renderings of household scenarios including kitchens, living rooms, bedrooms, studies, and bathrooms (Figure 3.2). Each generated scene incorpo-

rates physically realistic properties through AI2-THOR’s physics engine, enabling complex interactions such as object manipulation, container opening, and multi-step task execution. The environment’s perception system offers both RGB images and depth information, complemented by structured semantic metadata including object identifiers, semantic labels, and 3D bounding boxes for all interactable elements within the scene.

For trajectory collection and evaluation, we utilize the `procthor-r1` codebase [21], which provides standardized interfaces for reinforcement learning research in procedurally generated environments. The action space encompasses discrete navigation primitives including forward movement (`MoveAhead`), rotational adjustments (`RotateLeft`, `RotateRight`), and vertical camera control (`LookUp`, `LookDown`), alongside interaction primitives such as `PickupObject`, `OpenObject`, and `End` for task completion.



Figure 3.2: An example of ProcTHOR environment

3.3 Dataset

We evaluate our approach on the *object navigation* benchmark that requires agents to locate specific objects within complex indoor environments. The task definition consists of three key components: (1) an instruction specified as a natural language phrase (e.g., “find the

blue mug”), (2) an objective requiring navigation from a randomized starting position to within one meter of the target object, and (3) termination criteria based on either successful proximity achievement or failure after 500 timesteps.

3.3.1 Expert Trajectories

We collect the expert trajectories using the `proctor-rl` framework and use them as ground truth for policy learning. We use pre-trained model checkpoints from [21] to generate the expert trajectories.

We collected 2,000 trajectories distributed across 10 procedurally generated houses with 18 different types of target objects. This ensures statistical diversity in spatial layouts, object configurations, and task complexity. Each trajectory records a complete state-action sequence at every timestep, capturing RGB image from the agent’s camera, the action taken, and metadata including object positions, semantic labels, and scene configuration details.

3.3.2 Generating Scene Description

To train our state abstractor, we need to generate scene descriptions from the RGB frames. We explored three methods for generating scene descriptions: (1) querying GPT-4 directly, (2) constructing scene descriptions from metadata, and (3) using concise seed captions. In this section, we discuss each method in detail and evaluate its effectiveness in generating accurate and informative scene descriptions.

Query GPT-4

Large language models (LLMs) like GPT-4 have shown impressive capabilities in generating natural language descriptions from visual inputs. Our initial approach was directly querying GPT-4 with each RGB frame, using a fixed prompt such as “Imagine you are a robot navigating in a household. Describe what you see in the current scene.”



		
	Task: Find Vase	Task: Find Bowl
Basic “Describe the most important details in the scene.”	A monitor is on the top shelf, while the lower shelf holds a bowl, a green box, a cylindrical vase, a phone, and a wooden object.	The scene features a table with art supplies such as a pencil, eraser, sharpener, and a small sculpture or model.
Task-Relevant “Imagine you are an agent trying to get the {target_object}. This image shows the scene inside a house. Based on this image, describe the most important details.”	The vase is located on the bottom shelf of a wooden table, next to a phone, a ceramic bowl, and a rectangular green box.	The Bowl is visible on the lower shelf of a black table near a wall, next to a notebook, pen, candle, and a small stuffed animal.
Task-Relevant + Reasoning “You are given an image of a scene inside a house. Imagine you are an agent trying to get the {target_object}. Based on this image, reason about the most important details to help you solve the task.”	The vase is located on the lower shelf, next to a bowl and a smartphone; reach directly for it without moving other objects.	The bowl is on the lower shelf of the black table next to the wall.
Task-Relevant + History + Predict “You are given an image of a scene inside a house. Imagine you are an agent trying to get the {target_object}. You have taken {actions_taken} previous actions. Based on this image, reason about the most important details and predict the next action that would help you solve the task.”	MoveAhead to get closer to the table and access the bowl.	Take the 'Pickup' action to obtain the vase from the shelf.

Table 3.1: Comparison of scene descriptions generated with different prompts.

Table 3.1 compares scene descriptions generated from GPT-4 using prompts with different focus and specificity. The basic prompt produces generic descriptions. Inspired by [18], we tried task-relevant prompts. They help localize the target object more precisely, as seen in the *task-relevant* columns. However, it introduces hallucinated objects – in the column with bowl as the target object, GPT-4 falsely identifies a bowl while there is no bowl in the image. Adding reasoning and history (actions taken) to the prompt do not help much – the model still suffers from hallucinations.

In summary, LLMs frequently hallucinated objects not present in the image and struggled to accurately capture spatial relationships between objects. This led to inconsistencies in the generated scene descriptions, making them less reliable for training our state abstractor.

Reconstruct From Metadata

To address the limitations of directly querying GPT-4, we explored an alternative approach that leverages the simulator’s internal event metadata. This metadata provides detailed information about the objects in the scene, their 3D coordinates in the world frame, bounding boxes, and semantic labels such as IDs, colors, and textures. We explored the possibility of reconstructing deterministic scene descriptions directly from the metadata.

Specifically, we save the event metadata at each timestep to some JSON files, then process the metadata through a series of geometric and spatial reasoning steps to produce a structured textual description of the scene. Here are the steps we followed:

- We first isolate only the objects marked as visible according to the simulator’s rendering engine. Discard any entities that are outside the camera’s field of view or flagged as invisible due to rendering occlusion.
- Then we adapt a more sophisticated occlusion filtering mechanism that accounts for partial occlusion between objects. For every pair of visible objects, we compute their projected sizes and viewing angles from the agent’s perspective. If a nearer object covers a substantial portion of a farther object (exceeding a configurable threshold), we

remove the occluded object from further consideration. This step avoids generating descriptions that mention objects the agent cannot meaningfully perceive.

- Next, we construct a spatial scene graph representation where nodes correspond to the agent and each remaining visible object. We establish *perceives* edges directed from the agent to each object, annotated with multiple spatial properties: Euclidean distance in meters, horizontal relationship categories (front/back, left/right), vertical relationship categories (above/below), and coarse distance classifications (very close, close, moderate, far). Additionally, for objects within a proximity threshold of 2 meters, we compute precise relative spatial relationships and create bidirectional *spatial* edges between object pairs, capturing their relative positions in both horizontal and vertical dimensions. To capture the inside-outside relationship, we add *contains* and *contained-by* edges derived from parent-receptacle relationships recorded in the simulator’s metadata.
- Finally, we generate a textual description of the scene by traversing the scene graph and extracting relevant information. We use a template-based approach to construct sentences that describe the agent’s perception of the environment, including an overview of visible objects, agent-centric grouping, agent-object relations, and object-object relations.

An example of the generated scene description for the image in Figure 3.3 is shown in Appendix A.1. The description captures the spatial relationships between objects and provides a more structured representation of the scene.

While this method produced more accurate descriptions than querying GPT-4 directly, it is too verbose and suffers from rigid structural patterns that can lead VLMs to overfit on formatting rather than semantic content.



Figure 3.3: An example of a scene image

More Concise Seed Captions

To mitigate the issue of rigid structural patterns and to prevent overfitting on formatting, we condense each raw scene description into concise seed captions that capture essential spatial relationships and important objects.

We first extract and clean the raw scene metadata by removing object counts, redundant information, and irrelevant details while preserving essential spatial relationships. The cleaned data is then processed by GPT-4 using a structured prompt:

Prompt for Concise Seed Captions

Here are the objects within moderate distance of the agent:
{clean_input}

Task: Write ONE grammatically correct sentence that tells a robot where the key objects are. Mention up to three landmarks and their position relative to the agent. Do NOT list counts or “Scene with...”.

For example: If I gave you:

- Bowl is in front at same height.
- Box is in front at same height.
- Chair is in front below.

You would write: *“A bowl and box rest on a low surface directly ahead of the agent, with a chair slightly below and in front.”*

Now summarize the real scene above.

The output of this prompt is a concise caption that captures the essential spatial relationships and important objects in the scene. For example, the output for the scene in Figure 3.3 is condensed into the following concise caption:

Concise Caption for Scene in Figure 3.3

“The mug is positioned directly in front of the agent at the same level, while the sofa and basketball are located to the agent’s left and below, and the TV stand is situated in front of and below the agent.”

These seed captions serve as ground truth for training the textual component of our belief state representations, which ensures that the learned encodings capture meaningful spatial semantics rather than merely memorizing the structural patterns of template-generated descriptions from metadata.

3.4 State Abtractor

The state abtractor (Figure 3.4) serves as the bridge between raw observations from the environment and the high-level representations needed for planning and decision-making. It receives an RGB frame from the simulator along with a text prompt and transforms them into a semantic representation for downstream policy and dynamics modules. We use open-source vision-language models (VLM) to produce multimodal embeddings of the observations. Since the VLMs are pretrained on large-scale image-text pairs, they inherently capture rich representations that fuse both visual and textual information. Specifically, we use BLIP-2 [23] and Qwen-VL [24] as our VLMs. The state abtractor is a frozen model that is fine-tuned on the task-specific dataset.

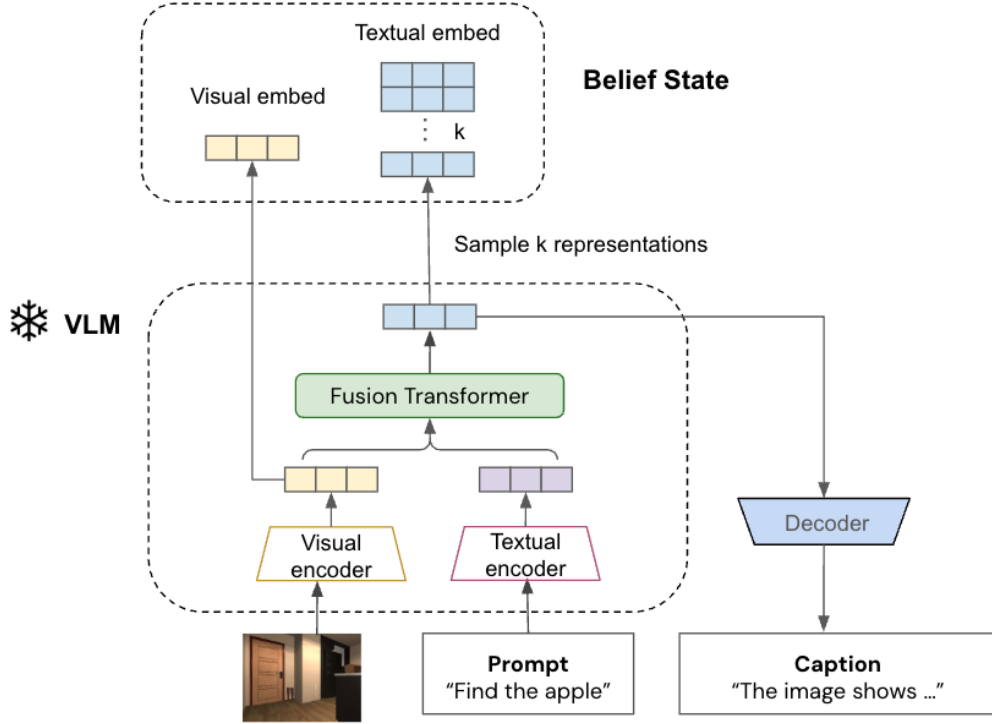


Figure 3.4: State abtractor

3.4.1 Formal Mapping

At every timestep, the robot perceives a raw observation

$$O_t = (I_t, u_t)$$

where $I_t \in \mathbb{R}^{H \times W \times 3}$ is an RGB frame and u_t is a text instruction (fixed prompt). We want to learn a mapping ϕ_θ from the observation space to a multimodal representation \mathbf{h}_t and a natural language caption y_t :

$$\phi_\theta : O_t \mapsto (y_t, \mathbf{h}_t) \quad (3.1)$$

Visual encoder. The visual encoder f_v is a ViT (BLIP) or EVA-CLIP (Qwen-VL). It produces a sequence of patch embeddings from the input image.

$$\mathbf{v}_t = f_v(I_t; \theta_v) \in \mathbb{R}^{L \times d} \quad (3.2)$$

where L is the number of patches and d is the embedding dimension.

Text encoder. The textual prompt is first tokenized into a sequence of token IDs, which are then mapped to word embeddings by the text encoder.

$$\mathbf{x}_{1:P} = f_t(u_t; \theta_t) \in \mathbb{R}^{P \times d} \quad (3.3)$$

where P is the number of tokens in the prompt. The prompt is fixed during training.

Prompt Fusion. The image and text embeddings are fused and processed by a stack of cross-attention blocks or transformer g_θ to produce a d -dimensional hidden vector \mathbf{h}_t :

$$\mathbf{h}_t = g_\theta(\mathbf{v}_t, \mathbf{x}_{1:P}) \quad (3.4)$$

The final hidden state is the multimodal representation of the observation.

Caption Decoding. A decoder can autoregressively produce a sentence Y_t as the natural language caption for the image.

$$P_\theta(y_k | y_{<k}, O_t) = \text{Softmax}(W \mathbf{h}_k), \quad (3.5)$$

$$y_t = \arg \max_{x_{1:P}} \prod_{k=1}^P P_\theta(x_k | x_{<k}, O_t) \quad (3.6)$$

3.4.2 Training Objective

Each training sample is a triplet (I_t, u_t, y_t^*) of the current frame, textual prompt, and the ground-truth sentence describing the scene.

In the forward pass, the model produces a sequence of hidden states \mathbf{h}_t and a sequence of predicted tokens y_t from the input image and prompt. The model is trained to minimize the cross-entropy loss between the predicted tokens and the ground-truth caption.

$$\mathcal{L}_{\text{CE}} = - \sum_{k=1}^T \log P_{\theta}(y_k^* | y_{<k}^*, I_t). \quad (3.7)$$

If a frozen teacher model $\tilde{\theta}$ is available, we add a knowledge-distillation term, $\mathcal{L}_{\text{KD}} = \tau^2 \text{KL}(P_{\tilde{\theta}} \| P_{\theta})$, and the total loss becomes

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda_{\text{KD}} \mathcal{L}_{\text{KD}}. \quad (3.8)$$

During fine-tuning, we *freeze* all original weights of the vision encoder f_v , the text tokenizer f_t , and the multimodal transformer g_{θ} . We insert low-rank adapters (LoRA) [25] into every linear projection of the attention and MLP blocks, and train only the LoRA parameters.

3.4.3 Generating Belief States

The state abstractor ϕ_{θ} produces a belief state BS_t that is a compact representation of the current observation O_t .

After the forward pass, the VLM encoder outputs the image patches $\mathbf{v}_t \in \mathbb{R}^{L \times d}$ and the prompt tokens $\mathbf{x}_{1:P} \in \mathbb{R}^{P \times d}$. The fusion module (transformer) processes these inputs and outputs a textual embedding $\mathbf{h}_t \in \mathbb{R}^{M \times d}$ that can be decoded into a natural language caption y_t describing the scene.

The belief state BS_t consists of two components, as shown in Figure 3.4:

- **Visual embedding \mathbf{v}_t :** The raw visual tokens from the image encoder that capture the spatial layout and visual features of the scene.
- **Textual embedding \mathbf{h}_t :** The output of the fusion module that encodes the semantic understanding of the scene. This embedding can be decoded into a natural language caption through the caption decoder.

By maintaining both visual and textual embeddings, we preserve both the low-level visual features and high-level semantic understanding of the scene. This dual representation enables

efficient updates to the policy and dynamics modules, as they can operate on either the visual or textual modalities depending on their needs.

In summary, the state abstractor ϕ_θ outputs both a human-readable caption and a vector embedding that grounds high-dimensional observation for downstream planning and dynamics learning. We separate the vision and language space of the belief state, and we maintain K caption samples per timestep to balance diversity with correctness in downstream reasoning.

3.5 Policy Model

The policy model is a lightweight transformer that takes the belief state BS_t as input and outputs a distribution over possible actions that the robot takes in each step. Figure 3.5 shows the architecture of the policy model. It is trained using behavior cloning, where the model learns to predict the actions taken by an expert agent in the training data.

We let the same frozen VLM encoder in section 3.4 that processes real observations also process imagined ones. That guarantees a shared embedding space and avoids retraining the policy from scratch.

3.5.1 What the Module Does

The policy model consists of a frozen VLM encoder, a recurrent GRU layer, and a linear head that outputs the action logits.

Reconstructing sensor-level inputs Given the separated belief state $BS_t = \{\mathbf{v}_t, \{\mathbf{x}_{1:P_t}^{(k)}\}\}$, we first map each modality back to its native domain:

$$\begin{aligned}\tilde{I}_t &= \text{ImgDec}(\mathbf{v}_t), \\ \tilde{u}_t^{(k)} &= \text{Detok}(\mathbf{x}_{1:P_t}^{(k)}), \quad k = 1, \dots, K.\end{aligned}$$

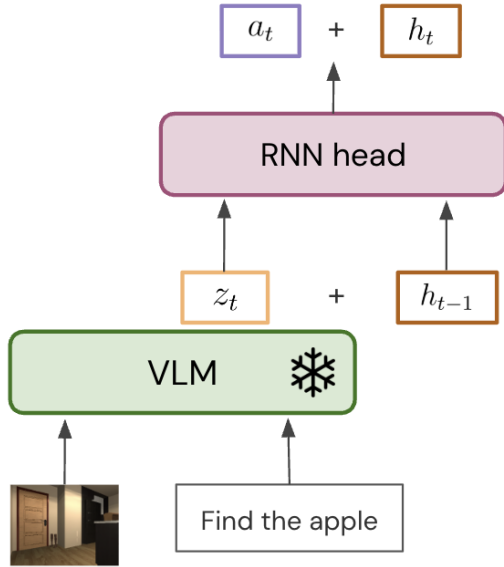


Figure 3.5: Policy Model Architecture

The image decoder `ImgDec` (VQ-GAN style) turns the visual latent embedding \mathbf{v}_t back into an RGB image \tilde{I}_t . The detokenizer `Detok` converts each sampled caption token sequence $\mathbf{x}_{1:P_t}^{(k)}$ into the natural language description $\tilde{u}_t^{(k)}$ of the scene. The reconstructed image and instruction are then passed to the policy model.

The reason we decode images and text back is that the same frozen VLM encoder can be reused for real and imagined frames since now they share the same embedding space.

VLM encoding and GRU policy head At each timestep t , each pair $(\tilde{I}_t, \tilde{u}_t^{(k)})$ is passed through the frozen VLM encoder to generate a multimodal embedding

$$z_t^{(k)} = f_{\text{VLM}}(\tilde{I}_t, \tilde{u}_t^{(k)}),$$

The embedding z_t is concatenated with the previous recurrent state h_{t-1} and fed to a single-layer GRU policy head:

$$(h_t, l_t) = \text{GRU}([z_t; h_{t-1}]),$$

yielding an updated hidden state h_t and logits $l_t \in \mathbb{R}^{|\mathcal{A}|}$ over the discrete action set $\mathcal{A} = \{\text{MoveAhead}, \text{RotateLeft}, \text{RotateRight}, \text{LookUp}, \text{LookDown}, \text{End}\}$. The action executed in the environment is $a_t = \arg \max_{a \in \mathcal{A}} l_t^{(a)}$.

3.5.2 Policy Training

We train the policy model using behavior cloning on a dataset of expert trajectories. The dataset is collected in the AI2-THOR simulator as detailed in section 3.3. Each trajectory consists of a sequence of images and the corresponding actions taken by the expert agent at each time step. The policy model is trained to predict the action given the image and text prompt describing the scene and the task.

The training objective is to minimize the cross-entropy loss between the predicted action distribution and the ground-truth action taken by the expert agent. We define the loss function as follows:

$$\mathcal{L}_{\text{BC}} = -\frac{1}{T} \sum_{t=1}^T \log p_{\theta}(a_t^* | I_{1:t}, u),$$

where a_t^* is the ground-truth label at time t .

3.6 Dynamics Model

The Dynamics Model predicts future observations, rewards, and termination flags from past observations and actions. It consists of two autoregressive transformer models: the Vision Dynamics Model (VDM) and the Language Dynamics Model (LDM). The VDM predicts future visual tokens, while the LDM predicts future textual tokens. Both models are trained on a dataset of trajectories collected in the AI2-THOR simulator, where each trajectory consists of a sequence of images, actions taken, rewards received, and done flags indicating whether the episode has ended.

To avoid training the visual dynamics model on high-dimensional raw pixel data, we pre-process the images using a VQ-GAN-style tokenizer to convert them into discrete tokens.

The tokenizer maps each image patch to a discrete token in a learned codebook, allowing the model to operate on a lower-dimensional discrete representation of the image. Each RGB frame is divided into a fixed grid of small patches (e.g., 16×16 pixels), from which feature vectors are extracted and matched to their nearest neighbors in a learned codebook. This produces a single integer token per patch, which are then assembled into a compact sequence. Rather than processing the original $3 \times 224 \times 224$ float image, the vision dynamics model receives this significantly shorter discrete token sequence. This tokenization process reduces the input size and allows the model to focus on the most relevant features of the image, improving training efficiency and performance.

The VDM and LDM are trained separately but share the same architecture and training procedure. The models are trained to minimize the cross-entropy loss between the predicted tokens and the ground-truth tokens, as well as the Huber loss for rewards and binary cross-entropy for done flags.

3.7 Outcome Evaluator

The outcome evaluator sits at the end of our planning pipeline and answers two questions about each imagined future: “How likely is this path to succeed?” and “Why?” Internally, it feeds the same frozen vision-language model that our policy uses into two small heads. One head outputs a single number between 0 and 1 — our confidence that the agent will achieve its goal if it follows this trajectory. The other head generates a brief, natural language sentence explaining the reasoning behind that confidence. During training, we teach the score head with binary success labels and train the language head on example explanations. At inference time, the score prunes our Monte Carlo tree search toward the most promising branches, while the explanation head provides a human-readable justification for the evaluator’s judgment.

Chapter 4

Experiments and Results

4.1 Training the VLMs

4.1.1 Training Setup

We fine-tune InstructBLIP and Qwen2.5-VL with LoRA adapters on image-caption pairs. For every sample, the dataloader loads an RGB tensor from disk, applies the BLIP processor’s resize and normalization to produce a $3 \times 224 \times 224$ tensor, and tokenizes the full scene description as both `input_ids` and `labels` (teacher forcing). We accumulate gradients over four mini-batches before each optimizer step to stay within GPU memory. We train our models on a single GPU lab machine (RTX 4090) and on MIT SuperCloud nodes with Volta GPUs. The training and fine-tuning hyperparameters are shown in Table B.1 and Table B.2 in Appendix B.1.

4.1.2 Performance Bottlenecks

On the lab machine, training is bottlenecked by CPU-side pre-processing. Each image is repeatedly converted between tensor, NumPy, and PIL formats. We used the raw scene description generated from metadata as the ground-truth, where one ground-truth description can have around 300-500 words, depending on how many objects are in the image. These

long captions expand to approximately 300-600 tokens, which loads the decoder with heavy compute requirements.

On MIT SuperCloud, raw compute scales almost linearly, but the file I/O becomes the rate limiter. The dataset is stored on a shared filesystem, and every worker must stream tens of gigabytes of image tensors at the start of each episode. The resulting I/O stalls dominate wall-clock time even though NCCL parameter broadcasts are quick.

4.1.3 Mitigations

To address the CPU-side pre-processing bottleneck, we cache the pre-processed tensors and tokenized captions once offline. This allows us to increase the number of dataloader workers to 8-16 and enable `pin_memory` for faster data transfer to the GPU.

To reduce the decoder’s compute and memory requirements, we replace the full scene descriptions with more concise seed captions as discussed in 3.3.2. This change can cut the decoder’s compute and memory requirements roughly in half.

On MIT SuperCloud, we can move the dataset from the shard filesystem to each node’s local NVMe SSD at job startup by `rsync` in the SLURM prolog. We can then launch training with `torchrun` and a `DistributedSampler` to ensure that each worker gets a unique subset of the data. To exploit the two Volta GPUs allocated per node on MIT SuperCloud, we can wrap the VLM model in `torch.nn.DataParallel` so that batches are split across both GPUs. This change alone could increase per-node throughput without altering the training loop.

Due to time constraints, we did not implement all of the above changes. We only cached the pre-processed tensors and tokenized captions. This change cut average per-step time on the lab machine from 6.8s to 1.9s, and shrank a full-epoch run on SuperCloud from roughly 5 hours to one hour. We believe that the other changes would yield more performance improvements and reduce training time further.

4.2 Evaluation for Policy Learning

During policy training, we encountered several challenges that impact policy performance. The primary issue is that the policy almost exclusively predicts the **MoveAhead** action regardless of the context. Example roll-outs are shown in Appendix B.2.1.

This issue is caused by the inherent *class imbalance* in the dataset that the policy model is trained on. The **MoveAhead** action comprised approximately 60% of all actions in the training data. This skewed distribution biases the policy model towards predicting the majority class, as cross-entropy loss penalizes incorrect predictions more heavily for the majority class. To address this, we applied inverse-frequency class weights to the loss function, assigning higher weights to underrepresented actions to encourage more balanced predictions. However, this approach only slightly improved the model’s performance, increasing the F1 score by approximately 0.05 points.

We also attempted standard regularization techniques, including dropout and early stopping to prevent overfitting, hypothesizing that the model might be memorizing training sequences rather than learning generalizable behaviors. However, these methods showed no significant impact on action diversity.

In our early experiments, we treated each $\langle BS_t, a_t^* \rangle$ pair in isolation, i.e., shuffled single-step batches. This made the hidden state h_{t-1} effectively re-initialized at every step, preventing the GRU from remembering the history. Training on the *full-episode* preserves recurrent context and allows the policy to condition on history when choosing actions. This did not by itself eliminate model collapse, but it was necessary for the GRU to leverage the belief state beyond the current frame.

A secondary factor contributing to poor performance was the *distribution shift* between the VLM’s pre-training data and our task domain. The vision-language model had been trained on web-scale image-text pairs representing diverse, global viewpoints, whereas our task required understanding egocentric, indoor scene navigation. This mismatch resulted in suboptimal feature representations for our specific use case, as the pre-trained encoder was

not optimized for first-person perspective navigation tasks. To mitigate this, we considered fine-tuning the VLM backbone on our dataset to align the feature space more closely with the task requirements. However, this approach was not pursued in the current implementation due to time constraints.

Finally, we observed that the model’s performance degraded over long roll-outs due to *covariate shift*. The policy model was trained on a dataset of expert trajectories, but during inference, it was required to make predictions based on its own actions. This discrepancy between training and inference led to compounding errors, as the model’s predictions became increasingly inaccurate over time. To address this issue, we implemented truncated backpropagation through time (BPTT) with a length of 20 steps. While this stabilized training and reduced gradient-related numerical issues, it did not improve the model’s ability to predict diverse action sequences.

4.2.1 Future Improvements

To address the limitations identified during policy training, we propose several directions for future research that target both the structural deficiencies of behavior cloning and the architectural constraints of our current system.

On-Policy Reinforcement Learning The single-action collapse and covariate drift issues inherent in behavior cloning can be addressed by transitioning to on-policy reinforcement learning methods. We propose warm-starting the policy from our current behavior-cloned model and then fine-tuning it using proximal policy optimization (PPO) [26] or Q-learning approaches. The reward structure would combine sparse success signals (reaching the target object or location) with intrinsic curiosity bonuses to encourage exploration of diverse action sequences. This approach would allow the agent to learn from its own experience rather than being limited to the distribution of expert demonstrations, potentially discovering more efficient navigation strategies while maintaining the representation benefits of the pre-trained

VLM.

Alternative Model Architectures Our current GRU-based recurrent architecture may be limiting the policy’s ability to maintain long-term context and execute complex action sequences. We propose exploring more sophisticated architectures including Transformer-XL, which offers better long-range dependency modeling through segment-level recurrence, or GRU-D (GRU with Decay) [27], which incorporates temporal decay mechanisms to better handle irregular time steps. Additionally, we suggest investigating split decoder architectures that separate locomotion actions (`MoveAhead`, `RotateLeft`, `RotateRight`) from camera control actions (`LookUp`, `LookDown`), allowing each subsystem to specialize in its respective control domain. This architectural modularity could improve both performance and interpretability while reducing the complexity of the joint action prediction task.

Chapter 5

Conclusions

This thesis has developed a unified, model-based planning framework that leverages pretrained vision language models to bridge perception, language, and decision-making in embodied agents. We began by designing a state abstractor that uses BLIP-2 and Qwen-VL backbones with LoRA adapters to convert raw RGB frames and task prompts into compact belief states, combining multimodal embeddings with natural language hypotheses. To make high-resolution images tractable for transformer-based dynamics models, we introduced a discrete image representation module that tokenizes each image into a small sequence of learned codebook indices. Our vision and language dynamics models then predict future belief states, rewards, and termination signals autoregressively in this discrete space. At the end of the pipeline, the outcome evaluator assigns success scores and generates human-readable justifications for imagined trajectories.

We also identified several challenges. The first challenge is collecting datasets. We explored different ways to generate the ground-truth dataset, including querying GPT-4 and constructing scene descriptions from metadata. We found that LLMs like GPT-4 often hallucinate objects and lack spatial precision, while template-based descriptions reconstructed from metadata were perfectly accurate but too verbose and structurally uniform that the model learned to mimic formatting rather than semantic content. By distilling each scene

into a short seed caption that faithfully captures only the essential spatial relationships, we collect a dataset that is both precise and concise.

Another main challenge is that training and fine-tuning the VLMs is computationally expensive and time-consuming. We discussed several optimizations to make training VLMs more efficient, including caching of preprocessed image tensors and tokenized captions, increasing dataloader parallelism, and staging data to local SSDs dramatically reduced CPU-GPU stalls.

For future work, we plan to integrate these components into a complete system that can perform complex tasks in real-world environments. We will also explore the use of reinforcement learning to improve the performance of the dynamics models and the outcome evaluator. By combining these techniques, we aim to create a robust and interpretable planning framework that can be used in a wide range of applications, from robotics to autonomous vehicles.

Appendix A

Prompt Evaluation

A.1 Scene Description From Metadata

Example Scene Description Generated from Metadata

Scene with 10 visible objects.

- 5 objects in front of the agent: BaseballBat, FloorLamp, Mug, TVStand, Vase.
- 4 objects to the left of the agent: BasketBall, SideTable, Sofa, Window.
- 1 objects to the right of the agent: Floor.

Relationships with agent:

- Floor is to the right of and below the agent, close to the agent.
- Mug is in front of and at same height as the agent, close to the agent.
- Sofa is to the left of and below the agent, at moderate distance from the agent.
- TVStand is in front of and below the agent, at moderate distance from the agent.
- BaseballBat is in front of and at same height as the agent, at moderate distance from the agent.
- BasketBall is to the left of and below the agent, at moderate distance from the agent.
- Vase is in front of and at same height as the agent, at moderate distance from the agent.
- FloorLamp is in front of and below the agent, far from the agent.
- SideTable is to the left of and below the agent, far from the agent.
- Window is to the left of and at same height as the agent, far from the agent.

(CONTINUED) Example Scene Description Generated from Metadata

Scene with 10 visible objects. Relationships between objects (from agent's perspective):

- Floor contains FloorLamp.
- Floor contains SideTable.
- Floor contains Sofa.
- Floor contains TVStand.
- Sofa contains BasketBall.
- TVStand contains BaseballBat.
- TVStand contains Mug.
- TVStand contains Vase.
- Mug: BaseballBat is behind and at same height as it; BasketBall is to the left of it; Sofa is to the left of and below it; Vase is behind and at same height as it.
- Sofa: BaseballBat is behind and above it; Mug is to the right of and above it; TVStand is behind and at same height as it; Vase is behind and above it.
- TVStand: BasketBall is to the left of and above it; Sofa is in front of and at same height as it.
- BaseballBat: BasketBall is to the left of it; Mug is in front of and at same height as it; Sofa is in front of and below it; Vase is to the left of and at same height as it.
- BasketBall: BaseballBat is to the right of it; Mug is to the right of it; TVStand is to the right of and below it; Vase is to the right of it.
- Vase: BaseballBat is to the right of and at same height as it; BasketBall is to the left of it; Mug is in front of and at same height as it; Sofa is in front of and below it.
- FloorLamp: SideTable is to the left of and at same height as it.
- SideTable: FloorLamp is to the right of and at same height as it; Window is to the left of and above it.
- Window: SideTable is to the right of and below it.

Appendix B

Supplemental Material for Model Training and Evaluation

B.1 VLM fine-tuning

We fine-tune the VLMs with LoRA adapters on image-caption pairs. Since the state abstractor, policy model, and the outcome evaluator share the same VLM backbone, we only need to fine-tune the LoRA adapters. For state abstractor, we use InstructBLIP-Flan-T5-XL and Qwen2.5-VL-7B-Instruct. For the policy model and the outcome evaluator, we use Qwen2.5-VL-3B-Instruct.

The hyper-parameters for the fine-tuning InstructBLIP and Qwen2.5-VL including the loss are shown in Table ?? and Table ??, respectively.

hyperparameters	Value
Epochs	100
Learning rate	5×10^{-5}
Batch size	1
Gradient accumulation steps	4
LoRA rank (r)	8
LoRA alpha	16
LoRA dropout	0.05
LoRA target	all-linear
Weight decay	1×10^{-4}
Scheduler	CosineAnnealingLR
Scheduler T_{\max}	500
Scheduler η_{\min}	1×10^{-6}
Max gradient norm	1.0
KL weight	0.2
Optimizer	AdamW
Loss type	Cross-entropy + KL divergence

Table B.1: Hyperparameters for fine-tuning InstructBLIP with LoRA.

hyperparameters	Value
Epochs	100
Learning rate	5×10^{-5}
Batch size	1
Gradient accumulation steps	4
LoRA rank (r)	8
LoRA alpha	16
LoRA dropout	0.05
LoRA target	q_proj, v_proj
Weight decay	1×10^{-4}
Scheduler	CosineAnnealingLR
Scheduler T_{\max}	500
Scheduler η_{\min}	1×10^{-6}
Max gradient norm	1.0
Early stopping patience	10
Optimizer	AdamW
Loss type	Cross-entropy

Table B.2: Hyperparameters for fine-tuning Qwen2.5-VL with LoRA.

B.2 Policy Model Evaluation

B.2.1 Comparison of Predicted and Ground Truth Action Sequences

Example Roll-out for Policy Model

Evaluating ...

checkpoint: final_model.pt

Example Action Sequences:

Example 1:

Predicted: MoveAhead -> MoveAhead -> MoveAhead -> MoveAhead -> MoveAhead
-> MoveAhead -> MoveAhead -> MoveAhead

Ground Truth: RotateLeft -> MoveAhead -> MoveAhead -> MoveAhead ->
MoveAhead -> MoveAhead -> MoveAhead -> MoveAhead

Example 2:

Predicted: MoveAhead -> MoveAhead -> MoveAhead -> MoveAhead -> MoveAhead
-> MoveAhead -> MoveAhead -> MoveAhead

Ground Truth: MoveAhead -> MoveAhead -> MoveAhead -> MoveAhead ->
RotateRight -> MoveAhead -> MoveAhead -> MoveAhead

Example 3:

Predicted: MoveAhead -> MoveAhead -> MoveAhead -> MoveAhead -> MoveAhead
-> MoveAhead -> MoveAhead -> MoveAhead

Ground Truth: MoveAhead -> MoveAhead -> MoveAhead -> MoveAhead ->
MoveAhead -> MoveAhead -> MoveAhead -> RotateLeft

References

- [1] T. B. Brown et al. “Language models are few-shot learners”. In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS ’20. Vancouver, BC, Canada: Curran Associates Inc., 2020. ISBN: 9781713829546.
- [2] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. “Gpt-4 technical report”. *arXiv preprint arXiv:2303.08774* (2023).
- [3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. “Llama: Open and efficient foundation language models”. *arXiv preprint arXiv:2302.13971* (2023).
- [4] A. Q. Jiang et al. *Mistral 7B*. 2023. arXiv: 2310.06825 [cs.CL]. URL: <https://arxiv.org/abs/2310.06825>.
- [5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. “Attention is all you need”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. ISBN: 9781510860964.
- [6] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. “Scaling laws for neural language models”. *arXiv preprint arXiv:2001.08361* (2020).
- [7] A. Radford et al. “Learning Transferable Visual Models From Natural Language Supervision”. In: *International Conference on Machine Learning*. 2021. URL: <https://api.semanticscholar.org/CorpusID:231591445>.
- [8] M. Ahn et al. “Do As I Can and Not As I Say: Grounding Language in Robotic Affordances”. In: *arXiv preprint arXiv:2204.01691*. 2022.
- [9] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. “Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents”. *International Conference on Machine Learning* (2022).
- [10] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su. “LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2023.

- [11] B. Y. Lin, C. Huang, Q. Liu, W. Gu, S. Sommerer, and X. Ren. “On grounded planning for embodied tasks with language models”. In: *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI’23/IAAI’23/EAAI’23. AAAI Press, 2023. ISBN: 978-1-57735-880-0. DOI: [10.1609/aaai.v37i11.26549](https://doi.org/10.1609/aaai.v37i11.26549). URL: <https://doi.org/10.1609/aaai.v37i11.26549>.
- [12] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Suenderhauf. “SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Robot Task Planning”. In: *Proceedings of The 7th Conference on Robot Learning*. Ed. by J. Tan, M. Toussaint, and K. Darvish. Vol. 229. Proceedings of Machine Learning Research. PMLR, June 2023, pp. 23–72. URL: <https://proceedings.mlr.press/v229/rana23a.html>.
- [13] V. S. Dorbala, J. F. Mullen Jr, and D. Manocha. “Can an Embodied Agent Find Your " Cat-shaped Mug"? LLM-Based Zero-Shot Object Navigation”. *arXiv preprint arXiv:2303.03480* (2023).
- [14] J. Yang, X. Chen, S. Qian, N. Madaan, M. Iyengar, D. F. Fouhey, and J. Chai. “LLM-Grounder: Open-Vocabulary 3D Visual Grounding with Large Language Model as an Agent”. *2024 IEEE International Conference on Robotics and Automation (ICRA)* (2023), pp. 7694–7701. URL: <https://api.semanticscholar.org/CorpusID:262084072>.
- [15] L. Guan, K. Valmeekam, S. Sreedharan, and S. Kambhampati. “Leveraging pre-trained large language models to construct and utilize world models for model-based task planning”. In: *Proceedings of the 37th International Conference on Neural Information Processing Systems*. NIPS ’23. New Orleans, LA, USA: Curran Associates Inc., 2023.
- [16] Z. Yang, C. Garrett, D. Fox, T. Lozano-Pérez, and L. P. Kaelbling. *Guiding Long-Horizon Task and Motion Planning with Vision Language Models*. 2024. arXiv: [2410.02193](https://arxiv.org/abs/2410.02193) [cs.R0]. URL: <https://arxiv.org/abs/2410.02193>.
- [17] H. Zhang, Z. Wang, Q. Lyu, Z. Zhang, S. Chen, T. Shu, B. Dariush, K. Lee, Y. Du, and C. Gan. “COMBO: Compositional World Models for Embodied Multi-Agent Cooperation”. In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: <https://openreview.net/forum?id=YXRyYkblim>.
- [18] W. Chen, O. Mees, A. Kumar, and S. Levine. “Vision-Language Models Provide Promptable Representations for Reinforcement Learning”. *arXiv preprint arXiv:2402.02651* (2024).
- [19] X. Li et al. “Vision-Language Foundation Models as Effective Robot Imitators”. In: *The Twelfth International Conference on Learning Representations*. 2024. URL: <https://openreview.net/forum?id=IFYj0oibGR>.
- [20] A. Eftekhari, K.-H. Zeng, J. Duan, A. Farhadi, A. Kembhavi, and R. Krishna. “Selective Visual Representations Improve Convergence and Generalization for Embodied AI”. In: *ICLR*. 2024.
- [21] M. Deitke, E. VanderBilt, A. Herrasti, L. Weihs, K. Ehsani, J. Salvador, W. Han, E. Kolve, A. Kembhavi, and R. Mottaghi. “ProcTHOR: Large-Scale Embodied AI Using Procedural Generation”. In: *NeurIPS*. 2022.

- [22] L. Weihs, J. Salvador, K. Kotar, U. Jain, K.-H. Zeng, R. Mottaghi, and A. Kembhavi. “AllenAct: A Framework for Embodied AI Research”. *arXiv preprint arXiv:2008.12760* (2020).
- [23] J. Li, D. Li, S. Savarese, and S. Hoi. “BLIP-2: bootstrapping language-image pre-training with frozen image encoders and large language models”. In: *Proceedings of the 40th International Conference on Machine Learning*. ICML’23. Honolulu, Hawaii, USA: JMLR.org, 2023.
- [24] J. Bai, S. Bai, S. Yang, S. Wang, S. Tan, P. Wang, J. Lin, C. Zhou, and J. Zhou. “Qwen-VL: A Versatile Vision-Language Model for Understanding, Localization, Text Reading, and Beyond”. *arXiv preprint arXiv:2308.12966* (2023).
- [25] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. “LoRA: Low-Rank Adaptation of Large Language Models”. In: *International Conference on Learning Representations*. 2022. URL: <https://openreview.net/forum?id=nZeVKeeFYf9>.
- [26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. “Proximal Policy Optimization Algorithms”. *CoRR* abs/1707.06347 (2017). arXiv: [1707.06347](https://arxiv.org/abs/1707.06347). URL: <http://arxiv.org/abs/1707.06347>.
- [27] P. Sunehag et al. “Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward”. In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS ’18. Stockholm, Sweden: International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 2085–2087.