Efficient and Interpretable Robot Manipulation with Graph Neural Networks

Yixin Lin*
Facebook AI Research
yixinlin@fb.com

Austin S. Wang
Facebook AI Research
wangaustin@fb.com

Eric Undersander
Facebook AI Research
eundersander

Akshara Rai*
Facebook AI Research
akshararai@fb.com

Abstract

Manipulation tasks like loading a dishwasher can be seen as a sequence of spatial constraints and relationships between different objects. We aim to discover these rules from demonstrations by posing manipulation as a classification problem over a graph, whose nodes represent task-relevant entities like objects and goals. In our experiments, a *single* GNN policy trained using imitation learning (IL) on 20 expert demonstrations can solve blockstacking and rearrangement tasks in both simulation and on hardware, generalizing over the number of objects and goal configurations. These experiments show that graphical IL can solve complex long-horizon manipulation problems without requiring detailed task descriptions.

1 Introduction

Everyday manipulation tasks deal with relationships and constraints between objects and environments. Loading a bowl in a dishwasher requires pre-conditions, like an open dishwasher and a grasped bowl. Specifying such pre-conditions for complex tasks can be tedious and error-prone. This is the central problem we address – how do we learn task structure from a few demonstrations and then generalize to arbitrary numbers of objects and related tasks? Our experiments show that a GNN policy trained using imitation learning (IL) can learn this structure and generalize to tasks of increased complexity and variable numbers of objects, starting with just 20 expert demonstrations (Fig. 1). The GNN structure provides interpretability and invariance to the number of objects in the scene. The main contributions of our work are (1) GNNs as a promising policy architecture for long-term manipulation tasks, (2) IL as a well-suited training scheme for such a policy choice, (3) a modified GNNExplainer to interpret the decisions made by our learned policy. We conduct experiments on a Franka arm in the real world and in two simulated environments - blockstacking & box rearrangement, and dishwasher loading (Figure 1). In dishwasher loading, the robot loads a dishwasher with plates and bowls; in box rearrangement, the robot moves blocks between boxes.

2 Background and related work

Graph neural networks (GNNs) [Battaglia et al., 2018] have effective inductive biases for learning the relations in graph datasets. Graphical representations of scenes have been used for learning high-dimensional dynamics models [Fragkiadaki et al., 2016, Ye et al., 2019], learning object-relevance in problems with large object instances [Silver et al., 2020], visual imitation learning [Sieb et al., 2020, Huang et al., 2019], and high-level policies [Li et al., 2020]. Unlike Sieb et al. [2020], we represent our policy as a trained GNN, and show zero-shot generalization to varied object counts. Instead of RL as in Li et al. [2020], we use IL to improve sample efficiency and generalization (Sec. 4).

Graph neural networks (GNNs) are deep neural networks designed to operate on graphs. Let G be a graph with nodes V and undirected edges E, where each node $v \in V$ is associated with a

*Equal contribution

Physical Reasoning and Inductive Biases for the Real World at NeurIPS 2021.

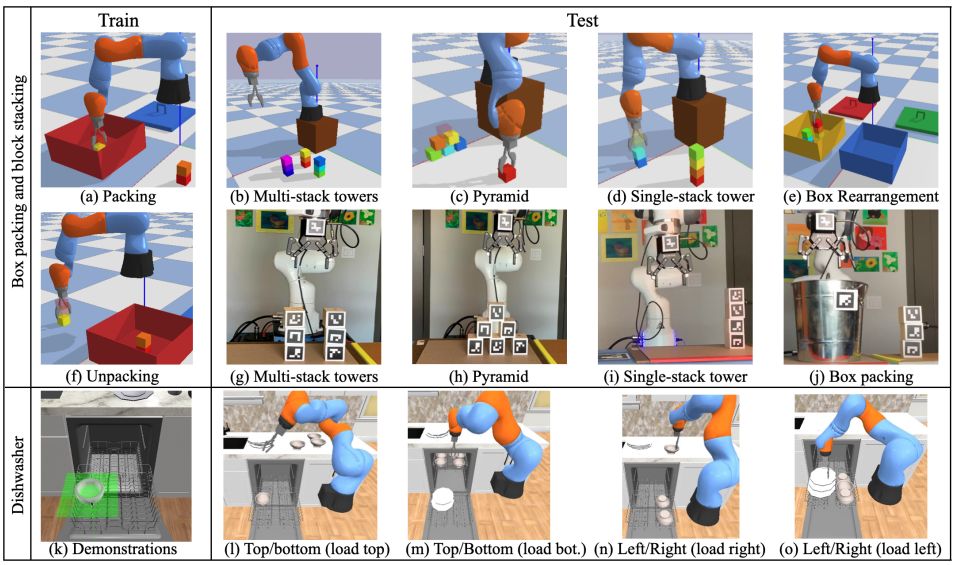


Figure 1: We train a policy on small instances of the problem (left column: (a), (f), (k)) and test generalization in both simulation ((b)-(e)), and on hardware ((g)-(j)). We simulate a complex dishwasher-loading environment, generating point-and-click demos ((k)) and testing on a variety of scenarios ((l)-(o)), described in Section 4.2.

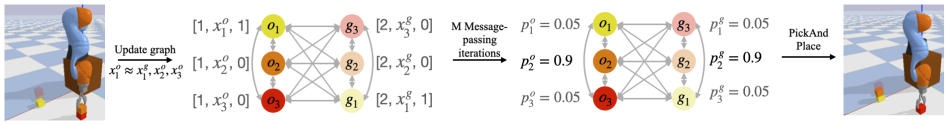


Figure 2: Overview of our algorithm at a timestep. Our method takes in an observation, transforms it into a graph, and passes it to the GNN policy, which selects an object and goal to input to `PickAndPlace` .

d-dimensional feature vector $\phi(v)$. At each layer l and for every node $v_i \in V$, by message-passing, we update the node's feature vector $h_i^l = f_\theta^l(h_i^{l-1}, \{h_j^{l-1}\}_{j \in \mathcal{N}_i})$, where $h_i^0 = \phi(v_i)$ is the input feature and $\mathcal{N}_i$ is its neighborhood. We use Pytorch Geometric [Fey and Lenssen, 2019, Paszke et al., 2019] to parallelize graph operations, and test four architectures (choices of f_θ): GCN [Morris et al., 2019], Sage [Hamilton et al., 2017], Attention [Veličković et al., 2017] and Gated [Li et al., 2015].

3 GNN policies for manipulation

In this section, we explain our formulation which casts manipulation tasks as operations over a graph. We assume a low-level `PickAndPlace` primitive which, given an object and a goal, grasps the chosen object and places it in the desired goal. A learned high-level GNN policy takes a graph state as input and selects the next block and goal location. Dishwasher loading additionally predicts involves orientations, and which action to use, like `OpenTray` for opening a dishwasher tray. For clarity, we will describe the next section using `PickAndPlace` (with dishwasher details in Sec 4.2).

3.1 Problem formulation: Graphical representation of state

We encode the environment scene as a graph, whose nodes consist of the task-relevant entities, such as objects and their target positions (goals). Let there be K objects, and L goals in the scene. We create a graph $G = (V, E)$, where the vertices $V = \{v_k^o\}_{k=1}^K \cup \{v_l^g\}_{l=1}^L$ represent the objects and goals in the scene, giving us a total of $K + L$ nodes. We create a dense, fully-connected graph, where all nodes are connected to all other nodes; $E = \{e_{i,j}\}$ for $i = 1 \dots K + L, j = 1, \dots K + L$.

Each node $v \in V$ has an input feature vector $\phi(v)$, which contains node-specific information. The input features of each node are 5-dimensional: a categorical feature $\{0, 1, 2, 3\}$ denoting if a node is a cover, goal for a cover, block or goal for a block, the 3-dimensional position of the object or goal in the frame of the robot, and a binary feature which is 1 if a goal is filled or an object is in a goal, and 0 for empty goals or objects. The current state graph is input to the GNN policy, which outputs a categorical distribution over objects and goals. The selected object and goal positions are sent as inputs to the `PickAndPlace` primitive. Fig 2 shows our setup for a $K = L = 3$ block stacking trajectory. Our approach also generalizes to different number of goals and objects when $K \neq L$.

3.2 Training the GNN from demonstrations

We pose manipulation as a classification problem at each high-level step where a decision is made over *which* object to move to *where* using *what* action. The output of the GNN policy is $K+L$ dimensional corresponding to the object and goal nodes of the original graph. This is reshaped as two K and L dimensional outputs $V_g^{\text{out}} = \{v_l^g\}_{l=1}^L$ and $V_o^{\text{out}} = \{v_k^o\}_{k=1}^K$. V_o^{out} is then passed through a softmax function to generate a K-dimensional categorical distribution $P_{\text{pred}}^o = \{p_1^o, p_2^o, \cdots p_K^o\}$ depicting the picking probabilities of objects. The GNN chooses the object with the highest predicted probability: $o_* = \arg\max_j p(o_j)$ where $p(o_j) = [\exp(v_j^o)]/[\sum_{k=1}^K \exp(v_k^o)]$. The same transformation is applied to the goals, resulting in $P_{\text{pred}}^g = \{p_1^g, p_2^g, \cdots p_L^g\}$ over the goals. Given target distributions P_{tgt}^o for objects and P_{tgt}^g for goals from expert data, the GNN policy parameters θ are trained to minimize the cross-entropy loss: $\arg\min_\theta[-\sum_{k=1}^K [P_{\text{tgt}}^o]_k \log(p_k^o) - \sum_{l=1}^L [P_{\text{tgt}}^g]_l \log(p_l^g)]$.

The expert demonstrations are also cast as a graph with target output distributions coming from the expert action. We collect N demonstrations of the expert solving the task. At each step t, we extract input-output pairs $\{(s_t = (o_{k=1,\cdots,K}, g_{l=1,\cdots,L})_t, a_t)\}$, where o_i and g_i are the objects and goals in the scene, and $a_t = \{o_t^{\text{exp}}, g_t^{\text{exp}}\}$ is the action taken by the expert, indicating the next object o_t^{exp} to be moved to the next goal g_t^{exp}. a_t is converted into two K and L-dimensional 1-hot target distributions P_{tgt}^o and P_{tgt}^g for goal and object prediction, respectively. $P_{\text{tgt}}^o = \mathbb{1}[o_k = o_b^{\text{exp}}]$, $P_{\text{tgt}}^g = \mathbb{1}[g_l = g_b^{\text{exp}}]$ are one-hot vectors: 1 for the object and goal chosen by the expert, and 0 for all others. GNN parameters θ are trained to minimize the cross-entropy loss between GNN prediction given and P_{tgt}^o and P_{tgt}^g.

3.3 Interpreting the learned GNN policy

Ying et al. [2019] propose a GNNExplainer which determines the importance of neighbouring nodes and input features for decision making in GNNs. We modify this GNNExplainer to suit our problem setting. The output of our trained GNN policy π_θ given an input graph G and features ϕ are two categorical probability distributions $P_{\text{pred}}^o, P_{\text{pred}}^g$. We aim to find a mutated graph G_S and feature mask F, such that the output of π_θ given G_S and masked features $\phi_S = \phi \odot F$ is close to $P_{\text{pred}}^o, P_{\text{pred}}^g$. This setup is different from Ying et al. [2019] where a categorical distribution is predicted for *every* node in a graph; our model instead predicts over *all* nodes. In our analysis, we aim to identify which spatial relationships or neighbour objects contributed most to the policy's decision. Given a trained GNN π_θ and input graph $G = (V, E)$, we aim to find a mutated graph $G_S = (V, E_S), E_S \subset E$ and a feature mask F, such that the mutual information between $Y = \pi_\theta(G, \phi)$, and $Y_S = \pi_\theta(G_S, \phi_S = \phi \odot F)$ is maximized: $G_S, F = \arg\max_{G_S,F} \text{MI}(Y, Y_S) = H(Y) - H(Y|Y_S)$.

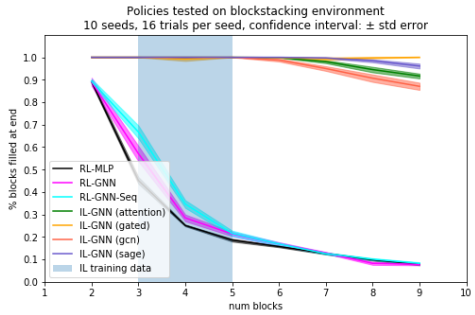
Since $H(Y)$ does not depend on G_S or F, maximizing the mutual information between Y and Y_S is equivalent to minimizing the conditional entropy $H(Y|\pi_\theta(G_S, \phi_S))$. Intuitively, the explanation for Y is a mutated graph G_S and feature mask F that minimize the uncertainty over Y .

4 Experiments

We use a Franka Panda manipulator controlled using the Polymetis framework [Lin et al., 2021], and solve blockstacking and box packing tasks on hardware. For detecting blocks on hardware, we utilize a RealSense depth camera with the ArUco ARTags library [Garrido-Jurado et al., 2016]. In simulation, we create two environments - dishwasher loading in AI Habitat [Szot et al., 2021], and blockstacking and box rearrangement in PyBullet [Coumans and Bai, 2016] - using a 7DoF robot manipulator (KUKA iiwa7). We show train and test environments in Figure 1.

4.1 Block stacking and box packing experiments

Each environment contains K blocks with different initial and goal positions. Success is measured by percentage of goals filled at the end of each trial. This experiment studies the generalization of the trained GNN policy across large number of blocks, multiple boxes and unseen tasks like pyramids and multiple stacks, shown in Fig 1a-j. We compare the 4 GNN architectures from Section 2. All GNNs have 3 hidden layers, 64 hidden units each, & ReLU activations. We also compare our IL approach against RL from scratch using both MLP and GNN architectures, as in [Li et al., 2020]. RL is given a much larger environment interaction budget (16000 interactions) since it must explore the environment without expert demos. We train our approach on 20 expert demos (90 interactions) of packing and unpacking $K = 3, 4$ blocks, then test on unseen tasks. We use the *same* learned GNN policies for all experiments in Table 1a, 1b 1c. We see that our approach outperforms all RL baselines



(a) Generalization over num. blocks in sim.

	6-Pyramid	3-block 3-stack	Box rearrangement
GCN	1.0 ± 0.0	0.88 ± 0.03	0.58 ± 0.11
Sage	1.0 ± 0.00	0.99 ± 0.00	0.95 ± 0.02
Attention	1.0 ± 0.00	1.0 ± 0.00	0.76 ± 0.08
Gated	0.95 ± 0.01	0.6 ± 0.05	0.20 ± 0.06

(b) Generalization of different GNN architectures to unseen tasks in simulation. In 6-pyramid robot stacks blocks in a pyramid, 3-block 3-stack is making 3 stacks of 3 blocks. Box rearrangement involves moving blocks from one box to another.

	4-Blocks	6-Pyramid	3-block 2-stack	Box packing
% Success	1.00	1.00	1.00	1.00
% Correct	0.91	1.00	1.00	0.80

(c) **Hardware**: 20 runs of 4 block stacking, 2-stack 3-blocks, 6-pyramid and box packing (400 `PickAndPlace` movements)

Scenario	6 objects	8 objects	10 objects (training)	12 objects
Top/bottom	0.80 ± 0.00	0.83 ± 0.02	1.00 ± 0.00	0.91 ± 0.03
Left/right	0.70 ± 0.03	0.76 ± 0.02	0.78 ± 0.05	0.79 ± 0.03

(d) **Dishwasher experiments**: we train using 5 demonstrations of the 5-plate, 5-bowl task and test on 2 target configurations: **(a)** bowls top, plates bottom (Fig. 1 (l)-(m)), **(b)** all objects top, with bowls right & plates left (Fig. 1 (n)-(o)).

Table 1: Simulation and hardware experiment results

(Fig 1a), and solves unseen tasks like pyramid stacking, and rearrangement nearly perfectly. We also validate our approach by training GNN policies in simulation and testing on hardware (Fig.1(g-j)) using a Franka arm without additional fine-tuning. Results in Table 1c consist of 400 real-world `PickAndPlace` movements showing that GNN policies are robust to hardware disturbances like perception noise and placement errors. Additional experiment details can be found in Appendix.

4.2 Dishwasher loading experiments

Finally, we apply our method to a more complex task: loading a dishwasher with plates and bowls in different configurations. We build a dishwasher environment in AI Habitat [Szot et al., 2021] using the Replica Synthetic - Apartment 0 dataset (a set of 3D models of an apartment, to be publicly released in the future), with two types of objects (bowls and plates) and a dishwasher with two racks (see Fig.1 (k)-(o)). The training data is created using a game-like interface in a point-and-click manner, where desired dishwasher-loading demonstrations can be easily generated by a layperson.

This environment has additional complexity: (1) multiple object types (bowls/plates), (2) preconditions for feasibility (two trays, loaded only when pulled out) (3) different desired configurations specified in demonstrations (i.e. different ways to load a dishwasher), (4) desired pick and place orientations for objects and (5) multiple actions like `PickAndPlace` and opening/closing trays.

The training procedure is similar to that of the previous experiments, though we are operating in the extremely low-sample regime and only train on 5 expert demonstrations. Results in Table 1d show that the trained policies generalize to varied object numbers despite the additional complexities, indicating our method can scale to more difficult environments with very few demonstrations.

4.3 Explaining the learned GNN policies

To investigate the difference between generalization of GNN architectures in Table 1b, we use the GNNExplainer (Section 3.3) on learned GNN policies. We find Gated GNN learns spatial relations which rely on x, y positions of the blocks, while other architectures learn to use the more informative "unfilled" and z feature. This points to overfitting: Gated uses spatial rules that work for single stacks, but prove insufficient for generalization. Other architectures learn to rely on task-relevant features like block height and empty goal. Here, GNNExplainer sheds light on the difficult-to-interpret outputs of neural networks and provides intuitive explanations of the form "node i was chosen because of its relationship with nodes j; the most important feature was block height z", easing debugging.

5 Conclusion

In this work, we present a graphical policy architecture for manipulation tasks trained with expert demonstrations. Once the graph neural network policies are trained, they demonstrate sample efficient, zero-shot generalization behavior across unseen, larger problem instances, including a real Franka robot. Our work explores research at the intersection of graphical architectures and task planning.

References

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards, 2016.

Yufei Ye, Dhiraj Gandhi, Abhinav Gupta, and Shubham Tulsiani. Object-centric forward modeling for model predictive control, 2019.

Tom Silver, Rohan Chitnis, Aidan Curtis, Joshua Tenenbaum, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Planning with learned object importance in large problem instances using graph neural networks, 2020.

Maximilian Sieb, Zhou Xian, Audrey Huang, Oliver Kroemer, and Katerina Fragkiadaki. Graph-structured visual imitation. In *Conference on Robot Learning*, pages 979–989. PMLR, 2020.

De-An Huang, Suraj Nair, Danfei Xu, Yuke Zhu, Animesh Garg, Li Fei-Fei, Silvio Savarese, and Juan Carlos Niebles. Neural task graphs: Generalizing to unseen tasks from a single video demonstration. In *CVPR*, pages 8565–8574, 2019.

Richard Li, Allan Jabri, Trevor Darrell, and Pulkit Agrawal. Towards practical multi-object manipulation using relational reinforcement learning. In *ICRA 2020*, pages 4051–4058. IEEE, 2020.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32:9240, 2019.

Yixin Lin, Austin S. Wang, Giovanni Sutanto, Akshara Rai, and Franziska Meier. Polymetis. https://polymetis-docs.github.io/, 2021.

Sergio Garrido-Jurado, Rafael Munoz-Salinas, Francisco José Madrid-Cuevas, and Rafael Medina-Carnicer. Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, 51:481–491, 2016.

Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, et al. Habitat 2.0: Training home assistants to rearrange their habitat. *arXiv preprint arXiv:2106.14405*, 2021.

Erwin Coumans and Yunfei Bai. Pybullet. *https://pybullet.org/*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. https://github.com/DLR-RM/stable-baselines3, 2019.