Object-Centric Learning of Neural Policies for Zero-shot Transfer over Domains with Varying Quantities of Interest

Vishal Sharma, Aniket Gupta, Prayushi Faldu, Rushil Gupta, Mausam, and Parag Singla

Indian Institue of Technology Delhi, India
vishal.sharma@cse.iitd.ac.in, cs1190327@iitd.ac.in, csy217548@cse.iitd.ac.in, rushilgupta358@gmail.com, mausam@cse.iitd.ac.in, parags@cse.iitd.ac.in

Abstract

Our goal is to learn policies that generalize across variation in quantities of interest in the domain (e.g., number of objects, motion dynamics, distance to the goal) in a zero shot manner. Recent work on object-centric approaches for image and video processing has shown significant promise in building models that generalize well to unseen settings. In this work, we present *Object Centric Reinforcement Learning Agent (ORLA)*, the first object-centric approach for model-free RL in perceptual domains. ORLA works in three phases: first, it learns to extract a variable number of objects masks, via an expert trained using encoder-decoder architecture, which in turn generates data for fine-tuning a YOLO based model for extracting bounding boxes in unseen settings. Second, bounding boxes are used to construct a symbolic state consisting of object positions across a sequence of frames. Finally, a Graph Attention Network (GAT) based architecture is employed over extracted object positions to learn a dense state embedding, which is then decoded to get the final policy that generalizes to unseen environments. Our experimentation over a number of domains shows that ORLA can learn significantly better policies that transfer across variations in different quantities of interest compared to existing baselines, which often fail to do any meaningful transfer.

Introduction

Deep reinforcement learning (RL) has achieved significant success in learning policies for unstructured environments, e.g., where a state is input as an image. Indeed, Deep RL has often surpassed humans on a number of such problems, such as Atari games (Silver et al. 2016). On the other hand, the task of learning policies that can generalize to unseen environments can be particularly challenging (Kirk et al. 2021). In this paper, we take a step towards strengthening Deep RL's zero-shot transfer capability, by studying the setting of transfer over *quantities of interest*.

Even if a (high-level) domain has an unstructured state, its each environment may be characterized by various (latent) quantities of interest (QoIs), such as number of objects of each type (e.g., number of legs of a centipede, number of balls/bricks in the game, number of enemies to kill), the numerical attributes of each object (e.g., position or velocity of

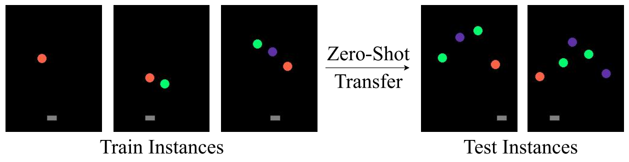


Figure 1: Given the interactions in environments with three or fewer balls, where the goal is to keep the balls from falling, our objective is to learn a policy which would do well on an environment with four or more balls.

each object, maximum force of every muscle), or other derived features that make the instance easy or hard (e.g., distance of the robot to the goal). Our goal is to study transfer across these QoIs. In particular, given a small set of learning environments governed by a set of QoIs, how do we effectively generalize to environments with unseen (generally higher) values of these QoIs? Figure 1 shows an example domain, where transfer is over the number of balls in game playing.

A natural question arises: what might be an effective model for this problem? Our premise is that existing RL models which work directly on the perceptual input, oblivious of the object representations may not generalize - a thesis confirmed by our experiments. We take recourse to recently developed object-centric models for images, and videos (Burgess et al. 2019; Kossen et al. 2020; Locatello et al. 2020), which have shown a lot of promise in achieving better interpretability, robustness to noise, and ability to generalize to unseen settings. Motivated by these works, we use the object-centric representations for the task of model-free RL and propose the first object-centric approach for learning neural policies, which can transfer well across QoIs in the underlying environment in a zero-shot manner.

Our model, referred to as Object-centric Reinforcement Learning Agent (ORLA), works in three phases. (1) We extract object masks that encode object positions in a sequence of frames. This is done using a novel pipeline where we first train an expert using an auto-encoder-based architecture where the object masks are regularized by a Gaussian distribution. The data from the expert is used to do supervised fine-tuning of a YOLO (Jocher et al. 2022) based model for learning object bounding boxes. This model has the capability to generalize to a variable number of objects. (2)

The bounding boxes output by YOLO are used to retrieve object positions in each frame to build an *object-centric* state representation for the underlying MDP. (3) This state representation is passed through a Graph Attention Network (Veličković et al. 2018) (GAT) to learn a dense state embedding, which captures object interactions. Each object node in GAT maintains numerical attributes of that object (e.g., position, velocity) as explicit features, enabling better transfer over QoIs that depend on such attributes. The dense embedding is then passed through an action decoder network to output the policy in the current state. This policy naturally generalizes to unseen environments, due to the inductive nature of GATs.

We design two different categories of domains to test ORLA compared to CNN-based agents and other relevant baselines. In the first category of domains, we study game-playing with balls and pedal, where we transfer across the number of balls, and motion dynamics in two different settings: (i) vertical ball movement, and (ii) diagonal ball movement. In the latter case, we also test with balls having different behavioral characteristics. In the second category of domains, we study navigation in environments with (i) unseen distance from the goal, and (ii) variable number of sub-goals in an adaptation of the traveling salesman problem. Our results indicate the superior performance of our approach compared to baselines, which often fail to do any meaningful transfer.

In summary, we make the following contributions. (1) We study the transfer in RL task, (a) on domains that are observed as unstructured images, even if they have underlying objects and other structures, and (b) for transfer to domains that vary quantities of interest, such as the number of objects or other numerical attributes of the objects/domain. (2) We present an object-centric RL formulation where the image is converted into a symbolic object representation, and RL agent is trained on top of the symbolic representation. (3) We construct new domains and experimental settings to study this task. (4) Through a series of experiments, we find that our object-centric RL is able to effectively achieve the required transfer to target domains, whereas other baselines are not able to do any meaningful transfer.

Related Work

Object-centric Representations: MONet (Burgess et al. 2019) learns to decompose scenes into various objects using an unsupervised loss. COBRA (Watters et al. 2019) uses MONet to learn a transition model of a continuous control environment to be used in model predictive control. Though similar to our work in spirit, their architecture does not support interactions among objects and does not do model-free RL. STOVE (Kossen et al. 2020) uses variational autoencoders to explicitly predict the object positions for learning the dynamics of the environment, rather than the policy. Another series of works (Kipf, van der Pol, and Welling 2020; Locatello et al. 2020; Wu et al. 2022) learn object-centric representations using slot attention for learning the dynamics of the environment, but do not focus on learning the policy.

Transfer Learning in RL: Earlier works use domain randomization for generalization in RL, focusing primarily on how to sample a set of training instances from an underlying distribution with the hope that the test instance will be close to the training instances (Kirk et al. 2021). Another line of work takes an object-centric approach; for example, NerveNet (Wang et al. 2018) studies the size and disability transfer tasks by formulating an agent as a graph and learning a Graph neural network-based policy. Shared Modular Policy (Huang, Mordatch, and Pathak 2020) (SMP) tries to learn a shared policy for multiple agent morphologies. However, both NerveNet and SMP assume that objects, their properties, and whether they affect each other or not are explicitly provided. That essentially means that nodes, node features, and edges of the graph are provided by the environment. This significantly restricts their application in comparison to ours, where we extract these from raw image frames in a self-supervised manner. (Zambaldi et al. 2019) also do experiments on the size-transfer task, by viewing the output of a CNN as a set of entities and using attention to learn a policy; however, they do not predict the positions of the objects explicitly, and the entities do not directly tie to a specific object in the scene. COBRA (Watters et al. 2019) also studies transfer learning but takes a model-based approach rather than a model-free one like ours. For a broad overview of generalization in RL, we direct readers to the survey (Kirk et al. 2021).

Generalized Neural Policies in Planning: Learning generalized policies that can solve any instance of a domain is a well-studied problem in planning (Srivastava, Immerman, and Zilberstein 2008; Hu and De Giacomo 2011; Belle and Levesque 2016). A series of recent works (Garg, Bajpai, and Mausam 2019, 2020; Sharma et al. 2022; Ståhlberg, Bonet, and Geffner 2022) focuses on learning generalized neural policies by converting an environment into an instance-graph and then learning a policy to study the size-transfer task in stochastic relational planning. However, unlike our work, where we learn the graph's structure from raw images, they rely entirely on the available transition model of the environment to learn the instance-graph.

The ORLA Agent

We are interested in learning generalizable neural policies that would work for zero-shot transfer over quantities of interest (QoIs) in a domain. In our setting, these QoIs may be latent, i.e., may not be directly accessible in the interactions with the environment, e.g., motion dynamics. Say, we are playing a game in an environment with perceptual input, with goal as moving the pedal to prevent the ball(s) from hitting the ground (Figure 1). We ask: if the agent is allowed to interact with environments with k or fewer balls, and learns a policy over them, would it generalize to play the same game in an environment with greater than k balls? Or equivalently, would the learned policy generalize when the motion dynamics in the game are different from those seen during training? We note that humans have a remarkable capability to achieve this kind of transfer, but it is not clear if modern-day AI agents can achieve this effectively.

A naïve approach to do this would be to train a standard

RL model, which takes a sequence of frames as input, and then produces an action to be taken as output. These algorithms have been shown to do very well, often beating human players, such as in the Game of Atari. But the problem, in this case, is more challenging since the network has to learn to transfer across variations in QoIs. We first did a simple experiment to examine whether a Dueling DQN based agent (Wang et al. 2016) generalizes in such a setting. Interestingly, it fails completely on this task (see Section). A follow-up question would be: what would be a better way of modeling such transfer tasks? What if we could directly learn the policy over a *symbolic* state representation, such as object positions and their interactions?

We borrow some definitions and notations from the area of Relational Markov Decision Processes (Boutilier, Reiter, and Price 2001; Sanner 2010) to define our problem statement. We deal with structured domains, consisting of objects of various types that can interact with each other. Each domain is characterized by some *quantities of interest* (QoIs), which are human interpretable. In this set-up, a domain can be thought of as a parameterized environment, where a specific assignment to the parameters instantiates a (ground) environment in which an agent interacts. Then, given a set of training environments of a domain, our objective is to learn a policy that is applicable to any environment of that domain.

Motivated by the earlier discussion, we take a three-pronged approach to attack this problem: (1) We first propose to extract object positions in a self-supervised manner without any object-level annotations. (2) The extracted positions are used to construct a symbolic state representation. (3) We learn a policy over the symbolic state in a manner that could generalize to varying values of QoIs. We next describe each of these aspects of our model in detail.

Self-Supervised Object Extraction

The aim of our Object Extractor (OE) is to extract the positions of each object in a given frame. Object extraction is a well-studied problem in computer vision, but most successful models require supervised data (Jocher et al. 2022; He et al. 2017), which is not directly available to us. Therefore, we pose the following question: can we use a supervised object extractor model pre-trained on an existing dataset and fine-tune it on our training instances in a self-supervised way? To this end, we propose a novel self-supervised training algorithm that generates a supervised dataset to fine-tune a pre-trained model for object extraction. Our algorithm first learns a set of *expert object extractors* (expert-OE), $Expert = \{Expert_1, ..., Expert_N\}$, where each $Expert_i$ learns to extract objects in the frames from the i^{th} environment in an unsupervised manner. We note that a single environment corresponds to a specific combination of values for QoIs, and we learn a different expert for each environment. Next, these experts are used to generate a single dataset of $\{(frame, objects)\}$, corresponding to all the training environments. Finally, a *supervised object extractor*, applicable on any environment from the domain, is fine-tuned using this dataset. We next describe the details.

Environment Specific Object Extraction The Expert Object Extractor (expert-OE) of an instance is an auto-encoder architecture that takes a frame in an environment as input and locates objects in it. For this, it first disassembles a given image (f_t) into a set of constituting objects and then uses a decoder to compose the objects to form the original input image. There are three sub-modules: 1) An Attention U-Net (Oktay et al. 2018) that takes the image as input and returns a set of K object masks (where K is the maximum number of objects in the instance), 2) A Gaussian module that enforces each object mask to look like a 2D Gaussian centered at the object's center, and 3) A Decoder that reconstructs the input image.

Encoder: Let there be an environment I with K objects in it. Given a frame f_t, the Attention U-Net gives K object masks $\{m_t^i\}_{i=1}^K$ representing the K objects in the instance (See Figure 2). Next, we want each of these object masks to look like a 2D Gaussian with the mean at the center of one of the objects. For this, for each object mask m_t^i of object i, our Gaussian module generates a 2D Gaussian mask with mean as $(\sum_j j * \sum_k m_t^i(j,k), \sum_k k * \sum_j m_t^i(j,k))$ and standard deviation as Σ[1]. This is similar to the Gaussian Module used in I-CSWM (Gupta et al. 2021). Here, the mean of each g_t^i represents the position of the i^{th} object. Next, to capture each object's visual depiction (e.g. color, texture, shape), a content image is generated as $c_t^i = f_t \otimes g_t^i$, where $\otimes$ represents the Hadamard product.

Decoder: Our decoder outputs the regenerated input image ($\hat{f}_t$) by taking all the object contents c_t^i and combining them at their respective positions (mean of the corresponding g_t^i) on a colored background[2]. For each training instance i of a given domain, we play 25 episodes using a random policy and collect a very small dataset D_i of states seen (ref. Table 1 in Appendix for exact details). We train each instance's expert-OE independently using the loss: $loss_{expert} = mse(f_t, \hat{f}_t) + \sum_{i=1}^K mse(m_t^i, g_t^i)$.

Generalized Object Extraction Next, for each training Environment i, we create a supervised dataset by labeling each image in D_i using the corresponding expert-OE i.e. $D_i^E = \{(f, Expert_i(f))| \ \forall f \in D_i\}$. Finally, we fine-tune a YOLO (Jocher et al. 2022), pretrained on ImageNet, jointly over all environments using the dataset $D = \cup_i D_i^E$. We picked YOLO as it satisfies our requirement of transfer, i.e., it can extract a varying number of objects in the frame. For an input image in an environment, YOLO provides a set of bounding boxes around objects along with a confidence value. We pick the bounding boxes with the top K confidence scores, where K is the maximum possible number of objects in that environment. We define the position $p_t^k = [p_t^k[x], p_t^k[y]]$ of an object o^k as the center of its bounding box.

We also need to assign a type to each object. For example, the balls and the pedal should be assigned a differ-

[1]for our expts, we assume a diagonal Σ with $\sigma_x = \sigma_y = 2.5$.

[2]In our work, all our environments had fixed background color. However, we can easily relax this assumption by using the standard practice of subtracting the mean image of the dataset from f_t.

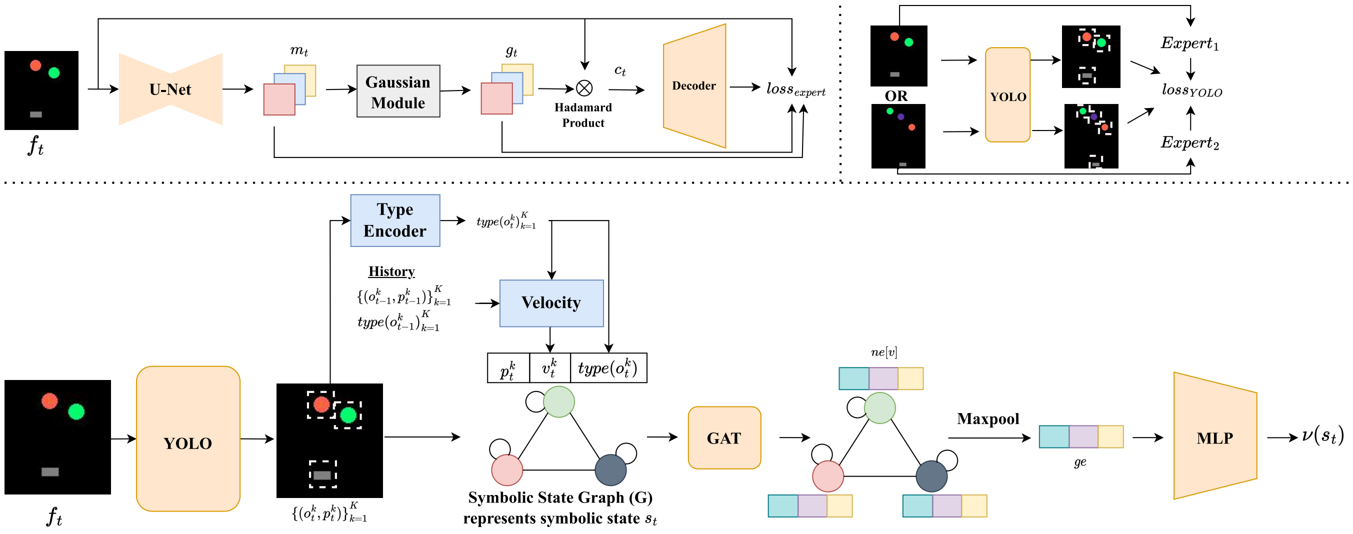


Figure 2: (Top Left) The figure shows training details of our unsupervised expert Object extractor. (Top Right) Shows training of YOLO based object extractor using multiple Experts. (Bottom) The Figure shows a forward pass from ORLA that takes an image as input, converts it into a symbolic state graph, and then uses GAT based architecture to get the policy.

ent type. We assume that the total number of types across all the environments in the domain is known apriori. Then, each object is assigned a type using k-means clustering over the extracted masks. Specifically, we assign $type(o^k)$ =k-means($YOLO(f_t)[o^k]$). During training, the clustering is done over the set of objects $\{Expert_i(f)\ \forall f \in D\}$, and assuming the number of clusters to be equal to the given number of object types in the domain.[3] During inference, we simply assign the type of object using the k-means.

Symbolic State Representation

Like conventional deep RL agents that stack the last few frames to incorporate the history, we incorporate the history in our symbolic state representation. However, since our OE gives an unordered set of objects, so it is not straightforward to provide the history; We first have to match each object o^k at time t to some object o^l at time $t-1$ (called history-object). In our case, for an object o^k, we pick the object o^l nearest to its position in the last time step, i.e., $l = \underset{k' \in L}{argmin}(dist(p_t^k, p_{t-1}^{k'}))$, $dist$ denotes the Euclidean distance, and L denotes all objects at time $t-1$ with type same as o^k. In our experiments, we found this simple method to work satisfactorily and we leave the use of finer methods for future work.

Next, rather than simply stacking object positions at the last few time steps, we compute a velocity vector at time t. It computes difference in positions and direction of change for each object and is given as $v_t^k = [p_t^k[x] - p_{t-1}^l[x], p_t^k[y] - p_{t-1}^l[y], tan^{-1}(\frac{p_t^k[y]-p_{t-1}^l[y])}{p_t^k[x]-p_{t-1}^l[x]})]$, where o^l is the history-object of the object o^k.

[3]If no. of object types is not known, an off-the-shelf method may be used to find the number of clusters, e.g. Elbow method

We organize the extracted objects in the form of a graph called *Symbolic State Graph (SSG)* $G(V, E)$ representing a symbolic space. Its vertex set V contains a node for each object detected in the scene, and has a fully connected adjacency matrix, i.e., $E = \{(u, v)|\forall u, v \in V\}$. A node for an object o^k has input features given by $[p_t^k \mid\mid v_t^k \mid\mid type(o^k)]$, where $\mid\mid$ denotes the concatenation operator.

Learning a Generalized Policy

We will now discuss the details of our architecture to learn a policy in the symbolic state. ORLA can be trained using any standard deep RL algorithm. In our current implementation, we have used a value-network based algorithm as described below. We also share details of how to adapt our set-up to other RL algorithms.

Value Network: Our value network is primarily based on Graph Attention Network (Veličković et al. 2018). First, we compute a set of node embeddings (ne) by applying a GAT on G. For a vertex $v \in V$, $ne[v] = GAT(G)[v]$. Next, a global embedding representing a global view of the state is computed by feature-wise pooling over all node embeddings, i.e., $ge = maxpool_{v \in V}(ne[v])$. Finally, a feed-forward network (MLP in Figure 2) takes ge as input and predicts the value of the state $\mathcal{V}(s) = mlp_{\mathcal{V}}(ge)$ and the Advantage function of an action $a \in A$ as $\mathcal{A}(s, a) = mlp_{\mathcal{A}}(ge)$. The q-value of an action $a \in A$ is computed as $q(s, a) = \mathcal{V}(s) + \mathcal{A}(s, a) - \frac{1}{|A|}\sum_{a' \in A}\mathcal{A}(s, a')$ where A denotes the set of all actions.

Training Details: We adapt dueling DQN to training on multiple instances and keep a separate experience replay buffer for each training instance. Each training instance is picked in a round-robin fashion to generate an episode, and the (s_t, a_t, r_t, s_{t+1}) pairs of that episode are pushed into the instance's experience buffer. Note that s_t are state representation of a frame f_t. A batch is sampled from the instance's experience buffer to update the value network.

While training the value network above, we freeze the weights of the YOLO. Expermenting with a jointly trained model is a direction for future work. We note that ORLA can be adapted to any other base algorithm by changing the final MLP that processes the ge. For example, to use an actor-critic method, the final MLP can be forked to provide the policy and a baseline function. We tried PPO (Schulman et al. 2017), A2C (Mnih et al. 2016), and dueling DQN (Wang et al. 2016) in our initial experiments and found that, in general, across all baselines, Dueling DQN performed best. We therefore use dueling DQN for ORLA and all baselines in our experiments for a fair comparison. Exploring other RL algorithms in detail is a direction for future work.

Experiments and Results

We compare four models in our experiments.

(1) **Random Policy (RND)**, which takes a random action with uniform probability.

(2) **Nature CNN (CNN)** represents a model with unstructured representation space. We use a nature CNN (Mnih et al. 2015) architecture with 3 CNN layers followed by dense layers and train it jointly on the set of training environments of a given domain.

(3) **Gold-GAT (G-GAT)** fetches the gold positions of each object in the frame from the environment and use these, rather than from our generalized object extractor's (YOLO) output, to train the value-network of ORLA while keeping everything else exactly the same as in ORLA. This helps us highlight the effectiveness of our object extractor.

(4) **ORLA**, for which we use the standard pretrained YOLO architecture (Jocher et al. 2022). We use a GAT with 2 layers. Our $mlp_{\mathcal{V}}$ has one hidden layer with 128 units and a single output unit. And, $mlp_{\mathcal{A}}$ has one hidden layer with 128 units and an output layer with units equal to the size of action space.

For finer details on the architecture and training, refer to section in the Appendix. We also tried the method proposed in (Zambaldi et al. 2019). However, their code is publicly not available, and we did not get any response from the authors. We tried replicating their architecture on our own, but it failed to train even on training environments for multiple hyperparameter settings.

We now discuss two case studies. In each case study, we have two domains that capture various complexities of the underlying task to be performed. For all experiments, we report the zero-shot transfer reward on the test environment vs. the number of frames seen during training (Figure 3). We also show goal reachability on one domain.

Case Study 1: Balls and Pedal

In this case study, we want to study our model's zero-shot transfer capabilities when the QoIs capturing the size and motion dynamics vary in the test environments in comparison to what is seen in the training environments. We create two domains in this category that have two types of objects (a set of balls and a pedal) that can interact with each other and have natural dynamics.

Domain 1: Column-Balls (CB) In the Column-Balls (CB) version of the domain, the balls always move in a fixed column, and the pedal is controlled by the agent using three actions: left, right, and noop. The agent's task is to reach a maximum score of $+21$ with a reward function given as +1 if the ball hits the pedal, -1 if it hits the ground, and 0 otherwise. We create five environments having 1, 2, 3, 4, and 5 balls, respectively. We use environments 1-3 for training and 4-5 for testing. For the training environments, the column of the balls is fixed, but the starting y-point and the direction of movement (up or down) are sampled for each episode. However, during testing, we also sample the column for each ball along with its starting y-point and the direction (up or down) at the start of each episode.

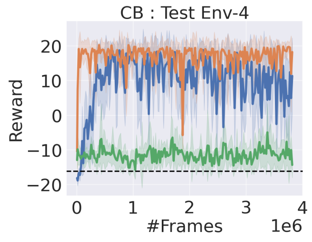
Size Transfer: Figure 3(a) shows the results of the zero-shot transfer reward vs. the number of frames seen for test environments 4 and 5 when trained on environments 1-3. We can see that ORLA is able to generalize to unseen test environments with 4 and 5 balls with a very high reward. As expected, the GAT trained on gold positions (G-GAT) performs the best. ORLA performs only slightly worse than G-GAT, highlighting the efficacy of our object extractor. Interestingly, the CNN policy fails to generalize, performing only slightly better than the random policy (RND).

Looking at the rewards vs. #frames curves on the training environments (Figure 5 in Appendix), we observe that the CNN model achieves the best possible reward in all training environments however it fails to generalize to environments 4-5, highlighting the lack of generalization capabilities of the CNN model.

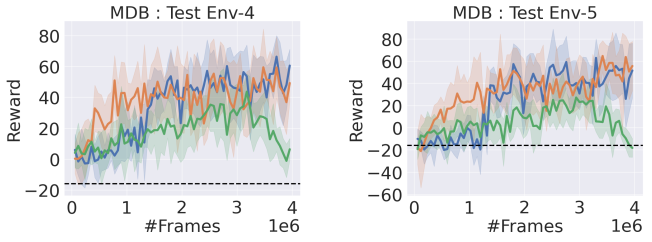
Dynamics and size Transfer (Ball Direction): Next, to study the dynamics transfer task, we create a domain called Diagonal-Balls (DB), where we allow the balls to move freely in any direction rather than just columns. Here the QoI is the direction of the ball movement. We sample the start position and direction of each ball at the start of each episode. We create two test environments of this domain with 4 and 5 balls each.

Figure 3(e) shows the results of this experiment where we compare ORLA, CNN, and RND policies. Interestingly, ORLA is able to do a zero-shot transfer with a very high margin in comparison to the CNN policy even when the underlying dynamics (movement direction and hence velocity direction) of the balls are changed.

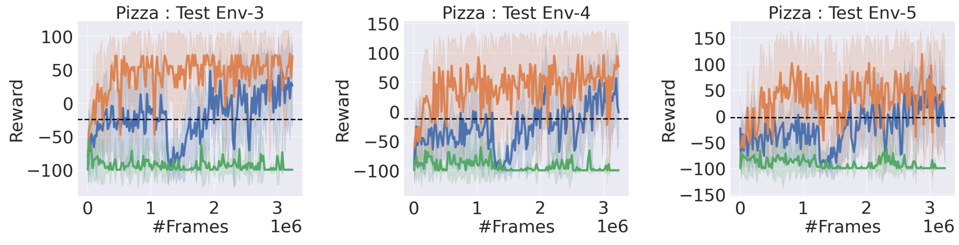
Domain 2: Multitype Diagonal Ball (MDB) In the Multitype Diagonal Ball (MDB) domain, the pedal has to avoid certain balls while trying to hit some other balls, thus increasing the task's complexity as compared to CB. There is a paddle and two types of balls: red and green. The balls can move in any direction, which is sampled at the start of each episode. The task is to get the maximum reward in a fixed-length episode with a reward function given as +1 if the paddle hits the green ball, -1 if the green ball hits the floor, -1 if the red ball hits the paddle, +1 if the red ball hits the ground, and 0 otherwise. We create a total of five environments; the first one has 1 green ball, the second has 1 green and 1 red ball, the third one has 2 green and 1 red ball, the four one has 2 green and 2 red balls, and the fifth one has



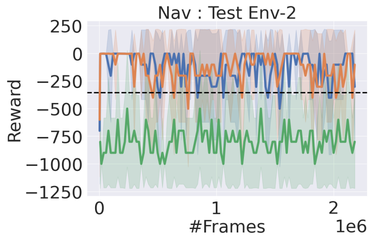
(a) Column Ball (CB)



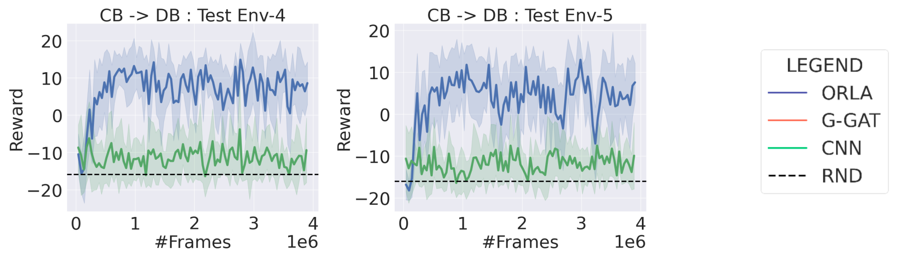
(b) Multitype Diagonal Ball (MDB)



(c) Pizza



(d) Navigation (Nav)



(e) Figure shows the results of the dynamics transfer experiment. It shows zero-shot reward on the Diagonal Ball (DB) domain when the model is trained on the environments 1-3 of the Column Ball (CB) domain

Figure 3: Figure shows the zero-shot transfer reward on the test environments vs. #frames seen on the training environments for all domains. The dashed line represents the reward of a uniform random policy over 500 episodes to counter stochasticity in the policy. For all other methods, we take an average of 10 episodes. (See Figure 5 in the Appendix for graphs on all train and test environments of all domains.)

3 green and 2 red balls. We train on environments 1-3 and test on environments 4 and 5.

Size Transfer: Figure 3(b) shows the results of this experiment regarding the zero-shot size transfer task. Interestingly, ORLA performs equally well as compared to the G-GAT on both test environments. While CNN also performs better than RND, ORLA's performance always remains above CNN's.

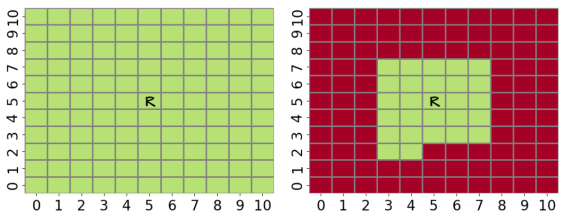
We want to note that since the domains CB and MDB have the same types of objects, we do not retrain our object extractor on MDB. Rather we use the same extractor from CB, highlighting the modularity of our approach.

In summary on this case study, we find that the object-centric approach of ORLA makes it much more amenable to transfer across both size of the domain (number of objects) and numerical attributes of objects (direction of movement). In comparison, the vanilla CNN based approach tends to overfit on the training environments, and is not able to perform much meaningful transfer.

Case Study 2: Robot Navigation

In this second case study, we experiment with two domains set in the context of robot Navigation in a grid world. In the first domain (Navigation), the QoI that we vary from train to test environments is the distance between the robot and the goal location. In the second domain (Pizza), a robot has to reach a goal with a choice to collect some Pizzas (sub-goals) to get a better reward. Here the QoI, which we vary from train to test environments, is the number of sub-goals (Pizzas) available (size-transfer task). The lack of any natural dynamics and inclusion of sparse rewards separates this case study from the earlier one (which has gravity as natural dynamics).

Domain 3: Navigation (Nav) The Navigation (Nav) domain has two objects: a robot and a goal. The aim of the



(a) ORLA (b) CNN

Figure 4: Figure shows the goal coverage for best-trained models of ORLA and CNN on the Navigation domain when trained on a smaller robot to goal distance and transferred to a much larger distance in a zero-shot manner. Here, the robot always starts from the location (5, 5) (marked as R in the figure), and the goal is kept in the remaining 120 cells one by one. In both images, a green cell represents that the robot was able to reach the goal placed at that cell and a red cell represents the robot could not reach the goal.

robot is to reach the goal in a 2D 11×11 grid. The Agent gets a 0 reward when at goal (and the episode ends) and -1 otherwise. At each time step, the (train and test) environment returns an image that represents the grid (without any grid lines). During both training and testing, the robot always starts from the center location (5,5).

Using the Navigation (Nav) domain, we want to study whether a robot can learn to reach a goal kept at a distance much farther than what is seen during training. For this, during training, the location of the goal is kept closer to the robot by randomly sampling the goal locations from the coordinate range $(5 \pm 2, 5 \pm 2)$, i.e., the maximum distance between the goal from the robot is 4. At test time, we again keep the robot at the center of the grid $(5, 5)$ and sample the goal anywhere in the 11×11 grid, i.e., with a maximum distance of 10.

Results: Figure 3(d) shows the zero-shot transfer results on this domain. We see that ORLA performs equally well as compared to the G-GAT, while CNN performs much worse than random. We suspect CNN's performance to be due to overfitting, since it is otherwise able to train well on the training environments.

To further analyze the performance, we additionally check the goal coverage of the robot by placing the robot at the center of the grid (5,5) and placing the goal at all the other 120 locations one by one. Figure 4 shows the goal coverage results for ORLA and CNN policy. Here, each green cell represents that the robot was able to reach a goal kept at that cell when starting from location (5,5). We can see that the goal-coverage of ORLA is 100% whereas CNN could only reach those cells that were seen during the training (with an exception of a couple of additional grid cells).

Domain 4: Pizza We create the Pizza domain to test our model's capabilities on the size-transfer task, where variation in size represents the number of sub-goals available to the robot. In the Pizza domain, there is a 6×6 2D grid world with three types of objects: Robot, Goal, and Pizza, placed at different locations. The goal of the robot is to reach the goal location while collecting as many pizzas as possible. Reaching the goal gives a +5 reward (ending the episode), collecting a Pizza gives a reward of +25, after which the pizza changes color to red and can not be collected again, and -1 otherwise. Note that the robot can directly reach the goal without collecting any Pizza, but for the best reward, it will have to solve a traveling salesman problem with a fixed final node (goal). Thus, this is quite a challenging domain for the network.

We create two training environments, one with a single Pizza and another one with two Pizzas. We tested on three environments with 3, 4, and 5 Pizzas, respectively. The location of the robot, all Pizzas, and the goal are sampled randomly at the start of each episode for both training and testing.

Results: Figure 3(c) shows the results of zero-shot transfer on the three test environments. We again notice that CNN fails to transfer. ORLA performs worse than random in the initial part of the training and trains slower than G-GAT but finally gets close to it. We hypothesize that the slow learning behavior of ORLA is possibly due to the noise in the output of the object extractor and/or in the k-Means clustering used for assigning the type to each object.

In summary, in case study 2, we study long horizon sparse-reward environments. We find that these are more challenging to both ORLA and CNN, but ORLA consistently outperforms CNN, showcasing its better transfer capabilities. Our results also indicate that the lack of good performance may sometimes be due to incorrect identification of object masks (or their clusters), suggesting a research direction for future.

Conclusions & Future Work

In this work, we have tackled the problem of zero-shot policy transfer across domains with variations in underlying quantities of interest. Our approach is object-centric, and consists of three major components. These include (1) a novel object extractor trained in a self-supervised manner, (2) a symbolic state consisting of object positions and velocities constructed from extracted objects, and (3) a GAT based module that takes the symbolic state and decodes it into a policy. Our approach effectively generalizes when transferring across the number of objects, motion dynamics, and distance from the goal state, which is demonstrated through an extensive set of experiments in two different domains, and a variety of settings. In contrast, a vanilla Dueling DQN baseline fails to do any meaningful transfer in most cases.

Directions for future work include experimenting with other model-free RL algorithms, evaluating the efficacy of our model in a few shot transfer setting, trying out on domains with varying object sizes and shapes, as well as more complicated motion dynamics, and extending our object-centric approach to a model-based RL setting.

Limitations

There are some technical limitations and assumptions made by our work. We assume that the number of types of objects (two in case study 1: balls and pedal; 2-3 in case study 2: robot, goal, pizza) are given to the model. We also assume a maximum number of objects known to us – this is used as an input to the object extractor, which attempts to extract no more than this number of objects. Finally, we assume that the background is not noisy and has a consistent color. We believe all of these assumptions can, in principle, be relaxed with suitable modifications to the architecture. Since our focus was to provide the first demonstration of transfer across QoIs, we did not perform these experiments and leave those for future work.

There are no direct ethical implications of this work, as it is still being tested in toy domains, with the goal of enhancing the capability of modern day deep RL systems.

References

Belle, V.; and Levesque, H. J. 2016. Foundations for generalized planning in unbounded stochastic domains. In *Fifteenth International Conference on the Principles of Knowledge Representation and Reasoning*.

Boutilier, C.; Reiter, R.; and Price, B. 2001. Symbolic dynamic programming for first-order MDPs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 1, 690–700.

Burgess, C. P.; Matthey, L.; Watters, N.; Kabra, R.; Higgins, I.; Botvinick, M. M.; and Lerchner, A. 2019. MONet: Unsupervised Scene Decomposition and Representation. *CoRR*, abs/1901.11390.

Garg, S.; Bajpai, A.; and Mausam. 2019. Size independent neural transfer for RDDL planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 631–636.

Garg, S.; Bajpai, A.; and Mausam. 2020. Symbolic network: generalized neural policies for relational MDPs. In *International Conference on Machine Learning*, 3397–3407.

Gupta, R.; Sharma, V.; Jain, Y.; Liang, Y.; Broeck, G. V. d.; and Singla, P. 2021. Towards an Interpretable Latent Space in Structured Models for Video Prediction. *arXiv preprint arXiv:2107.07713*.

He, K.; Gkioxari, G.; Dollár, P.; and Girshick, R. 2017. Mask R-CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)*, 2980–2988.

Hu, Y.; and De Giacomo, G. 2011. Generalized planning: Synthesizing plans that work for multiple environments. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

Huang, W.; Mordatch, I.; and Pathak, D. 2020. One policy to control them all: Shared modular policies for agent-agnostic control. In *International Conference on Machine Learning*, 4455–4464. PMLR.

Jocher, G.; Chaurasia, A.; Stoken, A.; Borovec, J.; NanoCode012; Kwon, Y.; Michael, K.; TaoXie; Fang, J.; imyhxy; Lorna; Yifu, Z.; Wong, C.; V, A.; Montes, D.; Wang, Z.; Fati, C.; Nadar, J.; Laughing; UnglvKitDe; Sonck, V.; tkianai; yxNONG; Skalski, P.; Hogan, A.; Nair, D.; Strobel, M.; and Jain, M. 2022. ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation.

Kipf, T.; van der Pol, E.; and Welling, M. 2020. Contrastive Learning of Structured World Models. In *International Conference on Learning Representations*.

Kirk, R.; Zhang, A.; Grefenstette, E.; and Rocktäschel, T. 2021. A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794*.

Kossen, J.; Stelzner, K.; Hussing, M.; Voelcker, C.; and Kersting, K. 2020. Structured Object-Aware Physics Prediction for Video Modeling and Planning. In *International Conference on Learning Representations*.

Locatello, F.; Weissenborn, D.; Unterthiner, T.; Mahendran, A.; Heigold, G.; Uszkoreit, J.; Dosovitskiy, A.; and Kipf, T. 2020. Object-Centric Learning with Slot Attention. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 11525–11538. Curran Associates, Inc.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 1928–1937. PMLR.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.

Oktay, O.; Schlemper, J.; Folgoc, L. L.; Lee, M.; Heinrich, M.; Misawa, K.; Mori, K.; McDonagh, S.; Hammerla, N. Y.; Kainz, B.; et al. 2018. Attention u-net: Learning where to look for the pancreas. *arXiv preprint arXiv:1804.03999*.

Sanner, S. 2010. Relational dynamic influence diagram language (rddl): Language description. *Unpublished ms. Australian National University*, 32: 27.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sharma, V.; Arora, D.; Geißer, F.; Mausam; and Singla, P. 2022. SymNet 2.0: Effectively handling Non-Fluents and Actions in Generalized Neural Policies for RDDL Relational MDPs. In *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*, volume 180 of *Proceedings of Machine Learning Research*, 1771–1781. PMLR.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529(7587): 484–489.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2008. Learning Generalized Plans Using Abstract Counting. In *Twenty-Third AAAI Conference on Artificial Intelligence*.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022. Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits. *Proceedings of* *the 32nd International Conference on Automated Planning and Scheduling*.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2018. Graph attention networks. *International Conference on Learning Representations*.

Wang, T.; Liao, R.; Ba, J.; and Fidler, S. 2018. NerveNet: Learning Structured Policy with Graph Neural Networks. In *International Conference on Learning Representations*.

Wang, Z.; Schaul, T.; Hessel, M.; Hasselt, H.; Lanctot, M.; and Freitas, N. 2016. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, 1995–2003. PMLR.

Watters, N.; Matthey, L.; Bosnjak, M.; Burgess, C. P.; and Lerchner, A. 2019. Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration. *arXiv preprint arXiv:1905.09275*.

Wu, Z.; Dvornik, N.; Greff, K.; Kipf, T.; and Garg, A. 2022. SlotFormer: Unsupervised Visual Dynamics Simulation with Object-Centric Models. *arXiv preprint arXiv:2210.05861*.

Zambaldi, V.; Raposo, D.; Santoro, A.; Bapst, V.; Li, Y.; Babuschkin, I.; Tuyls, K.; Reichert, D.; Lillicrap, T.; Lockhart, E.; Shanahan, M.; Langston, V.; Pascanu, R.; Botvinick, M.; Vinyals, O.; and Battaglia, P. 2019. Deep reinforcement learning with relational inductive biases. In *International Conference on Learning Representations*.

Appendix: Object-Centric Learning of Neural Policies for Zero-shot Transfer over Latent Symbolic Quantities

Architectures

Expert Object-Extractor: It takes as input an image with 3 channels (RGB) and outputs a set of K Gaussian masks with their means at the objects' centers. We use the Attention U-Net architecture as proposed in the original work (Oktay et al. 2018) for our encoder. It takes a (50, 50, 3) size preprocessed image as input and downsamples it using four convolution blocks with 32, 64, 128, and 256 hidden channels, with 2x2 maxpooling between consecutive layers. It then upsamples the output using convolution blocks with 128, 64, and 32 hidden channels. The last convolution block has output channels equal to the number of objects. We use the learning rate of 5e-4. The output of the Attention U-Net is passed through the Gaussian module to get a gaussian mask for each channel in the extractor output. It creates a gaussian mask with its mean at the object's center and fixed σ_x and σ_y of 2.5. Hadamard Product of the gaussian masks with the original image gives objects' content which are used to regenerate the original image by combining them at their respective positions on the background. We use a seed value of 4 for both Numpy and PyTorch.

YOLOv5: We use the YOLOv5 model as proposed in the work by (Jocher et al. 2022) with their default hyperparameters. The outputs of the Expert Object-Extractors are used to create a labelled dataset to give supervision to fine-tune pretrained YOLO model.

Policy Network: Our policy network uses a GAT based architecture to encode the *symbolic-state-graph* in a fixed-dimension (128 dim) feature vector, which is used to learn the advantage and value functions. The symbolic-state-graph is passed twice through graph attention networks. Maxpooling over the node features gives the fixed-dimension feature vector, which is passed through two MLPs to learn the advantage and value functions. The Advantage network consists of two layers with an output dimension equal to the size of the action space. The Value network consists of 2 layers with output of unit dimension. Both the MLP networks have 128 units in the hidden layers and use LeakyRelu non-linearity. For the policy network, we use the epsilon-greedy method with an initial exploration rate equal to 1, which decreases by a factor of 0.01 after every episode. It uses the Adam optimizer with a learning rate of 0.00025.

Training

We train all models on a cluster of Quadro P5000 GPUs with 16GB GPU memory and with 128 GB RAM. We train each of our Expert Object-Extractors with a maximum time limit of 4 hours and take the checkpoint with the least MSE on a validation dataset. These checkpoints are then used to generate labels for the training dataset to provide supervision to YOLO. We fine-tune YOLO for a maximum of 90 epochs with a maximum time limit of 1 hour time. It takes around 5 minutes to learn clusters in the K-Means clustering module. We give a maximum of 24 hours to train the policy network or other baselines for a satisfactory training time.

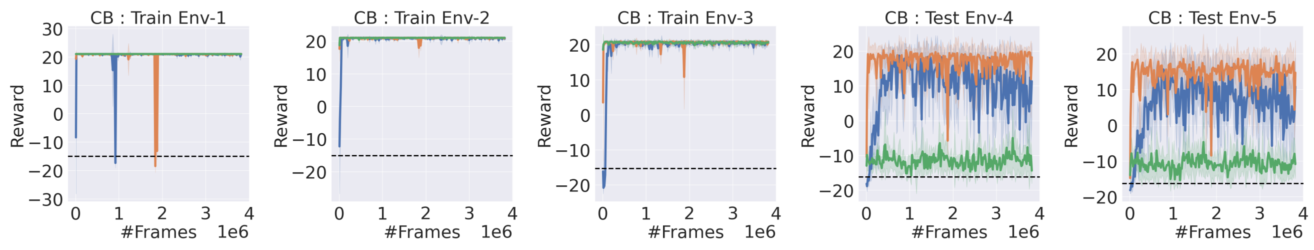
The number of frames used for training the Expert Object-Extractor for different environments is given in Table 1.

Detailed Results

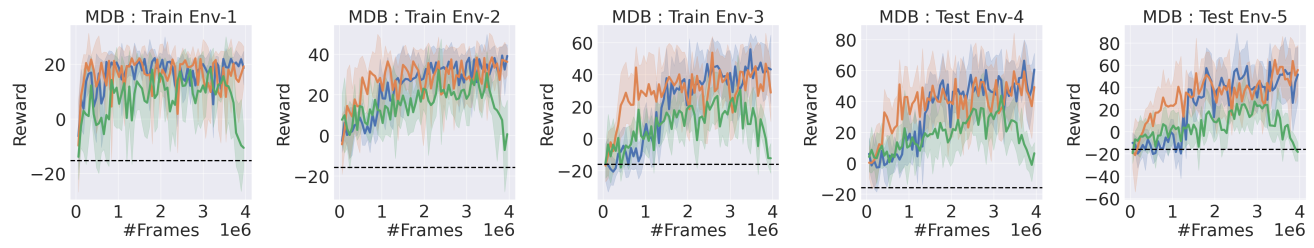
Figure 5 shows both the training and test results on various domains.

Environment	Instance	#Episodes (Train)	#Frames (Train)	#Episodes (Val)	#Frames (Val)
Column Ball (CB)	1	25	23443	10	9438
	2	25	12996	10	5214
	3	25	8961	10	3557
Multitype Diagonal Ball (MDB)	Uses CB's extractor				
Navigation (Nav)	-	25	6770	10	3495
Pizza (Pizza)	1	250	12972	100	5156
	2	250	13482	100	5262

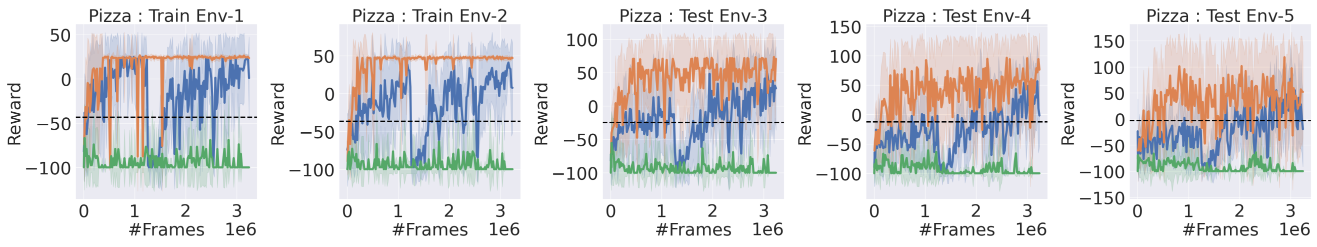
Table 1: Table shows the number of episodes and frames used to train experts in various domains. For MDB, we do not train any experts rather, we use the experts trained on Column Ball



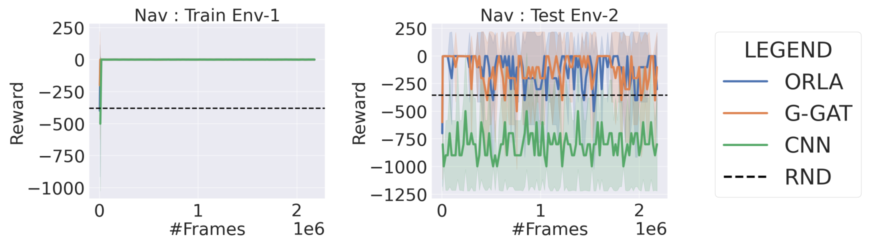
(a) Column Ball (CB)



(b) Multitype Diagonal Ball (MDB)



(c) Pizza



(d) Navigation (Nav)

Figure 5: Figure showing performance (average reward vs. #frames) of various methods on all domains on each of the train and test environments.