A Persistent Spatial Semantic Representation for High-level Natural Language Instruction Execution

Valts Blukis[1,2], **Chris Paxton**[1], **Dieter Fox**[1,3], **Animesh Garg**[1,4], **Yoav Artzi**[2]
[1]NVIDIA [2]Cornell University [3]University of Washington
[4]University of Toronto, Vector Institute

Abstract: Natural language provides an accessible and expressive interface to specify long-term tasks for robotic agents. However, non-experts are likely to specify such tasks with high-level instructions, which abstract over specific robot actions through several layers of abstraction. We propose that key to bridging this gap between language and robot actions over long execution horizons are persistent representations. We propose a persistent spatial semantic representation method, and show how it enables building an agent that performs hierarchical reasoning to effectively execute long-term tasks. We evaluate our approach on the ALFRED benchmark and achieve state-of-the-art results, despite completely avoiding the commonly used step-by-step instructions. `https://hlsm-alfred.github.io/`

Keywords: vision and language, spatial representations

1 Introduction

Mobile manipulation in a home environment requires addressing multiple challenges, including exploration and making long-term inference about actions to perform. In addition to reasoning, robots require an accessible, yet sufficiently expressive interface to specify their tasks. Natural Language provides an intuitive mechanism for task specification, and coupled with advances in automated language understanding, is increasingly applied to embodied agents [e.g., 1–11].

In this paper, we study the problem of learning to map high-level natural language instructions to low-level mobile manipulation actions in an interactive 3D environment [12]. Existing work largely studies language tightly aligned to the robot actions, either using single-sentence instructions [e.g., 1, 2, 5, 9] or sequences of instructions [13–18]. In contrast, we focus on high-level instructions, which provide more efficient human-robot communication, but require long-horizon reasoning across layers of abstraction to generate actions not explicitly specified in the instruction.

Robust reasoning about manipulation goals from unrestricted high-level natural language instructions has a variety of open challenges. Consider the instruction *secure two discs in a bedroom safe* (Figure 1). The robot must first locate the *safe* in the *bedroom*. It then needs to distribute the actions entailed by *secure* to two objects (*two discs*), each requiring a distinct sequence of actions, but targeting the same *safe*. It is also required to map the verb *secure* to its action space. In parallel, the robot must address mobile manipulation challenges, and often can only identify required actions as it observes and manipulates the world (e.g., if the *safe* needs to be opened).

We propose to construct and continually update a spatial semantic representation of the world from robot observations (Figure 2). Similar to widely used map representations [19–22], we retain the spatial properties of the environment, allowing the robot to navigate and reason about relations between objects, as required to accomplish its task. We propose the Hierarchical Language-conditioned Spatial Model (HLSM), a hierarchical approach that uses our spatial representation as a long-term memory to solve long-horizon tasks. HLSM consists of a high-level controller that generates subgoals, and a low-level controller that generates sequences of actions to accomplish them. In our example (Figure 1), the sequence of subgoals is ⟨pick up a CD, open the safe, put the CD in the safe, . . .⟩, each requiring a sequence of actions. The spatial representation allows selecting subgoals that use previously observed objects outside of the agent's view, or to decide about needed exploration.

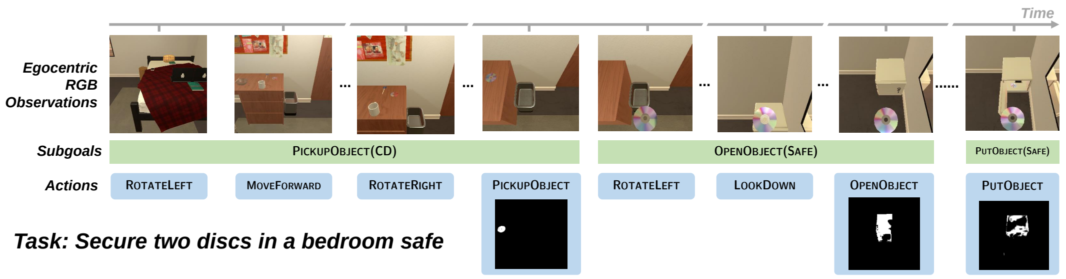

Figure 1: Illustration of the task and our hierarchical formulation. The agent receives a high-level task in natural language. It needs to map RGB images to navigation and manipulation actions to complete the task.

We evaluate our approach on the ALFRED [12] benchmark and achieve state-of-the-art results without using the low-level instructions used by previous work [16–18, 23], neither during training nor at test-time. This paper makes three key contributions: (a) a modular representation learning approach for the problem of mapping high-level natural language task descriptions to actions in a 3D environment; (b) a method for utilizing a spatial semantic representation within a hierarchical model for solving mobile manipulation tasks; and (c) state-of-the-art performance on the ALFRED benchmark, even outperforming all approaches that use detailed sequential instructions.

2 Related Work

Natural language has been extensively studied in robotics research, including with focus on instruction [1, 24], reference resolution [25], question generation [26–28], and dialogue [4, 29, 30]. Most work in this area has considered either synthetic instructions of relatively simple goals [7, 31–33], or natural language instructions where all intermediate steps are explained in detail [5, 12–14, 34–38]. In contrast, we focus on high-level instructions, which are more likely in home environments [39].

Representation of world state, action history, and language semantics plays a central role in robot systems and their algorithm design. Symbolic representations have been extensively studied for instruction following agents [1–4, 19, 20, 39–45]. While they simplify the symbol grounding problem and enable robustness, the ontologies on which they rely on are laborious to scale to new, unstructured environments and language. Representation learning presents an alternative by learning to map observations and language directly to actions [5, 8, 9, 11, 13, 34]. World state and language semantics are represented with vectors [13] or by memorizing past observations [8, 17]. Modelling improvements have enabled these approaches to achieve good performance on complex navigation tasks [7, 9, 11, 13, 14, 37], a success that has not yet translated to mobile manipulation [12, 46, 47].

We propose integrating a semantic voxel map state representation within a hierarchical representation learning system. Similar semantic 2D maps have been successfully used in navigation [7, 8, 48, 49] and more recently even in mobile manipulation instruction-following tasks [23]. We extend these maps to 3D and show state-of-the-art results on a challenging mobile manipulation benchmark. Our map design is related to sparse metric, topological and semantic maps [10, 19–21, 50] that have enabled grounding symbolic instruction representations. Our map does not impose a topological structure or require reasoning about object instances, instead modelling a distribution over semantic classes for every voxel.

3 Problem Definition

Let $\mathcal{A}$ be the set of agent actions, and $\mathcal{S}$ the set of world states. Given a natural language instruction L and an initial state $s_0 \in \mathcal{S}$, the agent's goal is to generate an execution $\Xi = \langle s_0, a_0, s_1, a_1, \ldots, s_T, a_T \rangle$, where $a_t \in \mathcal{A}$ is an action taken by the agent at time t, $s_t \in \mathcal{S}$ is the state before taking a_t, and $s_{t+1} = \mathcal{T}(s_t, a_t)$ under environment dynamics $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. The state s_t is defined by the environment layout and the poses and states of all objects and the agent. The agent does not have access to the state s_t, but only to an observation o_t. An observation $o_t = (I_t, P_t, v_t^S, L)$ includes a first-person RGB camera image I_t, the agent's pose P_t, a one-hot encoding of the object class the agent is holding v_t^S, and the instruction L.The task is considered

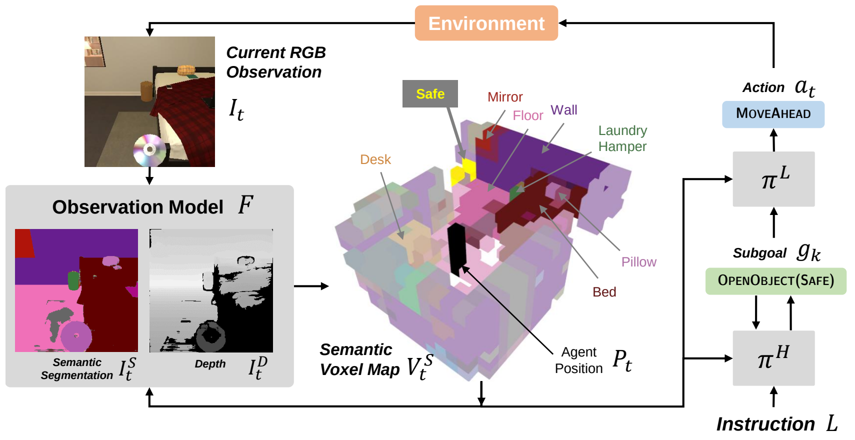

Figure 2: Model architecture consisting of an observation model, high-level controller (π^H), and low-level controller (π^L). The observation model updates the semantic voxel map state representation from RGB observations. π^H predicts the next subgoal given the instruction and the map. π^L outputs a sequence of actions to achieve the subgoal. The semantic voxel map is visualized in the middle with agent position illustrated as a black pillar, ans the current sugoal argument mask in yellow. Other colors are different segmentation classes. Saturated voxels are observed in the current timestep.

successful if all goal-conditions corresponding to the task L are true at the final state s_T. Partial success is measured as the fraction of goal-conditions that have been achieved.

The ALFRED dataset includes sets of seen and unseen environments. The set of actions $\mathcal{A} = \mathcal{A}_{\text{nav}} \cup \mathcal{A}_{\text{int}}$ includes parameter-free navigation actions $\mathcal{A}_{\text{nav}} = \{\text{MOVEAHEAD}, \text{ROTATELEFT}, \text{ROTATERIGHT}\}$ and interaction actions $\mathcal{A}_{\text{int}} = \{\text{PICKUP}, \text{PUT}, \text{TOGGLEON}, \text{TOGGLEOFF}, \text{OPEN}, \text{CLOSE}, \text{SLICE}\}$ parameterized by a binary mask that identifies the object of the interaction in the agent's current first-person view. We compute P_t and v_t^S using dead-reckoning from RGB observations and actions.

4 Hierarchical Model with a Persistent Spatial Semantic Representation

We model the agent behavior with a policy π that maps an instruction L and the observation o_t at time t to an action a_t. The policy π is made of an *observation model* F and two controllers: a *high-level controller* π^H and a *low-level controller* π^L. The observation model builds a spatial *state representation* $\hat{s}_t$ that captures the cumulative agent knowledge of the world at time t. $\hat{s}_t$ is used by both π^H for high-level long-horizon task planning, and π^L for near-term reasoning, such as object search, navigation, collision avoidance, and manipulation. Figure 2 illustrates the policy.

The high-level controller π^H computes a probability over *subgoals*. A subgoal g is a tuple $(\texttt{type}, \texttt{arg}^C, \texttt{arg}^M)$, where $\texttt{type} \in \mathcal{A}_{\text{int}}$ is an interaction type (e.g., OPEN, PICKUP), $\texttt{arg}^C$ is the semantic class of the interaction argument (e.g., SAFE, CD), and $\texttt{arg}^M$ is a 3D mask identifying the location of the argument instance. In ALFRED, each interaction action in the set $\mathcal{A}_{\text{int}}$ corresponds to a subgoal type. When predicting the k-th subgoal at time t, π^H considers the instruction L, the current state representation $\hat{s}_t$, and the sequence of past subgoals $\langle g_i, \rangle_{i<k}$. During inference, we sample from π^H. Unlike $\arg\max$, sampling allows the agent to re-try the same or different subgoal incase of a potentially random failure (e.g., if a MUG was not found, pick up a CUP).

The low-level controller π^L is given the subgoal g_k as its goal specification at time t. At every timestep $j > t$, π^L maps the state representation $\hat{s}_j$ and subgoal g_k to an action a_j, until it outputs one of the stop actions: a_{PASS} or a_{FAIL} to indicate successful or failed subgoal completion.

The execution flow is as follows. At time $t = 0$ the initial observation o_0 is received. At each timestep, we update the state representation $\hat{s}_t$ using the observation model. If there is no currently active subgoal, we sample a new subgoal g_k from π^H, and then sample an action a_t from π^L. If a_t is a_{PASS}, we increment subgoal counter k. If it is a_{FAIL}, we discard the current subgoal k. We repeat sampling subgoals and actions until an executable action a_t is sampled. We execute a_t, increment the timestep t, and receive the next observation o_t. The episode ends when the subgoal g_{STOP} is sampled or the horizon T_{max} is exceeded. Algorithm 1 in Appendix A.4 describes this process.

4.1 State Representation

The state representation $\hat{s}_t$ at time t captures the agent's current understanding of the state of the world, including the locations of objects observed and the agent's relation to them. The state representation is a tuple $(V_t^S, V_t^O, v_t^S, P_t)$. The semantic map $V_t^S \in [0,1]^{X \times Y \times Z \times C}$ is a 3D voxel map that for every position indicates which of the $c \in [1, C]$ object classes are present in the voxel. The observability map $V_t^O \in \{0,1\}^{X \times Y \times Z}$ is a 3D voxel map that indicates whether the corresponding position has been observed. The inventory vector $v_t^S \in \{0,1\}^C$ indicates which of the C object classes the agent is currently holding. The agent pose $P_t = (x, y, \omega_p, \omega_y)$ is specified by the 2D position (x, y), pitch angle ω_p, and yaw angle ω_y.

We also compute 2D *state affordance features* $\text{AFFORD}(\hat{s}_t) \in [0,1]^{7 \times X \times Y}$ in a top-down view that represent each position with one or more of seven affordance classes {`pickable`, `receptacle`, `togglable`, `openable`, `ground`, `obstacle`, `observed`}. Each $[\text{AFFORD}(\hat{s}_t)]_{(\tau,x,y)} = 1.0$ if at least one of the voxels at position (x, y) has affordance class τ, otherwise it is zero. $\text{AFFORD}(\hat{s}_t)$ is suited for object class agnostic reasoning, for example predicting a pose to pick up an object. [1]

4.2 Observation Model

The observation model $F(\hat{s}_{t-1}, o_t, g_k)$ updates the state representation with new observations. It considers the current subgoal g_k to actively acquire information relevant to g_k. The computation of F consists of three steps: perception, projection, accumulation.

Perception Step We predict semantic segmentation I_t^S and depth map I_t^D from the RGB observation I_t. We use neural networks pre-trained in the ALFRED environment. The semantic segmentation $[I_t^S]_{(u,v)}$ is a distribution over C object classes at pixel (u, v). The depth map $[I_t^D]_{(u,v)}$ is a binned distribution over B bins.[2] We also heuristically compute a binary mask M_t^D that indicates which pixels have confident depth readings. We allow more confidence slack in pixels that correspond to the current subgoal argument $\texttt{arg}_t^C$ according to I_t^S. Appendix A.3 provides further details. We use perception models based on the U-Net [51] architecture, but our framework supports other, potentially more powerful models as well (e.g. [52, 53]).

Projection Step We use a pinhole camera model to convert depth I_t^D and segmentation I_t^S to a point cloud that represents each image pixel (u, v) with a 3D position $(x, y, z) \in \mathbb{R}^{X \times Y \times Z}$ and a semantic distribution $[I_t^S]_{(u,v)}$. We use $\arg\max_B(I_t^D)$ to compute the 3D positions, and discard points at pixels (u, v) when the binary mask value is $[M_t^D]_{(u,v)} = 0$. We construct a discrete semantic voxel map $\hat{V}_t^S \in [0,1]^{X \times Y \times Z \times C}$, where X, Y, and Z are the width, height, and length. The value at each voxel $[\hat{V}_t^S]_{(x,y,z)}$ is the element-wise maximum of the segmentation distributions $[I_t^S]_{(u,v)}$ across all points (u, v) within the voxel. We additionally compute a binary observability map $\hat{V}_t^O \in \{0,1\}^{X \times Y \times Z}$ that indicates the voxels observed at time t. A voxel is observed if it contains points, or if a ray cast from the camera through the voxel centroid has expected depth greater than the distance from the camera to the centroid.

Accumulation Step We integrate $\hat{V}_t^S$ and $\hat{V}_t^O$ into a persistent state representation:

$$V_t^S = \hat{V}_t^S \times \hat{V}_t^O + V_{t-1}^S \times (1 - \hat{V}_t^O) \qquad V_t^O = \max(V_{t-1}^O, \hat{V}_t^O) \ . \tag{1}$$

This operation updates each voxel with the most recent semantic distribution, while retaining the values of all voxels not visible at time t. The output of the observation model is the spatial state representation $\hat{s}_t = (V_t^S, V_t^O, v_t^S, P_t)$. The inventory v_t^S and pose P_t are taken directly from o_t.

[1] We assume a known mapping between object semantic classes and affordance classes.

[2] We use B uniformly spaced depth bins $\{0, \Delta_D, 2\Delta_D, \ldots, (B-1)\Delta_D\}$, where Δ_D is a depth resolution. We suggest Δ_D should be less than 50% of the voxel size. We used voxels with edge length 0.25m.

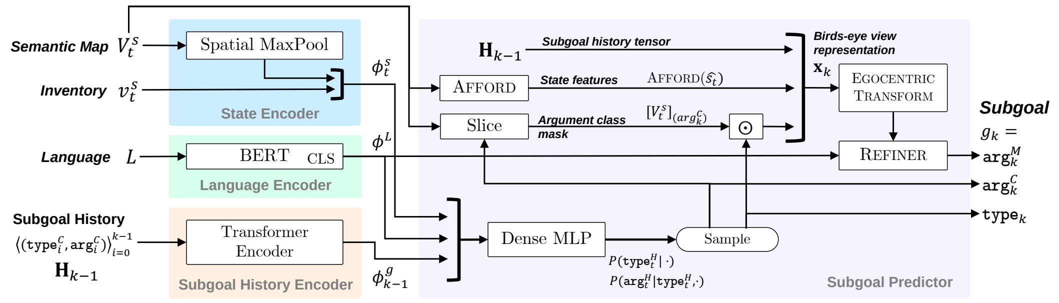

Figure 3: Illustration of the high-level controller π^H (Section 4.3).

4.3 High-level Controller (π^H)

At timestep t, when invoked for the k-th time, the input to π^H is the instruction L, the sequence of past subgoals $\langle g_i \rangle_{i<k}$, and the current state representation $\hat{s}_t$. The output is the next subgoal $g_k = (\texttt{type}_k, \texttt{arg}_k^C, \texttt{arg}_k^M)$. Figure 3 illustrates the high-level controller architecture.

Input Encoding We encode the text L using a pre-trained BERT [54] model that we fine-tune during training. We use the CLS token embedding as the task embedding ϕ^L. We encode the state representation $\hat{s}_t$ to account for classes of all observed objects, and the object that the agent is holding: $\phi^s(\hat{s}_t) = [v_t^S; \max_{(x,y,z)}(V_t^S)]$, where $\max_{(x,y,z)}$ is a max-pooling operation over spatial dimensions and $[\cdot;\cdot]$ denotes concatenation. We compute the representations of previous subgoals as $\langle \textsc{Repr}(g_i) \rangle_{i=0}^{k-1}$, where $\textsc{Repr}(g_i)$ is the sum of a sinusoidal positional encoding [55] of index i and learned embeddings for $\texttt{type}_i$ and $\texttt{arg}_i^C$. We process this sequence with a two-layer Transformer autoregressive encoder [55] to compute $\langle \phi_i^g \rangle_{i=0}^{k-1}$. We take ϕ_{k-1}^g as the subgoal history embedding vector. We additionally encode the argument mask information $\texttt{arg}_i^M$ from the subgoal history in an integer-valued subgoal history tensor $\mathbf{H}_{k-1} \in \mathbb{N}^{K \times X \times Y}$ where $[\mathbf{H}_{k-1}]_{(\tau,x,y)}$ is the number of times an interaction action type τ was performed at 2D position (x, y) in the birds-eye view:

$$[\mathbf{H}_{k-1}]_{(\tau,x,y)} = \sum_{\substack{i=0\dots k-1 \\ \texttt{arg}_i^C = \tau}}^{k-1} \max_z([\texttt{arg}_i^M]_{(x,y,z)}) \quad . \tag{2}$$

Subgoal Prediction We concatenate the three representations $\mathbf{h}_{(t,k)} = [\phi^L; \phi_t^s; \phi_{k-1}^g]$. We use a densely connected multi-layer perceptron [56] to predict two distributions $P(\texttt{type}_k \mid \mathbf{h}_{(t,k)})$ and $P(\texttt{arg}_k^C \mid \texttt{type}_k, \mathbf{h}_{(t,k)})$, from which we sample a subgoal type $\texttt{type}_k$ and argument class $\texttt{arg}_k^C$.

The remaining component of the subgoal is the action argument mask $\texttt{arg}_k^M$. Let $[V_t^S]_{(\texttt{arg}_k^C)}$ be a voxel map that only retains the object information for objects of class $\texttt{arg}_k^C$ in the semantic map V_t^S. We refine it to identify a single object instance. We compute a birds-eye view representation:

$$\mathbf{x}_t = [\textsc{Afford}(\hat{s}_t);\ \mathbf{H}_{k-1};\ \max_z([V_t^S]_{(\texttt{arg}_k^C)}) \otimes \mathbb{1}_{\texttt{type}_k}] \tag{3}$$

where $\textsc{Afford}(\hat{s}_t)$ is a birds-eye view state affordance feature map (Section 4.1) and $\mathbb{1}_{\texttt{type}_k}$ is a one-hot encoding of $\texttt{type}_k$.[3] Finally, we compute the 3D argument mask $\texttt{arg}_k^M \in [0,1]^{X \times Y \times Z}$:

$$\texttt{arg}_k^M = \textsc{Refiner}(\textsc{EgoTransform}(\mathbf{x}_t, P_t), \phi^L) \quad , \tag{4}$$

where $\textsc{EgoTransform}(\mathbf{x}, P_t)$ transforms the map $\mathbf{x}$ to the agent egocentric pose P_t, $\textsc{Refiner}$ is a neural network based on the LingUNet architecture [36], and ϕ^L is the language embedding. The refined $\texttt{arg}_k^M$ is a $[0,1]$-valued 3D mask that identifies the instance of the interaction argument object. If the object is believed to be unobserved, then $\texttt{arg}_k^M$ contains all zeroes. The controller output is the subgoal $g_k = (\texttt{type}_k, \texttt{arg}_k^C, \texttt{arg}_k^M)$.

4.4 Low-level Controller (π^L)

The low-level controller π^L is conditioned on the most recent subgoal $g_k = (\texttt{type}_k, \texttt{arg}_k^C, \texttt{arg}_k^M)$. At time t, it maps the state representation $\hat{s}_t$ to an action a_t. It combines engineered and learned

[3] $\otimes$ denotes multiplication of a $X \times Y$ tensor with a K-dimensional vector to obtain a $K \times X \times Y$ tensor. $[\cdot;\cdot;\cdot]$ denotes channel-wise concatenation.

components. Appendix A.6 provides the implementation details. The controller π^L invokes a set of procedures: `NavigateTo`, `SampleExplorationPosition`, `SampleInteractionPose`, and `InteractMask`. Their invokation follows a pre-specified execution flow across multiple timesteps. First, we perform a 360° rotation to observe the nearby environment. If no objects of type $\texttt{arg}_k^C$ are observed, we explore the environment by sampling a position $(x, y) = \texttt{SampleExplorationPosition}(\hat{s}_t)$, navigating there using the procedure $\texttt{NavigateTo}(x, y, \hat{s}_t)$, and performing a 360° rotation. We repeat exploration until a voxel in V_t^S contains the class $\texttt{arg}_k^C$ with >50% probability. To interact with an object, we sample an interaction pose $(x, y, \omega_y, \omega_p) = \texttt{SampleInteractionPose}(\hat{s}_t, g_k)$, invoke $\texttt{NavigateTo}(x, y, \hat{s}_t)$ to reach the position (x,y), and then rotate according to yaw and pitch angles (ω_y, ω_p). Finally, we generate the egocentric interaction mask $\texttt{mask}_t = \texttt{InteractMask}(\hat{s}_t, \texttt{arg}_k^M)$, and output the interaction action $(\texttt{type}_k, \texttt{mask}_t)$.

All procedures use the spatial representation $\hat{s}_t$. `NavigateTo` navigates to a goal position using a value iteration network (VIN) [57] that reasons over obstacle and observability maps from $\hat{s}_t$. `SampleExplorationPosition` samples positions on the boundary of observed space in $\hat{s}_t$. `SampleInteractionPose` uses a learned neural network NAVMODEL to predict a distributon of poses from which the interaction g_k will likely succeed. `InteractMask` uses the segmentation image I_t^S and the 3D argument mask $\texttt{arg}_t^M$ to compute the first-person mask of the target object.

5 Learning

The policy contains four learned models: the segmentation and depth networks, π^H, and the navigation model NAVMODEL used by π^L. We train all four networks independently using supervised learning. We assume access to a training dataset $\mathcal{D} = \{(L^{(j)}, \Xi^{(j)})\}_{j=1}^{N_\mathcal{D}}$ of high-level natural language instructions $L^{(j)}$ paired with demonstration execution $\Xi^{(j)}$ in a set of seen environments. Each execution $\Xi^{(j)}$ is a sequence of states and actions $\langle s_0^{(j)}, a_0^{(j)}, \ldots, s_T^{(j)}, a_T^{(j)} \rangle$. We denote N_P the total number of states in dataset $\mathcal{D}$, and N_G the total number of subgoals.

We process $\mathcal{D}$ into three datasets. The perception dataset $\mathcal{D}^P = \{([I]^{(i)}, [I^D]^{(i)}, [I^S]^{(i)}\}_{i=1}^{N_P}$ includes RGB images $[I]^{(i)}$ with ground truth depth $[I^D]^{(i)}$ and segmentation $[I^S]^{(i)}$. The subgoal dataset $\mathcal{D}^g = \{(L^{(i)}, \hat{s}_t^{(i)}, \langle g_j^{(i)} \rangle_{j=0}^k)\}_{i=1}^{N_G}$ contains natural language instructions $L^{(i)}$, state representations $\hat{s}_t^{(i)}$ at the start of k-th subgoal execution, and sequences of the first k subgoals $\langle g_j^{(i)} \rangle_{j=0}^k$ extracted from $\Xi^{(j)}$. The navigation dataset $\mathcal{D}^N = \{(\hat{s}^{(i)}, g^{(i)}, P^{(i)})\}_{i=1}^{N_P}$ consists of state representations $\hat{s}^{(i)}$, subgoals $g^{(i)}$, and agent poses $P^{(i)}$ at the time of taking the interaction action corresponding to subgoal $g^{(i)}$. The state representations $\hat{s}^{(\cdot)}$ in datasets $\mathcal{D}^g$ and $\mathcal{D}^N$ are constructed using the observation model (Section 4.2), but using ground-truth depth and segmentation images.

We train the perception models on $\mathcal{D}^P$ and the π^H on $\mathcal{D}^g$ to predict the k-th subgoal by optimizing cross-entropy losses. We use $\mathcal{D}^N$ to train the navigation model NAVMODEL by optimizing a cross-entropy loss for positions and yaw angles, and an L2 loss for the pitch angle.

6 Experimental Setup

Environment, Data, and Evaluation We evaluate our approach on the ALFRED [12] benchmark. It contains 108 training scenes, 88/4 validation seen/unseen scenes, and 107/8 test seen/unseen scenes. There are 21,023 training tasks, 820/821 validation seen/unseen tasks, and 1533/1529 test seen/unseen tasks. Each task is specified with a high-level natural language instruction. The goal of the agent is to map raw RGB observations to actions to complete the task. ALFRED also provides detailed low-level step-by-step instructions, which simplify the reasoning process. We do not use these instructions for training or evaluation. We collect a training dataset of language-demonstration pairs for learning (Section 5). To extract subgoal sequences, we label each interaction action $a_t = (\texttt{type}_t, \texttt{mask}_t)$ and any preceding navigation actions with a single subgoal of $\texttt{type} = \texttt{type}_t$. We compute the subgoal argument class $\texttt{arg}^C$ and 3D mask $\texttt{arg}^M$ labels from the first-person mask $\texttt{mask}_t$, and ground truth segmentation and depth. Completing a task requires satisfying several goal conditions. Following the common evaluation [58, 59], we report two metrics.

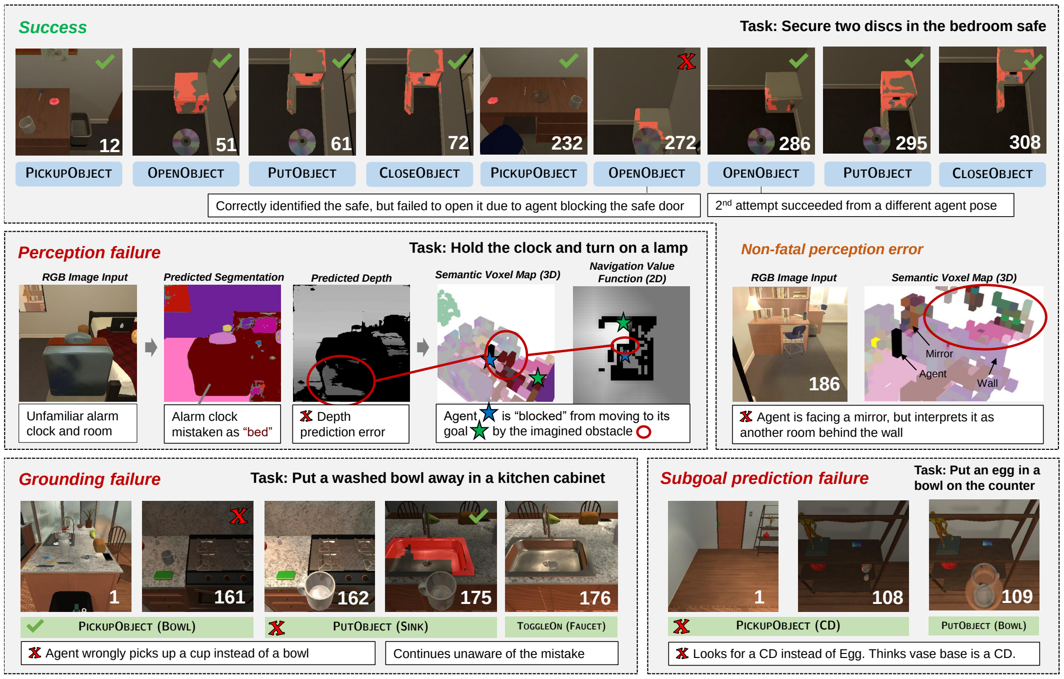
Success
Task: Secure two discs in the bedroom safe
12
51
61
72
232
272
286
295
308
PickupObject
OpenObject
PutObject
CloseObject
PickupObject
OpenObject
OpenObject
PutObject
CloseObject
Correctly identified the safe, but failed to open it due to agent blocking the safe door
2nd attempt succeeded from a different agent pose
Perception failure
Task: Hold the clock and turn on a lamp
RGB Image Input
Predicted Segmentation
Predicted Depth
Semantic Voxel Map (3D)
Navigation Value Function (2D)
Unfamiliar alarm clock and room
Alarm clock mistaken as "bed"
Depth prediction error
Agent is "blocked" from moving to its goal by the imagined obstacle
Non-fatal perception error
RGB Image Input
Semantic Voxel Map (3D)
186
Mirror
Agent
Wall
Agent is facing a mirror, but interprets it as another room behind the wall
Grounding failure
Task: Put a washed bowl away in a kitchen cabinet
1
161
162
175
176
PickupObject (Bowl)
PutObject (Sink)
ToggleOn (Faucet)
Agent wrongly picks up a cup instead of a bowl
Continues unaware of the mistake
Subgoal prediction failure
Task: Put an egg in a bowl on the counter
1
108
109
PickupObject (CD)
PutObject (Bowl)
Looks for a CD instead of Egg. Thinks vase base is a CD.

Figure 4: Qualitative results showcasing successes and failures of our approach. **Top row:** snapshots of every interaction action taken during a successful task. Action argument masks are overlaid in red over the RGB images. The white numbers are timesteps. **Middle-right**: illustration of a non-fatal perception error. **Middle-left**: illustration of a fatal perception error. The agent incorrectly interprets the reflection on the alarm clock as an obstacle, causing the agent (blue star) to believe that the path to the goal (green star) is blocked off. This is reflected in the navigation value function computed by the value iteration network (VIN) [57], where black cells are obstacles with value -1. White cell is the goal with value 1. **Bottom-left:** grounding failure. The agent wrongly picks up the cup instead of a bowl. Predicted subgoals are shown in green. **Bottom-right:** high-level controller and percepton failure. π^H predicts the wrong subgoal argument class (CD instead of EGG). The segmentation model then mistakes the vase for a CD.

Success rate (SR) is the fraction of tasks for which all goal conditions were satisfied. *Goal condition rate* (GC) is the fraction of goal-conditions satisfied across all tasks.

Systems We compare our approach, the Hierarchical Language-conditioned Spatial Model (HLSM) to others on the ALFRED leaderboard that only use the high-level instructions. At the time of writing, the only such published approach is HiTUT [47], an approach that uses a flat BERT [54] architecture to model a hierarchical task structure without using a spatial representation. See Appendix A.2 for a detailed comparison. We also compare to approaches that use the step-by-step instructions, which puts our method at a disadvantage. Of these, LAV [60] also imposes a hierarchical task structure and uses pre-trained depth and segmentation models, but without using a spatial state representation.

Additionally, we perform ablations and study sensory oracles. To study the observation model, we compare to using sensory oracles for ground truth depth, ground truth segmentation, and both. We report high-level controller ablations that remove the subgoal encoder, language encoder, and state representation encoder as used for predicting subgoal type $\texttt{type}_k$ and argument class $\texttt{arg}_k^C$, while still using the state representation $\hat{s}_t$ to predict the subgoal argument mask $\texttt{arg}_k^M$. We also study a low-level controller ablation that removes the exploration procedure.

7 Results

Table 1 shows test and validation results. Our approach achieves state-of-the-art performance across both seen and unseen environments in the setting with only high-level instructions. We achieve 10.04% absolute (98.1% relative) improvement in SR on the test unseen split, and 11.53% absolute (62.6% relative) improvement in SR on the test seen split compared to HiTUT G-only.

Method	Test				Validation			
	Seen		Unseen		Seen		Unseen	
	SR	GC	SR	GC	SR	GC	SR	GC
Low-level Sequential Instructions + High-level Goal Instruction								
SEQ2SEQ [12]	3.98	9.42	0.39	7.03	3.70	10.00	0.00	6.90
MOCA [46]	22.05	28.29	5.30	14.28	19.15	28.5	3.78	13.4
E.T. [17]	28.77	36.47	5.04	15.01	33.78	42.48	3.17	13.12
E.T. + synth. data [17]	38.42	45.44	8.57	18.6	46.59	52.82	7.32	20.87
LWIT [62]	30.92	45.44	9.42	20.91	33.70	43.10	9.70	23.10
HITUT[47]	21.27	29.97	13.87	20.31	25.24	34.85	12.44	23.71
ABP [61]	44.55	51.13	15.43	24.76	42.93	50.45	12.55	25.19
High-level Goal Instruction Only								
HITUT G-only[47]	18.41	25.27	10.23	20.27	13.63	21.11	11.12	17.89
LAV [60]	13.35	23.21	6.38	17.27	12.7	23.4	-	-
HLSM (Ours)	**29.94**	**41.21**	**20.27**	**30.31**	**29.63**	**38.74**	**18.28**	**31.24**

Table 1: Test results. Test seen/unseen and validation seen/unseen splits. Top section approaches use sequential step-by-step instructions. The bottom section uses only high-level instructions. Best results **using only high-level instructions** and using both types of instructions are highlighted.

Our approach performs competitively even when compared to approaches that also use the low-level step-by-step instructions. We achieve 4.84% absolute (31.4% relative) improvement in SR on the test unseen split compared to ABP [61]. On the test seen split, our approach performs reasonably well, however ABP [61] and LWIT [18] perform better, reflecting potentially stronger scene overfitting.

Tables 2 and 3 show development results. We performed five runs of the full HLSM model on the validation unseen data and found the sample standard deviation of the success rate is 1.1% (absolute). All other results are from a single-evaluation runs. Ground truth depth alone (+ gt depth) does not significantly affect performance. Ground truth segmentation (+ gt seg) provides 6.6%/16.4% absolute improvement in seen/unseen scenes. Using both (+ gt depth, gt seg) provides 11.1%/21.9% absolute improvement and narrows the seen/unseen gap from 11.3% to 0.5%. This points to perception being the main bottleneck in generalization to unseen scenes.

We report high-level controller π^H input encoder ablations. The poor performance without the language encoder reflects task difficulty. Zeroing the input to the subgoal history encoder (but keeping position encodings) does not significantly affect performance, showing that knowing the index of the current subgoal in addition to the state representation is often sufficient. Not using the state representation for predicting subgoal type and argument class gives mixed results in seen and unseen scenes, but without a significant difference in performance. Therefore, predicting the sequence of subgoal types and argument classes (i.e., what to do) is at times possible without spatial reasoning, while grounding the subgoal (i.e., where to do it) requires spatial information. Removing random exploration from π^L does not significantly affect unseen performance.

Figure 4 illustrates the model behavior, showing both successes and common failures. The main failures in valid unseen scenes are due to (1) perception errors that result in missing or extraneous obstacles or picking up wrong objects; (2) insufficiency of random exploration (e.g., not searching inside cabinets); (3) navigation model errors (e.g., blocking objects from opening); (4) subgoal prediction errors (e.g., picking up wrong objects); and (5) lack of state-aware multi-step planning and backtracking. More qualitative results are available in Appendix A.10.

8 Discussion and Limitations

We showed that a persistent spatial semantic representation enables a hierarchical model to achieve state-of-the-art performance on a challenging instruction-following mobile manipulation task. The main performance bottlenecks include long-horizon exploration, perception generalization to unseen environments, and low-level motion planning for continuous collision avoidance. In terms of learning, incorporating reinforcement learning to train π^H, π^L, and observation model F jointly could improve robustness. We defined the interface to π^L to be faithful to skills available on physical robots, but the exact implementation of π^L is not the focus of our work. Physical deployment

Method	Validation Seen SR	Validation Seen GC	Validation Unseen SR	Validation Unseen GC
HLSM	29.6	38.8	18.3	**31.2**
+ gt depth	29.6	40.5	20.1	33.7
+ gt depth, gt seg.	40.7	50.4	40.2	52.2
+ gt seg.	36.2	47.0	34.7	47.8
w/o language enc.	0.9	8.6	0.2	7.5
w/o subg. hist. enc.	29.4	38.5	16.6	29.2
w/o state repr enc.	30.0	40.6	**18.9**	30.8
w/o exploration	**32.2**	**42.4**	18.1	31.3

Table 2: Development results on validation split. Performance of our full approach, with perception oracles, a perception ablation, π^H ablations, and π^L ablations

Task Type	Validation Seen SR	Validation Seen GC	Validation Unseen SR	Validation Unseen GC
Overall	29.6	38.7	18.3	31.2
Examine	46.8	59.0	36.6	59.9
Pick & Place	57.0	57.0	34.8	34.8
Stack & Place	13.0	27.0	4.4	14.3
Clean & Place	25.0	39.5	11.3	25.8
Cool & Place	17.5	33.8	14.8	39.6
Heat & Place	9.3	29.1	0.0	17.0
Pick 2 & Place	34.7	51.9	18.0	34.7

Table 3: Performance breakdown per task type on the validation split.

would require changes to π^L, and study on robustness to errors in continuous environments, such as localization or motion uncertainty.

9 Acknowledgements

This research was supported by ARO W911NF-21-1-0106, a Google Focused Award, and NSF under grant No. 1750499. Animesh Garg is supported in part by CIFAR AI Chair and NSERC Discovery Grant. A significant part of the work was done during the first author's internship at Nvidia. We thank the authors of ALFRED for maintaining the benchmark. We thank Mohit Shridhar and Jesse Thomason for their help answering our questions, and the anonymous reviewers for their helpful comments.

References

[1] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. Gopal Banerjee, S. Teller, and N. Roy. "Approaching the Symbol Grounding Problem with Probabilistic Graphical Models. *AI Magazine*, 2011.

[2] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox. Learning to parse natural language commands to a robot control system. In *ISER*, 2012.

[3] D. K. Misra, J. Sung, K. Lee, and A. Saxena. Tell me dave: Context-sensitive grounding of natural language to mobile manipulation instructions. In *RSS*, 2014.

[4] J. Thomason, S. Zhang, R. J. Mooney, and P. Stone. Learning to interpret natural language commands through human-robot dialog. In *IJCAI*, 2015.

[5] D. Misra, J. Langford, and Y. Artzi. Mapping instructions and visual observations to actions with reinforcement learning. In *EMNLP*, 2017.

[6] D. Nyga, S. Roy, R. Paul, D. Park, M. Pomarlan, M. Beetz, and N. Roy. Grounding robot plans from natural language instructions with incomplete world knowledge. In *CoRL*, 2018.

[7] V. Blukis, N. Brukhim, A. Bennet, R. Knepper, and Y. Artzi. Following high-level navigation instructions on a simulated quadcopter with imitation learning. In *RSS*, 2018.

[8] V. Blukis, D. Misra, R. A. Knepper, and Y. Artzi. Mapping navigation instructions to continuous control actions with position-visitation prediction. In *CoRL*, 2018.

[9] V. Blukis, Y. Terme, E. Niklasson, R. A. Knepper, and Y. Artzi. Learning to map natural language instructions to physical quadcopter control using simulated flight. In *CoRL*, 2019.

[10] S. Patki, E. Fahnestock, T. M. Howard, and M. R. Walter. Language-guided semantic mapping and mobile manipulation in partially observable environments. In *CoRL*, 2019.

[11] V. Blukis, R. A. Knepper, and Y. Artzi. Few-shot object grounding and mapping for natural language robot instruction following. In *CoRL*, 2020.

[12] M. Shridhar, J. Thomason, D. Gordon, Y. Bisk, W. Han, R. Mottaghi, L. Zettlemoyer, and D. Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *CVPR*, 2020.

[13] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *CVPR*, 2018.

[14] H. Tan, L. Yu, and M. Bansal. Learning to navigate unseen environments: Back translation with environmental dropout. In *NAACL-HLT*, 2019.

[15] P. Anderson, A. Shrivastava, J. Truong, A. Majumdar, D. Parikh, D. Batra, and S. Lee. Sim-to-real transfer for vision-and-language navigation. In *CoRL*, 2020.

[16] B. Kim, S. Bhambri, K. P. Singh, R. Mottaghi, and J. Choi. Abp, alfred leaderboard, may 10th 2021. https://leaderboard.allenai.org/alfred/submission/c2t70j37q4q5ci4so89g. Accessed: June 16th, 2021.

[17] A. Pashevich, C. Schmid, and C. Sun. Episodic Transformer for Vision-and-Language Navigation, 2021.

[18] Anonymous. Lwit, alfred leaderboard, may 10th 2021. https://leaderboard.allenai.org/alfred/submission/bvppcin94ro4j7j0jqlg. Accessed: May 25th, 2021.

[19] M. R. Walter, S. Hemachandra, B. Homberg, S. Tellex, and S. Teller. Learning Semantic Maps from Natural Language Descriptions. In *RSS*, 2013.

[20] S. Hemachandra, F. Duvallet, T. M. Howard, N. Roy, A. Stentz, and M. R. Walter. Learning models for following natural language directions in unknown environments. In *ICRA*, 2015.

[21] S. Patki, A. F. Daniele, M. R. Walter, and T. M. Howard. Inferring compact representations for efficient natural language understanding of robot instructions. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6926–6933. IEEE, 2019.

[22] I. Kostavelis and A. Gasteratos. Semantic mapping for mobile robotics tasks: A survey. *Robotics and Autonomous Systems*, 2015.

[23] H. Saha, F. Fotouhi, Q. Liu, and S. Sarkar. A modular vision language navigation and manipulation framework for long horizon compositional tasks in indoor environment. *arXiv preprint arXiv:2101.07891*, 2021.

[24] T. Kollar, S. Tellex, D. Roy, and N. Roy. Toward understanding natural language directions. In *HRI*, 2010.

[25] C. Matuszek, N. FitzGerald, L. Zettlemoyer, L. Bo, and D. Fox. A Joint Model of Language and Perception for Grounded Attribute Learning. In *ICML*, 2012.

[26] S. Tellex, R. Knepper, A. Li, D. Rus, and N. Roy. Asking for help using inverse semantics. In *RSS*, 2014.

[27] R. A. Knepper, S. Tellex, A. Li, N. Roy, and D. Rus. Recovering from Failure by Asking for Help. *Autonomous Robots*, 2015.

[28] Z. Gong and Y. Zhang. Temporal spatial inverse semantics for robots communicating with humans. In *ICRA*, 2018.

[29] T. Brick and M. Scheutz. Incremental natural language processing for hri. In *HRI*, 2007.

[30] S. Tellex, P. Thaker, R. Deits, D. Simeonov, T. Kollar, and N. Roy. Toward information theoretic human-robot dialog. *Robotics*, 2013.

[31] K. M. Hermann, F. Hill, S. Green, F. Wang, R. Faulkner, H. Soyer, D. Szepesvari, W. Czarnecki, M. Jaderberg, D. Teplyashin, et al. Grounded language learning in a simulated 3d world. *arXiv preprint arXiv:1706.06551*, 2017.

[32] D. S. Chaplot, K. M. Sathyendra, R. K. Pasumarthi, D. Rajagopal, and R. Salakhutdinov. Gated-attention architectures for task-oriented language grounding. *AAAI*, 2018.

[33] C. Paxton, Y. Bisk, J. Thomason, A. Byravan, and D. Fox. Prospection: Interpretable plans from language by predicting the future. In *ICRA*, 2019.

[34] A. Suhr and Y. Artzi. Situated mapping of sequential instructions to actions with single-step reward observation. In *ACL*, 2018.

[35] D. Fried, R. Hu, V. Cirik, A. Rohrbach, J. Andreas, L.-P. Morency, T. Berg-Kirkpatrick, K. Saenko, D. Klein, and T. Darrell. Speaker-follower models for vision-and-language navigation. In *NeurIPS*, 2018.

[36] D. Misra, A. Bennett, V. Blukis, E. Niklasson, M. Shatkin, and Y. Artzi. Mapping instructions to actions in 3D environments with visual goal prediction. In *EMNLP*, 2018.

[37] C.-Y. Ma, J. Lu, Z. Wu, G. AlRegib, Z. Kira, R. Socher, and C. Xiong. Self-monitoring navigation agent via auxiliary progress estimation. In *ICLR*, 2019.

[38] H. Chen, A. Suhr, D. Misra, and Y. Artzi. Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *CVPR*, 2019.

[39] D. K. Misra, K. Tao, P. Liang, and A. Saxena. Environment-driven lexicon induction for high-level instructions. In *ACL*, 2015.

[40] M. MacMahon, B. Stankiewicz, and B. Kuipers. Walk the talk: Connecting language, knowledge, and action in route instructions. In *AAAI*, 2006.

[41] S. R. K. Branavan, L. S. Zettlemoyer, and R. Barzilay. Reading between the lines: Learning to map high-level instructions to commands. In *ACL*, 2010.

[42] S. Tellex, T. Kollar, S. Dickerson, M. Walter, A. Banerjee, S. Teller, and N. Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*, 2011.

[43] F. Duvallet, T. Kollar, and A. Stentz. Imitation learning for natural language direction following through unknown environments. In *ICRA*, 2013.

[44] Y. Artzi and L. Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *TACL*, 2013.

[45] E. C. Williams, N. Gopalan, M. Rhee, and S. Tellex. Learning to parse natural language to grounded reward functions with weak supervision. In *ICRA*, 2018.

[46] K. P. Singh, S. Bhambri, B. Kim, R. Mottaghi, and J. Choi. Moca: A modular object-centric approach for interactive instruction following. *arXiv preprint arXiv:2012.03208*, 2020.

[47] Y. Zhang and J. Chai. Hierarchical task learning from language instructions with unified transformers and self-monitoring. *ACL Findings*, 2021.

[48] D. Gordon, A. Kembhavi, M. Rastegari, J. Redmon, D. Fox, and A. Farhadi. Iqa: Visual question answering in interactive environments. In *CVPR*, 2018.

[49] P. Anderson, A. Shrivastava, D. Parikh, D. Batra, and S. Lee. Chasing ghosts: Instruction following as bayesian state tracking. In *NeurIPS*, 2019.

[50] S. Hemachandra, M. R. Walter, S. Tellex, and S. Teller. Learning spatial-semantic representations from natural language descriptions and scene classifications. In *ICRA*, 2014.

[51] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

[52] J. McCormac, A. Handa, S. Leutenegger, and A. J. Davison. Scenenet rgb-d: Can 5m synthetic images beat generic imagenet pre-training on indoor segmentation? In *ICCV*, 2017.

[53] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

[54] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.

[55] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

[56] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[57] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel. Value iteration networks. In *NeurIPS*, 2016.

[58] M. Shridhar and D. Hsu. Interactive visual grounding of referring expressions for human-robot interaction. In *RSS*, 2018.

[59] M. Shridhar, J. Thomason, D. Gordon, Y. Bisk, W. Han, R. Mottaghi, L. Zettlemoyer, and D. Fox. Alfred leaderboard. `https://leaderboard.allenai.org/alfred/`. Accessed: June 16th, 2021.

[60] K. Nottingham, L. Liang, D. Shin, C. C. Fowlkes, R. Fox, and S. Singh. Modular framework for visuomotor language grounding. *arXiv preprint arXiv:2109.02161*, 2021.

[61] B. Kim, S. Bhambri, K. P. Singh, R. Mottaghi, and J. Choi. Agent with the big picture: Perceiving surroundings for interactive instruction following. In *Embodied AI Workshop CVPR*, 2021.

[62] V.-Q. Nguyen, M. Suganuma, and T. Okatani. Look wide and interpret twice: Improving performance on interactive instruction-following tasks. In *IJCAI*, 2021.

[63] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis. Learning navigation behaviors end-to-end with autorl. *RA-L*, 2019.

[64] M. Sundermeyer, A. Mousavian, R. Triebel, and D. Fox. Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes. *arXiv preprint arXiv:2103.14127*, 2021.

[65] D. Meagher. Geometric modeling using octree encoding. *Computer graphics and image processing*, 1982.

[66] T. Takikawa, J. Litalien, K. Yin, K. Kreis, C. Loop, D. Nowrouzezahrai, A. Jacobson, M. McGuire, and S. Fidler. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In *CVPR*, 2021.

[67] H. P. Grice. Logic and conversation. In *Speech acts*. 1975.

[68] J. Roh, C. Paxton, A. Pronobis, A. Farhadi, and D. Fox. Conditional driving from natural language instructions. In *CoRL*, pages 540–551, 2020.

[69] C. Matuszek, D. Fox, and K. Koscher. Following directions using statistical machine translation. In *HRI*, 2010.

[70] N. Gopalan, D. Arumugam, L. L. Wong, and S. Tellex. Sequence-to-sequence language grounding of non-markovian task specifications. In *RSS*, 2018.

[71] D. Bahdanau, F. Hill, J. Leike, E. Hughes, A. Hosseini, P. Kohli, and E. Grefenstette. Learning to understand goal specifications by modelling reward. In *ICLR*, 2018.

[72] P. Goyal, S. Niekum, and R. J. Mooney. Pixl2r: Guiding reinforcement learning using natural language by mapping pixels to rewards. *CoRL*, 2020.

[73] J. Krantz, E. Wijmans, A. Majumdar, D. Batra, and S. Lee. Beyond the nav-graph: Vision-and-language navigation in continuous environments. *arXiv preprint arXiv:2004.02857*, 2020.

[74] V. Jain, G. Magalhaes, A. Ku, A. Vaswani, E. Ie, and J. Baldridge. Stay on the path: Instruction fidelity in vision-and-language navigation. In *ACL*, 2019.

[75] A. Ku, P. Anderson, R. Patel, E. Ie, and J. Baldridge. Room-across-room: Multilingual vision-and-language navigation with dense spatiotemporal grounding. In *EMNLP*, 2020.

[76] M. Persson, T. Duckett, C. Valgren, and A. Lilienthal. Probabilistic semantic mapping with a virtual sensor for building/nature detection. In *Computational Intelligence in Robotics and Automation*, 2007.

[77] H. Zender, O. M. Mozos, P. Jensfelt, G.-J. Kruijff, and W. Burgard. Conceptual spatial representations for indoor mobile robots. *Robotics and Autonomous Systems*, 2008.

[78] A. Pronobis, O. Martinez Mozos, B. Caputo, and P. Jensfelt. Multi-modal semantic place classification. *IJRR*, 2010.

[79] A. Pronobis. *Semantic mapping with mobile robots*. PhD thesis, KTH Royal Institute of Technology, 2011.

A Appendix

A.1 Frequently Asked Questions

- **Are the ALFRED sequential instructions needed during training?** The sequential step-by-step instructions are not needed neither during training, nor at test-time.
- **What has to be done to apply this approach to a real robot?** The observation model, high-level controller, state representation, and the interface to the low-level controller together constitute our contribution and are intended to generalize to physical robots. Deployment on a real robot would require an implementation of the low-level controller designed for continuous motion in cluttered environments, and an implementation of the ALFRED interface to enable execution of manipulation actions such as PICKUP and TOGGLEON. Such physical robot capabilities are subject of ongoing research [63, 64].
- **Does this simulated environment result constitute progress towards real-world capabilities?** Real-robot operation is the long-term motivation of this work and has been carefully considered in the design of the representation and the approach. However, we do not claim to execute high-level natural language mobile manipulation instructions from raw vision on real robots in unseen environments. To date, such capabilities haven't been demonstrated even in simulated environments, such as ALFRED. Even in this scenario, though our method achieves better results than existing work, it can still only solve 18.28% of problems in unseen environments.
- **Would the system scale to physically larger environments?** The main bottleneck towards scaling to larger environments is the memory constraint of the semantic memory. While our implementation is likely restricted to interior scenes when using commodity hardware, follow-up work could address this, perhaps using multi-scale representations such as Octress [65, 66].
- **How are the state dynamics modeled? Are they assumed to be known or are they learned?** The GOTO procedure in the low-level controller is based on a value-iteration network that utilizes a deterministic grid-navigation dynamics model on the internal representation, which is a crude approximation of the dynamics of the ROTATELEFT, ROTATERIGHT, MOVEAHEAD navigation commands. Other than that, the dynamics of the environment are assumed to be completely unknown to the agent, and are not explicitly learned or modeled.
- **How would localization uncertainty affect the approach?** Our representation approach assumes a reliable robot pose estimate. Precisely studying the effects of pose errors would require integration into a system for continuous environments. Intuitively, voxels further away are affected by pose errors more, but may better tolerate it due to being used mainly to decide navigation goals. Voxels close to the agent require more precision as they are used for object instance mask generation, but would be less affected by pose errors. Our voxel map uses a relatively coarse 25cm resolution.
- **Which model was used to obtain test results?** The full HLSM model was evaluated on the test set, even though the model without state representation encoding input to the high-level controller performed better in unseen environments on the validation set.
- **Why does the ablation without state representation encodings perform better in unseen environments?** In unseen environments, the semantic segmentation is erroneous due to the generalization gap, resulting in state encodings that contain errors. This may affect perfromance of the high-level controller that was trained on data with perfect segmentation, and thus with perfect state encodings.
- **What is the benefit of sampling the subgoals instead of attempting execution from most to least likely in order?** There are two types of subgoal execution failures: systematic and random. An example of a systematic failure is the selection of an incorrect subgoal. For example, TOGGLEON(FLOORLAMP) would fail if a FLOORLAMP does not

exist in the environment. An example of a random failure is the low-level controller sampling an interaction pose for which the interaction fails (e.g., Figure 4, row 1, timestep 272). A next-best approach would alleviate a systematic failure, but a sampling approach alleviates both: the systematic failures by trying different subgoals, and random failures by potentially sampling the same subgoal multiple times.

A.2 Extended Related Work

Grounding High-level Language to Actions in Robotics In order for natural language human-robot interfaces to be useful and widely adopted in practice, they should support instructions that are as brief as possible while still being informative of the task, i.e., that adhere to Grice's maxim of quantity [67]. Following such high-level instructions requires bridging the gap from high-level language to long sequences of low-level actions. This is commonly achieved using temporal abstraction, where subgoals or options abstract over sequences of low-level actions, reducing the effective time horizon of the problem. Most work on instruction following in robotics utilizes temporal abstraction [1, 3, 10, 20, 21, 33, 36, 39, 42, 44, 68].

Various methods explicitly model correspondences between linguistic constituents in a symbolic instruction representation, environment percepts in the world model, and subgoals (behavior primitives) [1, 3, 20, 24, 39, 69]. This requires the instruction to at least mention each subgoal, and precludes instructions that omit intermediate goals that are expected to be inferred. This limitation can be overcome by directly mapping from language to reward specifications [45, 70–72] or post-conditions [39], and then using a planner [39] or learning a task-specific policy [71, 72] to solve for the sequence of actions. Both are difficult in practice. Planning requires a compact, symbolic environment representation with an underlying ontology that is hard to construct for unstructured environments, such as the household environment studied in this work. Policy learning is computationally expensive, and poorly adapts to novel tasks specified in natural language in real-time.

Recently, methods that map language and observations directly to actions using neural networks have seen rising popularity and success on simulated [8, 13, 14, 38, 73–75] and real-robot [9, 11, 15] navigation, as well as simulated manipulation [5] tasks. Simulated mobile manipulation is a promising next frontier [12, 46, 47]. Representation learning approaches avoid planning, by using a direct sequence-to-sequence formulation and a data-driven approach that theoretically permits mapping arbitrarily terse input text to arbitrarily long action sequences that potentially include any necessary intermediate steps not explicitly mentioned in the text. In practice, however, most research has focuses on relatively detailed step-by-step instructions, sometimes using modelling tools such as attention [13, 17] and progress monitoring [37] to leverage the sequential nature of the instructions.

We learn to follow high-level instructions in an interactive mobile manipulation 3D environment. To bridge the gap between language and actions, we use temporal abstraction, where the high-level controller predicts subgoals that abstract over sequences of actions, and the low-level controller generates actions to fulfil each subgoal. The controllers rely on a spatial-semantic state representation to enable reasoning about what subgoals make progress towards the high-level task, and what actions make progress towards the specific subgoal, given all past sensory observations. The persistent representation enables operation over long time horizons. Using a shared world representation for both the high- and low-level controllers reduces representation engineering effort and error accumulation typically associated with pipeline approaches.

Semantic Maps for Language Grounding in Robotics The idea of building maps that combine spatial and semantic information [19, 50, 76–79] and using them for following natural language instructions [7, 9, 10, 20, 21, 49] has a long history in robotics. Common approaches can be classified into sparse topological and dense grid-based maps.

Walter et al. [19] introduced a sparse semantic graph that combines pose, semantic, and topological information, extracted from sensory observations and speech descriptions along a route. Hemachandra et al. [50] added a spatial map layer, and fused language with other sensory modalities. Hemachandra et al. [20] used these representations for grounding natural language route instructions. More recently, Patki et al. [21] extended this framework to build compact world models specific to the input instruction, and Patki et al. [10] enabled supporting previously unseen environments. This class of sparse topological maps are well suited for probabilistic language grounding from symbolic representations.

Dense grid-based 2D semantic maps are suited for downstream processing using learned neural network modules, and have been used in modular neural network approaches for language grounding [8, 9, 23, 48, 49]. Saha et al. [23] used a grid-based spatial representation and a map filtering method, showing promising early results on a subset of the ALFRED dataset. We extend this line of work to 3D voxel maps, add explicit tracking of occupancy and observability, and maintain the representation through time to facilitate grounding high-level language over long time horizons. Our dense representation has a number of advantages. First, it is easy to build in real-time from RGBD data using segmentation models and geometric operations. Second, it captures structures found in indoor environments, such as L-shaped countertops or kitchen islands with sinks that are hard to represent topologically. Third, it encodes spatial object relationships without requiring an ontology of spatial relations, or even tracking of object instances. The main limitation of our approach is a memory footprint that scales with the physical size of the environment, making it less suited for outdoor or field applications. Follow-up work could address this limitation, for example by using multi-scale representations such as octrees [65, 66].

Detailed comparison to HiTUT We provide a detailed technical comparison between our approach and HiTUT [Hierarchical Task Learning from Language Instructions with Unified Transformers and Self-Monitoring; 47], our main point of comparison.

Both our approach and HiTUT use a hierarchical task decomposition of goals into sequences of subgoals, and subgoals into sequences of actions. The set of subgoals assumed by our approach and HiTUT have differences. HiTUT has an additional subgoal GoTo(Location), while we view any navigation as a means to an end of a manipulation subgoal, and therefore do not have an explicit GoTo subgoal. HiTUT additionally has subgoals for Clean and Heat, (e.g., Clean(Obj) usually abstracts over the sequence Put(Sink), ToggleOn(Faucet), ToggleOff(Faucet), Pickup(Obj)), while our high-level policy would have to predict this entire sequence.

In terms of the model architecture, we use a hierarchical model with high-level and low-level controllers to mimic the task structure. In contrast, HiTUT uses a flat transformer model to jointly solve high-level subgoal planning and low-level action prediction. One of their main contributions is showing how a flat transformer model can be used to model a hierarchical task structure. The benefit of our hierarchical model decomposition in combination with a shared spatial state representation is its ability to solve low-level navigation and manipulation problems with specialized modules, while avoiding the representational error accumulation and representation engineering issues typically associated with modular pipeline approaches.

In terms of inference, HiTUT and our approach both sample subgoals one at a time, dynamically responding to changes in environment and execution. Both approaches perform backtracking to previous subgoals upon subgoal failure.

In terms of perception, our approach requires a pre-trained segmentation model, while HiTUT requires a pre-trained object detection model to generate object entity information that is fed into the transformer.

A.3 Observation Model Details

At time t, during the perception step, we predict first-person semantic segmentation I_t^S and depth I_t^D from the observation $o_t = (I_t, P_t, v_t^S, L)$, from the RGB image I_t with neural network models pre-trained in the ALFRED environment. Each pixel $[I_t^S]_{(u,v)}$ at coordinates (u, v) is a distribution over C object classes. Likewise, $[I_t^D]_{(u,v)}$ is a distribution over B uniformly spaced depth bins $\{0, \Delta_D, 2\Delta_D, \ldots, (B-1)\Delta_D\}$, where Δ_D is a depth resolution. In early experiments, we observed that Δ_D should be less than 50% of the voxel size. We use $\Delta_D = 0.1m$, $B = 50$, and voxel size of $0.25m$. We also heuristically compute a binary mask M_t^D that indicates which pixels have confident

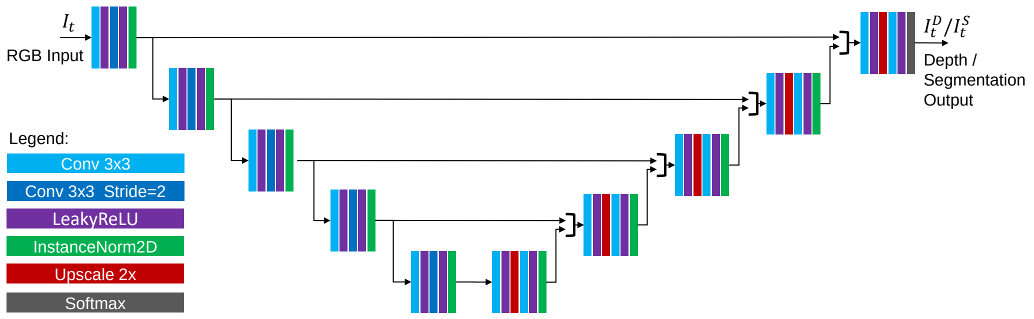

Figure 5: Illustration of the U-Net architecture used in the depth and segmentation networks.

depth readings. We allow more confidence slack in pixels that correspond to the current subgoal argument $\texttt{arg}_t^C$ according to I_t^S. The mask M_t^D is used in the projection step to discard points (x, y, z) that correspond to pixels (u, v) for which $[M^D]_{(u,v)} = 0$. The mask computation is:

$$M_t^D = (W_{90}[I_t^D] < c_1\,\mathbb{E}[I_t^D]) \vee ((W_{90}[I_t^D] < c_2\,\mathbb{E}[I_t^D]) \wedge ([I_t^S]_{\texttt{arg}_t^C} > 0.5)) \quad , \tag{5}$$

where $W_{90}[I_t^D]$ is the width of the 90% confidence interval at each pixel, $\mathbb{E}[I_t^D]$ is the expected depth at each pixel, and $[I_t^S]_{\texttt{arg}_t^C}$ is a 0-1 valued segmentation mask of the class of the current subgoal argument. We set the hyperparameters $c_1 = 0.3$ and $c_2 = 1.0$ to allow higher depth uncertainty for points corresponding to the subgoal argument.

If the agent is currently holding an object (i.e. $\sum_i [[v_t^S]_{(i)}] > 0$), we also discard points closer than $0.7m$ to the camera to make sure that the object in the agent inventory does not get added to the voxel map.

We use custom models based on the U-Net architecture [51] for depth and segmentation networks. The architecture is illustrated in Figure 5. It consists of a cascade of five downscale blocks followed by five upscale blocks with skip-connections. Each block includes two convolutions, two leakyReLU activations, and an instance normalization layer. The upscale blocks contain a 2x spatial upscaling operation. We found that training a separate network for depth and segmentation worked better than sharing one network for both tasks.

A.4 Model Execution Flow

Algorithm 1 Execution Flow

```
Input: Instr. $L$, Horizon $T_{max}$.
 1: $g_{0,1,2...}, \hat{s}_{-1} \leftarrow$ null
 2: $k \leftarrow 1$
 3: Observe initial $o_0$.
 4: for $t = 0, 1, 2, \dots H$ do
 5:   $\hat{s}_t \leftarrow F(\hat{s}_{t-1}, o_t, g_k)$
 6:   do
 7:     if $g_k =$ null then
 8:       $g_k \sim \pi^H(L, \hat{s}_t, \langle g_i \rangle_{i<k})$
 9:       if $g_k = g_{\text{STOP}}$ then
10:         End episode
11:     $a_t \sim \pi^L(g_k, \hat{s}_t)$
12:     if $a_t = a_{\text{PASS}}$ then
13:       $k \leftarrow k + 1$
14:     if $a_t = a_{\text{FAIL}}$ then
15:       $g_k \leftarrow$ null
16:   while $a_t \in \{a_{\text{FAIL}}, a_{\text{PASS}}\}$
17:   Perform $a_t$, observe $o_{t+1}$
18: End episode
```

Algorithm 1 describes the execution flow. At time $t = 0$ the initial observation o_0 is received. At each timesep, we update the state representation $\hat{s}_t$ (Line 5). If needed, we sample a new subgoal

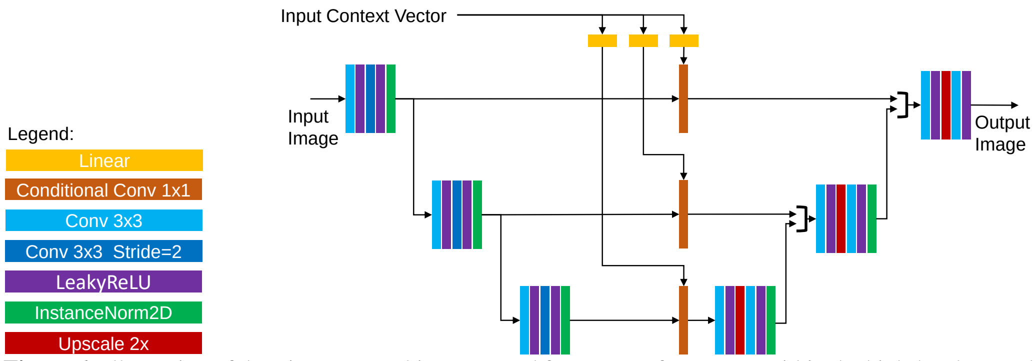

Figure 6: Illustration of the LingUNet architecture used for as part of REFINER within the high-level controller π^H, and as part of the navigation model NAVMODEL within the low-level controller. The conditional convolutions parameters are computed during the network forward pass.

g_k from π^H (Line 8), and then sample an action a_t from π^L. If a_t is a_{PASS}, we increment subgoal counter k (Line 13). If it is a_{FAIL}, we discard the current subgoal k (Line 15). We repeat Lines 8–15 until an executable action a_t is sampled. We execute a_t, receive the next observation (Line 17), and proceed to the next timestep. The episode ends when the subgoal g_{STOP} is sampled (Line 10) or the horizon T_{max} is exceeded (Line 18).

A.5 High-Level Controller Details

Subgoals are predicted periodically. Let $g_k = (\texttt{type}_k, \texttt{arg}_k^C, \texttt{arg}_k^M)$ be the k-th subgoal predicted at time t. Predicting the subgoal type $\texttt{type}_k$ and the argument class $\texttt{arg}_k^C$ is described in the main paper (Section 4.3). This section provides further details of REFINER, the model we use to generate $\texttt{arg}_k^M$. The mask refiner REFINER has four inputs:[4] (a) a spatial feature map $\mathbf{x}_t^{ego} \in [0,1]^{N\times W\times L}$ oriented in the agent egocentric reference frame; (b) $[V_t^S]_{(\texttt{arg}_k^C)} \in [0,1]^{W\times L\times H}$, a 3D mask indicating all voxels that contain objects of class $\texttt{arg}_k^C$ in the voxel map V_t^S; (c) the agent's pose P_t; and (d) a vector representation of the instruction ϕ^L. It outputs a 3D mask $\texttt{arg}_k^M \in [0,1]^{W\times L\times H}$ that identifies the subgoal argument object. Formally, the computation is:[5]

$$\begin{aligned}\textsc{Refiner}(\mathbf{x}_t^{ego}, [V_t^S]_{(\texttt{arg}_k^C)}, P_t, \phi^L) = \\ \textsc{AlloTransform}(\textsc{LingUNet}_m(\mathbf{x}_t^{ego}, \phi^L), P_t) \otimes [V_t^S]_{(\texttt{arg}_k^C)}\ ,\end{aligned} \tag{6}$$

where ALLOTRANSFORM transforms a spatial 2D map from an egocentric to the global reference frame, and LINGUNET$_m$ is a language-conditioned image-to-image encoder-decoder [36]. The architecture of LINGUNET$_m$ is illustrated in Figure 6.

A.6 Low-Level Controller Details

We describe the implementation of each of the low-level controller procedures. This implementation is not the focus of this paper, and could be improved or replaced with other algorithms. Some of the procedures cause actions in the AI2Thor environment, others simply process data to pass between procedures.

The procedures are `NavigateTo`, `SampleExplorationPosition`, `SampleInteractionPose`, and `InteractMask`. The low-level controller receives the subgoal g_k, and follows a pre-specified execution flow across multiple timesteps to complete it. The execution flow (Figure 7) consists of an *exploration* and *interaction* phase. In the exploration phase, we perform a 360° rotation by generating a sequence of three ROTATELEFT actions to observe the environment and add information to the semantic map. If the semantic map indicates that no object of type $\texttt{arg}_k^C$, the action argument, is observed, we explore the environment by sampling a position $(x, y) = \texttt{SampleExplorationPosition}(\hat{s}_t)$, navigating there using $\texttt{NavigateTo}(x, y, \hat{s}_t)$, and

[4]Errata: Equation 4 in the main paper is missing $[V_t^S]_{(\texttt{arg}_k^C)}$ and P_t arguments to the REFINER.

[5]$\otimes$ is an operation that multiplies a $W \times L$ matrix by a $W \times L \times H$ tensor to obtain a $W \times L \times H$ tensor

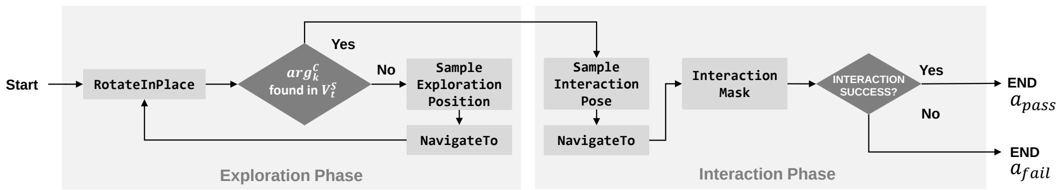

Figure 7: Illustration of the low-level controller execution flow, showing the order in which procedures are used to complete a subgoal g_k.

performing another 360° rotation. We repeat this process until a voxel in V_t^S contains the class $\texttt{arg}_k^C$ with >50% probability, at which point we move on to the interaction phase. In the interaction phase, we sample an interaction pose $(x, y, \omega_y, \omega_p) = \texttt{SampleInteractionPose}(\hat{s}_t, g_k)$, invoke $\texttt{NavigateTo}(x, y, \hat{s}_t)$ to reach the position (x,y), and rotate according to yaw and pitch angles (ω_y, ω_p). Finally, we generate the egocentric interaction action mask $\texttt{mask}_t = \texttt{InteractMask}(\hat{s}_t, \texttt{arg}_k^M)$, and execute the interaction action $(\texttt{type}_k, \texttt{mask}_t)$ in the ALFRED environment. We output a_{PASS} or a_{FAIL} depending if the interaction action has succeeded, and pass control back to the high-level controller to sample the next subgoal.

A.6.1 NavigateTo Procedure

At time t, the $\texttt{NavigateTo}$ procedure maps a 2D navigation goal position (x, y) and the state representation $\hat{s}_t$ to one of the actions: $\{\textsc{RotateLeft}, \textsc{RotateRight}, \textsc{MoveAhead}, a_{\text{STOP}}\}$. We implement it with a Value Iteration Network [VIN; 57] that solves a 2D grid-MDP to predict navigation actions using fast GPU-accelerated convolution and max-pooling operations. The VIN parameters are pre-defined, and not learned. Other motion planners such as A* could be used as well.

The VIN is defined by a state-space $\mathcal{S}^{vin}$, action space $\mathcal{A}^{vin}$, transition function $\mathcal{T}^{vin} : \mathcal{S}^{vin} \times \mathcal{A}^{vin} \to \mathcal{S}^{vin}$, a reward function $R^{vin} : \mathcal{S}^{vin} \times \mathcal{A}^{vin} \to \mathcal{A}^{vin}$, and terminal state set $\mathcal{M}^{vin}$. At each timestep t, VIN performs value iteration to compute the Q-function $Q^{vin} : \mathcal{S}^{vin} \times \mathcal{A}^{vin} \to \mathbb{R}$ that estimates the expected sum of future discounted rewards for taking action $a_t^{vin} \in \mathcal{A}^{vin}$ in state $s_t^{vin} \in \mathcal{S}^{vin}$, and thereafter following a greedy policy: $a_i^{vin} = \arg\max_{a^{vin} \in \mathcal{A}^{vin}} Q(s_i^{vin}, a^{vin})$, $i > t$. We implement the state space $\mathcal{S}^{vin}$ as a 2D grid of shape $W^{vin} \times H^{vin}$. Each state s^{vin} is tagged with three 0-1 valued attributes: $\textsc{Obstacle}, \textsc{Unobserved}, \textsc{Goal}$. At each timestep t, we set the values of state attributes according to the most recent state representation $\hat{s}_t$ and current navigation goal (x, y). States s^{vin} with occupied voxels in the height range $[0, 1.75m]$ are tagged $\textsc{Obstacle}(s^{vin}) = 1$, otherwise $\textsc{Obstacle}(s^{vin}) = 0$. States with all voxels unobserved are tagged $\textsc{Unobserved}(s^{vin}) = 1$, otherwise $\textsc{Unobserved}(s^{vin}) = 0$. The state at the goal position is tagged $\textsc{Goal}(s^{vin}) = 1$, for all others $\textsc{Goal}(s^{vin}) = 0$. The action space is $\mathcal{A}^{vin} = \{\textsc{North}, \textsc{East}, \textsc{South}, \textsc{West}, \textsc{Stop}\}$. The transition function encodes epsilon-greedy grid navigation dynamics: (1) the action $\textsc{North}$ moves the agent one state north and likewise for other actions, and (2) with probability $\epsilon = 8\%$ a random transition to a neighboring state occurs. Visiting any terminal state $s^{vin} \in \mathcal{M}^{vin}$ or executing the action $\textsc{Stop}$ terminates the episode. Terminal states are all states tagged with attributes $\textsc{Obstacle}$ and $\textsc{Goal}$, $\mathcal{M}^{vin} = \{s^{vin} \in \mathcal{S}^{vin} \mid (\textsc{Obstacle}(s^{vin}) > 0.5) \wedge (\textsc{Goal}(s^{vin}) > 0.5)\}$.

The reward function assigns different rewards for visiting states with different attributes:

$$\begin{aligned} R^{vin}(s^{vin}, a^{vin}) = -0.9 \cdot \textsc{Obstacle}(s^{vin}) + 1.0 \cdot \textsc{Goal}(s^{vin}) \\ - 0.02 \cdot \textsc{Unobserved}(s^{vin}) + 0.001 \cdot \mathbb{1}_{a^{vin} = \textsc{Stop}} \quad . \end{aligned} \tag{7}$$

$\textsc{Obstacle}$ states receive reward -0.9, $\textsc{Goal}$ states receive reward 1.0, and $\textsc{Unobserved}$ states receive reward -0.02. Taking the $\textsc{Stop}$ action in any state gives reward 0.001, which has the effect of the agent stopping in unsolvable cases. We use the VIN iteratively for N^{vin} iterations, and predict an action $a^{vin} = \arg\max_{a^{vin} \in \mathcal{A}^{vin}} (Q^{vin}(s_t^{vin}, a^{vin})$. We map from the VIN action a^{vin} to a single valid AI2Thor navigation action using a deterministic mapping (Table 4).

Current Heading	VIN Action	AI2Thor Action
North	West	RotateLeft
	North	MoveAhead
	East or South	RotateRight
East	North	RotateLeft
	East	MoveAhead
	South or West	RotateRight
South	East	RotateLeft
	South	MoveAhead
	West or North	RotateRight
West	South	RotateLeft
	West	MoveAhead
	North or East	RotateRight

Table 4: Mapping from VIN actions to AI2Thor actions.

A.6.2 SampleExplorationPosition

The `SampleExplorationPosition` procedure maps a state representation $\hat{s}_t$ to a discrete 2D position $p^{\text{explore}} = (x, y)$. Let $\mathcal{P}_s$ be the set of 2D positions corresponding to voxel centroids in the voxel map along the horizontal axes, and the ground set $\mathcal{P}_g$ as the set of all unoccupied positions that have the class FLOOR or RUG in at least one voxel. A position is unoccupied if all voxels in the height range $[0, 1.75m]$ are free of obstacles. We define a frontier set $\mathcal{P}_f$ as the set of all positions $\mathcal{P}_g$ for which at least one immediately neighboring position contains zero observed voxels. If $\mathcal{P}_f$ is non-empty, we sample the position p^{explore} uniformly at random from $\mathcal{P}_f$. Otherwise, we sample p^{explore} uniformly at random from $\mathcal{P}_g$.

A.6.3 SampleInteractionPose

The `SampleInteractionPose` procedure maps the state representation $\hat{s}_t$ and subgoal $g_k = (\texttt{type}_k, \texttt{arg}^C_k, \texttt{arg}^M_k)$ to a pose $P = (x, y, \omega_y, \omega_p)$, where (x, y) is a discrete 2D position, ω_y is the agent yaw angle, and ω_p is the agent camera pitch angle. The pose is predicted such that upon reaching it, the interaction action of type $\texttt{type}_k$ is likely to succeed on the object of class $\texttt{arg}^C_k$ at location identified by the mask $\texttt{arg}^M_k$.

We use a neural network model NAVMODEL to predict expected pitch $\mathbb{E}(\omega_p|x, y; g_k, \hat{s}_t)$ and a distribution $P(x, y, \omega_y|g_k, \hat{s}_t)$, factored as:

$$P(x, y, \omega_y|g_k, \hat{s}_t) = P(\omega_y|x, y; g_k, \hat{s}_t)P(x, y|g_k, \hat{s}_t) \tag{8}$$

The network NAVMODEL is based on the LingUNet architecture (Figure 6):

$$\begin{aligned}&\textsc{NavModel}(\hat{s}_t, g_k) = \\ &\qquad \textsc{LingUNet}(\textsc{Afford}(\hat{s}_t), \textsc{Linear}([\textsc{Lut}_T(\texttt{type}_k); \textsc{Lut}_C(\texttt{arg}^C_k)])) \ ,\end{aligned} \tag{9}$$

where AFFORD is an affordance feature map (Section 4.1), LINEAR is a linear layer with bias, $\textsc{Lut}_T$ and $\textsc{Lut}_C$ are embedding lookup tables, and $[\cdot;\cdot]$ is a vector concatenation.

To sample a pose P, we first sample a position $(x, y) \sim P(x, y|g_k, \hat{s}_t)$, then sample a yaw angle $\omega_y \sim P(\omega_y|x, y; g_k, \hat{s}_t)$, and finally lookup a pitch angle $\omega_p = \mathbb{E}(\omega_p|x, y; g_k, \hat{s}_t)$.

A.6.4 InteractionMask

The `InteractionMask` procedure maps a state representation $\hat{s}_t = (V^S_t, V^O_t, v^S_t, P_t)$, the most recent RGB observation I_t, the most recent predicted segmentation I^S_t, and a subgoal $g_k = (\texttt{type}_k, \texttt{arg}^C_k, \texttt{arg}^M_k)$ to a 0-1 valued mask $\texttt{mask}_t \in [0, 1]^{H \times W}$ that identifies the interaction object in the first-person view observation. The interaction mask $\texttt{mask}_t$ is in the format expected by

ALFRED. Formally, it is computed in three steps:

$$\texttt{mask}_t^A = [I_t^S]_{\texttt{arg}_k^C} \tag{10}$$

$$\texttt{mask}_t^B = \textsc{PinholeCam}(\texttt{arg}_k^M, P_t) \tag{11}$$

$$\texttt{mask}_t = \texttt{mask}_t^A \cdot \texttt{mask}_t^B \quad , \tag{12}$$

where PINHOLECAM projects the 0-1 valued 3D voxel map $\texttt{arg}_k^M$ to the agent's camera plane according to the pose P_t. The mask $\texttt{mask}_t^A$ is an egocentric 0-1 valued mask that identifies all objects of class $\texttt{arg}_k^C$ in the image I_t. The $\texttt{mask}_t^B$ is an egocentric 0-1 valued mask that identifies the voxels $\texttt{arg}_k^M$. For each pixel (u, v), the value $[\texttt{mask}_t^B]_{(u,v)}$ is the maximum of all values $[\texttt{arg}_k^M]_{(x,y,z)}$ over voxels with coordinates (x, y, z) that the ray cast from the camera through the pixel (u, v) intersects with. The final mask $\texttt{mask}_t$ is a 0-1 valued mask that identifies not only the correct object class, but also the correct instance according to the voxel mask $\texttt{arg}_k^M$.

A.7 Additional Learning Details

A.7.1 Observation Model Learning

Data As described in Section 5, we use a perception dataset $\mathcal{D}^P$ for training depth and segmentation models. The dataset $\mathcal{D}^P = \{([I]^{(i)}, [I^D]^{(i)}, [I^S]^{(i)}\}_{i=1}^{N_P}$ includes RGB images $[I]^{(i)}$ with ground truth depth $[I^D]^{(i)}$ and segmentation $[I^S]^{(i)}$. The ground truth depth $[I^D]^{(i)}$ at each pixel (u, v) is a distribution $[I^D]^{(i)}_{((u,v))}$ over B depth bins, where 100% of the probability mass is assigned to the bin containing the reference depth value. The ground truth segmentation $[I^S]^{(i)}$ is likewise at each pixel (u, v) a one-hot vector indicating the object class that pixel belongs to.

Data Augmentation The ALFRED dataset consists of 108 different training scenes, where each scene has a fixed furniture and light fixtures. Observations are highly correlated within each scene, which greatly reduces the effective size of the perception dataset and hurts generalization to unseen scenes. We use a custom segmentation-aware data augmentation strategy that increases the diversity of RGB observations.

We compute an augmented RGB image $\tilde{I} = \textsc{Augment}(I, I^S, O_v)$ that maps the image I, ground-truth segmentation I^S, and a set of semantic classes O_v to a new image $\tilde{I}$. O_v is the set of object classes that are likely to appear in different colors. O_v includes classes like FLOOR, COUNTERTOP, CABINET, VASE, SOAPBOTTLE, ALARMCLOCK that come in different designs and colors, but not classes like BANANA, APPLE, SPOON that tend to have even appearance. Algorithm 2 shows the implementation of AUGMENT. It emulates more diverse environments by applying a different random color offset to each object class in the RGB image. Figure 8 shows examples of images produced with this augmentation procedure.

During training, we apply AUGMENT with 50% probability to each training example. Additionally, with 50% probability we perform a horizontal flip.

A.8 Additional Experimental Details

We collect a training dataset of language-demonstration pairs as described in Section 5. The demonstrations in ALFRED typically navigate while looking down at the floor, likely a side-effect of the PDDL planner that had access to the world state during data generation, and as such has no need to explore or observe the visual environment. We modify the demonstration trajectories to get more informative first-person observations. First, we insert four ROTATELEFT actions at the start of each trajectory. Second, we maintain a nominal camera pitch angle of 30°during navigation, by inserting LOOKDOWN and LOOKUP actions before and after every interaction action. We discard trajectories for which these modifications cause failures. These modifications result in observations that are more useful for learning and constructing our persistent spatial representation.

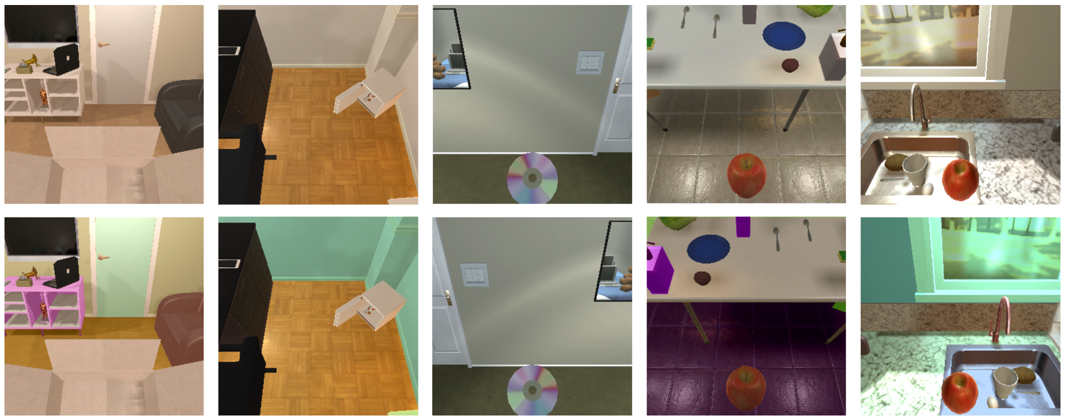
Figure 8: Examples of images produced with our AUGMENT procedure. The top row shows raw RGB images from ALFRED. The bottom row shows images generated by our segmentation-aware data augmentation method. Objects like walls, sinks, floors, and furniture randomly change color, while the apple and the spoons do not.

Algorithm 2 AUGMENT

```
Input: RGB Image $I$, ground truth segmentation $I^S$, set of object classes $O_v$.
 1: $\tilde{I} \leftarrow I$
 2: for $c \in O_v$ do
 3:   » Extract a binary mask corresponding to object class $c$
 4:   $M_c \leftarrow [I^S]_{(c)}$
 5:   » Apply modifications to the image, masked by the segmentation mask $M_c$
 6:   »   $\odot$ multiplies a $N$-dimensional vector with a $H \times W$ tensor to compute a $N \times H \times W$ tensor
 7:   »   $\cdot$ is an elementwise multiplication
 8:   if randomBernoulli(0.5) then
 9:     » Sample an additive color offset for class $c$ from a normal distribution.
10:     »   $I_3$ is the $3 \times 3$ identity matrix.
11:     $a \sim N(\vec{\mathbf{0}}, \sigma_a I_3)$
12:     $\tilde{I} \leftarrow \tilde{I} + a \odot M_c$
13:   if randomBernoulli(0.5) then
14:     » Sample additive gaussian noise for each pixel $(u, v)$ for class $c$
15:     for each pixel $(u, v)$ do
16:       $g_{u,v} \sim N(1, \sigma_g)$
17:       $[\tilde{I}]_{(u,v)} \leftarrow [\tilde{I}]_{(u,v)} + g_{u,v} \cdot [M_c]_{(u,v)}$
18:   if randomBernoulli(0.5) then
19:     » Sample an multiplicative color offset for class $c$ from a normal distribution
20:     $m \sim N(\vec{\mathbf{0}}, \sigma_m I_3)$
21:     $\tilde{I} \leftarrow \tilde{I} \cdot (m \odot M_c)$
22: clamp image $\tilde{I}$ within 0-1 bounds
23: return $\tilde{I}$
```

A.9 Hyperparameters

Table 5 shows hyperparameter values. The hyperparameters were hand-tuned on the validation unseen split.

A.10 Additional Results

Additional qualitative results are available at: `https://hlsm-alfred.github.io/`.

A successful example of task execution is available at: `https://drive.google.com/file/d/1APKe3cR_-vliyU2elT5Un30w7PvkEdYs/view?usp=sharing`

A failed example of task execution is available at: `https://drive.google.com/file/d/1j8BJ_ALoXGyf8a-IOkmQAg38awSWYt6f/view?usp=sharing`

Hyperparameter	**Value**
Observation Model	
Number of depth bins B	50
Depth resolution ΔD	$0.1m$
Spatial State Representation	
Voxel Size	$0.25m$
Voxel Map Dimensions in Voxels	$61 \times 61 \times 10$
Voxel Map Dimensions in Meters	$15.25m \times 15.25m \times 2.5m$
Number of semantic classes C	123
High-level Controller	
Subgoal history encoder hidden dimension	128
Subgoal history encoder transformer layers	2
Subgoal predictor dense MLP layers	3
Subgoal predictor dense MLP hidden dimension	128
Low-level Controller	
Number of VIN iterations N^{vin}	122
VIN state space size	61×61
Development Environment	
Programming Language	Python
ML and Math Library	PyTorch

Table 5: Hyperparameter values.