One-shot imitation learning with Graph Neural Networks for Pick-and-Place manipulation tasks

Francesco Di Felice, Salvatore D'Avella, Alberto Remus, Paolo Tripicchio and Carlo Alberto Avizzano

***Abstract*—The proposed work presents a framework based on Graph Neural Networks (GNN) that abstracts the task to be executed and directly allows the robot to learn task-specific rules from synthetic demonstrations given through imitation learning. A graph representation of the state space is considered to encode the task-relevant entities as nodes for a Pick-and-Place task declined at different levels of difficulty. During training, the GNN-based policy learns the underlying rules of the manipulation task focusing on the structural relevance and the type of objects and goals, relying on an external primitive to move the robot to accomplish the task. The GNN-policy has been trained as a node-classification approach by looking at the different configurations of the objects and goals present in the scene, learning the association between them with respect to their type for the Pick-and-Place task. The experimental results show a high generalization capability of the proposed model in terms of the number, positions, height distributions, and even configurations of the objects/goals. Thanks to the generalization, only a single image of the desired goal configuration is required at inference time.**

***Index Terms*—Learning from Demonstration, Imitation Learning, Task and Motion Planning**

I. Introduction

When robots appeared in the industry back in the 90s, the most popular programming method for the first decades was online programming due to its simplicity and intuitiveness [1]. Online programming is still nowadays the first choice for small and easy tasks, but when the tasks become more complex, offline programming is the preferred and most effective alternative. However, even though robotic research advances have grown very fast in the last few years, industrial robots execute mostly pre-programmed tasks. And if something changes, they still need to be programmed again by expert operators. Programming-by-demonstration [2] has shortened the time required to set up new procedures, but still, robots do not present a high level of autonomy. In line with the concept of flexibility pushed by Industry 4.0 [3], vision-guided robots are becoming the new generation of robots powered by computer vision and artificial intelligence techniques trying to shorten the gap toward autonomous robots capable of adapting to the changes in the surrounding environment [4].

Task-oriented or Task and Motion Programming (TAMP) [5] is a framework that tries to abstract at a high-level the robot programming process and the improvements in computer vision and machine learning boosted the development. Anyway, all the aforementioned methods, even if they allow adapting to the target changes, are able to employ just the single policy encoded in the execution of a task, and they should be reprogrammed from scratch to satisfy other needs, which can be inefficient and tedious. In this context, imitation learning [6], i.e., the ability of the robot to acquire knowledge by imitating humans, can be a key feature since it can enable the operator to dynamically program the robot without leveraging on strong programming skills.

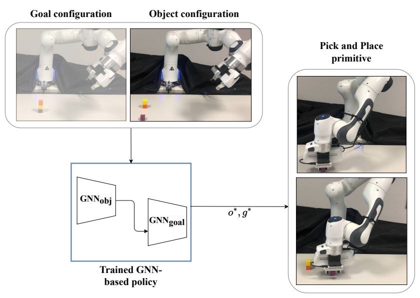

Fig. 1. Graphical representation of the proposed approach where a Graph Neural Network based policy abstracts the Pick-and-Place tasks as pairs of an object and a goal. Given a scene (goal configuration acquired at the beginning of the task and the current object configuration), the GNN policy plans at each time step which object o^* to move in which goal g^* to achieve the desired goal configuration. The GNN policy is composed of two GNNs: one focused on the goals and the other on the objects. The input is provided to the GNN through a perception module, while the decision taken by the GNN is executed by an external *PickAndPlace* primitive.

There exist three main approaches to imitation learning for robotics based on the way the knowledge is transferred to the robot [2]: kinesthetic teaching, teleoperation, and passive observation. The last one is the easiest for the operator since the human has to show the actions to be performed by the robot in a more human-like approach [7]. With regard to passive observations, a key characteristic would be to endow the robot with the ability to learn the rules of the task without the need to equip the human operator with on-body sensors [8]. However, the main problem is how to make the robot able to acquire knowledge about task rules for the planning process from the perception of the environment.

In this direction, the proposed work provides a framework based on Graph Neural Networks (GNNs) for Pick-and-Place tasks, exploiting the GNN's ability to capture relational inductive biases [9]. By abstracting the task to be executed, the GNN-based policy directly allows the robot to learn task-specific rules from abstract synthetic demonstrations in training and to one-shot generalize to unseen tasks in inference. Therefore, it is possible to consider a graph representation of the state space that encodes the task-relevant entities as nodes, in such a way the GNN-based policy could exploit

Manuscript received: March, 2, 2023; Revised June, 1, 2023; Accepted July, 12, 2023.

This paper was recommended for publication by Editor Hanna Kurniawati upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by Leonardo Company S.p.A. under grant No. LDO/CTI/P/0026995/21, July 2nd, 2021

[1]All the authors are with the Department of Excellence in Robotics & AI, Mechanical Intelligence Institute, Scuola Superiore Sant'Anna, Pisa, Italy firstname.lastname@santannapisa.it

Digital Object Identifier (DOI): see top of this page.

the intrinsic information present between the nodes of the graph for the acquisition of the task-specific rules. Basically, the GNN holds the role of orchestrating the steps for completing the manipulation task at a high-level, choosing which entities of the graph are relevant and should be passed to the externally encoded *PickAndPlace* motion primitives. The policy has been trained with a node-classification approach by looking at the different configurations of the objects and goals present in the scene. In Pick-and-Place tasks, especially in the industrial sector, recognizing the type of target objects is crucial. The developed GNN-based policy has been designed to take explicitly into consideration the types of objects present in the scene. Therefore, it is able to learn the association between objects to pick and goals positions for placing with respect to the object type, even for complex tasks like object stacking. Several experiments have been carried out to assess the performance of the proposed approach. The framework requires synthetic demonstrations to train the network, and the experimental results show a high generalization capability of the proposed model in terms of the number of objects/goals, objects/goals' stacks, heights distributions, planar position on the table, and configurations of objects/goals used during the training demonstrations.

The paper is organized as follows: Section II provides an overview of works that address the robotic manipulation problem; Section III presents the background on the Graph Neural Networks; Section IV shows the adopted methodology, while Section V illustrated the results of the experiments together with an ablation study of the architecture; Section VI and Section VII summarize the paper concluding the work and showing future research directions.

II. Related work

The Task And Motion Planning approach (TAMP) [5] for solving long-horizon manipulation tasks has proved to be a powerful tool in robotics, thanks to the combination of artificial intelligence methods applied to task planning level and robotics solutions to motion planning [10]. As shown in [11], the high-level reasoning ability, i.e., the capability of planning in very different situations, represents a key challenge. In particular, it is hard to deduce high-level planning in a world characterized by low-level sensing. One classical TAMP approach of common use relies on a set of pre-defined symbolic tasks by defining states, actions, and transition models that are given to symbolic planners [12]. Such kind of methods require a precise definition of the domain, and any modification in the environment requires a re-definition of the domain by an expert programmer. Another popular approach that is emerging in TAMP approaches is the integration of machine learning techniques inside planning [13].

Anyway, all the aforementioned methods, even if they allow adapting to the target changes, are able to employ just the single policy encoded in the execution of a task, and they should be re-programmed from scratch to satisfy other needs, which can be inefficient and tedious. In this context, Imitation Learning (IL) can be a key feature since it can enable the operator to dynamically program the robot without leveraging strong programming skills. However, the majority of works in Imitation Learning assume a close match between train and test environment [14] without the possibility of transferring knowledge to new situations. For this reason, One-Shot Imitation Learning (OSIL) tries to overcome the limitations imposed by classical IL approaches. The goal of OSIL, first formulated in [15], is to learn a policy that, given one demonstration of a new unseen task (w.r.t. the training set), is able to generalize and to act in the unseen instance of the task. In recent years, several works have addressed the problem of one-shot imitation learning by exploiting different techniques like Model-Agnostic Meta-Learning [16], Domain Adaptation [17], Transformers [18], or Graph Neural Networks [19]. The common denominator of these works is the use of visual input through video demonstrations. Instead, the proposed work focuses on the one-shot imitation learning problem from the planning manipulation point of view exploiting state space information of the scene. In practice, given a single demonstration of the desired goal configuration, the robot should be able to plan long-horizon manipulation sub-tasks, in a goal-conditioned manner.

Other works addressing the problem of OSIL try to learn modular task structures to be reused at inference time like [20] that proposes a graph approach to represent the action sequence of a task and [21] that formulates one-shot imitation learning as a symbolic planning problem. In this direction, [22] provides a combination of symbolic and geometric scene graphs to achieve symbolic goal specification, and [23] exploits GNNs to predict inter-objects relationships and a sequence of actions that can accomplish the specified symbolic goal. However, the outputs of these symbolic planners are highly abstracted in semantic concepts, thus assuming extensive domain knowledge and human-designed priors, and are not well-suited for those situations where a specific goal pose is necessary (like the industrial setting that is here considered).

Another approach based on GNN applied to the problem of robotic manipulation is [24], which exploits Graph Neural networks to forecast object motion in scenes in combination with Model Predictive Control but was demonstrated only in much simpler tasks and scenarios (pushing and falling) w.r.t. the cases considered in the proposed work. A recent work [25] uses Gated GNN for robotic grasping to understand the picking order of the objects, combining the state representation provided by the GNN with visual feature extractors without considering a goal configuration. In addition, it requires specific datasets to train the model in contrast to a purely synthetic dataset as the proposed work.

Differently from [26], which also exploited state space information to construct a GNN to tackle the long-horizon manipulation task, the proposed work focuses on the generalization ability of the proposed GNN approach to achieve one-shot imitation learning capabilities on more complex tasks and scenarios, even considering the types of the objects involved in the scene. In particular, the proposed approach is based on the combination of two different GNNs that, during training, directly learn task-specific rules and require only a single goal-configuration demo at inference time to accomplish the task. Considering the poses and types of objects involved in the scene drastically increases the task completion difficulty, especially in stacking tasks due to the presence of precedence constraints, i.e., putting the highest object in the lowest goal corresponding to that type. Respecting such constraints is compulsory in the industrial sector.

III. Background

Let $\mathcal{G} = (V, E)$ be a graph with V being the set of vertices and E the set of edges between nodes. Each node $v \in V$ has a feature vector $f(v)$ that contains the node's specific information. With message passing, neighboring nodes can exchange information encapsulated into feature vectors, obtaining updates of the feature vectors for each node in the graph. One message-passing layer consists of two main phases: transformation and aggregation. Each node's feature is updated through a transformation of the initial feature $f(v)$ and an aggregation of the feature of the neighboring nodes. Denoting with h_i^l the feature vector of node i at layer l, the general definition for message passing architectures is $h_i^{l+1} = \phi_\theta(h_i^l, h^l_{j\{j\in\mathcal{N}_i\}})$, where θ are parameters of the network that are optimized during training and $j \in \mathcal{N}_i$ are the neighbors of node i. In the beginning it results $h_i^0 = f(v)$.

Many different architectures exist, and most of them make different choices in the way they transform and aggregate information between nodes in the above general formulation. One key property of GNNs is the fact that, once the network is trained, the model's parameters can be shared across all nodes, giving them inductive learning capability. This means that it is possible to train the GNN on one graph, obtain parameters of the graph network, and then apply the same network to a new graph. In this work, two approaches are analyzed for training the graph neural network, GraphSAGE [27] and Graph Attention Networks [28].

GraphSAGE: Most existing approaches are inherently transductive since they require the presence of all nodes in a graph, and they do not intrinsically generalize to new unseen nodes. GraphSAGE takes explicitly into consideration this problem by constructing a model that, during training, given a node i, samples a set of neighbors $\mathcal{N}_i$ not evaluating all node's neighbors. Node's embeddings are computed by $h_i^{l+1} = \sigma\left(\theta_0 h_i^l + \frac{\theta_1}{|\mathcal{N}_i|}\sum_{j\in\mathcal{N}_i} h_j^l\right)$ where θ_0 and θ_1 are the parameters of the network, $\sum_{j\in\mathcal{N}_i}$ denotes the aggregation of the feature vectors of each neighbor j of node i, and σ denotes the nonlinear activation function.

Graph Attention Networks (GAT): This type of architecture takes into consideration how much the feature vector of node j is important for node i by using coefficients that weigh the degree of importance. Node's embeddings are computed by $h_i^{l+1} = \sigma\left(\theta_0 h_l^l + \sum_{j\in\mathcal{N}_i} a_{i,j}\theta_1 h_j^l\right)$ where θ_0 and θ_1 are the parameters of the network, $\sum_{j\in\mathcal{N}_i}$ denotes the aggregation of the feature vectors of each neighbor j of node i, σ denotes the nonlinear activation function, and a_i represents the attention coefficients, which are in turn computed by $a_{i,j} = \dfrac{\exp(a_{i,j})}{\sum_{k\in\mathcal{N}_i}(\exp a_{i,k})}$.

IV. Proposed approach

The proposed approach uses a GNN-based policy that abstracts the task of Pick-and-Place at a higher level. The policy is meant to find the relationships among objects involved in the scene that are the targets to be grasped and the goals corresponding to the targets' placement positions. Then, an external low-level *PickAndPlace* primitive takes as input the decision of the GNN-based policy π (representing the poses of chosen object and goal) and it is in charge of moving the robot to accomplish the task. In particular, the policy combines two GNNs that orchestrate the long-horizon manipulation task, choosing which object has to be located in which goal at each time step of the task execution. Indeed, the policy has been designed to take explicitly into consideration the type of objects present in the scene. At each time step t a fully-connected object-centric graph $\mathcal{G}$ of the scene s_t is constructed and it is evaluated by the GNN policy π.

A. Problem formulation

We consider goal-conditioned one-shot imitation learning as a supervised learning problem on a synthetic data set containing Pick-and-Place demonstrations. Each demonstration of the Pick-and-Place completion is characterized by state-action pairs (s_t, a_t) at each time step t. The state s_t is converted into a graph, and the action a_t represents the connection between the object o^* that must be moved in the correct goal g^* to achieve the new state s_{t+1} at the time step $t+1$. Each entire task completion $\mathcal{T}$ is composed of a set $\{(s_1, a_1), ..., (s_T, a_T)\}$ containing state-action pairs until the final time step T of the single manipulation task is reached. Policy π is trained on a dataset $D = \{\mathcal{T}_1, ..., \mathcal{T}_n\}$.

B. Method

Fig. 2 depicts graphically the proposed approach. From an image of the scene, a geometric graph is constructed. In the graph encoding step, the objects and the goals define the set of nodes. $\tilde{n}_o^\tau = \tilde{n}_g^\tau$, where $\tilde{n}_o^\tau$ represents the number of objects for each type τ and $\tilde{n}_g^\tau$ represents the number of goals for each type τ. This assumption implies that the total number of objects n_o is equal to the total number of goals n_g. The set of vertices is $V = \{v_l^o\}_{l=1}^{n_o} \cup \{v_i^g\}_{i=1}^{n_g}$ with cardinality $|V| = n_o + n_g = M$. The superscripts and subscripts o and g denote if the vertex refers respectively to an object or a goal.

The set of edges is $E = \{e_{i,j}\}$ with $i, j = 1, ..., n_o + n_g$. Each node $v \in V$ is characterized by a feature vector $f(v)$ containing the pose, the type, and structural information. From the geometric graph encoded scene $\mathcal{G}$, the policy π returns probability distributions over objects and goals. In the probability computation, the policy π has to capture the importance of structural relations among the entities and the importance of correct type association between object and goal. Denoting with $\{\tau_l^o\}_{l=1}^{N_o}$ the type of the object - where N_o represents the total number of objects' types - and with $\{\tau_i^g\}_{i=1}^{N_g}$ the type of the goal - where N_g represents the total number of goals' types - the study considers $N_o = N_g = N$. The total number of types is $N_o + N_g = 2N$. The policy π

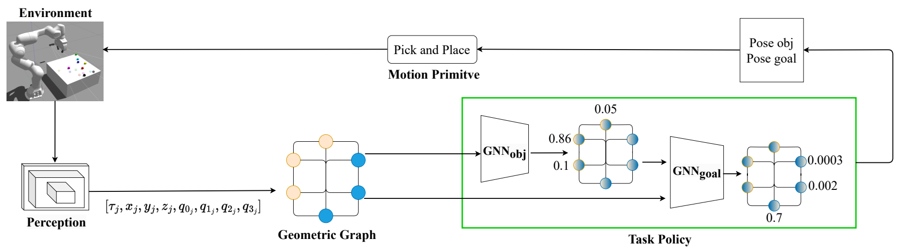

Fig. 2. Overview of the proposed approach. The scene is captured by the camera, which perceives objects and goals. The perception module has the role of providing information about the type (τ_j) and poses ($[x_j, y_j, z_j, q_{0_j}, q_{1_j}, q_{2_j}, q_{3_j}]$) of each object/goal j. The information is encoded in a geometric scene graph $\mathcal{G}$, where, in the illustration, yellow nodes represent objects, and cyan nodes represent goals. The GNN$_{obj}$ outputs probability distribution over objects. The objects' feature vectors are modified such that only the chosen object becomes reachable. Then GNN$_{goal}$ outputs a probability distribution over goals. Policy π chooses the object and the goal with the highest probability and extracts pose information from correspondent feature vectors. Finally, the poses are sent to the motion primitive, which picks the specified object pose and places it in the specified goal pose.

has to learn the mapping $\{\tau_j^o \rightarrow \tau_j^g\}_{j=1}^N$ during the execution. Pseudo-code of the algorithm is shown in Algorithm 1.

C. Feature encoding

At each time step, the graph is updated with the information coming from the scene. Each node of the graph has a feature vector $f(v) \in \mathbb{R}^{2N+9}$, where N are the types of objects and goals encoded as a categorical feature that is then converted in a one-hot encoded vector. Fig. 3 illustrates the information encapsulated in the feature vector: three values represent the x, y, z translational information in the world frame, four values represent the orientation information in the form of quaternion q_0, q_1, q_2, q_3, one feature encapsulates information about the reachability of the object/goal considered, and the last feature is a binary value, denoting if an object is inside a goal of the same type (1) or not (0), or if a goal is filled with an object of the same type (1) or not (0).

The reachability term indicates with the value of 1 if the considered entity can be manipulated by the low-level Pick-and-Place primitive being the highest free objects with the type corresponding to one of the types of one of the lowest free goals. It is worth noticing that such information is not restrictive since can be derived by the perception module that gives the input to the GNN.

D. Policy training

The policy π, which acts on $\mathcal{G}$, is composed of two GNNs. The first one (GNN$_{obj}$) takes in input the graph representation and outputs a probability distribution over the objects' nodes, while the second GNN (GNN$_{goal}$) takes as input both the graph representation of the state and the object selected by GNN$_{obj}$, i.e., the object node with the highest probability, and predicts a probability distribution over goals' nodes.

The *PickAndPlace* manipulation task is considered at each time-step as a binary classification problem: the GNN$_{obj}$ takes as input the feature vector $f(v)$ of each node $v \in V$ and, passing inside a sigmoidal function, outputs a choosing probability for each vertex. Then a value of 1 is assigned to the object that presents the highest probability and setting 0 to all the others, thus deciding which object has to be manipulated at each time step. In this way, each vertex in the graph results in a 1-dimensional binary vector. The GNN$_{goal}$ takes as input both the graph representation and the information about the chosen object, which is provided by modifying the reachability feature of the incoming graph representation's nodes setting all the reachability features of the objects to 0, except the one of the chosen object that is instead set to 1. Finally, the GNN$_{goal}$ outputs a probability distribution over the goals passing each vertex value inside the sigmoidal function.

Algorithm 1 GNN-based policy for manipulation task

```
1: for $t \leq T$ do
2:     $\mathcal{G} \leftarrow s_t$
3:     $p_o, p_g \leftarrow \pi(\mathcal{G})$
4:     $o^* = \arg\max p_o$
5:     $g^* = \arg\max p_g$
6:     Execute PickAndPlace with inputs $o^*$ and $g^*$
7: end for
```

The parameters of the two Graph Neural Networks are trained to minimize the binary cross-entropy loss. One target vector per GNN is used in the policy: $P_{y_{obj}}$ for GNN$_{obj}$ contains information about the correct object o^* and the correct goal g^*, while $P_{y_{goal}}$ for GNN$_{goal}$ is a modified version of the first one, where all reachable goals of the same object's type g_τ^* are set equal to 1. This gives rise to the target zero vector $P_{y_{obj}}$ having **1** where $[o_k = o^*, g_k = g^*]$ and to target zero vector $P_{y_{goal}}$ having **1** where $[o_k = o^*, g_k = g_\tau^*]$. This design choice allows GNN$_{obj}$ to choose the object on the basis of structural information encoded in the graph, and GNN$_{goal}$ to focus on those goals matching the type of first GNN's choice.

Given P_{y_k} the target label for each node $k = 1, ..., M$ in the graph and p_k the value of probability assigned in output by the GNN to the $k-th$ node, the loss function L to be minimized is:

$$L = \frac{1}{M} \sum_{k=1}^{M} -P_{y_k} \log(p_k) - (1 - P_{y_k}) \log(1 - p_k) \qquad (1)$$

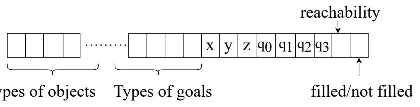

Fig. 3. Schematic view of $f(v) \in \mathbb{R}^{2N+9}$. The first N-dimensional one-hot encoding vector refers to the *objects* while the second one concerns the *goals*. The last 9 encompass the world coordinates x, y, z, the quaternion q_0, q_1, q_2, q_3, one binary element to indicate if the actual *object/goal* is reachable and another binary element that states if the *object* has been placed in the corresponding *goal*.

The loss function as the Eq. 1 is computed both for GNN$_{obj}$ as L_{obj} and GNN$_{goal}$ as L_{goal}, respectively using the target vector $P_{y_{obj}}$ and $P_{y_{goal}}$. The output probability over objects' nodes computed by GNN$_{obj}$ is then concatenated with the output probability computed over goals' nodes by GNN$_{goal}$, obtaining a unique tensor of probabilities denoted by p of length M. Given p_o the vector of choice probabilities for n_o objects and p_g the vector choosing probabilities for n_g goals, the chosen object is computed as $o^* = \arg\max p_o$, and similarly the chosen goal is computed as $g^* = \arg\max p_g$.

The parameters of the two GNNs are meant to maximize the generalization behavior: indeed, a binary cross-entropy loss function has been adopted for each node, differently from the categorical cross-entropy loss used by [26] to decouple the single GNN's prediction made for each node from the full state's predictions. This allowed the proposed approach scaling up over an increasing number of nodes.

The target data used to train the networks are collected by assigning a label to each node in each graph sampled from the collected dataset of Pick-and-Place demonstrations D. The two networks are trained separately, and, in the case of GNN$_{goal}$, the information about the chosen object is taken at training time from the collected objects' labels.

V. Experiments

We use synthetic data to train and test the policy π by generating different training $\mathcal{T}_{train}$ and testing tasks $\mathcal{T}_{test}$.

Once trained with synthetic data, the policy can be transferred to simulation environments or real-hardware platforms without further modifications. For simulation, we employed the ROS-Gazebo physical simulator, while we used the Franka Emika Panda as the robotic arm equipped with a parallel-jaw gripper. Objects and goals considered in the experiments are cubes whose color determines their type. In simulations, goals are represented with semi-transparent cubes of the same color as the objects. In real scenarios, a RealSense D435i RGB-D camera has been used.

Since the proposed framework is modular, the perception and motion execution parts (respectively responsible for providing the necessary information to the GNN and for grasping the target) can be different, depending on the user's needs. In particular, the perception module should provide the types and poses (translation + orientation) of the objects. The motion execution module, instead, should be able to synthesize a feasible grasp for the target object. For the real hardware experiments, we leveraged a category 6D pose estimator [29] to obtain the pose of each object/goal along with a simple computer vision component that determines the type of the objects through the HSV color map filtering. Instead, for the execution module, i.e., the Pick and Place primitive, we exploited the Moveit! framework in ROS.

It is worth noticing that in real hardware, for the sake of simplicity, the orientation of the objects has been ignored since it would have added only complexity in the grasping phase without affecting the GNN decision-making behavior. Experiments have been designed such that both GAT and SAGE models employ MLP-based mapping in the message-passing layers with the Relu activation function and 100 neurons for each layer.

A. Synthetic data generation and tasks

In this work, an instance of the task is designed to be a sequence of Pick-and-Place actions from the starting configuration of objects and goals until all the goals have been filled. Task instances can differ from each other depending on the *variables* of the task instance itself: the number of types, objects, goals (here we assume equal to the number of objects), objects' stacks, goals' stacks, position in the $x-y$ plane, and height distribution of both objects and goals stacks.

It is worth clarifying some nomenclature that can help in understanding the following description. Objects can be stacked in multiple stacks. One *highest free object* is an object that is the highest item in its stack. One *lowest free goa*l is a goal in the lowest position in its stack. The main rule underlying the task that GNN-policy has to learn is to choose at each time step one highest free object and to place it in one lowest free goal of the same type. From this characterization, it appears clear that there is a strong dependency constraint on types among objects and goals that must be respected to accomplish the whole task correctly. Indeed, it is not guaranteed that the *absolute highest object* (i.e., the highest object considering all the stacks) is the one that satisfies the type constraint for the available lowest free goals at each step of a task instance.

The dataset $D = \{\mathcal{T}_1, ..., \mathcal{T}_n\}$ used for training has been generated synthetically. For each $\mathcal{T}_i$, once all the *variables* have been set up, the synthetic code provides a generation of feature vectors like the one in Fig. 3 and the correct update for each state-action pair belonging to $\mathcal{T}_i$. Each object/goal has been considered as a cube of length $0.02m$ and has a position on the plane randomly generated with $x \in (0.3m, 0.7m)$ and $y \in (-0.2m, 0.2m)$, leaving out all positions that were inside a circle of radius $0.1m$ and center in $(0.5m, 0.0m)$ to be used in the testing phase. The height of the $x-y$ plane has been considered starting at $0.2m$. The number of types present in each $\mathcal{T}_i$ varies in $[1, n_{type\_max}]$ between demonstrations.

In order to evaluate the performance of the policy, a different set of variables has been provided at testing time. All test experiments have been considered to evaluate the generalization performance with respect to variables between $\mathcal{T}_{train}$ and $\mathcal{T}_{test}$, except for the higher number of types, for whose generalization, π was not designed.

B. Performance evaluation

In order to choose the hyperparameters of the networks to perform the validation of the proposed approach, we perform a greedy search varying the network model (GAT and GraphSAGE), the number of message-passing layers inside the model, and the number of demonstrations. From experimental training results, it has been observed that the SAGE model is always able to converge to good training accuracies. Given the same training conditions, the same does not happen when the employed architecture is GAT, failing to reach the training

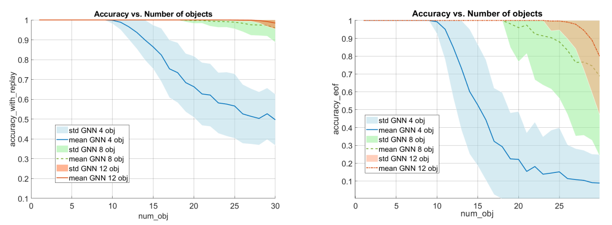

Fig. 4. Results of accuracy with replay (left) and accuracy with end-of-episode (right) w.r.t. the varying number of objects, showing three models trained on four types and a maximum of four, eight, and twelve objects respectively, as reported by the legend. The x-axis of the plots represents the number of objects used to test the generalization performance of the models, performing 300 runs per num_obj. The number of stacks for both objects and goals randomly varies across runs to obtain diverse configurations. The graphics show the mean value of accuracies over the 300 runs. The three models are able to generalize to a number of objects higher than the one presented at training time.

accuracies achieved with the SAGE model. For this reason, in the following training, the SAGE model has been used.

During training, three different variables of the task instance concerning the number of objects have been considered generating a scenario with 4, 8, and 12 maximum number of objects, respectively. In each scenario, the number of objects *per type* and the number of objects for each demonstration within the maximum value have been randomly generated. The number of stacks for both objects and goals varies in the range of [1-4] in each run to obtain diverse configurations. The accuracy has been computed as the total number of correct classifications (i.e., the correct association at each time step between the *highest free object* into the *lowest free goal* of the *same type*) over the total number of predictions. It is worth noticing that the accuracy is computed following two types of metrics that spot two different salient aspects:

- *accuracy with replay* for which if the policy π selects an object or a goal that is not reachable, or the type of object and the goal does not match, the choice is considered incorrect, but the state at the subsequent time step is shown with the correct object placed in the right goal to continue the demonstration till the end;
- *accuracy with end-of-episode* for which, if the policy's choice is wrong, the $\mathcal{T}_{test}$ is stopped, and the remaining choices to complete the task instance are all considered incorrect for the computation.

In both cases, (unfeasible) situations, in which the policy is unable to choose a reachable object matching the type of one of the free lowest goals, are not considered in the computation. The first metric evaluates the correctness of the policy's choices independently of the impact it might have on the continuation of the task. The second one considers that an error in the policy automatically makes it unfeasible to complete the task as the number of objects per type and the number of goals for the corresponding type is equal. The latter metric is more stringent and best suited for industrial tasks where for example, it is relevant that all the objects of a given type are located in the right place, and an error in a situation where there are precedence constraints dictated by the type of the objects can jeopardize the whole task. The complexity of these kinds of tasks, where the different numbers of stacks and the different numbers of objects *per type* are present, can lead

TABLE I
ACCURACY WITH REPLAY FOR HYPERPARAMETERS' CONFIGURATION. MODELS TRAINED ON MAX. 4, 8, AND 12 OBJECTS. EACH VALUE COMPUTED (ON 18 OBJECTS, VARYING THE NUMBER OF STACKS) AS THE MEAN ACCURACY OVER 300 SEPARATE TEST RUNS.

N. Demos	**2 M.P.L.**			**4 M.P.L.**			**8 M.P.L.**		
	4	8	12	4	8	12	4	8	12
20	0.56	0.52	0.47	0.56	0.47	0.52	0.53	0.49	0.48
300	0.61	0.70	0.51	0.61	0.79	0.92	0.65	0.80	0.97
1000	0.49	0.76	0.93	0.70	1.0	1.0	0.76	1.0	1.0

to particular situations in which the choice of a certain object could unlock a feasible path (since another object can become reachable), otherwise, a deadlock can occur (since the lowest free goal's type does not match the highest free objects' type forcing the policy to fail).

For the hyperparameters' configuration, we consider only the accuracy with replay since we want to find the more appropriate training setup, while for generalization performance evaluation, we consider both accuracies. In Table I, the results of the different hyperparameters' configurations are reported. Once the model is fixed, each accuracy value in the table is computed as the mean value of accuracies over a total number of 300 runs. For each model, we consider the same $\mathcal{T}_{test}$, with 18 objects of 4 types varying the number of objects/goals' stacks between each demonstration. The number of objects *per type* in each demo has been randomly generated.

From the performance measures, it is clear that a given number of demonstrations and a suitable architecture (in terms of message-passing layers) are needed to obtain a good generalization. Indeed few demonstrations are not sufficient to obtain good results independently from the number of training objects and message-passing layers. However, by increasing the number of demonstrations, also the accuracies increase in proportion to the number of message-passing layers and the number of training objects. The maximum accuracy is obtained with 4 or 8 message-passing layers and 1000 demonstrations using either a maximum of 8 or 12 objects during the demonstrations. Fig. 4 shows the generalization performances of the 3 models trained with 8 message-passing layers and 1000 demonstrations with respect to the varying number of objects. The way the accuracy measure is computed on $\mathcal{T}_{test}$ is equal to the one used for Table I. The required number of training demonstrations is also strictly related to the number of types. As depicted in Table II, when the number of types increases, the number of training demonstrations required to have a high accuracy increases as well. Table II also confirms that the policy's generalization performance can deteriorate with the decrease in the number of training demonstrations.

C. Baseline comparison

We compared our approach against IL-GNN [26] and two RL baselines (RL-GNN and RL-GNN-Seq) on block stack-

TABLE II
MEAN AND STANDARD DEVIATION RESULTS ON 18 OBJECTS TEST TASK AND VARYING STACKS OVER 300 RUNS PER N. TYPES WITH 600 AND 1000 TRAINING DEMONSTRATIONS.

N. training demos		**N. types**	**3**	**4**	**5**	**6**	**7**	**8**
	600	Acc. with replay	1.0 ± 0.0	0.99 ± 0.01	0.99 ± 0.03	0.89 ± 0.10	0.87 ± 0.1	0.11 ± 0.12
		Acc. end-of-ep.	1.0 ± 0.0	0.97 ± 0.16	0.91 ± 0.28	0.40 ± 0.4	0.34 ± 0.37	0.04 ± 0.12
	1000	Acc. with replay	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.003	0.99 ± 0.014	0.87 ± 0.11
		Acc. end-of-ep.	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.98 ± 0.06	0.95 ± 0.21	0.34 ± 0.3

TABLE III
COMPARISON RESULTS BETWEEN IL-GNN [26] AND OUR APPROACH.

	6-Pyramid	**3-block 3-stack**	**40-Block Stacking**
IL-GNN	1.0 ± 0.0	1.0 ± 0.0	$0.75 \pm$ n.a.
RL-GNN	$0.2 \pm$ n.a	$0.13 \pm$ n.a	n.a
RL-GNN-Seq	$0.19 \pm$ n.a	$0.14 \pm$ n.a	n.a.
Ours	1.0 ± 0.0	1.0 ± 0.0	$\mathbf{1.0} \pm 0.0$

ing and pyramid scenarios. These RL baselines have been designed and trained in [26] to highlight the generalization abilities of GNN policy over RL.

IL-GNN uses a single geometric Graph Neural Network to predict at each time step which object and which goal have to be chosen (using information encoding similar to ours). The model is trained by collecting expert demonstrations of bin packing and unpacking (opening a box cover, displacing the cubes inside or outside the box, and finally closing the box) with a Cross-entropy Loss minimization and considering only two types of objects (blocks and cover). Therefore, cubes are all considered to be of the same type regardless of their color, resulting in a simpler scenario w.r.t. the experimental tasks addressed by the proposed approach in which the type is considered for each cube entity.

RL-GNN is IL-GNN trained using RL on stacks of size 2 to 9, and RL-GNN-Seq is IL-GNN trained using sequential training curriculum described in [30]. For such a comparison, we replicated the same conditions of the tests performed in [26], training our policy on a set of 20 task instances consisting of a maximum of 4 objects of 1 type, with a number of stacks ranging from 1 to 3. In particular, we compare the proposed method against the 6-Pyramid having a pyramid-like goal configuration with 6 cubes in total and 3 layers, and block stacking, i.e., size-variable stacks of objects and goals. The latter is characterized by 3-block 3-stack, and 40-block stacking configurations.

In Table III, we show that our method can outperform the IL-GNN baseline approach in the generalization behavior considering one type. The improvement in performance can be appreciated most for the 40-Block stacking scenario in which we reach the maximum accuracy. For the two RL baseline's performances on 6-pyramid and 3-block 3-stack scenarios, RL-GNN reaches 0.2 and 0.13, respectively, while RL-GNN-Seq obtains 0.19 and 0.14. Indeed, in [26], RL methods on block stacking tasks have already been shown to have much lower performance than the baseline, demonstrating the advantage of the GNN-based approach in such tasks.

D. Architecture ablation

By removing the reachability information from the feature vector, the networks should still be able to learn the different structural importance of the cubes to be manipulated from the z-value during training time, but this adds more complexity to the training process and affects the robustness of the policy. Indeed, the reachability factor helps the networks learn the correct structural behavior and generalize to height distributions not seen during training, giving much more robustness to real-world applications. The model trained with a maximum of 12 objects, 4 types, 1000 demonstrations, and 8 message-passing layers without the reachability features achieved a mean accuracy with replay of 0.18 and 0.06 with a std of 0.12 and 0.06 on test tasks presenting 4 and 8 objects respectively.

In addition, the use of the two GNNs, one after the other, is fundamental for the successful completion of the task. Training experiments have shown that the type matching rule is not respected most of the time if only GNN$_{obj}$ or only GNN$_{goal}$ is employed, as it is not able to predict both the object and goal to be manipulated at the same time. The same model used in the reachability analysis has been evaluated by employing one GNN at a time on testing tasks with 18 objects. The GNN_obj achieved a mean accuracy with replay of 0.49 with a std of 0.18, and the GNN_goal achieved a mean accuracy with replay 0.75 with a std of 0.16, thus showing performance that is lower than the one obtained with the cascade of the two GNNs.

E. Hardware results

Once the GNN policy has been trained on synthetic data, it can be directly deployed on robot hardware without any modifications. In real-world experiments, the information about the scene is obtained from the RGB-D images provided by the camera, which is mounted on the wrist of the robotic arm. The perception module makes use of the i2c-net [29] opportunely trained on a synthetic dataset containing equally-sized cubes, rendered through pose, background, and texture randomization and an additional computer vision component that detects the type of the object through the HSV color map filter. The experiments require an operator to show the desired goal configuration, which is captured by the camera and memorized. Then objects are randomly displaced in the scene. The trained GNN-based policy, given the information of the scene, decides which of the cubes has to be placed in the corresponding goal configuration. Finally, the external high-level *PickAndPlace* primitive executes the task exploiting the Moveit! framework. The employed model has been trained on 5 types with 1000 training demonstrations and 12 objects.

It is worth noticing that the cubes employed in real scenarios are bigger (length equal to $0.025m$) than the ones used in the simulation, and the height distribution is different from the one seen at training time. A total number of 30 $\mathcal{T}_{test}$ runs have been conducted by varying the number of objects, objects/goals' configurations, and the number of objects per type, achieving a mean accuracy with replay and with end-of-episode equal to 1.0 with 0.0 standard deviation in both cases. It is not required that all the types should be present in each demonstration.

The real-hardware performance results show to be consistent with synthetically computed ones since the policy is not related to the perception modules. In addition, the policy has proved to be robust to cases in which the stacks or cubes can fall during robot execution due to inaccurate perception values or robot grasping because the new state is perceived again as being able to continue the task until all goals are filled.

VI. DISCUSSION

We presented an approach based on Graph Neural Network for one-shot imitation learning in the robotic manipulation domain. As shown by experimental results, the method is able to generalize to different aspects of the tasks: position in the $x - y$ plane, numbers, and height distribution of stacks, and the number of objects. For the latter, the diagrams in Fig. 4 show that the policy is able to generalize to a total number of objects quite far from the one seen during training.

The modular structure of the proposed approach is a key advantage allowing to exploit the perception and grasp module that best fits the task to be executed since the GNN policy has the capability of abstracting the underlying rule of the task and being agnostic to the specific objects involved in the scene. Being modular the perception module can be substituted following the progress in the computer vision field as far as it provides the translation and quaternion information of the objects. Therefore, errors coming from the perception module reflected in wrong decisions of the GNN policy could not be ascribed to the proposed method.

It is worth noticing that, even if the GNN policy has not been designed to generalize over a number of types higher than the one employed in training, the policy can handle lower or equal numbers of types present at inference time. This happens because the number of types in the training dataset varies between demonstrations.

VII. Conclusions

In this paper, a goal-oriented graph policy architecture for one-shot imitation learning of Pick-and-Place manipulation tasks is presented. The policy learns the underlying rules of the task by synthetically generated demonstrations and shows good generalization behavior, and can be transferred in real situations without further modifications. In particular, the proposed framework is modular, and the GNN component takes the required information from a perception module and outputs the decision to a motion execution module. The GNN component actually exploits two GNNs, one for the goals and the other for the objects involved in the scene. Future works consist in extending the decision-making policy to cluttered scenarios and expanding the capabilities of the policy to multiple actions like push/pull or open/close rather than only Pick-and-Place to make the approach more versatile.

References

[1] Z. Pan, J. Polden, N. Larkin, S. Van Duin, and J. Norrish, "Recent progress on programming methods for industrial robots," *Robotics and Computer-Integrated Manufacturing*, vol. 28, no. 2, pp. 87–94, 2012.

[2] H. Ravichandar, A. Polydoros, S. Chernova, and A. Billard, "Recent advances in robot learning from demonstration," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, 05 2020.

[3] E. Oztemel and S. Gursev, "Literature review of industry 4.0 and related technologies," *Journal of intelligent manufacturing*, vol. 31, pp. 127–182, 2020.

[4] S. D'Avella, C. A. Avizzano, and P. Tripicchio, "Ros-industrial based robotic cell for industry 4.0: Eye-in-hand stereo camera and visual servoing for flexible, fast, and accurate picking and hooking in the production line," *Robotics and Computer-Integrated Manufacturing*, vol. 80, p. 102453, 2023.

[5] K. Zhang, E. Lucet, J. A. D. Sandretto, S. Kchir, and D. Filliat, "Task and motion planning methods: applications and limitations," in *19th International Conference on Informatics in Control, Automation and Robotics ICINCO 2022)*. SCITEPRESS-Science and Technology Publications, 2022, pp. 476–483.

[6] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters *et al.*, "An algorithmic perspective on imitation learning," *Foundations and Trends® in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018.

[7] V. Villani, F. Pini, F. Leali, and C. Secchi, "Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications," *Mechatronics*, vol. 55, pp. 248–266, 2018.

[8] R. Skoviera, K. Stépánová, M. Tesar, G. Sejnova, J. Sedlár, M. Vavrecka, R. Babuska, and J. Sivic, "Teaching robots to imitate a human with no on-teacher sensors. what are the key challenges?" *CoRR*, vol. abs/1901.08335, 2019.

[9] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, and A. Santoro, "Relational inductive biases, deep learning, and graph networks," *CoRR*, vol. abs/1806.01261, 2018.

[10] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *CoRR*, vol. abs/2010.01083, 2020.

[11] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, "From skills to symbols: Learning symbolic representations for abstract high-level planning," *J. Artif. Int. Res.*, vol. 61, no. 1, p. 215–289, 2018.

[12] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Ffrob: Leveraging symbolic planning for efficient task and motion planning," *CoRR*, vol. abs/1608.01335, 2016.

[13] M. Mansouri, F. Pecora, and P. Schüller, "Combining task and motion planning: Challenges and guidelines," *Frontiers in Robotics and AI*, p. 133, 2021.

[14] M. Zhao, F. Liu, K. Lee, and P. Abbeel, "Towards more generalizable one-shot visual imitation learning," *CoRR*, vol. abs/2110.13423, 2021.

[15] Y. Duan, M. Andrychowicz, B. C. Stadie, J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba, "One-shot imitation learning," *CoRR*, vol. abs/1703.07326, 2017.

[16] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine, "One-shot visual imitation learning via meta-learning," *CoRR*, vol. abs/1709.04905, 2017.

[17] T. Yu, C. Finn, A. Xie, S. Dasari, T. Zhang, P. Abbeel, and S. Levine, "One-shot imitation from observing humans via domain-adaptive meta-learning," *CoRR*, vol. abs/1802.01557, 2018.

[18] S. Dasari and A. Gupta, "Transformers for one-shot visual imitation," *CoRR*, vol. abs/2011.05970, 2020.

[19] M. Sieb, X. Zhou, A. Huang, O. Kroemer, and K. Fragkiadaki, "Graph-structured visual imitation," *CoRR*, 2019.

[20] D.-A. Huang, S. Nair, D. Xu, Y. Zhu, A. Garg, L. Fei-Fei, S. Savarese, and J. C. Niebles, "Neural task graphs: Generalizing to unseen tasks from a single video demonstration," 06 2019, pp. 8557–8566.

[21] D. Huang, D. Xu, Y. Zhu, A. Garg, S. Savarese, L. Fei-Fei, and J. C. Niebles, "Continuous relaxation of symbolic planner for one-shot imitation learning," *CoRR*, vol. abs/1908.06769, 2019.

[22] Y. Zhu, J. Tremblay, S. Birchfield, and Y. Zhu, "Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs," *CoRR*, vol. abs/2012.07277, 2020.

[23] Y. Huang, A. Conkey, and T. Hermans, "Planning for multi-object manipulation with graph neural network relational classifiers," 2022.

[24] H. F. Tung, X. Zhou, M. Prabhudesai, S. Lal, and K. Fragkiadaki, "3d-oes: Viewpoint-invariant object-factorized environment simulators," *CoRR*, vol. abs/2011.06464, 2020.

[25] M. Ding, Y. Liu, C. Yang, and X. Lan, "Visual manipulation relationship detection based on gated graph neural network for robotic grasping," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 1404–1410.

[26] Y. Lin, A. S. Wang, E. Undersander, and A. Rai, "Efficient and interpretable robot manipulation with graph neural networks," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2740–2747, 2022.

[27] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *CoRR*, vol. abs/1706.02216, 2017.

[28] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[29] A. Remus, S. D'Avella, F. D. Felice, P. Tripicchio, and C. A. Avizzano, "i2c-net: Using instance-level neural networks for monocular category-level 6d pose estimation," *IEEE Robotics and Automation Letters*, pp. 1–8, 2023.

[30] R. Li, A. Jabri, T. Darrell, and P. Agrawal, "Towards practical multi-object manipulation using relational reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 4051–4058.