Look Wide and Interpret Twice: Improving Performance on Interactive Instruction-following Tasks

Van-Quang Nguyen\textsuperscript{1}, Masanori Suganuma\textsuperscript{2,1}, Takayuki Okatani\textsuperscript{1,2}
\textsuperscript{1}Graduate School of Information Sciences, Tohoku University
\textsuperscript{2}RIKEN Center for AIP
\{quang,suganuma,okatani\}@vision.is.tohoku.ac.jp

Abstract

There is a growing interest in the community in making an embodied AI agent perform a complicated task while interacting with an environment following natural language directives. Recent studies have tackled the problem using ALFRED, a well-designed dataset for the task, but achieved only very low accuracy. This paper proposes a new method, which outperforms the previous methods by a large margin. It is based on a combination of several new ideas. One is a two-stage interpretation of the provided instructions. The method first selects and interprets an instruction without using visual information, yielding a tentative action sequence prediction. It then integrates the prediction with the visual information etc., yielding the final prediction of an action and an object. As the object’s class to interact is identified in the first stage, it can accurately select the correct object from the input image. Moreover, our method considers multiple egocentric views of the environment and extracts essential information by applying hierarchical attention conditioned on the current instruction. This contributes to the accurate prediction of actions for navigation. A preliminary version of the method won the ALFRED Challenge 2020. The current version achieves the unseen environment’s success rate of 4.45\% with a single view, which is further improved to 8.37\% with multiple views.

1 Introduction

There is a growing interest in the community in making an embodied AI agent perform a complicated task following natural language directives. Recent studies of vision-language navigation tasks (VLN) have made significant progress [Anderson et al., 2018b; Fried et al., 2018; Zhu et al., 2020]. However, these studies consider navigation in static environments, where the action space is simplified, and there is no interaction with objects in the environment.

To consider more complex tasks, a benchmark named ALFRED was developed recently [Shridhar et al., 2020]. It requires an agent to accomplish a household task in interactive environments following given language directives. Compared with VLN, ALFRED is more challenging as the agent needs to (1) reason over a greater number of instructions and (2) predict actions from larger action space to perform a task in longer action horizons. The agent also needs to (3) localize the objects to manipulate by predicting the pixel-wise masks. Previous studies (e.g., [Shridhar et al., 2020]) employ a Seq2Seq model, which performs well on the VLN tasks [Ma et al., 2019]. However, it works poorly on ALFRED. Overall, existing methods only show limited performance; there is a huge gap with human performance.

In this paper, we propose a new method that leads to significant performance improvements. It is based on several ideas. Firstly, we propose to choose a single instruction to process at each timestep from the given series of instructions. This approach contrasts with previous methods that encode them into a single long sequence of word features and use soft attention to specify which instruction to consider at each timestep implicitly [Shridhar et al., 2020; Yeung et al., 2020; Singh et al., 2020a]. Our method chooses individual instructions explicitly by learning to predict when the agent completes an instruction. This makes it possible to utilize constraints on parsing instructions, leading to a more accurate alignment of instructions and action prediction.

Secondly, we propose a two-stage approach to the interpretation of the selected instruction. In its first stage, the method interprets the instruction without using visual inputs from the environment, yielding a tentative prediction of an action-object sequence. In the second stage, the prediction is integrated with the visual inputs to predict the action to do and the object to manipulate. The tentative interpretation makes it clear to interact with what class of objects, contributing to an accurate selection of objects to interact with.

Moreover, we acquire multiple agent egocentric views of a scene as visual inputs and integrate them using a hierarchical attention mechanism. This allows the agent to have a wider field of views, leading to more accurate navigation. To be specific, converting each view into an object-centric representation, we integrate those for the multiple views into a single feature vector using hierarchical attention conditioned on the current instruction.

Besides, we propose a module for predicting precise pixel-wise masks of objects to interact with, referred to as the mask decoder. It employs the object-centric representation of the center view, i.e., multiple object masks detected by the object...
detector. The module selects one of these candidate masks to specify the object to interact with. In the selection, self-attention is applied to the candidate masks to weight them; they are multiplied with the tentative prediction of the pairs of action and an object class and the detector’s confidence scores for the candidate masks.

The experimental results show that the proposed method outperforms all the existing methods by a large margin and ranks first in the challenge leaderboard as of the time of submission. A preliminary version of the method won the ALFRED Challenge 2020. The present version further improved the task success rate in unseen and seen environments to 8.37% and 29.16%, respectively, which are significantly higher than the previously published SOTA (0.39% and 3.98%, respectively) [Shridhar et al., 2020].

2 Related Work

2.1 Embodied Vision-Language Tasks

Many studies have been recently conducted on the problem of making an embodied AI agent follow natural language directives and accomplish the specified tasks in a three-dimensional environment while properly interacting with it. Vision-language navigation (VLN) tasks have been the most extensively studied, which require an agent to follow navigation directions in an environment.

Several frameworks and datasets for simulating real-world environments have been developed to study the VLN tasks. The early ones lack photo-realism and/or natural language directions [Kempka et al., 2016; Kolbe et al., 2017; Wu et al., 2018]. Recent studies consider perceptually-rich simulated environments and natural language navigation directions [Anderson et al., 2018b; Chen et al., 2019; Hermann et al., 2020]. In particular, since the release of the Room-to-Room (R2R) dataset [Anderson et al., 2018b] that is based on real imagery [Chang et al., 2017], VLN has attracted increasing attention, leading to the development of many methods [Fried et al., 2018; Wang et al., 2019; Ma et al., 2019; Tan et al., 2019; Majumdar et al., 2020].

Several variants of VLN tasks have been proposed. A study [Nguyen et al., 2019] allows the agent to communicate with an adviser using natural language to accomplish a given goal. In a study [Thomason et al., 2020], the agent placed in an environment attempts to find a specified object by communicating with a human by natural language dialog. A recent study [Suhr et al., 2019] proposes the interactive environments where users can collaborate with an agent by not only instructing it to complete tasks, but also acting alongside it. Another study [Krantz et al., 2020] introduces a continuous environment based on the R2R dataset that enables an agent to take more fine-grained navigation actions. A number of other embodied vision-language tasks have been proposed such as visual semantic planning [Zhu et al., 2017; Gordon et al., 2019] and embodied question answering [Das et al., 2018; Gordon et al., 2018; Wijmans et al., 2019; Puig et al., 2018].

2.2 Existing Methods for ALFRED

As mentioned earlier, ALFRED was developed to consider more complicated interactions with environments, which are missing in the above tasks, such as manipulating objects. Several methods for it have been proposed so far. A baseline method [Shridhar et al., 2020] employs a Seq2Seq model with an attention mechanism and a progress monitor [Ma et al., 2019], which is prior art for the VLN tasks. In [Singh et al., 2020a], a pre-trained Mask R-CNN is employed to generate object masks. It is proposed in [Yeung et al., 2020] to train the agent to follow instructions and reconstruct them. In [Corona et al., 2020], a modular architecture is proposed to exploit the compositionality of instructions. These methods have brought about only modest performance improvements over the baseline. A concurrent study [Singh et al., 2020b] proposes a modular architecture design in which the prediction of actions and object masks are treated separately, as with ours. Although it achieves notable performance improvements, the study’s ablation test indicates that the separation of the two is not the primary source of the improvements. Closely related to ALFRED, ALFWorld [Shridhar et al., 2021] has been recently proposed to combine TextWorld [Côté et al., 2018] and ALFRED for creating aligned environments, which enable transferring high-level policies learned in the text world to the embodied world.

3 Proposed Method

The proposed model consists of three decoders (i.e., instruction, mask, and action decoders) with the modules extracting features from the inputs, i.e., the visual observations of the environment and the language directives. We first summarize ALFRED and then explain the components one by one.

3.1 Summary of ALFRED

ALFRED is built upon AI2Thor [Kolve et al., 2017], a simulation environment for embodied AI. An agent performs seven types of tasks in 120 indoor scenes that require interaction with 84 classes of objects, including 26 receptacle object classes. For each object class, there are multiple visual instances with different shapes, textures, and colors.

The dataset contains 8,055 expert demonstration episodes of task instances. They are sequences of actions, whose average length is 50, and they are used as a ground truth action sequence at training time. For each of them, language directives annotated by AMT workers are provided, which consist of a goal statement $G$ and a set of step-by-step instructions, $S_1,...,S_L$. The alignment between each instruction and a segment of the action sequence is known. As multiple AMT workers annotate the same demonstrations, there are 25,743 language directives in total.

We wish to predict the sequence of agent’s actions, given $G$ and $S_1,...,S_L$ of a task instance. There are two types of actions, navigation actions and manipulation actions. There are five navigation actions (e.g., `MoveAhead` and `RotateRight`) and seven manipulation actions (e.g., `Pickup` and `ToggleOn`). The manipulation actions accompany an object. The agent specifies it using a pixel-wise mask.

\[\text{ALFRED Challenge 2020 https://askforalfred.com/EVAL}\]
in the egocentric input image. Thus, the outputs are a sequence of actions with, if necessary, the object masks.

3.2 Feature Representations

Object-centric Visual Representations

Unlike previous studies [Shridhar et al., 2020; Singh et al., 2020a; Yeung et al., 2020], we employ the object-centric representations of a scene [Devin et al., 2018], which are extracted from a pretrained object detector (i.e., Mask R-CNN [He et al., 2017]). It provides richer spatial information about the scene at a more fine-grained level and thus allows the agent to localize the target objects better. Moreover, we make the agent look wider by capturing the images of its surroundings, aiming to enhance its navigation ability.

Specifically, at timestep $t$, the agent obtains visual observations from $K$ egocentric views. For each view $k$, we encode the visual observation by a bag of $N$ object features, which are extracted from the object detector. Every detected object is associated with a visual feature, a mask, and its confidence score. We project the visual feature into $\mathbb{R}^d$ with a linear layer, followed by a ReLU activation and dropout regularization [Srivastava et al., 2014] to obtain a single vector; thus, we get a set of $N$ object features for view $k$, $V^k_t = (v^k_{t,1}, ..., v^k_{t,N})$. We obtain $V^1_t, ..., V^K_t$ for all the views.

Language Representations

We encode the language directives as follows. We use an embedding layer initialized with pretrained GloVe [Pennington et al., 2014] vectors to embed each word of the $L$ step-by-step instructions and the goal statement. For each instruction $i = 1, ..., L$, the embedded feature sequence is inputted to a two-layer LSTM [Hochreiter and Schmidhuber, 1997], and its last hidden state is used as the feature $s_i \in \mathbb{R}^d$ of the instruction. We use the same LSTM for all the instructions with dropout regularization. We encode the goal statement $G$ in the same manner using an LSTM with the same architecture different weights, obtaining $h_G \in \mathbb{R}^d$.

3.3 Instruction Decoder

Selecting Instructions

Previous studies [Shridhar et al., 2020; Singh et al., 2020a; Yeung et al., 2020] employ a Seq2Seq model in which all the language directives are represented as a single sequence of word features, and soft attention is generated over it to specify the portion to deal with at each timestep. We think this method could fail to correctly segment instructions with time, even with the employment of progress monitoring [Ma et al., 2019]. This method does not use a few constraints on parsing the step-by-step instructions that they should be processed in the given order and when dealing with one of them, the other instructions, especially the future ones, will be of little importance.

We propose a simple method that can take the above constraints into account, which explicitly represents which instruction to consider at the current timestep $t$. The method introduces an integer variable $m_t \in \{1, L\}$ storing the index of the instruction to deal with at $t$.

To update $m_t$ properly, we introduce a virtual action representing the completion of a single instruction, which we treat equally to the original twelve actions defined in ALFRED. Defining a new token COMPLETE to represent this virtual action, we augment each instruction’s action sequence provided in the expert demonstrations always end with COMPLETE. At training time, we train the action decoder to predict the augmented sequences. At test time, the same decoder predicts an action at each timestep; if it predicts COMPLETE, this means completing the current instruction. The instruction index $m_t$ is updated as follows:

$$m_t = \begin{cases} m_{t-1} + 1, & \text{if } \text{argmax}(p^t_{m_{t-1}}) = \text{COMPLETE} \\ m_{t-1}, & \text{otherwise} \end{cases} \tag{1}$$

where $p^t_{m_{t-1}}$ is the predicted probability distribution over all the actions at time $t-1$, which will be explained in Sec. 3.4. The encoded feature $s_{m_t}$ of the selected instruction is used in all the subsequent components, as shown in Fig. 1.
LSTM LSTM LSTM

Faucet Faucet Faucet None None None

(a) For tentative action predictions
(b) For tentative object predictions

Figure 2: An example illustrates how we reinitialize the hidden states of the two LSTMs in the instruction encoder by $s_{m_t}$ when $m_t = m_{t-1} + 1$ ($m_t = 4$).

Decoder Design
As explained earlier, our method employs a two-stage approach for interpreting the instructions. The instruction decoder (see Fig. 1) runs the first stage, where it interprets the instruction encoded as $s_{m_t}$ without any visual input. To be specific, it transforms $s_{m_t}$ into the sequence of action-object pairs without additional input. In this stage, objects mean the classes of objects.

As it is not based on visual inputs, the predicted action-object sequence has to be tentative. The downstream components in the model (i.e., the mask decoder and the action decoder) interpret $s_{m_t}$ again, yielding the final prediction of an action-object sequence, which are grounded on the visual inputs. Our intention of this two-stage approach is to increase prediction accuracy; we expect that using a prior prediction of (action, object class) pairs helps more accurate grounding.

In fact, many instructions in the dataset, particularly those about interactions with objects, are sufficiently specific so that they are uniquely translated into (action, object class) sequences with a perfect accuracy, even without visual inputs. For instance, “Wash the mug in the sink” can be translated into (Put, Sink), (TurnOn, Faucet), (TurnOff, Faucet), (PickUp, Mug). However, this is not the case with navigation instructions. For instance, “Go straight to the sink” may be translated into a variable number of repetition of MoveAhead; it is also hard to translate “Walk into the drawers” when it requires to navigate to the left/right. Therefore, we separately deal with the manipulation actions and the navigation actions. In what follows, we first explain the common part and then the different parts.

Given the encoded feature $s_{m_t}$ of the selected instruction, the instruction decoder predicts the action and the object class to choose at $t$. To be precise, it outputs the probability distributions $p^a_t \in \mathbb{R}^{N_a}$ and $p^o_t \in \mathbb{R}^{N_o}$ over all the actions and the object classes, respectively; $N_a$ and $N_o$ are the numbers of the actions and the object classes.

These probabilities $p^a_t$ and $p^o_t$ are predicted separately by two LSTMs in an autoregressive fashion. The two LSTMs are initialized whenever a new instruction is selected; to be precise, we reset their internal states as $h^{i_a}_{t-1} = h^{i_o}_{t-1} = s_{m_t}$ for $t$ when we increment $m_t$ as $m_t = m_{t-1} + 1$ (see the example in Fig. 2). Then, $p^a_t$ and $p^o_t$ are predicted as follows:

\begin{align}
    p^a_t &= \text{softmax}(W_{ia}\text{LSTM}(E_a(p^a_{t-1}), h^{i_a}_{t-1}) + b_{ia}), \quad (2a) \\
    p^o_t &= \text{softmax}(W_{io}\text{LSTM}(E_o(p^o_{t-1}), h^{i_o}_{t-1}) + b_{io}), \quad (2b)
\end{align}

where $W_{ia} \in \mathbb{R}^{N_a \times d}$, $b_{ia} \in \mathbb{R}^{N_a}$, $W_{io} \in \mathbb{R}^{N_o \times d}$, and $b_{io} \in \mathbb{R}^{N_o}$ are learnable parameters; $E_a$ maps the most likely action into the respective vectors according to the last predictions $p^a_{t-1}$ using a dictionary with $N_a \times d$ learnable parameters; $E_o$ does the same for the object classes. The predicted $p^a_t$ and $p^o_t$ are transferred to the input of these LSTMs at the next timestep and also inputted to the downstream components, the mask decoder and the action decoder.

Now, as they do not need visual inputs, we can train the two LSTMs in a supervised fashion using the pairs of instructions and the corresponding ground truth action-object sequences. We denote this supervised loss, i.e., the sum of the losses for the two LSTMs, by $L_{\text{aux}}$. Although it is independent of the environment and we can train the LSTMs offline, we simultaneously train them along with other components in the model by adding $L_{\text{aux}}$ to the overall loss. We think this contributes to better learning of instruction representation $s_{m_t}$, which is also used by the mask decoder and the action decoder.

As mentioned above, we treat the navigation actions differently from the manipulation actions. There are three differences. First, we simplify the ground truth action sequence for the navigation actions if necessary. For instance, suppose an instruction “Turn left, go ahead to the counter and turn right” with a ground truth action sequence “RotateLeft, MoveAhead, MoveAhead, MoveAhead, MoveAhead, RotateRight”. The repetition of MoveAhead reflects the environment and cannot be predicted without visual inputs. Thus, by eliminating the repeated actions, we convert the sequence into the minimum-length one, “RotateLeft, MoveAhead, RotateRight”, and regard it as the ground truth sequence, training the instruction decoder. Second, as there is no accompanied object for the navigation actions, we use the object-class sequence “None, None, None” as the ground truth. Third, in the case of navigation actions, we do not transfer the outputs $p^a_t$ and $p^o_t$ to the mask decoder and the action decoder and instead feed constant (but learnable) vectors $p^a_{\text{nav}} \in \mathbb{R}^{N_a}$ and $p^o_{\text{nav}} \in \mathbb{R}^{N_o}$ to them. As the instruction decoder learns to predict the minimum-length action sequences as above, providing such predictions will be harmful for the action decoder. We avoid this by feeding $p^a_{\text{nav}}$ and $p^o_{\text{nav}}$.

3.4 Action Decoder
The action decoder receives four inputs and predicts the action at $t$. The inputs are as follows: the encoded instruction $s_{m_t}$, the output $p^a_t$ and $p^o_t$ of the instruction decoder$^2$ and aggregated feature $v_t$ of visual inputs, which will be described below.

Hierarchical Attention over Visual Features
As explained in Sec. 3.2, we use the multi-view object-centric representation of visual inputs. To be specific, we aggregate $N \times K$ outputs of Mask R-CNN from $K$ ego-centric images, obtaining a single vector $v_t$. The Mask R-CNN outputs for view $k = 1, \ldots, K$ are the visual features $(v_{t,1}^k, \ldots, v_{t,N}^k)$ and the confidence scores $(p_{t,1}^k, \ldots, p_{t,N}^k)$ of $N$ detected objects.

$^2$These are replaced with $p^a_{\text{nav}}$ and $p^o_{\text{nav}}$ if $\text{argmax}(p^a_t)$ is not a manipulation action, as mention above.
To do this feature aggregation, we employ a hierarchical approach, where we first search for the objects relevant to the current instruction in each view and then merge the features across the views to form a single feature vector. In the first step, we compute and apply soft-attention on $N$ objects for each view. To be specific, we compute attention weights $\alpha^k_{n} \in \mathbb{R}^N$ across $v^{k,1}_{t}, \ldots, v^{k,N}_{t}$ guided by $s_m$ as

$$\alpha^k_{n} = \text{softmax}(v^k_n \top W^k_{s_m}),$$  

(3)

where $W^k_s \in \mathbb{R}^{d \times d}$ is a learnable matrix, for $k = 1, \ldots, K$. We then apply the weights to the $N$ visual features multiplied with their confidence scores for this view, yielding a single $d$-dimensional vector as

$$v^k_t = \sum_{n=1}^{N} \alpha^k_{n} v^{k,n}_t \rho^k_{n},$$  

(4)

where $\rho^k_{n}$ is the confidence score associated with $v^k_{n}$. In the second step, we merge the above features $v^1_t, \ldots, v^K_t$ using gated-attention. We compute the weight $\alpha^g_k (\in \mathbb{R})$ of view $k (=1, \ldots, K)$ guided by $s_m$ as

$$\alpha^g_k = \text{sigmoid}(v^g_k \top W^g_s s_m),$$  

(5)

where $W^g_s \in \mathbb{R}^{d \times d}$ is a learnable matrix. Finally, we apply the weights to $\{v^k_t\}_{k=1, \ldots, K}$ to have the visual feature $v_t \in \mathbb{R}^{d}$ as

$$v_t = \sum_{k=1}^{K} \alpha^g_k v^k_t.$$  

(6)

As shown in the ablation test in the appendix, the performance drops significantly when replacing the above gated-attention by soft-attention, indicating the necessity for merging observations of different views, not selecting one of them.

**Decoder Design**

The decoder predicts the action at $t$ from $v_t$, $s_{m(t)}$, $p_t^{m}$ and $p_t^{i}$. We employ an LSTM, which outputs the hidden state $h^{a}_{t} \in \mathbb{R}^{d}$ at $t$ from the previous state $h^{a}_{t-1}$ along with the above four inputs as

$$h^{a}_{t} = \text{LSTM}([v_t; s_{m(t)}; p_t^{m}; p_t^{i}; h^{a}_{t-1}]),$$  

(7)

where $[; ;]$ denotes concatenation operation. We initialize the LSTM by setting the initial hidden state $h^{a}_{0}$ to $h_{G}$, the encoded feature of the goal statement; see Sec. 3.2. The updated state $h^{a}_{t}$ is fed into a fully-connected layer to yield the probabilities over the $N_a + 1$ actions including COMPLETE as follows:

$$p^{a}_{t} = \text{softmax}(W_a h^{a}_{t} + b_{a}),$$  

(8)

where $W_a \in \mathbb{R}^{(N_a+1) \times d}$ and $b_{a} \in \mathbb{R}^{N_a+1}$. We choose the action with the maximum probability for the predicted action. In the training of the model, we use cross entropy loss $\mathcal{L}_{\text{action}}$ computed between $p^{a}_{t}$ and the one-hot representation of the true action.

### 3.5 Mask Decoder

To predict the mask specifying an object to interact with, we utilize the object-centric representations $V^c_t = (v^{c,1}_t, \ldots, v^{c,N}_t)$ of the visual inputs of the central view ($k = c$). Namely, we have only to select one of the $N$ detected objects. This enables more accurate specification of an object mask than predicting a class-agnostic binary mask as in the prior work [Shridhar et al., 2020].

To do this, we first apply simple self-attention to the visual features $V^c_t$, aiming at capturing the relation between objects in the central view. We employ the attention mechanism inside the light-weight Transformer with a single head proposed in [Nguyen et al., 2020] for this purpose, obtaining $\hat{A}_{V^c_t}(V^c_t) \in \mathbb{R}^{N \times d}$. We then apply linear transformation to $\hat{A}_{V^c_t}(V^c_t)$ using a single fully-connected layer having weight $W \in \mathbb{R}^{N \times d}$ and bias $b \in \mathbb{R}^{d}$, with a residual connection as

$$V^m_t = \text{ReLU}(W \hat{A}_{V^c_t}(V^c_t) + 1_K \cdot b^\top) + V^c_t,$$  

(9)

where $1_K$ is $K$-vector with all ones.

We then compute the probability $p_{t,n}^{m}$ of selecting $n$-th object from the $N$ candidates using the above self-attended object features along with other inputs $s_{m(t)}$, $p_t^{a}$, and $p_t^{i}$. We concatenate the latter three inputs into a vector $g_{t,n}^{m} = [s_{m(t)}; p_t^{a}; p_t^{i}]$ and then compute the probability as

$$p_{t,n}^{m} = \text{sigmoid}(g_{t,n}^{m} \top W_m c_t),$$  

(10)

where $W_m \in \mathbb{R}^{d + N_a + N_i \times d}$ is a learnable matrix. We select the object mask with the highest probability (i.e., $\text{argmax}_{n=1, \ldots, N}(p_{t,n}^{m})$) at inference time. At training time, we first match the ground truth object mask with the object mask having the highest IoU. Then, we calculate the BCE loss $\mathcal{L}_{\text{mask}}$ between the two.

### 4 Experiments

#### 4.1 Experimental Configuration

**Dataset.** We follow the standard procedure of ALFRED; 25,743 language directives over 8,055 expert demonstration episodes are split into the training, validation, and test sets. The latter two are further divided into two splits, called seen and unseen, depending on whether the scenes are included in the training set.

**Evaluation metrics.** Following [Shridhar et al., 2020], we report the standard metrics, i.e., the scores of Task Success Rate, denoted by Task and Goal Condition Success Rate, denoted by Goal-Cond. The Goal-Cond score is the ratio of goal conditions being completed at the end of an episode. The Task score is defined to be one if all the goal conditions are completed, and otherwise 0. Besides, each metric is accompanied by a path-length-weighted (PLW) score [Anderson et al., 2018a], which measures the agent’s efficiency by penalizing scores with the length of the action sequence.

**Implementation details.** We use $K = 5$ views: the center view, up and down views with the elevation degrees of $\pm 15^\circ$, and left and right views with the angles of $\pm 90^\circ$. We employ a Mask R-CNN model with ResNet-50 backbone that receives a $300 \times 300$ image and outputs $N = 32$ object candidates. We train it before training the proposed model with
Table 1: Task and Goal-Condition Success Rate. For each metric, the corresponding path weighted metrics are given in parentheses. The highest values per fold and metric are shown in **bold**. Our winning entry in the ALFRED Challenge 2020 is denoted with ⋄.

| Sub-goal | [Shridhar et al., 2020] | [Singh et al., 2020b] | Ours |
|----------|-------------------------|-----------------------|------|
| Goto     | 51                      | 22                    | 54   |
|          | 32                      | 44                    | 44   |
|          | 82                      | 66                    | 66   |
|          | 51                      | 51                    | 55   |
|          | 88                      | 87                    | 88   |
|          | 85                      | 84                    | 86   |
|          | 81                      | 79                    | 79   |
|          | 100                     | 93                    | 93   |
| Average  | 68                      | 70                    | 71   |

Table 2: Sub-goal success rate. All values are in percentage. The agent is evaluated on the Validation set. Highest values per fold are indicated in **bold**.

800K frames and corresponding instance segmentation masks collected by replaying the expert demonstrations of the training set. We set the feature dimensionality \( d = 512 \). We train the model using imitation learning on the expert demonstrations by minimizing the following loss:

\[
\mathcal{L} = \mathcal{L}_{\text{mask}} + \mathcal{L}_{\text{action}} + \mathcal{L}_{\text{aux}}.
\]

(11)

We use the Adam optimizer with an initial learning rate of \( 10^{-3} \) which is halved at epoch 5, 8, and 10, and a batch size of 32 for 15 epochs in total. We use a dropout with the dropout probability 0.2 for both the visual features and LSTM decoder hidden states.

4.2 Experimental Results

Table 1 shows the results. It is seen that our method shows significant improvement over the previous methods [Shridhar et al., 2020; Yeung et al., 2020; Singh et al., 2020a; Singh et al., 2020b] on all metrics. Our method also achieves better PLW (path length weighted) scores in all the metrics (indicated in the parentheses), showing its efficiency. Notably, our method attains 8.37% success rate on the unseen test split, improving approximately 20 times compared with the published result in [Shridhar et al., 2020]. The higher success rate in the unseen scenes indicates its ability to generalize in novel environments. Detailed results for each of the seven task types are shown in the appendix.

The preliminary version of our method won an international competition, whose performance is lower than the present version. It differs in that \((p^a_t, p^{ip}_t)\) are not forwarded to the mask decoder and the action decoder and the number of Mask R-CNN’s outputs is set to \( N = 20 \). It is noted that even with a single view (i.e., \( K = 1 \)), our model still outperforms [Shridhar et al., 2020; Yeung et al., 2020; Singh et al., 2020a] in all the metrics.

Sub-goal success rate. Following [Shridhar et al., 2020], we evaluate the performance on individual sub-goals. Table 2 shows the results. It is seen that our method shows higher success rates in almost all of the sub-goal categories.

4.3 Ablation Study

We conduct an ablation test to validate the effectiveness of the components by incrementally adding each component to the proposed model. The results are shown in Table 3.

Table 3: Ablation study for the components of the proposed model. We report the success rate (Task score) on the validation seen and unseen splits. The X mark denotes that a corresponding component is removed from the proposed model.

Model variants 1-4 use a single-view input (\( K = 1 \)); they do not use multi-view inputs and the hierarchical attention method. Model 1 further discards the instruction decoder by replacing it with the soft-attention-based approach [Shridhar et al., 2020], which yields a different language feature \( s_{at} \) at each timestep. Accordingly, \( p^a_t \) and \( p^{ip}_t \) are not fed to the mask/action decoders; we use \( p^{m}_{at} = [s_{at}; h^t_t] \). These changes will make the method almost unworkable. Model 2 retains only the instruction selection module, yielding \( s_{ms} \). It performs much better than Model 1. Model 3 has the instruction decoder, which feeds \( p^a_t \) and \( p^{ip}_t \) to the subsequent decoders. It performs better than Model 2 by a large margin, showing the effectiveness of the two-stage method.

Model 4 replaces the mask decoder with the counterpart of the baseline method [Shridhar et al., 2020], which upsamples
a concatenated vector \([g; \nu]\) by deconvolution layers. This change results in inaccurate mask prediction, yielding a considerable performance drop. Model 5 is the full model. The difference from Model 3 is the use of multi-view inputs with the hierarchical attention mechanism. It contributes to a notable performance improvement, validating its effectiveness.

### 4.4 Qualitative Results

#### Entire Task Completion

Figure 3 shows the visualization of how the agent completes one of the seven types of tasks. These are the results for the unseen environment of the validation set. Each panel shows the agent’s center view with the predicted action and object mask (if existing) at different time-steps. See the appendix for more results.

#### Mask Prediction for Sub-goal Completion

Figure 4 shows an example of the mask prediction by the baseline [Shridhar et al., 2020] and the proposed method where the agents are moved to the same location to accomplish Slice sub-goal.

Figure 4 shows an example of the mask prediction by the baseline [Shridhar et al., 2020] and the proposed method. It shows our method can predict a more accurate object mask when performing Slice sub-goal. More examples are shown in the appendix. Overall, our method shows better results, especially for difficult sub-goals like Pickup, Put, and Clean, for which a target object needs to be chosen from a wide range of candidates.

### 5 Conclusion

This paper has presented a new method for interactive instruction following tasks and applied it to ALFRED. The method is built upon several new ideas, including the explicit selection of one of the provided instructions, the two-stage approach to the interpretation of each instruction (i.e., the instruction decoder), the employment of the object-centric representation of visual inputs obtained by hierarchical attention from multiple surrounding views (i.e., the action decoder), and the precise specification of objects to interact with based on the object-centric representation (i.e., the mask decoder). The experimental results have shown that the proposed method achieves superior performances in both seen and unseen environments compared with all the existing methods. We believe this study provides a useful baseline framework for future studies.

### Acknowledgments

This work was partly supported by JSPS KAKENHI Grant Number 20H05952 and JP19H01110.

### References

- Anderson et al., 2018a. P. Anderson, A. X. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, and A. R. Zamir. On evaluation of embodied navigation agents. *arXiv:1807.06757*, 2018.
- Anderson et al., 2018b. P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sunderhauf, I. Reid, S. Gould, and A. van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *CVPR*, 2018.
- Chang et al., 2017. A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang. Matterport3d: Learning from rgb-d data in indoor environments. In *3DV*, 2017.
- Chen et al., 2019. H. Chen, A. Suhr, D. Misra, N. Snavely, and Y. Artzi. Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *CVPR*, 2019.
- Corona et al., 2020. R. Corona, D. Fried, C. Devin, D. Klein, and T. Darrell. Modularity improves out-of-domain instruction following. *arXiv:2010.12764*, 2020.
[Côté et al., 2018] Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Ruo Yu Tao, Matthew Hausknecht, Layla El Asri, Mahmud Adada, Wendy Tay, and Adam Trischler. Textworld: A learning environment for text-based games. *CoRR*, abs/1806.11532, 2018.

[Das et al., 2018] A. Das, S. Datta, G. Gkioxari, S. Lee, D. Parikh, and D. Batra. Embodied Question Answering. In *CVPR*, 2018.

[Devin et al., 2018] C. Devin, P. Abbeel, T. Darrell, and S. Levine. Deep object-centric representations for generalizable robot learning. In *ICRA*, 2018.

[Fried et al., 2018] 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.

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

[Gordon et al., 2019] D. Gordon, D. Fox, and A. Farhadi. What should i do now? marrying reinforcement learning and symbolic planning. *arXiv:1901.01492*, 2019.

[He et al., 2017] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *CVPR*, 2017.

[Hermann et al., 2020] K. M. Hermann, M. Malinowski, P. Mirowski, A. Banki-Horvath, K. Anderson, and R. Hadsell. Learning to follow directions in street view. In *AAAI*, 2020.

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[Kempka et al., 2016] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *CIG*, 2016.

[Kolve et al., 2017] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Wehs, A. Herrasti, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi. Al2-Thor: An Interactive 3D Environment for Visual AI. *arXiv:1712.05474*, 2017.

[Krantz et al., 2020] J. Krantz, E. Wijmans, A. Majumdar, D. Batra, and S. Lee. Beyond the nav-graph: Vision-and-language navigation in continuous environments. In *ECCV*, 2020.

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

[Majumdar et al., 2020] A. Majumdar, A. Srivastava, S. Lee, P. Anderson, D. Parikh, and D. Batra. Improving vision-and-language navigation with image-text pairs from the web. In *ECCV*, 2020.

[Nguyen et al., 2019] K. Nguyen, D. Dey, C. Brockett, and B. Dolan. Vision-based navigation with language-based assistance via imitation learning with indirect intervention. In *CVPR*, 2019.

[Nguyen et al., 2020] V. Q. Nguyen, M. Suganuma, and T. Okatani. Efficient attention mechanism for visual dialog that can handle all the interactions between multiple inputs. In *ECCV*, 2020.

[Paszke et al., 2019] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[Pennington et al., 2014] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.

[Puig et al., 2018] X. Puig, K. Ra, M. Boben, J. Li, T. Wang, S. Fidler, and A. Torralba. Virtualhome: Simulating household activities via programs. In *CVPR*, 2018.

[Shridhar et al., 2020] 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.

[Shridhar et al., 2021] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. {ALFW}world: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

[Singh et al., 2020a] K. P. Singh, S. Bhambri, B. Kim., and J. Choi. Improving mask prediction for long horizon instruction following. In *ECCV EVAL Workshop*, 2020.

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

[Srivastava et al., 2014] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[Suhr et al., 2019] Alane Suhr, Claudia Yan, Jack Schlager, Stanley Yu, Hadi Khader, Marwa Mouallem, Iris Zhang, and Yoav Artzi. Executing instructions in situated collaborative interactions. 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)*, pages 2119–2130, Hong Kong, China, November 2019. Association for Computational Linguistics.

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

[Thomason et al., 2020] J. Thomason, M. Murray, M. Cakmak, and L. Zettlemoyer. Vision-and-dialog navigation. In *Conference on Robot Learning*, 2020.

[Wang et al., 2019] X. Wang, Q. Huang, A. Celikyilmaz, J. Gao, D. Shen, Y.-F. Wang, W. Y. Wang, and L. Zhang. Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation. In *CVPR*, 2019.

[Wijmans et al., 2019] E. Wijmans, S. Datta, O. Maksymets, A. Das, G. Gkioxari, S. Lee, I. Essa, D. Parikh, and D. Batra. Embodied question answering in photorealistic environments with point cloud perception. In *CVPR*, 2019.

[Wu et al., 2018] Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yandong Tian. Building generalizable agents with a realistic and rich 3d environment. In *ICLR*, 2018.

[Yeung et al., 2020] L. Yeung, Y. Bisk, and O. Polozov. Alfred speaks: Automatic instruction generation for egocentric skill learning. In *ECCV EVAL Workshop*, 2020.

[Zhu et al., 2017] Y. Zhu, D. Gordon, E. Kolve, D. Fox, L. Fei-Fei, A. Gupta, R. Mottaghi, and A. Farhadi. Visual semantic planning using deep successor representations. In *ICCV*, 2017.
Appendix

A Additional Experimental Results

A.1 Performance by Task Type

Table 4 shows the success rates across the 7 task types achieved by the existing methods including ours on the validation set of ALFRED. It is seen that our method outperforms others by a large margin in both seen and unseen environments.

| Task-Type     | [Shridhar et al., 2020] | [Singh et al., 2020b] | Ours    |
|---------------|-------------------------|-----------------------|---------|
|               | Seen        | Unseen     | Seen | Unseen     | Seen | Unseen     |
| Pick & Place  | 7.0         | 0.0        | 29.5 | 5.0        | 40.1 | 13.0        |
| Stack & Place | 0.9         | 0.0        | 5.2  | 1.8        | 17.4 | 11.9        |
| Pick Two      | 0.8         | 0.0        | 11.2 | 1.1        | 21.8 | 11.1        |
| Clean & Place | 1.8         | 0.0        | 22.3 | 2.4        | 40.2 | 15.0        |
| Heat & Place  | 1.9         | 0.0        | 15.8 | 2.7        | 41.2 | 9.6         |
| Cool & Place  | 4.0         | 0.0        | 26.1 | 0.7        | 40.0 | 13.8        |
| Examine       | 9.6         | 0.0        | 20.2 | 13.2       | 34.4 | 12.9        |
| Average       | 3.7         | 0.0        | 18.6 | 3.8        | 33.6 | 11.0        |

Table 4: Success rate across 7 task types. All values are in percentages. The agent is evaluated on the validation set. Highest values per split are indicated in bold.

A.2 Full Results of Ablation Tests

Table 5 shows the full results of the ablation test reported in the main paper. We also provide additional results in Table 6 with different activation functions (i.e., sigmoid or softmax) in the second step of the proposed hierarchical attention mechanism, and with different K’s; K is selected from 1 (only ‘center’ view), 3 (‘center’, ‘left’, and ‘right’ views), or 5 (‘center’, ‘left’, ‘right’, ‘up’, and ‘down’ views)). The results show that the use of gated-attention in Eq.(5) (of the main paper) is essential. We also confirm the number of views also affect the success rate.

B Qualitative Results

B.1 Mask Prediction for Sub-goal Completion

Figure 3 shows examples of mask predictions by the baseline [Shridhar et al., 2020] and the proposed method for different sub-goals. Overall, it is seen that our method can predict more accurate object masks. It shows better results especially for difficult sub-goals like Pickup, Put, and Clean, where a target object needs to be chosen from a wide range of candidates.

B.2 Entire Task Completion

Figures 4-10 show example visualization of how the agent completes the seven types of tasks. These are the results for the unseen environments of the validation set. Each panel shows the agent’s center view with the predicted action and object mask (if existing) at different timesteps.

We also provide seven video clips as independent files, which contain several examples of the agent’s entire task completion for seven above task instances in unseen environments.

C Analyses of Failure Cases

We analyze the failure cases of our method using the results on the validation splits. We categorize them into navigation failures and manipulation failures.

C.1 Navigation Failures

It is seen from the sub-goal results of Table 2 in the main paper that the Goto sub-goal is the most challenging. Failures with it tend to make it hard to complete the entire goal, since they will inevitably affect the subsequent actions to take. We think there are three major cases for the navigation failures.

The first case, which occurs most frequently, is that the agent follows a navigation instruction and reaches a position that should be fine as far as the instruction goes; nevertheless, it is not the right position for the next manipulation action to take. For instance, following the instruction “Go to the table,” the agent goes to the table. The next instruction is “Pickup the remote control at the table,” but the remote lies on the other side of the table. This is counted as a failure of completing the Goto sub-goal.

The second case is when the instructions are either abstract or misleading. An example is that when the agent has to take several left and right turns together with multiple MoveAhead steps to reach the destination, e.g., a drawer, the provided instruction is simply “Go to the drawer.”

The third case, which occurs less frequently, is that while there is an obstacle in front of the agent, e.g., wall, it attempts to take the MoveAhead action. This occurs because of the lack of proper visual inputs. This is demonstrated by the fact that when we reduce the number of views, the task success rate drops significantly, as shown in the second block of Table 6.

C.2 Manipulation Failures

As shown in Table 2, after it has moved to the ideal position right before performing any interaction sub-goals (i.e., all the sub-goals but Goto), the agent can manipulate objects with high success rates of 91% and 69% in the seen and unseen environments, respectively.

However, the success rates for completing the Goto subgoal in the seen and unseen environments are only 59% and 39%, respectively. Therefore, the primary cause of the manipulation failures is that the agent cannot find the target object because it fails to reach the right destination due to a navigation failure.

Even if the agent has successfully navigated to the right destination, it can fail to detect the target object. This seems to happen mostly because the object is either too small or indistinguishable from the surroundings. The agent tends to fail to detect, for example, a small knife placed on the steel/metal-made sink of the same color.

The agent also fails to detect an object that has not seen in the training. This is confirmed by the fact that the performance drops considerably in unseen environments for some
interaction sub-goals (including **Put**, **Clean**, and **Toggle**). There are also a small number of cases where failures are attributable to bad instructions, e.g., incorrect statement of objects.

### D Further Details of Implementation

Table 7 summarizes the hyperparameters used in our experiments. We perform all the experiments on a GPU server that has four Tesla V100-SXM2 of 16GB memory. It has Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz of 20 cores with the RAM of 240GB memory. We use Pytorch version 1.6 [Paszke et al., 2019]. As for the Mask R-CNN model, we train it in advance, separately from the main model, for 15 epochs with the learning rate $10^{-3}$ and halved at the epoch 5, 8, and 12. We train models with batch size of 32 and 4 workers per GPU.

Table 7: Hyperparamters used in the training.

| Hyperparameter   | Value |
|------------------|-------|
| $\beta_1$ in Adam| 0.9   |
| $\beta_2$ in Adam| 0.999 |
| $\epsilon$ in Adam| $1e-8$ |
| Number of workers per GPU| 4 |
| Batch size | 32 |

### E Entry Submission to the ALFRED Embodied AI Challenge 2021

In this section, we describe the details of our entry submission to the ALFRED Embodied AI Challenge 2021, which is organized in conjunct with the CVPR 2021.

To further improve the performance of our model, we create an ensemble of 4 models with some differences, from initializations with different random seeds to whether to use self-attention mechanism in the mask decoder. We train these 4 models independently with the same pretrained Mask R-CNN, whose weights fixed during the training phase. Refer to these above sections for training and implementation details.

During the evaluation, a shared detector Mask R-CNN is to extract the features from visual inputs, which are forwarded into all the models. Each model will output the probability distributions over the actions and objects of interest if any. We then take average these probabilities and select the actions and objects of highest probabilities. Figure 2 illustrates the use of ensemble during the inference.
Table 8: Task and Goal-Condition Success Rate. For each metric, the corresponding path weighted metrics are given in (parentheses). Our entry submission to the EAI Challenge 2021 is denoted with ◦.

Table 8 shows the performance of our single model and its corresponding ensemble. It is seen that ensembling increases the success rate of the agent on both seen and unseen environments in comparison with a single model.

It is noted that the evaluation speed is slowed down proportionally when capturing multiple ego-centric views from the agent’s camera. It is due to the fact that the current version of AI2Thor does not support direct acquisition of multiple/panoramic views; we thus have to move agents into different view points to capture such corresponding views. Our best-performing results from multi-view models suggest that it be of importance to improve the benchmark which is capable of obtaining multiple/panoramic views directly. Also noted that using ensemble during evaluation requires running all of its models simultaneously. Therefore, it leads to a proportional increase in memory and compute.
Figure 3: Examples of the object masks predicted by the baseline method [Shridhar et al., 2020] (left on each panel) and our method (right) when the corresponding agents are at the same location and perform one of the six manipulation sub-goals.

Figure 4: Our agent completes an Examine task “Examine an empty box by the light of a floor lamp” in an unseen environment.
Figure 5: Our agent completes a **Pick & Place** task “Place the green bottle on the toilet basin” in an unseen environment.

Figure 6: Our agent completes a **Pick Two & Place** task “To move two bars of soap to the cabinet” in an unseen environment.

Figure 7: Our agent completes a **Cool & Place** task “Put chilled lettuce on the counter” in an unseen environment.
Figure 8: Our agent completes a **Heat & Place** task “Put a heated apple next to the lettuce on the middle shelf in the refrigerator” in an unseen environment.

Figure 9: Our agent completes a **Clean & Place** task “Put a cleaned rag in the cabinet under the sink” in an unseen environment.

Figure 10: Our agent completes a **Stack & Place** task “Put a cup with knife in it, on the counter” in an unseen environment.