Neural Modular Control for Embodied Question Answering

Abhishek Das[1⋆] **Georgia Gkioxari**[2] **Stefan Lee**[1] **Devi Parikh**[1,2] **Dhruv Batra**[1,2]

[1]Georgia Institute of Technology [2]Facebook AI Research

Abstract:

We present a modular approach for learning policies for navigation over long planning horizons from language input. Our hierarchical policy operates at multiple timescales, where the higher-level master policy proposes subgoals to be executed by specialized sub-policies. Our choice of subgoals is compositional and semantic, *i.e*. they can be sequentially combined in arbitrary orderings, and assume human-interpretable descriptions (*e.g*. 'exit room', 'find kitchen', 'find refrigerator', *etc*.).

We use imitation learning to warm-start policies at each level of the hierarchy, dramatically increasing sample efficiency, followed by reinforcement learning. Independent reinforcement learning at each level of hierarchy enables sub-policies to adapt to consequences of their actions and recover from errors. Subsequent joint hierarchical training enables the master policy to adapt to the sub-policies.

On the challenging EQA [1] benchmark in House3D [2], requiring navigating diverse realistic indoor environments, our approach outperforms prior work by a significant margin, both in terms of navigation and question answering.

1 Introduction

Abstraction is an essential tool for navigating our daily lives. When seeking a late night snack, we certainly do not spend time planning out the mechanics of walking and are thankfully also unburdened of the effort of recalling to beat our heart along the way. Instead, we conceptualize our actions as a series of higher-level semantic goals – `exit bedroom; go to kitchen; open fridge; find snack;` – each of which is executed through specialized coordination of our perceptual and sensorimotor skills. This ability to abstract long, complex sequences of actions into semantically meaningful subgoals is a key component of human cognition [3] and it is natural to believe that artificial agents can benefit from applying similar mechanisms when navigating our world.

We study such hierarchical control in the context of a recently proposed task – Embodied Question Answering (EmbodiedQA) [1] – where an embodied agent is spawned at a random location in a novel environment (*e.g*. a house) and asked to answer a question (*'What color is the piano in the living room?'*). To do so, the agent must navigate from egocentric vision alone (without access to a map of the environment), locate the entity in question (*'piano in the living room'*), and respond with the correct answer (*e.g*. *'red'*). From a reinforcement learning (RL) perspective, EmbodiedQA presents challenges that are known to make learning particularly difficult – partial observability, planning over long time horizons, and sparse rewards – the agent may have to navigate through multiple rooms in search for the answer, executing hundreds of primitive motion actions along the way (`forward; forward; turn-right; ...`) and receiving a reward based only on its final answer.

To address this challenging learning problem, we develop a hierarchical **Neural Modular Controller (NMC)** – consisting of a *master* policy that determines high-level *subgoals*, and *sub-policies* that execute a series of low-level actions to achieve these subgoals. Our NMC model constructs a hierarchy that is arguably natural to this problem – navigation to rooms and objects *vs*. low-level motion actions. For example, NMC seeks to break down a question *'What color is the piano in the living room?'* to the series of subgoals `exit-room; find-room[living]; find-object[piano]; answer;` and execute this plan with specialized neural 'modules' corresponding to each subgoal. Each module is trained to issue a variable length series of primitive actions to achieve its titular subgoal – *e.g*. the

⋆Work partially done during an internship at Facebook AI Research.

2nd Conference on Robot Learning (CoRL 2018), Zürich, Switzerland.

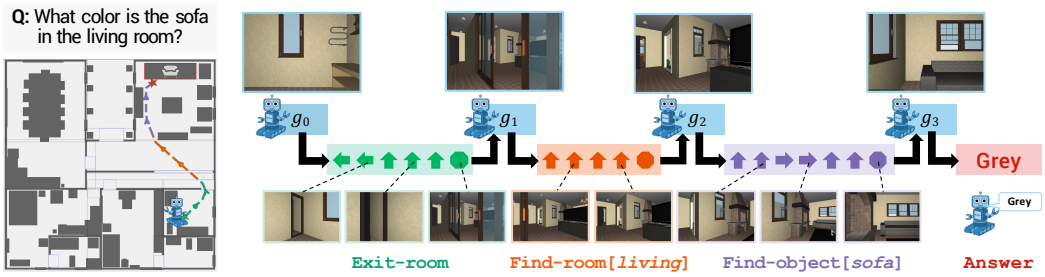


Figure 1: We introduce a hierarchical policy for Embodied Question Answering. Given a question ("What color is the sofa in the living room?") and observation, our master policy predicts a sequence of subgoals – `Exit-room`, `Find-room[living]`, `Find-object[sofa]`, `Answer` – that are then executed by specialized sub-policies to navigate to the target object and answer the question ("Grey").

`find-object[piano]` module is trained to navigate the agent to the input argument *piano* within the current room. Disentangling semantic subgoal selection from sub-policy execution results in easier to train models due to shorter time horizons. Specifically, this hierarchical structure introduces:

– **Compressed Time Horizons:** The master policy makes orders of magnitude fewer decisions over the course of a navigation than a *'flat model'* that directly predicts primitive actions – allowing the answering reward in EmbodiedQA to more easily influence high-level motor control decisions.
– **Modular Pretraining:** As each module corresponds to a specific task, they can be trained independently before being combined with the master policy. Likewise, the master policy can be trained assuming ideal modules. We do this through imitation learning [4, 5] sub-policies.
– **Interpretability:** The predictions made by the master policy correspond to semantic subgoals and exposes the reasoning of the agent to inspection (*'What is the agent trying to do right now?'*) in a significantly more interpretable fashion than just its primitive actions.

First, we learn and evaluate master and sub-policies for each of our subgoals, trained using behavior cloning on expert trajectories, reinforcement learning from scratch, and reinforcement learning after behavior cloning. We find that reinforcement learning after behavior cloning dramatically improves performance over each individual training regime. We then evaluate our combined hierarchical approach on the EQA [1] benchmark in House3D [2] environments. Our approach significantly outperforms prior work both in navigational and question answering performance – our agent is able to navigate closer to the target object and is able to answer questions correctly more often.

2 Related Work

Our work builds on and is related to prior work in hierarchical reinforcement and imitation learning, grounded language learning, and embodied question-answering agents in simulated environments.

Hierarchical Reinforcement and Imitation Learning. Our formulation is closely related to Le *et al*. [6], and can be seen as an instantiation of the options framework [7, 8], wherein a global master policy proposes subgoals – to be achieved by local sub-policies – towards a downstream task objective [9–11]. Relative to other work on automatic subgoal discovery in hierarchical reinforcement learning [12–14], we show that given knowledge of the problem structure, simple heuristics are quite effective in breaking down long-range planning into sequential subgoals. We make use of a combination of hierarchical behavior cloning [4] and actor-critic [15] to train our modular policy.

Neural Module Networks and Policy Sketches. At a conceptual-level, our work is analogous to recent work on neural module networks (NMNs) [16–18] for visual question answering. NMNs first predict a 'program' from the question, consisting of a sequence of primitive reasoning steps, which are then executed on the image to obtain the answer. Unlike NMNs, where each primitive reasoning module has access to the entire image (completely observable) our setting is partially observable – each sub-policy only has access to first-person RGB – making active re-evaluation of subgoals after executing each sub-policy essential. Our work is also closely related to policy sketches [16], which are symbolic descriptions of subgoals provided to the agent without any grounding or sub-policy for executing them. There are two key differences w.r.t. to our work. First, an important framework difference – Andreas *et al*. [16] assume access to a policy sketch *at test time*, *i.e*. for every task to be performed. In EmbodiedQA, this would correspond to the agent being provided with a high-level

plan (`exit-room; find-room[living]; ...`) for every question it is ever asked, which is an unrealistic assumption in real-world scenarios with a robot. In contrast, we assume that subgoal supervision (in the form of expert demonstrations and plans) are available on training environments but not on test, and the agent must *learn* to produce its own subgoals. Second, a subtle but important implementation difference – unlike [16], our sub-policy modules accept input arguments that are embeddings of target rooms and objects (*e.g.*`find-room[living], find-object[piano]`). This results in our sub-policy modules being shared not just across tasks (questions) as in [16], but also across instantiations of *similar* navigation sub-policies – *i.e.*, `find-object[piano]` and `find-object[chair]` share parameters that enable data efficient learning without exhaustively learning separate policies for each.

Grounded Language Learning. Beginning with SHRDLU [19], there has been a rich progression of work in grounding language-based goal specifications into actions and pixels in physically-simulated environments. Recent deep reinforcement learning-based approaches to this explore it in 2D gridworlds [16, 20, 21], simple visual [22–27] and textual [28, 29] environments, perceptually-realistic 3D home simulators [1, 30–33], as well as real indoor scenes [34–36]. Our hierarchical policy learns to ground words from the question into two levels of hierarchical semantics. The master policy grounds words into subgoals (such as `find-room[kitchen]`), and sub-policies ground these semantic targets (such as `cutting board`, `bathroom`) into primitive actions and raw pixels, both parameterized as neural control policies and trained end-to-end.

Embodied Question-Answering Agents. Finally, hierarchical policies for embodied question answering have previously been proposed by Das *et al*. [1] in the House3D environment [2], and by Gordon *et al*. [30] in the AI2-THOR environment [37]. Our hierarchical policy, in comparison, is human-interpretable, *i.e*. the subgoal being pursued at every step of navigation is semantic, and due to the modular structure, can navigate over longer paths than prior work, spanning multiple rooms.

3 Neural Modular Control

We now describe our approach in detail. Recall that given a question, the goal of our agent is to predict a sequence of navigation subgoals and execute them to ultimately find the target object and respond with the correct answer. We first present our modular hierarchical policy. We then describe how we extract optimal plans from shortest path navigation trajectories for behavior cloning. And finally, we describe how the various modules are combined and trained with a combination of imitation learning (behavior cloning) and reinforcement learning.

3.1 Hierarchical Policy

Notation. Recall that NMC has 2 levels in the hierarchy – a master policy that generates subgoals and sub-policies for each of these subgoals. We use i to index the sequence of subgoals and t to index actions generated by sub-policies. Let $\mathcal{S} = \{s\}$ denote the set of states, $\mathcal{G} = \{g\}$ the set of variable-time subgoals with elements $g = \langle g_{\text{task}}, g_{\text{argument}} \rangle$, *e.g.* $g = \langle$`exit-room,None`$\rangle$, or $g = \langle$`find-room,bedroom`$\rangle$. Let $\mathcal{A} = \{a\}$ be the set of primitive actions (`forward`, `turn-left`, `turn-right`). The learning problem can then be succinctly put as learning a master policy $\pi_\theta : \mathcal{S} \to \mathcal{G}$ parameterized by θ and sub-policies $\pi_{\phi_g} : \mathcal{S} \to \mathcal{A} \cup \{\texttt{stop}\}$ parameterized by ϕ_g, $\forall g \in \mathcal{G}$, where the `stop` action terminates a sub-policy and returns control to the master policy.

While navigating an environment, control alternates between the master policy selecting subgoals and sub-policies executing these goals through a series of primitive actions. More formally, given an initial state s_0 the master policy predicts a subgoal $g_0 \sim \pi_\theta(g|s_0)$, the corresponding sub-policy executes until some time T_0 when either (1) the sub-policy terminates itself by producing the `stop` token $a_{T_0} \sim \pi_{\phi_{g_0}}(a|s_{T_0}) =$ `stop` or (2) a maximum number of primitive actions has been reached. Either way, this returns the control back to the master policy which predicts another subgoal and repeats this process until termination. This results in a state-subgoal trajectory:

$$\Sigma = \Big(\underbrace{s_0, g_0}_{\text{subgoal } 0}, \underbrace{s_{T_0}, g_1}_{\text{subgoal } 1}, \ldots, \underbrace{s_{T_i}, g_{i+1}}_{\text{subgoal } i}, \ldots, \underbrace{s_{T_{\mathcal{T}-1}}, g_{\mathcal{T}}}_{\text{subgoal } \mathcal{T}} \Big) \tag{1}$$

for the master policy. Notice that the terminal state of the i^{th} sub-policy s_{T_i} forms the state for the master policy to predict the next subgoal g_{i+1}. For the $(i+1)^{\text{th}}$ subgoal g_{i+1}, the low-level trajectory

Subgoal	Argument(s)	Description	Success
`Exit-room`	None	When there is only 1 door in spawn room, or 1 door other than door entered through in an intermediate room; agent is forced to use the remaining door.	Stopping after exiting through the correct door.
`Find-room`	Room name (*gym, kitchen, ...*)	When there are multiple doors and the agent has to search and pick the door to the target room.	Stopping after entering target room.
`Find-object`	Object name (*oven, sofa, ...*)	When the agent has to search for a specific object in room.	Stopping within 0.75m of the target object.
`Answer`	None	When the agent has to provide an answer from the answer space.	Generating the correct answer to the question.

Table 1: Descriptions of our subgoals and conditions we use to extract them automatically from expert trajectories.

of states and primitive actions is given by:

$$\sigma_{g_{i+1}} = \Big(\underbrace{s_{T_i}, a_{T_i}}_{\text{action 0}}, \underbrace{s_{T_i+1}, a_{T_i+1}}_{\text{action 1}}, \ldots, \underbrace{s_{T_i+t}, a_{T_i+t}}_{\text{action t}}, \ldots, s_{T_{i+1}} \Big). \tag{2}$$

Note that by concatenating all sub-policy trajectories in order $(\sigma_{g_0}, \sigma_{g_1}, \ldots, \sigma_{g_\mathcal{T}})$, the entire trajectory of states and primitive actions can be recovered.

Subgoals $\langle$Tasks, Arguments$\rangle$. As mentioned above, each subgoal is factorized into a task and an argument $g = \langle g_{\text{task}}, g_{\text{argument}} \rangle$. There are 4 possible tasks – `exit-room`, `find-room`, `find-object`, and `answer`. Tasks `find-object` and `find-room` accept as arguments one of the 50 objects and 12 room types in EQA v1 dataset [1] respectively; `exit-room` and `answer` do not accept any arguments. This gives us a total of $50 + 12 + 1 + 1 = 64$ subgoals.

$\langle$`exit-room,none`$\rangle$, $\langle$`answer,none`$\rangle$, $\}$ 0 args
$\langle$`find-object,couch`$\rangle$, $\langle$`find-object,cup`$\rangle$, ..., $\langle$`find-object,xbox`$\rangle$, $\}$ 50 args
$\langle$`find-room,living`$\rangle$, $\langle$`find-room,bedroom`$\rangle$, ..., $\langle$`find-room,patio`$\rangle$. $\}$ 12 args

Descriptions of these tasks and their success criteria are provided in Table 1.

Master Policy. The master policy π_θ parameterized by θ is implemented as a single layer Gated Recurrent Unit (GRU). At each high-level step $i+1$, the master policy $\pi_\theta(g|s_{T_i})$ takes as input the concatenation of a encoding of the question $q \in \mathbb{R}^{128}$, the image feature $v_{T_i} \in \mathbb{R}^{128}$ of the current frame and an encoding $o_i \in \mathbb{R}^{32}$ computed from a 1-hot representation of the i^{th} subgoal, *i.e.* $\mathbb{1}(g_i)$. This information is used to update the hidden state $h_i \in \mathbb{R}^{1048}$ that encodes the entire trajectory up to time t and serves as the state representation. The policy then produces a probability distribution over all possible (64) subgoals $\mathcal{G}$. We train these policies with actor-critic methods and thus the network also produces a value estimate.

Sub-policies. To take advantage of the comparatively lower number of subgoal tasks, we decompose sub-policy parameters ϕ_g into $\phi_{g_{\text{task}}}$ and $\phi_{g_{\text{argument}}}$, where $\phi_{g_{\text{task}}}$ are shared across the same task and $\phi_{g_{\text{argument}}}$ is an argument specific embedding. Parameter sharing enables us to learn the shared task in a sample-efficient manner, rather than exhaustively learning separate sub-policies for each combination.

Like the master policy, each sub-policy π_{ϕ_g} is implemented as a single-layer GRU. At each low-level time step t, a sub-policy $\pi_{\phi_g}(a|s_t)$ takes as input the concatenation of the image feature $v_t \in \mathbb{R}^{128}$ of the current frame, an encoding $p_{t-1} \in \mathbb{R}^{32}$ computed from a 1-hot representation of the previous primitive action *i.e.* $\mathbb{1}(a_{t-1})$, and the argument embedding $\phi_{g_{\text{argument}}}$. These inputs are used to update the hidden state $h_t^g \in \mathbb{R}^{1048}$ which serves as the state representation. The policy then outputs a distribution over primitive actions (`forward, turn-left, turn-right, stop`). As with the master policy, each sub-policy also output a value estimate. shows this model structure.

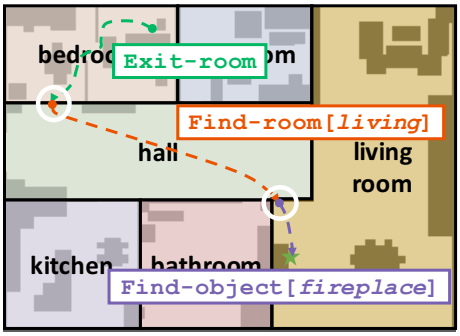
Perception and Question Answering. To ensure fair comparisons to prior work, we use the same perception and question answering models as used by Das *et al.* [1]. The perception model is a simple

convolutional neural network trained to perform auto-encoding, semantic segmentation, and depth estimation from RGB frames taken from House3D [2]. Like [1], we use the bottleneck layer of this model as a fixed feature extractor. We also use the same post-navigational question-answering model as [1], which encodes the question with a 2-layer LSTM and performs dot-product based attention between the question encoding and the image features from the last five frames along the navigation path right before the `answer` module is called. This post-navigational answering module is trained using visual features along the shortest path trajectories and then frozen. By keeping these parts of the architecture identical to [1], our experimental comparisons can focus on the differences only due to our contributions, the Neural Modular Controller.

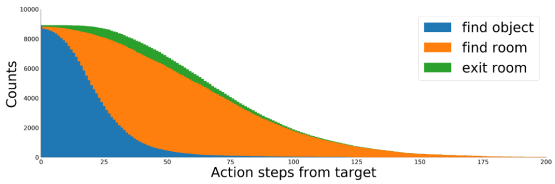
3.2 Hierarchical Behavior Cloning from Expert Trajectories

The questions in EQA v1 dataset [1] (*e.g.* *'What color is the fireplace?'*) are constructed to inquire about attributes (color, location, *etc.*) of specific target objects (*'fireplace'*). This notion of a target enables the construction of an automatically generated *expert trajectory* $(s_0^*, a_0^*, \ldots, s_T^*, a_T^*)$ – the states and actions along the shortest path from the agent spawn location to the object of interest specified in the question. Notice that these shortest paths may only be used as supervision on training environments but may not be utilized during evaluation on test environments (where the agent must operate from egocentric vision alone).

Specifically, we would like to use these expert demonstrations to pre-train our proposed NMC navigator using behavior cloning. However, these trajectories $(s_0^*, a_0^*, \ldots, s_T^*, a_T^*)$ correspond to a series of primitive actions. To provide supervision for both the master policy and sub-policies, these shortest-path trajectories must be annotated with a sequence of subgoals and segmented into their respective temporal extents, resulting in Σ^* and $(\sigma_{g_i}^*)$.



(a) **Q**: What color is the fireplace? **A**: Brown



(b) Distribution of subgoals with number of actions from the target object as per expert plans. Closer to the target object, the expert plan predominantly consists of `Find-object`, while as we move farther away, the proportion of `Find-room` and `Exit-room` goes up.

Figure 2: We extract expert subgoal trajectories from shortest paths by dividing paths on room transition boundaries (circled in (a))and following the rules in Tab. 1.

We automate this 'lifting' of annotation up the hierarchy by leveraging the object and room bounding boxes provided by the House3D [2]. Essentially, a floor plan may be viewed as an undirected graph with rooms as nodes and doorways as edges connecting a pair of adjacent rooms. An example trajectory is shown in Fig. 2a for the question *'What color is the fireplace?'*. The agent is spawned in a bedroom, the shortest path exits into the hall, enters the living room, and approaches the fireplace. We convert this trajectory to the subgoal sequence (`exit-room, find-room[living], find-object[fireplace], answer`) by recording the transitions on the shortest path from one room to another, which also naturally provides us with temporal extents of these subgoals.

We follow a couple of subtle but natural rules: (1) `find-object` is tagged only when the agent has reached the destination room containing the target object; and (2) `exit-room` is tagged only when the 'out-degree' of the current room in the floor-plan-graph is exactly 1 (*i.e.* either the current room has exactly one doorway or the current room has two doorways but the agent came in through one). Rule (2) ensures a semantic difference between `exit-room` and `find-room` – informally, `exit-room` means *'get me out of here'* and `find-room[name]` means *'look for room* `name`*'*.

Tab. 1 summarizes these subgoals and the heuristics used to automatically extract them from navigational paths. Fig. 2b shows the proportions of these subgoals in expert trajectories as a function of the distance from target object. Notice that when the agent is close to the target, it is likely to be within

the same room as the target and thus `find-object` dominates. On the other hand, when the agent is far away from the target, `find-room` and `exit-room` dominate.

We perform this lifting of shortest paths for all training set questions in EQA v1 dataset [1], resulting in N expert trajectories $\{\Sigma^*_n\}_{n=1}^N$ for the master policy and $K(>> N)$ trajectories $\{\sigma^*_{g_k}\}_{k=1}^K$ for sub-policies. We can then perform hierarchical behavior cloning by minimizing the sum of cross-entropy losses over all decisions in all expert trajectories. As is typical in maximum-likelihood training of directed probabilistic models (*e.g.* hierarchical Bayes Nets), full supervision results in decomposition into independent sub-problems. Specifically, with a slight abuse of notation, let $(s^*_i, g^*_{i+1}) \in \Sigma^*$ denote an iterator over all state-subgoal tuples in Σ^*, and $\sum_{(s^*_i, g^*_{i+1})\in\Sigma^*}$ denote a sum over such tuples.

Now, the independent learning problems can be written as:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \sum_{n=1}^{N} \sum_{(s^*_i, g^*_{i+1})\in\Sigma^*_n} -\log\left(\pi_\theta(g^*_{i+1}|s^*_i)\right) \quad \text{(master policy cloning)} \tag{3a}$$

$$\phi^*_g = \underset{\phi}{\operatorname{argmin}} \underbrace{\sum_{k=1}^{K} [\![g_k = g]\!]}_{\text{demonstrations}} \underbrace{\sum_{(s^*_t, a^*_{t+1})\in\sigma^*_{g_k}}}_{\text{transitions}} \underbrace{-\log\left(\pi_{\phi_g}(a^*_{t+1}|s^*_t)\right)}_{\text{negative-log-likelihood}} \quad \text{(sub-policy cloning)} \tag{3b}$$

Intuitively, each sub-policy independently maximizes the conditional probability of actions observed in the expert demonstrations, and the master policy essentially trains assuming perfect sub-policies.

3.3 Asynchronous Advantage Actor-Critic (A3C) Training

After the independent behavior cloning stage, the policies have learned to mimic expert trajectories; however, they have not had to coordinate with each other or recover from their own navigational errors. As such, we fine-tune them with reinforcement learning – first independently and then jointly.

Reward Structure. The ultimate goal of our agent is to answer questions accurately; however, doing so requires navigating the environment sufficiently well in search of the answer. We mirror this structure in our reward R, decomposing it into a sum of a sparse terminal reward R_{terminal} for the final outcome and a dense, shaped reward R_{shaped} [38] determined by the agent's progress towards its goals. For the master policy π_θ, we set R_{terminal} to be 1 if the model answers the question correctly and 0 otherwise. The shaped reward R_{shaped} at master-step i is based on the change of navigable distance to the target object before and after executing subgoal g_i. Each sub-policy π_{ϕ_g} also has a terminal 0/1 reward R_{terminal} for stopping in a successful state, *e.g.* `Exit-room` ending outside the room it was called in (see Tab. 1 for all success definitions). Like the master policy, R_{shaped} at time t is set according to the change in navigable distance to the sub-policy target (*e.g.* a point just inside a living room for `find-room[living]`) after executing the primitive action a_t. Further, sub-policies are also penalized a small constant (-0.02) for colliding with obstacles.

Policy Optimization. We update the master and sub-policies to minimize expected discounted future rewards $J(\pi_\theta)$ and $J(\pi_{\phi_g})$ respectively through the Asynchronous Advantage Actor Critic [15] policy-gradient algorithm. Specifically, for the master policy, the gradient of the expected reward is written as:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}\left[\nabla_\theta \log(\pi_\theta(g_i|s_{T_i}))\left(Q(s_{T_i}, g_i) - c_\theta(s_{T_i})\right)\right] \tag{4}$$

where $c_\theta(s_{T_i})$ is the estimated value of s_{T_i} produced by the critic for π_θ. To further reduce variance, we follow [39] and estimate $Q(s_{T_i}, g_i) \approx R_\theta(s_{T_i}) + \gamma c_\theta(s_{T_{i+1}})$ such that $Q(s_{T_i}, g_i) - c_\theta(s_{T_i})$ computes a generalized advantage estimator (GAE). Similarly, each sub-policy π_{ϕ_g} is updated according to the gradient

$$\nabla_{\phi_g} J(\pi_{\phi_g}) = \mathbb{E}\left[\nabla_\theta \log(\pi_{\phi_g}(a_i|s_i))\left(Q(s_i, a_i) - c_{\phi_g}(s_i)\right)\right]. \tag{5}$$

Recall from Section 3.1 that these critics share parameters with their corresponding policy networks such that subgoals with a common task also share a critic. We train each policy network independently using A3C [15] with GAE [39] with 8 threads across 4 GPUs. After independent reinforcement fine-tuning of the sub-policies, we train the master policy further using the trained sub-policies rather than expert subgoal trajectories.

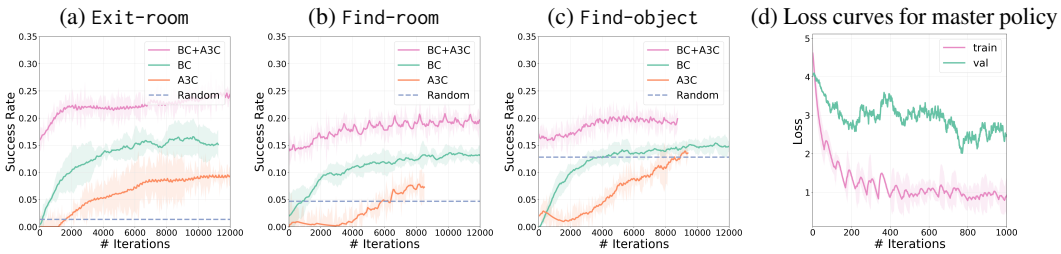


Figure 3: (a,b,c) Success rate over training iterations for each sub-policy task using behavior cloning (BC), reinforcement learning from scratch (A3C), and reinforcement finetuning after behavior cloning (BC+A3C) training regimes. We find BC+A3C significantly outperforms either BC or A3C alone. Each of these is averaged over 5 runs. (d) Losses for master policy during behavior cloning *i.e.* assuming access to perfect sub-policies.

Initial states and curriculum. Rather than spawn agents at fixed distances from target, from where accomplishing the subgoal may be arbitrarily difficult, we sample locations along expert trajectories for each question or subgoal. This ensures that even early in training, policies are likely to have a mix of positive and negative reward episodes. At the beginning of training, all points along the trajectory are equally likely; however, as training progresses and success rate improves, we reduce the likelihood of sampling points nearer to the goal. This is implemented as a multiplier α on available states $[s_0, s_1, ..., s_{\alpha T}]$, initialized to 1.0 and scaled by 0.9 whenever success rate crosses a 40% threshold.

4 Experiments and Results

Dataset. We benchmark performance on the EQA v1 dataset [1], which contains $\sim 9,000$ questions in 774 environments – split into $7129(648)$ / $853(68)$ / $905(58)$ questions (environments) for training/validation/testing respectively[1]. These splits have no overlapping environments between them, thus strictly checking for generalization to novel environments. We follow the same splits.

Evaluating sub-policies. We begin by evaluating the performance of each sub-policy with regard to its specialized task. For clarity, we break results down by subgoal task rather than for each task-argument combination. We compare sub-policies trained with behavior cloning (**BC**), reinforcement learning from scratch (**A3C**), and reinforcement fine-tuning after behavior cloning (**BC+A3C**). We also compare to a **random** agent that uniformly samples actions including `stop` to put our results in context. For each, we report the success rate (as defined in Tab. 1) on the EQA v1 validation set which consists of 68 novel environments unseen during training. We spawn sub-policies at randomly selected suitable rooms (*i.e.* `Find-object[sofa]` will only be executed in a room with a sofa) and allow them to execute for a maximum episode length of 50 steps or until they terminate.

Fig. 3 shows success rates for the different subgoal tasks over the course of training. We observe that:

- **Behavior cloning (BC) is more sample-efficient than A3C from scratch.** Sub-policies trained using BC improve significantly faster than A3C for all tasks, and achieve higher success rates for `Exit-room` and `Find-room`. Interestingly, this performance gap is larger for tasks where a random policy does *worse* – implying that BC helps more as task complexity increases.
- **Reinforcement Fine-Tuning with A3C greatly improves over BC training alone.** Initializing A3C with a policy trained via behavior cloning results in a model that significantly outperforms either approach on its own, nearly doubling the success rate of behavior cloning for some tasks. Intuitively, mimicking expert trajectories in behavior cloning provides dense feedback for agents about how to navigate the world; however, agents never have to face the consequences of erroneous actions *e.g.* recovering from collisions with objects – a weakness that A3C fine-tuning addresses.

Evalating master policy. Next, we evaluate how well the master policy performs during independent behavior cloning on expert trajectories *i.e.* assuming perfect sub-policies, as specified in Eq. 3a. Even though there is no overlap between training and validation environments, the master policy is able to generalize reasonably and gets $\sim 48\%$ intersection-over-union (IoU) with ground truth subgoal sequences on the validation set. Note that a sequence of sub-goals that is different from the one corresponding to the shortest path may still be successful at navigating to the target object and answering the question correctly. In that sense, IoU against ground truth subgoal sequences is a strict metric. Fig. 3d shows the training and validation cross-entropy loss curves for the master policy.

[1]Note that the size of the publicly available dataset on `embodiedqa.org/data` is larger than the one reported in the original version of the paper due to changes in labels for color questions.

	Navigation									QA		
	$\mathbf{d_0}$ (For reference)			$\mathbf{d_T}$ (Lower is better)			$\mathbf{d_\Delta}$ (Higher is better)			**accuracy** (Higher is better)		
	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}
PACMAN (BC) [1]	1.15	4.87	9.64	1.19	4.25	8.12	-0.04	0.62	1.52	48.48%	40.59%	39.87%
PACMAN (BC+REINFORCE) [1]	1.15	4.87	9.64	**1.05**	4.22	8.13	**0.10**	0.65	1.51	50.21%	42.26%	40.76%
NMC (BC)	1.15	4.87	9.64	1.44	4.14	8.43	-0.29	0.73	1.21	43.14%	41.96%	38.74%
NMC (BC+A3C)	1.15	4.87	9.64	1.06	**3.72**	**7.94**	0.09	**1.15**	**1.70**	**53.58%**	**46.21%**	**44.32%**

Table 2: Evaluation of EmbodiedQA agents on navigation and answering metrics for the EQA v1 test set.

Evalating NMC. Finally, we put together the master and sub-policies and evaluate navigation and question answering performance on EmbodiedQA. We compare against the PACMAN model proposed in [1]. For accurate comparison, both PACMAN and NMC use the same publicly available and frozen pretrained CNN[2], and the same visual question answering model – pretrained to predict answers from last 5 observations of expert trajectories, following [1]. Agents are evaluated by spawning 10, 30, or 50 primitive actions away from target, which corresponds to distances of 1.15, 4.87, and 9.64 meters from target respectively, denoted by $\mathbf{d_0}$ in Tab. 2. When allowed to run free from this spawn location, $\mathbf{d_T}$ measures final distance to target (how far is the agent from the goal at termination), and $\mathbf{d_\Delta} = \mathbf{d_T} - \mathbf{d_0}$ evaluates change in distance to target (how much progress does the agent make over the course of its navigation). Answering performance is measured by **accuracy** (*i.e.* did the predicted answer match ground-truth). Note that [1] report a number of additional metrics (percentage of times the agent stops, retrieval evaluation of answers, *etc.*). Accuracies for PACMAN are obtained by running the publicly available codebase released by authors[2], and numbers are different than those reported in the original version of [1] due to changes in the dataset[1].

As shown in Tab. 2, we evaluate two versions of our model – 1) NMC (BC) naively combines master and sub-policies without A3C finetuning at any level of hierarchy, and 2) NMC (BC+A3C) is our final model where each stage is trained with BC+A3C, as described in Sec. 3. As expected, NMC (BC) performs worse than NMC (BC + A3C), evident in worse navigation $\mathbf{d_T}$, $\mathbf{d_\Delta}$ and answering **accuracy**. PACMAN (BC) and NMC (BC) go through the same training regime, and there are no clear trends as to which is better – PACMAN (BC) has better $\mathbf{d_\Delta}$ and answering **accuracy** at T_{-10} and T_{-50}, but worse at T_{-30}. No A3C finetuning makes it hard for sub-policies to recover from erroneous primitive actions, and for master policy to adapt to sub-policies. A3C finetuning significantly boosts performance, *i.e.* NMC (BC + A3C) outperforms PACMAN with higher $\mathbf{d_\Delta}$ (makes more progress towards target), lower $\mathbf{d_T}$ (terminates closer to target), and higher answering **accuracy**. This gain primarily comes from the choice of subgoals and the master policy's ability to explore over this space of subgoals instead of primitive actions (as in PACMAN), enabling the master policy to operate over longer time horizons, critical for sparse reward settings as in EmbodiedQA.

5 Conclusion

We introduced Neural Modular Controller (NMC), a hierarchical policy for EmbodiedQA consisting of a master policy that proposes a sequence of semantic subgoals from question (*e.g.* *'What color is the sofa in the living room?'* → `Find-room[living]`, `Find-object[sofa]`, `Answer`), and specialized sub-policies for executing each of these tasks. The master and sub-policies are trained using a combination of behavior cloning and reinforcement learning, which is dramatically more sample-efficient than each individual training regime. In particular, behavior cloning provides dense feedback for how to navigate, and reinforcement learning enables policies to deal with consequences of their actions, and recover from errors. The efficacy of our proposed model is demonstrated on the EQA v1 dataset [1], where NMC outperforms prior work both in navigation and question answering.

Acknowledgments

This work was supported in part by NSF, AFRL, DARPA, Siemens, Google, Amazon, ONR YIPs and ONR Grants N00014-16-1-{2713,2793}. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government, or any sponsor.

[2] github.com/facebookresearch/EmbodiedQA

References

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

[2] Y. Wu, Y. Wu, G. Gkioxari, and Y. Tian. Building Generalizable Agents With a Realistic And Rich 3D Environment. *arXiv preprint arXiv:1801.02209*, 2018.

[3] B. Hayes-Roth and F. Hayes-Roth. A cognitive model of planning. *Cognitive Science*, 1979.

[4] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004.

[5] S. Ross and J. A. Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

[6] H. M. Le, N. Jiang, A. Agarwal, M. Dudík, Y. Yue, and H. Daumé III. Hierarchical imitation and reinforcement learning. In *ICML*, 2018.

[7] R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 1999.

[8] R. S. Sutton, D. Precup, and S. P. Singh. Intra-option learning about temporally abstract actions. In *ICML*, 1998.

[9] P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *AAAI*, 2017.

[10] T. D. Kulkarni, K. R. Narasimhan, A. Saeedi, and J. B. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NIPS*, 2016.

[11] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *AAAI*, 2017.

[12] B. Bakker and J. Schmidhuber. Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization. In *IAS*, 2004.

[13] S. Goel and M. Huber. Subgoal discovery for hierarchical reinforcement learning using learned policies. In *AAAI*, 2003.

[14] A. McGovern and A. G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *ICML*, 2001.

[15] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

[16] J. Andreas, D. Klein, and S. Levine. Modular multitask reinforcement learning with policy sketches. In *ICML*, 2017.

[17] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein. Learning to Compose Neural Networks for Question Answering. In *NAACL HLT*, 2016.

[18] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein. Neural Module Networks. In *CVPR*, 2016.

[19] T. Winograd. Understanding natural language. *Cognitive Psychology*, 1972.

[20] H. Yu, H. Zhang, and W. Xu. Interactive Grounded Language Acquisition and Generalization in a 2D World. In *ICLR*, 2018.

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

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

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

[24] F. Hill, K. M. Hermann, P. Blunsom, and S. Clark. Understanding grounded language learning agents. *arXiv preprint arXiv:1710.09867*, 2017.

[25] J. Oh, S. Singh, H. Lee, and P. Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *ICML*, 2017.

[26] T. Shu, C. Xiong, and R. Socher. Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. In *ICLR*, 2018.

[27] A. Vogel and D. Jurafsky. Learning to follow navigational directions. In *ACL*, 2010.

[28] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox. Learning to parse natural language commands to a robot control system. In *ISER*, 2013.

[29] K. Narasimhan, T. Kulkarni, and R. Barzilay. Language understanding for text-based games using deep reinforcement learning. In *EMNLP*, 2015.

[30] D. Gordon, A. Kembhavi, M. Rastegari, J. Redmon, D. Fox, and A. Farhadi. IQA: Visual Question Answering in Interactive Environments. In *CVPR*, 2018.

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

[32] 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.

[33] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *ICRA*, 2017.

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

[35] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. In *CVPR*, 2017.

[36] S. Gupta, D. Fouhey, S. Levine, and J. Malik. Unifying map and landmark based representations for visual navigation. *arXiv preprint arXiv:1712.08125*, 2017.

[37] E. Kolve, R. Mottaghi, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017.

[38] A. Y. Ng, D. Harada, and S. J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, 1999.

[39] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. In *ICLR*, 2016.