FEATURE-ATTENDING RECURRENT MODULES FOR GENERALIZING OBJECT-CENTRIC BEHAVIOR

Wilka Carvalho∗,1 Andrew Lampinen2
Kyriacos Nikiforou2 Felix Hill2 Murray Shanahan2
1University of Michigan  2DeepMind

ABSTRACT

To generalize in object-centric tasks, a reinforcement learning (RL) agent needs to exploit the structure that objects induce. Prior work has either hard-coded object-centric features, used complex object-centric generative models, or updated state using local spatial features. However, these approaches have had limited success in enabling general RL agents. Motivated by this, we introduce “Feature-Attending Recurrent Modules” (FARM), an architecture for learning state representations that relies on simple, broadly applicable inductive biases for capturing spatial and temporal regularities. FARM learns a state representation that is distributed across multiple modules that each attend to spatiotemporal features with an expressive feature attention mechanism. This enables FARM to represent diverse object-induced spatial and temporal regularities across subsets of modules. We hypothesize that this enables an RL agent to flexibly recombine its experiences for generalization. We study task suites in both 2D and 3D environments and find that FARM better generalizes compared to competing architectures that leverage attention or multiple modules.

1 INTRODUCTION

Objects are key to real-world tasks. For example, a self-driving car needs to represent the movements of other cars, and a household robot needs to recognize and use kitchen items. In order to generalize across tasks with objects, a reinforcement learning (RL) agent should capture and exploit object-induced structure present across the tasks.

One way to capture this structure is in an agent’s state representation. Unfortunately, flexibly capturing objects in a state representation is challenging because objects manifest in a multitude of ways. Consider a household robot tasked with cooking. Completing the task might require memory about objects that range in size, shape, and color (e.g. a stove vs. a tomato). Additionally, other agents may be in motion requiring that the agent represent temporal information about objects in order to coordinate navigation. It is unclear how to best incorporate objects into a state representations to enable generalization.

Prior work has attempted to capture object-induced task structure by hand-designing object-centric state features [Diuk et al. 2008; Carvalho et al. 2021; Borsa et al. 2018; Marom & Rosman 2018]. The “COBRA” agent [Watters et al. 2019] avoids hand-designing features by learning an object-centric generative model. However, these methods are limited in their generality because they rely on relatively strong inductive biases. For example, COBRA relies on environments being fully-observable and objects having regular shapes to learn representations by predicting object segmentations. We focus on weak inductive biases in order to maximize an architecture’s flexibility.

Objects can be described by subsets of features over space and time. We conjecture that weak inductive biases for capturing subsets of features over space and time may enable agents that can flexibly incorporate objects into state across a wide range of environments.

We propose Feature Attending Recurrent Modules (FARM), a simple but flexible architecture for learning state representations when tasks share object-induced structural regularities. FARM learns

∗Contact author: wcarvalh@umich.edu  Work done during internship.
state representations that are distributed across multiple, smaller recurrent modules. To help motivate this, consider word embeddings. A word embedding can represent more information than a one-hot encoding of the same dimension because subsets of dimensions can coordinate activity to represent different patterns of word usage. Analogously, learning multiple modules enables FARM to coordinate subsets of modules to represent different temporal segments in an agent’s experiences. To capture general object-induced patterns, modules select observation information to update with by weighting observation features.

We study FARM across three diverse object-centric environments, each with their own suite of tasks that share object-induced structural regularities. Tasks in the Ballet environment share regularities induced by object motions; tasks in the Place environment share regularities induced by navigating towards and around 3D objects; and tasks in the KeyBox environment share regularities induced by object configurations. These environments present a number of challenges. First, their state-space grows exponentially with the number of objects. The more distractor objects an environment has, the larger the chance an object will obstruct an agent’s path. This requires learning a policy that can navigate around distractor-based obstacles. When task objects appear in sequence, this can require long-horizon memory of object information (e.g., of goal information). Finally, tasks defined by language can require an agent learn a complex mapping (e.g., to object motions and to irregular shapes in our tasks). Across these environments, we study an agent’s ability to recombine object-oriented memory, obstacle-avoidance, and navigation to longer tasks with more objects.

We compare against methods with weak inductive biases for enabling objects to emerge in a state representation. Recent work has shown that spatial attention is a simple inductive bias for strong performance on object-centric vision tasks because it enables attending to individual objects (Greff et al., 2020; Locatello et al., 2020; Goyal et al., 2020a). Thus, we compare against recent RL agents that leverage spatial attention for object-centric state-update functions (Goyal et al., 2020b; Mott et al., 2019).
Summary of contributions.

1. We introduce FARM, a simple architecture for learning an agent’s state representation when an environment has object-induced structural regularities (§4).
2. We show that FARM’s simple inductive biases—feature attention and multiple modules—synergistically enable generalizing (a) memory to longer combinations of object motions (§5.1); (b) navigation to 3D objects in larger environments (§5.2); and (c) memory of goal information to longer tasks with more distractors (§5.3).
3. We compare against spatial attention, which has been shown to enable object-centric state updates. We find mixed benefits in for our diverse object-centric tasks, including interference with learning multiple modules.
4. We analyze FARM’s state representations and provide evidence that object-induced regularities are represented in patterns that are flexibly distributed across subsets of modules (§5.3.1).

2 RELATED WORK ON GENERALIZATION IN DEEP RL

The key question for generalization is how to capture structure in the problem in a flexible way. How much structure do you build in? How much do you let the agent discover? Some work takes a data-driven approach (Tobin et al., 2017; Packer et al., 2018; Hill et al., 2020; Justesen et al., 2019). Others have a policy that captures task structure with either hierarchical RL (Oh et al., 2017; Zhang et al., 2018; Sohn et al., 2018; 2021; Brooks et al., 2021) or successor features (Borsa et al., 2018; Barreto et al., 2020). A final strand focuses on learning invariant representations (Higgins et al., 2017; Chaplot et al., 2018; Lee et al., 2020; Zhang et al., 2021) or building in inductive biases (Mott et al., 2019; Goyal et al., 2020b). In this work we focus on weak inductive biases for capturing structure. Below we review approaches most closely related to ours.

Generalizing across object-centric tasks dates back at least to object-oriented MDPs (Dzeroski et al., 2001; Duk et al., 2008), which enabled generalization by representing dynamics with logical object attributes (Kansky et al., 2017; Marom & Rosman, 2018). Successor features have also leveraged objects for generalization by formulating rewards as linear with object-centric features (Borsa et al., 2018; Barreto et al., 2020). A common thread among these directions is that they relied on hand-designed object features. Watters et al. (2019) avoided hand-designing features by learning an object-centric generative model (Burgess et al., 2019). However, they focused on fully-observable top-down environments with regular shapes, which allowed them to predict future object masks. This is incompatible with our environments. While research on object-centric models (Kabra et al., 2021; Zoran et al., 2021) has progressed, these methods commonly add training complexity (more objective terms, extra modules, etc.) and make stronger assumptions (e.g. on the number of objects). We differ from this work because we focus on simple, broadly applicable inductive biases for capturing object-induced task regularities.

Generalizing with feature attention has also been studied by Chaplot et al. (2018). They showed that mapping language instructions to non-linear feature coefficients enabled generalizing to tasks specified over unseen feature combinations in a 3D environment. While FARM also learns non-linear feature coefficients, our work has two important differences. First, we develop a multi-head version where individual feature coefficients are produced by their own recurrent modules. This enables FARM to leverage this form of attention in settings where language instructions don’t indicate what to attend to (this is true in 2/3 of our tasks). Second, we are the first to show that feature attention enables generalizing memory of object motions and of goal information to longer tasks (Figure 4 and Figure 6, respectively).

Generalizing with top-down spatial attention. Most similar to FARM are the Attention Augmented Agent (AAA) (Mott et al., 2019) and Recurrent Independent Mechanisms (RIMs) (Goyal et al., 2020b). Both are recurrent architectures that leverage spatial attention to learn an object-centric state-update function. Both showed generalization to novel distractors. The major difference between AAA, RIMs, and FARM is that FARM attends to an observation with feature attention as opposed to spatial attention. Our experiments indicate that spatial attention has limited utility in updating state during reinforcement learning of tasks defined by object motions (Figure 4) or 3D objects (Figure 5). In terms of modularity, we also show different results from RIMs who showed that their modules “specialize”. Our experiments suggest that in FARM, a modular state instead leads subset of modules to jointly represent regularities in an agent’s experience (§5.3.1).
3 Problem Setting

We study generalization across tasks within deterministic, partially-observable, pixel-based environments. Within an environment, a task is defined by a Partially Observable Markov decision process (POMDP): \( M = \langle S, A, O, R, T, \psi, \rangle \). \( S \) corresponds to environment states, \( A \) corresponds to actions that agent can take, \( O \) corresponds to the agent’s observations, \( r = R(s, a) \in \mathbb{R} \) is the reward function, \( s' = T(s, a) \in S \) is the environment transition function, and \( o = \psi(s) \in O \) is an observation function that maps the underlying environment state to an RGB image.

We seek an RL agent that learns to perform tasks by finding a policy \( \pi \) that maximizes the expected discounted sum of rewards it obtains starting at a state \( s \):

\[
V(s) = \mathbb{E}\left[ \sum_{t=0}^{\infty} \gamma^t R(S_t, A_t) \right]
\]

—also known as the value of a state. In a POMDP, the agent doesn’t have access to the environment state.

Object-induced structural regularities. We study object-centric environments, where objects induce structural regularities across tasks in the reward functions \( R \), transition functions \( T \), and observation functions \( \psi \). For example, consider the KeyBox environment in Figure 1 (c). First, \( R(s, a) \) always specifies the goal key based on a goal box. Second, whenever the agent has to navigate around an obstacle (see Figure 2, b), the agent always sees the sprite it controls move closer to an object and then around it. This is true regardless of where in the hallway the agent observes the obstacle because of regularities in the transition function \( T(s, a) \) and observation function \( \psi(s) \).

We want an agent that captures these regularities in its representation for state to enable zero-shot generalization to new tasks.

4 Architecture: FARM

We propose a new architecture, “Feature Attending Recurrent Modules” (FARM) for learning an agent’s state representations when an environment has object-induced structural regularities. We provide an overview of the architecture in Figure 2. Instead of representing agent state with a single recurrent function, FARM learns a state representation that is distributed across \( n \) recurrent functions \( \{\eta_k\}_{k=1}^{n} \), which we call modules (Figure 2, a). Distributing state across modules allows subsets of modules to jointly represent different regularities in the agent’s experience (Figure 2, b). We hypothesize that having subsets of modules represent different regularities in the agent’s experience enables the agent to flexibly recombine its experience for generalization.

At each time-step \( t \), each module updates with both observation features and information from other modules. First, the agent computes observation features with a recurrent observation encoder,

\[
Z_t = \phi(o_t, Z_{t-1}).
\]

Afterward, each module creates a query vector by combining its previous module-
We now describe each computation in more detail. In order to update with features that describe both visual and temporal regularities, the agent learns structured spatiotemporal features \( Z_t \in \mathbb{R}^{m \times d_z} \) that share \( d_z \) spatio-temporal features across \( m \) spatial positions. Here we show toy computations where static observations features (blue) on the top and bottom row move to spatial positions to the right. The resultant spatio-temporal features also include temporal information about the features (here, that the features came from leftward spatial positions). \( f^{\text{att}}_k \) computes coefficients for features and applies them uniformly across all spatial positions. This allows the agent to attend to all spatial position that possess desired features.

\[
Z_t = \phi(o_t, Z_{t-1}) \quad \text{obs features} \quad (1) \\
q^{k-1}_t = [h^{k-1}_t, a_{t-1}, r_{t-1}] \quad \text{query} \quad (2) \\
u^k_t = f^{\text{att}}_k(Z_t, q^{k-1}_t) \quad \text{obs attention} \quad (3) \\
u^k_t = f^{\text{share}}(s^k_{t-1}, q^{k-1}_t) \quad \text{share info} \quad (4) \\
h^k_t = \eta^k(u^k_t, v^k_t, q^{k-1}_t) \quad \text{module update} \quad (5) \\
s^k_t = [h^k_1, \ldots, h^k_m] \quad \text{agent state} \quad (6)
\]

We now describe each computation in more detail.

### Structured spatiotemporal observation features.
Our first insight is that modules can attend to features describing an object’s motion if an agent learns observation features that describe both spatial and temporal regularities. An agent can accomplish this by learning structured spatiotemporal features with a recurrent observation encoder \( Z_t = \phi(x_t, Z_{t-1}) \in \mathbb{R}^{m \times d_z} \) that share \( d_z \) features across \( m \) spatial positions. At each spatial position, these features both describe what is there visually along with temporal information about the recent dynamics of these features. We show example toy computations in Figure 3(b).

### Dynamic feature attention.
Our second insight is that feature attention is an expressive attention function that can focus on desired information present across all spatial positions in observation features. An agent accomplishes this by having a module predict feature coefficients that it applies to uniformly across all spatial positions in \( Z_t \) [Perez et al., 2018; Chaplot et al., 2018]. We show example toy computations in Figure 3(c). We found it useful to linearly project the features before and after using shared parameters as in [Andreas et al., 2016; Hu et al., 2018]. The operations are summarized below:

\[
f^{\text{att}}_k(Z_t, q^{k-1}_t) = (Z_t W_1 \odot \sigma(W^{\text{att}}_k q^{k-1}_t)) W_2
\]

\( W_1, W_2, W^{\text{att}}_k \) are learnable parameters.

---

\( f^{\text{att}}_k \), \( f^{\text{share}} \), and \( \eta^k \) are learnable functions.\(^1\)

---

\( \phi \) and \( \sigma \) are activation functions.

---

\( Z_t \) are features.

---

\( o_t, a_{t-1}, r_{t-1} \) are observations, actions, and rewards.

---

\( h^k_t \) are hidden states.

---

\( m \) is the number of spatial positions.

---

\( d_z \) is the dimension of spatiotemporal features.

---

\( W \) are learnable parameters.

---

\( \odot \) is the Hadamard product.

---

\( \sigma \) is the logistic sigmoid function.

---

\( \phi \) is a function that maps observations to features.

---

\( \sigma \) is a function that maps features to hidden states.

---

\( \phi \) and \( \sigma \) are activation functions.

---

\( Z_t \) are features.

---

\( o_t, a_{t-1}, r_{t-1} \) are observations, actions, and rewards.

---

\( h^k_t \) are hidden states.

---

\( m \) is the number of spatial positions.

---

\( d_z \) is the dimension of spatiotemporal features.

---

\( W \) are learnable parameters.

---

\( \odot \) is the Hadamard product.

---

\( \sigma \) is the logistic sigmoid function.

---

\( \phi \) is a function that maps observations to features.

---

\( \sigma \) is a function that maps features to hidden states.

---

\( Z_t \) are features.

---

\( o_t, a_{t-1}, r_{t-1} \) are observations, actions, and rewards.

---

\( h^k_t \) are hidden states.

---

\( m \) is the number of spatial positions.

---

\( d_z \) is the dimension of spatiotemporal features.

---

\( W \) are learnable parameters.

---

\( \odot \) is the Hadamard product.

---

\( \sigma \) is the logistic sigmoid function.

---

\( \phi \) is a function that maps observations to features.

---

\( \sigma \) is a function that maps features to hidden states.

---

\( Z_t \) are features.
where $\odot$ denotes an element-wise product over the feature dimension and $\sigma$ is a sigmoid non-linearity. Since our features capture dynamics information, this allows a module to attend to object motion (§5.1). When updating, we flatten the attention output. Flattening leads all spatial positions to be treated uniquely and allows a module to represent aspects of the observation that span multiple positions, such as 3D objects (§5.2) and spatial arrangements of objects (§5.3). Since the feature-coefficients for the next time-step are produced with observation features from the current time-step, modules can dynamically shift their attention when task-relevant events occur (see Figure 7, b for an example).

Sharing information. Similar to RIMs (Goyal et al., 2020b), before updating, each module retrieves information from other modules using transformer-style attention (Vaswani et al., 2017). We define the collection of previous module-states as $H_{t-1} = [h^{(1)}_{t-1}; \ldots; h^{(n)}_{t-1}; 0] \in \mathbb{R}^{(n+1) \times d_h}$, where 0 is a null-vector used to retrieve no information. A module computes a “retrieval query” to search for information as $q^k = q^{k}_{t-1} W_{\text{que}} \in \mathbb{R}^{d_h}$. That module computes “retrieval keys and values” as $K^k = H_{t-1} W^{\text{key}} \in \mathbb{R}^{n+1 \times d_h}$ and $V^k = H_{t-1} W^{\text{val}} \in \mathbb{R}^{n+1 \times d_h}$, respectively. Each module then retrieves information as follows:

$$f^{k}_{\text{share}}(s^A_{t-1}, q^{k}_{t-1}) = \text{softmax} \left( \frac{q^{k}_{t-1} K^k \cdot \sigma}{\sqrt{d_h}} \right) V^k. \quad (8)$$

Intuitively, the dot-product inside the softmax is computing $n + 1$ scores (one for each “key”), which then form probabilities. The outer dot-product multiplies each “value” by its probability and sums them to perform soft-selection.

## 5 Experiments

In this section, we study the following questions:

1. Can FARM generalize memory to longer spatiotemporal combinations of object motions?
2. Can FARM generalize navigation towards and avoidance of 3D objects to larger environments?
3. Can FARM generalize memory of goal-information to larger maps with more distractor-based obstacles?

### Baselines.

Our first baseline is a common choice for learning state-representations, a Long Short-term Memory (LSTM) (Hochreiter & Schmidhuber, 1997). We study two other baselines that also attend to observation features: Attention Augmented Agent (AAA) (Mott et al., 2019) and Recurrent Independent Mechanisms (RIMs) (Goyal et al., 2020b). Both employ transformer-style attention (Locatello et al., 2020; Vaswani et al., 2017) to attend to individual spatial positions by reducing observation features to a weighted average over spatial positions. We instead attend to features shared across all spatial positions. RIMs, like FARM, represents state with a set of recurrent modules.

### Implementation details.

We implement our recurrent observation encoder, $\phi$, as a ResNet (He et al., 2016) followed by a Convolutional LSTM (ConvLSTM) (Shi et al., 2015). We implement the update function of each module with an LSTM. We used multihead-attention (Vaswani et al., 2017) for $f^{k}_{\text{share}}$. We trained the architecture with the IMPALA algorithm (Espeholt et al., 2018) and an Adam optimizer (Kingma & Ba, 2015). We tune hyperparameters for all architectures with the “Place X next to Y” task from the BabyAI environment (Chevalier-Boisvert et al., 2019) (§B.2). For details on hyperparameters, see §A.

### Table 1: Baselines.

| Method | Observation | Attention | Modular State |
|--------|-------------|-----------|---------------|
| LSTM   | X           | X         |               |
| AAA    | Spatial     | X         |               |
| RIMs   | Spatial     | ✓         |               |
| FARM (Ours) | Feature | ✓         |               |

5.1 Generalizing Memory to More Object Motions

We study this with the “Ballet” grid-world (Lampinen et al., 2021) shown in Figure 1 (a). Tasks.

The agent controls a white square that begins in the middle of the grid. There are $m$ other “ballet-dancer” objects that move with a one of 15 distinct object motions. The dances move in sequence for 16 time-steps with a 48-time-step delay in between. After all dancers finish, the agent is given a
Figure 4: **FARM enables generalizing memory to longer spatiotemporal combinations of object motions.** We present the success rate means and standard errors computed using 5 seeds. (a) Only FARM is able to go above chance performance for each setting. (b) Given spatiotemporal features, we find that *either* using multiple modules *or* feature attention enables learning memory of object motions. These results suggest that spatial attention removes the benefits of using multiple modules for learning to remember object motions. Encouragingly, feature attention by itself can support it.

Figure 5: **FARM enables generalizing navigation towards and avoidance of 3D objects to a larger environment.** We present the success rate means and standard errors computed using 3 seeds. (a) FARM generalizes best. (b) Our performance benefits mainly come from learning multiple modules, though feature attention slightly improves performance and lowers variance. These results suggest that spatial attention interferes with reinforcement learning of 3D objects.

language instruction indicating the correct ballet dancer to navigate towards. All shapes and colors are randomized making motion the only feature indicating the goal object. **Observations.** The agent observes a top-down RGB image of the environment. **Actions.** The agent can move left, right, up, and down. **Reward** is 1 if it touches the correct dancer and 0 otherwise. **Tasks split.** Training tasks always consists of seeing \(m = \{2, 4\}\) dancers; testing tasks always consists of seeing \(m = \{8\}\) dancers. All agents learn with a sample budget of 2 billion frames. A poorly performing agent will obtain chance performance, \(1/m\).

We present the training and generalization success rates in Figure 4. We learned spatiotemporal observation features with RIMs and AAA for a fair comparison. We found that only FARM is able to obtain above chance performance for training and testing. In order to understand the source of our performance, we ablate using a recurrent observation encoder, using multiple modules, and using feature-attention. We confirm that a recurrent encoder is required. Interestingly, we find that either using multiple modules or using our feature-attention enables task-learning, with our feature-attention mechanism being slightly more stable.

5.2 GENERALIZING NAVIGATION WITH MORE 3D OBJECTS

Here, we study the 3D Unity environment from [Hill et al., 2020](#) shown in Figure 1(b). **Tasks.** The agent is an embodied avatar in a room filled with task objects and distractor objects. The agent receives a language instruction of the form “X on Y” — e.g., “toothbrush on bed”. We partition objects into two sets as follows: pickup-able objects \(O_1 = A \cup B\) and objects to place them on \(O_2 = C \cup D\).

**Observations.** The agent receives first-person egocentric RGB images. **Actions.** The agent has 46 actions that allow it to navigate, pickup and place objects. **Reward** is 1 if it completes the task and 0 otherwise. **Tasks split.** During training the agent sees \(A \times D\) and \(B \times C\) in a \(4m \times 4m\) room with 4
Figure 6: FARM enables generalizing memory of goal-information and avoidance of obstacles to larger maps with more objects. We show the the success rate mean and standard error computed using 10 seeds. (a) In the densely populated setting, FARM better generalizes to longer hallways with more distractors. (b) In the sparsely populated setting, FARM has slightly better performance than AAA for $2n_{\text{max}}$ but comparable performance for $3n_{\text{max}}$. (c) Using multiple modules and feature attention both improve generalization. These results suggest that spatial attention interferes with generalization benefits of learning multiple modules. Learning feature attention and multiple modules, instead, act synergistically.

distractors, along with $A \times C$ and $B \times D$ in a $3m \times 3m$ room with 0 distractors. We test the agent on $A \times C$ and $B \times D$ in a $4m \times 4m$ room with 4 distractors. We also train with “Go to X” and “Lift X”.

We present the generalization success rate in Figure 5. We find that baselines which used spatial attention learn more slowly than an LSTM or FARM. Additionally, both models that use spatial attention have poor performance until the end of training where AAA begins to improve. FARM achieves relatively good performance, achieving a success rate of 60% and 80% on the two test settings, respectively.

5.3 Generalizing to Larger Maps with More Objects

To study this, we create the “keyBox” environment depicted in Figure 1 (c). Tasks are defined with $n$ levels. Each level is a hallway with a single box and a key of the same color that the agent must retrieve. The agent and the box always starts in the left-most end and the goal key always starts in the right-most end. The agent always begins in the first level. It is teleported to the next level after placing the goal key next to the goal box. The hallway for level $n$ consists of a length-$n$ sequence of $w \times w$ environment subsections. Each subsection contains $d$ distractor objects. Observations. The agent observes egocentric top-down images over a short segments of the hallway. Actions. The agent can move forward, turn left, turn right, pick up objects and drop them. Rewards. When completing a level, the agent gets a reward of $n/n_{\text{max}}$ where $n_{\text{max}}$ is the maximum level. Tasks split. Learning tasks include levels 1 to $n_{\text{max}} = 10$. Test tasks only use levels $2n_{\text{max}}$ and $3n_{\text{max}}$. We study two generalization settings: a densely populated setting with subsections of area $w^2 = 9$ and $d = 2$ distractors, and a sparsely populated setting with subsections of area $w^2 = 25$ and $d = 4$ distractors.

We present the generalization success rates in Figure 6. In the dense setting, we see an LSTM quickly overfits in both settings. All architectures with attention continue to improve in generalization performance as they continue training. In the dense setting, we find that FARM generalizes better (by about 20% for AAA and about 30% for RIMs). In the sparse setting, both RIMs and an LSTM fail to generalize above 30%. FARM generalizes better than AAA for level 20 but gets comparable performance for level 30. In some ways, this is our most surprising result since it is not obvious that either learning multiple modules or using feature attention should help with this task. In the next section we study possible sources of our generalization performance.

5.3.1 Analysis of State Representations

We study the state representations FARM learns for categories of regularly occurring events. We collect 2000 generalization episodes in level 20. We segment these episodes into 6 categories: pickup ball, drop ball, pickup wrong key, drop wrong key, pickup correct key, and drop correct key. We study the time-series of the L2 norm of each module-state and their attention coefficients. For reference, we also show the L2 norm for the entire episode. We note that we observed consistent activity that
We show evidence that different subsets of modules jointly represent object-induced task regularities. (a) Module 0 commonly exhibits salient activity when the agent moves around an obstacle. (b) Module 6 activates its attention coefficients as the agent picks up the goal key. (c) Modules 2 and 6 correlate for picking up the correct key but anti-correlate for dropping the wrong object. This is similar to when neurons in word embeddings correlate for some words (e.g. man and king), but anti-correlate for other words (e.g. man and woman). In general, we find rich patterns of correlation and anti-correlation between the modules. These results suggest that FARM is representing task regularities across the modules in complicated and interesting ways. Videos of the state-activity and attention coefficients: [https://bit.ly/3qCxatr](https://bit.ly/3qCxatr).

was not captured by our simple programmatic classification of states; for example, salient activity from module 0 when the agent moved around obstacles. We show an example in Figure 7a.

Due to space constraints, we present a subset of results in Figure 7. For all results, please see the §C. While some modules are selective for different recurring events such as attending to goal information (Figure 7b), it seems that subsets of modules jointly represent different aspects of state. We hypothesize that this enables FARM to leverage overlapping sets of modules to store goal-information or to navigate around obstacles in a decoupled way that supports recombination. This is further supported by our ablation where we find that having 4 or 8 modules significantly outperforms using a single large module (all had about 8M params) (Figure 6c). Feature attention consistently improves performance.

6 DISCUSSION AND CONCLUSION

We have presented FARM, a novel state representation learning architecture for environments that have object-induced structural regularities. Our results show that learning feature attention and multiple modules acts synergistically to support generalizing (a) memory to longer combinations of object-motions (§5.1); (b) navigation around 3D objects to larger environments (§5.2); and (c) memory of goal information to longer sequences of obstacles (§5.3). Our ablations suggest that feature attention mainly helps with long-horizon memory. Interestingly, learning multiple modules helped across all conditions (memory, obstacle-avoidance, and language learning). Our analysis suggests that learning multiple modules enables subsets to represent object-centric task-relevant events in flexible ways. We hypothesize that this enables a deep RL agent to flexibly recombine its experience for generalization.

We compared FARM to other architectures that used spatial attention as a weak inductive bias for enabling objects to emerge in a state representation. We found that spatial attention hindered learning tasks with object motions and 3D objects. In the KeyBox task, spatial attention seemed to help AAA most in the sparse setting with many objects. This makes sense since spatial attention has been
shown to help with distractors and the agent mainly needed to ignore objects and move forward. Interestingly, pairing spatial attention with multiple modules (RIMs) removed the benefits of both.

One limitation of FARM is that feature attention is not spatially invariant since it treat all positions as unique. Future work can look to adapt this attention for something that still describes multiple positions but in a spatially invariant way. Another limitation of FARM is the length of temporal regularities it can capture. Transformers (Vaswani et al., 2017) have shown strong performance for representing long sequences. An interesting next-step might be to integrate FARM with a transformer. We hope that our work contributes to future RL algorithms that leverage weak inductive biases for capturing object-centric task regularities.

REFERENCES

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 39–48, 2016.

André Barreto, Shaobo Hou, Diana Borsa, David Silver, and Doina Precup. Fast reinforcement learning with generalized policy updates. Proceedings of the National Academy of Sciences, 117(48):30079–30087, 2020.

Diana Borsa, André Barreto, John Quan, Daniel Mankowitz, Rémi Munos, Hado van Hasselt, David Silver, and Tom Schaul. Universal successor features approximators. arXiv preprint arXiv:1812.07626, 2018.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs. 2018. URL http://github.com/google/jax.

Ethan A Brooks, Janarthanan Rajendran, Richard L Lewis, and Satinder Singh. Reinforcement learning of implicit and explicit control flow in instructions. ICML, 2021.

Christopher P Burgess, Loic Matthey, Nicholas Watters, Rishabh Kabra, Irina Higgins, Matt Botvinick, and Alexander Lerchner. Monet: Unsupervised scene decomposition and representation. arXiv preprint arXiv:1901.11390, 2019.

Wilka Carvalho, Anthony Liang, Kimin Lee, Sungryull Sohn, Honglak Lee, Richard L Lewis, and Satinder Singh. Reinforcement learning for sparse-reward object-interaction tasks in first-person simulated 3d environments. IJCAI, 2021.

Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. Gated-attention architectures for task-oriented language grounding. In Proceedings of the AAAI Conference on Artificial Intelligence, 2018.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: First steps towards grounded language learning with a human in the loop. In International Conference on Learning Representations, 2019. URL https://openreview.net/forum?id=rJeXCo0cYX.

Carlos Diuk, Andre Cohen, and Michael L Littman. An object-oriented representation for efficient reinforcement learning. In Proceedings of the 25th ICML, pp. 240–247, 2008.

Sašo Džeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. Machine learning, 43(1):7–52, 2001.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In ICML, pp. 1407–1416. PMLR, 2018.

Anirudh Goyal, Alex Lamb, Phanideep Gampa, Philippe Beaudoin, Sergey Levine, Charles Blundell, Yoshua Bengio, and Michael Mozer. Object files and schemata: Factorizing declarative and procedural knowledge in dynamical systems. arXiv, 2020a.
Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. *ICLR*, 2020b.

Klaus Greff, Sjoerd van Steenkiste, and Jürgen Schmidhuber. On the binding problem in artificial neural networks. *arXiv*, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pp. 770–778, 2016.

Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. Darla: Improving zero-shot transfer in reinforcement learning. In *ICML*, pp. 1480–1490. PMLR, 2017.

Felix Hill, Andrew Lampinen, Rosalia Schneider, Stephen Clark, Matthew Botvinick, James L McClelland, and Adam Santoro. Environmental drivers of systematicity and generalization in a situated agent. *ICLR*, 2020.

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

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. Illuminating generalization in deep reinforcement learning through procedural level generation. *AAAI*, 2019.

Rishabh Kabra, Daniel Zoran, Goker Erdogan, Loic Matthey, Antonia Creswell, Matthew Botvinick, Alexander Lerchner, and Christopher P Burgess. Simone: View-invariant, temporally-abstracted object representations via unsupervised video decomposition. *arXiv preprint arXiv:2106.03849*, 2021.

Ken Kansky, Tom Silver, David A Mély, Mohamed Eldawy, Miguel Lázaro-Gredilla, Xinghua Lou, Nimrod Dorfman, Szymon Sidor, Scott Phoenix, and Dileep George. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. In *ICML*, pp. 1809–1818. PMLR, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

Andrew Kyle Lampinen, Stephanie CY Chan, Andrea Banino, and Felix Hill. Towards mental time travel: a hierarchical memory for reinforcement learning agents. *arXiv*, 2021.

Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. In *ICLR*, 2020.

Francesco Locatello, Dirk Weissendorn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. *arXiv preprint arXiv:2006.15055*, 2020.

Ofir Marom and Benjamin Rosman. Zero-shot transfer with deictic object-oriented representation in reinforcement learning. In *NeurIPS*, 2018.

Alex Mott, Daniel Zoran, Mike Chrzanowski, Daan Wierstra, and Danilo J Rezende. Towards interpretable reinforcement learning using attention augmented agents. *NeurIPS*, 2019.

Junhyuk Oh, Satinder Singh, Honglak Lee, and P. Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. *ICML*, abs/1706.05064, 2017.

Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. *arXiv*, 2018.
Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. *Advances in neural information processing systems*, 28, 2015.

Sungryull Sohn, Junhyuk Oh, and Honglak Lee. Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies. *NeurIPS*, 2018.

Sungryull Sohn, Hyunjae Woo, Jongwook Choi, and Honglak Lee. Meta reinforcement learning with autonomous inference of subtask dependencies. *ICLR*, 2021.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS*, pp. 23–30. IEEE, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv*, 2017.

Nicholas Watters, Loic Matthey, Matko Bosnjak, Christopher P Burgess, and Alexander Lerchner. Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration. *arXiv preprint arXiv:1905.09275*, 2019.

Amy Zhang, Sainbayar Sukhbaatar, Adam Lerer, Arthur Szlam, and Rob Fergus. Composable planning with attributes. In *ICML*, pp. 5842–5851. PMLR, 2018.

Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. *ICLR*, 2021.

Daniel Zoran, Rishabh Kabra, Alexander Lerchner, and Danilo J Rezende. Parts: Unsupervised segmentation with slots, attention and independence maximization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10439–10447, 2021.
A Hyperparameters

All neural networks were built using the Jax library (Bradbury et al., 2018). In all experiments, training was carried out using a distributed A3C setup (Espeholt et al., 2018) with discrete actions. For 3D Unity Env experiments, we added an additional Pixel Control loss (Jaderberg et al., 2016) for all agents. We used a single learner and 256 actors. Important training hyper-parameters are shown in Table 2 along with the components of the agent’s architecture that are shared between the different models. The parameter values used for each model presented in the main paper are shown below in Table 3.

A.1 Hyperparameter search

We tuned all models using the BabyAI “Place X next to Y” task (Chevalier-Boisvert et al., 2019) (§B.2). For each architectures, we tuned using a random search. Additionally, we increased the size of the LSTM so that all architectures had approximately the same number of parameters.

Feature Attending Recurrent Modules. We found that we did not need to tune the model much beyond a random search over attention projection dims $W \in [16, 32]$ and Conv LSTM kernel size $[3, 5]$.

Attention Augmented agent. We used hyper-parameters from their paper put tuned the following: LSTM hidden size $[256, 512]$, Attention query MLP size $[\{\}, \{256\}, \{256, 256\}]$, number of attention heads $[4, 8]$. We consulted the authors about our implementation.

Recurrent Independent Mechanisms. We used hyper-parameters from their paper put tuned the following: LSTM hidden size $[100, 128, 256]$, Observation/communication head size $[32, 64, 128]$, number of observation/communication heads $[4, 5, 6]$, number of RIMs $[4, 6, 9, 12]$. We consulted the authors about our implementation and used their source code for replication https://github.com/anirudh9119/RIMs We note that we did not use their “top-k” feature where only $k$ RIMs are active because that doubled run-time. Since our experiments typically required $\geq 1$ billion frames, using “top-k” led individual runs to take $\geq 2$ days.

Long Short-Term Memory. We searched an LSTM hidden size $[128, 256, 512]$. We consistently found that a larger memory had better results.
Table 2: Training hyper-parameters and shared network components used in experiments.

| Loss Hyper-parameters          | 3D Unity Env | Gridworlds |
|--------------------------------|--------------|------------|
| V-trace baseline cost          | 1.0          | 0.5        |
| V-trace entropy cost           | $10^{-4}$    | 0.01       |
| V-trace $\gamma$              | 0.95         | 1.0        |
| V-trace loss scaling           | 0.1          | 1.0        |
| Pixel Control loss scaling     | 0.1          | –          |
| Pixel Control loss cell size   | 4            | –          |
| Pixel Control discount factor  | 0.9          | –          |
| Pixel Control de-convolution sizes | (6, 9, 32), (8, 11, 32) | – |
| Pixel Control kernel shape     | (3, 4)       | –          |
| Pixel Control de-convolution output shape | (18, 42) | – |
| Optimizer                      | clipped Adam | clipped Adam |
| Learning rate                  | $2 \times 10^{-4}$ | $10^{-4}$ |
| Max gradient Norm              | 40.0         | 40.0       |
| Optimizer epsilon              | $5 \times 10^{-8}$ | $10^{-8}$ |
| Adam $\beta_1$                | 0.0          | 0.9        |
| Adam $\beta_2$                | 0.95         | 0.999      |

| Shared Network Components      |              |            |
|--------------------------------|--------------|------------|
| Language encoder               | GRU          | GUR        |
| Language encoder hidden sizes  | 128          | 128        |
| Language word embedding size   | 128          | 128        |
| Image encoder                  | Res-Net      | Res-Net    |
| Res-Net channels               | (16, 32, 32) | (16, 32, 32) |
| Res-Net residual blocks        | (2, 2, 2)    | (2, 2, 2)  |
| Res-Net stride                 | 2            | 2          |
| Res-Net kernel size            | 3            | 3          |
| Res-Net padding                | SAME         | SAME       |
| Previous action encoding       | Identity     | Identity   |
| Reward encoding                | Identity     | Identity   |
| Image-language-reward-action combination | Concatenation | Concatenation |
| Input Convolutional dims       | (9, 12, 32)  | (7, 7, 32) |
| Policy Head MLP shapes         | [512, 46]    | [200, 7]   |
| Value Head MLP shapes          | [512, 1]     | [200, 1]   |
Table 3: Model specific parameters.

| Model parameter                        | 3D Unity Env | Gridworld Ballet | Gridworld Keybox |
|----------------------------------------|--------------|------------------|------------------|
| **Observation Dims**                   |              |                  |                  |
| Observation Dims                       | 72 × 96      | 99 × 99          | 56 × 56          |
| **Feature-Attending Recurrent Modules**|              |                  |                  |
| Parameters (millions)                  | 5.1          | 7.1              | 7.6              |
| Input Convolutional dims               | (9, 12, 32)  | (12, 12, 32)     | (7, 7, 32)       |
| Policy-state size                      | 512          | 512              | 1024             |
| Number of modules                      | 4            | 4                | 8                |
| module-state dimension                 | 128          | 128              | 128              |
| Relation heads                         | 2            | 2                | 4                |
| Projection dims $W_1, W_2$             | 16           | 16               | 16               |
| ConvLSTM kernel size                   | 3            | 3                | 3                |
| ConvLSTM hidden size                   | 32           | 32               | 32               |
| **LSTM**                               |              |                  |                  |
| Parameters (millions)                  | 5.6          | 7.2              | 7.6              |
| LSTM Hidden size                       | 896          | 768              | 1024             |
| **Attention Augmented Agent**          |              |                  |                  |
| Parameters (millions)                  | 5.1          | 6.9              | 7.5              |
| ConvLSTM kernel size                   | 3            | 3                | 3                |
| ConvLSTM output size                   | 128          | 128              | 128              |
| LSTM hidden size                       | 704          | 512              | 960              |
| Number of attention heads              | 4            | 4                | 4                |
| Attention query MLP size               | (256, 256)   | (256, 256)       | (256, 256)       |
| Positional basis dim                   | 4            | 4                | 4                |
| **RIMs**                               |              |                  |                  |
| Parameters (millions)                  | 5            | 6.6              | 7.6              |
| Number of RIMs                         | 12           | 9                | 9                |
| Individual RIM size                    | 128          | 128              | 128              |
| Observation heads                      | 6            | 6                | 6                |
| Communication heads                    | 6            | 6                | 6                |
| Observation head size                  | 32           | 32               | 32               |
| Communication head size                | 32           | 32               | 32               |
| Basis size (learned)                   | 4            | 4                | 4                |
| Dropout                                | 0.2          | 0.2              | 0.2              |
B ADDITIONAL EXPERIMENTS

B.1 GENERALIZING MEMORY-RETENTION TO NOVEL SPATIAL COMPOSITIONS OF OBJECT-DYNAMICS

We use a variant of the the task in §5.1. The main difference is that in this setting, the dancers dance in parallel as opposed to in sequence. This task is no longer a test of memory but only a test of whether the agent can recognize separate object-motions.

![Graph](image)

Figure 8: We present the success rate means and standard errors computed using 5 seeds. We find that FARM more quickly learns and generalizes. The next best performance comes from using an LSTM. These results indicate that using spatial attention is an impediment to learning to recognize object-motions.

We present results in Figure 9. In the parallel dancing setting, we find that only FARM and the LSTM can learn these tasks efficiently. Both baselines that use spatial attention learn more slowly and with higher variance.

B.2 GENERALIZING TO AN UNSEEN NUMBER OF DISTRACTORS

We study this with the “Place X next to Y” task in the BabyAI gridworld (Chevalier-Boisvert et al., 2019) (Figure 17). The agent is a red triangle. Other objects can be squares, boxes or circles and they can take on 7 colors. The agent receives a partial, egocentric observation of the environment (Figure 17 right) and is given a synthetic language instruction. The agent gets a reward of 1 if chooses the correct dancer, and 0 otherwise. During training the agent sees either 0 or 2 distractors. During testing, the agent sees 11 distractors. As the number of distractors increases, the likelihood a distractor is either (a) confounding with the task objects or (b) blocks/confuses the agent also increases.

![Graph](image)

Figure 9: RIMs, which uses spatial attention, better generalizes to more distractors. We show train and test success rate performance for “Place X next to Y” in the BabyAI environment (10 runs).
We present results in Figure 9. On the left two panels, we present training results for \( \{0, 2\} \) distractors. All architectures can learn this task. On the right-most panel, we present test results for 11 distractors. FARM and an LSTM get comparable performance (\( \approx 70\% \)). RIMs has the best generalization success rate (\( \approx 80\% \)).

B.3 Analysis of representations learned for KeyBox-like environment

We found no way to programmatically categorize the agent’s experience with object-configurations. Thus, we found no way to study the representations learned for subsections in the KeyBox environment. In order to study this question, we created a toy “Abstract MDP” environment (Figure 18). Importantly, each episode consists of performing a task in \( 3 \times 3 \) environment. This is similar to the KeyBox task since its a sequence of \( 3 \times 3 \) subsections. In this task, there are a fixed number of abstract MDPs which have their own unique object-placement, which mimics the fixed number of object-configurations that can be sampled for the KeyBox task. For an example of these differences, see Figure 18. The object-placements of an abstract MDP are identical but the actual objects in the positions are completely random (i.e. both shape and color are random). We sample 1000 episodes from 20 abstract MDPs. By training FARM on this environment, we can see how it uses different modules to represent different categories. In this experiment, FARM had 4 RNNs of dimensionality 20, requiring that they all be used. In order to see how FARM represents these MDPs, we study the time-series of sum of each module LSTM-state: \( \sum d_i h_i^{(s)} \). We present results in Figure 10. We find that modules respond to object-configurations, not to object types (e.g. key, ball, or box), nor to colors (green or blue).

Figure 10: Modules respond to object-configuration not to individual object colors or shapes. On the left, we plot the starting state of the episode. On the right, we plot the sum of each module LSTM-state.
C  Full Correlations for Analysis

In this section, we show full plots for the analysis in §5.3.1. We show

1. Average L2 norm of all module-states (trained and random weights) (Figure 13).
2. Comparison of average pair-wise correlation of module-states for trained and random weights (Figure 14).
3. Average pair-wise correlation between L2 norm of all module-states (Figure 15).
4. More in-depth plots of activity and attention coefficients (Figure 16).

Figure 11: Trained weights.

Figure 12: Random weights.

Figure 13: Average L2 norm of module-states. We find that when weights are trained on the task, some modules are selective for different events. For example, Module 4 is selective for “drop wrong key” and module 6 is selective for “pickup correct key”. When we use random weights, we see that all modules have the same activity for all events. This indicates that they have not learned any task-specific activity.
Figure 14: When looking at trained weights (left), we find that pairs of modules will have high correlation on some events and high anti-correlation on other events. For example, modules 7 and 2 correlate for drop wrong object and drop wrong key but anti-correlate pickup wrong object and pickup correct key. If we look at random weights (right), we see that pairs of modules will either fully correlate (modules 6 and 2), fully anti-correlate (modules 6 and 1), or have weak/no correlation (modules 6 and 4) for events. Importantly, we don’t see a significant mixture correlation and anti-correlation like we see with trained weights. This suggests that the random weights have less task-specific learning/uses by the agent.
Figure 15: Average pair-wise correlation between L2 norm of module-states.
Figure 16: Panels (a) and (b) both show that different modules have selective activity on different events. (a) Module 0 exhibits salient activity when the agent moves around an obstacle. (b) Module 6 shows selective activity for representing goal information. (c) Module 6 also shifts its attention coefficients as the agent picks up the goal key. (d) We generally find that multiple modules activate for an event. Here, modules 3 and 6 show correlated activity for picking up a ball or non-goal key. Videos of the state-activity and attention coefficients over test episodes: https://bit.ly/3qCxatr.
D Environments

Figure 17: Place X on Y task in BabyAI environment.

Figure 18: Abstract MDP Environment based on BabyAI.

Figure 19: KeyBox task.

Figure 20: Ballet task.

Figure 21: Additional Environment Images.

D.1 Ballet

Please refer to Lampinen et al. (2021) for details on this task. Our only difference was to use tasks with \{2, 4\} dancers during training and tasks with 8 dancers for testing.
D.2 KeyBox

Observation Space. The agent receives a $56 \times 56$ partially observable, egocentric image of the environment as in Figure [7] right.

Action Space. The action space is composed of the 7 discrete actions turn left, turn right, go forward, pickup object, drop object, toggle, and done/no-op.

Reward function. When the agent completes level $n$, it gets a reward of $n/n_{\text{max}}$ where $n_{\text{max}}$ is the maximum level the agent can complete. We set $n_{\text{max}} = 10$ during training. The agent has $50n$ time-steps to complete a level.

Table 4: Object and colors available for objects in the KeyBox task.

| Set       | Contains                           |
|-----------|------------------------------------|
| Shapes    | ball, key, box                     |
| Colors    | red, green, blue, purple, pink, yellow, white |

D.3 3D Unity Environment

For the “place X on Y” experiments in 3D, all pickupable objects were split into two sets $O_1 = A \cup B$ and all object to place something on into another two sets $O_2 = C \cup D$, as shown in Table 5. Given the challenging nature of the 3D environment (huge number of possible states, partial observability, language commands, long credit assignment), we had to employ a set of curriculum tasks in order for the agents to make any progress on the actual task of interest “Put X on Y”. The agent co-trained on the full set of tasks. This was possible since we used a distributed A3C setup for our training (Espeholt et al., 2018), where each of the actors generating the experience was running on one of the possible training levels. The different training tasks used during training and evaluation are shown in Table 6.

All episodes lasted for a maximum of 120 seconds and an action repeat of 4 was used. The images observations were rendered at $96 \times 72 \times 3$ and given to the agent along with a text language instruction, where each word in the instruction was mapped into a continuous vector of size 128 using a fixed vocabulary of maximum size 1000.

Reward function. An agent gets a reward of 1 if it completed the task and 0 otherwise.

Action Space. The action space for the experiments in 3D Unity Environment was 46 discrete actions that allow the agent to move its body and change its head direction, to grab objects while moving and manipulate the held objects by rotating, pulling or pushing the held object. The object is while as long as the agent is emitting a GRAB action, and dropped in the first instance that a GRAB action is not emitted. The full list of possible actions in the 3D Unity Environment environment is presented in Table 7.

Table 5: Object and color set splits for the 3D Unity Environment “Put X on Y” experiments.

| Set                      | Contains                                                  |
|--------------------------|-----------------------------------------------------------|
| Set A (pickupable objects)| toilet roll, toothbrush, toothpaste                       |
| Set B (pickupable objects)| bus, car, carriage, helicopter, keyboard                 |
| Set C (support object)   | stool, tv cabinet, wardrobe, wash basin                   |
| Set D (support object)   | bed, book case, chest, dining, table                      |
| Colors                   | red, green, blue, aquamarine, magenta, orange, purple, pink, yellow, white |
Table 6: Descriptions of all the tasks used during training and evaluation. D refers to number of distractors and S to the room size.

| Task name       | S  | D  | Description                                                                                                                                 |
|-----------------|----|----|---------------------------------------------------------------------------------------------------------------------------------------------|
| Find X          |    |    | The agent is spawned randomly.                                                                                                              |
| (Set A or B)    | 4 × 4 | 5  | Room has 3 objects from Set A (or B) and 3 from C ∪ D and instructed to go to an object from Set A (or B). The purpose of these training tasks is   |
|                 |    |    | to associate objects from Set A and B with their names and the “find” instruction with finding them.                                          |
| Find Y          |    |    | The agent is spawned randomly.                                                                                                              |
| (Set C ∪ D)     | 4 × 4 | 5  | Room has 3 objects from Set A (or B) and 3 from C ∪ D and instructed to go to an object from Set C ∪ D. The purpose of these training tasks is to   |
|                 |    |    | associate objects from Set C ∪ D with their names and the “find” instruction with finding them.                                               |
| Lift X          |    |    | The agent is spawned randomly.                                                                                                              |
| (Set A or B)    | 4 × 4 | 5  | Room has 3 objects from Set A (or B) and 3 from C ∪ D and instructed to lift an object from Set A (or B). The purpose of these training tasks is   |
|                 |    |    | to associate the “lift” instruction with lifting the said object.                                                                             |
| Put X near Y    |    |    | The agent is spawned randomly.                                                                                                              |
| (X = Set A or B, | 3 × 3 | 0  | Room has 1 object from Set A (or B) and 1 from C ∪ D and instructed to put the object from Set A (or B) near the other. The purpose of these     |
| Y = Set C ∪ D)  |    |    | training tasks is to learn to move one object near another before putting it on it.                                                           |
| Put X on Y      |    |    | The agent is spawned randomly.                                                                                                              |
| (X = Set A or B, | 3 × 3 | 0  | Room has 1 object from Set A (or B) and 1 from C ∪ D and instructed to put the object from Set A (or B) on top of the other. The purpose of these    |
| Y = Set C ∪ D)  |    |    | training tasks is to learn to move one object and place it on top of another.                                                               |
| Put X on Y      |    |    | The agent is spawned randomly.                                                                                                              |
| (X = A, Y = D)  | 4 × 4 | 4  | Room has 3 objects from Set A (or B) and 3 from Set D (or C) and instructed to put the object from Set A (or B) on top of the other. This is the    |
| or X = B, Y = C |    |    | training task most similar to the test task and requires mastering all the other ones.                                                     |
| Put X on Y (test)| 4 × 4 | 4  | The agent is spawned randomly.                                                                                                              |
| (X = A, Y = C)  |    |    | Room has 3 objects from Set A (or B) and 3 from Set C (or D) and instructed to put the object from Set A (or B) on top of the other. This is the    |
| or X = B, Y = D |    |    | test task.                                                                                                                                |
Table 7: 3d Unity Environment action space.

| General body movement          | Fine grain movement          |
|-------------------------------|------------------------------|
| NOOP                          | MOVE_RIGHT_SLIGHTLY          |
| MOVE_FORWARD_FULL              | MOVE_LEFT_SLIGHTLY           |
| MOVE_BACKWARD_FULL             | LOOK_RIGHT_MID               |
| MOVE_RIGHT_FULL                | LOOK_LEFT_MID                |
| MOVE_LEFT_FULL                 | LOOK_DOWN_MID                |
| LOOK_RIGHT_FULL                | LOOK_UP_MID                  |
| LOOK_LEFT_FULL                 | LOOK_RIGHT_SLIGHTLY          |
| LOOK_DOWN_FULL                 | LOOK_LEFT_SLIGHTLY           |
| LOOK_UP_FULL                   |                              |

| Fine grained movement with grip | General body movement with grip |
|--------------------------------|--------------------------------|
| GRAB + MOVE_RIGHT_MID          | GRAB                           |
| GRAB + MOVE_LEFT_MID           | GRAB + MOVE_FORWARD_FULL       |
| GRAB + LOOK_RIGHT_MID          | GRAB + MOVE_BACKWARD_FULL      |
| GRAB + LOOK_LEFT_MID           | GRAB + MOVE_RIGHT_FULL         |
| GRAB + LOOK_DOWN_MID           | GRAB + MOVE_LEFT_FULL          |
| GRAB + LOOK_UP_MID             | GRAB + LOOK_RIGHT_FULL         |
| GRAB + LOOK_RIGHT_SLIGHTLY     | GRAB + LOOK_LEFT_FULL          |
| GRAB + LOOK_LEFT_SLIGHTLY      | GRAB + LOOK_DOWN_FULL          |
| GRAB + PULL_CLOSER_MID         | GRAB + LOOK_UP_FULL            |
| GRAB + PUSH_AWAY_MID           |                                |

| Object manipulation            |
|--------------------------------|
| GRAB + SPIN_RIGHT              |
| GRAB + SPIN_LEFT               |
| GRAB + SPIN_UP                 |
| GRAB + SPIN_DOWN               |
| GRAB + SPIN_FORWARD            |
| GRAB + SPIN_BACKWARD           |
| GRAB + PULL_CLOSER_FULL        |
| GRAB + PUSH_AWAY_FULL          |
| PULL_CLOSER_MID                |
| PUSH_AWAY_MID                  |