Building Situated Agents

Doctoral Dissertation

Machine Learning Department
CARNEGIE MELLON UNIVERSITY
Pittsburgh, Pennsylvania, 15213

June 2025
CMU-ML-25-106

By

So Yeon Min

Thesis committee

Submitted in partial fulfillment of the requirements of
for the degree of Doctor of Philosophy (PhD) in Machine Learning

This research was funded by: Defense Advanced Research Projects Agency award HR00112490375; Office of Naval Research awards N000141812861 and N000142312368; grants from Amazon, Apple and Singapore's Defence Science and Technology Agency; and a graduate fellowship from Apple.

Keywords: Embodied AI, AI Agents, Situated Cognition, Metacognitive Intelligence

Acknowledgements

Completing this thesis was a remarkable journey made possible through the unwavering support, guidance, and friendship of numerous people, and I am profoundly grateful to each of them.

First and foremost, I extend my deepest gratitude to my advisors, Professors Ruslan Salakhutdinov and Yonatan Bisk. The day Ruslan accepted me as his student remains vividly joyful in my memory; his mentorship has shaped me profoundly, both intellectually and personally, guiding me with fatherly wisdom and deep kindness. Yonatan fundamentally transformed my understanding of research, teaching me to rigorously approach unsolved problems and refine my perspectives. His philosophical insights and thoughtful mentorship have greatly enriched my academic journey. Together, they have supported me through every challenge. Throughout my PhD journey, they have been like second parents to me, and I will always hold them in the highest regard.

I would like to express my sincere appreciation to my thesis committee members, Professor Daniel Fried, Professor Katerina Fragkiadaki, and Professor Joyce Chai, for their invaluable feedback, thoughtful questions, and guidance throughout this process. Their diverse perspectives and expertise significantly strengthened this work.

I am deeply grateful for the support of the Apple AI/ML Fellowship, which provided invaluable resources for this research.

My heartfelt thanks go to the wonderful colleagues and friends I have made in both labs. I am especially thankful to Devendra Chaplot for being my mentor and giving me numerous learnings, Hubert Yaohung Tsai and Jian Zhang for invaluable internship opportunities and support at Apple, and my amazing labmates including Shrimai Prabhumoye, Ben Eysenbach, Ruosong Wang, Kelly He, Brandon Trabucco, Murtaza Dalal, Paul Liang, Minji Yoon, J. Y. Koh, Fahim Tajwar, Hao Zhu, Jared Fernandez, Yingshan Chang, Leena Mathur, Abhitha Thankaraj, Rosa Vitellio, Quanting Xie, Jimin Sun, and Vidhi Jain. Paul's friendship provided emotional grounding during challenging moments. Minji's insightful advice and infectious laughter brightened many days, making even routine lab meetings genuinely enjoyable. J.Y. Koh helped me navigate web agents with unwavering support, while Fahim introduced me to the exciting intricacies of reinforcement learning. Hao's encouragement and help in my embodied dialogue project and emotional solidarity were irreplaceable. Jared infused our lab with delightful humor, Yingshan's candidness always brought refreshing honesty and laughter, and Leena's amusing memes and humble nature kept spirits high. I deeply thank Quanting for collaborating with me on the groundbreaking embodied-RAG project and teaching me multiple aspects of research. I thank Jimin Sun for offering me the opportunity to collaborate on her project. I fondly recall and miss our many spontaneous video calls during the summer of 2022, where we enthusiastically exchanged numerous ideas.

I am grateful to my officemates Jacob Springer, Gaurav Ghosal, Christina Baek, Charvi Rastogi, and Chi-Kuan Yeh, who have consistently provided a welcoming environment and stimulating discussions.

My PhD life was tremendously enriched by close friendships beyond the academic environment. Sanket Mehta's presence marked a joyful turning point in my graduate journey, filling my life with positivity and laughter through our deep mutual understanding. I regret that our friendship blossomed relatively late, despite attending the same group meetings for so long. Emily Byun, my roommate and closest confidante, turned daily campus life into memorable experiences with endless fun and camaraderie. Jimin Sun's charisma and insightfulness brought immense joy and valuable perspectives into my life. Kundan Krishna's genuine warmth, constant readiness to lift my spirits through badminton, and humble generosity have been a continuous source of comfort. Jimin Mun's kindness and warmth provided me strength in difficult times. I thank Hwijeen Ahn for his delicious food, uplifting words, and positivity, all of which have consistently brightened my days and kept my spirits vibrant. Xuhui Zhou's infectious humor and fascinating personality made every moment we spent together truly enjoyable - he is not only a great researcher but also someone whose company brought so much joy, and I deeply regret that we never had the opportunity to collaborate academically. Li-Wei Chen's exceptionally considerate nature and thoughtful personality made our time together genuinely delightful. Daye Nam's remarkable maturity and warmth have greatly inspired and comforted me, while Sang Keun Choe's thoughtful advice and deep conversations have been instrumental in my personal growth. Finally, I thank Jae Yeon (Jane) Pyo, a kindred spirit who provided intellectually stimulating discussions and genuine friendship.

Outside CMU, I am deeply indebted to my dear friend Jihyun Min, whose emotional support, exceptional social insight, and exemplary approach to life have guided and uplifted me profoundly. Hyung Won Chung's intellectual brilliance and visionary advice ignited my passion for research. I am deeply grateful for his invaluable guidance and wish him continued success at the forefront of innovation.

Finally, I express profound gratitude to my family, whose unwavering love and support have been my foundational strength. My father's visionary passion for AI and his encouragement, despite his medical background, deeply motivated my pursuit of this field. My mother's selfless dedication to my education and happiness has been a constant source of strength, a realization that grows deeper with each passing year. Words cannot fully capture the depth of my appreciation for the sacrifices and unconditional love they have continuously provided. My younger brother, despite our limited time together due to my early departure for boarding school and later to the U.S., has shown exceptional maturity and dedication, fulfilling family responsibilities that I could not perform from afar. I deeply admire his commitment as he embarks on his journey through medical school and military service in Korea, and eagerly look forward to cherishing more moments together in the future.

Abstract

In this thesis, we outline how agents can leverage their pretrained knowledge to effectively operate within their specific environments, focusing on perception, cognition, and metacognition. Chapter 1 introduces the topic and establishes the concept of situated agent operation.

Chapters 2 and 3 explore the perceptual capabilities of agents. In Chapter 2, we examine how an agent can utilize common sense to interpret and make sense of incomplete or ambiguous sensory data, enabling intelligent navigation and exploration. Chapter 3 delves into how an agent can apply physical common sense to adapt their perceptual strategies when introduced to new environmental context.

Chapters 4 and 5 assess the cognitive abilities of agents in understanding and executing situated language instructions. Chapter 4 explores embodied dialogue, focusing on how agents built from different training mechanisms process and respond to instructions given in dynamic dialogue settings. Chapter 5 investigates the challenges agents face in following situated instructions, particularly when human intent is ambiguous or incomplete.

Chapter 6 addresses metacognition by developing a framework for training agents to recognize their limitations and request assistance judiciously. We formulate metacognitive help-requesting as a reinforcement learning problem that simultaneously optimizes both the reward function and the help-requesting policy itself.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The rapid advancement of artificial intelligence has produced systems of remarkable capability, yet their deployment in real-world environments continues to reveal fundamental limitations. These limitations are not merely a matter of computational power or model scale, but rather stem from a deeper challenge: the gap between artificial intelligence trained on *static* datasets and the dynamic, contextual demands of *situated* environments. While large language models and computer vision systems demonstrate impressive performance on benchmark tasks, they often struggle when faced with the nuanced, ambiguous, and incomplete information that characterizes real-world scenarios. Effective embodied intelligence requires more than powerful pretrained models—it demands the ability to operate situatedly.

This thesis explores situated intelligence through three interconnected dimensions that emerge when artificial agents must operate in real-world environments. At the perceptual level, agents must construct coherent understanding from fragmentary and sometimes contradictory sensory data. Unlike controlled datasets, real environments present partial observability, sensor noise, and dynamic changes that require agents to leverage common sense and physical consistency to maintain accurate world models. At the cognitive level, agents must interpret human communication that is inherently contextual and often underspecified. Real instructions rely on shared understanding, environmental context, and pragmatic inference that extends far beyond literal semantic parsing. At the metacognitive level, agents must recognize the boundaries of their own knowledge and capabilities, knowing when to act confidently and when to seek assistance.

Each of these dimensions presents distinct technical challenges that this thesis addresses through novel computational approaches. Rather than treating these as separate problems, we demonstrate how they form an integrated framework for situated intelligence, where perceptual robustness enables cognitive understanding, and metacognitive awareness guides appropriate action selection.

Perceptual Challenges Chapters 2 and 3 address perceptual challenges in situated environments. Chapter 2 introduces a modular approach to embodied instruction following, demonstrating how pretrained components can be integrated with semantic search policies that leverage common sense to navigate incomplete sensory data. This work reveals that explicit spatial memory coupled with semantic reasoning can significantly outperform end-to-end approaches, particularly in unseen environments. Chapter 3 extends this foundation by developing self-supervised methods for adapting visual perception to new environments. Through location consistency as a source of self-supervision, we show how agents can adapt both their visual understanding and navigation policies without requiring expensive labeled data, enabling practical deployment in novel environments.

Situated Instructions Chapters 4 and 5 examine the cognitive challenges of interpreting and executing situated language instructions. Chapter 4 analyzes embodied dialogue through the lens of training methodology, comparing modular approaches with end-to-end behavior cloning. Our analysis reveals that agents trained via behavior cloning are particularly vulnerable to replicating errors and struggling with pragmatic grounding, while both approaches face fundamental challenges in handling the inherent ambiguity of natural dialogue. Chapter 5 introduces the Situated Instruction Following (SIF) benchmark to systematically evaluate agents' ability to handle ambiguous, temporally evolving, and dynamic human intent. Through comprehensive evaluation of state-of-the-art models, including GPT-4, we demonstrate that current approaches excel at common-sense reasoning but struggle significantly when faced with uncertainty that requires active disambiguation.

Requesting Interventions Chapter 6 develops a framework for metacognitive awareness in AI agents. We formulate the problem of knowing when to request help as a reinforcement learning challenge that simultaneously optimizes both the reward structure and the help-requesting policy. Our key insight is to predict expected intervention usage before training the policy, enabling efficient identification of optimal configurations without repeated retraining. By integrating LLM-based scoring with tabular reinforcement learning, we achieve systems that match the performance of always-intervening approaches while using only a fraction of the interventions, demonstrating principled approaches to confidence calibration and help-seeking behavior.

Together, these contributions establish a comprehensive framework for situated intelligence that moves beyond the limitations of purely data-driven approaches. By addressing perception, cognition, and metacognition as interconnected challenges, this thesis provides both theoretical insights and practical solutions for developing AI systems that can operate effectively in the complex, dynamic environments where human-AI collaboration ultimately takes place.

Chapter 2

FILM: Following Instructions in Language with Modular Methods

In this chapter, we introduce FILM [98], a modular agent that operates in the ALFRED framework [133]. I discuss the integration of pretrained components into a fully embodied agent and how it utilizes common sense to navigate environments with incomplete or ambiguous sensory data. The commensense is implemented by a learned semantic search policy module that augments the agent's imperfect perception.

2.1 Introduction

Human intelligence simultaneously processes data of multiple modalities, including but not limited to natural language and egocentric vision, in an embodied environment. Powered by the success of machine learning models in individual modalities [**bert**, 58, 149, 4], there has been growing interest to build multimodal embodied agents that perform complex tasks. An incipient pursuit of such interest was to solve the task of Vision Language Navigation (VLN), for which the agent is required to navigate to the goal area given a language instruction [6, 46, 197].

Embodied instruction following (EIF) presents a more complex and human-like setting than VLN or Object Goal Navigation [57, 23, 41]; beyond just navigation, agents are required to execute sequences of sub-tasks that entail both navigation and interaction actions from a language instruction (Fig. 2.1). The additional challenges posed by EIF are threefold - the agent has to understand compositional instructions of multiple types and subtasks, choose actions from a large action space and execute them for longer horizons, and localize objects in a fine-grained manner for interaction [105].

Most existing methods [190, 70, 106] for EIF have relied on neural memory of various types (transformer embeddings, LSTM state), trained end-to-end with expert trajectories upon raw or pre-processed language/visual inputs. However, EIF remains a very challenging task for end-to-end methods as they require the neural net to simultaneously learn state-tracking, building spatial memory, exploration, long-term planning, and low-level control.

In this work, we propose **FILM** (Following Instructions in Language with Modular methods). FILM consists of several modular components that each (1) processes language instructions into structured forms (*Language Processing*), (2) converts egocentric visual input into a semantic metric map (*Semantic Mapping*), (3) predicts a search goal location (*Semantic Search Policy*), and (4) outputs subsequent navigation/ interaction actions (*Deterministic Policy*). FILM overcomes some of the shortcomings of previous methods by leveraging a modular design with structured spatial components. Unlike many of the existing methods for EIF, FILM does ***not*** require any input that provides sequential guidance, namely expert trajectories or low-level language instructions. While [11] recently introduced a method that uses a structured spatial memory, it comes with some limitations from the lack of explicit semantic search and the reliance on expert trajectories.

On the ALFRED [133] benchmark, FILM achieves State-of-the-Art performance (24.46%) with a large margin (8% absolute) from the previous SOTA [11]. Most approaches rely on low-level instructions, and we too find that including them leads to an additional 2% improvement in success rate (26.49%). FILM's strong performance and our analysis indicate that an explicit structured spatial memory coupled with a semantic search policy can provide better state-tracking and exploration, even in the absence of expert trajectories or low-level instructions.

2.2 Prior Work

A plethora of works have been published on embodied vision and language tasks, such as VLN [6, 46, 197], Embodied Question Answering [35, 54], and topics of multimodal representation learning [154, 10], such as Embodied Language Grounding [111]. For Visual Language Navigation, which is the most comparable to the setting of our work, methods with impressive performances [68, 157, 87] have been proposed since the introduction of R2R [6]. While far from conquering VLN, these methods have shown up to 61% success rate on unseen test environments [68].

For the more challenging task of Embodied Instruction Following (EIF), multiple methods have been proposed with differing levels of modularity in the model structure. As a baseline, [133] has presented a Seq2Seq model with an attention mechanism and a progress

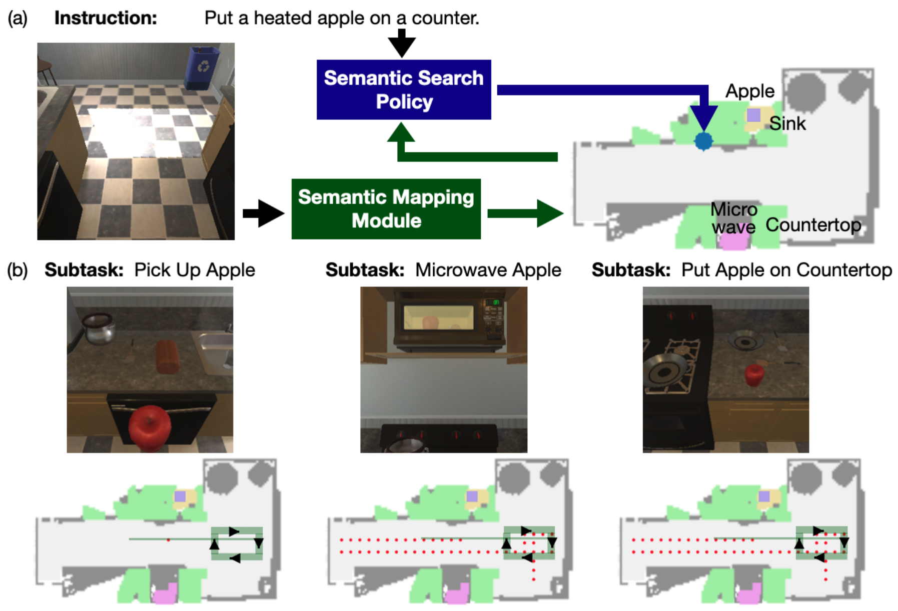

(a) Instruction: Put a heated apple on a counter.
Semantic Search Policy
Semantic Mapping Module
Apple
Sink
Micro wave
Countertop
(b) Subtask: Pick Up Apple
Subtask: Microwave Apple
Subtask: Put Apple on Countertop

Figure 2.1: An Embodied Instruction Following (EIF) task consists of multiple subtasks. (a) **FILM method overview**: The agent receives the language instruction and the egocentric vision of the frame. At every time step, a semantic top-down map of the scene is updated from predicted depth and instance segmentation. Until the subgoal object is observed, a search goal (blue dot) is sampled from the semantic search policy. (b) **Example trajectories**: Trajectory of an existing model (HiTUT [190]) is plotted in a straight green line, and that of FILM is in dotted red. While HiTUT's agent travels repeatedly over a path of closed loop (thick green line, arrow pointing in the direction of travel), FILM's semantic search allows better exploration and the agent sufficiently explores the environment and completes all subtasks.

monitor, while [110] proposed to replace to seq2seq model with an episodic transformer. These methods take the concatenation of language features, visual features, and past trajectories as input and predict the subsequent action end-to-end. On the other hand, [70, 190, 105] modularly process raw language and visual inputs into structured forms, while keeping a separate "action prediction module" that outputs low-level actions given processed language outputs. Their "action taking module" itself is trained end-to-end and relies on neural memory that "implicitly" tracks all of spatial, progressive, and states of the agent. Unlike these methods, FILM's structured language/ spatial representations make reasons for failure transparent and elucidates directions to improve individual components.

Recently, [11] has proposed a more modular method with a persistent and structured spatial memory. Language and visual input are transformed into respectively high-level actions and the 3D map. With the 3D map and high-level actions as input, low-level actions are predicted with a value-iteration network (VIN). Navigation goals for the VIN are sampled from a model trained on interaction pose labels from expert trajectories. Among all proposed methods for EIF, FILM necessitates the least information (neither low-level instructions nor expert trajectories are needed, although the former can be taken as an additional input). Furthermore, FILM addresses the problem of search/ exploration of goal objects.

Various works in visual navigation with semantic mapping are also relevant. Simultaneous Localization and Mapping (SLAM) methods, which build 2D or 3D obstacle maps, have been widely used [47, 65, 138]. In contrast to these works, recent methods [23, 22] build semantic maps with differentiable projection operations, which restrain egocentric prediction errors amplifying in the map. The task of [23, 22] is object goal navigation, a much simpler task compared to EIF. Furthermore, while [23] employs a semantic exploration policy, our and their semantic policies serve fundamentally different purposes; while their policy guides a general sense of direction among multiple rooms in the search for large objects (e.g. fridge), ours guides the search for potential locations of small and flat objects which have little chance of detection at a distance. Also, our semantic policy is conditioned on language instructions. [12, 13] also successfully utilized semantic 2D maps in grounded language navigation tasks. These works are for quadcopters, whose fields of view almost entirely cover the scene and the need for "search" or "exploration" is less crucial than for pedestrian agents. Moreover, their settings only involve navigation with a single subtask.

2.3 Task Expxlantion

We utilize the ALFRED benchmark. The agent has to complete household tasks given only natural language instructions and egocentric vision (Fig. 2.1). For example, the instruction may be given as "Put a heated apple on the counter," with low-level instructions (which FILM does not use by default) further explaining step-by-step lower level actions. In this case, one way to "succeed" in this episode is to sequentially (1) pick up the apple, (2) put the apple in the microwave, (3) toggle the microwave on/off, (4) pick up the apple again, and (4) place it on the countertop. Episodes run for a significantly longer number of steps compared to benchmarks with only single subgoals; even expert trajectories, which are maximally efficient and perform only the strictly necessary actions (without any steps to search for an object), are often longer than 70 steps.

There are seven types of tasks (§A.1), from relatively simple types (e.g. Pick & Place) to more complex ones (e.g. Heat & Place). Furthermore, the instruction may require that an object is "sliced" (e.g. Slice bread, cook it in the microwave, put it on the counter). An episode is deemed "success" if the agent completes all sub-tasks within 10 failed low-level actions and 1000 max steps.

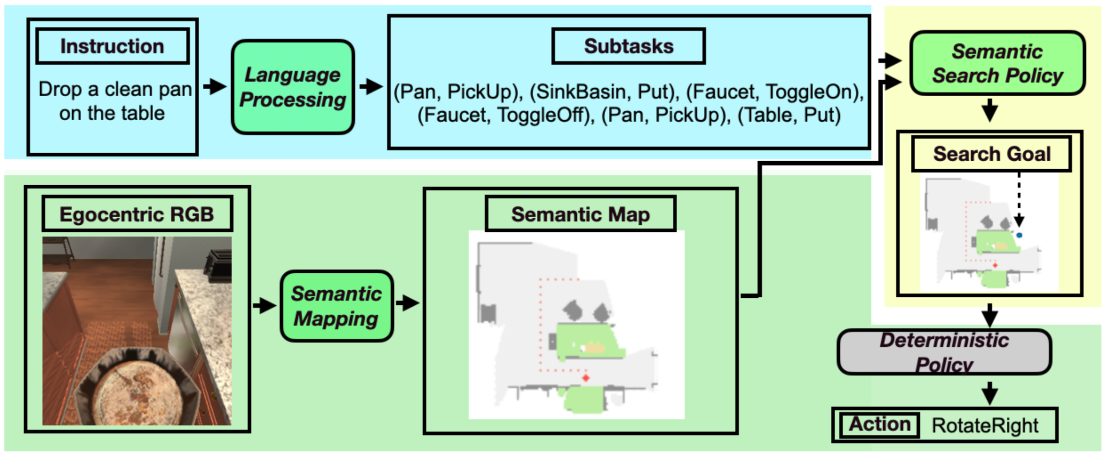


FIGURE 2.2: **FILM method overview**. The "grouping" in blue, green, and yellow denote the coarseness of time scale (blue: at the beginning of the episode, green: at every time step, yellow: at a coarser time scale of every 25 steps). At the beginning of the episode, the Language Processing module processes the instruction into subtasks. At every time step, Semantic Mapping converts egocentric into RGB a top-down semantic map. The semantic search policy outputs the search goal at a coarse time scale. Finally, the Deterministic Policy decides the next action. Modules in bright green are learned; the deterministic policy (grey) is not.

2.4 Methods

FILM consists of three learned modules: (1) Language Processing (LP), (2) Semantic Mapping, and (3) Semantic Search Policy; and one purely deterministic navigation/ interaction policy module (Fig. 2.2). At the start of an episode, the LP module processes the language instruction into a sequence of subtasks. Every time step, the semantic mapping module receives the egocentric RGB frame and updates the semantic map. If the goal object of the current subtask is not yet observed, the semantic search policy predicts a "search goal" at a coarse time scale; until the next search goal is predicted, the agent navigates to the current search goal with the deterministic policy. If the goal is observed, the deterministic policy decides low-level controls for interaction actions (e.g. "Pick Up" object).

2.4.1 Language Processing (LP)

The language processing (LP) module transforms high-level instructions into a structured sequence of subtasks (Fig. 2.3). It consists of two BERT [37] submodules that receive the instruction as an input at the beginning of the episode. The first submodule (*BERT type classification*) receives the instruction and predicts the "type" of the instruction - one of the seven types stated in §A.1. The second submodule (*BERT argument classification*) receives both the instruction and the predicted type as input and predicts the "arguments" - (1) *"obj"* for the object to be picked up, (2) *"recep"* for the receptacle where *"obj"* should be ultimately placed, (3) *"sliced"* for whether *"obj"* should be sliced, and (4) *"parent"* for tasks with intermediate movable receptacles (e.g. "cup" in "Put a knife in a cup on the table" of §A.1). An object in ALFRED is always an instance of either *"obj"* or *"recep"*; *"parent"* objects are a subset of *"recep"* objects that are movable. We train a separate BERT model for each argument predictor. The two submodules are easily trainable with supervised learning since the type and the four arguments are provided in the training set. Models use only the CLS token for classification, and they do not share parameters; all layers of "bert-base-uncased" were fine-tuned.

Due to the patterned nature of instructions, we can match the predicted "type" of the instruction to a "type template" with blank arguments. For example, if the instruction is classified as the "clean & place" type, it is matched to the template "(*Obj*, PickUp), (SinkBasin, Put), (Faucet, ToggleOn), (Faucet, ToggleOff), (*Obj*, PickUp), (*Recep*, Put)". If the "sliced" argument is predicted to be true from argument classification, subtasks of "(Knife, PickUp), (*Obj*, Slice), (Sink, PutObject)" will be added at the beginning of the template (with the (Sink, PutObject) to make the agent drop the knife). Filling in the "type template" with predictions of the second model, we obtain a list of subtasks (bottom of Fig. 2.3b) to be completed in the current episode. The "type templates" were designed by hand in less than 20 minutes. In §2.5.2, we discuss the effect of using a LP module without the template assumption, for fair comparison with other works. §A.9 contains more details.

2.4.2 Semantic Mapping Module

We designed the semantic mapping module (§A.2) with inspirations from prior work [23]. Egocentric RGB is first processed into depth map and instance segmentation, with MaskRCNN [59] (and its implementation by [134]) and the depth prediction method of [11]; details of the training are explained in §2.5 [1] . These pre-trained, off-the-shelf models were finetuned on the training scenes of ALFRED. Once processed, the depth observation is transformed to a point cloud, of which each point is associated with the predicted semantic categories. Finally, the point cloud is binned into a voxel representation; this summed over height is the semantic map. The map is locally updated and aggregated over time.

The resulting semantic map is an allocentric $(C + 2) \times M \times M$ binary grid, where C is the number of object categories and each of the $M \times M$ cells represents a 5cm $\times$ 5cm space of the scene. The C channels each represent whether a particular object of interest was observed; the two extra channels denote whether obstacle exists and whether exploration happened in a particular 5cm $\times$ 5cm space. Thus, the $C + 2$ channels are a semantic/spatial summary of the corresponding space. We use $M = 240$ (12 meters in the

[1]We use the publicly released code of [134, 11].

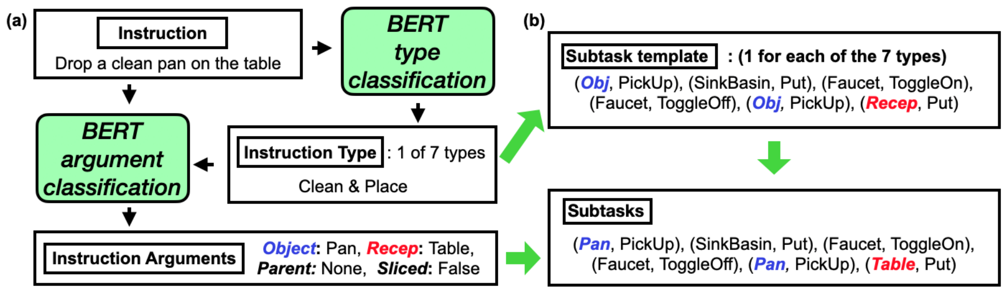

(a)
Instruction
Drop a clean pan on the table
BERT type classification
BERT argument classification
Instruction Type : 1 of 7 types
Clean & Place
Instruction Arguments
Object: Pan, Recep: Table, Parent: None, Sliced: False
(b)
Subtask template : (1 for each of the 7 types)
(Obj, PickUp), (SinkBasin, Put), (Faucet, ToggleOn), (Faucet, ToggleOff), (Obj, PickUp), (Recep, Put)
Subtasks
(Pan, PickUp), (SinkBasin, Put), (Faucet, ToggleOn), (Faucet, ToggleOff), (Pan, PickUp), (Table, Put)

FIGURE 2.3: **The Language Processing module.** (a): Two BERT models respectively predict the "type" and the "arguments" of the instruction. (b): The predicted "type" from (a) is matched with a template, and the "arguments" of the template is filled with the predicted "argument."

physical world) and $C = 28+$(number of additional subgoal objects in the current task). "28" is the number of "receptacle" objects (e.g. "Table", "Bathtub"), which are usually large and easily detected; in the example of Fig. 2.1, there is one additional subgoal object ("Apple"). Please see §A.2 on details of the dynamic handling of C.

2.4.3 Semantic Search Policy

The semantic search policy outputs a coarse 2D distribution for potential locations of a small subgoal object (Fig. 2.4), given a semantic map with the 28 receptacle objects only (e.g. "Countertop", "Shelf"). The discovery of a small object is difficult in ALFRED due to three reasons - (1) many objects are tiny (some instances of "pencil" occupies less than 200 pixels even at a very close view), (2) the field of view is small due to the camera horizon mostly being downward[2], (3) semantic segmentation, despite being fine-tuned, cannot detect small objects at certain angles. The role of the semantic search policy is to predict search locations for small objects, upon the observed spatial configuration of larger ones. While existing works surmise the "implicit" learning of search locations from expert trajectories, we directly learn an explicit policy without expert trajectories.

The policy is trained via supervised learning. For data collection, we deploy the agent without the policy in the training set and gather the (1) semantic map with only receptacle objects and (2) the ground truth location of the subgoal object after every 25 steps. A model of 15 layers of CNN with max-pooling in between (details in §A.3) outputs an $N \times N$ grid, where N is smaller than the original map size M; this is a 2D distribution for the potential location of the subgoal object. Finally, the KL divergence between this and a pseudo-ground truth "coarse" distribution whose mass is uniformly distributed over all cells with the true location of the subgoal object is minimized ($\min_p KL(p||q)$ where p is the coarse ground truth and q is the coarse prediction). At deployment, the

[2]The agent mostly looks down 450.1em in FILM for correct depth prediction. Looking down is common in existing models as well [70, 190, 11].

"search goal" is sampled from the predicted distribution, resized to match the original map size of $M \times M$ (e.g. 240 $\times$ 240), with mass in the coarse $N \times N$ (e.g. 8 $\times$ 8) grid uniformly spread out to the $\lfloor \frac{M}{N} \rfloor \times \lfloor \frac{M}{N} \rfloor$ area centered on it. Because arriving at the search goal requires time, the policy operates at a "coarse" time scale of 25 steps; the agent navigates towards the current search goal until the next goal is sampled or the subgoal object is found (more details in §2.4.4).

Fig. 2.4 shows a visualization of the semantic search policy's outputs. The policy provides a reasonably close estimate of the true distribution; the predicted mass of "bowl" is shared around observed furniture that it can appear on, and that of "faucet" peaks around the sink/ the end of the bathtub. While we chose $N = 8$ as the grid size, §A.4 provides a general bound for choosing N.

2.4.4 Deterministic Policy

Given (1) the predicted subtasks, (2) the most recent semantic map, and (3) the search goal sampled at a coarse time scale, the deterministic policy outputs a navigation or interaction action (Fig. 2.2).

Let $[(obj_1, \, action_1), \, ... \, , (obj_k, \, action_k)]$ be the list of subtasks and the current subtask be $(obj_i, action_i)$. If obj_i is observed in the current semantic map, the closest obj_i is selected as the goal; otherwise, the sample from the semantic search policy is chosen as the goal (§2.4.3). The agent then navigates towards the goal via the Fast Marching Method [129] and performs the required interaction actions. While this "low-level" policy could be learned with imitation or reinforcement learning, we used a deterministic one based on the findings of earlier work that observed that the Fast Marching Method performs as well as a learned local navigation policy [23]. When the agent successfully executes the required interaction $action_i$ (which can be determined by the change in the egocentric RGB), the pointer of subtasks is advanced to $i + 1$ or the task is completed. More details are provided in §A.5.

2.5 Experiments and Results

We explain the metrics, evaluation splits, and baselines against which FILM is compared. Furthermore, we describe training details of each of the learned components of FILM.

Metrics Success Rate (SR) is a binary indicator of whether all subtasks were completed. The goal-condition success (GC) of a model is the ratio of goal-conditions completed at the end of an episode. For example, in the example of Fig. 2.1, there are three goal-conditions - a pan must be "cleaned", a pan should rest on a countertop, and a

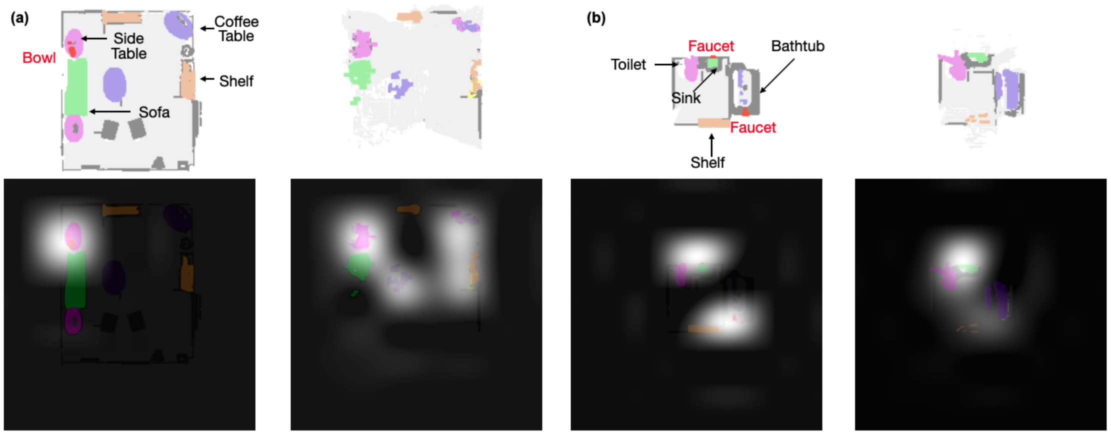


FIGURE 2.4: **Example visualization of semantic search policy outputs.** In each of (a), (b), Top left: map built from ground truth depth/ segmentation, Top right: map from learned depth/ segmentation, Bottom left: ground truth "coarse" distribution, Bottom right: predicted "coarse" distribution. (a): While the true location of the "bowl" was on the upper left coffee table, the policy distributes mass over all furniture likely to have it on. (b): The true location of the faucet is on the sink and at the end of the bathtub. While the policy puts more mass near the sink, it also allocates some to the end of the bathtub.

"clean" pan should rest on a countertop. Both SR and GC can be weighted by (path length of the expert trajectory)/ (path length taken by the agent); these are called path length weighted SR (PLWSR) and path length weighted GC (PLWGC).

Evaluation Splits The test set consists of "Tests Seen" (1533 epsiodes) and "Tests unseen" (1529 episodes); the scenes of the latter entirely consist of rooms that do not appear in the training set, while those of the former only consist of scenes seen during training. Similarly, the validation set is partitioned into "Valid Seen" (820 epsiodes) and "Valid Unseen" (821 epsiodes). The official leaderboard ranks all entries by the SR on Tests Unseen.

Baselines There are two kinds of baselines: those that use low-level sequential instructions [70, 190, 105, 110] and those that do not [106, 11]. While FILM does not necessitate low-level instructions, we report results with and without them and compare them against methods of both kinds.

Training Details of Learned Components In the LP module, BERT type classification and argument classification were trained with AdamW from the `Transformer` [164] package; learning rates are 1e-6 for type classification and {1e-4,1e-5,5e-5,5e-5} for each of "object", "parent", "recep", "sliced" argument classification. In the Semantic Mapping module, separate depth models for camera horizons of 45nd 0ere fine-tuned from an existing model of HLSM [11], both with learning rate 1e-3 and the AdamW optimizer (epsilon 1e-6, weight decay 1e-2). Similarly, separate instance segmentation models for small and large objects were fine-tuned, starting from their respective parameters released by [134], with learning rate 1e-3 and the SGD optimizer (momentum 0.9, weight decay 5e-4). Finally, the semantic search policy was trained with learning rate 5e-4 and the AdamW optimizer (epsilon 1e-6). §A.2 and §A.3 discuss more details on the architectures of semantic mapping/ semantic search policy modules. The readme of our code states protocols and commands so that readers can reproduce all expriments.

2.5.1 Results

Table 2.1 shows test results. FILM achieves state-of-the-art performance across both seen and unseen scenes in the setting where only high-level instructions are given. It achieves 8.17% absolute (50.15% relative) gain in SR on Tests Unseen, and 0.66% absolute (2.63% relative) gain in SR on Tests Seen over HLSM, the previous SOTA.

FILM performs competitively even compared to methods that require low-level step-by-step instructions. They can be used as additional inputs to the LP module, with the low-level instruction appended to the high-level instruction for both BERT type classification and BERT argument classification. Under this setting, FILM achieves 11.06% absolute (71.68% relative) gain in SR on Tests Unseen compared to ABP. Notably, FILM performs similarly across Tests Seen and Tests Unseen, which implies FILM's strong generalizability. This is in contrast to that methods that require low-level instructions, such as ABP, E.T., LWIT, MOCA, perform very well on Tests Seen but much less so on unseen scenes. In a Sim2Real situation, these methods will excel if the agent can be trained in the exact household it will be deployed in, with multiple low-level instructions and expert trajectories. In the more realistic and cost-efficient setting where the agent is trained in a centralized manner and has to generalize to new scenes, FILM will be more adequate.

It is also notable that the semantic search policy significantly increases not only SR and GC, but also their path-length weighted versions. On Tests Seen, the gap of PLWSR between FILM with/ without semantic search is larger than the corresponding gap of SR (for both with/ without low-level instructions). This suggests that the semantic policy boosts the efficiency of trajectories. The results in §A.8 show that the improvement by the semantic policy is reproduced across multiple seeds; the protocols for reproduction are explained along with the result.

2.5.2 Ablations Studies and Error Analysis

Errors due to perception and language processing. To understand the importance of FILM's individual modules, we consider ablations on the base method, the base method with low-level language, and with ground truth visual/ language inputs. Table 2.2 shows

TABLE 2.1: Test results. Top section uses step-by-step instructions; bottom section does not. **Bold** numbers are top scores in each section. Blue numbers are the top SR on Tests Unseen (by which the leaderboard is ranked).

Method		**Tests Seen**				**Tests Unseen**			
		PLWGC	GC	PLWSR	SR	PLWGC	GC	PLWSR	SR
Low-level Sequential Instructions + High-level Goal Instruction									
SEQ2SEQ	[133]	6.27	9.42	2.02	3.98	4.26	7.03	0.08	3.9
MOCA	[137]	22.05	28.29	15.10	22.05	9.99	14.28	2.72	5.30
E.T.	[110]	-	36.47	-	28.77	-	15.01	-	5.04
E.T. + synth. data	[110]	**34.93**	45.44	27.78	38.42	11.46	18.56	4.10	8.57
LWIT	[105]	23.10	40.53	**43.10**	30.92	16.34	20.91	5.60	9.42
HITUT	[190]	17.41	29.97	11.10	21.27	11.51	20.31	5.86	13.87
ABP	[70]	4.92	**51.13**	3.88	**44.55**	2.22	24.76	1.08	15.43
FILM W.O. SEMANTIC SEARCH		13.10	35.59	9.43	25.90	13.37	35.51	10.17	23.94
FILM		15.06	38.51	11.23	27.67	**14.30**	**36.37**	**10.55**	**26.49**
High-level Goal Instruction Only									
LAV	[106]	13.18	23.21	6.31	13.35	10.47	17.27	3.12	6.38
HITUT G-only	[190]	-	21.11	-	13.63	-	17.89	-	11.12
HLSM	[11]	11.53	35.79	6.69	25.11	8.45	27.24	4.34	16.29
FILM W.O. SEMANTIC SEARCH		12.22	34.41	8.65	24.72	12.69	34.00	9.44	22.56
FILM		**14.17**	**36.15**	**10.39**	**25.77**	**13.13**	**34.75**	**9.67**	**24.46**

ablations on the development sets. While the improvement from gt depth is large in unseen scenes (10.64%), it is incremental on seen scenes (1.48%); on the other hand, gt segmentation significantly boosts performances in both cases (9.26% / 9.26%). Thus, among visual perception, segmentation is a bottleneck in both seen/ unseen scenes, and depth is a bottleneck only in the latter. On the other hand, while a large gain in SR comes from using ground truth language (7.43 % / 4.22 %), that from adding low-level language as input is rather incremental. We additionally analyze the effect of the template assumption (explained in the second paragraph of §2.4.1), by reporting the performance with a Language Processing module without this assumption. The results drop without the templates but not by a large margin. §A.9 explains the details of this auxiliary Language Processing module.

Error modes. Table 2.3 shows common error modes of FILM; the metric is the percent of episodes that failed (in SR) from a particular error out of all failed episodes. The main failures in valid unseen scenes are due to failures in (1) locating the subgoal object (due to the small field of view, imperfect segmentation, ineffective exploration), (2) locating the subgoal object because it is in a closed receptacle (cabinet, drawer, etc), (3) interaction (due to object being too far or not in field of view, bad segmentation mask), (4) navigation (collisions), (5) correctly processing language instructions, (6) others, such as the deterministic policy repeating a loop of actions from depth/ segmentation failures and 10 failed actions accruing from a mixture of different errors. A failed episode is classified to the error type that occurs "earlier" (e.g. If the subtasks were processed incorrectly and also there were 10 consecutive collisions, this episode is classified as (5) (failure in incorrectly processsing language instructions) since the LP module comes "earlier" than the collisions). More details are in §A.6. As seen in Table 2.3, *goal object not found* is the most common error mode, typically due to objects being small and not

Table 2.2: Ablation results on validation splits. Base Method is FILM with semantic search policy.

Method	**Val Seen** GC	**Val Seen** SR	**Val Unseen** GC	**Val Unseen** SR
Base Method	37.20	24.63	32.45	20.10
+ low-level language	38.54	25.24	32.89	20.61
+ gt seg.	45.46	34.02	42.88	29.35
+ gt depth	38.21	26.59	42.91	30.73
+ gt depth, gt seg.	55.54	43.22	64.31	55.05
+ gt depth, gt seg., gt lang.	59.47	47.44	69.13	62.48
- template assumption	31.46	20.37	31.14	18.03

Table 2.3: **Error Modes**. Table showing percentage of errors due to each failure mode for FILM on the Val set.

Error mode	**Seen**	**Unseen**
Goal object not found	23.30	26.07
Interaction failures	6.96	8.54
Collisions	6.96	11.00
Object in closed receptacle	18.44	16.16
Language processing error	18.53	24.54
Others	25.81	13.69

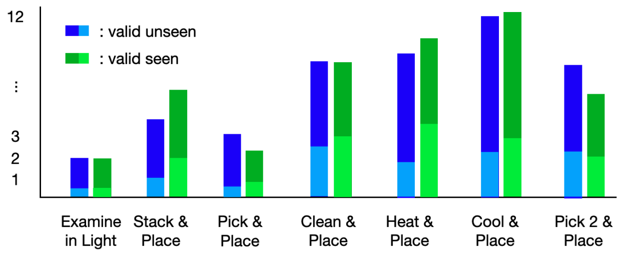


Figure 2.5: Average number of subtasks *completed until failure*, by task type (light green/ light blue respectively for valid seen/ unseen). Dark green/ blue: average number of *total* subtasks in valid seen/ unseen.

Table 2.4: Performance by task type of base model on validation.

Task Type	**Val Seen** GC	**Val Seen** SR	**Val Unseen** GC	**Val Unseen** SR
Overall	37.20	24.63	32.45	20.10
Examine	50.00	34.41	45.06	29.65
Pick & Place	27.46	26.92	16.67	16.03
Stack & Place	23.74	10.71	9.90	1.98
Clean & Place	58.56	44.04	48.89	33.63
Cool & Place	27.04	12.61	27.41	14.04
Heat & Place	40.21	22.02	37.77	23.02
Pick 2 & Place	40.37	23.77	29.28	11.84

visible from a distance or certain viewpoints. Results of the next subsection show that this error is alleviated by the semantic search policy in certain cases.

Performance over different task types. To understand FILM's strengths/ weaknesses across different types of tasks, we further ablate validation results by task type in Table 2.4. Figure 2.5 shows the average number of subtasks completed for failed episodes, by task type. First, the SR and GC for "Stack & Place" is remarkably low. Second, the number of the subtasks entailed with the task type does not strongly correlate with performance. While "Heat & Place" usually involves three more subtasks than "Pick & Place", the metrics for the former are much higher than those of the latter. Since task types inevitably occur in different kinds of scenes (e.g. "Heat & Place" only occurs in kitchens) and therefore involve different kinds of objects (e.g. "Heat & Place" involves food only), the results suggest that the success of the first PickUp action largely depends on the kinds of the scene and size and type of the subgoal objects rather than number of subtasks.

While the above error analysis is specific to FILM, its implications regarding visual perception may generally represent the weaknesses of existing methods for EIF, since most recent methods (ABP, HLSM, HiTUT, LWIT, E.T.) use the same family of segmentation/ detection models as FILM, such as Mask-RCNN and Fast-RCNN [156]. Specifically, it could be that the inability to find a subgoal object is a major failure mode in the mentioned existing methods as well. On the other hand, FILM is not designed to search inside closed receptacles (e.g. cabinets), although subgoal objects dwell in receptacles quite frequently (Table 2.3); a future work to extend FILM should learn to perform a more active search.

2.5.3 Effects of the Semantic Search Policy

With Valid Unseen as the development set, we observed that the semantic search policy significantly helps to find small objects (Table 2.5); we use the percent of episodes in which the first goal object was found (%1st Goal Found) as a proxy, since it can be picked up (e.g. "Apple", "Pen") and thus is usually small. Thus, we use FILM with semantic search as the "base method" (default) for all experiments/ ablations.

Table 2.5: Dev set results (*valid unseen*) of FILM with/ without semantic search policy.

Method	**% 1st Goal Found**	**SR**
HLSM [11]	N/A	11.8
FILM with Search	**80.51**	**20.09**
FILM w.o. Search	76.12	19.85

To further analyze when the semantic search policy especially helps, we ablate on room sizes and task types. Table 2.6 shows the SR and %1st Goal Found with and without search, by room size (details on the assignment of Room Size are in §A.7). As expected, the semantic policy increases both metrics, especially so in large scenes. This is desirable since the policy makes the agent less disoriented in difficult scenarios (large scenes); the model without it is more susceptible to failing even the first subtask. Figure 2.6 is consistent with the trend of Table 2.6; it shows example trajectories of FILM with and without the semantic search policy in a large kitchen scene. Since the countertop appears in the bottom right quadrant of the map, it is desirable that the agent travels there to search for a "knife". While FILM travels to this area frequently (straight red line in Fig. 2.6), FILM without semantic search mostly wanders in irrelevant locations (e.g. the bottom left quadrant).

Table 2.7 further shows the performance with and without search by task type. Notably, the gap of performance for the "clean & place" type is very large. In the large kitchen scene of "Valid Unseen" (Fig. 2.6), the "Sink" looks very flat from a distance and is hardly detected. The semantic policy induces the agent to travel near the countertop area and improves the localization of the 1st Recep ("Sink") for the "clean & place" type (Table 2.7). In conclusion, the semantic policy improves the localization of small and flat objects in large scenes.

Table 2.6: Performance with and without semantic search policy, by room size.

Room Size	**Small**		**Large**	
	FILM	FILM w.o. Search	FILM	FILM w.o. Search
SR	26.70	26.63	15.17	14.74
% 1st Goal Found	79.32	81.02	80.13	73.72

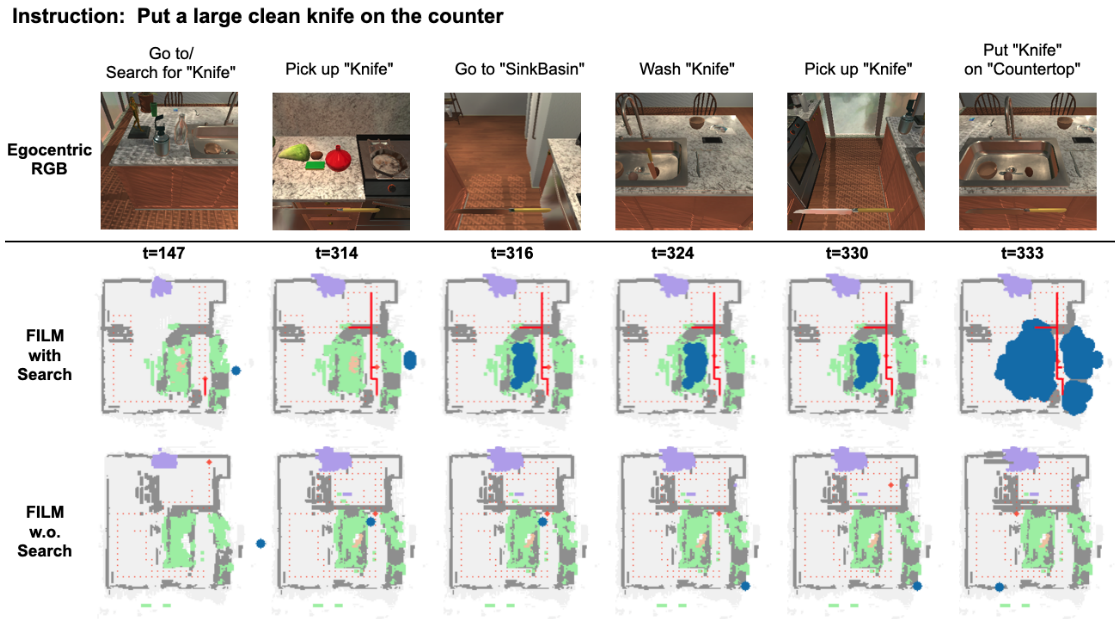


FIGURE 2.6: **Example trajectories of FILM with and without semantic search policy.** Paths near the subgoals that were traveled 3 times or more are in straight red. The goal (which can be the search goal or an observed instance of a subgoal object) is in blue.

TABLE 2.7: Performance with and without semantic search policy, by task type.

Task Type	**Clean & Place**		**Other Types**	
	FILM	FILM w.o. Search	FILM	FILM w.o. Search
SR	33.63	14.16	17.94	20.16
% 1st Goal Found	87.61	79.65	79.38	75.56
% 1st Recep Found	80.53	69.03	58.05	55.93

2.6 Conclusion and Next Steps

We presented FILM, a new modular method for embodied instruction following which (1) processes language instructions into structured forms (*Language Processing*), (2) converts egocentric vision into a semantic metric map (*Semantic Mapping*), (3) predicts a likely goal location (*Semantic Search Policy*), and (4) outputs subsequent navigation/ interaction actions (*Algorithmic Planning*). FILM achieves the state of the art on the ALFRED benchmark without any sequential supervision.

In this chapter, we identify two primary bottlenecks. The first challenge is maintaining consistent visual perception over time and across various viewpoints. Despite implementing a semantic policy that incorporates common sense, achieving consistent visual perception remains a significant hurdle. This consistency is crucial as it forms the foundation for reliable memory and serves as the cornerstone for all subsequent planning activities.

The second bottleneck involves interactions with those located within closed receptacles, which pose unique challenges for situated planning. For instance, if commonsense suggests a salt shaker should be present but it is not visible, the agent may need to deduce that it should search inside closed compartments. This level of inferential reasoning and environmental interaction goes beyond simple common sense and requires sophisticated decision-making capabilities.

Chapter 3

Self-Supervised Object Goal Navigation with In-Situ Finetuning

In the previous chapter, we identified visual perception as a critical bottleneck for embodied agents. In this chapter, we explore the application of physical common sense, particularly through location consistency, to refine an agent's perceptual strategies. This approach not only improves visual perception but also bolsters downstream exploration policies, adapting an agent effectively to new environmental contexts.

3.1 Introduction

In practical deployment scenarios where a robot is delivered to a customer's house, it should quickly generalize and *adapt* to new objects and layouts. For the task of object goal navigation (ObjectNav), this means that the robot should adapt both 1) its visual perception (*which pixel is which object?*) to the objects and 2) the navigation policy (*where to look next?*) to the house layout in the deployment setting. Because this should happen automatically, without the user annotating data, we propose *in-situ* learning (i.e. learning in the deployed *real-world* setting) of ObjectNav agents. Our proposed *in-situ* adaptation is not applicable to existing approaches [23, 119] as they depend on expensive labeled semantic 3D meshes of scanned and reconstructed houses to train their agents and perception stacks.

For this purpose, we introduce a novel and strong source of self-supervision (*Location Consistency - LocCon*), and propose a method to train all components of an ObjectNav agent *without* labeled 3D meshes. At a high level, our approach is composed of two

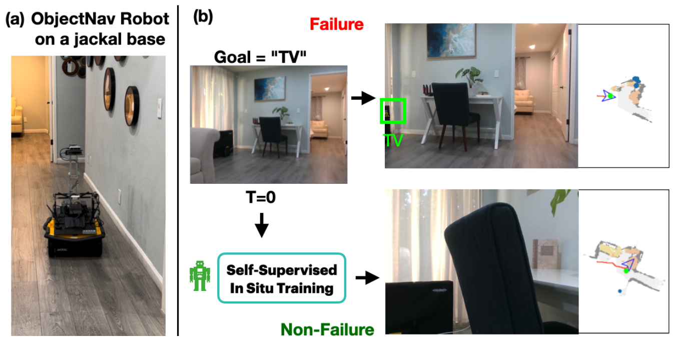

(a) ObjectNav Robot on a jackal base
(b)
Failure
Goal = "TV"
TV
T=0
Self-Supervised In Situ Training
Non-Failure

FIGURE 3.1: **Real World ObjectNav and *in-Situ* Training** (a) Picture of our robot in one of the airbnbs rented for ObjectNav experiments. (b) Example ObjectNav (real-world) failure mode and its remedy with *in-situ* self-supervised (location consistency) training. The agent starts at the living room and wrongly detects a small patch of a black object as "TV." The segmentation model avoids this distractor after *in-situ* location consistency training.

stages: Stage I (visual perception) utilizes the embodiment of an agent to train a semantic segmentation model, and then Stage II (Nav Policy) uses the trained semantic segmentation model to self-label the 3D mesh, which then can be used to train an ObjectNav policy. In particular, in Stage I, we use location consistency as a self-supervision signal: an agent collects images from different views/angles for a given location. We scale the location-consistency collection to 100 unlabeled houses and apply contrastive training to fine-tune the backbone of a pretrained semantic segmentation model [27] (*LocCon* training). In Stage II, we train a navigation policy inspired by PONI [119], which learns potential functions over map frontiers (analytic functions of unexplored area and geodesic distances to goal locations), with self-labeled semantic maps.

Our analysis finds that (1) our fully self-supervised ObjectNav agent can perform competitively in the real world, (2) *in-situ* training with *LocCon* further improves visual perception performance at deployment, and (3) despite the presence of noisy artifacts in simulation, our self-supervised training can learn useful semantics for sim2real transfer. We present a comprehensive set of experiments in real houses and simulation (Gibson ObjectNav datasets [23, 169, 3]).

First, we perform a sim2real experiment that shows our agent trained in simulation can be directly run in the real world for object goal navigation, demonstrating the robustness of our approach. We find that the largest error mode of real-world ObjectNav is visual perception errors, which often *nullifies* the need to adopt a better ObjectNav policy. This analysis, together with evidence from previous work [23] that the Nav Policy, captures abstract knowledge of house layouts which transfers to the real world, shows that *in-situ* training is most needed for updating visual perception. Second, we perform self-supervised location-consistency training directly in the *real houses* – a step towards agents learning on their own. In particular, with location consistency data

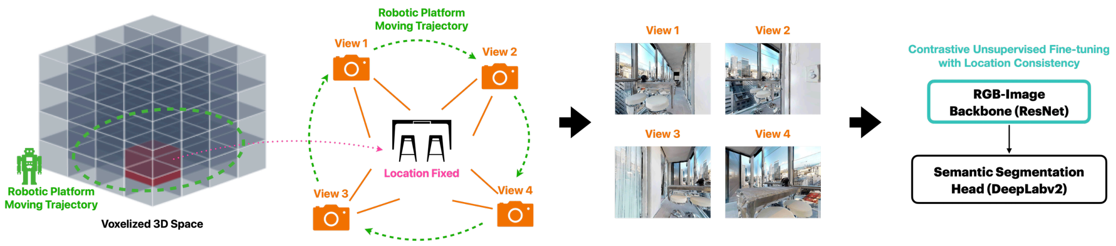

Robotic Platform Moving Trajectory
Voxelized 3D Space
Robotic Platform Moving Trajectory
View 1
View 2
View 3
View 4
Location Fixed
View 1
View 2
View 3
View 4
Contrastive Unsupervised Fine-tuning with Location Consistency
RGB-Image Backbone (ResNet)
Semantic Segmentation Head (DeepLabv2)

FIGURE 3.2: **Fine-tuning visual perception using self-supervision:** First, we produce a 3D voxel grid of the scene from depth sensors of a robot. We randomly select a location and put 50cm^3 cube, and make the robot view it from 8 different angles and 40 different poses. If the ray reflects back from this cube to the robot's camera, we save the corresponding egocentric RGB frame. These collected images are labeled with "location consistency" pseudo-labels in a self-supervised manner. We also show examples of collected "location consistent" images. The ResNet backbone of pre-trained semantic segmentation models can be fine-tuned with our data using contrastive loss.

collected from a *real robot* from 4 AirBnB houses, we show that segmentation models can be improved with our training scheme. Finally, within simulation, we compare our self-supervision pipeline against training with annotations (*FullSup* agent). We find that *FullSup*'s segmentation model learns the visual noise from simulation - which is irrelevant and harmful for real-world transfer. This causes *FullSup* to perform *worse* and makes it a harmful initialization in the real world.

Our first-of-a-kind full stack system for self-supervised ObjectNav enables us to investigate where simulation is most helpful or harmful in training real world ObjectNav agents. The primary approach in the literature has been to focus on collecting scans of houses for reconstruction and training in simulation; however, our results indicate that simulated data often introduces both reconstruction and annotation noise, thereby inhibiting real world transfer. Where supervised learning incentives models to overfit to such artifacts, we find that our self-supervised training is robust to reconstruction errors and help real world transfer. This disparity begs the question - *What data collection and annotation efforts are most effective in the real world and simulation?* We argue, that while simulation holds promise, in the absence of high-quality annotations or renderings, researchers should adopt our self-supervised approach which can be applied directly in the real world for *in-situ* training with raw, unannotated, images that can be easily collected.

3.2 Prior Work

We enumerate related work for visual perception learning and policy learning. We will transition the discussions from supervised learning, which requires a lot of annotated labels, to self-supervised learning, which avoids labels.

TABLE 3.1: **Comparison of self-supervised ObjectNav methods.** Our work (1) employs end-to-end self-supervised training both for the visual perception and navigation policy, and (2) demonstrates the sim2real transfer of both components to the real world.[2]

Method	SSL Visual Perception	SSL Nav Policy	Real Robot Transfer	Real World SSL
SEAL [24]	✓	✗	✗*	✗
OVRL [174]	✓	✗	✗	✗
ZER [3]	✓	✓	✗	✗
CoW, ZSON [49, 90]	✓	✓	✗	✗
Ours	✓	✓	✓	✓

Visual Representation Learning. In ObjectNav, ground truth 3D semantic labels are often given in the dataset and simulators, such as Matterport3D [21], Replica [140], Habitat [125], Gibson [169], iGibson [76], and Habitat-Matterport3D [118]. Existing methods often leverage the ground truth semantic 3D labels as strong supervision signals for ObjectNav [23, 119]. Nonetheless, because of lacking real-world semantic labels, it is unclear if the methods will generalize well from the simulator to the real world.

To avoid the need for ground truth 3D semantic labels, previous work has proposed workarounds: Mimic behaviors from a large collection of human demonstrations [120], gather large-scale image collections from 3D mesh and apply SSL techniques for static images [174], or leverage the embodiment of a moving agent for self-supervised learning by taking the action of the agent and the interaction with the environment into account when building their representations [42, 24, 177].[1] For example, in SEAL [24] an agent roams in an environment, records predictions from an initial semantic segmentation model, propagates the predictions from high-confidence to low-confidence regions, and finally uses the predictions to fine-tune the semantic segmentation model. The resulting model is then used for navigation. In principle, SEAL and LocCon could be used together; however, LocCon is more general as, unlike SEAL, it does ***not*** assume a pre-defined category of objects for training. In CRL [42], the agent learns a policy to search for images that are deemed helpful for contrastive learning. In both SEAL and CRL, self-supervisedly collecting data requires a new training/calibration of policies - which can be a hindrance to real-world transfer; our method, on the other hand, uses algorithmic data collection upon a 3D map, which is simpler and does not require learning. The simplicity and independence of our method allows for easy transfer of our self-supervised learning scheme in the real world (§3.5.1.2).

[0]While [23], a closely related work of [24], shows real world transfer of their semantic mapper and semantic nav policy, [24] itself does not show real world transfer results of its self-supervised components.

[1]Most self-supervised learning approaches leverage only static images without considering the embodiment of an agent. SimCLR[28], MoCo [60], DINO [19], and InfoMax[107] build consistent representations for augmented variants of an same image and maximize their difference to other images.

Policy Learning. In ObjectNav, policies fall into two categories: end-to-end and modular approaches. End-to-end methods [91, 101, 179, 40, 36, 69] directly predict low-level actions (e.g. "move forward", "turn left") from input RGB-D images and the camera pose. Modular methods [**ogn**, 119, 90, 174, 57] consist of a pipeline of components - semantic mapping (persistent semantic and spatial memory), a high-level semantic policy (decides the general direction to move towards the predicted goal location), and low-level navigation modules (a low-level planner for navigating to the high-level goal chosen by the semantic policy). Learning is typically constrained to 1) the semantic (instance) segmentation model inside the semantic mapper and 2) the high-level semantic policy; our method is modular.

Existing methods for semantic policy training often use the object goal's ground truth location, either to define reward functions for reinforcement learning [**ogn**, 174] or potential functions for supervised learning [119] - these are expensive requirements that make these approaches difficult to train in the real world. To avoid the need for ground truth object goal locations, ZER, ZSON, and CoW [3, 90, 49] perform *image* goal navigation (navigating to a target image rather than a category). These images are converted to goal embeddings via CLIP [116] or ResNet[58]-based encoders, allowing for object categories to be mapped to scenes in service of performing ObjectNav. In lieu of web-scale pretraining, given an object goal our method learns to find the object in our self-supervised semantic 3D map without the use of any ground truth location information.

Other Related Work. While not related to self-supervision, [36] uses data at scale to improve performances in ObjectNav. More specifically, it trains EmbCLIP [69] on 10K diverse houses procedurally generated. While we use more homogenous scenes from Gibson [169] and HM3D [118], the implication of our work well agrees with [36] in that scaling brings gains. [83] introduced a framework for lifelong learning for navigation, to prevent policy forgetting after adaptation; our work addresses test-time self-adaptation of both visual perception and policy. [52], a concurrent work, presents large-scale comparisons of ObjectNav methods when transferred to the real world. However, the focus of [52] and our work are different in that we focus on the need of/method for self-supervised real-world *in-situ* training.

3.3 Task Explanation

An ObjectNav [5, 126] agent aims to navigate to an instance of the given object category such as "table" or "sofa" as efficiently as possible, with its perception of the world and semantic priors. The agent is placed at a random location in the environment and receives the goal object category (e.g. "table"). With each action (a_t) the agent takes, it records visual observations (V_t) and pose information (x_t).

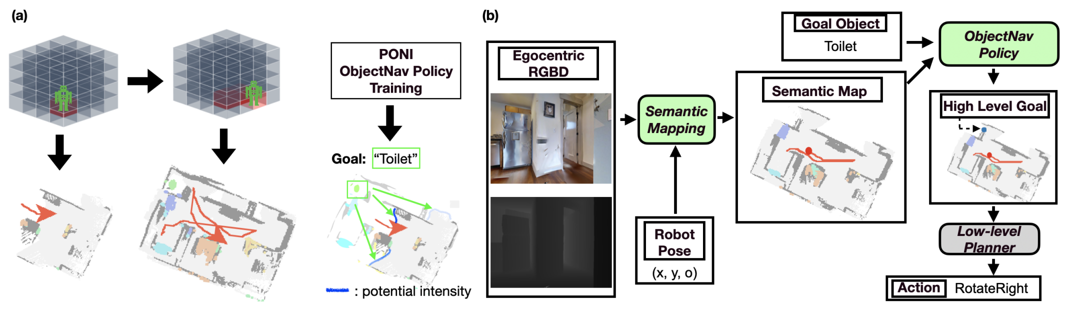


FIGURE 3.3: **ObjectNav policy training and pipeline.** (a) Self-supervised ObjectNav policy training: The agent constructs self-labeled semantic maps of the environment using the segmentation model trained in Stage I. Then, we create a partial map out of full maps, and compute potential functions upon the partial map from the unexplored areas and locations of goal objects. Finally, we apply PONI policy training with the self-labeled potential functions. (b) ObjectNav agent pipeline: Following previous work [**ogn**, 119, 98], we equip our ObjectNav agent with the semantic mapper and the Object Nav policy. The learned components are colored in green.

In this work, we perform ObjectNav both on real robots and within simulation. This leads to several key changes in the action space, and sensor noise in both visual observations and pose information. In both conditions, visual observations will consist of first-person RGB-D images.

Simulation Within simulation, we have an idealized action space A that contains four actions: `forward` (0.25 meters), `left` (30°), `right` (30°), and `stop`. The agent takes the `stop` action when it believes it has reached the goal. The episode is considered successful, if the agent's final location is within a distance $1m$ of the goal. Additionally, we set a maximum number of timesteps (500).

Robot Platform For the real-world Object Nav, we rented three AirBnB's in North America, using the same set of object categories used in simulation. Using the fast-marching method, the robot plans collision-free paths connecting its position to the desired goal location; then, using the `ROS move_base` package, it regulates velocities and yaw rates to follow the planned paths. Since 0.25 meters and 30 degrees are too small for velocity/yaw rate control, we directly use (x, y, o) (where x, y are the "forward" and "left" positions from the starting pose in meters and o is the orientation in radian) as the action space. After each time step, we pass the next desired global pose to `move_base`. To account for velocity control and the body radius, the task is considered successful if the robot stops within 1.5 meters of the desired goal. Since the action space is no longer discrete, we set a maximum duration of time (7 minutes) instead of timesteps. Furthermore, to account for localization errors, we allow up to two trials per task. Additional robot hardware specifications are in §3.5.1.1.

While we explain the details of a static ObjectNav task above, we further note that we also present results for in-situ test-time adaptation (§3.4.1).

3.4 Methods

Our agent is composed of a semantic mapper and a navigation policy (Fig. 3.3 (b)), following previous work [**ogn**, 119]. Mapping converts egocentric RGB-D and robot pose information to a 5cm^3 voxel representation of the scene, with labels for occupancy, explored areas, and semantic categories for each voxel [**ogn**]. Core to mapping, is learning semantic (instance) segmentation, which enables object category labeling. The navigation policy (Fig. 3.3 (a)) is responsible for spatial understanding and deciding where to look/move next.

3.4.1 Location Consistency for Visual Perception

The self-supervised finetuning of the semantic segmentation model is composed of two steps (Fig. 3.2). First, we algorithmically program a robot to navigate around objects to link images with location consistency. Second, we train the model in an off-line manner, using a contrastive loss.

Location consistency data collection In a given scene, we initiate an agent and create a 3D occupancy map (O) of the environment with frontier-based exploration [175]; the robot continues to move to unexplored areas of the map until there is no unexplored area left. Once complete, we use O to sample the (next) location that the agent will observe from multiple views (Fig. 3.2). First, we sample a point (x, y) among the occupied regions (O summed across height). Second, from $O[x, y]$, we sample z (height) among occupied voxels. Third, we place a 50 cm^3 cube C centered around (x, y, z) and have the agent move around this cube at increment angles of 45^o (i.e. 0, 45, 90, ...) and distances of $0.3, 0.6, 0.9, 1.2$ meters from (x, y, z). "Intermediate stop" poses (where the agent stops temporarily to stare at C) are placed in navigable regions of O. When the agent pauses at an "intermediate stop", it ray casts and if it hits C the current egocentric RGB view R is recorded as an image of C. When the agent has visited all "intermediate stops" for the current C, it saves all the views as a "location-consistent" datum. The agent then repeats the entire process by sampling and collecting data from a new location (x, y) from O until 70% of the map is covered or the agent has moved 7,000 steps.

We conduct the same data collection process in multiple real houses to obtain a collection of images. We only consider images from the same cube C as location-consistent. In the physical environment, there are additional complications as agents cannot achieve perfect angles and distances, but the procedure still succeeds and is not adversely affected by these minor perceptual changes. We enumerate factors that could influence the data collection result, such as camera extrinsics, architectural choices, and robot morphology; we hope future work explores these questions.

Contrastive training (LocCon) As with previous work, we initialize our backbone with a pre-trained semantic segmentation model, but training then commences using the aforementioned location consistency data and contrastive training. The location serves as the label tieing together images from disparate views (Fig. 3.2).

We denote a model under training as M (backbone and segmentation head), and its backbone as M_b. We denote I as an image, $\{I^k_{\text{pos},i}\}^n_{i=1}$ as n images collected from the same k-th cube C_k or augmented variants of these images, $\{I^k_{\text{neg},j}\}^m_{j=1}$ as m images not collected from C_k, $\text{sim}(u, v) = M_b(u)^\top M_b(v)/\|M_b(u)\|\|M_b(v)\|$ as the cosine similarity between two embedding vectors $M_b(u)$ and $M_b(v)$, and τ as a temperature hyper-parameter. We consider the popular contrastive loss as in prior work [60, 28]:

$$\ell_B = \sum_{k=1}^{K}\sum_{i=1}^{n} -\log \frac{e^{\text{sim}(I, I_{\text{pos},i})/\tau}}{e^{\text{sim}(I, I_{\text{pos},i})/\tau} + \sum_{j=1}^{m} e^{\text{sim}(I, I_{\text{neg},j})/\tau}}. \tag{3.1}$$

To synchronize the segmentation head with the pretrained backbone, we experimented with the addition of a regularizer, but we found no effect on downstream performance. For each minibatch, the loss is therefore simply ℓ_B. We select 64 different locations in an environment, and for each location we sample four images from different views and apply image augmentations to obtain two augmented variants of each of the four images. This construction, with a mini-batch of 512 images, results in each image having seven positively-paired images ($n = 7$) and 504 negatively-paired image ($m = 504$). We are interested in the semantic segmentation of 15 semantic categories Table 3.7), following previous work [**ogn**, 119]. Since the pre-trained model comes with a segmentation head for more than 15 classes, we treat the rest as "background." Adam optimizer with weight decay $1e-4$ and the learning rate $\in \{1e-6, 1e-5, 1e-4\}$ that outputs the best validation accuracy ($1e-5$) was used.

3.4.2 A Navigation Policy from Self-Labeled Scenes

The ObjectNav policy determines the next intermediate stop to find the goal, based on the partial observations of the agent (Fig. 3.3 (a)). Unlike in prior work [119], we demonstrate how training can be done without a labeled navigation mesh.

We follow a process similar to the pre-mapping of §3.4.1, but now also record semantic information. We equip an agent with the semantic segmentation model trained in §3.4.1,

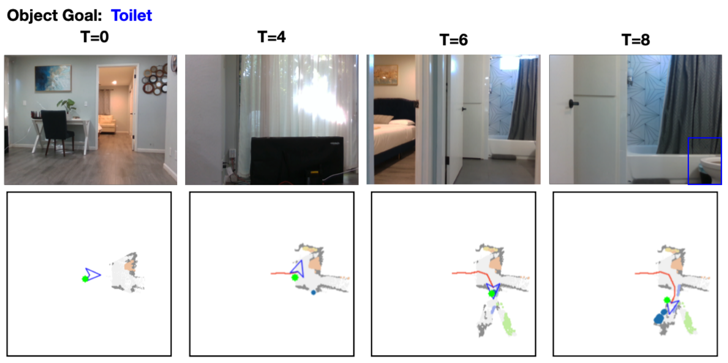


FIGURE 3.4: **Example ObjectNav task in a real house.** The goal object is toilet. Visual perception, semantic mapping, and the ObjecNav policy were all accurate enough to lead to a task success.

and enforce frontier-based exploration; this creates a semantic map of the scenes, self-labeled by the visual perception model. Then, we learn the "potential functions" over the frontiers of partial maps; a high potential function at a point in the map indicates that the point is worth visiting. The total potential function is a sum of the "area potential function" (high intensity means that the point is where the agent should "explore" to gain a more complete map) and the "object potential function" (high intensity means that the point is likely to be close to the object being looked for). The area and object potentials can be directly calculated from the geodesic distances obtained from the *"full map"*, and the policy network learns the potentials given a *"partial map,"* similar to one that the agent will create during a task.

3.5 Experiments and Results

3.5.1 Real World

In §3.5.1.1, we first show that our agent, trained with self-supervision only, can perform ObjectNav competitively in the real world and in simulation; in §3.5.1.2, we show that self-supervised training of visual perception results in good real world transfer, and real-world *in-situ* training can further remedy ObjectNav error modes.

The structure of our ObjectNav agent is shown in Fig. 3.3 (b). The agent relies on our self-supervised segmentation model (*Self-Seg*) and ObjectNav Policy (*Self-Pol*). *Self-Seg* is trained by taking an "off-the-shelf" model (aka deeplabv2[27] pre-trained with COCO-things 164k[81]) and applying LocCon (§3.4.1) with data sampled from 100 houses, randomly selected from a pool of 25 houses from the training split of Gibson-mini and 720 houses from HM3D [118]. We again highlight that *no labeled 3d mesh* was used in this training, since they are expensive and mostly unavailable for HM3D. *Self-Pol* is trained using the method of §3.4.2, from semantic maps created and self-labeled (with *Self-Seg*) from frontier-based exploration in the 25 Gibson houses.

We define notations used in the remaining sections. We denote the "off-the-shelf" segmentation model (aka deeplabv2 [27] pre-trained with COCO-things 164k [81]) as *O-t-S*. We replace *Self-Seg* and *Self-Pol* with their fully supervised (using all the available simulator annotations) counterparts; we call this agent "*FullSup* agent" (and its components *Full-Seg* and *Full-Pol*) and our self-supervised agent "*SelfSup* agent." *Full-Seg*, initialized with *O-t-S.*, is trained with fully supervised training from 75K randomly sampled, labeled images from the 25 Gibson houses, and *Full-Pol* is trained with PONI using the ground-truth labeled 3d mesh of the 25 houses. We follow the same protocols of [27] and [119] (e.g. the pool of hyperparameters to optimize within, output dimensions) to train *Full-Seg* and *Full-Pol*, respectively.

3.5.1.1 Real World ObjectNav

To show the transfer of our ObjectNav agent (§3.4.2, §3.5.2.1) to real houses, we built a robot from a Jackal base that has similar specifications to the simulation settings (§3.3). Perception is performed with an Intel Realsense D435 RGB-D camera mounted at 0.88 meters from the floor and all computation was run locally in real-time on an Nvidia Orin (Fig. 3.4 (a)). For localization, we pre-map the environment with the `ROS RTAB-map` package when first entering a new house. When executing a policy, the robot receives its pose from the ROS SLAM package and uses the existing map only for its occupancy map instead of that from the semantic mapper; all other outputs of the semantic mapper, including mapping of explored areas and semantic categories, are as in simulation.

TABLE 3.2: **ObjectNav Results** Success rate of 18 tasks in 3 AirBnBs.

Category	**Success**
Total	66.6%
Chair	33.3%
Couch	100.0%
Potted Plant	66.6%
Bed	100.0%
Toilet	100.0%
TV	33.3%

Experiments were performed in three rented AirBnBs where we set up three tasks for each of the 6 goal objects. Because a Jackal base is larger than the agent radius in simulation, we start the agent in areas that are wider than 1.5 meters, which is usually in the living room or the kitchen. An example experiment is shown in Figure 3.4. Across 18 tasks, we obtain success rate of 66.6%. We show two example runs in the supplementary video. Out of 6 failures, 4 were due to misrecognition of goal objects (e.g. recognizing a *"white couch"* as *"bed"*), and 1 was due to depth/ segmentation misalignment, 1 was due to localization failure.

Table 3.3: Semantic segmentation gains from real-world *in-situ LocCon* finetuning.

	Off-the-Shelf.	**Self.**	**Full.**
Initial	61.1	61.9	58.8
with In-Situ	62.2	62.8	61.7
Performance Δ	1.1 $\uparrow$	0.9 $\uparrow$	2.9$\uparrow$

3.5.1.2 Self-Supervised In-Situ Training of Visual Perception

In §3.5.1.1, we saw that a major error mode in real world ObjectNav is visual perception; the robot stops at wrong goal objects or goal objects are seen but misrecognized. Since we find that visual perception generalization failures somewhat *nullify* efficient exploration/ nav policy, we focus on *in-situ* learning for visual perception. We show that *LocCon* training of semantic segmentation can remedy these issues.

To replicate applying LocCon in a the real world, we collected location consistency images from 4 airbnbs, of which three are disjoint from the ones in §3.5.1.1. We replicate the sampling procedure of §3.4.1 as follows; we use a pre-map of the environment to sample an (x, y) and then pick z as in §3.4.1. We place a tripod with Intel Realsense D435 RGBD camera mounted at 0.88 meters at 5 to 10 locations around the picked (x, y, z) among navigable points from which (x, y, z) is visible, mimicking the "intermediate stops" of §3.4.1. We collected a total of 479 images and 70 (x, y, z)'s. For validation, we annotate five images (and exclude them from the training set) from the AirBnB that was used in §3.5.1.1, so that all goal object classes appear at least once. We show examples of train/ val images in the supplementary video.

In Table 3.3, we show the Mean IOU of *Off-the-Shelf.*, *Self-Seg* (Self.), and *Full-Seg* (Full.), when evaluated and *in-situ* trained on real world data with *LocCon*. First, we find that Full. gives the worst and Self. gives the best initialization in the real world. This suggests that our *LocCon* training makes simulation training useful for Sim2Real transfer, despite the data challenges shown in §3.5.2. Second, we find that our LocCon training gives performance boost for all models. While we do not include plots due to space limmitations, we found that as we increase the number of training images for LocCon training, the IOU increases, in both simulation and the real world.

We retrofit *Self-Seg* trained with In-Situ (Table 3.3) to real world ObjectNav (§3.5.1.1) and examine if this adapted model can change any failures to non-failures (Fig. 3.1). More specifically, we replicate the same trajectory the agent took in each of the failed tasks, but use *Self-Seg* trained with In-Situ (Table 3.3) in the semantic mapper. We observe that this turns one out of six failed tasks as a non-failure (Fig. 3.1).

3.5.2 Simulation

In §3.5.2.1, we present ObjectNav results in simulation. In §3.5.2.2, we analyze navigation error modes, with special attention to those caused by simulation artifacts (Fig.3.5). In §3.5.2.3, we provide an in-depth analysis of self-supervision vs. full supervision (with annotations) for visual perception, and conclude that the latter tends to learn artifacts from bad rendering and annotations. In §3.5.3, we analyze the performance and behavior of a self-supervised nav policy.

3.5.2.1 ObjectNav in simulation

While our focus is on the deployable real-world system, we also evaluate our agent in simulation (two validation sets of Gibson ObjectNav) for fair comparison to both existing supervised models [23] (Gibson-1) and fully self-supervised approaches [3] (Gibson-2). Table 3.4 shows ObjectNav results of our approach, compared with other fully supervised and partially (visual perception is self-supervised while policy was trained with annotated mesh)/fully self-supervised methods. Succ. denotes task success rate across all validation tasks, SPL the Succ. weighted by path length, and DTS the average distance to the 1 meter (stop threshold) radius of the true goal. Our method outperforms fully self-supervised methods on Gibson-2 and performs as competitively as methods with more supervision on Gibson-1.

From Table 3.4, we do observe that directly training with simulation annotations gives advantages; however, we find that this in-domain advantage is due to adapting to the *artifacts* and *noise* of simulation, not learning the indoor semantics that is transferrable to the real world (Fig. 3.5). In §3.5.2.2, §3.5.2.3, §3.5.3, we provide in-depth analyses of the challenges of supervised training with simulation artifacts.

3.5.2.2 Error Modes

Table 3.5 shows the breakdown of the error modes of *FullSup* agent (denoted Full.) and *SelfSup* agent (denoted Self.). We find that simulation reconstruction and annotation can be very noisy from certain views and can produce RGB that are highly unlikely in the real world (Fig. 3.5), and first enumerate the errors that are *unique to simulation*, highlighted with ∗ in Table 3.5. These errors are *Bad Rendering* (Fig. 3.5(a); deformed

TABLE 3.5: **Error modes of Gibson-1 ObjectNav.** ∗: Error modes *unique to simulation.*

Error Modes	Self.	Full.
Misrecognition	13.2%	16.0%
Bad Rendering*	***8.0%***	***1.2%***
The Rest		
Policy Error	3.6%	2.4%
Mapping Error	6.0%	7.2%
Mislabeled Goal*	5.6%	5.2%
Depth Render Error	1.6%	0.8%
⇒ Total	16.8%	15.6%

Table 3.4: **ObjectNav validation results in Simulation (Gibson-1 and -2).** We compare fully supervised, partially self-supervised, and fully self-supervised methods with our approach. We perform competitively with SOTA fully supervised methods on Gibson-1 and significantly outperforms end-to-end self supervised methods on Gibson-2. Cells with N.A. are due to gaps in the literature; ZER and ZSON only report results on Gibson-2, while the other methods only do so on Gibson-1. $*$ stands for our own implementation of these methods, since the original code of [119] that reports results of SemExp and PONI on Gibson-1 was not fully available.

	Gibson - 1 [5]			Gibson - 2[3]	
Method	Succ. ↑	SPL ↑	DTS ↓	Succ. ↑	SPL ↑
Fully Supervised w. Annotations					
`DD-PPO` [163]	15.0	10.7	3.24	N.A.	N.A.
`SemExp` [**ogn**]	65.5*	36.5*	1.45*	42.8*	20.8*
`PONI` [119]	66.2*	36.8*	1.41*	34.9*	20.3*
Partially Self-Supervised					
`SemExp + SEAL` [24]	62.7	33.1	N.A.	N.A.	N.A.
Fully Self-Supervised					
`ZER` [3]	N.A	N.A	N.A	11.3	N.A
`ZSON` [90]	N.A	N.A	N.A	31.3	12.0
`Ours`	**60.0**	31.2	1.89	**36.0**	**19.2**

rendering causes Self. to incorrectly detect
goal objects and incorrectly terminate Nav task) and *Mislabeled Goal* (Fig. 3.5(b); the goal objects are incorrectly labeled). We find that Self. is more susceptible to these errors, with especially *Bad Rendering* happening ~7 times more in Self. than Full. While this gap essentially explains the performance gap between *SelfSup* agent and fully supervised methods (Table 3.4), we note that these errors are caused by simulation artifacts. This agrees with our result in §3.5.1.2, that adapting to these errors *harm* real-world transfer.

Among errors not caused by reconstruction/ annotation challenges, the most common error mode for both agents is *misrecognition* - goal objects not recognized even though an instance of it was seen or misrecognized (e.g. "couch" as "bed"); surprisingly, Self. is more robust to this error. The other errors consist of policy error (bad exploration led to not finding a goal object), mapping/ planning error (agent goes out of the map and gets lost), and depth render error (agent stops outside the 1m radius due to depth error).

3.5.2.3 Full. vs Self. Semantic Segmentation

We further analyze the behavior of *Self-Seg* (denoted Self.) and *Full-Seg* (denoted Full.), to better understand the results of Table 3.5. We present results on two test sets, which each consist of 500 images from 5 validation houses of Gibson. We realize that simply randomly teleporting agents and collecting images (Test with *Sim Noise*) results in many images with bad rendering (that will not appear in the real world) and bad

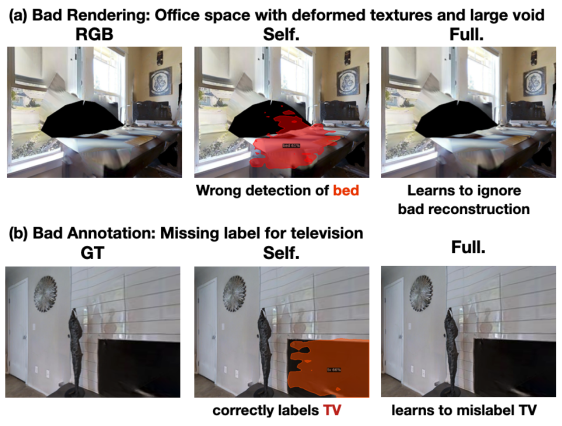

(a) Bad Rendering: Office space with deformed textures and large void
RGB
Self.
Full.
Wrong detection of bed
Learns to ignore bad reconstruction
(b) Bad Annotation: Missing label for television
GT
Self.
Full.
correctly labels TV
learns to mislabel TV

FIGURE 3.5: Data challenges: bad rendering and annotations.

TABLE 3.6: **Semantic segmentation results on simulated data.** Results are on the 6 goal objects (IOUs of all 15 objects are in Table 3.7); *Self-Seg* and *Full-Seg* are denoted Self. and Full.

Metric	**Test with *Sim Noise***			**Test w.o. *Sim Noise***		
	O-t-S.	Self.	Full.	O-t-S.	Self.	Full.
Mean IOU	38.3	44.1	52.7	44.2	53.1	53.1
Freq. IOU	92.2	93.0	94.0	94.9	95.6	95.6
Pixel Acc.	95.1	95.9	96.5	96.8	97.4	97.4

annotations (Fig. 3.5). Thus, we further evaluate on a test set of 500 images with obvious reconstruction/ annotation errors filtered by the authors (Test without *Sim Noise*).

Table 3.6 shows that *Self-Seg* (Self.) yields performance on par with *Full-Seg* (Full.) across all metrics, when evaluated without simulation noise. However, on Test with *Sim Noise*, the performance of Self. sharply drops while that of Full. nearly stays the same. This leads to the conclusion that supervised training with reconstruction noise leads to learning simulation noise and artifacts; in Fig 3.5, we observe that Full. learns to both ignore bad renderings and adapt to the bad annotations of simulation. This explains the $\sim$7x gap in the *Bad Rendering* error in Table 3.5 and the result that Full. harms real-world transfer, presented in §3.5.1.2.

Table 3.7 shows the class IOU of the three models across object categories. Interesting cases are bolded, but we draw the reader's attention in particular to cases where full supervision has hurt performance. This appears to happen with small objects (*e.g. Cup* and *Book*); we find these objects are often mislabeled. Because *Self.* does not use labels, we are unaffected by this noise – a good sign for more realistic scenarios. However, at present, small objects pose a major challenge for all models regardless of their supervision.

3.5.3 Full. vs Self. Nav Policy

Table 3.7: **Semantic segmentation by object category**. We provide a category level IOU breakdown of results in Table 3.6. *Self.* shows a significant boost over *O-t-S.* for **Toilet**, Bed, and Oven; Full. can hurt performance sometimes (*e.g.* ***Book***, ***Cup***, ***TV***).

Category	**O-t-S.**	**Self**	**Full**
Chair	41.9	40.3	42.6
Couch	57.7	60.7	62.4
Potted Plant	25.0	29.6	30.1
Bed	49.0	64.8	75.8
Toilet	**5.53**	**37.1**	**48.7**
TV	***32.7***	***41.5***	***14.5***
Dining Table	11.0	11.9	19.6
Oven	17.6	35.1	54.4
Sink	17.6	25.7	34.8
Refrigerator	25.3	36.7	45.2
Vase	0.88	1.79	4.93
Bottle	0.00	0.22	0.00
Clock	0.00	0.00	0.00
Cup	***28.4***	***33.2***	***0.00***
Book	***11.2***	***5.33***	***1.18***
Total Mean	27.4	32.5	33.2

Table 3.8: ObjectNav *"Success Rate/ SPL"* on Gibson-1 for *segmentation models* with different *navigation policy models*.

	Segmentation Models	
Nav Policy	Self-Seg	Full-Seg
Self-Pol	60.0/ 31.2	64.4/ 33.2
Full-Pol[3]	56.3/ 28.9	58.8/ 28.8
SemExp	59.0/ 32.3	63.4/ 32.2

First, we investigate if the gap of *policy error* in Table 3.5 is meaningful. In Table 3.8, we compare ObjectNav performance on Gibson - 1 with a cross product of *two segmentation models* and *three ObjectNav policy models*; the former are *Self-Seg* and *Full-Seg*, and the latter are *Self-Pol*, *Full-Pol*, and Sem-Exp [23] trained with ground truth locations from labeled 3d meshes. Table 3.8 shows that *Self-Pol* performs as well as *Full-Pol* across segmentation models, implying that existing expensive methods that train the Nav policy with self-supervision, such as using ground truth locations of objects, collecting demonstrations, or using ImageNav (in which the goal location comes for free as mentioned in [90]) are not needed.

Furthermore, this result implies that policy training is fairly agnostic to noise; in Table 3.5, we see that *Self-Seg* introduces more noise to the self-labeled semantic map (from *Bad Rendering*) than the mislabeling in the mesh (from *Mislabeled Goal*). Still, *Self-Pol*, while trained with more noisy data, performs on par with (even slightly better than) its counterparts trained with mesh annotations. We hypothesize that this is because the Nav Policy is trained to capture abstract knowledge on house layouts (as mentioned in [23, 119]) and thus is fairly robust to both noises in Fig. 3.5.

3.6 Conclusion and Next Steps

We present a self-supervised method to train all components of an ObjectNav agent that works in the real world. We find that our method can provide robust transfer to the real world, while supervised training with simulation annotations can actually harm transfer

[3]The lower performance of Full-Sup PONI may be because of misalignment between the pre-trained model provided by [119] and our reimplementation of the navigation components which we could not reproduce with the provided code. We also compare against Full-Sup SemExp.

performances. Most importantly, we show that our method can work for *in-situ* training to further improve ObjectNav performance, which encourages future research efforts on *in-situ* training.

However, we also find in our results that the improvement through self-supervision alone has its limits. Ideally, if the system could adapt through manageable human interactions—recognizing what it knows and doesn't know and requesting information through dialogue—it would significantly enhance its functionality. On the other hand, it can adapt to a partitcular environment by observing human activity. For instance, a robot might learn about object locations and their contexts by observing human actions and listening to instructions like "Put it on the short table," followed by observing where humans actually place items. To this end, we explore embodied dialogue in Chapter 4, investigate understanding human activity and language in Chapter 5, and propose initiatives in metacognition in Chapter 6. These discussions aim to equip robots with better tools for learning from and interacting.

Chapter 4

Data and Model challenges in Embodied Dialogue

This chapter examines embodied dialogue, with the lens of evaluation and training. For evaluation, we observe that imitation learning and related low-level metrics do not effectively meet the objectives of embodied dialogue research, leading to significant challenges during both the training and evaluation stages. For training, we compare how agents developed via modular approaches and those trained through end-to-end behavior cloning respond to instructions within a human dialogue context. Our findings indicate that agents trained end-to-end are particularly vulnerable to issues such as misaligned intent and incorrect demonstrations in the training data. Additionally, we find that existing models struggle to effectively ground dialogues, highlighting a key area for future improvement.

4.1 Introduction

Dialogue is key to how humans collaborate; through dialogue, we query information, confirm our understanding, or banter in a friendly manner. Since communication helps us work more efficiently and successfully, it is only natural to imbue for collaborative agents with this same ability. Most work has focused on grounded dialogues for embodied navigation [145, 32, 123] or limited interaction [141], which are narrower domains than the larger instruction following literature [144, 142, 133, 13, 11, 98].

The first step towards engaging in a dialogue, is being able to understand and learn from it. Picture a child watching their parents with the goal to learn by imitation. They witness instructions, clarifications, mistakes, and banter. Begging the question: *What should one learn from noisy natural dialogues?*

Unlike in alinguistic tasks where modeling humans has recently proved helpful for search strategies [36], we focus on language based tasks that require learning lexical-visual-action correspondences. We discuss and compare three paradigms: Instruction Following (IF), actions from Entire Dialogue History (EDH) and Trajectory from Dialogue (TfD). The novel TEACh dataset [109] proposes EDH as the primary metric and uses the Episodic Transformer (ET) [110] trained with behavior cloning as their baseline. We also include comparisons to the EDH competitive Symbiote[1] system and we adapt FILM [98], a recent method for general IF, to dialog instruction following (DIF) on TEACH. FILM and Symbiote belong to a different family of models, focusing on abstract planning trained at a higher semantic level than behavior cloning. This approach appears crucial for generalization and TfD evaluations.

Most importantly, we analyze the human behaviors in TEACH and the corresponding effect on ET, Symbiote, and FILM, as representatives of existing model classes. From our findings, we suggest there are three major challenges the community must tackle to move forward in the nascent field of Dialogue based Instruction Following:

Recognizing mistakes Behavior cloning encourages replication of low-level errors, but not high-level intentions. Agents should learn to construe high-level intentions of demonstrations and to deviate from demonstration errors.

Grounding queries No approaches correctly ground *"queries"* requesting information.

Evaluation Agent evaluation should focus on achieving goals rather than immitating procedures.

4.2 Piror Work

Instruction Following A plethora of works have been introduced for instruction following without dialogue [25, 94]; an agent is expected to perform a task given a language instruction at the beginning and visual inputs at every time step. Representative tasks are Visual Language Navigation [6, 46, 197] and instruction following (IF) [133, 137], which demands both navigation and manipulation. Popular methods rely on imitation learning [110, 137] and modularly trained components [11, 98] (e.g. for mapping and depth).

[1]Model outputs provided by correspondence with the team.

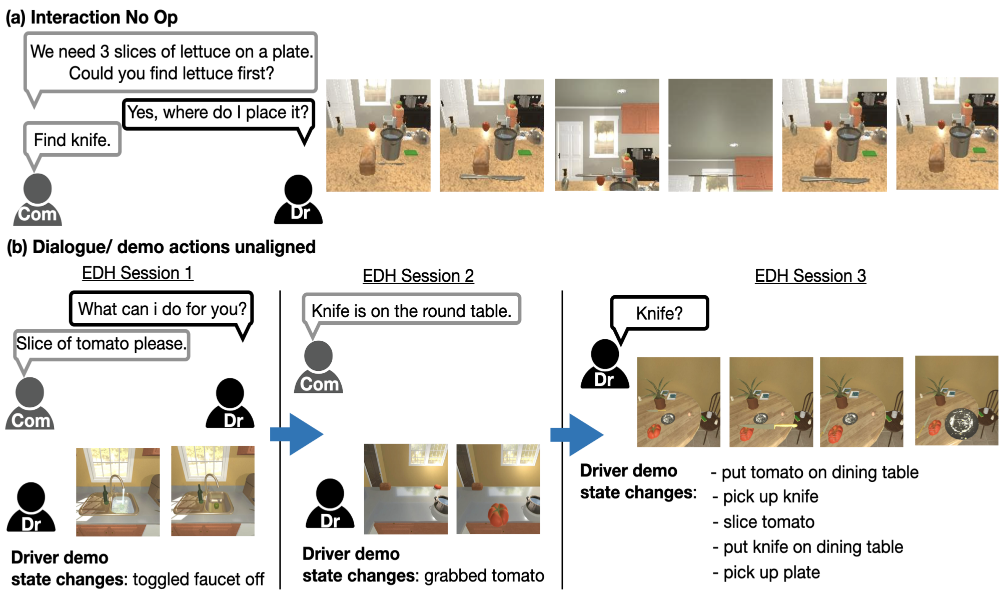

(a) Interaction No Op
We need 3 slices of lettuce on a plate. Could you find lettuce first?
Yes, where do I place it?
Find knife.
Com
Dr
(b) Dialogue/ demo actions unaligned
EDH Session 1
What can i do for you?
Slice of tomato please.
Com
Dr
Dr
Driver demo
state changes: toggled faucet off
EDH Session 2
Knife is on the round table.
Com
Dr
Driver demo
state changes: grabbed tomato
EDH Session 3
Knife?
Dr
Driver demo
state changes:
- put tomato on dining table
- pick up knife
- slice tomato
- put knife on dining table
- pick up plate

FIGURE 4.1: Examples of suboptimal demonstrations that can be harmful for training and evaluation. (a: **no-op**) The driver grabs a knife, looks up and down, and put its down, although nowhere in the dialogue indicates to do these actions, nor do they facilitate the high-level goal. (b: **unaligned** intent) In EDH sessions 1 and 2, the commander asks for an item (a slice of tomato) and provides the location of the knife, but the driver performs unaligned actions. In session 3, the driver suddenly asks "knife?", but performs a long sequence of implied but not stated actions.

Dialogue Instruction Following Instruction Following with Dialogue [131] has mostly addressed navigation. [145, 141] built navigation agents that ground human-human dialogues, while [32, 104] showed that obtaining clarification via simulated interactions can improve navigation. Manipulation introduces grounding query utterances that involve more complex reasoning than in navigation-only scenarios [143]; for example, the agent may hear that the object of interest (e.g. "apple") is inside "the third cabinet to the right of the fridge."

Imitation Learning vs Higher semantics While behavior cloning (BC) is a popular method used to train IF agents, it assumes that expert demonstration is optimal [189, 168]. TEACh demonstrations are more "ecologically valid" [150] but correspondingly suboptimal, frequently containing mistakes and unnecessary actions. Popular methods that deal with suboptimal demonstrations involve annotated scoring labels or rankings for the quality of demonstrations [168, 14]. Such additional annotations are not available in existing IF and DIF benchmarks. In this work, we empirically demonstrate the effect of noisy demonstrations on an episodic trained with BC for DIF.

4.3 Tasks Explanation

TEACh focuses on two tasks: Entire Dialogue History and Trajectory from Dialogue. Despite what the name implies, EDH is an evaluation over partial dialogues (e.g. from state S_t begin execution to S_T). TfD starts an agent at S_0 and asks for a complete task completion provided the full dialogue.

In both settings, the agent (driver) completes household tasks conditioned on text, egocentric RGB observations, and the current view. An instance of a dialogue will take the form of a command: *Prepare coffee in a clean mug. Mugs are in the microwave.*, the agent response *How many do I need?*, and commander's answer: *One*, together with a sequence of RGB frames and actions that the agent performed during the dialogue. As in this example, the agent has to achieve multiple subtasks (e.g. find mug in the microwave, clean mug in the sink, turn on the coffee machine, etc) to succeed.

In TfD, the full dialogue history is given, and the agents succeeds if it completes the full task itself (e.g. make coffee). In EDH, the dialog history is partitioned into "sessions" (e.g. Fig. 4.1) with the corresponding action/vision/dialogue history until the first utterance of the commander (*Prepare* $\sim$ *microwave.*) being the first session and those after it being the second. In EDH evaluation, the agent takes one session as input and predicts actions until the next session. An agent succeeds if it realizes all state changes (e.g. *Mug: picked up*) that the human annotator performed. Succinctly, TfD measures the full dialogue while EDH evaluates subsequences.

4.4 Methods

TEACh is an important new task for the community. We explain the models used for the analysis in this chapter. We analyze the provided baseline (ET), retrofit the ALFRED FILM model, and requested outputs from the authors of Symbiote on the EDH leaderboard.

ET is a transformer for direct sequence imitation approach, that produces low-level actions conditioned on the accumulated visual and linguistic contexts. In contrast, FILM consists of four submodules - semantic mapping, language processing, semantic policy, and deterministic policy modules. For the adaptation, we refactored the original code of FILM to the TEACH API, retrained the learned components of the semantic mapping module for the change in height and camera horizon, and retrained/rewrote the language processing module to take a dialogue history as input. The language processing (LP) module of FILM maps an instruction to a *task type* and instruction-specific *arguments*. For TfD this maps a dialogue to a sequence of tasks, while for EDH only the subsequence is mapped to an immediate action. Symbiote is a competitive modular method for EDH whose language understanding component is designed for dialogues (§B.1).

4.5 Experiments and Results

First we present how TEACh and, by extension, future embodied dialogue settings present novel training and evaluation challenges as the data, by virtue of its authenticity, includes substantial noise in both the training and evaluation (despite filtering by the authors, as explained in §B.3). See §B.2 for how statistics were computed, for those not explained in this section.

4.5.1 Explanation of Metrics

Evaluation for both EDH and TFD is done by SR (success rate), GC (goal condition success rate), and their path-length-weighted versions. Success Rate (SR) is a binary indicator of whether all subtasks were completed. The definition of "subtasks" is different for EDH and TfD; for the former, they are all tasks required to realize state changes done by the **human demonstration** that are relevant to the ultimate task (e.g. The demo state changes in each session of Fig.4.1 (b)). Thus, the state changes brought by the human is considered ground truth in EDH evaluation; this brings multiple challenges further discussed in §4.5.2. On the other hand, for TfD, the subtasks are independent of what was done in the demo; for example, as long as an agent "slices the tomato" correctly for the task of Fig.4.1 (b), its SR will be 1 for this task. [2]

The goal-condition success (GC) of a model is the ratio of goal-conditions completed at the end of an episode. Both SR and GC can be weighted by (path length of the expert trajectory)/ (path length taken by the agent); these are called path length weighted SR (PLWSR) and path length weighted GC (PLWGC). Higher is better for all metrics.

4.5.2 Challenges in Evaluation

Irrelevant Actions Humans often explore the environment, or simply play around in the middle of a task. This means they may flip a switch completely unrelated to the goal. Table 4.1 are representative state changes that do *not* have direct correspondence with the dialogue, and the percentage of human demonstrations that contain these actions.

[2]While the github repository `https://github.com/alexa/teach#downloading-the-dataset` mentions that the EDH tasks were filtered so that "the state changes checked for to evaluate success are only those that contribute towards task success in the main task of the gameplay session the EDH instance is created from", we find that even after this filtering, there exist many EDH tasks with subotpimal demonstrations as in Fig.4.1.

Table 4.1: Representative state changes that do not have direct correspondence with the dialogue, and the percentage of human demonstrations that contain these actions. The action types listed here bring "state changes" that are counted during EDH evaluation. For example, an agent would "fail" an EDH task if the human annotator of the task left coffee machine off at the end, although the task (e.g. "Make coffee") or dialogue itself does not mention that it be left on.

Unnecessary State Changes	**Val Seen**	**Val Unseen**
Coffee Machine on/off	47.73	47.54
Picked up and not placed	25.49	23.41
Faucet on/off	12.68	10.59
Stove/ Microwave on/off	35.61	28.31
⇒ Total	38.98	35.60

It is not always clear if this behavior is because of misunderstandings, boredom, or curiosity. For example, we can classify a large number of navigation and interaction "No Op"s, or action sequences that return to the original state (e.g. turning around in place). In principle, these might be information seeking, to build a better map of the environment, but in practice, many of the demonstrations do not seem to exhibit those properties, particularly in extreme cases like repeatedly picking up and putting down the same object. The percentages of prevalence of these unnecessary actions in both training and validation are shown in Table 4.2.

Table 4.2: Representative unnecessary action types that do not have associations with the high level goal or the dialogue, and the percentage of demonstrations that contain these action types in train/ valid seen/ valid unseen splits.

Suboptimal Actions	**Train**	**V. Seen**	**V. Un**
Navigation No Op			
Turn Left/ Right x 4	3.07	2.30	2.33
Forward + Backward	4.80	7.57	4.23
Pan Right + Pan Left	5.83	4.77	8.10
Turn Right + Turn Left	13.13	13.49	10.89
⇒ Total	22.41	23.03	21.36
Interaction No Op			
Toggle off + on same obj	1.39	1.81	1.58
Open + Close same obj	1.46	1.80	1.30
Place + Pick up same obj	25.06	27.80	28.34
⇒ Total	26.76	30.10	30.57
Interaction w. unrelated obj.	14.10	16.61	13.59
Demo unaligned w. dialog	25.25	22.49	23.40

The prevalence of these actions can be viewed as a positive for realism and even helpful if teaching how to search, but pose a challenge for evaluation.

Table 4.3: EDH and TfD performances of E.T., Symbiote, and FILM. While the SR on TfD is very low for all models, E.T.'s performance on TfD drops significnatly due to replication of errors and lack of grounding of high-level semantics.

	Valid Seen		**Valid Unseen**	
	GC	SR	GC	**SR**
Entire Dialogue History (EDH)				
E.T.	15.7 [4.1]	10.2 [1.7]	9.1[1.7]	7.8[0.9]
Symbiote	25.9 [5.3]	**16.1 [2.6]**	17.2 [2.9]	10.1 [1.2]
FILM	**26.4 [5.6]**	14.3 [2.1]	**18.3 [2.7]**	10.2 [1.0]
Trajectory from Dialogue (TfD)				
E.T.	1.4 [4.8]	1.0 [0.2]	0.4 [0.6]	0.5 [0.1]
FILM	**5.8 [11.6]**	**5.5 [2.6]**	**6.1 [2.5]**	**2.9 [1.0]**

Penalizing Agents for Accuracy Using a human's action trace as the ground truth, means agents are penalized for skipping erroneous actions. This leads to a misleading mismatch in performance between EDH and TfD. Additionally, EDH inflates model performance as it includes subsequences which are nearly deterministic (e.g. all but the last "placing" action). Table 4.3 contains EDH scores for our three comparison models and TfD for ET/FILM. As suggested by authors of related papers, we treat Unseen Success Rate as the most important metric (seen in **blue**).

Note, that an ideal evaluation would capture both "actions in context" and "task success." In the following section breakdown the overall numbers presented here to understand if models more carefully.

4.5.3 Challenges in Training

Behavior Cloning with Suboptimal Demonstrations We find that ET trained with behavior cloning repeats the same mistakes in novel scenes that are frequent in demonstrations. We examine two kinds of mistakes in demonstrations - (1) No Op interactions, in which consecutive interactions produces futile state changes (e.g. Placing and immediately picking up the same object) and (2) Interactions with unrelated objects (e.g. picking up "saltshaker" while making coffee). In Table 4.4 we compare what percent of model predictions in seen and unseen scenes replicate the no-op behavior.

While hard to quantify, we also note that the higher intention of seemingly unnecessary human demonstrations (e.g. to explore, to understand, etc) are not replicated by ET. This is backed by our observation that ET tends to be stuck in many (10 or more) repetitions of the same No Op/ unnecessary actions, until the end of the task or before resuming to perform other actions.

Table 4.4: Percentage of tasks in which a model exhibited replication of No Op actions.

Suboptimal Actions	**ET**		**Symbiote**		**FILM**	
	S	U	S	U	S	U
No Op (same obj)						
Toggle off + on	0.0	0.1	0.2	0.1	0.0	0.0
Open + Close	2.5	1.5	0.0	0.2	0.0	0.0
Place + Pick up	45.1	47.1	4.9	2.5	0.0	0.0
⇒ Total	46.2	48.1	5.1	2.8	0.0	0.0
Unrelated obj.	24.0	20.3	27.5	30.6	15.9	12.10

Note that even Symbiote is exhibiting some no-op behaviors, but as the model supervision/structure becomes more abstract (ET vs FILM) this disappears, leaving only object choice errors.

Grounding Queries Key to dialogue is language based information seeking. A target object may be located in a closed receptacle (cabinet, etc); in this case, the agent has to query the commander for its location, as a human would. We examine whether models ground *query utterances* into meaning and accurate actions, since this is one essential aspect of dialog grounding. While there are utterances with other essential intents, such as confirmation, we focus on query utterances since these are relatively easy to extract mechanically.

In Table 4.5 we consider a subset of tasks that involve "query utterances" that can be detected automatically. Specifically, we present the performance of models in terms of success rate and goal condition success on tasks that require opening a receptacle based on the answer to a question – and then measure if the models leverage the query. Not all query utterances will be of this type, but these tasks necessarily involve grounding query utterances for task success.

Queries are present in 23.05% and 25.31% of valid seen and unseen splits, respectively. This is a key challenge as it demonstrates a clear use case for dialogue and limitation of current models.

Given a statement like "the fork is *in the cabinet* left to the refrigerator", the evaluation mismatch occurs if an agent grabs a different fork on a table. This allows them to succeed, as measured by SR/GC, but not in SR/GC with Query. Notably, all models fail at query grounding, indicating they are simply ignoring the language instructions. This shows that enabling complex dialogue grounding is an important open problem for DIF. Especially, for the ultimate goal of two-agent task completion (TaTC), it is necessary that models can ground query and other essential utterances in a dialogue.

Table 4.5: We consider a task as involving "query utterances", if in its demonstration, a relevant object inside an originally closed receptacle was picked up. SR/GC measure the vanilla task success on tasks with "query utterances"; SR/GC w. Query measure if the success was achieved using information in the "query utterances."

Method	SR	GC	SR w. Query	GC w. Query
Validation Seen				
ET	8.97	14.13	0.00	0.00
Symbiote	0.00	11.76	0.00	0.00
FILM	9.59	20.26	0.00	0.00
Validation Unseen				
ET	2.39	8.69	0.00	**0.49**
Symbiote	0.00	0.00	0.00	0.00
FILM	1.29	9.79	0.00	0.00

4.6 Conclusion and Next Steps

This chapter is not an indictment of TEACh, nor an endorsement of a particular model, rather it seeks to lay out important questions and challenges that NLP will need to tackle as it moves into embodied dialogue. Unlike existing work in dialogue that looks to model human satisfaction [53] or state-tracking, DIF has the advantage of explicit and verifiable semantic goals. We pose a challenge to the community: How can we build agents where success is not tied to specific actions yet language understanding and production are accurate and fluent? As a first step, we posit that imitation learning should be avoided.

The intersection of human activity and language often occurs in contexts that are inherently ambiguous, a topic we have not explored in depth in this chapter. In the forthcoming chapter, we will introduce a dataset specifically designed to address these ambiguities and examine the performance of LLM-based modular models in interpreting and acting within such complex scenarios. This investigation aims to shed light on the capabilities of these models in real-world, dynamic environments where clarity and precision in understanding human instructions are critical.

Chapter 5

Situated Instruction Following

This chapter delves into the challenges agents face when tasked with following instructions under conditions where human intent is ambiguous or incomplete. Human intent is often ambiguous in real world scenario, since humans speak situated language. Our analysis reveals that agents demonstrate efficiency in planning and execution when the locations of objects or humans are explicitly known. However, they encounter substantial difficulties in uncertain environments where critical information may be obscured or missing. This is evident even in advanced models, such as GPT-4, which struggle to interpret and act on instructions without clear contextual cues.

5.1 Introduction

Humans naturally engage in communication that is contextually situated, providing just enough information as necessary. This is because our use of language is predicated on an assumed common ground [20], which encompasses our shared history, actions, and environment. For instance, the instruction "Can you bring me a cup?" can vary in meaning depending on the context. If spoken while the speaker is donning rubber gloves by the kitchen sink, it likely refers to a dirty cup located on the living room table. Conversely, the same request made in front of the bathroom sink typically implies a need for a clean cup. While it is possible to seek clarification, humans generally interpret and respond to such requests accurately without additional information. This capability demonstrates how humans skillfully use environmental and action cues to interpret ambiguous language, crafting meanings that are intricately nuanced and context-specific.

As robotic agents increasingly become integral to our daily lives, their effectiveness and utility critically depend on their ability to comprehend and respond to situated language— natural language spoken by humans. Without this capability, agents may prove more of a hindrance than a help, forcing users to perform tasks themselves rather

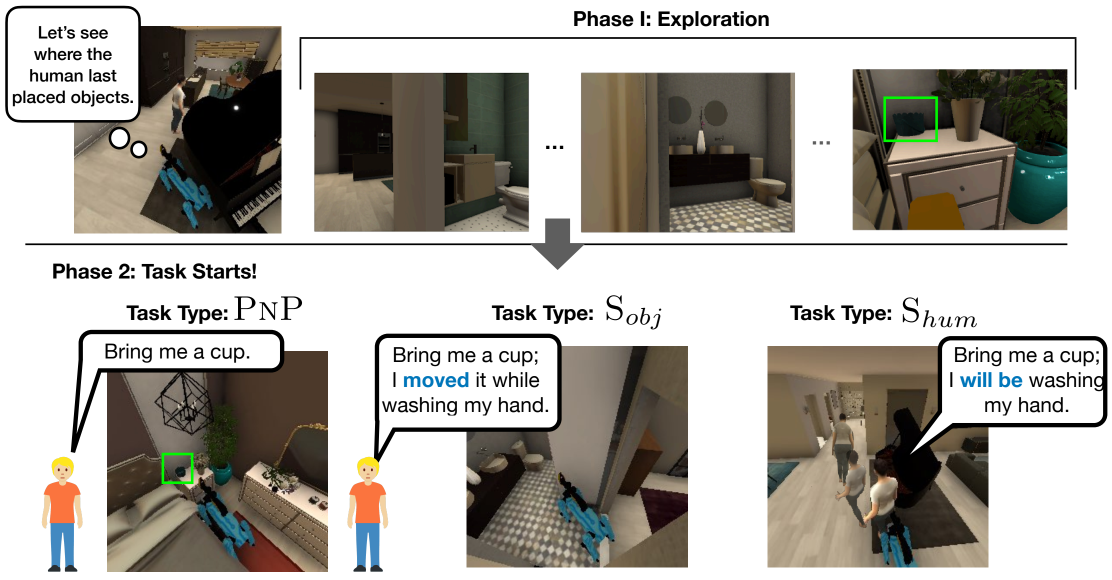

Let's see where the human last placed objects.
Phase I: Exploration
...
...
Phase 2: Task Starts!
Task Type: PNP
Bring me a cup.
Task Type: S_{obj}
Bring me a cup; I moved it while washing my hand.
Task Type: S_{hum}
Bring me a cup; I will be washing my hand.

FIGURE 5.1: **Situated Instruction Following.** The tasks in `SIF` consist of two phases: an exploration phase (phase 1) and a task phase (phase 2). PNP represents a conventional static Pick-and-Place task used for comparison, wherein the environment remains unchanged after the exploration phase. S_{hum} and S_{obj} introduce two novel types of situated instruction following tasks. In these tasks, the *objects* and *human* subjects move during the task phase. Nuanced communication regarding these movements is provided, necessitating reasoning about ambiguous, temporally evolving, and dynamic human intent.

than entrusting them to an assistant. As discussed in the field of agent alignment [74], it is often difficult for users to precisely define or articulate ideal task specifications. Consequently, an agent that demands detailed explanations might render manual task execution by humans more attractive.

Current instruction-following tasks prioritize accurate low-level instruction interpretation [133, 6, 109, 56] or use commonsense to achieve underspecified goals like object navigation [23, 35]. In contrast, our work `SIF` aims to generalize *Embodied* Instruction Following to *Situated* Instruction Following, with instructions closer to the language naturally spoken by humans. Specifically, we focus on three dimensions of situated reasoning:

1. **Ambiguity:** As in the cup example above, there is ambiguity in the instruction given by the speaker.
2. **Temporal:** A speaker's actions change how their instruction should be interpreted (e.g., clarifying an underspecified reference).
3. **Dynamic:** When the environment changes, the agent needs to decide what actions will reduce their uncertainty (e.g., following the human).

We implement our tasks in Habitat 3.0[112] as it includes simulated human agents. To ensure a fair comparison to prior work we include tasks where the environment is static (prior work) and dynamic (this work) – Fig. 5.1. The static task is a classic pick-and-place (PNP) paradigm. Formally, the instructor tells the agent to `Put [Obj] in/on [Recep]`. While prior work extends this paradigm with linguistic complexity [133, 161, 109], the baselines used for such tasks can be fairly evaluated here. We introduce a simplifying assumption and allow the agent to explore the environment to minimize the role of mapping on the performance in our reasoning benchmark – we will return to this in the ablations.

The focus of our benchmark is dynamic tasks where the agent must combine their understanding of the instruction with the human's movements to determine the correct action. The two domains (Figure 5.1) are either where the human has moved the object (S_{obj}) or where the moving human is the receptacle (S_{hum}). In these tasks, the agent receives a goal instruction (e.g., "Bring me a mug" for S_{hum} or "Put the mug in the bathroom" for S_{obj}), accompanied by hints on relocation (e.g., "I will be taking a bath" for S_{hum} or "I have moved the cup next to the be" for S_{obj}). Furthermore, in S_{hum} tasks, the human begins to walk at the start of the task, embedding the intent of relocation in both verbal directives and observable movement. The goal of the agent is to adhere to the instructions with an efficient path, accurately identifying and retrieving the specified object, and then placing it in the correct receptacle (which is "human" for S_{hum} tasks). Note, these are incredibly simple tasks that will prove very difficult for current models. If someone asks you to retrieve a mug from the kitchen and bring it to the bedroom, you intuitively wait just long enough to see which room they enter before searching for the mug. These types of ambiguity reduction strategies will elude the models, exposing their over-reliance on commonsense. Moreover, if informed that a mug has been moved to a bedroom, you would thoroughly search each bedroom, ensuring no area is overlooked. Our findings show that reasoning in ambiguous situations often confuses models, posing more challenges than simple commonsense reasoning.

We specifically target evaluation of state-of-the-art Embodied Instruction Following (EIF) baselines. We implement two such systems inspired by papers on related tasks. The first baseline, which we refer to as REASONER, is a closed-loop system incorporating a semantic map, a prompt generator, and a Large Language Model (LLM) planner. For the prompt generator, we integrated components from Voyager[151], LLMPlanner[139], and ReAct[181], tailoring them to suit our dataset's specific requirements. The second baseline, PROMPTER[64], was very successful at executing ALFRED [133] tasks despite being open-loop. We see the desired result that our static scenarios match those from existing EIF datasets [139, 64], and these LLM based approaches perform very well in tasks requiring common sense. However, their performance significantly declines when faced with situations that require reasoning about the human's behavior.

Table 5.1: Comparison of Embodied Instruction Following Datasets.

	Task Specification			**Env. Dynamics**		**Intent Change**	
	Abstract	Over-specified	Middle-ground	Ego	Ego + + Human	Language Change	Intent Unfolding
ObjectNav [5]	✓	✗	✗	✓	✗	✗	✗
ImageNav [199, 71]	✓	✗	✗	✓	✗	✗	✗
Course-grained VLN [113, 56]	✓	✗	✗	✓	✗	✗	✗
Fine-grained VLN [72, 6]	✗	✓	✗	✓	✗	✗	✗
Embodied QA [35]	✓	✓	✗	✓	✗	✗	✗
Excalibur [198]	✗	✗	✓	✓	✗	✗	✗
ALFRED [133]	✓	✓	✗	✓	✗	✗	✗
TEACh [109], DIALFRED [50]	✓	✓	✗	✓	✓	✓	✗
SIF	✓	✓	✓	✓	✓	✓	✓

5.2 Prior Work

SIF builds on instruction following, agent alignment, and situated reasoning. **Instruction Following.** We contextualize our baselines models and benchmark, in terms of both text-only and embodied research. The recent trend is to factor the task into a planner (often an LLM), memory (e.g., a map), perception, and implement tools for taking actions. Papers then often decide which components to be learned or implemented heuristically [152].

Embodied Instruction Following (EIF) We focus on tasks involving high-level task planning and navigation, excluding those with low-level (joints and end effector) motion control [176, 78, 191]. Relevant tasks are summarized in Table 5.1.

Modular models with the structure of planner (e.g. LLM), memory (e.g. semantic mapping), and perception/execution tools have shown much success in the tasks of our interest. In semantic navigation, ESC [194] and l3vm[184] have achieved SOTA performance in ObjectNav[23, 99, 5] and ImageNav[96, 199, 71] by leveraging LLMs for frontier selection based on common sense. [192, 26, 130] have used LLM for planning and tracking in Visual Language Navigation [56]. In mobile manipulation, FILM[98, 97], Prompter [64], and LLMPlanner[139] have used language models for planning and semantic search in ALFRED[133]. In real-robot domains, code-as policies [79], inner monologue [63], and ProgPrompt[135] have shown success in using LLMs for generating abstracted manipulation plans.

The effectiveness of decoupled LLM reasoners largely stems from existing benchmarks focusing on abstract common-sense reasoning [133, 23, 35, 113] or detailed, sequential guidance [56, 6, 72, 109]. These modular models tackle visual grounding and navigation through separate semantic memory modules [194, 98, 23, 130]. Thus, the *learning* challenges posed by either abstract instructions or detailed guidance boil down to common-sense object search and visually grounded tracking. Yet, these foundational skills fall short of meeting real-world needs and utility. Real-world instructions often exist in a middle ground: not overly abstract yet not excessively detailed, with both explicit statements and implied expectations. Thus, we introduce `SIF`, aiming to surpass the limitations of basic reasoning and literal planning by bridging the gap to real-world applicability.

Text-only agents Modularly chained models of planner[15, 1, 147], memory (in text) [166], meta-reasoner[183], and refiner[89] have shown much success in code generation[187], playing games[167], web navigation[195], and text-only embodied agents[181]. While most efforts have concentrated on enhancing agent reasoning through tool-use [181, 128] and self-evaluation (reflection[89, 132] and tree-of-thought [183]), less attention has been paid to navigating the complexities of real-world instructions. [75, 188, 85] have explored eliciting clarifications when human intent is ambiguous. Clarification is necessary at times, but it also comes at a cost; the focus of `SIF` is understanding ambiguous intent that is clear upon holistic understanding of the environment and speaker. [85, 45, 141] have studied when instruction intent changes and is communicated with language; `SIF` instead infers changes to intent from the human's actions. Unlike prior work focusing on agent-centric challenges in dynamic settings (e.g. moving zombies)[167, 151], `SIF` explores understanding dynamically evolving human intent.

Agent Alignment. Traditional agent alignment research[74, 2, 92] recognizes the difficulty in articulating real-world task specifications, and has primarily focused on learning reward functions from underspecified specification. Recent works on LLM and LLM agents[151, 167, 195] have delved into alignment for real-world applications, focusing on understanding intent[108], value alignment with social norms [77] and individual preference [180], and guaranteeing safety[146]. Despite these advancements, the ability of agents to interpret instructions characterized by ambiguity, evolving intent, and the interplay between agent actions and environmental dynamics is still underexplored. We introduce `SIF`, aiming to push the boundaries of agent alignment closer to real-world complexity.

Situated Reasoning. Environments evolve when there is another agent other than oneself; reasoning about action and change requires situated logical reasoning. While early work [95, 121] and STAR[165] employed formal logic and scene-hypergraphs for abstract reasoning, `SIF` focuses on fostering holistic understanding and temporal awareness in uncertain contexts.

5.3 Task Expalantion

SIF extends previous work that primarily focus on abstract, decontextualized, commonsense reasoning [50, 23, 109] by evaluating reasoning scenarios constructed in situ. Below, we explain the dataset design choices and guiding philosophy.

5.3.1 Assumptions and Scope

While real-world assistance could take many forms—including teleoperation, natural language hints, visual clarifications, or contextual corrections—this work focuses specifically on high-level action guidance as the primary form of intervention. This choice allows us to isolate the core challenge of temporal reasoning and ambiguity resolution while maintaining experimental tractability. Future work should explore the rich space of multimodal assistance types.

5.3.2 Tasks

Overview. Our tasks are structured into two distinct phases: (1) the exploration phase and (2) the task phase. During the exploration phase, the agent is allotted N steps to navigate around a static house environment where object assets are positioned. The value of N is determined to ensure the agent has sufficient steps to thoroughly scan the environment; specifically, $N = 1.5$ x (the number of steps required to achieve a complete map using frontier-based exploration techniques). Following the exploration phase, some objects are repositioned without the agent's knowledge. As the task phase commences, the agent receives an instruction (e.g., "Bring me a cup," "Put the cup in the sink"), accompanied by either direct or ambiguous information regarding which objects have been moved (e.g., "I took a cup with me. I'll be getting ready for bed"). If the task involves delivering an object to a human, the human walks into the agent's field of view as the task begins, simultaneously providing hints about their intended location ("I will be in the bathroom washing my face"). These elements, along with other strategic design decisions, ensure that the exploration phase effectively contextualizes the language directives, rendering tasks sufficiently solvable.

Note that these tasks, while challenging for current AI systems, represent basic pragmatic reasoning that humans handle effortlessly in daily interactions. For instance, when someone says "Bring me a cup; I will be washing my face" while there are multiple bathrooms, humans naturally wait to observe which bathroom the person enters before searching for the cup. This type of situated, contextual interpretation demonstrates the fundamental gap between human pragmatic competence and current AI capabilities, making SIF an important benchmark for evaluating progress toward human-level situated understanding.

Task Format. A task is defined by $\langle H, I, C, P_e, P_t, P_g \rangle$. A task is embedded in the $House$, with starting $Poses$ of assets during , agent, and the human for the *exploration* (P_e), *task* (P_t) phases and *goal* state (P_g), the goal $Instruction$, and humans $Communication$ about which objects were moved where after exploration phase.

TABLE 5.2: **Dataset stats.** Data statistics across splits and conditions.

Axis	**PnP**	**S$_{obj}$**	**S$_{hum}$**
Dynamic			
Object	✗	✓	✗
Human	✗	✗	✓
# Tasks			
Valid Seen/Unseen	40/40	40/40	40/40
Test Seen/Unseen	40/40	40/40	40/40

Agent Specs & Predefined Skills. The agent in our simulation is modeled after the Spot robot[44], equipped with an odometry sensor and RGB and Depth cameras mounted on its arm at a height of 0.73 meters; this setup follows the specifications of 640x480 resolution and a 79-degree horizontal field of view by previous work [23]. At each timestep, the agent is capable of executing one of several actions: moving forward by 0.17 meters, rotating left or right by 10 degrees, grabbing an object, or placing an object. The "grab object" action allows the agent to pick up the nearest graspable object within a 2-meter radius, whereas the "put object" action enables it to place the held item onto the nearest suitable receptacle within the same distance.

Task Types. There are three types of tasks - static (PnP), situated-object (S$_{obj}$), and situated-human (S$_{hum}$). For all three types, the overarching goal for the agent is to pick `[Obj]` and place it in/on `[Recep]`. In the S$_{obj}$ tasks, the `[Obj]` of interest is relocated between the exploration and task phases; in the S$_{hum}$ tasks, the `[Recep]` (human) begins to move at the start of the task phase. In the PnP task type, both `[OBJ]` and `[RECEP]` remain stationary; although other objects may be moved to introduce variability, the `[OBJ]` relevant to the instruction (I) is not displaced. This setup is analogous to tasks found in existing research [133, 23] and serves as a means for sanity checking and ensuring a fair comparison across models. Examples of PnP, S$_{obj}$, S$_{hum}$ tasks are in Fig. 5.1b.

Three dimensions of situated reasoning We explain how the aspects of ambiguous, temporal, and dynamic are implemented below:

- **Ambiguous**: Ambiguity in S$_{obj}$ tasks arises when multiple potential locations exist for searching an object, as informed by communication cues. For instance, if a human says, "I took the cup and moved it with me. I am washing my face," but there are multiple bathrooms in the house, the task becomes ambiguous. Likewise, ambiguous S$_{hum}$ tasks arise when the human could potentially be in several different locations. The ambiguity in these tasks was annotated by human reviewers.

TABLE 5.3: Template Instructions (I)

Task Descriptions
Static Receptacle:
Put a `[ObjectCat]` on the `[Recep]`
Put a `[ObjectCat]` in the `[RoomFunction]`
Dynamic Receptacle:
Bring a `[ObjectCat]` to me

TABLE 5.4: Template Relocations (C)

I moved the object with me; I am ...
next to the `[Recep]`
next to the `[Recep1]`, `[Recep2]`, `[Recep3]`
in `[RoomFunction]`
in `[RoomFunction]` `[WithObjects]`
doing `[RoomFunctionActivity]`

- **Temporal**: For example, if a human says, "Bring me the cup. I will be in the bathroom," and there are multiple bathrooms, the intent (the exact location of which bathroom) unfolds and becomes more clear real-time, with the human walking towards one of the bathrooms (Fig. 5.3).
- **Dynamic**: In both S_{obj} and S_{hum} tasks, an agent can decide which location to search or whether to follow the human or decide, to decrease ambiguity or understand temporally unfolding intent.

Two axes of difficulty Our tasks incorporate commonsense related to room functions and human activities, based on object placement (Tables 5.3,5.4). We add two layers of difficulty to this foundation:

Holistic understanding of language instructions: We designed and filtered tasks to avoid being trivial or solvable solely through commonsense, yet not so ambiguous as to require exhaustive search. Phase 1 exists so that the agent can scan the layout of the house, and tasks are solvable with targeted reasoning rather than comprehensive searching.

Resolving ambiguity: Tasks necessitate methodical reasoning under ambiguous intent (ambiguous tasks of S_{obj} and S_{hum}), with designs that favor agents taking actions to resolve ambiguity (intent probing), rather than comprehensively searching later. At the same time, unambiguous tasks should be solved without intent probing. Evaluation details in Sec. 6.5.

5.3.3 Dataset Construction

We explain how the tuple $\langle H, I, C, P_e, P_t, P_g \rangle$ is constructed. In a total of 10 houses, each with human-annotated room metadata (which includes details such as the top-down (x,y) coordinates corresponding to each room, the function of the room (e.g., bedroom), and grounding details (e.g., a bedroom with a yellow wall)), we place four to ten assets in `[ObjectCat]`[1]. Assets are from YCB[17], Google Scanned Objects[39], and ABO

[1]basket, book, bowl, cup, hat, plate, shoe, stuffed_toy

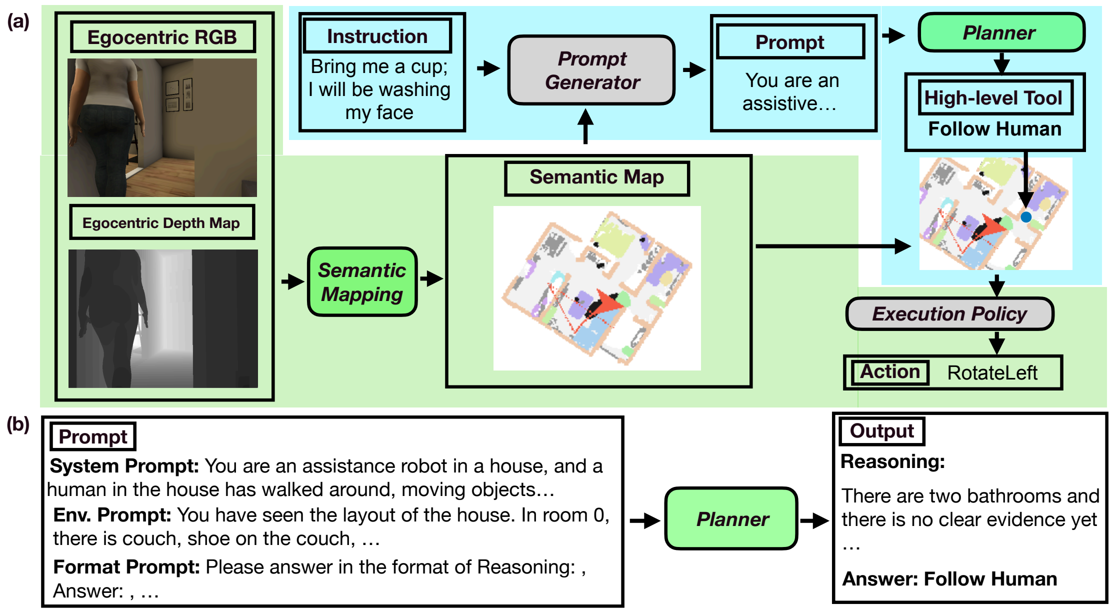
[34] datasets. We sample P_e, P_t of assets so that they are initialized in a visible space and graspable. More details on filtering trivial and unsolvable tasks are in Appendix C.1.1.

Language Directives. We explain the generation of I and C. The Instruction carries information about P_g, which is the desired location of an object necessary for task success. We first sample P_g, by sampling a room in the scene, a receptacle in the room, and then a puttable point on the receptacle. Then, we use templated language (Table 5.3) to express this information. For PnP and S$_{obj}$ tasks, we use "Put a `[ObjectCat]` on the `[Recep]`" or "Put a `[ObjectCat]` in the `[RoomFunction]`"; for S$_{hum}$ tasks, we use "Bring a `[ObjectCat]` to me". The list of possible `[ObjectCat]`, `[Recep]`, `[RoomFunction]` is shown in Table A.1.

The Communication on movement describes P_t of objects whose poses are changed from P_e. For each object that was moved, C is given with the templates in Table 5.4. In addition, for S$_{hum}$ types, C also contains where the human will be after walking during the task phase; this also follows the template of Table 5.4, with "I am" replaced by "I will be" (e.g., I: 'Bring a cup to me'. C: 'I will be organizing my bed'). `[RoomFunctionActivity]` describes common human activities in each room (e.g., kitchen - washing vegetables, bedroom - preparing to sleep); the complete list is presented in Table A.2. C is presented together with I at the start of the task phase. Although we use templated language, the diverse combination with the scene layout and object/human poses introduces substantial reasoning challenges.

Human Trajectories. For PnP and S$_{obj}$ tasks, the human is stationary. In S$_{hum}$ tasks, the human trajectory is deterministically determined given the human's P_t and P_s. The human moves naturally at a speed of 0.08m per time step. Human appearance and motion is naturally implemented as explained in [112].

Statistics and Splits The specifics of the dataset are outlined in Table 5.2. For validation and testing, the dataset comprises 40 seen and 40 unseen tasks across each type, resulting in 240 validation and 240 testing tasks in total. Additionally, we provide code to facilitate further data and trajectory generation for training purposes. The seen subsets (both validation and test) incorporate the same six houses used in the training, allowing for evaluation in familiar environments. Conversely, the unseen subsets employ four new houses to test generalization across different settings, bringing the total to ten unique houses for evaluation.



Prompt
System Prompt: You are an assistance robot in a house, and a human in the house has walked around, moving objects…
Env. Prompt: You have seen the layout of the house. In room 0, there is couch, shoe on the couch, …
Format Prompt: Please answer in the format of Reasoning: , Answer: , …

Output
Reasoning:
There are two bathrooms and there is no clear evidence yet …
Answer: Follow Human

FIGURE 5.2: **Reasoner**: (a) The semantic mapper is updated at every timestep, whereas the prompt generator and planner are activated either upon completion of the last high-level action or when a new decision is required. (b) The prompt consists of system prompt, environment prompt, format prompt.

5.4 Methods

As discussed in Sec. 5.2, many recent state-of-the-art EIF agents are modular models with an LLM planner, connected to learned/engineered episodic memory, perception, and execution tools. We present two baselines within this high-performing family. The first is REASONER, a closed-loop baseline that adapts FILM[98] and the prompts of llm-planner[139], and ReAct [181], and prompter [64], an open-loop SOTA agent built for ALFRED [133].

5.4.1 Reasoner

We adopt the modular structure of FILM [98]. REASONER operates through three main components: (1) a semantic mapper that updates an allocentric map from egocentric RGB and depth inputs, (2) a prompt generator, and (3) the planner (GPT-4o[1]) that generates high-level actions (Fig. 5.2).

Semantic Mapper. The semantic mapper creates a global representation for visual observation. As in previous work[23, 98], we process egocentric RGB and depth into an allocentric top-down map of obstacles and semantic categories using Detic[196]. The semantic categories of interest are `[ObjectCat]`, `[Recep]`, and "human." In contrast to previous works[23, 98], the most recent human and object positions are refreshed post

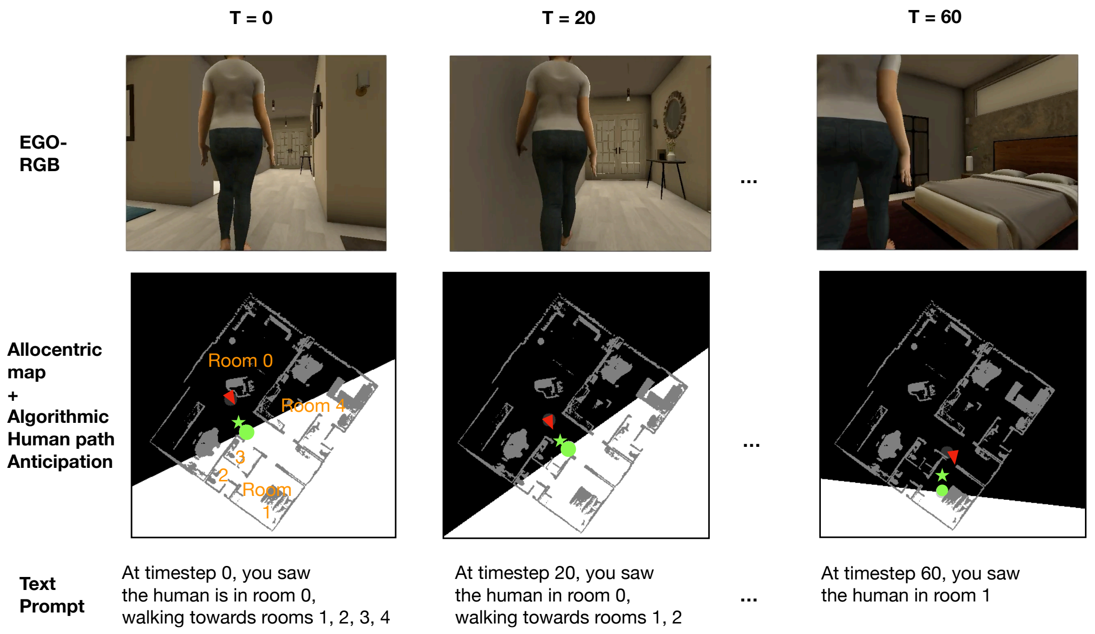

T = 0
T = 20
T = 60
EGO-RGB
Allocentric map + Algorithmic Human path Anticipation
Room 0
Room 4
3
2
Room 1
...
Text Prompt
At timestep 0, you saw the human is in room 0, walking towards rooms 1, 2, 3, 4
At timestep 20, you saw the human in room 0, walking towards rooms 1, 2
At timestep 60, you saw the human in room 1

FIGURE 5.3: **Text Prompt Generation of Human Trajectory**: The white regions in the maps are possible regions that the human might walk towards; rooms with more than half of the area included in the white region are included in the text prompt. The red triangle is the agent position/direction, green star and dot are respectively current observed human position, anticipated human position in 10 steps. The text prompt at every 20 timesteps is given to REASONER (and at time step 0 to PROMPTER which is open-loop), to decide if there is enough evidence for the clarity of the human's intent.

new observations and pick/place actions, ensuring a dynamic and accurate representation of the environment (Sec. C.3.1).

Text representation generator. The semantic map and other contexts are converted into prompts. It is a concatenation of three components: the system prompt, environment prompt, and the format prompt:

- **System:** The system prompt outlines the agent's role and and encourages it to account for uncertainty. It is presented as "You are an assistive robot in a house, aiding a human. Your observations may be incomplete or wrong."
- **Environment:** The environment prompt is a conversion of the episodic memory into text format, and contains information of the agent's current state and previously completed/failed actions. It is given in the following sequence: (1) observation of P_e during exploration phase, based on the semantic map, (2) C, regarding object/ human movements, (3) the goal instruction I, (4) the high-level action executed by agents at timesteps and their observed consequences (success/fail), (5) the agent's latest observation, based on the latest semantic map (example in Appendix C.2). Every observation is given with the caveat that it can be incorrect or missing. Past actions are interleaved with observations as in ReAct.

- **Format:** The format prompt explains action affordance and a format for chain of thought [160]. It also explains the desired effect of actions (e.g. "If you want to keep searching for object(s) or human that might exist (but you have not detected) in the current room, choose 'Explore Room X' (Table C.3).")

A complete example of the prompt is given in Appendix C.2. For S_{hum} tasks, every 20 steps, we ask whether the planner has enough evidence about the human's goal location, based on the system prompt, the current observation of human trajectory (Fig. 5.3), and the human's utterance (e.g. "I will be organizing my bed"). Example prompts about the observed human trajectory are in Fig. 5.3. If the planner answers "Enough Evidence," we proceed to ask for a high-level action (e.g. "Grab [Obj]"), providing the concatenation of the system prompt, environment prompt, and the format prompt above. On the other hand, if it answers "Not Enough Evidence," we call "follow human." We ask this question every 20 steps, until the "follow human" execution tool deems that either the agent lost track of the human or the human has stopped. Details are in Appendix C.3.3.

Execution Tools Upon receiving the prompt, the planner is prompted to choose a high-level action (Tab. 5.5); then corresponding execution tools are called. A complete list of tools are listed in Table C.3. When the execution is done, the tool sends this message, and the prompt generator creates a new prompt and the planner calls a new tool.

TABLE 5.5: Execution tools for REASONER; details in Tab. C.3.

Navigation	**Manipulation**
Go to Room X	Grab Obj
Explore Room X	Put Obj
Follow Human	Give Obj to Human.

5.4.2 Prompter

PROMPTER employs an open-loop planner.
We utilize GPT 3.5 for planning and search in lieu of BERT[37] (as in the original work[64]), tailored to our dataset's requirements. Its operational logic is straightforward: if an object has been identified on the map, PROMPTER interacts with it; otherwise, it initiates a search based on object and receptacle relationships. The planning phase utilizes a template that is populated with specific details, executed once at the task's outset. For instance, in response to a command like "Put a shoe on the couch," GPT 3.5 is prompted to formulate a high-level plan adhering to a structured format (e.g. "`[Pick up OBJ, Put on the RECEP]`)." This ensures GPT 3.5 primarily focuses on filling in the template's variables. For S_{hum} tasks, we ask if there is enough evidence about the human goal location, based on the layout of the house and the human utterance; due to

TABLE 5.6: **SPL** performance of PROMPTER and REASONER across splits. In each sectioned-row, the top row assumes oracle perception (semantic segmentation and manipulation); the bottom row assumes learned semantic segmentation and heuristic manipulation. To minimize the burden on API costs and time, we have limited LLM API calls for plan generation to 15 times for both PROMPTER and REASONER. SR performance is shown in Table D.4.

Model		Val Seen			Val Unseen			Test Seen			Test Unseen		
Planning	Perception	PnP	S_{obj}	S_{hum}	PnP	S_{obj}	S_{hum}	PnP	S_{obj}	S_{hum}	PnP	S_{obj}	S_{hum}
Oracle	Oracle	98	100	95	100	100	100	98	93	98	95	100	98
	Learned	46	46	59	41	30	54	52	30	69	44	47	46
Prompter[64]	Oracle	66	27	25	67	38	28	61	25	23	52	30	19
	Learned	16	10	11	19	8	10	24	3	8	18	7	6
Reasoner	Oracle	82	61	23	78	49	39	73	58	29	81	49	34
	Learned	21	8	12	24	11	12	29	2	15	18	14	15

open-loop planning, we ask this only at the beginning of the task. If the planner answers "Not enough evidence", then we add "Follow Human" as the first high-level action (so the plan becomes e.g. `[Follow Human, Pick up OBJ, Give OBJ to Human]`).

For the search phase, PROMPTER, in the original work, determines where to look by sampling from the logit values from a text query akin to "Something you find at [MASK] is apple." Instead of sampling directly, we query the LLM, by integrating the latest map's text representation into a search prompt (e.g., "In which room is the shoe likely to be? Please answer in the format of Room X."). This caters to our dataset's complexity, featuring multiple smaller rooms. Since object search happens multiple times, this has the same effect as sampling.[2]

5.5 Experiments and Results

We evaluate the baseline models against the validation and test splits of `SIF`.

Oracle: The oracle baseline replaces the learned planner of REASONER with ground-truth plans like ["Go to Room X", "Grab Obj X"]. Its purpose is to demonstrate the most reasonable path length achievable with optimal reasoning strategies. Further information is in Appendix C.4.1.

Evaluation Metrics: Task Success is determined by whether the object is correctly placed in or on the intended receptacle—or the right room—within 600 timesteps. A clear task in S_{hum} is deemed unsuccessful if it necessitates following the human for over

[2]We grid-search the LLM's hyperparameters — temperature and top_p, by trying temperature $\in \{0.1, 0.3, 0.5\}$ and top_p $\in \{0.1, 0.3, 0.5\}$ in the validation set.

50 timesteps, since unconditional probing should not is a behavior that reduces the utility of the robot. The Success Rate is calculated as the average of individual task successes across the dataset. The SPL (Success weighted by Path Length) [5] is calculated using the formula:

$$SPL = E_{tasks}[s\frac{L^*}{max(L, L^*)}],$$

where s is task success, L is path-length outputted by the model for a task, and L^* is the path given by the oracle baseline. SPL is the primary metric of evaluation, because it tests correct reasoning strategies.

5.5.1 Results

Results from our experiments are presented in Table 5.6. This table notably shows the following facts about our dataset and baselines. First, the gap of model performance (both Reasoner and Prompter) across PnP versus S$_{hum}$, S$_{obj}$ shows that PnP can be solved with commonsense and mechanistic combination, and the rest two tasks cannot. Prompter shows SPL of ∼20% (∼ 60% with oracle perception) on PnP tasks, showing that these tasks are on par with existing tasks like ALFRED. Conversely, on S$_{obj}$ and S$_{hum}$ tasks, it shows much lower performance (∼ 30% with reasoning alone); this shows that these tasks have challenges beyond common sense and progress tracking. The reasoning challenges of S$_{obj}$ and S$_{hum}$ are backed by the performance of Reasoner with oracle perception/manipulation; it shows a stark contrast in PnP tasks (∼ 80%) and S$_{obj}$, S$_{hum}$ tasks (∼ 45%). Overall, Reasoner shows a better performance than Prompter, with and without GT perception/manipulation.

Second, it shows that perception is still a bottleneck, as found in works on previous datasets[98, 97, 23]. Even the oracle baseline, which has perfect reasoning, suffers with learned semantic segmentation and manipulation. When combined with learned reasoning, the drop tends to be larger (oracle perception vs. learned perception of Prompter and Reasoner), since the planner faces a further uncertainty from perception error.

5.5.2 Ablations and Analysis

To analyze the impact of non-reasoning components (visual perception, manipulation) on task performance, Table 5.7 shows the effect of using heuristic manipulation and learned segmentation across our three settings. Our findings align with previous studies [98, 64, 99], further emphasizing that segmentation continues to be a significant obstacle to progress.

Table 5.7: Ablation SR with Oracle plan, for visual and execution errors on Val Seen & Unseen combined.

Method	**PnP**	**S$_{obj}$**	**S$_{hum}$**
G.T. Oracle	99	100	98
+ heuristic man.	88	77	88
+ learned seg.	60	58	77
+ both	49	49	64

The focal point of our study is reasoning. To this end, Table 5.8 analyzes the failure modes of REASONER and PROMPTER when they leverage accurate semantic segmentation and manipulation. This table enumerates the proportions of various reasoning error modes that we observed in unsuccessful tasks. It categorizes these errors as follows: parsing errors, where the format of the LLM's response deviates from expectations; planning errors, which include inadequate tracking of progress and incorrect actions; strategic errors in locating humans, objects, or rooms; and actuation errors, such as mismanaging object interactions. The table lists these errors chronologically, noting that early errors like parsing mistakes can preclude later ones. Notably, strategic errors often manifest as unnecessary repetitive movements—more than five high-level navigational or exploratory actions—or as overly confident yet inaccurate predictions about a human's location.

TABLE 5.8: Reasoning Error Modes. Percentage of failed tasks for each error (w/ oracle perception) on Val Seen & Unseen combined.

Error	REASONER			PROMPTER		
	PnP	**S_{obj}**	**S_{hum}**	**PnP**	**S_{obj}**	**S_{hum}**
Parsing	-	5	-	46	30	-
Planning	20	9	7	-	-	-
Strategy	40	82	88	46	43	100
Obj	-	55	-	-	33	-
Room	40	27	-	46	27	-
Human	-	-	88	-	-	100
Actuation	20	5	5	8	9	-

This table reveals several key insights. First, PNP tasks exhibit fewer strategy errors compared to the other two tasks, which happens primarily due to room grounding issues with less obvious rooms when placing objects—for instance, identifying a "study room." This implies that PNP tasks revolve around more common-sense reasoning, such as inferring room functions from objects and familiar human activities typically associated with those rooms. In contrast, S_{obj} and S_{hum} tasks exhibit higher rates of strategy errors due to their need for a more situated understanding of human motion and activity. Second, PROMPTER's open-loop planning leaves it more susceptible to parsing errors. Notably, PROMPTER often fails to generate a plan for instructions that, while atypical, are feasible (e.g., "Put a shoe on the couch"), responding with "I'm sorry, but I can't comply with that request." In contrast, REASONER exhibits fewer fundamental errors (those that precede strategy errors) and records a higher count of strategy errors. However, this does not imply that REASONER commits more strategic mistakes than PROMPTER; from Table 5.6, it achieves a higher number of successful tasks with correct strategic execution.

Table 5.9 examines model performance on clear versus ambiguous tasks. Ambiguity in S_{obj} tasks emerges when multiple potential locations exist for an object, as indicated by communicative cues (Sec. 5.3.2). For example, the statement "I am washing my face" becomes ambiguous when multiple bathrooms are available. Similarly, ambiguous S_{hum} tasks occur when the human could be in several different locations. Ambiguous S_{obj} tasks require a systematic search of possible locations; common failures include models

TABLE 5.9: **Ambiguous vs Clear tasks**. SPL and SR of REASONER and PROMPTER with G.T./learned vision and manipulation on Val seen & unseen combined.

Model	**Metric**	**G.T. Vis. & Man.**				**Learned Vis. & Man.**			
		S$_{obj}$		**S**$_{hum}$		**S**$_{obj}$		**S**$_{hum}$	
		Clear	Amb.	Clear	Amb.	Clear	Amb.	Clear	Amb.
REASONER	SPL	62	52	13	42	9	11	3	17
	SR	76	71	14	67	15	14	6	26
PROMPTER	SPL	38	29	3	42	11	8	0	17
	SR	54	36	4	66	18	10	0	27

fixating on the incorrect room across multiple attempts or missing a potential room altogether. Clear and ambiguous S$_{hum}$ tasks also demand careful interpretation of the ambiguity of instructions (e.g., deciding whether to follow the human), informed by both contextual clues and observations of the environment and human movement (Fig. 5.3).

For S$_{obj}$ tasks, PROMPTER displays significant performance discrepancies across clear and ambiguous tasks, struggling particularly with ambiguous scenarios. On the other hand, REASONER exhibits a more consistent performance across both conditions, demonstrating its capability to navigate between rooms and conduct systematic searches. In S$_{hum}$ tasks, both models underperform in clear tasks due to a tendency to conservatively judge that there is insufficient evidence of the human's destination, even when only one plausible location exists. PROMPTER notably has an almost zero success rate in clear tasks, consistently concluding insufficient evidence and choosing to follow the human. Conversely, REASONER attempts some calibration but generally leans towards following the human. Qualitative analysis reveals that in ambiguous tasks, REASONER often disengages prematurely, assuming it has accumulated enough evidence.

5.6 Conclusion and Next Steps

We present Situated Instruction Following (`SIF`), a new dataset to evaluate situated and holistic understanding of language instructions. Our dataset reflects aspects of real-world instruction following: (1) ambiguous task specification, (2) evolving intent over time, and (3) dynamic interpretation influenced by agent action. `SIF` is carefully crafted to assess language comprehension and reasoning in situ. We show that current state-of-the-art models struggle with this level of understanding, further highlighting the complexity and uniqueness of our dataset.

In this chapter, we did not delve into methods for training models or LLMs to better manage uncertain states or to seek assistance proactively when faced with problems beyond their resolution capabilities. We address these strategies and propose methodologies to enhance model responsiveness and adaptability in uncertain environments in the next chapter.

Chapter 6

Training LLM Agents to Request Interventions under Budget Constraints

We have outlined a progression from perception to cognition in the previous chapters. Chapters 2 and 3 explored how agents can leverage common sense and physical consistency to navigate perceptual challenges. Chapters 4 and 5 examined cognitive abilities in understanding and executing situated language instructions, revealing that agents struggle particularly when human intent is ambiguous or incomplete. This brings us to the next critical step: the development of agents that are self-aware—capable of understanding their own uncertainties and recognizing their limitations. This level of metacognition is crucial for agents to effectively evaluate their actions and adapt their strategies in complex environments.

Human intelligence involves metacognitive abilities to recognize limitations and seek assistance only when needed. While LLM agents excel in many domains, they often lack this awareness. Overconfident agents risk catastrophic failures, while those that seek help excessively hinder efficiency. In this chapter, we formulate metacognitive help-requesting as a reinforcement learning problem in which we simultaneously optimize both the reward function (the penalty for requesting help) and the help-requesting policy itself. To avoid the cost of repeated retraining, we predict the expected intervention usage of a reward configuration *before* training the help-requesting policy; this enables efficient and autonomous identification of the optimal reward–policy pair, followed by a single pass of helper model training. To balance both robustness and computational efficiency, our method is implemented with integrating LLM-based scoring with tabular reinforcement learning. We empirically find that our method delivers optimal help-requesting behavior on Situated Instruction Following tasks.

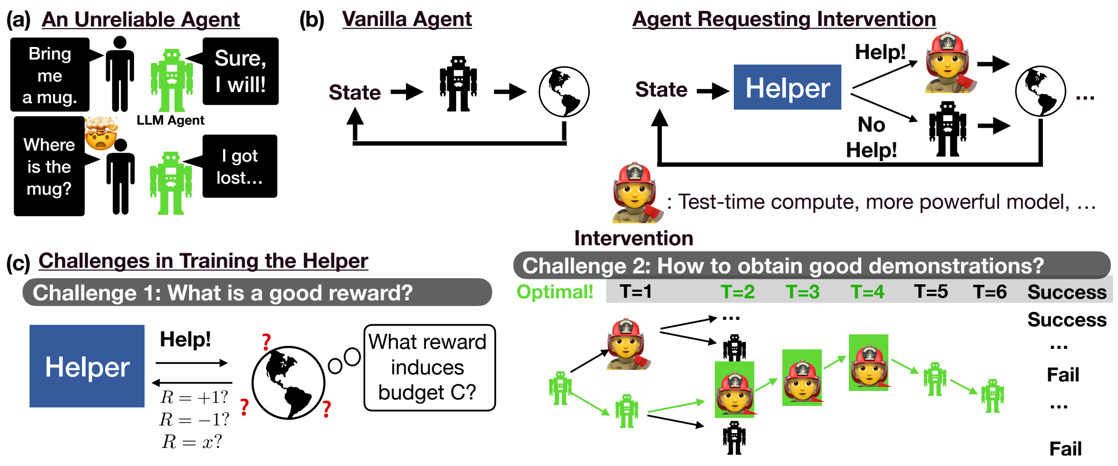


FIGURE 6.1: **Unreliable agents and training challenges. (a)** An unreliable agent does not communicate its inability in advance, causing surprise and catastrophe. **(b)** A "helper", a state-wise classifier, can decide when are optimal timings to request interventions under budget constraints. **(c)** With budget C on interventions, challenge lies in determining a reward function that guides appropriate help-requesting. Concurrently, we have to find the optimal help-requesting policy/demonstration.

6.1 Introduction

Human intelligence is distinguished by metacognitive abilities — particularly the capacity to monitor limitations and requesting targeted assistance only when needed (Fig 6.1). By recognizing and communicating uncertainties, individuals can delegate tasks or seek help before failure becomes inevitable. This approach prevents costly mistakes and fosters trust, as admitting uncertainty and asking for help at the right time is reassuring.

Despite advancements in Large Language Models (LLMs), current AI agents often lack metacognitive awareness. Overconfident agents risk catastrophic errors, while those seeking help excessively are inefficient. Ideally, an AI agent should gauge uncertainty and selectively request assistance, ensuring reliability and efficient use of human effort. While existing AI safety research addresses unintended and malicious behaviors [48, 33, 9], reliability with true agency also requires the ability to recognize and communicate limitations.

A core problem in training an intervention-requesting agent within a limited intervention budget C is determining when to request assistance. Under a reinforcement learning regime, it is unclear how to appropriately penalize the agent for intervention requests; excessively incentivizing help requests prematurely exhausts the budget, while insufficient incentives may lead to avoiding necessary interventions altogether. On the other hand, under a supervised fine-tuning regime, generating annotated trajectories under budget constraints is resource-intensive (Fig. 6.1(c)), as the trajectory space grows exponentially, and even human annotators can struggle to identify optimal intervention timing for each budget constraint. For instance, human annotators may find it challenging to determine precisely when it is most beneficial to use a specific type of test-time compute given a particular budget C.

We introduce a framework for training a “helper” — a state-wise classifier that decides when an agent should invoke a costly intervention under a fixed budget (Fig 6.1 b). We formulate budget-constrained asking as a reinforcement learning problem that involves simultaneous reward and policy search; unlike the conventional RL paradigm, which assumes a fixed reward function and searches only for the policy, our core challenge is *automatically* identifying the appropriate reward (or penalty) that aligns help-seeking behaviors with a given intervention budget C, while *concurrently* optimizing the help-requesting policy under that reward (Fig. 6.1 c, d). A straightforward solution might involve repeatedly setting the penalty, training the help-requesting policy, evaluating whether the policy meets the budget constraints, and readjusting the penalty. However, this iterative retraining is inefficient as it repeatedly incurs the computational overhead of training and evaluating the policy with each new candidate penalty. To address this, we propose predicting the *expected intervention usage* of a reward configuration *before* training the help-requesting policy. By decoupling the evaluation of whether a reward meets the budget constraint from the actual policy training (Fig. 6.3 a, b), the expensive policy training occurs only once—after identifying the optimal reward—rather than repeatedly for each budget adjustment.

Concretely, our method combines tabular reinforcement learning with LLM-based scoring, enhancing robustness while avoiding deep RL inefficiencies (Sec.6.4). Empirical results on Situated Instruction Following tasks show that our approach achieves performance comparable to systems using interventions at every step, yet requires far fewer interventions—often just one per task versus eight (Sec.6.5). We first introduce task settings (Sec. 6.3) before presenting methods and results. By training LLM agents to request assistance judiciously, we advance reliable and efficient deployment of LLM-based systems.

6.2 Related Work

LLM agents Recent breakthroughs in LLM agents [182, 178, 132] have allowed the creation of AI systems which can complete a range of real-world tasks in an open-ended environment [102, 127, 159]. Most work on training LLM agents focuses on SFT for tool-use [127], prompting closed-source LLMs [178, 158, 159], or applying RL in domains with a clear objective such as code generation or math [43, 30]. Instead, we focus on applying RL techniques to an environment with ambiguous instructions [100]. Although previous works target such environments [186, 51], they do not address how to request intervention, which is the central focus of our work.

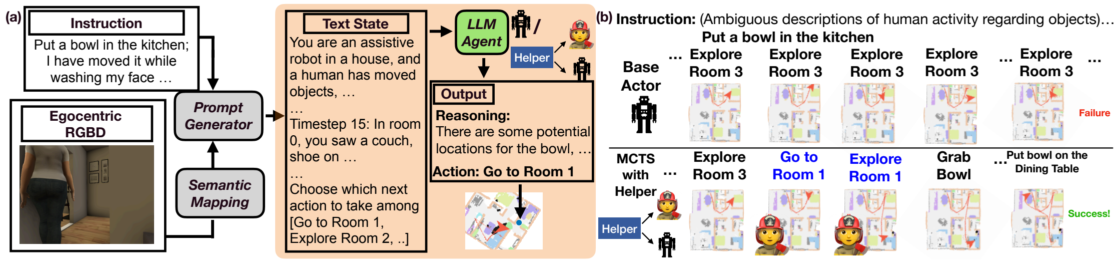

(a)
Instruction
Put a bowl in the kitchen; I have moved it while washing my face …
Egocentric RGBD
Prompt Generator
Semantic Mapping
Text State
You are an assistive robot in a house, and a human has moved objects, …
…
Timestep 15: In room 0, you saw a couch, shoe on …
…
Choose which next action to take among [Go to Room 1, Explore Room 2, ..]
LLM Agent
Helper
Output
Reasoning:
There are some potential locations for the bowl, …
Action: Go to Room 1
(b) Instruction: (Ambiguous descriptions of human activity regarding objects)…
Put a bowl in the kitchen
Base Actor
Explore Room 3
Explore Room 3
Explore Room 3
Explore Room 3
Explore Room 3
Failure
MCTS with Helper
Helper
Explore Room 3
Go to Room 1
Explore Room 1
Grab Bowl
Put bowl on the Dining Table
Success!

FIGURE 6.2: (a) A SIF task requires the agent to locate objects, interact with humans, and perform household tasks in a sequence of discrete actions. Relevant segment is highlighted in orange; states are represented in text. (b) Example task progression with base actor only and helper with base actor/intervention (MCTS); the helper triggered two interventions and salvaged the agent to success.

Safe and Trustworthy AI safety often focuses on *value alignment*—ensuring AI systems follow human values [48, 33, 9]—and *AI security*—ensuring robustness to adversarial attacks [61, 16]. However, these may not guarantee safety in high-stakes contexts, where an agent's limited *capabilities* can lead to harmful failures (e.g., prescribing the wrong medicine [124]). We therefore situate our work under the more expansive concept of *Trustworthy AI* [38], which includes the requirement that agents pursue tasks robustly without unintended failures.

Self-improvement for LLMs Previous work in self-improvement has explored the potential of enhancing LLM responses. Environmental feedback [55, 114, 86, 29, 7] and model-generated critiques [88, 155, 162, 82, 115] have enabled models to perform better in subsequent iterations. Reward models combined with search algorithms further guide decoding toward better answers [102, 148, 185, 172, 80]. However, most such methods assume the model can inherently solve the task, with the challenge lying in eliciting that capability. When a task exceeds the model's ability, intervention of more capable models/augmented compute is needed.

Prompting vs Reinforcement Learning-based Optimization Prompting approaches for uncertainty expression and help-seeking exist [173, 93, 8], and they offer several attractive qualities including simplicity, rapid deployment, and interpretable decision-making processes. An alternative to our reinforcement learning framework might involve heavily prompting large models with explicit budget constraints and help-seeking instructions. Such approaches could potentially leverage the natural language understanding capabilities of large models for flexible help-seeking behaviors. However, prompting-based methods would likely face several challenges in budget-constrained scenarios: (1) they typically lack systematic budget optimization and principled timing strategies, focusing instead on binary confidence estimation or unlimited help-seeking rather than the constrained optimization problem we address; (2) difficulty in precise budget calibration without extensive trial-and-error; (3) potential inconsistency in help-requesting behavior across different contexts; and (4) lack of principled optimization for the exploration-exploitation trade-off inherent in budget-constrained scenarios. Our systematic optimization framework provides more reliable and controllable intervention timing through principled reward and policy search.

Calibration & Meta-cognition Meta-cognitive agents that recognize their own limitations can guide human trust and seek external knowledge to improve accuracy [93, 8]. Previous work estimates confidence via semantic entropy [73], logit values [66, 67], direct questioning [193, 82, 173], or conformal prediction [122]. Although these methods can help decide when to intervene, their estimates are calculated from logits, and may be biased by training data and fail out-of-distribution [170, 193]. Another approach is learning an RL policy that treats assistance seeking as an action [31, 103, 84, 136, 171, 62]. In contrast, we use a process reward model with tabular RL to adapt to budget constraints without additional training.

6.3 Task and Setup

Task We use the Situated Instruction Following (SIF) task [100], which requires finding objects, interacting with humans, and performing household tasks in highly uncertain and dynamic environments (Fig.6.2). To the best of our knowledge, SIF is among the most suitable benchmarks for evaluating how well LLM-driven agents handle nuanced and uncertain instructions. The environment provides textual descriptions at each step, with agents issuing discrete commands like *Go to Room X* in a multi-turn setting.

Tasks are challenging because speaker intent is often ambiguous, and humans may dynamically alter the scene (e.g., placing objects in different rooms), forcing agents to decide when to gather more evidence or clarify instructions. Even advanced models like GPT-4o struggle with these inherent ambiguities[100].

Following SIF, we use two task types. *PnP* is a vanilla pick-and-place scenario. *S_obj* is more challenging, with ambiguous human hints, partial observability, and the possibility of objects being moved. We use 1,000 training tasks and 40 validation/test tasks per split. Also following SIF, we adopt Success Rate (SR) and Success weighted by Path Length (SPL) as our primary evaluation metrics. SR measures the percentage of tasks successfully completed, while SPL additionally rewards efficiency by penalizing long paths to task completion.

State Representation In our framework, a state s consists of a textual description that bundles the agent's observational history (rooms visited, objects seen, human locations) and action history (navigation and manipulation attempts). This text-based state representation enables tractable tabular RL while capturing the essential context needed for intervention decisions. We following the same state formulation as SIF, but we utilize ground truth vision through ground truth semantic segmentation masks, which corresponds to the orange component shown in Figure 6.2(a). This allows us to focus on the core decision-making challenges while abstracting away visual perception complexities.

Base Actors and Interventions The *base actor* (the robot of Fig. 6.2 b) is the *default* decision-making actor—a LLaMA 3B [43] that outputs actions given text states. The base actor operates as the *default* decision-making component, outputting high-level actions (e.g., "Go to Room X", "Grab Object") based on textual state descriptions. It handles the full pipeline from state interpretation to action selection, but with limited reasoning capabilities compared to the interventions.

While the original SIF dataset used GPT-4o, we train LLaMA models on GPT-4o [1] and GPT-4o-mini trajectories for *S_obj* and *PnP* tasks respectively, to save costs for generating rollouts to train our helper. *Interventions* (the firefighter of Fig. 6.2 b) are the costly, powerful aids that the helper can invoke under budget constraints. We consider two types of interventions:

Depth-1 MCTS: A simple Monte Carlo Tree Search approach that, guided by a process reward model, generates up to five candidate actions and selects the best. Details on MCTS implementation are in Appendix D.1.

More Powerful Models: For a GPT-4o-mini base actor, we invoke GPT-4o. For a LLaMA 3B base actor, we use a LLaMA 3B with a better performance, that has been fine-tuned on oracle-agent trajectories for the train tasks.

Helper The *helper* (Fig. 6.2 b) is a LLM-based binary classifier built on a LLaMA 3B with a scalar head. Crucially, this helper is designed to recognize potential failure states in advance—much like a child asking for help to reach candy on a high shelf before attempting and failing. This proactive help-seeking behavior (Fig. 6.1) contrasts with reactive approaches that only request assistance after encountering failures, enabling more efficient and less frustrating human-AI collaboration.

At each step it consumes a textual state—which bundles the agent's observations and action history— as input and outputs a single score indicating whether to request help. Through our reinforcement learning framework, the helper learns to become an evaluator of the base actor's certainty and success likelihood at each state, effectively developing an instinct for when intervention will be most beneficial. The helper can be trained with a standard cross-entropy loss so that its output probability aligns with the optimal help/no-help decisions. Sec.6.4 details our reinforcement learning-based approach to generate optimal state-wise intervention labels for training the helper under varying budget constraints. We formulate this as a combined reward and policy search problem that accounts for how interventions affect future trajectory paths. Throughout this chapter, **"policy" refers exclusively to the helper's intervention-requesting**

strategy. Our method is devoted to training this helper policy under an intervention budget, while the base actors and interventions remain pre-specified and are **not updated during training**.

6.4 Method: Requesting Targeted Interventions

A straightforward approach of the helper (Fig. 6.1 b) is to model state difficulty and threshold it, to request help for the most challenging states within budget C. However, as we later demonstrate (Sec. 6.5), we find simply thresholding on state difficulty largely insufficient for optimal intervention timing. The core challenge is that effective help-requesting requires accounting for how interventions affect future state trajectories—a property not captured by isolated state difficulty scores. To address this, we formulate help-requesting as a combined reward and policy search problem within the reinforcement learning paradigm. In this section, we first discuss our reward regime to ground our discussion within reinforcement learning. Then, we provide our proposed method overview(Sec. 6.4.1) and algorithm (Sec. 6.4.2).

Reward Regime At any non-terminal state s, the help-requesting policy π can choose between requesting help (*help*) or not requesting (*nohelp*). The reward regime that governs π can be given by:

- **Intermediate states:**
  - *help*: incurs an immediate penalty of $-r$
  - *nohelp*: incurs no immediate penalty (0)
- **Terminal states:** success yields a reward of +1 and failure 0

This formulation creates a direct trade-off between help-requesting and performance. The reward (penalty parameter) r indirectly controls the intervention budget—larger values of r discourage intervention requests, resulting in lower usage. Given intervention budget C, the challenge lies in finding the optimal value of r, such thatπ_r^*, the optimal help-requesting policy under r, yields (exactly) the desired budget C.

Notations We define the success probabilities under different intervention choices as:

$$p_{\text{nohelp}}(s) = \Pr(\text{success} \mid \text{state} = s, \text{ action} = \text{nohelp}),$$
$$p_{\text{help}}(s) = \Pr(\text{success} \mid \text{state} = s, \text{ action} = \text{help}).$$

State transition dynamics are denoted as $P_{\text{help}}(s' \mid s)$ and $P_{\text{nohelp}}(s' \mid s)$ for the probability of transitioning from state s to s' when choosing not to help or to help, respectively.

6.4.1 Method Overview

Given the reward regime defined above, our goal is to find both: (1) The optimal reward parameter r that enforces budget constraint C, and (2) The optimal help-requesting policy π_r^* under this reward. A straightforward approach would involve:

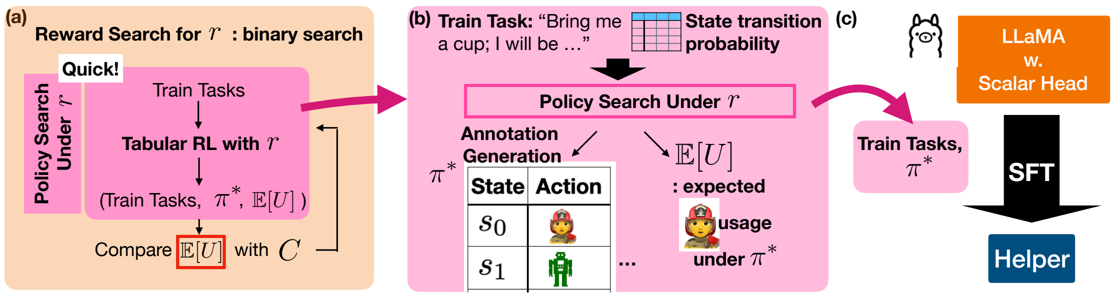


FIGURE 6.3: **Method Overview.** Our method is composed of reward/policy search and one-pass helper training. **(a)**: Reward/policy search is implemented with a *quick* inner loop of policy search that outputs $\mathbb{E}[U]$, expected usage of interventions under optimal policy (π^*) with r. The outer loop (reward search) is binary search that compares $\mathbb{E}[U]$ with the budget constraint C and adjusts r. **(b)**: Policy search generates annotation (optimal tabular policy π^*) on train tasks as well as $\mathbb{E}[U]$. **(c)**: Helper model is trained **once** at the end, with supervised fintuening, using the annotation π^*).

1. Select an initial penalty value r
2. Train a help-requesting policy π_r^* through reinforcement learning
3. Evaluate how many interventions π_r^* uses on a validation set
4. If usage exceeds C, increase r to discourage interventions; if usage is below C, decrease r
5. Repeat steps 2–4 until converging on a policy that meets budget C

This naive approach is computationally prohibitive—each iteration requires extensive data collection, model training, and evaluation. For example, one iteration of training π_r^* can require thousands of environment interactions and rollouts without multiple GPU hours, multiplied by the number of reward adjustments needed to find the correct r.

Our key insight is to decouple the expensive policy training step from the reward parameter (r) search. As illustrated in Fig. 6.3, our method consists of two main components: (1) an efficient reward/policy search procedure and (2) a one-pass helper training stage. For the reward/policy search, we implement a nested loop structure where the inner loop (Fig. 6.3a) performs quick policy search using tabular RL (dynamic programming) to calculate the expected intervention usage $\mathbb{E}[U]$ under the optimal policy π^* for a given r. The outer loop then employs binary search to adjust r by comparing $\mathbb{E}[U]$ against the budget constraint C. During this process, the policy search (Fig. 6.3b) generates annotations representing the optimal tabular policy π^* for each training task. Finally, once the optimal reward parameter r is identified, we train the helper model exactly once (Fig. 6.3c) via supervised fine-tuning on these annotations—whether to call *help* or *nohelp* given state s.

This approach offers significant advantages for automated reward/policy search: (1) the tabular policy calculation is extremely fast (we outline how we make it fast in Sec. 6.4.2), completing in minutes; (2) we can rapidly explore different reward parameters without repeatedly training neural network policies; (3) we only incur the computational cost of training the actual helper model once, after finding the optimal r.

6.4.2 Algorithm

The implementation of our method consists of prerequisites (Phase I), reward/policy search (Phase II), and SFT (Phase III). We explain the algorithmic implementation of each Phase.

Phase I: Prerequisites: Building State Dynamics and Success Probability Cache To enable quick reward/policy search (Phase II) as described in Sec. 6.4.1, we collect the state dynamics $\hat{P}(s' \mid s, a)$ and success probability cache—cached values of $p_{\text{help}}(s)$ and $p_{\text{nohelp}}(s)$.

We collect transitions by randomly triggering interventions using the base actor on training tasks. For each transition, we update the count with $\texttt{count}[s][a][s']{+}{+}$. Specifically, we perform three randomly seeded repetitions of rollouts where interventions are triggered with probabilities of 0.0, 0.1, 0.3, 0.5, 0.7, 0.9, and 1.0 for each task (Fig.6.4 a). For scenarios involving multiple interventions (see Table 6.2), we collect transitions for each intervention individually and additionally for combinations of intervention probabilities, with 0.1/0.1, 0.3/0.3, 0.1/0.3, and 0.3/0.1 per intervention. Finally, raw counts are normalized into transition probabilities, where $a \in \{\text{help}, \text{nohelp}\}$: $\hat{P}(s' \mid s, a) = \texttt{count}[s][a][s'] \,/\, \sum_x \texttt{count}[s][a][x]\,.$

To build the success probability cache, we train two separate LLaMA 3B classifiers (LLaMA 3B with scalar head) to estimate success probabilities under different conditions. For each state s in our collected transitions, we compute and store both $p_{\text{help}}(s)$ and $p_{\text{nohelp}}(s)$ (Fig. 6.4 b). We define these as the probability of successfully completing the task from state s when using intervention (help) or the base actor (nohelp) respectively. To estimate these probabilities, we collect training data by rolling out trajectories from each state s using either the base actor or intervention, giving us state-outcome pairs $(s_i, O(s_i))$, where $O(s_i)$ indicates whether the trajectory starting from state s_i eventually succeeded. Each classifier is trained through supervised finetuning using binary cross-entropy loss:

$$\mathcal{L}(\theta) = -\sum_{i=1}^{N} \left[O(s_i) \ln \hat{p}_\theta(s_i) + (1 - O(s_i)) \ln(1 - \hat{p}_\theta(s_i)) \right],$$

where $O(s_i)$ is the binary outcome label. By caching these values for all states in our transition table $\hat{P}(s' \mid s, a)$, we eliminate any GPU usage during the policy search loop of Phase II, enabling rapid dynamic programming calculations. This pre-computation is crucial for efficiency—it provides immediate access to values needed to determine optimal intervention points without requiring additional rollouts or inference during policy optimization.

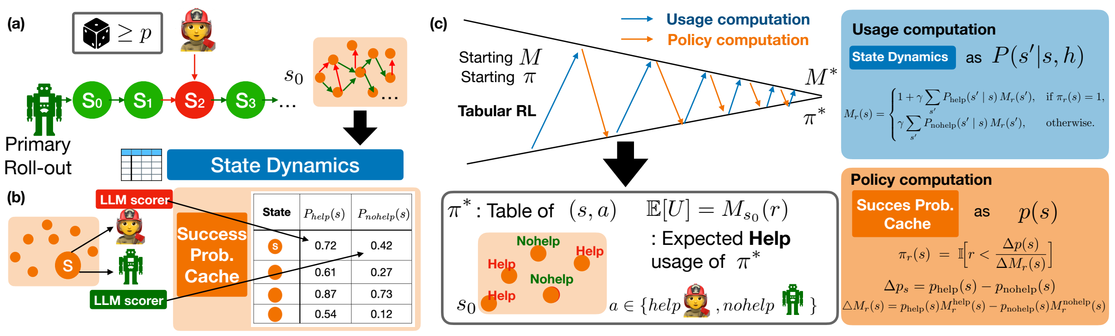


FIGURE 6.4: **Policy search process.** Our method involves: **(a)** Collecting state dynamics by rolling out trajectories with base actors and randomly triggered interventions. **(b)** Caching success probabilities $p_{\text{help}}(s)$ and $p_{\text{nohelp}}(s)$ for each state using separate LLM scorers from intervention and base actor rollouts. **(c)** Performing tabular reinforcement learning through alternating usage and policy computation until convergence, outputting the optimal policy π^* and expected intervention usage $\mathbb{E}[U]$. Right panels show the key mathematical formulations guiding these computations.

Phase II: Offline DP for Iterative Reward/Policy Search In this phase, we efficiently search for **both** the optimal reward parameter r that enforces budget constraint C and the corresponding optimal help-requesting policy π_r^*. We implement this search through a nested loop approach (Fig. 6.3a):

The outer loop performs binary search over possible r values, while the inner loop uses fast offline dynamic programming (DP) to compute the expected intervention usage $\mathbb{E}[U]$ and the optimal policy for each candidate r. This structure allows us to efficiently find the reward parameter that yields exactly the desired budget C.

For the inner loop DP algorithm (Fig. 6.4c), we initialize usage estimates $M_r^{(0)}(s) = 0$ for all states and then iteratively perform:

1. Usage Computation: For each state s, we compute the expected future interventions:

$$M_r^{\text{help}}(s) \leftarrow 1+\sum_{s'} P_{\text{help}}(s' \mid s)\, M_r^{(\texttt{iter}-1)}(s'), \quad M_r^{\text{nohelp}}(s) \leftarrow \sum_{s'} P_{\text{nohelp}}(s' \mid s)\, M_r^{(\texttt{iter}-1)}(s').$$

2. Policy Computation: We calculate the benefit-to-cost ratio of requesting help:

$$\text{ratio} = \frac{\triangle p(s)}{\triangle M_r(s)} = \frac{p_{\text{help}}(s) - p_{\text{nohelp}}(s) \quad \text{(extra success)}}{p_{\text{help}}(s)M_r^{\text{help}}(s) - p_{\text{nohelp}}(s)M_r^{\text{nohelp}}(s) \quad \text{(extra usage)}}$$

and set the policy to request help if the ratio exceeds penalty r:

$$\pi_r(s) \ = \mathbb{I}\Big[r < \frac{\Delta p(s)}{\Delta M_r(s)}\Big], \quad M_r^{(\texttt{iter})}(s) = \begin{cases} M_r^{\text{help}}(s), & \text{if } r < \text{ratio} \\ M_r^{\text{nohelp}}(s), & \text{otherwise.} \end{cases}$$

We repeat these steps until convergence ($|M_r^{(\texttt{iter})}(s) - M_r^{(\texttt{iter}-1)}(s)| < \varepsilon$ for all s), as in Fig. 6.4c. The expected intervention usage of the converged policy π_r^* is given by $\mathbb{E}[U] = M_r(s_0)$. If $\mathbb{E}[U]$ differs from budget C, r is adjusted with binary search in the

Table 6.1: Performance and intervention usage comparison of our method and baselines, across task types. A more powerful model was used as the intervention.

	S_Obj					**Pick N Place**				
	SR ↑	SPL ↑	$L\downarrow$	$U\downarrow$	$\mathbb{E}[U]$	SR ↑	SPL ↑	$L\downarrow$	$U\downarrow$	$\mathbb{E}[U]$
0% Interv.	30.0	26.6	12.3	0.0	-	35.0	30.1	8.0	0.0	-
100% Interv.	67.5	65	7.8	7.8	-	60.0	52.0	4.6	4.6	-
Random										
10%	47.5	38	10.5	1.1	-	37.5	32.9	6.8	0.6	-
30%	50	44.1	8.9	2.9	-	50	42.6	5.8	1.5	-
State-wise Difficulty Thresholding										
10%	42.5	35.5	11.8	1.0	-	32.5	27.1	7.0	0.6	-
30%	42.5	35.5	11.1	1.8	-	57.5	39.3	4.8	3.2	-
Our Method										
r high	47.5	39.3	11.4	0.4	0.4	47.5	36.6	6.0	1.2	0.7
r mid	**62.5**	**57.5**	**9.1**	**1.0**	**0.8**	60.0	49.7	5.7	1.6	0.9
r low	60.0	54.5	8.7	2.2	1.1	**64.9**	**58.9**	**5.2**	**2.9**	**1.8**

outer loop. The entire process is extremely efficient (within minutes, not hours) with computations purely tabular and reusing cached transition and success probabilities from Phase I.

The equations are derived by combining the Bellman equations with our reward regime. The iterative usage/policy computation provably converges to a unique fixed point of optimal intervention policies; see Appendices D.3 and D.4 for the full derivation and convergence proof.

Phase III: Final Policy Training via SFT After deriving tabular π_r^* for all states $s \in \mathcal{S}$ collected in Phase 1, we train the helper model using standard supervised finetuning (SFT). Concretely, the helper model learns to replicate the DP policy π_r^*'s help/nohelp decisions *from the train tasks* for downstream deployment. This helper is plugged in as in Fig. 6.2(b) and is evaluated on the test set.

6.4.3 Extension to Multiple Interventions

Our algorithm extends to multiple intervention types, each with its own budget. The same framework in Sec. 6.4.2 can be applied, with individual budget constraints for intervention types, each with a different cost and expected usage (e.g. r_1, $M_{r_1}(s)$ for *help1* and r_2, $M_{r_2}(s)$ for *help2*). We can adapt Phase II to select the action with minimal combined cost $r_1 M_{r_1}(s) + r_2 M_{r_2}(s)$. Details are in Appendix D.5 and results are in Tab. 6.2.

Table 6.2: Performance and usage comparison across intervention types on S_obj tasks. Our method achieves near-100% intervention performance while using significantly fewer interventions.

	A More Powerful Model					MCTS						Multiple Interventions						
	SR ↑	SPL ↑	$L\downarrow$	$U\downarrow$	$\mathbb{E}[U]$	SR ↑	SPL ↑	$L\downarrow$	$U\downarrow$	$\mathbb{E}[U]$		SR ↑	SPL ↑	$L\downarrow$	$U_1\downarrow$	$U_2\downarrow$	$\mathbb{E}[U_1]$	$\mathbb{E}[U_2]$
0% Interv.	30	27	12.3	0	–	30	27	12.3	0	–	0% Interv.	30	27	12.3	0	–	–	–
100% Interv.	68	65	7.8	7.8	–	63	52	9.4	9.4	–	–	–	–	–	–	–	–	–
Random																		
10%	48	38	10.5	1.1	–	43	37	11.3	1.4	–	10%, 10%	43	33	11.3	1.1	0.8	–	–
30%	50	44	8.9	2.9	–	48	38	11.2	3.7	–	30%, 30%	48	40	9.9	3.2	2.6	–	–
State-wise Difficulty Thresholding																		
10%	43	36	11.8	1.0	–	40	31	12.1	1.1	–	20% & random	40	29	12.4	0.6	0.9	–	–
30%	43	36	11.1	1.8	–	38	30	11.6	2.5	–	50% & random	48	39	10.8	2.0	2.2	–	
Our Method																		
r high	48	39	11.4	0.4	0.4	38	27	12.3	0.5	0.7	r_1 high, r_2 high	38	34	11.5	0.1	1.2	0.3	0.8
r mid	**63**	**58**	**9.1**	**1.0**	**0.8**	45	37	11.6	1.4	1.0	r_1 high, r_2 mid	43	35	10.7	0.1	1.8	0.4	1.0
r low	60	55	8.7	2.2	1.1	**50**	**36**	**10.3**	**3.2**	**1.3**	r_1 mid, r_2 high	**48**	**42**	**9.9**	**2.0**	**0.6**	**1.0**	**0.7**

6.5 Results

We provide results for our method, with the LLaMA base actor and better model/MCTS interventions. We plug in our trained helper from Sec. 6.4, as in Fig. 6.2(b), to call intervention(s) when the helper chooses *help*. We implement two baseline approaches for help-requesting under budget constraints:

Difficulty-based Thresholding This approach models state difficulty and requests help for the most challenging states within budget C. We define the difficulty of a state s as $1 - p(\text{terminal success} \mid s)$, representing the probability of failing to complete the task from state s. To estimate this probability, we leverage the $p_{\text{nohelp}}(s)$ classifier from Phase I of Sec. 6.4.2, which was trained on base actor rollouts only. This is the appropriate classifier since we want to identify states where the base actor struggles most. State difficulty is defined as $1 - p_{\text{nohelp}}(s_i)$, where $p_{\text{nohelp}}(s_i)$ estimates the probability of the base actor successfully completing the task from state s_i. For intervention decisions, we apply a simple thresholding mechanism: we request help at state s_i if $p_{\text{nohelp}}(s_i) < \tau$, with threshold τ calibrated on a held-out validation set to match budget constraint C:

$$\frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\{p_{\text{nohelp}}(s_i) < \tau\} = C.$$

Random Selection As a simple baseline, we randomly select states for intervention with probability set to match the desired budget constraint C. For example, with budget $C = 0.1$ (10% intervention rate), each state has a 10% chance of triggering an intervention independent of state features or trajectory history.

We compare our method to both baselines under equivalent budget constraints.

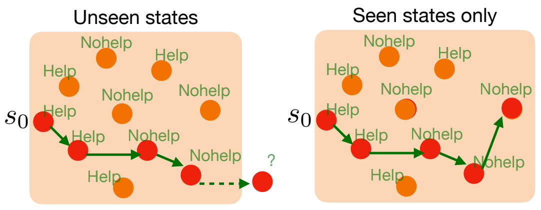


FIGURE 6.5: Seen vs. unseen states in the training data of the helper. The orange region highlights all states collected in Phase I, each labeled with π^* (Nohelp or Help). The green arrow shows rollout from s_0.

TABLE 6.3: Performance and Intervention Usage Comparison (*S_obj* only).

	Unseen		Seen		$\mathbb{E}[U]$	Overall	
	SR	U	SR	U		Train SR	Test SR
Our Method (All States)							
r high	26	0.25	55	0.46	0.49	46	48
r mid	49	0.78	63	0.92	0.43	56	63
r low	52	1.81	60	1.86	0.92	60	60
Trained on Trajectory Only							
r high	16	1.82	62	2.19	0.18	43	43
r mid	29	3.35	60	3.33	0.45	48	50
r low	37	5.86	64	4.92	0.76	54	60

6.5.1 Main Results

Results across tasks Table 6.1 compares our method to baselines in terms of success rate (SR), path-length weighted success (SPL), task execution length (L), observed intervention usage (U), and expected intervention usage ($\mathbb{E}[U]$). We compute $\mathbb{E}[U]$ by averaging $M(s_0)$ under π^*, across starting states s_0 of train tasks. Note that $\mathbb{E}[U]$ is only applicable to our method; it is not straightforward to know this for other methods. We train our approach using different reward scale values (r high, mid, low), inducing varying intervention frequencies.

With just a fraction of the interventions used by a policy that always intervenes (7.8 and 4.5 times on average), our method nearly matches that policy's performance. For example, in *S_obj*, we achieve a 62.5% success rate using only 1.0 intervention on average, outperforming baselines with similar or higher usage. Moreover, $\mathbb{E}[U]$ closely matches observed usage, especially for smaller U (e.g. U is 0.4 and $\mathbb{E}[U]$ is also 0.4). They tend to diverge more with higher r's, but $\mathbb{E}[U]$ still provides good expectations of the model's intervention usage, allowing us to select r based on training data alone, without exhaustive training and evaluation. For the performance drop at *(r) high*, we find this is caused by the base actor encountering out-of-distribution states after interventions—not by any limitation in our helper training method. The base actor, having been trained only on its own trajectories, struggles with the new state distribution created by frequent interventions, producing invalid actions in approximately 75% of these failure cases; see Appendix D.6 for detailed analysis.

Results across interventions Table 6.2 compares our method and baselines on the more challenging *S_obj* split, evaluating three intervention setups: a better model, MCTS, or both. For multiple interventions, we use the Phase 3 extension from Sec. 6.4.3 to assign individual budgets. Consequently, we present results with different (r_1, r_2) configurations, where r_1 controls usage of the better model and r_2 controls MCTS. For baselines for multiple interventions, we randomly select states (10% or 30%) for each intervention, resulting, for example, in 10%,10% and 30%,30%. In the state-wise PRM thresholding baseline, we calibrate thresholds for 20% and 50% of states and trigger each intervention randomly half of the times. For single interventions, we follow the same protocols as in Table 6.1.

In general, the trends from Table 6.1 hold here as well. First, our method optimally calls interventions, whether MCTS or both, achieving higher performance than baselines while using fewer interventions (e.g., with only 0.5 MCTS calls on average, we match the success rate of a 30% thresholding baseline that uses 2.5 calls). Second, using multiple interventions does not yield substantially better results than a single intervention under similar usage constraints, likely due to strategy clashes (See App. D.2.1). Nevertheless, our method still outperforms multi-intervention baselines. Finally, we find that $\mathbb{E}[U]$ remains a reliable predictor of actual usage U, especially at higher r values, providing a useful guide for choosing budgets in advance.

Qualitative example Figure 6.2 (b) shows an *S_obj* task execution comparing the base actor to our helper approach, which applies MCTS-based interventions with $r = 0.5$ (see Table 6.2). The primary challenge is locating a stuffed toy in a cluttered environment, where ambiguous descriptions can point to multiple potential locations. The base actor begins by visiting different rooms but soon becomes stuck, repeatedly exploring Room 3. In contrast, our helper detects this stall point and calls for just two well-timed interventions, enabling a shift toward a more effective strategy and resulting in successful task completion.

Ineffectiveness of thresholding Before discussing full results, we verified that our p_{nohelp} effectively learns state difficulty. Using minimum $p_{\text{nohelp}}(s_i)$ values (values of the most "difficult" states) from trajectories to predict task success/failure, we observed high accuracy (88-90%), precision (70-88%), and recall (83-100%), confirming the premise (see Tab B.1).

However, as we observe in Tab. 6.1, 6.2, when thredholsindg p_{nohelp} for intervention decisions ($p_{\text{nohelp}}(s_i) < \tau$), we found a counterintuitive result: difficulty-based thresholding consistently *underperforms* random intervention selection despite identical budget constraints. While we list detailed analysis of failure modes in App. D.2.1), these results reveal that optimal intervention timing does *not* simply correspond to state difficulty; thresholding does not account for how interventions affect future state trajectories. There exist critical decision points in state trajectories where timely intervention yields greater benefits, even though these states may not register as the "most difficult", and our method successfully finds these points.

6.5.2 Analysis

A key concern of off-line and tabular state collection is coverage and robustness to unseen states. Table 6.3 investigates how different training data selections (selecting different outcomes from the DP process) influence the helper model's performance and intervention usage. We compare two strategies for using the outcomes of the DP (Phase 2) as training data for the helper (Fig. 6.5):

All States – Includes every (s, a) pair of π^*, for all s collected in Phase I (the primary approach in Tables 6.2 and 6.1).

Trajectory Only – Starting from s_0, follow π^* (green arrow of Fig. 6.5) and use the (s, a) pair from this trajectory (the red states only); if π^* encounters an unseen state, do not include this task/trajectory for training the helper.

We then evaluate them under splits of the train set (Fig. 6.5):

Seen tasks – The task terminates following π^*.

Unseen tasks – Unseen state encountered while rolling π^*.

Note that we do not have π^* on val/test sets (since DP searches for trajectories), and this splitting is only applicable to train tasks. Under **Trajectory Only** method, the helper policy struggles in Unseen tasks – showing high intervention usage (U), low success rates (SR), and a large discrepancy between realized and expected usage (e.g., 5.4 vs. 1.14). By contrast, **All States** maintains better alignment between U and $\mathbb{E}[U]$ alongside higher SR. Broader sampling in Phase I could further improve performance for Unseen tasks.

6.6 Conclusion, Limitations, and Broader Impacts

We introduce a framework for LLM agents to request interventions under budget constraints, formulating metacognitive help-requesting as a simultaneous reward and policy search problem. Our key innovation—decoupling reward optimization from policy training—enables efficient identification of optimal intervention timing without repeated retraining. By combining tabular RL with LLM-based scoring, our approach achieves performance comparable to always-intervening systems while using only a fraction of interventions.

Limitations include scaling challenges with tabular RL in large state spaces and resource-intensive data collection (scalability and state coverage). Our tabular approach relies on comprehensive state coverage from base actor rollouts, which becomes challenging in larger state spaces such as software engineering domains. For scalable deployment, two complementary strategies emerge: (1) state abstraction techniques like clustering could reduce the effective state space while preserving the efficiency of our tabular optimization, and (2) traditional deep RL approaches could replace our tabular method, using reward adjustment (r) to iteratively match intervention frequency with budget constraints, though this sacrifices the theoretical guarantees and efficiency of our current approach. While our neural helper training already performs implicit state clustering, explicit clustering during the tabular RL phase would ensure convergence and high-quality annotations for helper training across vastly larger state spaces. Future work could leverage function approximation and state abstraction techniques such as clustering similar states or compact embeddings.

Furthermore, this work assumes uniform intervention costs and known budget constraints. However, real-world deployment would benefit from more nuanced cost models that account for contextual factors—for example, avoiding help requests when humans are sleeping, busy, or stressed. Such adaptive cost functions could be learned from user feedback or inferred from environmental context, extending our framework to more sophisticated utility models that balance task completion with human convenience.

Another key challenge for scaling help-seeking research is the difficulty of building realistic human simulators for training and evaluation. Similar to the sim-to-real gap in robotics, human behavioral modeling presents fundamental challenges. Thus, realistic directions of future work should continue to exlore effective success-based heuristics, learned reward functions, or other training paradigms that can operate effectively without perfect human simulation while still producing helpful and non-intrusive help-seeking behaviors.

Finally, there is the challenge of temporal granularity in real-world deployment. Our simulation-based evaluation abstracts away critical timing considerations essential for real-world collaboration. Physical robot deployment introduces complex temporal dynamics—human response times, travel costs, attention switching overhead, and context-dependent availability—that significantly impact intervention utility. Rather than discrete, instantaneous help at the subgoal level, real systems require fine-grained temporal reasoning about when human assistance becomes available, how long interventions take, and the opportunity costs of interrupting human activities. Future work should incorporate richer cost models that account for these temporal factors, potentially learning dynamic cost functions that adapt to human schedules, current activities, and environmental context. This temporal sophistication is crucial for achieving truly collaborative human-AI systems that respect human cognitive load and workflow patterns.

This work represents a meaningful step toward reliable AI agents that recognize their limitations and judiciously request assistance, enhancing both safety and efficiency in real-world deployments. Beyond potential implications for LLM agent safety and reliability, we have not identified additional societal consequences requiring discussion.

Chapter 7

Conclusions

7.1 Summary

In this thesis, we have developed a comprehensive framework for building situated agents that can effectively operate in real-world environments through the integration of perception, cognition, and metacognition. Our central thesis is that effective embodied intelligence requires more than powerful pretrained models—it demands the ability to operate situatedly, interpreting ambiguous instructions, adapting to dynamic environments, and recognizing one's own limitations. Through six chapters of technical contributions, we have demonstrated how these three dimensions can be addressed through novel computational approaches that move beyond purely data-driven methods.

We began by exploring how agents can leverage common sense and modular architectures to navigate perceptual challenges. In Chapter 2, we introduced FILM, a modular method that processes language instructions into structured forms, maintains semantic maps, and employs semantic search policies to handle incomplete sensory data. This work demonstrated that explicit spatial memory coupled with semantic reasoning significantly outperforms end-to-end approaches, achieving state-of-the-art performance on the ALFRED benchmark.

Building on these perceptual foundations, in Chapter 3, we developed self-supervised methods for adapting visual perception to new environments through location consistency. This approach enables agents to adapt both their visual understanding and navigation policies without expensive labeled data, demonstrating successful sim-to-real transfer and in-situ learning capabilities in real-world settings.

We then turned to cognitive challenges in Chapters 4 and 5. Our analysis of embodied dialogue revealed that agents trained via behavior cloning are particularly vulnerable to replicating errors and struggle with pragmatic grounding. Through the Situated

Instruction Following (SIF) benchmark, we systematically evaluated agents' abilities to handle ambiguous, temporally evolving, and dynamic human intent, revealing that even advanced models like GPT-4 struggle when faced with uncertainty requiring active disambiguation.

Finally, in Chapter 6, we addressed metacognition by developing a framework for training LLM agents to request interventions under budget constraints. By formulating help-requesting as a reinforcement learning problem that simultaneously optimizes reward functions and policies, we achieved systems that match always-intervening performance while using only a fraction of the interventions.

7.2 Takeaways

This thesis encompasses a broad investigation into the fundamental challenges of building agents that can operate effectively in situated, real-world environments. Through our work spanning perception, cognition, and metacognition, we have identified several core principles that can guide future research and development in embodied artificial intelligence. We distill our key insights into three overarching takeaways that capture the essential lessons learned from this comprehensive investigation.

Leveraging pretrained knowledge through modular integration accelerates learning. A central theme throughout this thesis is that connecting pretrained components in a modular fashion provides significant advantages over training from scratch, even though it introduces its own challenges. This principle manifests across multiple domains and scales. In Chapter 2, our modular FILM architecture leverages pretrained BERT for language understanding and pretrained visual models for perception, connecting them through learned semantic reasoning modules. While this approach requires training additional components like the semantic search policy to bridge between modules, it achieves state-of-the-art results by building upon existing knowledge rather than learning everything from scratch.

This advantage of leveraging pretrained knowledge becomes particularly evident in Chapter 4. In Chapter 4, we extend our FILM modular framework to the embodied dialogue setting of the TEACh benchmark. Here, pretrained semantic mapping, dialogue-aware language processing, and semantic search modules combine to disentangle perception, language understanding, and planning into distinct components. This decomposition not only yields robust performance but also surfaces critical failure modes: behavior cloning baselines tend to replicate noisy or misaligned human demonstration errors—such as unnecessary "no-op" interactions—and even our pretrained modules alone fail to ground essential query utterances required for task success.

However, we also acknowledge the challenges this introduces—maintaining consistency between modules during adaptation and determining how to propagate learning signals through the modular structure remain open problems. The comparison between modular and end-to-end approaches reveals a fundamental trade-off: modularity enables leveraging existing knowledge and provides interpretability, but may sacrifice the seamless integration that end-to-end learning can achieve when sufficient data is available.

The practical implications are nuanced. For practitioners building embodied AI systems, starting with pretrained components and connecting them through learned interfaces often provides a faster path to working systems than training from scratch. However, this approach requires careful consideration of how to train the connecting modules (like our semantic search policy) and how to enable end-to-end fine-tuning when needed. Future work should explore hybrid approaches that can leverage the benefits of pretrained modular components while still enabling end-to-end optimization for task-specific performance. This might involve techniques like differentiable module interfaces, careful initialization strategies, or staged training procedures that gradually relax module boundaries.

Design agents for environmental adaptation through self-supervision and contextual learning. A fundamental insight from this thesis is that agents must be designed from the ground up to adapt to their deployment environments rather than assuming that training distributions will generalize effectively. This principle challenges the common paradigm of training once and deploying everywhere, instead advocating for systems that can continuously learn and adapt through interaction with their specific operational contexts.

Our work on location consistency in Chapter 3 exemplifies this principle in action. Rather than relying solely on pre-collected datasets, our approach enables agents to improve their visual understanding through self-supervised learning from their own sensory experiences. The location consistency signal—the observation that objects maintain spatial coherence across viewpoints—provides a rich source of supervisory information that requires no human annotation yet enables significant performance improvements. Critically, this approach proves more robust to domain shift than fully supervised alternatives, as demonstrated by our finding that supervised training on simulation data can actually harm real-world performance by encouraging overfitting to simulation artifacts.

This adaptation principle extends beyond visual perception to language understanding and reasoning. In Chapter 5, we demonstrate that agents must learn to interpret contextual, underspecified instructions through interaction with their specific environments and users. The Situated Instruction Following benchmark reveals that static training approaches, even those using state-of-the-art models like GPT-4, struggle with the dynamic, context-dependent nature of real-world instructions. Effective agents must develop strategies for active disambiguation, environmental exploration, and contextual inference that are tailored to their specific deployment scenarios.

The implications for system design are profound. Rather than viewing adaptation as a post-deployment optimization, our work suggests that adaptation capabilities should be core architectural components. This includes designing reward signals that can be extracted from environmental interaction, developing memory systems that can accumulate and leverage experience over time, and creating learning algorithms that can operate safely and efficiently in live deployment scenarios. For the broader field, this suggests moving away from static benchmarks toward evaluation frameworks that explicitly test adaptation capabilities and environmental sensitivity.

Incorporate metacognitive help-requesting as an integral component for reliable agent deployment. A key contribution of this thesis, detailed in Chapter 6, is formulating a principled reinforcement learning framework that trains agents to judiciously recognize their limitations and proactively request assistance under intervention budget constraints. Unlike conventional approaches that rely on manually tuned confidence thresholds or expect uncertainty estimates to naturally arise during training, our method systematically optimizes both the reward function and the help-requesting policy simultaneously. This structured approach ensures intervention policies align precisely with specific operational constraints, enhancing both reliability and efficiency.

Our technical innovation lies in predicting expected intervention usage before policy training, enabling efficient exploration of trade-offs between autonomy and intervention frequency. This approach allows us to calibrate the agent's help-seeking behaviors to meet practical deployment constraints effectively, demonstrating that agents can achieve near-optimal performance with significantly fewer interventions compared to always-intervening baselines. This resolution of the efficiency-reliability trade-off represents a substantial advancement toward practical, dependable autonomous systems.

Furthermore, this metacognitive capability has broader implications for AI safety and human-AI collaboration. By providing agents with a rigorous mechanism to identify and communicate their uncertainties, we enable more robust and effective interactions where human intervention is leveraged strategically. This capability is especially crucial in domains where oversight is costly or limited, yet catastrophic failures must be mitigated.

For future research, our findings underscore the importance of explicitly incorporating uncertainty quantification, calibrated confidence estimation, and proactive help-requesting into the core design of AI systems intended for complex, real-world applications. The technical frameworks and principles established in this thesis serve as foundational tools for integrating these capabilities across diverse domains, paving the way for safer, more reliable, and more collaborative AI deployments.

7.3 Looking Forward

This thesis establishes foundational techniques for situated intelligence across perception, cognition, and metacognition, yet the path toward truly capable embodied agents remains rich with opportunities and challenges. The techniques we have developed provide important building blocks, but scaling these approaches to the full complexity of real-world deployment will require addressing several critical areas that extend beyond the scope of this thesis. We outline these key research directions and propose concrete steps for advancing the field.

Continual learning for lifelong deployment. Our current paradigm of train-then-deploy is fundamentally limited for real-world applications. Robots operating in homes, factories, or hospitals will encounter new objects, receive updated instructions, and need to adapt to changing environments continuously. The location consistency work in Chapter 3 showed that in-situ adaptation is possible for perception, but extending this to full behavioral adaptation remains an open challenge. Future systems will need to balance multiple objectives: improving on new tasks while maintaining performance on old ones, learning from limited data without overfitting, and ensuring safety during adaptation. This may require new architectures designed for continual learning, such as modular networks that can add new capabilities without interfering with existing ones, or memory systems that can store and retrieve relevant past experiences. Additionally, we need theoretical frameworks for understanding when and how continual learning is possible—which types of distribution shifts can be handled through adaptation versus requiring retraining from scratch.

Scaling situated learning to complex, multi-agent environments Future research should explore hierarchical approaches that can decompose complex multi-agent environments into more manageable subproblems. This might involve learning social models that predict how different agents will behave, developing coordination protocols that enable effective collaboration without explicit communication, or creating meta-learning frameworks that can quickly adapt to new social contexts. Additionally, we need theoretical frameworks for understanding when and how situated learning can remain stable in multi-agent settings, potentially drawing from game theory and multi-agent reinforcement learning.

Furthermore, true agency in AI systems requires more than environmental action-taking—it demands the wisdom to know when to ask questions. While larger language and vision models continue to improve performance on many tasks, feal-world language is inherently underspecified—including formal specifications themselves—but rather than viewing this as a limitation, we should recognize it as an opportunity for meaningful interaction. Consider a simple coding request: ”write a function to sort a list.” While the task outcome seems clear, the language specification leaves critical details unresolved—which sorting algorithm, what data types, how to handle edge cases, performance requirements, or whether to sort in-place. An effective AI coding assistant should recognize these ambiguities and ask clarifying questions rather than making assumptions that may not align with the user's intent. This pattern extends across domains where natural language instructions, no matter how carefully crafted, often contain less information than the task outcomes require.

The most useful AI agents will be those that humans can confidently delegate to while attending to other responsibilities or managing broader objectives. This requires agents that proactively communicate their uncertainties, seek clarification when needed, and demonstrate metacognitive awareness about their own limitations. Just as effective human collaborators know when to work independently and when to consult with teammates, AI systems must develop similar social and metacognitive competencies. Future AI development may benefit more from this kind of interactive intelligence—principled uncertainty quantification, strategic help-seeking, and collaborative communication—than from simply increasing model parameters.

Another critical direction involves extending our metacognitive framework to multi-agent settings. When should an agent seek help from a human versus consulting another AI system? How can agents coordinate their help-seeking behavior to avoid overwhelming human supervisors? These questions require new theoretical foundations and empirical evaluation frameworks that can capture the full complexity of multi-agent situated intelligence.

Developing unified architectures for cross-modal situated reasoning Our thesis treats perception, cognition, and metacognition as distinct levels that must be integrated, but effective situated intelligence likely requires much deeper integration between these components than current approaches achieve. The modular architectures we have developed provide clear benefits in terms of interpretability and targeted improvement, but they may also impose artificial boundaries that limit the emergence of more sophisticated reasoning capabilities.

Consider how perceptual uncertainty should inform cognitive reasoning strategies: when visual perception is uncertain about object identity, language understanding modules should weight alternative interpretations differently and reasoning processes should explore multiple hypotheses rather than committing to single interpretations. Similarly, metacognitive assessments of task difficulty should influence both perceptual attention (focusing sensing resources on task-critical elements) and cognitive processing (allocating more reasoning time to difficult decisions). Current architectures handle these interactions through simple interfaces, but truly effective situated intelligence may require much more intimate coupling between components.

An important design question emerges regarding whether different types of uncertainty—ambiguous instructions, failure risk, personalization needs—require separate handling mechanisms or can be unified under a general framework. Following human cognition patterns, where individuals first sense that "something is wrong" before diagnosing specific issues, AI agents could develop a general success instinct learned from success probabilities and similar heuristics. This overarching uncertainty awareness could then trigger more specific diagnostic processes to identify and address particular uncertainty types. Architecturally, this could be implemented through uncertainty wrappers that preserve existing modular designs (like FILM) while adding metacognitive capabilities, or through end-to-end architectures where such instincts emerge naturally within the learned representations.

A fundamental tension exists between modular architectures that provide clean interfaces and theoretical guarantees, and end-to-end systems that enable seamless information flow but sacrifice interpretability and robustness. Modular approaches excel at propagating explicit uncertainty estimates and maintaining behavioral guarantees, but struggle to share nuanced uncertainty information across API boundaries. Conversely, end-to-end architectures can naturally integrate uncertainty across components but lose the ability to provide systematic guarantees about behavior or failure modes. Future research should explore hybrid architectures that achieve selective permeability—maintaining module boundaries for core functionalities while enabling uncertainty and confidence information to flow freely across components. This might involve differentiable interfaces with uncertainty-aware information passing, or hierarchical architectures where high-level uncertainty coordination operates above well-defined modular components.

The challenge extends to temporal integration as well. Situated agents must maintain coherent beliefs and intentions over extended time horizons while continuously updating their understanding based on new information. This requires memory architectures that can store and retrieve relevant experiences, belief updating mechanisms that can handle contradictory information gracefully, and planning systems that can revise their strategies as situations evolve. Current approaches handle these challenges through separate modules, but more integrated approaches might enable more robust and adaptive behavior.

Building robust evaluation frameworks for situated intelligence One of the most significant limitations revealed by our work is the inadequacy of current evaluation frameworks for assessing situated intelligence capabilities. The benchmarks we have developed, including the Situated Instruction Following dataset, capture important aspects of situated reasoning but remain limited in scope and realism compared to the full complexity of real-world deployment scenarios. The field urgently needs evaluation frameworks that can systematically assess agents' abilities to handle the multifaceted challenges of real-world operation.

Current benchmarks typically focus on task-specific performance metrics that may not reflect transferable capabilities. An agent that achieves high performance on navigation tasks may still fail catastrophically when deployed in environments with different lighting conditions, object arrangements, or social contexts. We need evaluation frameworks that explicitly test adaptation capabilities, robustness to distribution shift, and appropriate help-seeking behavior across diverse domains and conditions.

The evaluation challenge extends beyond individual agent capabilities to system-level properties that emerge from long-term operation. How do agents maintain performance as their environments evolve over weeks or months? How do they handle gradual drift in user preferences or environmental conditions? How do they recover from occasional failures without losing accumulated knowledge? These questions require longitudinal evaluation frameworks that can assess agent behavior over extended time periods under realistic conditions.

Future research should develop standardized evaluation protocols that span multiple dimensions of situated intelligence. This includes creating environments with controllable complexity parameters, developing metrics that capture both task performance and process quality (such as appropriate uncertainty expression and help-seeking), and establishing benchmarks that test transfer capabilities across domains. We also need evaluation frameworks that can handle the inherent subjectivity of many real-world tasks, where success depends on user satisfaction rather than objective task completion.

Additionally, the field needs better methods for evaluating safety and robustness in situated agents. Current approaches often rely on worst-case analysis or adversarial testing, but situated agents must handle the much more subtle challenges of graceful degradation, appropriate conservatism, and effective communication of limitations. Developing evaluation frameworks that can assess these capabilities will be crucial for enabling real-world deployment.

Incorporating social and ethical considerations into situated agent architectures As situated agents become more capable and are deployed in human environments, questions of social appropriateness, privacy, fairness, and ethical behavior become paramount. Our current work focuses primarily on task performance and basic safety considerations, but real-world deployment requires agents that can navigate complex social norms, respect diverse human values, and operate safely in shared spaces where their actions affect multiple stakeholders.

The challenge of social appropriateness is particularly complex because social norms are highly context-dependent, culturally specific, and constantly evolving. An agent that learns appropriate behavior in one social context may violate important norms when deployed in a different setting. Furthermore, social norms often involve implicit understanding and unstated expectations that are difficult to communicate explicitly or learn through observation alone.

Future research should explore how situated learning techniques can be extended to social and ethical domains. This might involve learning social models from multi-modal observation (including verbal and non-verbal cues), developing value alignment techniques that can accommodate diverse and potentially conflicting human preferences, or creating participatory design approaches that involve human stakeholders directly in the agent development process.

The privacy implications of situated agents are also significant and under-explored. Agents that can adapt to their environments necessarily collect and process information about human behavior, preferences, and activities. Ensuring that this information is handled appropriately requires technical solutions for differential privacy, federated learning, and secure multi-party computation, as well as governance frameworks that can balance the benefits of personalization with protection of individual privacy.

Additionally, the deployment of situated agents raises questions about economic impact, technological dependence, and social equity. How do we ensure that the benefits of situated AI are distributed fairly across different communities and economic classes? How do we maintain human agency and decision-making authority in environments increasingly mediated by AI systems? These questions require interdisciplinary collaboration between technologists, ethicists, policymakers, and affected communities.

Advancing theoretical foundations for situated intelligence While this thesis provides empirical demonstrations of situated intelligence capabilities, the field still lacks strong theoretical foundations for understanding when and why these approaches work. Developing such foundations is crucial for moving beyond trial-and-error development toward principled design of situated agent architectures.

Key theoretical questions include: What are the fundamental limits of self-supervised adaptation in situated environments? Under what conditions can situated learning remain stable and avoid catastrophic forgetting? How do we characterize the trade-offs between adaptation speed, robustness, and generalization in situated agents? What theoretical guarantees can we provide about metacognitive calibration and help-seeking behavior?

Addressing these questions will require drawing from multiple disciplines including learning theory, cognitive science, control theory, and information theory. We need theoretical frameworks that can model the dynamics of situated learning, characterize the complexity of different types of environmental adaptation, and provide guidance for designing learning algorithms that can operate safely and effectively in non-stationary environments.

The path toward truly situated intelligence remains challenging and multifaceted, but this thesis provides both theoretical foundations and practical techniques for making meaningful progress. By addressing perception, cognition, and metacognition as interconnected challenges rather than isolated problems, and by maintaining focus on real-world deployment requirements throughout the research process, we can build agents that are not only capable but also safe, interpretable, and appropriate for the complex, dynamic environments where human-AI collaboration will ultimately flourish. The future of embodied AI lies not in building ever-more-powerful individual components, but in understanding how to create systems that can adapt, learn, and grow alongside the humans and environments they serve.

Appendix A

Appendix for Chapter 2

A.1 Task Definition

High and low-level instructions are both available to agents. There are 7 types of tasks (Fig 7. b) and the sequence of subtasks is templated according to the task type.

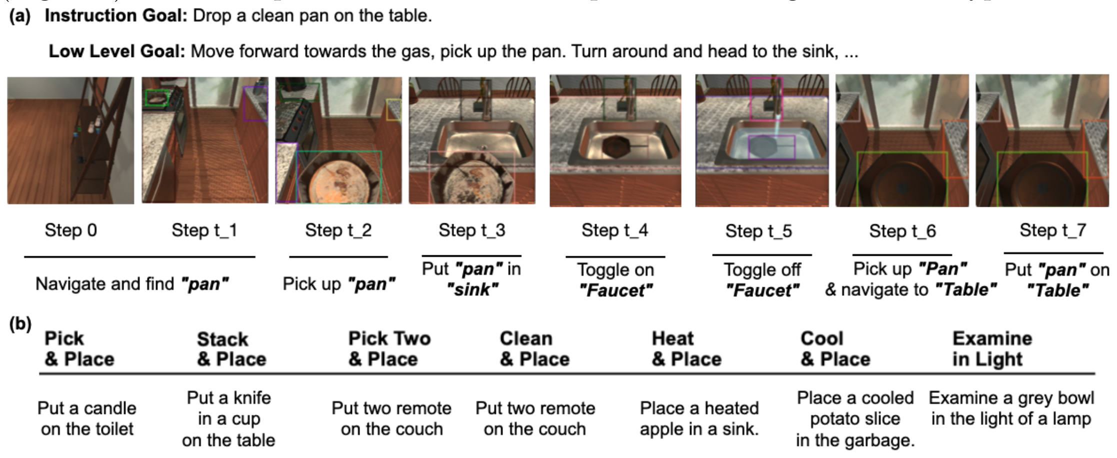

(a) Instruction Goal: Drop a clean pan on the table.
Low Level Goal: Move forward towards the gas, pick up the pan. Turn around and head to the sink, ...
Step 0
Step t_1
Step t_2
Step t_3
Step t_4
Step t_5
Step t_6
Step t_7
Navigate and find "pan"
Pick up "pan"
Put "pan" in "sink"
Toggle on "Faucet"
Toggle off "Faucet"
Pick up "Pan" & navigate to "Table"
Put "pan" on "Table"
(b)
Pick & Place
Stack & Place
Pick Two & Place
Clean & Place
Heat & Place
Cool & Place
Examine in Light
Put a candle on the toilet
Put a knife in a cup on the table
Put two remote on the couch
Put two remote on the couch
Place a heated apple in a sink.
Place a cooled potato slice in the garbage.
Examine a grey bowl in the light of a lamp

Figure A.1: **ALFRED overview.** The goal is given in high level and low level language instructions. For and agent to achieve "success" of the goal, it needs to complete a sequence of interactions (as in the explanations in the bottom of the figure) and the entailed navigation between interactions.

A.2 Semantic Mapping Module

Figure 8 is an illustration of the semantic mapping module. A depth map and instance segmentation is predicted from Egocentric RGB. Then the first and the later are respectively transformed into a point cloud and a semantic label of each point in the cloud, together producing voxels. The voxels are summed across height to produce the semantic map. Partial maps obtained at particular time steps are aggregated to the global map simply via “sum/ logical or.”

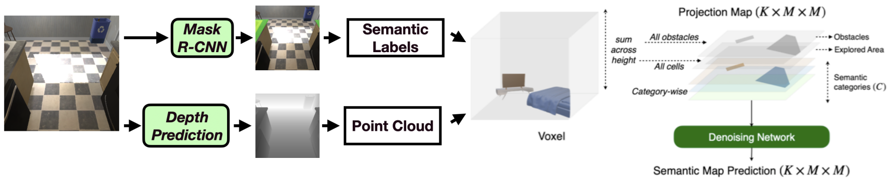


FIGURE A.2: **Semantic mapping module.** Figure was partially taken from chaplot2020object

We dynamically control the number of objects C for efficiency (because there are more than 100 objects in total). All receptacle objects (for input to the semantic policy) and all non-receptacle objects that appear in the subtasks are counted in C. For example, in an episode with the subtask [(Pan, PickUp), (SinkBasin, Put), (Faucet, ToggleOn), (Faucet, ToggleOff), (Pan, PickUp), (Table, Put)], all receptacle objects and ”Pan”, ”Faucet” will be the C objects indicated on the map.

A.3 Semantic Search Policy Module

The map from the previous subsection is passed into 7 layers of convolutional nets, each with kernel size 3 and stride 1. There is maxpooling between any two conv nets, and after the last layer, there is softmax over the 64 (8×8) categories, for each of the C_o (73) channels.

At deployment/ validation, if the agent is currently searching for the cth object, then a search location is sampled from the cth channel of the outputted $8 \times 8 \times C_o$ grid.

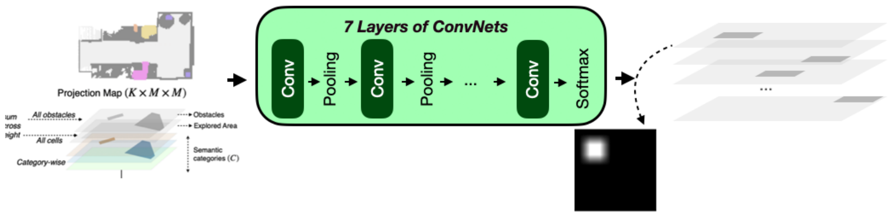


FIGURE A.3: **Semantic search policy.**

A.4 Impact of Grid Size on the Effectiveness of the Semantic Search Policy

While we chose $N = 8$, $\lfloor \frac{M}{N} \rfloor = 30$ for the size of the “coarse” cell of the semantic search policy, the desirable choice of N may be different if a practitioner attempts to transfer

FILM to different scenes/ tasks. While a "too fine" semantic policy will be hard to train due to sparseness of labels, a "too coarse" one will spread the mass of the distribution to widely.

Let us examine the "coarse" and "actual" ground truth distributions just in one direction (e.g. the horizontal direction). Let $F_X(x), C_X(x)$ be the "actual" and "coarse" ground truth CDFs in the horizontal direction. Also, let $L = \lfloor \frac{M}{N} \rfloor$ If the goal object occurs "k" times in the horizontal direction, then,

$$\sup_x |F_X(x) - C_X(x)| \leq \frac{1}{k}(1 - \frac{1}{L}).$$

A similar result holds in the vertical direction. The bound above suggests that if the goal object occurs more frequently (smaller $\frac{1}{k}$), then a coarser L (larger $1 - \frac{1}{L}$) is tolerable. On the other hand, if the goal object occurs very infrequently (larger $\frac{1}{k}$), then a coarse M (larger $1 - \frac{1}{L}$) will result in F_X and C_X becoming too different in the worst case. Thus, it is desirable that practitioners choose L (and in turn, N) based on the frequency of their goal objects, on average. Furthermore, a search policy with adaptive grid sizing should be explored as future work.

A.5 Details on the Deterministic Policy

Following the discussion of Section 4.4, let [(obj_1, $action_1$), ... , (obj_k, $action_k$)] be the list of subtasks, where the current subtask is ($obj_i, action_i$). If obj_i is observed in the current semantic map, the closest obj_i is selected as the goal to navigate; otherwise, the sample from the semantic search policy is chosen as the goal (Section 2.4.3). The agent then navigates towards the closest obj_i via the Fast Marching Method Sethian1591. Once the stop distance is reached, the agent rotates 8 times to the left (at camera horizon 0, 45, 90,...) until obj_i is detected in egocentric vision. Once obj_i is in the current frame, the agents decides to take $action_i$ if two criteria are met: whether obj_i is in the "center" of the frame, or whether the minimum depth towards obj_i is in visibility distance of 1.5 meters). Otherwise, the agent "sidesteps" to keep obj_i in the center frame or continue rotating to the left with horizon 0/45 until obj_i is seen within visibility distance. If the agent executes $action_i$ and fails, the agent "moves backwards" and the map gets updated.

A.6 More Explanations on Table 3

Table 3 shows common error modes and the percentage they take out of all failed episodes, with regards to SR. More specifically, it is showing the distribution of episodes into exactly one error mode, out of the 79.9% of all "Val Unseen" episodes that have failed (the episodes not in the 20.10% of Table 2). The common error modes are failures in (1) locating the subgoal object (due to the small field of view, imperfect segmentation, ineffective exploration), (2) locating the subgoal object because it is in a closed receptacle (cabinet, drawer, etc), (3) interaction (due to object being too far or not in field of view, bad segmentation mask), (4) navigation (collisions), (5) correctly processing language instructions, (6) others, such as the deterministic policy repeating a loop of actions from depth/ segmentation failures and 10 failed actions accruing from a mixture of different errors. These errors occur in the order of (5), (1)/ (2), (3), (4) in an episode, since the LP module operates in the beginning and the object has to be first localized to be interacted with, etc. If an episode ended with errors in multiple categories, it was classified as an example of an "earlier" error in making Table 3. For example, if the language processing module made an error and later there were also 10 collisions, this episode shown as a case of error (5) in Table 3.

A.7 Assignments of Rooms into "Large" and "Small" in Valid Unseen

There are 4 distinct scenes in Valid Unseen (one kitchen scene, one living room, one bed room, one bathroom). The kitchen (Large) has a significantly larger area than all the others (Small).

A.8 Protocols for Reproducing the Semantic Policy

The primary result in Table 1 is from architecture tuning of the language processing, the semantic mapping, and the semantic search policy modules on the development data (validation unseen). Reviewers correctly noted that it is possible random seeds will also effect performance so the model was retrained four additional times and test results are reported here. Since components of the language processing and the semantic mapping module were trained from pre-trained weights, we report the performance of FILM with semantic search policy trained from different seeds.

The improvement by the semantic policy as shown in Table 1 is reproducible across multiple seeds. Table 8 shows results on Tests Unseen with semantic policy trained with different starting seeds (where SEED 1 denotes that the policy was trained with `torch.manual_seed(1)`). With learning rate of 0.001 and evaluation of every 50 steps, the model with the lowest test loss subject to train loss < 0.62 was chosen. The exact code and commands can be found here: `https://github.com/soyeonm/FILM#train-the-semantic-policy`.

Table A.1: Results of FILM reproduced across different starting seeds of the semantic policy. The ± error bar in the Avg. row denotes the sample variance.

Method	**Tests Unseen**			
	PLWGC	GC	PLWSR	**SR**
Low-level + High-level Instructions				
Table 1	15.06	36.37	10.55	26.49
Seed 1	15.12	38.55	11.34	27.86
Seed 2	13.82	36.58	10.13	25.96
Seed 3	10.47	37.12	14.05	25.64
Seed 4	14.22	37.37	10.69	26.62
Avg.	13.74	37.20	11.352	26.51 ± 0.58
High-level Instruction Only				
Table 1	13.13	34.75	9.67	24.46
Seed 1	14.05	36.75	10.47	25.51
Seed 2	12.60	34.59	9.07	23.48
Seed 3	12.86	35.02	9.23	23.68
Seed 4	13.61	36.10	10.10	25.18
Avg.	13.25	35.44	9.71	24.87 ± 0.64

A.9 A Language Processing module without the template assumption

The second paragraph of section 2.4.1 explains the template assumption, with the tasks belonging to one of the 7 types. For direct comparison with existing methods that do not take direct advantage of this assumption, we trained a new Language Processing module that does not make use of templates but makes use of the subtasks sequences annotations ALFRED provides.[1] Fine-tuning a pre-trained BART lewis-etal-2020-bart model, we directly learned a mapping from a high-level instruction to a sequence of subtasks (e.g. "Drop a clean pan on the table" → "(PickupObject, Pan), (PutObject, Sink), ..."). Without any assumption on the structure of the input and the output, this model takes a sequence of tokens as input and outputs a sequence of tokens. With the new LP module, we obtained SR of 18.03% on valid unseen, which is a slight drop compared to our original 20.10%, indicating that templates are only marginally helpful in performance.

For future research, we believe templates should be used instead of subtasks annotations, since they are much cheaper to obtain in naturalistic settings. In this work, we created the 7 templates (one for each type) by writing down an intuitive canonical set of interactions to successfully perform the task. To do so, we looked at just 7 episodes in the training set and spent less than 20 minutes creating them; these cheaply obtained templates cover all 20,000 training episodes. Even to train an agent to perform more complex tasks, it is more realistic to use templates than assume sub-task annotations.

[1]Existing worksblukis2021persistent, abp, hitut, episodictransformer use subtask sequence annotations (or expert trajectories that contain the subtask annotations) as well.

On the other hand, our findings simultaneously suggest the need for a better program synthesis method from instructions to subtask sequences, for general purpose instruction following not bound to certain “types” of instructions.

Appendix B

Appendix for Chapter 4

B.1 More Discussion of Symbiote

Symbiote has a modular structure, which consists of language understanding, mapping, and low-level planning components. It is not trained with imitation learning of low-level demonstrations (e.g. move right, move left, etc.). Demonstrations are used only in the sense that they provide subgoals that suervise the training of the language understanding component.

More specifically, a pretrained T5 model [117] fine-tuned with the ground truth subgoals (`edh_instance['future_subgoals']`), serves as the language understanding component. The model takes the driver and commander's dialogue and previous actions as input; it is trained to output a sequence of subgoals of the form "{action} {obj}", where {action} is either "navigate" or any of the primitive interactions commands "pickup", "cut", "toggle", etc, and {obj} is any of the object classes in ai2thor.

For the mapping component, a DETR detector [18] was finetuned on the train set scenes of TEACh and the depth prediction model from FILM was used off-the-shelf. Frontier based exploration is used for environment exploration. Similarly as in FILM, the agent navigates to object goals in the map using the fast marching method.

B.2 How the Statistics of Section 5 were Obtained

We explain how the statistics that appear in each table of Section 4 were obtained. All analyses, except for TfD results in **Penalizing Agents for Accuracy**, were done on EDH tasks.

Irrelevant Actions The first table shows some representative unnecessary state changes that EDH tasks require for "task success' in evaluation. For example, in our common sense, it is not necessary that we leave the coffee machine on to successfully make coffee (indeed, it is better to turn it off after use). However, since EDH evaluation requires that the agent exactly follows state changes done in the demonstration, the agent will have to leave coffee machine turned on for a particular validation task, if this was done in its corresponding demonstration.

Each row shows unnecessary state changes that are exemplary and the average frequency of these noises across relevant tasks. More specifically,

- **Coffee Machine on/ off**: *'Coffee'* tasks
- **Picked up and not placed**: all tasks
- **Faucet on/ off**: all tasks that may involve using the faucet (*'Coffee', 'Clean All X', 'Boil X','Water Plant','Sandwich', 'Breakfast', 'Plate Of Toast', 'Salad'*)
- **Stove/ Microwave on/off**: all tasks that may involve using a heating appliance (*'Boil X','N Cooked Slices Of X In Y'*)

"Total" accounts for the percentage of EDH tasks that fall into any of the above criteria. Please refer to [109] for the possible types (e.g. *'Coffee'*) of tasks.

While the first table shows statistics of irrelevant state changes of "relevant objects", the second table shows those of more random actions, at a lower level. Navigation No Op, the first kind, was simply obtained by detecting the existence of consecutive Turn Lef/Right x 4, Forward + Backward, Pan Right + Pan Left, Turn Right + Turn Left. The second kind, interaction No Op, was similarly detected. Whether an consecutive and opposite interactions were done on the same "object" was detected by replaying the `pred_actions` in the model outputs. Interaction w. unrelated objects denotes whether the demonstration an object that is completely unrelated from task type (e.g. picking up *saltshaker* for a task whose type is *'Coffee'*). Demonstrations unaligned with dialogue were counted manually since there is no automatic way to filter these.

Penalizing Agents for Accuracy The statistics in this subsection were straightforwardly obtained by averaging over the evaluation outputs (whose formats follow that of the original ET code from TEACh) of each task.

Behavior Cloning with Suboptimal Demonstrations The same procedures for the second table in **Irrelevant Actions** were used.

B.3 TEACh Prefiltering

Only necessary state changes are checked in EDH evaluation, but all are present in training. `https://github.com/alexa/teach#downloading-the-dataset` mentions that the authors filtered the EDH tasks so that "the state changes checked for to evaluate success are only those that contribute towards task success in the main task of the gameplay session the EDH instance is created from." Our analysis is on data that has already been filtered and cleaned and yet still exhibits these problems.

Appendix C

Appendix for Chapter 5

C.1 Task Details

C.1.1 Task Filtering

Tasks that are invalid or trivial are filtered, using the oracle agent (Section 6.5). First, a task is invalid if, for the goal (`[Obj]`) asset, its P_e is not findable or P_t is not reachable. To filter tasks with invalid P_e, we check whether `[Obj]` was detected in the oracle agent's semantic map during the exploration phase; to filter tasks with invalid P_t, we run the oracle agent for task phase, and filter tasks where "Grab Obj" was unsuccessful. Furthermore, a task is trivial if, for the goal (`[Obj]`) asset, one of $< P_e, P_t >$ or $< P_t, P_g >$ are very close or within the same receptacle. This makes the task trivial because there is not much change between exploration and task phase, or between the initial state at task phase and the goal state. We use simulator data to access the parent receptacle of `[Obj]` asset for P_e, P_t, P_g and filter the task if any of these belong to the same parent receptacle.

C.1.2 Details on Language Directives

The complete list of arguments (Sec. 5.3.2) for language directives are shown in Tab.A.1 The full list of [RoomFunctionActivity] is shown in Table A.2.

TABLE A.1: Complete list of arguments (`[ObjectCat]`, `[Recep]`, `[RoomFunction]`) for language directives I and C (Sec. 5.3.2)

`[ObjectCat]`	`[Recep]`	`[RoomFunction]`
basket	chair	living room
book	shelves	bedroom
bowl	bed	kitchen
cup	toilet	dining room
hat	bench	bathroom
plate	bathtub	garage
shoe	couch	empty room
stuffed_toy	counter	dressing room
	table	study room

TABLE A.2: Human Activity, by each room function

Room	Activities
living room	watching TV; hanging out near the couch; vaccuming the living room
bedroom	preparing to sleep; organizing my bed; reading on my bed
kitchen	washing vegetables; preparing my meal; sorting groceries
dining room	setting up the table; eating dinner
bathroom	washing my face; washing my hand; taking a bath; brushing my teeth; shaving
garage	washing my car; fixing my car
empty room	meditating in the empty room; stretching in the exercise room
dressing room	choosing on my outfit; trying on clothes; organizing my clothes
study room	studying; cleaning my desk

C.2 Prompt Examples

C.2.1 Reasoner prompt examples

We show an example prompt for a S$_{obj}$ task (Prompt 1). Furthermore, we show an example ambiguity calibration prompt for a S$_{hum}$ task (Prompt 2).

```
% System Prompt
You are an assistive robot in a house, helping a human. Your observations
    may be incomplete or wrong.

While premapping, you saw the layout of the house;  you saw that Room 0 has
    : chair, shoe on a chair, couch, book on a couch, table; Room 1 has:
    fridge, counter, cabinet; Room 2 has: chair, bed, table. Room 3 has:
    chair, stuffed_toy on a chair, bench, table, stuffed_toy on a table.
    Room 4 has: toilet. Your observations (and the map) can be stale/
    incorrect/ missing, because the human moved objects since you had
    premapped the house, or your vision system is imperfect.

```

```
Then, a little after that, the human said `I took a shoe and moved it with
    me. I put it in the livingroom'. A little after that, the human said `I
    took a book and moved it with me. I put it in the livingroom'. A
    little after that, the human said `I took a stuffed_toy and moved it
    with me. I put it in the livingroom'. A little after that, the human
    said `I put a book in the livingroom'. A little after that, the human
    said `I moved a shoe. I put it in the livingroom'. After all of this,
    the human finally commanded to you: 'Put a book in bedroom.' ”

% Environment Prompt
Timestep 12 : You further explored another location in room 2. Based on the
    preamap and the parts of the rooms that you scanned just now, in map
    12 (at time step 12), you saw that Room 2 has: chair, bed, table; Your
    observations (and the map) can be stale/ incorrect/ missing, because
    the human moved objects since you had premapped the house, or your
    vision system is imperfect.
Timestep 24 : You further explored another location in room 2. Based on the
    preamap and the parts of the rooms that you scanned just now, in map
    24 (at time step 24), you saw that Room 2 has: chair, bed, table; Your
    observations (and the map) can be stale/ incorrect/ missing, because
    the human moved objects since you had premapped the house, or your
    vision system is imperfect.
Timestep 125 : You checked room 0; you scanned some parts of it (and did
    not scan other parts). Based on the preamap and the parts of the rooms
    that you scanned just now, in map 125 (at time step 125), you saw that
    Room 0 has: chair, shoe on a chair, couch, book on a couch, table, shoe
    on a table; Your observations (and the map) can be stale/ incorrect/
    missing, because the human moved objects since you had premapped the
    house, or your vision system is imperfect.

Combining your latest observations, In map 125 (at time step 125), you saw
    that Room 0 has: chair, shoe on a chair, couch, book on a couch, table,
    shoe on a table; Room 1 has: fridge, counter, cabinet, human; Room 2
    has: chair, bed, table. Room 3 has: chair, stuffed_toy on a chair,
    bench, table, stuffed_toy on a table. Room 4 has: toilet. Your
    observations (and the map) can be stale/ incorrect/ missing, because
    the human moved objects since you had premapped the house, or your
    vision system is imperfect.

% Format Prompt
The human said your goal is to ``Put a book in bedroom''. The human may be
    nuanced or ambiguous. To achieve the goal, choose the next action among
    Available Actions: [`Explore Room 0', `Go to Room 1', `Go to Room 2',
    `Go to Room 3', `Go to Room 4', `Pick up shoe', `Pick up book', `Pick
    up shoe', `Done!']' .

You are in room 0 and you can ONLY grab/ put objects inside Room 0. If you
    want to keep searching for object(s) or human that might exist (but you
    have not detected) in the current room, choose `Explore Room 0'. To go
    to another room, choose ``Go to Room 'another room number'''.
```

```
Please ONLY respond in the format: RESPONSE FORMAT: Reasoning: reason
    about the next action. Action: one of Available Actions (e.g.Explore
    Room 0). If task is complete, answer Action: Done!
Now, which action should you take? RESPONSE FORMAT:
```

Listing C.1: **Prompt 1:** Example prompt for Reasoner for a S_{obj} task at timestep 125.

```
% System Prompt
You are an assistive robot in a house, helping a human. Your observations
    may be incomplete or wrong.
While premapping, you saw the layout of the house;  you saw that Room 0 has
    : chair, stuffed_toy on a chair, bed, stuffed_toy on a bed, bench,
    couch, stuffed_toy on a couch, table; Room 1 has: chair, bed, couch,
    table, cabinet; Room 2 has: chair, bathtub, table, cabinet; Room 3 has:
     bed, fridge, cabinet. Room 4 has: chair, bed, fridge, table, cabinet.
    Room 5 has: chair, toilet, plate on a toilet, couch, book on some
    receptacle , bowl on some receptacle . Room 6 has: bed, toilet, fridge,
     cabinet. Room 7 has: toilet, bathtub, cabinet. Your observations (and
    the map) can be stale/ incorrect/ missing, because the human moved
    objects since you had premapped the house, or your vision system is
    imperfect.
Then,  After all of this, the human finally commanded to you: `Bring a shoe
     to me. I will be washing vegetables.'  Right after the human said this
    , you saw the human in room 1, starting to walk towards somewhere. When
     you last saw the human, the human was in room 4, walking towards rooms
     0, 2, 3, 6, 7.

% Environment Prompt
Timestep 2 : You were following the human;the human was in room 1.

Timestep 22 : You were following the human;the human was in room 1.

Timestep 42 : You were following the human;the human was in room 1.

Timestep 62 : You were following the human;the human was in room 1.

Timestep 82 : You were following the human;the human was in room 4, walking
     towards rooms 0, 2, 3, 4, 6, 7.

Timestep 102 : You were following the human;the human was in room 4,
    walking towards rooms 0, 2, 3, 6, 7.

% Format Prompt
```

```
Combining your latest observations, In map 102 (at time step 102), you saw
    that Room 0 has: chair, book on a chair, stuffed_toy on a chair, bed,
    book on a bed, stuffed_toy on a bed, bench, couch, stuffed_toy on a
    couch, table, book on a table; Room 1 has: chair, book on a chair, shoe
     on a chair, stuffed_toy on a chair, bed, shoe on a bed, couch, book on
     a couch, shoe on a couch, stuffed_toy on a couch, table, book on a
    table, shoe on a table, cabinet, book on a cabinet; Room 2 has: chair,
    hat on a chair, bathtub, book on a bathtub, table, cabinet, book on a
    cabinet, hat on a cabinet; Room 3 has: bed, fridge, book on a fridge,
    cabinet, book on a cabinet. Room 4 has: chair, book on a chair, bed,
    book on a bed, bench, fridge, couch, book on a couch, table, book on a
    table, cabinet; Room 5 has: chair, book on a chair, bowl on a chair,
    shoe on a chair, toilet, couch, book on a couch, bowl on a couch,
    stuffed_toy on a couch. Room 6 has: bed, toilet, fridge, book on a
    fridge, cabinet, book on a cabinet. Room 7 has: toilet, book on a
    toilet, bathtub, book on a bathtub, cabinet, book on a cabinet. Your
    observations (and the map) can be stale/ incorrect/ missing, because
    the human moved objects since you had premapped the house, or your
    vision system is imperfect.
The human said your goal is to ``Bring a shoe to me. I will be washing
    vegetables''. The human may be nuanced or ambiguous.Now, from the human
    's utterance (Bring a shoe to me. I will be washing vegetables) and
    your observations of the human (When you last saw the human, the human
    was in room 4, walking towards rooms 0, 2, 3, 6, 7.), do you have
    enough evidence which room (among rooms0, 1, 2, 3, 4, 5, 6, 7 ) the
    human is walking towards and will stop in?Please ONLY respond in the
    format: RESPONSE FORMAT: Reasoning: reason about `Enough Evidence'/ `
    Not Enough Evidence'. Action: choose among [`Enough Evidence', `Not
    Enough Evidence']. RESPONSE FORMAT:
```

LISTING C.2: **Prompt 2:** Example ambiguity calibration prompt for REASONER for a S_{hum} task at timestep 103.

C.2.2 Prompter prompt examples

We show an example planning prompt (Prompt 3) and search prompt (Prompt 4) for a S_{obj} task. Furthermore, we show ambiguity calibration prompt for a S_{hum} task (Prompt 5).

```
The human commanded to you: `Put a hat in bathroom.' Please make a high
    level plan. Please answer in the format of ``[Pick up OBJ, Put in the
   ROOM]''. For example, ``[Pick up saltshaker, Put in the bedroom]''.
```

LISTING C.3: **Prompt 3:** Example planning prompt for PROMPTER for a S_{obj} task at timestep 0.

```
You are an assistive robot in a house, helping a human. Your observations
    may be incomplete or wrong.While premapping, you saw the layout of the
    house; Room 0 has: chair, bench, couch, counter, table, ; Room 1 has:
    chair, bed, table, ; Room 2 has: fridge, counter, cabinet, . Room 3 has
    : chair, bed, . Room 4 has: chair, bed, counter, . Room 6 has: bathtub,
     table, cabinet, . Room 7 has: fridge, . Room 8 has: chair, toilet,
    cabinet. The human had said `I took a shoe and moved it with me. I put
    it in the kitchen'.; `I took a basket and moved it with me. I put it in
     the livingroom'.; `I took a stuffed_toy and moved it with me. I put it
     in the bedroom'.; `I took a basket and moved it with me. I put it in
    the livingroom'.\n In which room, is the basket likely to be in? Please
     answer in the format of Room X. For example, Room 1. Please ONLY
    respond in the format: Answer: Room X.
```

Listing C.4: **Prompt 4:** Example search prompt for Prompter for a S$_{obj}$ task at timestep 154.

```
You are an assistive robot in a house, helping a human. Your observations
    may be incomplete or wrong.While premapping, you saw the layout of the
    house; Room 0 has: shelves, bathtub, fridge, couch, table, ; Room 1
    has: bed, table, ; Room 2 has: shelves, fridge, cabinet, . Room 3 has:
    fridge, . Room 4 has: chair, shelves, table, . Room 5 has: toilet,
    bathtub, . The human commanded to you: `Bring a stuffed_toy to me. I
    will be in study room.' Right after the human said this, you saw the
    human in room None. Now, from the human's utterance (Bring a
    stuffed_toy to me. I will be in study room) and your observations of
    the human, do you have enough evidence which room (among rooms0, 1, 2,
    3, 4, 5 ) the human is walking towards and will stop in? Please choose
    among [`Enough Evidence', `Not Enough Evidence'].
```

Listing C.5: **Prompt 5:** Example ambiguity calibration prompt for Prompter in a S$_{hum}$ task at time step 0.

C.3 Execution Details

C.3.1 Map Update

In the semantic map, position of objects and humans are algorithmically updated. There are channels for the latest position of the human and the entire human trajectory. The latest human position channel is updated to the projection from the current semantic segmentation mask, upon detecting a human. A dedicated human trajectory channel tracks all observed movements; the latest timestep is inputted to the projected area from the semantic segmentation mask of the human. For grabbable objects, updates occur upon Grab Obj/Put Obj actions. When a Grab Obj action is executed and the grasper is closed, the closest target object is removed from the map, reflecting the agent's changed perception. When Put Obj is executed and the object is detected in the agent's view, the projection of the object is put back on its corresponding channel.

C.3.2 Execution Tools

Execution tools for Reasoner/Prompter and their working details/affordance are in Table C.3.

Execution Tool	Description & Affordance
Navigation	
Go to Room X	FMM Planner navigates to a random point in Room X.
Explore Room X	FMM Planner navigates to a random point in Room X; then, agent turns 15 times to the right to look around.
Follow Human	The last observed position of the human is given as the goal, to the human-following wrapper (more explanation is Sec. C.3.3) on top of FMM Planner.
Manipulation	
Grab Obj	The closest object within 2 meters of the grasper is grabbed, and agent's grasper is closed.
Put Obj	Grasped object is put on the closest receptacle within 2 meters of the grasper is grabbed, and agent's grasper is opened.
Give Obj to Human	The agent goes within 1 meter of the human and gives grasped object to human, if human is visible from current view.

Table C.3: Execution tools for Reasoner/Prompter and their working details/affordance.

C.3.3 Following and Anticipating the Human

When the "Follow Human" skill is called (Table C.3), Reasoner and Prompter uses an algorithmic execution tool to follow the human. First, the latest position of the human is set as the goal, and the FMM Planner navigates until agent is within 1m. of the goal. When the goal is reached, the agent rotates 40 times to get a full 360° view, to scan the last location of the human. If the human is visible in a new location, this location is set as the new goal for the FMM Planner. Otherwise, stop is declared; in the prompt, it is stated that "the agent either lost track of the human or the human stopped."

To describe human motion in prompts (illustrated in the bottom row of Fig. 5.3), we use the following process. If the human has been visible for over 30 timesteps, we estimate their expected position by extending a vector from their position 30 timesteps prior to their last observed position, scaled to represent 10 future timesteps. The end point of this vector is the expected human position (the green star in the middle row of Fig. 5.3). We then create a plane orthogonal to this vector, selecting the side opposite to the most recent human position (the depicted white side in the middle row of Fig. 5.3). Rooms covered more than 50% by this white area are included in the "walking towards" section of the prompt (e.g. rooms 1 and 2 at $T = 20$ in Fig. 5.3). If no room meets this criterion, "walking towards" is omitted, and the prompt only mentions the current room with the human (e.g. human seen in room 1 at $T = 60$ in Fig. 5.3).

C.4 Detailed Results

C.4.1 Oracle Baseline

The oracle baseline is established by supplying the reasoner with the ground truth plan and adding exploration as required. For S_{obj} and PnP tasks, the protocol is to activate `explore_room` when the target object is not immediately found. For S_{hum} tasks, we employ a trajectory that avoids following, as this approach is consistently more efficient than those involving following, regardless of whether the tasks are clear or ambiguous. These trajectories are considered the optimal paths for computing the SPL (Success weighted by Path Length). However, there are a small number of instances when the oracle trajectory might fail, as documented in Table 5.6. In these cases, we default to using the maximum permitted task duration of 600 timesteps as the optimal path length for SPL calculation.

C.4.2 Full Results with Success Rate

Results on success rate, that correspond to the SPL results of Table 5.6 are shown in Table D.4 below.

Model		Val Seen			Val Unseen			Test Seen			Test Unseen		
Planning	Perception	**PnP**	$\mathbf{S}_{obj}$	$\mathbf{S}_{hum}$	**PnP**	$\mathbf{S}_{obj}$	$\mathbf{S}_{hum}$	**PnP**	$\mathbf{S}_{obj}$	$\mathbf{S}_{hum}$	**PnP**	$\mathbf{S}_{obj}$	$\mathbf{S}_{hum}$
Oracle	Oracle	98	100	95	100	100	100	98	93	98	95	100	98
	Learned	53	60	68	45	38	60	60	35	75	48	53	53
Prompter[64]	Oracle	68	38	40	68	50	43	63	33	33	53	38	38
	Learned	18	13	18	23	13	15	25	3	15	18	8	10
Reasoner	Oracle	95	78	35	93	68	58	85	80	43	93	60	50
	Learned	23	13	18	33	15	20	38	3	23	23	20	25

TABLE D.4: **SR** performance of PROMPTER and REASONER across splits. In each sectioned-row, the top row assumes oracle perception (semantic segmentation and manipulation); the bottom row assumes learned semantic segmentation and heuristic manipulation.

Appendix D

Appendix for Chapter 6

D.1 MCTS Implementation Details

$$UCT = Q(s,a) + c \times \sqrt{\frac{\ln N(s)}{N(s,a)}}$$

We train a separate LLaMA 3B with a scalar head as a process reward model (PRM) to get $Q(s,a)$ ($p_{\text{help}}(s)$, $p_{\text{nohelp}}(s)$ in Sec. 6.4.2 Phase I takes only s as input) to judge the compatibility of action a with state s. $Q(s,a)$ is the value function of the "goodness" of action a at state s. The training is done with the same procedure as [153]; rollouts are collected and each state s_i is paired with the binary terminal outcome (success/failure) of the rollout, to train cross entropy loss simiarly as in Sec. 6.4.2 Phase I.

The UCT score $c \times \sqrt{\frac{\ln N(s)}{N(s,a)}}$ is calculated from $N(s)$ and $N(s,a)$ that count the number of times state s has been visited and (s,a) has been proposed in the current task.

We propose five actions a from a the base actor but with temperature 1.0 (we use 0.0 without MCTS), and we use $c = 0.25$. In steps where MCTS is not used (in case wehre it is applied to selected states as in Tab. 6.2), we update the chosen $N(s)$ and $N(s,a)$ by multiplying the chosen count by five times.

D.2 Detailed Explanation of Section 5.1

D.2.1 Explanation of "Strategy Clash"

As noted in Section 6.5.1, our experiments reveal an intriguing phenomenon when multiple interventions are available: despite offering diverse strategic approaches, agents tend

Table B.1: **Task outcome prediction performance with $p_{\mathbf{nohelp}}(s)$.**

	GPT 4o-mini		LLaMA	
	Pnp	S_Obj	Pnp	S_Obj
Accuracy	90%	90%	88%	90%
Precision	88%	70%	88%	83%
Recall	100%	100%	100%	83%

to converge on using a single intervention type, and combining multiple interventions does not yield significantly better performance than dedicating resources to a single intervention. We explain the underlying "strategy clash" among interventions that cause this issue.

D.2.1.1 Divergent Exploration Strategies

The core issue stems from fundamental differences in how MCTS and more powerful models approach exploration and decision-making:

MCTS Strategy: MCTS employs a systematic exploration approach, spending considerable time searching within a promising region before the UCT score (exploration benefit) incentivizes exploration of alternative paths. This depth-first tendency means MCTS commits to thoroughly investigating local neighborhoods before broadening its search.

Powerful Model Strategy: In contrast, more powerful models leverage their superior pattern recognition to identify informative clues early in the trajectory. This allows them to make exploratory decisions sooner, often switching between different regions of the state space based on learned heuristics rather than explicit exploration bonuses. These contrasting approaches create situations where each intervention type excels under different circumstances. MCTS performs better in scenarios requiring deep, systematic search within constrained regions, while powerful models excel when early recognition of subtle patterns can guide efficient exploration across broader state spaces.

D.2.1.2 The Single Intervention Convergence

Due to these strategic differences, we observe that agents naturally converge to using predominantly one intervention type rather than mixing them. Table 6.2 demonstrates this empirically: when offered both MCTS and a more powerful model as interventions, the optimal allocation tends to heavily favor one over the other, even when we explicitly encourage diversification through our multi-intervention budget allocation in Phase 3. This convergence occurs because:

1. **Strategic Interference:** Switching between interventions mid-trajectory can disrupt the coherent strategy each intervention type is pursuing. For instance, an MCTS intervention's deep local search may be rendered ineffective if a powerful model intervention subsequently redirects exploration based on different criteria.

2. **Temporal Misalignment:** The optimal timing for each intervention type differs. MCTS benefits from early deployment to build comprehensive search trees, while powerful models can effectively intervene later by quickly recognizing critical decision points. This temporal mismatch makes simultaneous usage suboptimal.

3. **State-Dependent Effectiveness:** Certain trajectory states inherently favor one intervention type. Rather than benefiting from diversity, using multiple interventions often means applying suboptimal strategies to states where a single, well-chosen intervention would suffice.

D.2.2 Explanations of Table 4

Before evaluating our primary intervention rule ($p_{\text{nohelp}}(s) < \tau$), we first verify whether $p_{\text{nohelp}}(s)$ reliably captures state easiness. For this sanity check, we assess if our model can predict task success/failure at test time. Specifically, we identify the minimum $p_{\text{nohelp}}(s_i)$ encountered in each trajectory and set a prediction threshold (separate from our intervention threshold τ) to optimize accuracy in predicting trajectory outcomes. Table B.1 shows that binary classification with minimum $p_{\text{nohelp}}(s_i)$ delivers high accuracy, precision, and recall across both base actors (See Sec. 6.3), confirming that $p_{\text{nohelp}}(s)$ effectively estimates state easiness.

D.2.3 Failure modes of thresholding-based baseline

We analyze two primary failure modes behind the counterintuitive finding, that often the thresholding-based baseline performs worse than random selection under similar budget. The major failure modes are: (1) The thresholding rule often identifies difficult states too late in the trajectory when they're beyond salvaging, even with powerful interventions; (2) The approach suffers from "toggling" behavior - once an intervention moves the agent to an "easier" state, the thresholding rule no longer triggers, allowing the base actor to potentially return to difficult states, creating inefficient oscillation patterns.

We show an example of the toggling behavior. As illustrated in Figure D.1, once an intervention briefly reduces the difficulty (and increases $p_{\text{nohelp}}(s)$), the base actor immediately takes control. Before long, the difficulty surpasses the threshold again, prompting another intervention. This repeated "toggling" between the intervention and the base actor leads to suboptimal performance and illustrates the pitfalls of using only

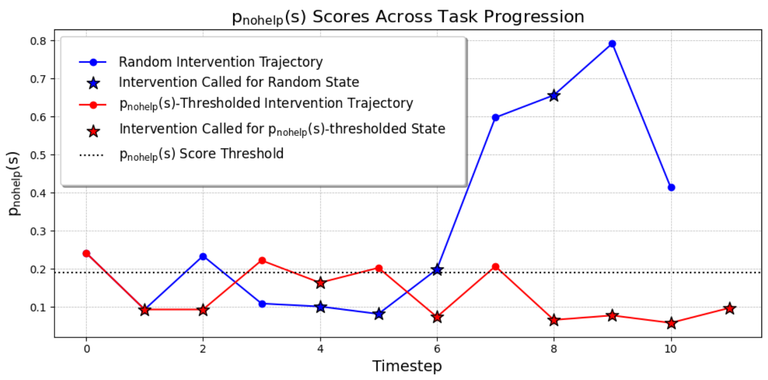


FIGURE D.1: $p_{\text{nohelp}}(s)$ measured by the PRM across the task. Interventions on PRM-chosen states (red line and stars) cause repeated toggling that traps the agent in low-$p_{\text{nohelp}}(s)$ regions, resulting in worse outcomes than random interventions (blue line and stars), which ends at step 10 with task success.

difficulty-based thresholds without accounting for sequential dependencies. Consequently, we *cannot* rely on the same measure $(1 - p_{\text{nohelp}}(s))$ used in self-regulation *without* explicitly incorporating transition dynamics $\big(P_{\text{help}}(s' \mid s), P_{\text{nohelp}}(s' \mid s)\big)$ to identify states that need intervention in this more granular setting.

These findings reveal a fundamental limitation: optimal intervention timing does *not* simply correspond to state difficulty; thresholding does not account for how interventions affect future state trajectories. **There exist critical decision points in state trajectories where timely intervention yields greater benefits, even though these states may not register as the "most difficult".**

D.3 Detailed Derivation of Usage/Policy Iteration

D.3.1 Part I: Decomposing Value Function into Success and Usage

If π is any policy, then the value under π is the expected sum:

$$V_r^\pi(s) \;=\; \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t \big(1_{\text{success at time } t} \;-\; r\, 1_{\text{help at time } t}\big) \;\Big|\; s_0 = s\right].$$

Rewriting,

$$V_r^\pi(s) = \underbrace{\mathbb{E}_\pi\Big[\sum_{t=0}^{\infty} \gamma^t \, 1_{\{\text{success at } t\}}\Big]}_{\text{(discounted successes)}} - r \, \underbrace{\mathbb{E}_\pi\Big[\sum_{t=0}^{\infty} \gamma^t \, 1_{\{\text{help at } t\}}\Big]}_{\text{(discounted helps)}}. \tag{D.1}$$

Hence,

$$V_r^\pi(s) \;=\; S^\pi(s) \;-\; r \, M_r^\pi(s).$$

where we denote

$$S^\pi(s) \;=\; \mathbb{E}_\pi\Big[\sum_{t=0}^{\infty} \gamma^t \, 1_{\text{success at time } t}\Big], \quad M_r^\pi(s) \;=\; \mathbb{E}_\pi\Big[\sum_{t=0}^{\infty} \gamma^t \, 1_{\text{help at time } t}\Big].$$

When taking the max over all policies π, we get $V_r(s) = S^*(s) - r\,M_r(s)$, where $S^*(s)$ and $M_r(s)$ come from the *optimal* policy.

Thus, we can decompose the value function into

$$\underbrace{\text{(expected discounted success)}}_{S(s)} - r \, \underbrace{\text{(expected discounted helps)}}_{M_r(s)},$$

D.3.2 Part II: Arriving at Piecewise Definition of Usage

Now, let's substitute $\boldsymbol{V_r(s) = S(s) - r\,M_r(s)}$ into the Bellman Equation of Section 6.4:

$$V_r(s) = \begin{cases} \gamma \sum_{s'} P_{\text{nohelp}}(s' \mid s)\, V_r(s'), & \text{if nohelp at } s, \\ -r \;+\; \gamma \sum_{s'} P_{\text{help}}(s' \mid s)\, V_r(s'), & \text{if help at } s, \end{cases} \tag{D.2}$$

and

$$V_r(s_{\text{term}}) = \begin{cases} 1, & \text{if task success,} \\ 0, & \text{if task failure.} \end{cases}$$

for terminal states.

As we plug in $\boldsymbol{V_r(s) = S(s) - r\, M_r(s)}$ to each case

1. if nohelp at s

$$S(s) - r\, M_r(s) \;=\; \gamma \sum_{s'} P_{\text{nohelp}}(s' \mid s) \left(S(s') - r\, M_r(s')\right).$$

From the piecewise definition of $S(s)$ (eq. D.5), we have $S(s) \;=\; \gamma \sum_{s'} P_{\text{nohelp}}(s' \mid s)\, S(s'),.$ Thus,

$$-r\, M_r(s) \;=\; -\, r\, \gamma \sum_{s'} P_{\text{nohelp}}(s' \mid s)\, M_r(s').$$

Dividing through by $-r$ (assuming $r > 0$) gives

$$M_r(s) \;=\; \gamma \sum_{s'} P_{\text{nohelp}}(s' \mid s)\, M_r(s'),$$

for the nohelp branch.

2. The help branch.

$$S(s) - r\, M_r(s) \;=\; -r \;+\; \gamma \sum_{s'} P_{\text{help}}(s' \mid s) \left(S(s') - r\, M_r(s')\right).$$

Again using the piecewise definition of eq. D.5 that $\gamma \sum_{s'} P_{\text{help}}(s' \mid s)\; S(s')$, we can isolate the usage terms:

$$-r\, M_r(s) \;=\; -\, r \;-\; r\, \gamma \sum_{s'} P_{\text{help}}(s' \mid s)\, M_r(s'),$$

$$\iff \quad r\left(1 - \gamma \sum_{s'} P_{\text{help}}(s' \mid s)\, M_r(s')\right) \;=\; r\, M_r(s).$$

Dividing through by r and rearranging yields

$$M_r(s) \;=\; 1 \;+\; \gamma \sum_{s'} P_{\text{help}}(s' \mid s)\, M_r(s').$$

Thus, in the help branch, we add 1 for the immediate usage plus the (discounted) future usages under help transitions.

Thus, we get

$$\boxed{M_r(s) = \begin{cases} \gamma \sum_{s'} P_{\text{nohelp}}(s' \mid s)\, M_r(s') & \text{if nohelp,} \\ 1 \;+\; \gamma \sum_{s'} P_{\text{help}}(s' \mid s)\, M_r(s') & \text{if help.} \end{cases}} \quad (D.3)$$

D.3.3 Part III: Arriving at Optimal Policy and Usage

First, let's derive the threshold condition. From the value function Bellman Equation (eq. D.2), *help* is chosen iff:

$$-r \;+\; \gamma \sum_{s'} P_{\text{help}}(s' \mid s)\, V_r(s') > \gamma \sum_{s'} P_{\text{nohelp}}(s' \mid s)\, V_r(s')$$

Rewriting $V_r(s) = S(s) - r\, M_r(s)$ and isolating the cost component $-r$ yields the *threshold* condition:

$$r \;<\; \frac{\Delta p(s)}{\Delta M_r(s)},$$

where we denote

$$\begin{aligned} \Delta p(s) \;&=\; p_{\text{help}}(s) \;-\; p_{\text{nohelp}}(s), \\ \Delta M_r(s) \;&=\; \underbrace{p_{\text{help}}(s)\Big[1 + \gamma \sum_{s'} P_{\text{help}}(s' \mid s)\, M_r(s')\Big]}_{\text{help\_val at } s} \\ &\quad -\; \underbrace{p_{\text{nohelp}}(s)\Big[\gamma \sum_{s'} P_{\text{nohelp}}(s' \mid s)\, M_r(s')\Big]}_{\text{nohelp\_val at } s}. \end{aligned}$$

Intuitively, $\Delta p(s)$ and $\Delta M_r(s)$ capture how much additional *success probability* vs. *usage* we get by choosing help over nohelp at s.

Hence, we arrive at

$$\boxed{\pi_r(s) \;=\; \begin{cases} \texttt{help}, & \text{if } r < \frac{\Delta p(s)}{\Delta M_r(s)}, \\ \texttt{nohelp}, & \text{otherwise.} \end{cases}} \qquad (D.4)$$

Now, combining eq. D.3 and eq. D.4, we get

$$\boxed{\begin{aligned} &M_r(s) = \begin{cases} M_r^{\text{help}}(s) = 1 + \gamma \sum_{s'} P_{\text{help}}(s' | s) M_r(s'), & \text{if } \pi_r^*(s) = \text{help}, \\ M_r^{\text{nohelp}}(s) = \gamma \sum_{s'} P_{\text{nohelp}}(s' | s) M_r(s'), & \text{otherwise.} \end{cases} \\ &\pi_r^*(s) = \text{help} \Longleftrightarrow r < \frac{\Delta p(s)}{\Delta M_r(s)}. \end{aligned}}$$

Lemma: Piecewise Definition of $S(s)$ (Help vs. Nohelp). If $S(s)$ is interpreted as the *discounted probability of eventually reaching success* under a policy π that may choose either help or nohelp, we can write

$$S^\pi(s) \;=\; \mathbb{E}_\pi\Big[\sum_{t=0}^{\infty} \gamma^t \, 1_{\{\text{state at time } t \text{ is success}\}} \;\Big|\; s_0 = s\Big].$$

In an MDP setting with two possible actions, help or nohelp, the policy π dictates which action to take at each state s. Correspondingly, the recursion for $S^\pi(s)$ becomes:

$$S^\pi(s) \;=\; \begin{cases} 1, & \text{if } s \text{ is a terminal success state,} \\ 0, & \text{if } s \text{ is a terminal failure state,} \\ \gamma \sum\limits_{s'} P_{\text{nohelp}}(s' \mid s)\; S^\pi(s'), & \text{if } \pi \text{ chooses } \texttt{nohelp} \text{ at } s, \\ \gamma \sum\limits_{s'} P_{\text{help}}(s' \mid s)\; S^\pi(s'), & \text{if } \pi \text{ chooses } \texttt{help} \text{ at } s. \end{cases} \tag{D.5}$$

- **If s is success:** We set $S^\pi(s) = 1$. This means that if you start in a success state, the probability of "having achieved success" (discounted or not) is exactly 1.
- **If s is nonterminal:** Then there is no immediate success contribution at s itself, and we simply recurse to the next state via either $P_{\text{nohelp}}(\cdot \mid s)$ or $P_{\text{help}}(\cdot \mid s)$, multiplied by the factor γ. Thus, no explicit $\mathbf{1}\{s \text{ is success}\}$ is needed inside the sum, because we have already distinguished the success case in the first line of the piecewise definition.

Thus, under a given policy π, each nonterminal state s follows whichever transition probabilities (nohelp or help) π prescribes at that state. The boundary condition $S^\pi(s_{\text{term}}) = 1$ applies to all terminal success states.

D.4 Proof of Convergence

In Appendix D.3, we showed that the boxed equation

$$\boxed{\begin{aligned} &M_r(s) = \begin{cases} M_r^{\text{help}}(s) = 1 + \gamma \sum_{s'} P_{\text{help}}(s'|s) M_r(s'), & \text{if } \pi_r^*(s) = \text{help}, \\ M_r^{\text{nohelp}}(s) = \gamma \sum_{s'} P_{\text{nohelp}}(s'|s) M_r(s'), & \text{otherwise.} \end{cases} \\ &\pi_r^*(s) = \text{help} \iff r < \frac{\Delta p(s)}{\Delta M_r(s)}. \end{aligned}}$$

is equivalent to the standard Bellman recursion for

$$V_r^\pi(s) = \begin{cases} \gamma \sum_{s'} P_{\text{nohelp}}(s' \mid s)\, V_r^\pi(s'), & \text{if nohelp,} \\ -r \;+\; \gamma \sum_{s'} P_{\text{help}}(s' \mid s)\, V_r^\pi(s'), & \text{if help.} \end{cases}$$

where

$$V_r^\pi(s_{\text{term}}) = \begin{cases} 1, & \text{if task success,} \\ 0, & \text{if task failure.} \end{cases}$$

Because solving value iteration for the above $V_r^\pi(s)$ converges to a unique fixed point $V_r(s)$ and the corresponding policy π_r^*, we know that the iteration of the boxed equation also converges to a unique fixed point $M_r^*(s)$ and π_r^*.

D.5 Details on Extensions to Multiple Interventions

Our algorithm naturally extends to multiple intervention types, as explained in Sec. 6.4.3. We explain the details.

D.5.1 Formulation and Reward Regime

We consider a stochastic process with states $s \in \mathcal{S}$ and transition probabilities $P(s' \mid s)$. At any *non-terminal* state s, the agent may choose from multiple interventions $\{\text{help1}, \text{help2}, \ldots, \text{help}K\}$ or nohelp. Each intervention help_i can improve the probability of success at the cost of incurring usage. Conversely, nohelp avoids usage costs but may have a lower chance of success. We aim to *maximize* the task success rate while keeping the expected discounted number of each intervention (or total usage) below a certain budget C.

Concretely, let $\gamma \in (0, 1)$ be the discount factor. Suppose from an initial state s_0, we want

$$\begin{aligned} &\mathbb{E}\big[(\text{Success})\big] \\ \text{subject to} \quad &\mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t \,\#(\text{helps at time } t)\Big] \;\leq\; C. \end{aligned}$$

One can equivalently encode this via a *cost* $(r_1, r_2, \ldots, r_K)$ for each intervention help$i$, or treat it via a usage-based dynamic programming approach. Below, we use a **reward regime** that translates each help call into a negative reward. This allows standard value-iteration (VI) or usage-based iteration for an MDP with multiple interventions.

Reward Regime. At each non-terminal state s, the agent chooses among:

$$\text{Actions} \;=\; \{\text{nohelp}, \text{help}_1, \ldots, \text{help}_K\}.$$

The immediate reward is:

- help_i: a reward of $-r_i$.
- nohelp: a reward of 0.
- *Terminal States:* success yields a reward of +1, failure yields 0.

Hence, if an agent eventually succeeds, it gains +1 minus the sum of costs $\sum_i r_i$ times the discounted number of times each help_i was used.

Notation for Success Probabilities. When analyzing usage or success, we often use a *probability-of-success* model:

$$p_{\text{nohelp}}(s) \;=\; \Pr(\text{success} \mid s, \text{nohelp}),$$

$$p_{\text{help}_i}(s) \;=\; \Pr(\text{success} \mid s, \text{help}_i).$$

We likewise denote state transition kernels $P_{\text{nohelp}}(s'|s)$ or $P_{\text{help}_i}(s'|s)$ to capture the distribution over next states under each chosen action.

D.5.2 Derivation of Usage/Policy Iteration for Multiple Interventions

Overview. We start from value iteration, as in Sec. **??**. The value function is:

$$\begin{aligned} V(s) = \max\Big\{&0 + \gamma \sum_{s'} P_{\text{nohelp}}(s'|s)\, V(s'), \\ &\{-r_i + \gamma \sum_{s'} P_{\text{help}i}(s'|s)\, V(s')\}_{i=1}^{K}\Big\}. \end{aligned}$$

with $V(s_{\text{success}}) = 1$ and $V(s_{\text{failure}}) = 0$ for terminal states. Now, we derive a *usage-based* DP from this:

$$\begin{aligned} M_{r_i}^{i}(s) = &\text{ expected discounted \# of times we use intervention } i, \\ &\text{starting from } s. \end{aligned}$$

If we pick help_i in state s, then

$$M_{r_i}^{i}(s)(\text{help}_i) = 1 + \gamma \sum_{s'} P_{\text{help}_i}(s'|s)\; M_{r_i}^{i}(s'),$$

$$M^j_{r_j}(s)(\mathrm{help}_i) = 0 + \gamma \sum_{s'} P_{\mathrm{help}_i}(s'|s)\, M^j_{r_j}(s'),$$

for $j \neq i$. If we pick nohelp,

$$M^i_{r_i}(s)(\mathrm{nohelp}) \;=\; \gamma \sum_{s'} P_{\mathrm{nohelp}}(s'|s)\, M^i_{r_i}(s'), \quad \forall i = 1, \ldots, K.$$

Threshold Conditions for Multiple Interventions. In the single-intervention case, we derived ratio $= \dfrac{\Delta\mathrm{success}}{\Delta\mathrm{usage}}$. For multiple interventions, each help$_i$ has:

$$\Delta p^i(s) \;=\; p_{\mathrm{help}_i}(s) \;-\; p_{\mathrm{nohelp}}(s),$$

$$\Delta M^i_{r_i}(s) \;=\; p_{\mathrm{help}_i}(s)\,\big[M^i_{r_i}(s)(\mathrm{help}_i)\big] \;-\; p_{\mathrm{nohelp}}(s)\,\big[M^i_{r_i}(s)(\mathrm{nohelp})\big].$$

We say help$_i$ is cost-effective (vs. nohelp) if

$$\mathrm{ratio}_i(s) \;=\; \frac{\Delta p^i(s)}{\Delta M^i_{r_i}(s)} \;>\; r_i.$$

If $\mathrm{ratio}_i(s) \leq r_i$ for all i, we choose nohelp. If exactly one help$_i$ is cost-effective, we pick help$_i$. If *multiple* helps pass the ratio test, we pick whichever yields the smallest *combined* cost

$$\big(r_1\, M^1_{r_1}(s),\, r_2\, M^2_{r_2}(s),\, \ldots, r_K\, M^K_{r_K}(s)\big) \Longrightarrow \text{minimize} \sum_{i=1}^{K} r_i\, M^i_{r_i}(s)(\mathrm{help}_i).$$

These local decisions define a *policy update* at each state s. Iterating the usage functions $\{M^i_{r_i}(s)\}$ and reselecting among $\{\mathrm{help}_i, \mathrm{nohelp}\}$ converges to a stable fixed point. This final stable policy is π^*. This π^* is exactly the same solution a standard value iteration approach (with reward $\{-r_i\}$) would find, with arguments similar to Appendix D.4.

D.5.3 Algorithm

The algorithm for the multiple intervention setting are in three main phases:

Phase 1: Data Collection and Transition Model.

- Collect transitions offline by running partial-rollouts with $\{\mathrm{help}i, \mathrm{nohelp}\}$ chosen randomly or by partial heuristics.
- Maintain counts $\texttt{count}[s][a][s']$ for each action $a \in \{\mathrm{help}1, \ldots, \mathrm{help}K, \mathrm{nohelp}\}$.
- Estimate $\hat{P}(s'|s,a) = \texttt{count}[s][a][s'] / \sum_x \texttt{count}[s][a][x]$, for $a \in \{\mathrm{nohelp},\ \mathrm{help}_1,\ \ldots, \mathrm{help}_K\}$

Phase 2: Offline Usage/Policy Iteration (Multiple Interventions). First, initialize usage counters $\{M^i_{r_i}(s)\}_{i=1}^K$ to zero (or any guess). Then, Repeat until convergence:

1. *Compute usage for each action*:

$$M^i_{r_i}(s)(\text{help}j) = \begin{cases} 1 + \gamma \sum_{s'} P_{\text{help}j}(s'|s)\, M^i_{r_i}(s'), & \text{if } i = j, \\ \gamma \sum_{s'} P_{\text{help}j}(s'|s)\, M^i_{r_i}(s'), & \text{if } i \neq j, \end{cases}$$

$$M^i_{r_i}(s)(\text{nohelp}) = \gamma \sum_{s'} P_{\text{nohelp}}(s'|s)\, M^i_{r_i}(s').$$

2. *Compute $\Delta p^i(s)$ and $\Delta M^i_{r_i}(s)$*, then check the ratio test $\text{ratio}_i(s) > r_i$.

3. *Policy update*:

$$\pi^*_r(s) = \arg \min_{a \in \{\text{help}1,\ldots,\text{help}K,\text{nohelp}\}} \left\{ \sum_{i=1}^{K} r_i\, M^i_{r_i}(s)(a) \right\}$$
$$\text{subject to} \quad \text{ratio}_i > r_i.$$

4. *Update counters*: $M^i_{r_i}(s) \leftarrow M^i_{r_i}(s)(\pi^*_r(s))$.

5. *Check convergence*: if $\max_{s,i} \left| M^i_{r_i}(s) - \text{old}^i_s \right| < \varepsilon$, stop.

Finally, we output the stable usage counters $\{M^i_{r_i}(s)\}_{i=1}^K$ and the final policy π^*.

Phase 3: Final Policy Representation (SFT or Other).

- We store the final help/nohelp decisions in a table $\pi^*(s)$.
- For states s in the training data, we know exactly which action the usage-based DP prescribes.
- Train the actual "helper" model (e.g. a neural policy or large language model) via *supervised finetuning* to mimic $\pi^*(s)$ on the collected states s.

Relation to Single-Intervention Case. If $K = 1$, the above steps reduce exactly to the single-intervention usage-based iteration. If $r_1 = r_2 = \cdots = r_K = r$, then each help has the same cost, and we can unify them if needed.

D.6 Table 3 Results Detail

We find that performance drops at r_{high} are driven by the base LLaMA actor encountering out-of-distribution (OOD) states, not by limitations of our method. This phenomenon stems from a fundamental training–inference mismatch in our experimental setup.

Training Distribution Mismatch Crucially, the base actor was trained exclusively on rollout trajectories where no interventions occurred (intervention rate = 0). During training data collection, the base actor followed its own decision-making process throughout entire episodes, creating a self-consistent behavioral distribution. However, during evaluation under our intervention framework, the base actor encounters states that arise after intervention actions—states it has never seen during training.

Intervention Frequency and OOD Exposure The intervention penalty parameter r directly controls intervention frequency: r_{low} leads to frequent interventions (high trigger rate), r_{mid} produces moderate intervention frequency, and r_{high} results in rare interventions (low trigger rate). When interventions occur frequently (at r_{low}), the base actor is repeatedly placed into post-intervention states that lie outside its training distribution. These OOD states cause the base actor to produce invalid actions, such as actions not in the allowed action set or contextually inappropriate responses.

Quantitative Impact This training–inference distribution shift explains approximately 75% of failures at r_{low}, where interventions are most frequent and OOD exposure is highest. Conversely, at r_{high}, interventions rarely occur, keeping the base actor largely within its familiar training distribution, resulting in under 5% of failures being attributed to this cause. The intermediate case of r_{mid} shows 15% of failures from this source, consistent with moderate intervention frequency.

Comparison with Other Methods We do not observe these anomalies for PnP tasks or other intervention types (Table 6.2). This is likely because the strategies employed by MCTS or stronger intervention models produce state transitions that remain closer to the base actor's original training distribution, even when interventions occur.

Training vs. Test Task Performance Notably, this behavior does not appear in training tasks, where success rates at r_{high} exceed those at r_{mid} or r_{low} (Tab. 6.3). This supports our hypothesis: on training tasks, the base actor has effectively memorized successful behavior patterns, so even repeated interventions on the same task types keep the resulting states within the actor's familiar distribution. However, on novel test tasks, any deviation from the base actor's self-generated trajectory creates genuinely unfamiliar states. ‘‘‘

Bibliography

[1] Josh Achiam et al. “Gpt-4 technical report”. In: *arXiv preprint arXiv:2303.08774* (2023).

[2] Rishabh Agarwal et al. “Learning to generalize from sparse and underspecified rewards”. In: *International conference on machine learning*. PMLR. 2019, pp. 130–140.

[3] Ziad Al-Halah, Santhosh Kumar Ramakrishnan, and Kristen Grauman. “Zero experience required: Plug & play modular transfer learning for semantic visual navigation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 17031–17041.

[4] Peter Anderson et al. “On Evaluation of Embodied Navigation Agents”. In: *arXiv preprint arXiv:1807.06757* (2018).

[5] Peter Anderson et al. “On evaluation of embodied navigation agents”. In: *arXiv* (2018).

[6] Peter Anderson et al. “Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 3674–3683.

[7] Anonymous. “LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code”. In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: `https://openreview.net/forum?id=chfJJYC3iL`.

[8] Akari Asai et al. “Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection”. In: *The Twelfth International Conference on Learning Representations*. 2024. URL: `https://openreview.net/forum?id=hSyW5go0v8`.

[9] Yuntao Bai et al. “Constitutional ai: Harmlessness from ai feedback”. In: *arXiv preprint arXiv:2212.08073* (2022).

[10] Yonatan Bisk et al. “Experience Grounds Language”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2020.

[11] Valts Blukis et al. “A Persistent Spatial Semantic Representation for High-level Natural Language Instruction Execution”. In: *Proceedings of the Conference on Robot Learning (CoRL).* 2021.

[12] Valts Blukis et al. “Following high-level navigation instructions on a simulated quadcopter with imitation learning”. In: *Robotics: Science and Systems (RSS).* 2018.

[13] Valts Blukis et al. “Mapping navigation instructions to continuous control actions with position-visitation prediction”. In: *Conference on Robot Learning.* PMLR. 2018, pp. 505–518.

[14] Daniel Brown et al. “Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations”. In: *International conference on machine learning.* PMLR. 2019, pp. 783–792.

[15] Tom Brown et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[16] Miles Brundage et al. “The malicious use of artificial intelligence: Forecasting, prevention, and mitigation”. In: *arXiv preprint arXiv:1802.07228* (2018).

[17] Berk Calli et al. “Yale-CMU-Berkeley dataset for robotic manipulation research”. In: *The International Journal of Robotics Research* 36.3 (2017), pp. 261–268.

[18] Nicolas Carion et al. “End-to-end object detection with transformers”. In: *European conference on computer vision.* Springer. 2020, pp. 213–229.

[19] Mathilde Caron et al. “Emerging properties in self-supervised vision transformers”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision.* 2021, pp. 9650–9660.

[20] Khyathi Raghavi Chandu, Yonatan Bisk, and Alan W Black. “Grounding ‘Grounding’ in NLP”. In: *Findings of The 2021 Conference of the Association for Computational Linguistics.* 2021. URL: `https://arxiv.org/abs/2106.02192`.

[21] Angel Chang et al. “Matterport3d: Learning from rgb-d data in indoor environments”. In: *arXiv preprint arXiv:1709.06158* (2017).

[22] Devendra Singh Chaplot et al. “Learning to explore using active neural slam”. In: *arXiv preprint arXiv:2004.05155* (2020).

[23] Devendra Singh Chaplot et al. “Object goal navigation using goal-oriented semantic exploration”. In: *Advances in Neural Information Processing Systems* 33 (2020).

[24] Devendra Singh Chaplot et al. “SEAL: Self-supervised embodied active learning using exploration and 3d consistency”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 13086–13098.

[25] David Chen and Raymond J Mooney. "Learning to interpret natural language navigation instructions from observations". In: *Proceedings of the National Conference on Artificial Intelligence.* 2011.

[26] Jiaqi Chen et al. "MapGPT: Map-Guided Prompting for Unified Vision-and-Language Navigation". In: *arXiv preprint arXiv:2401.07314* (2024).

[27] Liang-Chieh Chen et al. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs". In: *IEEE transactions on pattern analysis and machine intelligence* 40.4 (2017), pp. 834–848.

[28] Ting Chen et al. "A simple framework for contrastive learning of visual representations". In: *International conference on machine learning.* PMLR. 2020, pp. 1597–1607.

[29] Xinyun Chen et al. "Teaching Large Language Models to Self-Debug". In: *The Twelfth International Conference on Learning Representations.* 2024. URL: `https://openreview.net/forum?id=KuPixIqPiq`.

[30] Zixiang Chen et al. "Self-play fine-tuning converts weak language models to strong language models". In: *arXiv preprint arXiv:2401.01335* (2024).

[31] Ta-Chung Chi et al. "Just Ask: An Interactive Learning Framework for Vision and Language Navigation". In: *AAAI Conference on Artificial Intelligence.* 2019. URL: `https://api.semanticscholar.org/CorpusID:208527038`.

[32] Ta-Chung Chi et al. *Just Ask:An Interactive Learning Framework for Vision and Language Navigation.* 2019. DOI: `10.48550/ARXIV.1912.00915`. URL: `https://arxiv.org/abs/1912.00915`.

[33] Paul F Christiano et al. "Deep reinforcement learning from human preferences". In: *Advances in neural information processing systems* 30 (2017).

[34] Jasmine Collins et al. "ABO: Dataset and Benchmarks for Real-World 3D Object Understanding". In: *CVPR* (2022).

[35] Abhishek Das et al. "Embodied question answering". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2018, pp. 1–10.

[36] Matt Deitke et al. *ProcTHOR: Large-Scale Embodied AI Using Procedural Generation.* 2022. DOI: `10.48550/ARXIV.2206.06994`. URL: `https://arxiv.org/abs/2206.06994`.

[37] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[38] Natalia Díaz-Rodríguez et al. "Connecting the dots in trustworthy Artificial Intelligence: From AI principles, ethics, and key requirements to responsible AI systems and regulation". In: *Information Fusion* 99 (2023), p. 101896.

[39] Laura Downs et al. "Google scanned objects: A high-quality dataset of 3d scanned household items". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 2553–2560.

[40] Raphael Druon et al. "Visual object search by learning spatial context". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 1279–1286.

[41] Heming Du, Xin Yu, and Liang Zheng. "VTNet: Visual Transformer Network for Object Goal Navigation". In: *arXiv preprint arXiv:2105.09447* (2021).

[42] Yilun Du, Chuang Gan, and Phillip Isola. *Curious Representation Learning for Embodied Intelligence*. 2021. DOI: `10.48550/ARXIV.2105.01060`. URL: `https://arxiv.org/abs/2105.01060`.

[43] Abhimanyu Dubey et al. "The llama 3 herd of models". In: *arXiv preprint arXiv:2407.21783* (2024).

[44] Boston Dynamics. *Spot - The Agile Mobile Robot.* Available online: `https://bostondynamics.com/products/spot/` (accessed on March 6, 2024). 2023.

[45] Anna Effenberger et al. "Analysis of language change in collaborative instruction following". In: *arXiv preprint arXiv:2109.04452* (2021).

[46] Daniel Fried et al. "Speaker-Follower Models for Vision-and-Language Navigation". In: *Advances in Neural Information Processing Systems*. 2018.

[47] Jorge Fuentes-Pacheco, José Ruiz-Ascencio, and Juan Manuel Rendón-Mancha. "Visual simultaneous localization and mapping: a survey". In: *Artificial intelligence review* 43.1 (2015), pp. 55–81.

[48] Iason Gabriel. "Artificial intelligence, values, and alignment". In: *Minds and machines* 30.3 (2020), pp. 411–437.

[49] Samir Yitzhak Gadre et al. "CLIP on Wheels: Zero-Shot Object Navigation as Object Localization and Exploration". In: *arXiv preprint arXiv:2203.10421* (2022).

[50] Xiaofeng Gao et al. "Dialfred: Dialogue-enabled agents for embodied instruction following". In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 10049–10056.

[51] Jonas Gehring et al. "Rlef: Grounding code llms in execution feedback with reinforcement learning". In: *arXiv preprint arXiv:2410.02089* (2024).

[52] Theophile Gervet et al. "Navigating to Objects in the Real World". In: *arXiv preprint arXiv:2212.00922* (2022).

[53] Asma Ghandeharioun et al. "Approximating Interactive Human Evaluation with Self-Play for Open-Domain Dialog Systems". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: `https://proceedings.neurips.cc/paper/2019/file/fc9812127bf09c7bd29ad6723c683fb5-Paper.pdf`.

[54] Daniel Gordon et al. “Iqa: Visual question answering in interactive environments”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2018, pp. 4089–4098.

[55] Zhibin Gou et al. “CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing”. In: *International Conference on Learning Representations.* 2024.

[56] Jing Gu et al. “Vision-and-language navigation: A survey of tasks, methods, and future directions”. In: *arXiv preprint arXiv:2203.12667* (2022).

[57] Saurabh Gupta et al. “Cognitive mapping and planning for visual navigation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2017, pp. 2616–2625.

[58] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016, pp. 770–778.

[59] Kaiming He et al. “Mask r-cnn”. In: *Proceedings of the IEEE international conference on computer vision.* 2017, pp. 2961–2969.

[60] Kaiming He et al. “Momentum contrast for unsupervised visual representation learning”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.* 2020, pp. 9729–9738.

[61] Dan Hendrycks et al. “Unsolved problems in ml safety”. In: *arXiv preprint arXiv:2109.13916* (2021).

[62] Zhiyuan Hu et al. “Uncertainty of Thoughts: Uncertainty-Aware Planning Enhances Information Seeking in LLMs”. In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems.* 2024. URL: `https://openreview.net/forum?id=CVpuVe1N22`.

[63] Wenlong Huang et al. “Inner monologue: Embodied reasoning through planning with language models”. In: *arXiv preprint arXiv:2207.05608* (2022).

[64] Yuki Inoue and Hiroki Ohashi. “Prompter: Utilizing large language model prompting for a data efficient embodied instruction following”. In: *arXiv preprint arXiv:2211.03267* (2022).

[65] Shahram Izadi et al. “KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera”. In: *Proceedings of the 24th annual ACM symposium on User interface software and technology.* 2011, pp. 559–568.

[66] Zhengbao Jiang et al. “How Can We Know When Language Models Know? On the Calibration of Language Models for Question Answering”. In: *Transactions of the Association for Computational Linguistics* 9 (2021). Ed. by Brian Roark and Ani Nenkova, pp. 962–977. DOI: `10.1162/tacl_a_00407`. URL: `https://aclanthology.org/2021.tacl-1.57/`.

[67] Saurav Kadavath et al. “Language Models (Mostly) Know What They Know”. In: *ArXiv* abs/2207.05221 (2022). URL: `https://api.semanticscholar.org/CorpusID:250451161`.

[68] Liyiming Ke et al. “Tactical rewind: Self-correction via backtracking in vision-and-language navigation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2019, pp. 6741–6749.

[69] Apoorv Khandelwal et al. “Simple but effective: Clip embeddings for embodied ai”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2022, pp. 14829–14838.

[70] Byeonghwi Kim et al. “Agent with the Big Picture: Perceiving Surroundings for Interactive Instruction Following”. In: *Embodied AI Workshop CVPR.* 2021.

[71] Jacob Krantz et al. “Instance-Specific Image Goal Navigation: Training Embodied Agents to Find Object Instances”. In: *arXiv preprint arXiv:2211.15876* (2022).

[72] Alexander Ku et al. “Room-across-room: Multilingual vision-and-language navigation with dense spatiotemporal grounding”. In: *arXiv preprint arXiv:2010.07954* (2020).

[73] Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. “Semantic Uncertainty: Linguistic Invariances for Uncertainty Estimation in Natural Language Generation”. In: *The Eleventh International Conference on Learning Representations.* 2023. URL: `https://openreview.net/forum?id=VD-AYtP0dve`.

[74] Jan Leike et al. “Scalable agent alignment via reward modeling: a research direction”. In: *arXiv preprint arXiv:1811.07871* (2018).

[75] Belinda Z Li et al. “Eliciting human preferences with language models”. In: *arXiv preprint arXiv:2310.11589* (2023).

[76] Chengshu Li et al. “Igibson 2.0: Object-centric simulation for robot learning of everyday household tasks”. In: *arXiv preprint arXiv:2108.03272* (2021).

[77] Shimin Li, Tianxiang Sun, and Xipeng Qiu. “Agent Alignment in Evolving Social Norms”. In: *arXiv preprint arXiv:2401.04620* (2024).

[78] Wenzhao Lian et al. “Benchmarking off-the-shelf solutions to robotic assembly tasks”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE. 2021, pp. 1046–1053.

[79] Jacky Liang et al. “Code as policies: Language model programs for embodied control”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA).* IEEE. 2023, pp. 9493–9500.

[80] Hunter Lightman et al. “Let’s Verify Step by Step”. In: *The Twelfth International Conference on Learning Representations.* 2024. URL: `https://openreview.net/forum?id=v8L0pN6EOi`.

[81] Tsung-Yi Lin et al. “Microsoft coco: Common objects in context”. In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[82] Zhen Lin, Shubhendu Trivedi, and Jimeng Sun. “Generating with Confidence: Uncertainty Quantification for Black-box Large Language Models”. In: *Transactions on Machine Learning Research* (2024). ISSN: 2835-8856. URL: `https://openreview.net/forum?id=DWkJCSxKU5`.

[83] Bo Liu, Xuesu Xiao, and Peter Stone. *A Lifelong Learning Approach to Mobile Robot Navigation*. 2021. arXiv: `2007.14486 [cs.RO]`.

[84] Iou-Jen Liu et al. “Asking for knowledge (afk): Training rl agents to query external knowledge using language”. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 14073–14093.

[85] Jijia Liu et al. *LLM-Powered Hierarchical Language Agent for Real-time Human-AI Coordination*. 2024. arXiv: `2312.15224 [cs.AI]`.

[86] Xiao Liu et al. “AgentBench: Evaluating LLMs as Agents”. In: *The Twelfth International Conference on Learning Representations*. 2024. URL: `https://openreview.net/forum?id=zAdUB0aCTQ`.

[87] Chih-Yao Ma et al. “The regretful agent: Heuristic-aided navigation through progress estimation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 6732–6740.

[88] Aman Madaan et al. “Self-Refine: Iterative Refinement with Self-Feedback”. In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023. URL: `https://openreview.net/forum?id=S37hOerQLB`.

[89] Aman Madaan et al. “Self-refine: Iterative refinement with self-feedback”. In: *Advances in Neural Information Processing Systems* 36 (2024).

[90] Arjun Majumdar et al. “Zson: Zero-shot object-goal navigation using multimodal goal embeddings”. In: *arXiv preprint arXiv:2206.12403* (2022).

[91] Oleksandr Maksymets et al. “THDA: Treasure hunt data augmentation for semantic navigation”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 15374–15383.

[92] Shehryar Malik et al. “Inverse constrained reinforcement learning”. In: *International conference on machine learning*. PMLR. 2021, pp. 7390–7399.

[93] Alex Mallen et al. “When Not to Trust Language Models: Investigating Effectiveness of Parametric and Non-Parametric Memories”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 9802–9822. DOI: `10.18653/v1/2023.acl-long.546`. URL: `https://aclanthology.org/2023.acl-long.546/`.

[94] Cynthia Matuszek et al. “Learning to Parse Natural Language Commands to a Robot Control System”. In: *Proc. of the 13th International Symposium on Experimental Robotics (ISER)*. Québec City, Quebec, Canada, 2012.

[95] John McCarthy et al. *Situations, actions, and causal laws.* Comtex Scientific, 1963.

[96] Lina Mezghan et al. “Memory-augmented reinforcement learning for image-goal navigation”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2022, pp. 3316–3323.

[97] So Yeon Min et al. “Don’t Copy the Teacher: Data and Model Challenges in Embodied Dialogue”. In: *arXiv preprint arXiv:2210.04443* (2022).

[98] So Yeon Min et al. “Film: Following instructions in language with modular methods”. In: *arXiv preprint arXiv:2110.07342* (2021).

[99] So Yeon Min et al. “Self-Supervised Object Goal Navigation with In-Situ Fine-tuning”. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2023, pp. 7119–7126.

[100] So Yeon Min et al. “Situated instruction following”. In: *European Conference on Computer Vision*. Springer. 2025, pp. 202–228.

[101] Arsalan Mousavian et al. “Visual representations for semantic target driven navigation”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8846–8852.

[102] Reiichiro Nakano et al. “Webgpt: Browser-assisted question-answering with human feedback”. In: *arXiv preprint arXiv:2112.09332* (2021).

[103] Khanh Nguyen, Yonatan Bisk, and Hal Daum’e. “Learning When and What to Ask: a Hierarchical Reinforcement Learning Framework”. In: *ArXiv* abs/2110.08258 (2021). URL: `https://api.semanticscholar.org/CorpusID:239016703`.

[104] Khanh Nguyen and Hal Daumé III. “Help, Anna! Visual Navigation with Natural Multimodal Assistance via Retrospective Curiosity-Encouraging Imitation Learning”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 684–695. DOI: `10.18653/v1/D19-1063`. URL: `https://aclanthology.org/D19-1063`.

[105] Van-Quang Nguyen, Masanori Suganuma, and Takayuki Okatani. “Look Wide and Interpret Twice: Improving Performance on Interactive Instruction-following Tasks”. In: *arXiv preprint arXiv:2106.00596* (2021).

[106] Kolby Nottingham et al. *LAV*. 2021. URL: `https://leaderboard.allenai.org/alfred/submission/c2cm7eranqs9puf9uvjg`.

[107] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. "Representation learning with contrastive predictive coding". In: *arXiv preprint arXiv:1807.03748* (2018).

[108] Long Ouyang et al. "Training language models to follow instructions with human feedback". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 27730–27744.

[109] Aishwarya Padmakumar et al. *TEACh: Task-driven Embodied Agents that Chat.* 2021. URL: `https://arxiv.org/abs/2110.00534`.

[110] Alexander Pashevich, Cordelia Schmid, and Chen Sun. "Episodic Transformer for Vision-and-Language Navigation". In: *arXiv preprint arXiv:2105.06453* (2021).

[111] Mihir Prabhudesai et al. "Embodied language grounding with 3d visual feature representations". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2020, pp. 2220–2229.

[112] Xavier Puig et al. "Habitat 3.0: A co-habitat for humans, avatars and robots". In: *arXiv preprint arXiv:2310.13724* (2023).

[113] Yuankai Qi et al. "Reverie: Remote embodied visual referring expression in real indoor environments". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2020, pp. 9982–9991.

[114] Shuofei Qiao et al. "Making Language Models Better Tool Learners with Execution Feedback". In: *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers).* Ed. by Kevin Duh, Helena Gomez, and Steven Bethard. Mexico City, Mexico: Association for Computational Linguistics, June 2024, pp. 3550–3568. DOI: `10.18653/v1/2024.naacl-long.195`. URL: `https://aclanthology.org/2024.naacl-long.195/`.

[115] Yuxiao Qu et al. "Recursive introspection: Teaching language model agents how to self-improve". In: *Advances in Neural Information Processing Systems* 37 (2024), pp. 55249–55285.

[116] Alec Radford et al. "Learning transferable visual models from natural language supervision". In: *International Conference on Machine Learning.* PMLR. 2021, pp. 8748–8763.

[117] Colin Raffel et al. "Exploring the limits of transfer learning with a unified text-to-text transformer." In: *J. Mach. Learn. Res.* 21.140 (2020), pp. 1–67.

[118] Santhosh K Ramakrishnan et al. "Habitat-matterport 3D dataset (HM3D): 1000 large-scale 3D environments for embodied AI". In: *arXiv preprint arXiv:2109.08238* (2021).

[119] Santhosh K. Ramakrishnan et al. "PONI: Potential Functions for ObjectGoal Navigation with Interaction-free Learning". In: *Computer Vision and Pattern Recognition (CVPR), 2022 IEEE Conference on.* IEEE. 2022.

[120] Ram Ramrakhya et al. "Habitat-Web: Learning Embodied Object-Search Strategies from Human Demonstrations at Scale". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 5173–5183.

[121] Raymond Reiter. "The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression." In: *Artificial and Mathematical Theory of Computation* 3 (1991).

[122] Allen Z. Ren et al. "Robots That Ask For Help: Uncertainty Alignment for Large Language Model Planners". In: *7th Annual Conference on Robot Learning*. 2023. URL: `https://openreview.net/forum?id=4ZK8ODNyFXx`.

[123] Homero Roman Roman et al. "RMM: A Recursive Mental Model for Dialog Navigation". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*. 2020. URL: `https://arxiv.org/abs/2005.00728`.

[124] Yangjun Ruan et al. "Identifying the Risks of LM Agents with an LM-Emulated Sandbox". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: `https://openreview.net/forum?id=GEcwtMk1uA`.

[125] Manolis Savva et al. "Habitat: A platform for embodied ai research". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 9339–9347.

[126] Manolis Savva et al. "MINOS: Multimodal indoor simulator for navigation in complex environments". In: *arXiv preprint arXiv:1712.03931* (2017).

[127] Timo Schick et al. "Toolformer: Language models can teach themselves to use tools". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 68539–68551.

[128] Timo Schick et al. "Toolformer: Language models can teach themselves to use tools". In: *Advances in Neural Information Processing Systems* 36 (2024).

[129] J A Sethian. "A fast marching level set method for monotonically advancing fronts". In: *Proceedings of the National Academy of Sciences* 93.4 (1996), pp. 1591–1595. ISSN: 0027-8424. DOI: `10.1073/pnas.93.4.1591`. eprint: `https://www.pnas.org/content/93/4/1591.full.pdf`. URL: `https://www.pnas.org/content/93/4/1591`.

[130] Dhruv Shah, Błażej Osiński, Sergey Levine, et al. "Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action". In: *Conference on Robot Learning*. PMLR. 2023, pp. 492–504.

[131] Lanbo She et al. “Back to the Blocks World: Learning New Actions through Situated Human-Robot Dialogue”. In: *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*. Philadelphia, PA, U.S.A.: Association for Computational Linguistics, June 2014, pp. 89–97. DOI: 10.3115/v1/W14-4313. URL: https://aclanthology.org/W14-4313.

[132] Noah Shinn et al. “Reflexion: Language agents with verbal reinforcement learning”. In: *Advances in Neural Information Processing Systems* 36 (2024).

[133] Mohit Shridhar et al. “ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 10740–10749.

[134] Mohit Shridhar et al. “ALFWorld: Aligning text and embodied environments for interactive learning”. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2021.

[135] Ishika Singh et al. “Progprompt: Generating situated robot task plans using large language models”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 11523–11530.

[136] Kunal Pratap Singh et al. “Ask4Help: Learning to Leverage an Expert for Embodied Tasks”. In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. 2022. URL: https://openreview.net/forum?id=_bqtjfpj8h.

[137] Kunal Pratap Singh et al. “Moca: A modular object-centric approach for interactive instruction following”. In: *arXiv preprint arXiv:2012.03208* (2020).

[138] Noah Snavely, Steven M Seitz, and Richard Szeliski. “Modeling the world from internet photo collections”. In: *International journal of computer vision* 80.2 (2008), pp. 189–210.

[139] Chan Hee Song et al. “Llm-planner: Few-shot grounded planning for embodied agents with large language models”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 2998–3009.

[140] Julian Straub et al. “The Replica dataset: A digital replica of indoor spaces”. In: *arXiv preprint arXiv:1906.05797* (2019).

[141] Alane Suhr et al. “Executing instructions in situated collaborative interactions”. In: *arXiv preprint arXiv:1910.03655* (2019).

[142] Stefanie Tellex et al. “Robots That Use Language”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 3.1 (2020), null. DOI: 10.1146/annurev-control-101119-071628. URL: https://doi.org/10.1146/annurev-control-101119-071628.

[143] Stefanie Tellex et al. “Toward information theoretic human-robot dialog”. In: *Robotics: Science and Systems*. 2013.

[144] Stefanie Tellex et al. “Understanding natural language commands for robotic navigation and mobile manipulation”. In: *Proceedings of the National Conference on Artificial Intelligence*. 2011.

[145] Jesse Thomason et al. “Vision-and-dialog navigation”. In: *Conference on Robot Learning*. PMLR. 2020, pp. 394–406.

[146] Yu Tian et al. “Evil geniuses: Delving into the safety of llm-based agents”. In: *arXiv preprint arXiv:2311.11855* (2023).

[147] Hugo Touvron et al. “Llama: Open and efficient foundation language models”. In: *arXiv preprint arXiv:2302.13971* (2023).

[148] Jonathan Uesato et al. *Solving math word problems with process- and outcome-based feedback*. 2022. arXiv: `2211.14275 [cs.LG]`. URL: `https://arxiv.org/abs/2211.14275`.

[149] Athanasios Voulodimos et al. “Deep learning for computer vision: A brief review”. In: *Computational intelligence and neuroscience* 2018 ().

[150] Harm de Vries, Dzmitry Bahdanau, and Christopher Manning. “Towards Ecologically Valid Research on Language User Interfaces”. In: *arXiv:2007.14435* (July 2020). URL: `https://arxiv.org/abs/2007.14435`.

[151] Guanzhi Wang et al. “Voyager: An open-ended embodied agent with large language models”. In: *arXiv preprint arXiv:2305.16291* (2023).

[152] Lei Wang et al. “A survey on large language model based autonomous agents”. In: *arXiv preprint arXiv:2308.11432* (2023).

[153] Peiyi Wang et al. “Math-shepherd: Verify and reinforce llms step-by-step without human annotations”. In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2024, pp. 9426–9439.

[154] Ruocheng Wang et al. “Language-Mediated, Object-Centric Representation Learning”. In: *arXiv preprint arXiv:2012.15814* (2020).

[155] Tianlu Wang et al. *Shepherd: A Critic for Language Model Generation*. 2023. arXiv: `2308.04592 [cs.CL]`. URL: `https://arxiv.org/abs/2308.04592`.

[156] Xiaolong Wang, Abhinav Shrivastava, and Abhinav Gupta. “A-fast-rcnn: Hard positive generation via adversary for object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2606–2615.

[157] Xin Wang et al. “Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 6629–6638.

[158] Xingyao Wang et al. “Executable code actions elicit better llm agents”. In: *arXiv preprint arXiv:2402.01030* (2024).

[159] Xingyao Wang et al. *OpenHands: An Open Platform for AI Software Developers as Generalist Agents.* 2024. arXiv: `2407.16741 [cs.SE]`. URL: `https://arxiv.org/abs/2407.16741`.

[160] Jason Wei et al. "Chain-of-thought prompting elicits reasoning in large language models". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 24824–24837.

[161] Luca Weihs et al. "Visual room rearrangement". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.* 2021, pp. 5922–5931.

[162] Sean Welleck et al. "Generating Sequences by Learning to Self-Correct". In: *International Conference on Learning Representations.* 2023.

[163] Erik Wijmans et al. "Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames". In: *arXiv preprint arXiv:1911.00357* (2019).

[164] Thomas Wolf et al. "Huggingface's transformers: State-of-the-art natural language processing". In: *arXiv preprint arXiv:1910.03771* (2019).

[165] Bo Wu et al. "Star: A benchmark for situated reasoning in real-world videos". In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2).* 2021.

[166] Yue Wu et al. "Plan, Eliminate, and Track–Language Models are Good Teachers for Embodied Agents". In: *arXiv preprint arXiv:2305.02412* (2023).

[167] Yue Wu et al. "SPRING: Studying Papers and Reasoning to play Games". In: *Advances in Neural Information Processing Systems* 36 (2024).

[168] Yueh-Hua Wu et al. "Imitation learning from imperfect demonstration". In: *International Conference on Machine Learning.* PMLR. 2019, pp. 6818–6827.

[169] Fei Xia et al. "Gibson Env: Real-World Perception for Embodied Agents". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2018, pp. 9068–9079. DOI: `10.1109/CVPR.2018.00945`.

[170] Yuxin Xiao et al. "Uncertainty Quantification with Pre-trained Language Models: A Large-Scale Empirical Analysis". In: *Conference on Empirical Methods in Natural Language Processing.* 2022. URL: `https://api.semanticscholar.org/CorpusID:247613322`.

[171] Annie Xie et al. "When to Ask for Help: Proactive Interventions in Autonomous Reinforcement Learning". In: *Advances in Neural Information Processing Systems.* Ed. by Alice H. Oh et al. 2022. URL: `https://openreview.net/forum?id=L9EXtg7h6XE`.

[172] Yuxi Xie et al. "Self-Evaluation Guided Beam Search for Reasoning". In: *Thirty-seventh Conference on Neural Information Processing Systems.* 2023. URL: `https://openreview.net/forum?id=Bw82hwg5Q3`.

[173] Miao Xiong et al. “Can LLMs Express Their Uncertainty? An Empirical Evaluation of Confidence Elicitation in LLMs”. In: *The Twelfth International Conference on Learning Representations*. 2024. URL: `https://openreview.net/forum?id=gjeQKFxFpZ`.

[174] Karmesh Yadav et al. “Offline Visual Representation Learning for Embodied Navigation”. In: *arXiv preprint arXiv:2204.13226* (2022).

[175] Brian Yamauchi. “A frontier-based approach for autonomous exploration”. In: *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'*. IEEE. 1997, pp. 146–151.

[176] Boling Yang et al. “Benchmarking robot manipulation with the rubik's cube”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 2094–2099.

[177] Jianwei Yang et al. “Embodied visual recognition”. In: *arXiv preprint arXiv:1904.04404* (2019).

[178] John Yang et al. “SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering”. In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024. URL: `https://arxiv.org/abs/2405.15793`.

[179] Wei Yang et al. “Visual semantic navigation using scene priors”. In: *arXiv preprint arXiv:1810.06543* (2018).

[180] Zonghan Yang et al. “Towards Unified Alignment Between Agents, Humans, and Environment”. In: *arXiv preprint arXiv:2402.07744* (2024).

[181] Shunyu Yao et al. “React: Synergizing reasoning and acting in language models”. In: *arXiv preprint arXiv:2210.03629* (2022).

[182] Shunyu Yao et al. “ReAct: Synergizing Reasoning and Acting in Language Models”. In: *International Conference on Learning Representations (ICLR)*. 2023.

[183] Shunyu Yao et al. “Tree of thoughts: Deliberate problem solving with large language models”. In: *Advances in Neural Information Processing Systems* 36 (2024).

[184] Bangguo Yu, Hamidreza Kasaei, and Ming Cao. “L3MVN: Leveraging Large Language Models for Visual Target Navigation”. In: *arXiv preprint arXiv:2304.05501* (2023).

[185] Eric Zelikman et al. “STaR: Bootstrapping Reasoning With Reasoning”. In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. 2022. URL: `https://openreview.net/forum?id=_3ELRdg2sgI`.

[186] Yuexiang Zhai et al. “Fine-Tuning Large Vision-Language Models as Decision-Making Agents via Reinforcement Learning”. In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024. URL: `https://openreview.net/forum?id=nBjmMF2IZU`.

[187] Kechi Zhang et al. "CodeAgent: Enhancing Code Generation with Tool-Integrated Agent Systems for Real-World Repo-level Coding Challenges". In: *arXiv preprint arXiv:2401.07339* (2024).

[188] Michael JQ Zhang and Eunsol Choi. "Clarify When Necessary: Resolving Ambiguity Through Interaction with LMs". In: *arXiv preprint arXiv:2311.09469* (2023).

[189] Songyuan Zhang et al. "Confidence-Aware Imitation Learning from Demonstrations with Varying Optimality". In: *Advances in Neural Information Processing Systems* 34 (2021).

[190] Yichi Zhang and Joyce Chai. "Hierarchical Task Learning from Language Instructions with Unified Transformers and Self-Monitoring". In: *arXiv preprint arXiv:2106.03427* (2021).

[191] Gaoyue Zhou et al. "Train Offline, Test Online: A Real Robot Learning Benchmark". In: *arXiv preprint arXiv:2306.00942* (2023).

[192] Gengze Zhou, Yicong Hong, and Qi Wu. "NavGPT: Explicit Reasoning in Vision-and-Language Navigation with Large Language Models". In: *arXiv preprint arXiv:2305.16986* (2023).

[193] Kaitlyn Zhou, Dan Jurafsky, and Tatsunori Hashimoto. "Navigating the Grey Area: How Expressions of Uncertainty and Overconfidence Affect Language Models". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 5506–5524. DOI: `10.18653/v1/2023.emnlp-main.335`. URL: `https://aclanthology.org/2023.emnlp-main.335/`.

[194] Kaiwen Zhou et al. "Esc: Exploration with soft commonsense constraints for zero-shot object navigation". In: *arXiv preprint arXiv:2301.13166* (2023).

[195] Shuyan Zhou et al. "Webarena: A realistic web environment for building autonomous agents". In: *arXiv preprint arXiv:2307.13854* (2023).

[196] Xingyi Zhou et al. "Detecting twenty-thousand classes using image-level supervision". In: *European Conference on Computer Vision*. Springer. 2022, pp. 350–368.

[197] Fengda Zhu et al. "Vision-language navigation with self-supervised auxiliary reasoning tasks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 10012–10022.

[198] Hao Zhu et al. "EXCALIBUR: Encouraging and Evaluating Embodied Exploration". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 14931–14942.

[199] Yuke Zhu et al. “Target-driven visual navigation in indoor scenes using deep reinforcement learning”. In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, pp. 3357–3364.