11777 Report 4: Final

Raj Mehta * **(Joseph) Cole Ramos*** **Roochi Shah** * **Meghana Tandon***
{rajm, jcramos, rtshah, mtandon}@andrew.cmu.edu

1 Background

1.1 Introduction

In recent years, multimodal machine learning has emerged as a key area of research in the field of artificial intelligence. In particular, vision and language models have gained significant attention due to their ability to integrate both visual and textual information to perform tasks such as image captioning, visual question answering, and more. In this paper, we propose a multimodal transformer that uses vision and language instructions to act in an environment to complete household chores.

1.2 Task definition

The goal of our paper was to create a machine learning model capable of following insturctions to complete household tasks in an environment. To achieve this goal, we trained and evaluated our model on the ALFRED (Action Learning From Realistic Environments and Directives) dataset (Shridhar et al., 2020). We selected this dataset because of its similarity to how humans follow instructions. When given instructions, humans have to understand their meaning, process their environment, and execute actions to complete them. This dataset closely simulates this paradigm, containing both navigation and non-reversible object interaction; object, instruction, and environment variation; and access to only a partially observable environment at inference time. Additionally, since tasks are naturally decomposable into sub-goals, we found this dataset to be a good benchmark for our adaptation of the dynamic neural module based architecture previously used in visual question answering (Andreas et al., 2015).

*Alphabetical order

1.3 Related Work

In the following subsections, we outline the literature review we performed before designing our model. There were many model architectures and techniques that resulted in impressive performance metrics on the ALFRED dataset. We highlight some of the key ideas that we decided into implement in our model, how they could improve performance compared to the baseline, and why we decided to use them.

Modular Design Singh et al. (2020) proposed a Modular Object-Centric Approach (MOCA) which uses two separate disjoint streams. First is the Action Policy Module (APM) for predicting a sequence of actions. This is followed by an Interactive Perception Module (IPM) which localizes objects to interact with. This approach is motivated by the human visual cortex which contains a ventral stream involved with object perception and a dorsal stream involved with action control. At the time of submission, MOCA was the state-of-the-art. This approach motivated our model design to create two separate modules for predicting object masks and actions.

Zhang and Chai (2021) also decompose goals in a hierarchical manner into sub-goal planning, scene navigation, and object manipulation, wherein the latter two sub-problems comprise sub-goal execution, and use a unified language transformer to integrate the module outputs into actions. Their version of subgoal planning is conducted dynamically as subgoals are processed, rather than a static sequence preceding all actions, to better handle variation in environments. Additionally, they have distinct navigation and manipulation action prediction modules that notably take only their respective action histories up until the current step. Lastly, they implement a simple backtracking heuristic to deal with subgoal failure which involves return-

ing to previous subgoals and re-attempting them. Though we are not adopting dynamic subgoal planning or separating action prediction modules, we were heavily inspired by the breakdown of action histories by navigation and manipulation as well as the idea of using subgoals as an intermediary between high level goals and low level action predictions.

Subtask Decomposition FILM (Min et al., 2021) uses 3 submodules. A Language Processor converts higher level (single sentence) task descriptions into a list of structured subtasks. A Semantic Mapper takes visual observations and builds a map of the environment. The Semantic Search Policy outputs a search goal based on the other two submodules, generally a guess at where to find a task-relevant object. This is all fed into a deterministic search policy that outputs the low level actions the agent should take. FILM was the SOTA in early 2022.

We plan to use the idea of a specific subtask predictor module in our own architecture. We believe that a subtask predictor will allow us to construct a more condensed, task-relevant representation of our language instructions and better focus its attention on the important parts of the instructions.

Despite the effectiveness of mapping methods, we did not use a mapper in our model. The ground truth information required to train a semantic map is present in the simulation, but it is not easily accessible in the training data. We anticipate properly training a mapper would require running many simulations and is something we consider to be too costly for this project.

Speaker-Follower Models Fried et al. (2018) proposed using an embedded "speaker model" which performs two functions: synthesizes new data to augment the training set and assist action choosing during inference time. The overall model is composed of a follower module (which predicts the next actions) and the speaker module (which generates instructions). The speaker is trained before the follower and instructions generated by the speaker is used to augment the training dataset of the follower module. At inference time, the follower generates candidate-routes, uses the speaker to generate instructions for each candidate-route, and chooses the route where the speaker output is most similar to the instructions given to the follower as input. This speaker-follower approach is used directly in our model. However, our proposed model uses the speaker to generate sub-goals as opposed to instructions.

Vision Embeddings (He et al., 2017) improved on Faster R-CNN to create an instance segmentation model where object mask prediction is done in parallel with bounding box recognition. Mask-R-CNN creates these attention masks with little additional overhead to Faster R-CNN. Faster R-CNN was not designed for pixel-to-pixel alignment, which is crucial for the object interaction masks in the ALFRED environments. This extension of Faster R-CNN is the reason why we initially proposed on using the masks generated by Mask R-CNN for our visual embeddings.

Language Embeddings (Suglia et al., 2021) The Embodied BERT model (EmBERT) made use of the large language model (BERT) to embed language instructions. They used a Mask-R-CNN for object detection and combined the two modalities with an OSCAR transformer. Other than using a LLM, another contribution this paper was made was using object centric navigation losses. When navigating, the agent would always move towards a target object and object receptacle, which improved navigation in the environment. In our model we used BERT models as well to generate language embeddings from language instructions and language goals.

Vision and Language Interactions OSCAR (Li et al., 2020) is an encoder for the visual-language modalities, and is well-suited for tasks focused on object detection. To align language and visual inputs, OSCAR uses an object detector to predict names and locations of objects in the scene. Since such object names (or semantically similar words) are often also in the caption for the image, OSCAR can leverage this information during training know what image regions to align to certain parts of the caption. ALFRED instructions always mention objects the agent needs to find and interact with, so we believe this type of language-visual alignment will also be present in the ALFRED training data. This motivates our choice to fine-tune an OSCAR model for our model's vision and language embedding component.

Progress Monitoring Chiang et al. (2021a) focus heavily on alignment of text and vision modalities in ALFRED, and argue it is critical to task completion. They introduce a mask on attention

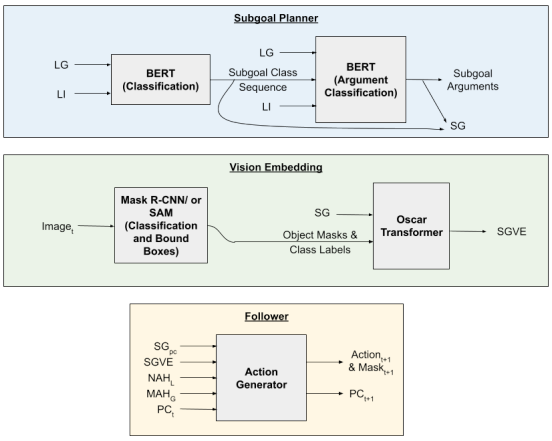
mechanism, computed as a function of a soft program counter trained via an auxiliary MSE loss to learn stronger alignment, and demonstrably improve two models with their modification. Though the authors learned progress on instructions, we intend on adapting this method to subgoal sequences.

Self Monitoring Navigation Agent via Auxillary Progress (Ma et al., 2019) employs a progress monitoring module, that is conditioned on the history of grounded images and instructions, the current observation of the surrounding images,and the positions of grounded instructions. The instructions are grounded by the agents observations to identify what part of text the agent should currently attend do. It uses an auxiliary loss function called 'distance to goal' that measures how far the agent is from completing its task. Intuitively minimizing this loss function makes sense as trajectories should be selected in a way that encourages progress towards completing a task. In our model we aim to use progress monitoring in a similar fashion as the self monitoring agent. However, instead of using it to evaluate actions and estimate distance from the goal, we will use to evaluate when to move toward the next language instruction.

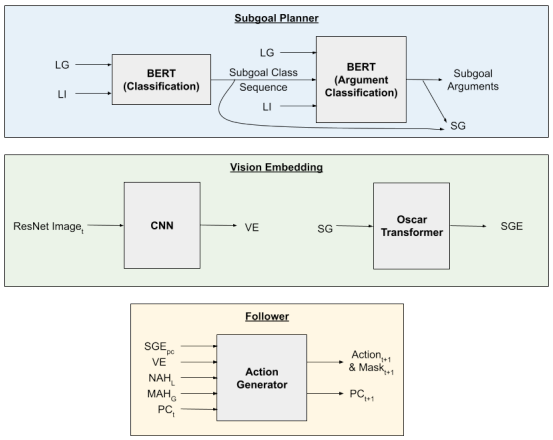
2 Approach

The approach of our final model differs from what was presented in previous reports because of constraints on model size and code base implementations. Originally, we proposed combining BERT encoded language subgoals with MaskR-CNN outputs of object masks and class labels in the multimodal OSCAR transformer. This output was then to be passed in to the action generator. However, due to the aforementioned constraints, this component of our model was replaced by the ResNet Features compressed through two convolutional and linear layers, and the BERT encoded language subgoals passed through the Oscar transformer as two separate inputs into the action generator. The difference between these two components is that the vision and language are not grounded together through the OSCAR transformer, and that the Vision Embedding module does not have an object mask extractor.

2.1 Model



The proposed model architecture.



The implemented model architecture.

2.1.1 Subgoal Planner

The purpose of the subgoal planner, shown in blue in both diagrams, is to create a condensed language representation by translating the high level language instructions into formulaic subgoals. This is in order to exploit the formulaic nature of the trajectories and create a consistent representation of high-level tasks that have to be met. This subgoal planner is composed of two BERT models fine-tuned for two different classification tasks: BERT Task Classification and BERT Argument Classification. Language Goals are concatenated with Language Instructions and are passed into the BERT Task Classification model to classify the input into one of seven task types. Each task type is then mapped to a MANUAL template of their subgoals pattern. This template resembles tuples corresponding to an action and a corresponding object that action applies to. For each tuple in the array of templated subgoals, the BERT Argument Classification module classifies the object that the action is being applied to. This array of templated subgoals is then converted manually back into a condensed language representation. An example of this is as follows.

The concatenated language goal and instructions representation:
take a rinsed egg to the microwave[SEP]go to the counter to the right of the sink[SEP]take the egg from the counter[SEP]take the egg to the sink[SEP]put the egg in the sink then take back out[SEP]take the egg to the microwave[SEP]put the egg in the microwave

The corresponding subgoal representation template:
('Egg', 'PickupObject'), ('SinkBasin', 'PutObject'), ('Faucet', 'ToggleObjectOn'), ('Faucet', 'ToggleObjectOff'), ('Egg', 'PickupObject'), ('Microwave', 'OpenObject'), ('Microwave', 'PutObject'), ('Microwave', 'CloseObject')

Translating back to English: Pick up the Egg. Put the Egg inside the Sink Basin. Turn on the Faucet. Turn off the Faucet. Pick up the Egg. Open the Microwave. Put the Egg inside the Microwave. Close the Microwave.

Pretraining This subgoal planner module uses pretrained weights from FILM's model and it was

not finetuned further as this module was trained independently.

2.1.2 Vision Embedding

The purpose of the vision embedding module is to extract relevant features from the images given in the ALFRED dataset and then ground them to the language subgoals from the subgoal planner module. This module begins with applying BERT embeddings to the language subgoals output from the previous subgoal planner module. Then, in general, we extract image features and pass the embedded subgoals through the multimodal Oscar transformer.

Proposed The proposed approach was to use MaskRCNN as our vision encoder, which takes in images as inputs and outputs a list of objects identified in the scene along with their class and mask. The output from the MaskRCNN is concatenated with the provided ResNet features and current subgoal SG and passed into an OSCAR transformer. The OSCAR transformer is especially well-suited for our task as it is very effective at aligning language with object masks and names specifically. As these are the inputs to our transformer, we believe it was the best architecture to choose. The output from this transformer is a latent visual representation that encodes information about the current subgoal and an embedding of the objects in the scene which we denote as $SGVE$. This latent representation is used by another module to influence the actions the agent takes. A similar vision module was used in the Embodied BERT model (EmBERT Citation).

Implemented This proposed approach changed in our implementation as there were issues adapting the MASK R-CNN code base to ours (lack of modularity, versioning isseues, configuration issues). As a result, our approach for this module changed slightly. First, we use the ResNet images features given in the original dataset for the scene images. This ResNet images features are then passed through a small two-layer convolutional layer to create a hidden smaller representation of these features. As we could not run Mask R-CNN and receive that output of object masks and class labels for the current image, we proceed to only pass the BERT embeddings for the language subgoals output by the Subgoal Planner into the Oscar transformer. The weights of this transformer have been frozen and we do not train it further. We chose to do this because the hidden smaller representation of the ResNet images is not what the Oscar transformer was trained on. Additionally, we were limited on AWS compute resources. The output of this frozen Oscar transformer is a latent representation of the language subgoals which are then passed into the follower module.

2.1.3 Follower

The action generator submodule, also known as the follower, generates actions using the outputs from the Vision Embedding module and subgoal Translator alongside newly introduced inputs. These newly introduced inputs are the local navigation action history NAH_L (where local refers to the current subgoal) and the global manipulation history MAH_G. Additionally, the follower module inputs a progress counter PC_t which is incremented to reflect the current subgoal being processed. The action generator then outputs the next action and object mask as well as the progress counter at the next time step PC_{t+1} which tells the model whether or not to move on to generating actions for the next subgoal. We chose not to utilize recurrence to compress action history into a hidden state, so that attention mechanisms could learn what is relevant, even if it is distant in the action history. This choice is reinforced by our unimodal baselines as the multimodal baseline without the hidden state had the best performance on the validation unseen set.

Proposed Our proposed architecture for this action generator was a multimodal transformer with variable size encodings for all of the inputs. In addition to the inputs described above, our proposed architecture would take in $SGVE$ which is the latent representation output by the Oscar transformer in the vision embedding module, and SG_{pc} which is the language subgoal output of the subgoal planner with a progress counter weighting over it.

Implemented The approach we implemented used standardized embedding size (128) for all inputs and are fed into one transformer. All of these inputs are separated and ordered similar to positional encodings in regular transformers. However, the fixed embedding size is an issue because Vision Embedding suffered from information loss through that bottleneck whereas the action history's embedding size was enlarged 10-15x its actual size.

Action Loss The action loss is the loss of whether or not the correct action was predicted. The action loss is a cross entropy loss between the target actions in the trajectory and the predicted actions from the model.

Mask Loss The mask loss is the loss on whether the action mask was predicted correctly. This mask loss is computed from a weighted binary cross-entropy loss which accounts for the weight-imbalance between 0 and 1 pixels.

PC-Increment Loss The PC-Increment Loss is the loss from whether the pc increment was predicted correctly. This is computed as a cross entropy loss between the predicted and target PC increment prediction (scalar from 0 to 1).

Total Loss The total loss used to back-propagate through this model is the sum of the Action Loss, Mask Loss, and PC-Increment Loss. All of these individual losses are computed from the action generator and there are no auxiliary losses in the Subgoal Planner Module and Vision Embedding Module. This is because the Subgoal Planner module uses pre-trained, frozen weights. So, there is no direct loss or information being back propagated through this module. The Oscar transformer weights in the vision embedding are frozen and the losses from the Action Generator are backpropagated through the CNN.

2.2 Novelty

The novelty in our model architecture is three-fold and lies in the the specific modules utilized, our action generator and the modalities selected to contribute to final action and mask generation, as well as the soft progress counter reweighting of subgoal self-attention. Since the first has already been elaborated on, we focus on the latter two here. Our proposed model also had further novelty in the intentionally limited early cross-modality interaction in the form of the subgoal aware vision embedding. We hypothesized that a SubGoal-informed Vision Embedding, or SGVE, would focus greater attention to relevant targets in the frame via an object-centric approach, which would in particular benefit navigation, a known challenge in ALFRED. However, as detailed earlier, this was not implemented due to constraints.

Multimodal Action Generator
The multimodal action generator takes in the vision and predicted subgoal modalities as generated by the preceding modules. More notably, it takes as input the progress counter, indicating which subgoal in the sequence the actor is currently on, as well as fractured action histories. This concept of breaking apart action histories is novel and was motivated by the idea that excessive navigation information can add unnecessary noise and confuse the actor. Instead, we keep only local navigation history (where local is specific to the current subgoal) and global manipulation history, since it's important for the actor to know what it interacted without throughout task completion. We introduce a subgoal positional encoding so as not to lose relative action distances after splitting and discarding elements of the action sequence rendering the default assumption that every action is consecutive meaningless.

The generator itself comprises of a merged attention transformer encoder, preceded by a self-attention transformer encoder layer for each modality. One downside of the merged attention approach is that is requires all tokens, regardless of modality, to map to the same embedding size, which likely isn't appropriate since a one-size-fits-all approach will likely over-compress information-dense vision and unnecessarily expand out the progress counter, which is merely an integer value.

Finally, the original goal of our architecture was to backpropagate into and finetune OSCAR, to refine the vision and language embeddings. We still let gradients flow beyond the generator itself and train the embedding modules, yielding an architecture that has not been tested in prior literature.

Progress Counter Subgoal Re-Weighting
Another novel concept was the idea of reweighting subgoal self-attention by a distance function from the current progress counter PC. This was inspired by the work in Chiang et al. (2021b) and adapted from their usage in the instruction token setting to the subgoal setting. Assuming sg_t is the position of the subgoal sg in the sequence of subgoals, the masked reweighting factor m at the current timestep is:

$$m = exp\{-\lambda|sg_t - PC|\}$$

where λ is a hyperparameter. The reweighting factor is applied element-wise to attention weights. The function operates by exponentially decreasing in value as the distance from the true (or predicted, at inference time) subgoal, as quantified by PC increases.

2.3 Experiments

GroundTruthSubgoals
As part of this experiment, we wanted to see the value of using Subgoals as an input into our action transformer. A set of language instructions are classified into various subgoals. These subgoals are accesible through our json file containing expert trajectories based on the PDDL solver. As such, these are ground truth subgoal labels.

PredictorSubgoals
As part of our second experiment we used a frozen module from the FILM model. As mentioned before, FILM achieved SOTA results on the ALFRED database using two BERT models in its language module. We froze a module of the FILM model that used language instructions concatenated with language goals as an input, and outputted subgoals. These subgoals are structured slightly differently to the ground truth subgoals. These were fed into the 'follower' part of our model to predict actions. The vision module of our model remained the same. It is important to note that FILM's pre-processed subgoals only contained about 40% of the training set and approximately 75% of the validation set.

PCReweighting
Our third experiment uses ground truth subgoals but in addition to the base model, we employ a progress counter re-weighting on the self-attention transformer encoder layer applied exclusively to the subgoals. Here, we test whether introducing

this structural bias into the attention mechanism simplifies learning in the action generator and therefore improves performance. This particular experiment was conducted with $\lambda = 1$.

2.4 Unimodal Baseline

1. **Baseline-Vision** - Seq2Seq architecture using the *Frames*, *Action*, and *HS* inputs only. The model is constructed by reducing the setting the language dropout to 1.

2. **Baseline-LangInstruct** - Seq2seq architecture using the *BERT GoalInstruct*, *Action*, and *HS* inputs only. We tested the BERT encodings over using the default encodings because (Shridhar et al., 2020) already reported results for a unimodal language baseline for the latter. We mean to compare these two results to evaluate the usefulness of BERT as an encoder for this task, particularly since our proposed model will utilize BERT.

3. **Baseline-Subgoal** - Seq2seq architecture using the *BERT Subgoal*, *Action*, and *HS* inputs only. Though subgoals are not direct inputs to the model, our current plan for the final architecture is to parse subgaols as an intermediate, condensed language representation, and then use those subgoals as inputs to other modules in the architecture. By comparing **Baseline-LangInstruct** to **Baseline-Subgoal** we hope to see if using subgoals over the raw language instructions is a viable strategy.

4. **Baseline-Action-History** Seq2seq architecture using the *Action* and *HS* inputs only. We masked over all of the vision and language modalities. Therefore, only the previous action and the hidden state which carries information about the action history are used. We ran this baseline to assess how often the previous action can predict the next action. Two types of actions exist in this data set: navigation and interaction actions. We expect the action history baseline to be more successful predicting navigation actions than interaction actions.

5. **Baseline-Unmodified Seq2seq** Seq2Seq is the primary multimodal baseline and uses the *GoalInstr*, *Frames*, *Action*, and *HS* modalities. This is the baseline from the ALFRED paper (Shridhar et al., 2020). In order to make valid comparisons, we retrained this model instead of using the results reported in the paper since we were running on a subset of the data.

6. **Baseline-No-Hidden-State** Seq2Seq using the *GoalInstruct*, *Frames*, and *Action* modalities. As mentioned earlier, this experiment was run to assess the influence of history on task completion. Because we expect history to be critical to this objective, we anticipate this model will perform poorly.

7. **Baseline-BERT-Multimodal** Seq2Seq using the *BERT GoalInstr*, *Frames*, *Action*, and *HS* modalities. This test was to assess if the BERT embeddings offer an advantage over a traditional language embedding in a multimodal setting.

8. **FILM** (Min et al., 2021) - The state of the art model in early 2022. FILM followed a very modular approach. A Language Processor converts higher level (single sentence) task descriptions into a list of structured subtasks. A Semantic Mapper takes visual observations and builds a map of the environment. The Semantic Search Policy outputs a search goal based on the other two submodules, generally a guess at where to find a task-relevant object. This is all fed into a deterministic search policy that outputs the low level actions the agent should take.

   FILM was one of the main inspirations behind trying to parse subgoals as an intermediate language representation, and why we ran **Baseline-Subgoal**.

2.4.1 Unimodal Baseline Results

Training Analysis All relevant plots are included in the Appendix. Some models were not run to completion because they crashed midway. However, we achieved convergence in loss so we did not attempt to rerun them more than twice due to cost constraints.

The first set of models plotted in the Appendix performed poorly. These were the **Baseline-LangInstr**, **Baseline-Subgoal** and **Baseline-Action-History** models. For all three models, loss was quite erratic, stagnating around 1.15, and validation loss mostly grew as the model trained. This demonstrates that perhaps in isolation, the language, subgoal, and action history modalities

are insufficient to make good action and mask predictions. Observe that **Baseline-LangInstr** and **Baseline-Subgoal** performed the same on the seen and unseen validation sets. This validates our original architecture design based around forming a more condensed language representation with subgoals.

Because we were unable to run **Baseline-Action-History** for more than two epochs, this claim is slightly weaker. However, if we observe the second set of models in which loss clearly dropped, the first couple epochs are where this decrease is most visible, so we can extrapolate that action-history is likely on trend with the other poorly performing unimodal baselines.

The second set of models in the Appendix compare **Baseline-No-Hidden-State**, **Baseline-BERT-Multimodal**, **Baseline-Unmodified-Seq2Seq**, and **Baseline-Vision**. All models achieved convergence at around the same mask loss of 0.1 and varied more for action loss, with **Baseline-Vision** and **Baseline-Unmodified-Seq2Seq** performing the best in both respects. This makes sense since vision is very informative for mask generation. It is surprising, however, that the other modalities in **Baseline-Unmodified-Seq2Seq** had marginal benefit on top of vision. This either suggests that these other modalities are redundant, or more likely, that this architecture is unable to meaningfully extract new information from them. We do see a more noticeable difference in the progress monitoring loss, for which the holistic baseline is more performant. Because our proposed model has a component of predicted progress monitoring, this serves as evidence for merging modalities when doing so. Seen validation loss drops, however, all models struggle with unseen environments, and we actually see loss increasing over the training period, indicating we are overfitting and unable to navigate new environments. We postulate that our proposed cycle consistency will help with this by better utilizing language to inform how the agent moves in a new space.

Best Model Analysis Overall, we find that our multimodal model which uses BERT embeddings as well as the original seq2seq baseline in (Shridhar et al., 2020) have the most completed goal conditions. While the BERT multimodal models achieved twice the completed goal conditions as the vision only baseline, there is little variation in their PLC GC success rate. This indicates that the BERT multimodal model generated inefficient trajectories even for goal conditions that were met.

Furthermore, in the unseen results, we see that the baseline seq2seq significantly outperforms the other baselines on completed goal conditions. Additionally, it is the only baseline to achieve a non-zero path length weighted success rate.

Seen	Completed GC	GC succ. rate	PLW GC succ. rate	PLW succ. rate
Baseline Seq2seq	18	0.0678	0.0440	0.006
BERT Multimodal	19	0.0725	0.0360	0.006
Baseline-LangInstr	12	0.0458	0.0350	0
Baseline-Subgoal	12	0.0458	0.0350	0
Baseline-Action-History	12	0.0458	0.0107	0
No-Hidden-State	12	0.0458	0.0315	0
Baseline-Vision	9	0.0370	0.0317	0
FILM	-	0.288	0.1559	0.112

Fig 1

Unseen	Completed GC	GC succ. rate	PLW GC succ. rate	PLW succ. rate
Baseline Seq2seq	15	0.058	0.044	0.001
BERT Multimodal	9	0.035	0.0367	0
Baseline-LangInstr	9	0.0348	0.0367	0
Baseline-Subgoal	9	0.0348	0.0367	0
Baseline-Action-History	9	0.0348	0.0282	0
No-Hidden-State	11	0.0426	0.0329	0
FILM	-	0.3852	0.1513	0.1132

Fig 2

GC	Seen	Baseline	BERT	LU	SU	AH	HS	V
'Put a cooked apple in the sink'	Unseen	Y	Y	Y	Y	Y	Y	Y
'Place a clean cup in the coffee maker'	Seen	Y	Y	Y	Y	Y	Y	NA
'Turn on the desk lamp while holding a yellow disc'	Seen	Y	Y	N	N	N	N	NA
"Put heated apple in sink"	Seen	Y	Y	N	N	N	N	N
'Place a clean ladle on a table.'	Seen	Y	Y	Y	Y	Y	Y	NA
'Put a heated apple back in the garbage where you got it.'	Unseen	Y	Y	Y	Y	Y	Y	Y
'Get an apple from the sink and heat it up in the microwave'	Unseen	Y	Y	Y	Y	Y	Y	Y
"Place a cold tomato slice on the counter"	Unseen	N	N	N	N	N	N	N

Fig 3

2.5 Final Results

Overall, we find that using the subgoal planner module offers the best masking loss. Additionally, we see that applying a progress counter reweighting of the subgoals does not appear to affect the convergence of any loss beyond what is accounted for in the **GroundTruthSubgoals** experiment. We also find that the pc loss, action loss, and overall training loss converge quickly in all three experiments.

2.5.1 GroundTruthSubgoals

Overall, we notice that the total loss during training (the sum of the action, mask, and progress counter losses) decreases steadily and converges quickly during training. This is also observed with the action and progress counter losses. On the other hand, the loss over the the correctly predicted action mask did not decrease and showed no noticeable signs of converge. This result is intuitive as our implemented vision embedding module did not address creating action mask creation explicitly. We believe that our proposed approach utilizing the Mask R-CNN outputs of segmented objects and their classes combined with the grounded Oscar encoding would rectify this shortcoming. Additionally, we notice that the total loss on the validation seen data set did not decrease or show any signs of convergence. However, our loss on the unseen validation set did not converge and began increasing. The lack of convergence in the validation losses is likely due to the model's poor performance on action mask prediction. Moreover, the increasing validation unseen loss is an indication that even when training for less than 5 epochs, the model has been quickly over-fitting to seen scenes. This is a counter-intuitive result as this model relies heavily upon pre-trained backbones which, in theory, still passes lots of information to the model when evaluating unseen trajectories. The over-fitting on the unseen validation set is likely due to the poor performance at predicting object masks. It's likely that without a mask detector the model is just overfitting to the positions of common objects in the environment (i.e. microwave and fridge) which is why it is so poor at generalizing to unseen environments.

2.5.2 PredictorSubgoals

We notice that the overall training loss as well as each individual loss shows decrease. The training loss, progress counter loss, and mask loss appear to fall below the values in the **GroundTruthSubgoals** experiment. This also may be due to slight overfitting on a smaller dataset as this model was training on 40% of the training data and 75% of the validation data (see 3.4). In contrast to the previous experiment, the mask loss appears to decrease slightly with overall significantly lower values than in the **GroundTruthSubgoals** experiment. This is a surprising positive outcome from this condensed language representation. We hypothesize that this is because each sentence contains no more than two objects to interact with and the language is clear and concise (i,e. put egg in sink). Furthermore, when preprocessing, there are objects defined in the trajectory jsons that are not exactly English words (i.e. sink basin represented as SinkBasin). When converting from tuple subgoals back to English, these object tokens were converted back to english (i.e. SinkBasin back to sink basin). Therefore, they will be recognized as real words in the BERT encodings and will not be mapped to [unk] tokens (as SinkBasin would have been). We also hypothesize this "English-ification" of the ALFRED objects to be the reason why the mask loss is decreasing. This conversion back to proper English words was done in **SubgoalPredictor**, but not in **GroundTruthSubgoals**, which is why this decrease did not happen in **GroundTruthSubgoals**. Interestingly, the action loss did not decrease as much as the other losses. This is a counter-intuitive result as the concise language representation was expected to directly assist action prediction. However, the ground truth subgoals appears to have the exact same effect.

2.5.3 PCReweighting

When examining the PC-Reweighting experiment plots, the overall training loss, action loss, and pc increment loss decreases steadily. Overall, the mask loss decreases slightly over the 1.5 epochs trained. However, it still remains at approximately the same value as **GroundTruthSubgoals** experiments but higher than the **PredictorSubgoals**. Combined training loss and action loss still remain higher than the other experiments. This could be due to the reduced training time of only 1.5 epochs. From the results of he first epoch, it appears that the validation seen loss hovers between 1.6 and 2. This is unsurprising as it is similar to the other two experiments. Overall, it appears that PC-reweighting does not add value to decreasing loss beyond the **GroundTrutherSubgoals** experiment. Perhaps we can replicate this experiment with a larger value of

lambda to determine if any effect is visible from reweighting.

2.6 Qualitative Analysis

2.6.1 Baselines Qualitative Analysis

Figure 3 below shows example goal conditions from the language instructions as well as whether or not that goal condition was met by the model. As mentioned previously, common goal conditions met by the model involve cleaning, cooling, and heating tasks. They also involve interacting with an apple, egg, sink, coffee maker, microwave, and garbage can. In general, we noticed that the same goal conditions were met by the model. It is interesting to note that the instructions "Put a cooked apple in the sink" and "Place a cold tomato slice on the counter" follow the same syntactic structure. However, none of the models meet the second goal condition and all of the models meet the first goal condition. The could be due to apples appearing at a higher frequency than tomato in the training data. Furthermore, this discrepancy can also be due to kitchen counter tops being generated in different locations across scenes whereas sinks are in the same place across different environments. Similarly, "Put a cooked apple in the sink" is syntactically similar to "Put heated apple in sink", but only the seq2seq baseline models completed the first condition. One hypothesis as to why that is the case is because in the second example "heated" was not used is previous goal conditions. Therefore, the other baselines had trouble "understanding" that cooked and heated are synonymous.

2.6.2 Experiments Qualitative Analysis

After examining model predictions against ground truth trajectory actions, it appears that both the GroundTruthSubgoal Agent and SubgoalPredictor Agent outputs reasonable action sequences on the validation seen set. The difference between these two agents is their ability to interact with objects. The SubgoalPredictor qualitatively appears to be better at producing both logical interaction and navigation actions while the GroundTruthSubgoal Agent appears to only produce logical navigation actions. Examples of these comparisons are shown in Section 6.

GroundTruthSubgoal Agent When comparing the predicted outputs from this experiment to the ground truth actions, it appears that this model uses navigation actions similar to the ground truth. However, this agent "gets lost" at the beginning and uses one extra PickupObject and PutObject action. This could be due to the poor object mask generation as demonstrated in experiments. After the agent re-corrects, it appears to be moving in the correct direction and rotating correctly. This indicates that this model is stronger at the navigation component of this task as opposed to detecting the correct object masks.

SubgoalPredictor Agent Comparing the predicted outputs from this experiment to the ground truth actions, it appears that this agent is better than the GroundTruthSubgoal Agent at object interactions. Following the action output, it appears that this agent completed the task in fewer actions than the ground truth. While this claim can only be validated running in the simulator, evidence for this are the actions between the first "PickupObject" and "CloseObject". In this sub-sequence, it appears that the agent has picked up an object, navigated to a receptacle, opened a receptacle, placed the object inside, and closed the receptacle. This is a very logical action sequence and is a potential indicator that this agent completed their task early. Since these agents are evaluated statically, their output sizes must match that of the ground truth trajectory. This potentially explains the extra actions at the end of the generated actions after the aforementioned sub-sequence.

3 Future

To improve the existing architecture, we would like to first implement and test the proposed model architecture with OSCAR vision and subgoal embeddings. We believe utilizing a pretrained model designed for generating cross-modality embeddings will be quite informative. OSCAR should also help with our poor mask predictions since OSCAR takes in object masks from MaskRCNN as input.

Finally, we would perform ablations on this architecture to eliminate redundancies. For example, because weighted subgoal information would ideally merge with vision to generate a representation that encompasses both, we may not require feeding the subgoal sequence to the action generator once again.

With respect to subgoal reweighting, we can perform hyperparameter tuning for the parameter lambda from Equation 2. When closer to 0, the reweighting has to effect and as it approached infinity, the soft reweighting turns into a hard mask. We tested a lambda of 1.0 but we can modulate its

strictness to determine if or how the reweighting helps performance.

In a more exploratory direction, one model architecture that we would like to test is a speaker-follower structure ((Fried et al., 2018)). This structure would not only enable us to enforce cycle consistency between language inputs or subgoal intermediate outputs and actions at inference time, but the speaker would serve as a method for data augmentation ((Fried et al., 2018)). As opposed to different types of data, we wish we had more data to work with. Our model had over 100 million trainable parameters, however there were only less than 3,000 trajectories to train on. Having more data and/or developing a data augmentation strategy has been a reliable way to improve performance ((Fried et al., 2018)). We recommend that a future researcher looking to attack this problem starts completely from scratch and does not rely on existing code implementations. Our group experience tremendous difficulty implementing code between different repositories with incompatible python packaging versioning issues as well as limited documentation on an older version of the simulator environment. We recommend that new researchers looking to attack this problem do so without heavily relying on repositories from previous works. Additionally, we recommend a future researcher focus their efforts on novel methods of combining modalities through a multimodal transformer as opposed to focusing on pretrained embeddings of each separate modality. We also recommend future research to focus on the incorporation of subgoals as an additional modality.

4 Ethics

The end goal of any model being trained on ALFRED would be for it to be fine tuned on and deployed to real world household environments. The biggest ethical concern coming from any such deployment is safety. Households are not a well controlled environment where the damages caused by a misbehaving agent could be easily contained. Though interactions are still very high level in ALFRED, ALFRED trains agents to use knives or kitchen appliances like microwaves which could cause safety issues if used improperly. Even without interaction actions, a household agent large enough to do the manipulation tasks required by ALFRED could cause damage from bad navigation alone.

These concerns are important but could still, ignoring the Sim2Real problem any simulator dataset has, be evaluated by looking at the agent's performance on ALFRED itself. However, a less obvious and harder to solve safety issue is the agent's lack of exposure to any humans or other agents in the ALFRED dataset. A sofa looking slightly different in a simulator as opposed to real life does not pose much harm. However, right now in this simulator environment, the ALFRED agent is the only thing changing the environment. The ALFRED agent would not know how to deal with an environment that can be changed and moved from other agents.

Possible negative uses of this technology would be surveillance and a potential invasion of privacy. If an ALFRED-bot was implemented in human households. It would have a constant data on both vision and audio of the domestic environment it is in. Some might consider this a full breach of privacy as all of that data is being sent back to a center to be utilized by engineers. This would impact all of the consumers of the product (most likely upper-middle class working people who would like help with housekeep)

5 Training Plots

X-axis is normally the number of steps in the training loop. For training plots there are roughly 21,000 steps in 1 epoch. For validation plots there are roughly 800 steps per epoch.

For each of these methods, we plot the following plots:

1. loss - total loss during training: loss_action + loss_mask + loss_pc_incr

2. loss_action - loss on whether the correct action was predicted

3. loss_mask- loss on whether the action mask was accurately predicted

4. loss_pc_incr - loss on if the pc increment was predicted correctly

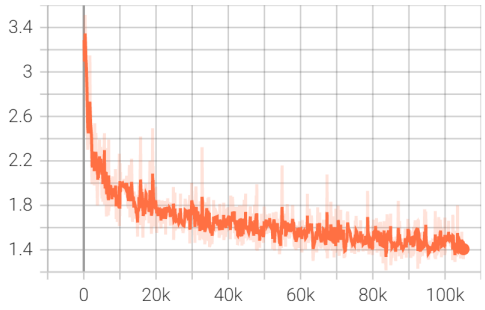
5. valid_seen loss - total loss on valid seen data

6. valid_unseen loss - total loss on valid unseen data

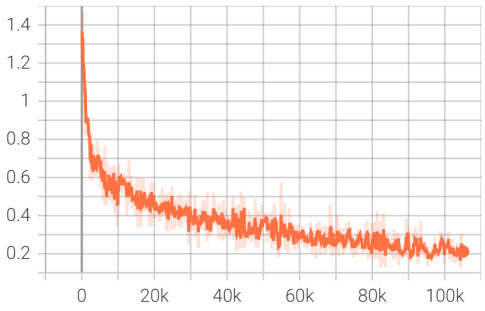
5.1 GroundTruthSubgoals

Plotted over 5 epochs

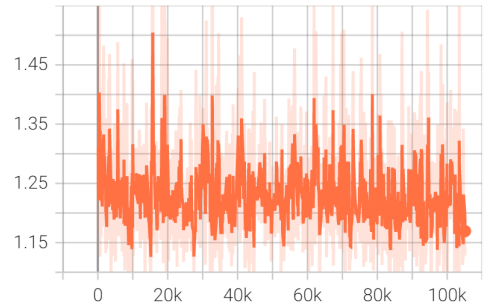
loss
tag: train/loss



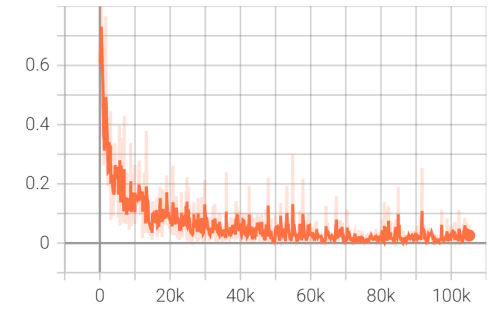
loss_action
tag: train/loss_action



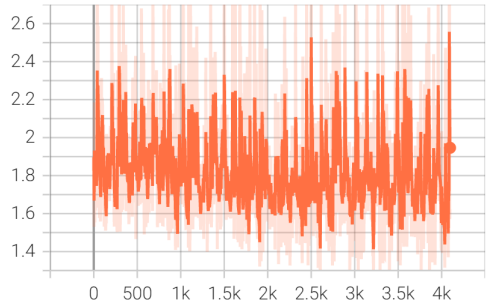
loss_mask
tag: train/loss_mask



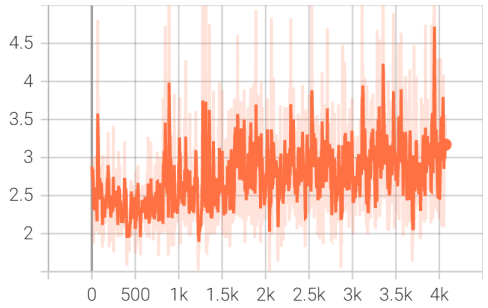
loss_pc_incr
tag: train/loss_pc_incr



loss
tag: valid_seen/loss



loss
tag: valid_unseen/loss

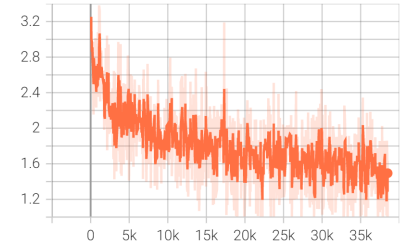


5.2 PredictorSubgoals

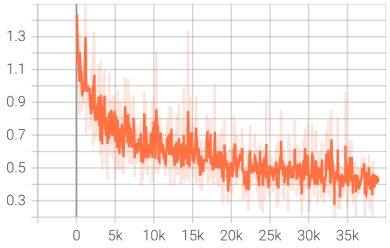
Plotted over 2 epochs

Note, as described in Section 3.4, the model had effectively ∼60% less training data and ∼25% less validation data

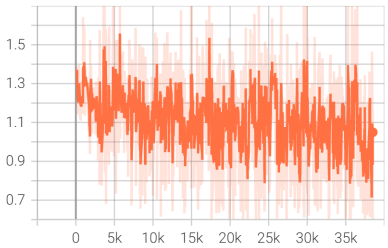
loss
tag: train/loss



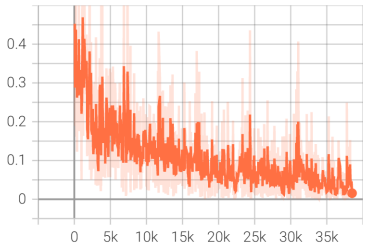
loss_action
tag: train/loss_action



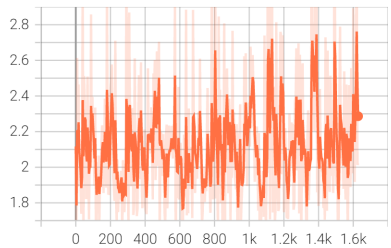
loss_mask
tag: train/loss_mask



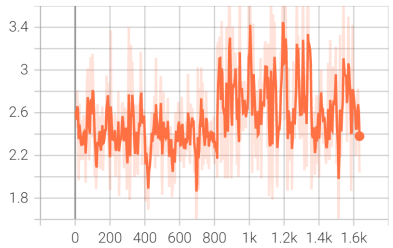
loss_pc_incr
tag: train/loss_pc_incr



loss
tag: valid_seen/loss



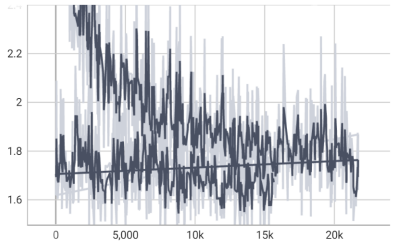
loss
tag: valid_unseen/loss



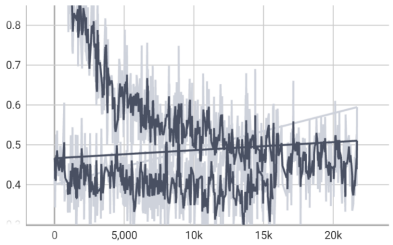
5.3 PCReweighting

Plotted over 1.7 epochs

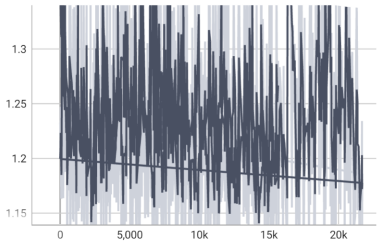
train/loss



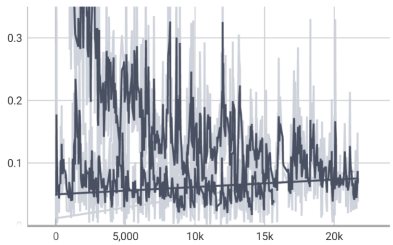
train/loss_action



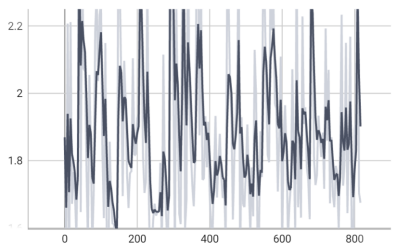
train/loss_mask



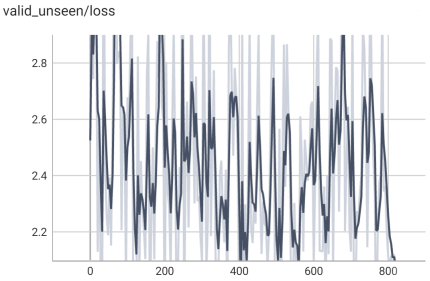
train/loss_pc_incr



valid_seen/loss



valid_unseen/loss



6 Raw Trajectory Examples

Examples of some of the predictions from some of our agents.

6.1 Example 1

GroundTruthSubgoals Evaluated with Ground Truth inputs. On a trajectory from val seen:

action_low is the ground truth true actions that agent should have taken

p_action_low is the action predicted by the agent


```
 "trial_T20190918_184236_557252_2": {
   "lang_goal": "move a knife over to the microwaves cabinet",
   "action_low": [
     "LookDown_15",
     "MoveAhead_25",
     "MoveAhead_25",
     "RotateLeft_90",
     "MoveAhead_25",
     "MoveAhead_25",
     "MoveAhead_25",
     "RotateRight_90",
     "MoveAhead_25",
     "MoveAhead_25",
     "MoveAhead_25",
     "MoveAhead_25",
     "LookUp_15",
     "PickupObject",
     "LookDown_15",
     "RotateRight_90",
     "MoveAhead_25",
     "MoveAhead_25",
     "MoveAhead_25",
     "MoveAhead_25",
     "MoveAhead_25",
     "MoveAhead_25",
     "MoveAhead_25",
     "MoveAhead_25",
     "MoveAhead_25",
     "RotateRight_90",
     "MoveAhead_25",
     "MoveAhead_25",
     "MoveAhead_25",
     "MoveAhead_25",
     "MoveAhead_25",
     "MoveAhead_25",
     "PutObject"
   ],
```

```
  "p_action_low": [
    "LookDown_15",
    "MoveAhead_25",
    "LookDown_15",
    "PickupObject",
    "LookUp_15",
    "RotateLeft_90",
    "MoveAhead_25",
    "RotateLeft_90",
    "RotateLeft_90",
    "LookUp_15",
    "PutObject",
    "LookDown_15",
    "RotateRight_90",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "LookDown_15",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "RotateRight_90",
    "PickupObject",
    "RotateRight_90",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "RotateRight_90",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "RotateRight_90",
    "LookUp_15",
    "PutObject"
  ]
}
```

6.2 Example 2

SubgoalPredictor agent evaluated with subgoal predictor subgoal inputs. On a trajectory from val seen:


```
  "trial_T20190909_061130_844814_2": {
"lang_goal": "Put a clean wooden bowl of water in a cabinet.",
"action_low": [
  "LookDown_15",
  "RotateLeft_90",
  "MoveAhead_25",
  "RotateLeft_90",
  "MoveAhead_25",
  "MoveAhead_25",
  "MoveAhead_25",
  "MoveAhead_25",
  "RotateRight_90",
  "MoveAhead_25",
  "MoveAhead_25",
  "MoveAhead_25",
  "MoveAhead_25",
  "MoveAhead_25",
  "MoveAhead_25",
  "MoveAhead_25",
  "MoveAhead_25",
  "MoveAhead_25",
  "MoveAhead_25",
  "MoveAhead_25",
  "MoveAhead_25",
  "MoveAhead_25",
  "RotateRight_90",
  "MoveAhead_25",
  "MoveAhead_25",
  "MoveAhead_25",
  "MoveAhead_25",
  "RotateRight_90",
  "PickupObject",
  "RotateRight_90",
  "MoveAhead_25",
  "MoveAhead_25",
  "MoveAhead_25",
  "MoveAhead_25",
  "RotateLeft_90",
  "MoveAhead_25",
  "MoveAhead_25",
  "MoveAhead_25",
  "MoveAhead_25",
  "MoveAhead_25",
```

```
    "RotateLeft_90",
    "PutObject",
    "ToggleObjectOn",
    "ToggleObjectOff",
    "PickupObject",
    "RotateLeft_90",
    "RotateLeft_90",
    "MoveAhead_25",
    "LookUp_15",
    "LookUp_15",
    "LookUp_15",
    "LookUp_15",
    "LookUp_15",
    "OpenObject",
    "PutObject",
    "CloseObject"
  ],


  "p_action_low": [
    "LookUp_15",
    "RotateLeft_90",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "LookUp_15",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "MoveAhead_25",
    "LookUp_15",
    "LookUp_15",
    "LookUp_15",
    "LookUp_15",
    "OpenObject",
    "PickupObject",
    "PickupObject",
    "LookDown_15",
    "LookDown_15",
    "LookDown_15",]
```

```
      "RotateRight_90",
      "MoveAhead_25",
      "MoveAhead_25",
      "RotateRight_90",
      "RotateRight_90",
      "MoveAhead_25",
      "RotateRight_90",
      "MoveAhead_25",
      "MoveAhead_25",
      "RotateRight_90",
      "OpenObject",
      "PutObject",
      "CloseObject",
      "MoveAhead_25",
      "MoveAhead_25",
      "MoveAhead_25",
      "LookUp_15",
      "LookUp_15",
      "LookUp_15",
      "LookUp_15",
      "LookUp_15",
      "LookUp_15",
      "OpenObject",
      "PickupObject",
      "PickupObject",
      "LookDown_15",
      "LookDown_15",
      "LookDown_15",
      "LookDown_15",
      "LookDown_15",
      "RotateRight_90",
      "RotateRight_90",
      "RotateRight_90",
      "RotateRight_90",
      "MoveAhead_25",
      "RotateRight_90",
      "OpenObject",
      "OpenObject",
      "CloseObject"
    ]
  }
}
```

7 Team member contributions

Not including contributions reported on previous reports.

Raj Mehta Code: Majority of preprocessesing of data (includes padding, reshaping, extracting) to allow for shape compatibility in models. Also wrote decoder modules and loss functions for training the action generator transformer. Report: Wrote related work section, introduction, and contributed to some areas of section 3.

(Joseph) Cole Ramos **Code**: Various parts of the code base. Primarily the Vision and Language Embeddings, which involved adapting code from Embert. Also wrote the integration code for combining the different subcomponents and debugging latent issues in the Follower that emerged after integration. The GitHub history has a full list of everything I touched. Note the folders alfred_source_utils, embert_utils, and film_utils were mostly copied from other repositories and may inflate my GitHub metrics a bit. **Report** I mostly compiled metrics and plots for use in the final report/ for our own analysis, as well as wrote part of ethics. Most of my work on the report was writing scripts to get information for others to perform analysis on.

Roochi Shah Adapted the subgoal predictor from FILM. Generated subgoal planner outputs to be passed forward in the PredictedSubgoal experiments. Generally assisted teammates with AWS setup. Wrote the entirety of 2.1, 2.4, 2.5, 2.6, and 3. Wrote parts of Ethics and Experiments. Proofreading for report and minor debugging help.

Megahana Tandon Code: Debugging and authoring the action generator and associated positional encoding, subgoal encoding, and modality encodings. Wrote PC reweighting based on Chiang et al. (2021b). Debugged preprocessing. Training: Ran the PCReweighting experiment. Report: Wrote 2.2 Novelty, 2.3 PCReweighting section, miscellaneous formatting and editing,

References

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2015. Neural module networks.

Ting-Rui Chiang, Yi-Ting Yeh, Ta-Chung Chi, and Yau-Shian Wang. 2021a. Are you doing what i say? on modalities alignment in alfred.

Ting-Rui Chiang, Yi-Ting Yeh, Ta-Chung Chi, and Yau-Shian Wang. 2021b. Are you doing what i say? on modalities alignment in alfred.

Daniel Fried, Ronghang Hu, Volkan Cirik, Anna Rohrbach, Jacob Andreas, Louis-Philippe Morency, Taylor Berg-Kirkpatrick, Kate Saenko, Dan Klein, and Trevor Darrell. 2018. Speaker-follower models for vision-and-language navigation.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn.

Xiujun Li, Xi Yin, Chunyuan Li, Pengchuan Zhang, Xiaowei Hu, Lei Zhang, Lijuan Wang, Houdong Hu, Li Dong, Furu Wei, Yejin Choi, and Jianfeng Gao. 2020. Oscar: Object-semantics aligned pre-training for vision-language tasks.

Chih-Yao Ma, Jiasen Lu, Zuxuan Wu, Ghassan AlRegib, Zsolt Kira, Richard Socher, and Caiming Xiong. 2019. Self-monitoring navigation agent via auxiliary progress estimation. *CoRR*, abs/1901.03035.

So Yeon Min, Devendra Singh Chaplot, Pradeep Ravikumar, Yonatan Bisk, and Ruslan Salakhutdinov. 2021. Film: Following instructions in language with modular methods.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Kunal Pratap Singh, Suvaansh Bhambri, Byeonghwi Kim, Roozbeh Mottaghi, and Jonghyun Choi. 2020. Factorizing perception and policy for interactive instruction following.

Alessandro Suglia, Qiaozi Gao, Jesse Thomason, Govind Thattai, and Gaurav S. Sukhatme. 2021. Embodied BERT: A transformer model for embodied, language-guided visual task completion. *CoRR*, abs/2108.04927.

Yichi Zhang and Joyce Chai. 2021. Hierarchical task learning from language instructions with unified transformers and self-monitoring.