A learning from demonstration framework for adaptive task and motion planning in varying package-to-order scenarios

MA, Ruidong <http://orcid.org/0000-0002-8035-5746>, CHEN, Jingyu and OYEKAN, John

Robotics and Computer–Integrated Manufacturing 82 (2023) 102539

Contents lists available at ScienceDirect

Robotics and Computer-Integrated Manufacturing

journal homepage: www.elsevier.com/locate/rcim

A learning from demonstration framework for adaptive task and motion planning in varying package-to-order scenarios

Ruidong Ma [a,*], Jingyu Chen [a], John Oyekan [a,b]

[a] Department of Automatic and Control System Engineering, University of Sheffield, Mapping Street, Sheffield S1 3JD, United Kingdom
[b] Human-Centred AI for Manufacturing, Institute of Safe Autonomy, Department of Computer Science, University of York, Heslington, York YO10 5GH, United Kingdom

ARTICLE INFO

Keywords:
Task and motion planning
Graph neural network
Learning from demonstration
Package-to-Order

ABSTRACT

Current advances in Task and Motion Planning (TAMP) framework often rely on a specific and static task structure. A task structure is a sequence of how work pieces should be manipulated towards achieving a goal. Such systems can be problematic when task structures change as a result of human performance during human-robot collaboration scenarios in manufacturing or when redundant objects are present in the workspace, for example, during a Package-To- Order scenario with the same object type fulfilling different package configurations. In this paper, we propose a novel integrated TAMP framework that supports learning from human demonstrations while tackling variations in object positions and product configurations during massive-Package-To-Order (mPTO) scenarios in manufacturing as well as during human-robot collaboration scenarios. We design and apply a Graph Neural Network(GNN) based high-level **reasoning module** that is capable of handling variant goal configurations and can generalize to different task structures. Moreover, we also built a two-level **motion module** which can produce flexible and collision-free trajectories based on important features and task labels produced by the **reasoning module**. Through simulations and physical experiments, we show that our framework holds several advantages when compared with state-of-the-art previous work. The advantages include sample-efficiency and generalizability to unseen goal configurations as well as task structures.

1. Introduction

The increasing demand for personalized products has caused more and more manufacturing enterprises to apply a Make-To-Order (MTO) production strategy. Such an approach usually prepares inventory in advance and performs the final production and packaging when the customer orders are placed [1]. There are other variant types of MTO, such as Assemble-To-Order (ATO), Configure-To-Order (CTO), and Package-To-Order (PTO). Sometimes, a packaging company has to configure a batch (e.g. 500 - 10,000) of products in a certain way for a customer before switching to making a batch of orders in another configuration. An example of this scenario is packaging companies that assemble hamper baskets for their employees during festive periods. We term this massive Package-To-Order (mPTO). In this study, we focus on the mPTO scenario where the product types are common across different customers, while the final packaged product is determined by customer specification (Fig. 1). In this case, if the packaging process relies only on a human worker, several human factors such as fatigue, boredom from repetition and repetitive strain injury may introduce errors into the packaged product hence impacting the quality of the final products and the productivity of the manufacturing line.

One way of solving this is to automate the line using industrial robots. However, this requires heavy investment by these companies which are often small to medium-sized enterprises and cash-strapped. Furthermore, this approach does not lend itself to flexibility and due to the continuous changes in customer specifications, automation solutions would need to be reprogrammed often thereby increasing costs further. Also, there are still some tasks where human dexterity is still required.

The application of collaborative robots (cobot) is one way of solving the above challenge. These cobots can work in the same vicinity as humans and are more cost-effective. Their modularity means that they can be flexibly moved from one workbench to another. However, the issue of programming the cobot still remains. Current approaches make use of classical teach-in methods such as lead-through or kinesthetic teaching that directly feed the fixed poses and paths to the cobot.

An adaptive TAMP framework with Cobot for varying mPTO problem

* Corresponding author.
E-mail addresses: rma17@sheffield.ac.uk (R. Ma), jchen118@sheffield.ac.uk (J. Chen), j.oyekan@sheffield.ac.uk, john.oyekan@york.ac.uk (J. Oyekan).

https://doi.org/10.1016/j.rcim.2023.102539
Received 26 August 2022; Received in revised form 6 December 2022; Accepted 24 January 2023
Available online 2 February 2023

Fig. 1. A real-world example of a package-to-order from Amazon in which the seller often needs to package a variety of products with different goal configurations according to different customer demands.

However, these are not practical in such a flexible production problem [2] because these training methods are static and cannot deal with variations in the environment. In this paper, we want to address a couple of variations in the mPTO scenario.

The first variation exists in goal configurations. In this variation, we address various customer orders such as that shown in Fig. 1. Different customers can request different products from the inventory. In order to properly package them according to the customer's specifications, it is required to assign each selected product with a specific goal position. In this scenario, due to the different requirements of customers, there will be different packaging configurations and appearances derived using the same products. This means that the same product may occupy different goal positions. The second variation of mPTO is caused by situations in which humans flexibly collaborate with the cobot. When humans collaborate with cobots, different humans might decide to use different sequences to complete a task. In such a flexible production problem, the cobot needs to acquire knowledge not only about the motions necessary to complete the task but also about the task structure. The task structure in this paper refers to the sequence of manipulation of the related products making up the final goal configuration. For example, the cobot would pick and place the first product followed by the next one until the goal configuration is reached. In an unbounded environment, the task structure could change due to disturbances (e.g. a worker placing required objects in a different sequence).

For dealing with the above specified mPTO problem and the challenge of adapting to variations, we define five main aspects that are required from the cobot:

a) The cobot should have knowledge of the task's goal configuration.
b) From a scene containing a number of observed objects, the cobot should be able to identify the appropriate objects needed to complete the task.
c) The cobot should be able to schedule the necessary sequence to complete a task.
d) The cobot should be able to complete the sequence if it has been started by a human. In this case, if the human has started packaging some of the products randomly, the cobot should be to observe and understand what has taken place and then manipulate the rest of the objects in response to the human's performance. We term this as responsive Human -Robot-Collaboration (HRC) according to the International Federation of Robotics (IFC) [3]
e) The motions generated by the cobot should be able to avoid collisions with objects in the workspace.

To deal with the aforementioned variations in mPTO scenarios, traditional approaches often require a detailed manual design that takes into account the possible variations that can happen. For example, planning with manipulation trees requires a list of all possible product combinations [4] and logic operations are conditioned on every possible action [5].

In contrast to traditional approaches, we apply Learning from Demonstrations (LfD) in which we unintrusively demonstrate the required goal configuration (final product positions in the package) and multi-task manipulations (task structure) to the cobot. By unintrusive, we mean that we do not use kinesthetic teaching but rely on vision via passive observation from the demonstration. After passive observation of the demonstrated task by the cobot, we leave the cobot to learn the underlying task structures.

We also provide demonstrated motions that are required to complete tasks by learning the planning strategy. It is important to produce adaptive motions that ensure objects' safety

In addition to the above, we make use of the Task and Motion Planning (TAMP) framework that can integrate discrete high-level decision-making and continuous motion generations. It enables the cobot to deal with variations in the object positions in the environment as well as work collaboratively with humans when they decide to use different sequences to complete a task. Our main contributions in this paper are as follows:

a) We integrate different machine learning-based algorithms to form a Task and Motion Planning (TAMP) architecture that can handle long horizon mPTO tasks (i.e. multiple products packaging tasks) after small amounts of demonstrations.
b) We apply a high-level **reasoning module** with a Graph Neural Network (GNN) in its architecture to enable the handling of variant goal configurations and directly learn the demonstrated task structure with raw observations while generalizing to different task structures caused by variation in human performance. This is

different from [6] where their approach could not deal with variations in the sequences of objects required to complete a task.

c) We propose a **motion module** that can learn and achieve a safe planning strategy from demonstrations. By safe, we mean adaptive collision-free motions for object handling during different stages including pick and place.

d) Our trained architecture can be easily adapted from simulation to the physical world without any further training.

The rest of this paper is organized as follows: in Section 2, we present and discuss related work while in Section 3, we present a detailed design of our architecture. In Section 4, we present the results of experiments conducted in both simulation and the physical world while in Section 5, we conclude and discuss the future directions of our work.

2. Related work

As discussed in the introduction section, we designed, developed and applied a novel TAMP approach to program a cobot towards achieving task flexibility as well as to deal with variations in the environment. In this section, we discuss current advances in high-level decision making, low-level motion planning techniques as well as integrated Task and Motion Planning frameworks that researchers have applied to deal with multi-task scenarios.

2.1. Plan learning from demonstrations

Plan learning from demonstrations refers to techniques that are capable of learning task-level abstractions [7]. Traditional planning approaches often require predefined symbolic rules that can recursively solve a domain-specific problem. For example, in [4], task-level abstractions were defined using the hierarchical properties of a task. In their work, they were able to form a planning and execution tree for dynamic planning. [8] adopted Answer Set Programming (ASP) for checking the feasibility of different levels of task and motion planning.

Planning Domain Definition Language (PDDL) is another traditional approach that can be used. In [5], an action- centered planner with symbolic description and logical formulations were used to describe the effect and applicability of actions. PDDL was extended in [9] to achieve temporal planning and [10] extended it into hierarchical planning problems. Hierarchical Task Network (HTN) was utilized in [11]. They combined symbolic and geometric planners to achieve task and motion planning. This required constraints and domain expert knowledge to better understand the geometric side effect produced by the symbolic planner.

A Markov Decision Process (MDP) was adopted in [1] for production and inventory control in the MTO problem, while [12] also studied the MTO problem by using Group-based scheduling with the tabu search algorithm. [13] utilized AND/OR graph to represent all possible assemble plans in multi-object manipulation tasks while [14] combined it with reinforcement learning to achieve the flexible assembly of a single product.

The aforementioned traditional approaches generally do not need any training process. However, they need hand- coded task descriptions, task transition models (i,e task structure) or constraints, for example, if-else logic to condition possible actions. Thus, these approaches often require domain experts to design the rules while considering all possible situations.

Current advances in Learning from Demonstrations (LfD) have raised the opportunity for automatically learning the tasks without any hand-coding process. In [15], task-level abstraction was achieved using natural language to represent action symbols. These action symbols were obtained from sub-tasks in video demonstrations with Sequence-to-Sequence model. However, how they integrated their action symbols with position-based motion plans is not clear. Researchers are also increasingly using Graph Neural Network (GNN) [16] for task-level abstraction. This is because of its natural capability to encode graphical relationships between objects or multi-tasks. Some work focus on encoding the symbolic task descriptions as graph representations. For example, [17] utilized Conjugate Task Graph (CTG) to form sequences of sub-tasks while [18] built manipulation graphs containing sets of motor primitives to perform manipulation tasks. In [19], they propose a Neural Task Graph (NTG) by using LSTM to interpret demonstration as task nodes and thus produce action transitions as edges to link valid task nodes through CTG. Their work shows generalizability to unseen tasks. Nevertheless, these works still require a lot of effort to design the ground truths. For example, the ordering of each sub-tasks and how each node is connected to each other in graph representations must be defined.

On the other hand, some works adopt GNN to reason about the relationship between each object. In order to form high-level abstractions, [20] captured the interactions of object-object and robot-objects through graphical interaction networks [21] and used the relationships as states in a Model Predictive Controller. Moreover, in [22], they used object properties instead of position information. And then use the graphic observations to perform importance ranking and incremental planning among large amounts of objects at once. Nevertheless, their work is not suitable for a sequential planning problem. [6] has shown the capability of GNNs to directly learn the underlying task structures from pure objects' information without the need of defining symbolic descriptions of tasks with fully connected graph observations. However, in their work, they only take the initial state of each sub-task into consideration. Once the objects and goal positions are known, they leave the low motion generation to the conventional motion planner.

Compared to traditional approaches, the main scope of our high-level task planning is to allow a cobot to learn task structures without any hand-coded task descriptions or rules. Instead, the cobot learns the underlying task policy with only passive observations of an expert's demonstrations. Furthermore, we show that a GNN-based high-level planner is able to handle multiple environment changes including task structure changes, customer demand changes and inventory changes (see Section 4.1 and 4.5)

Compared to current GNN-based works, our high-level planner is able to provide more detailed guidance for low- level trajectory planning. Unlike the work in [6], our approach is able to produce different task abstractions during **picking** and **placing** stages. Moreover, in [6] and [22], they require extensive information regarding the object type, object fulfillment (i.e. whether the object has reached its goal position) and goal type. As a result, their trained high-level GNN model still requires extra effort in defining if-else-like statements. In our work, we release the designer from this burden and make use of only position information and necessary object features in our GNN-based **reasoning module**. These were obtained from passive expert demonstrations.

2.2. Trajectory learning from demonstrations

For solving long-horizon manipulation tasks, learning from demonstration (LfD) methods offer a convenient way for easy mapping from human to robot and fast trajectory level reproduction ability. Gaussian-based approaches such as Gaussian Mixture Regression (GMR) are often used to reproduce a trajectory from diverse demonstrations. From these demonstrations, a joint probability density is derived using Gaussian Mixture Models (GMM) [23,24]. Similarly, Hidden Markov Model (HMM) can describe the distribution of data through a mixture of multivariate Gaussian distribution as transition probabilities [25]. Nevertheless, such methods can only reproduce one single trajectory based on a specific initial and goal pose. In order to improve variability and adaptability, Task-Parameterized-GMM has been proposed in [26]. This works by extracting parameterized GMMs from different task frames. Nevertheless, it requires an extra algorithm to recognize the different task frames and it cannot generalize to unseen tasks. [27] solve these issues by using reinforcement learning-based optimization. However, most of the above methods still only focus on the Cartesian space

(3-Dimensional spaces) and are unable to directly produce robot joint actions.

Another common approach is to apply motion primitives. For example, [28] applied symbols as high-level task instructions while [29] used symbolic representations to optimize motion primitives. In [30], they decompose one complex motion as different phases. They then learnt the transition functions between each phase with model-based reinforcement learning that relies on a built library of motor primitives. Probabilistic Context-Free Grammars were used to build a sequence of motor primitives in [31] while [32] used skill trees (a form of motion primitive) segmented from expert trajectories for long-horizon manipulation tasks. Nevertheless, these works often require a careful design of motion primitive symbols.

An alternative way is to take advantage of the hierarchical structures to decompose a complex goal-oriented trajectory into smaller sub-goals. [33] adopts deep-learning-based methods to first identify the sub-goals and switch to reinforcement learning for continuous control. In [34], sub-goal trees were proposed to recursively predict positions in each task segment while [35] integrated inverse reinforcement learning with sub-goal-based demonstrations. These approaches still require rich explorations of the environment even with demonstrations.

In mPTO problem, there will be various objects to be manipulated. Thus, learning one trajectory is not enough. The collection of demonstrated trajectories will also be costly in terms of time. Meanwhile, the objects' safety should also be considered. Towards addressing these issues, we adopt a hierarchical structure for our **motion module** that can directly map observations to joint actions. The hierarchical structure consists of sub-goals and their related actions for trajectory learning from demonstrations. As a result, by accessing a task's relevant features and information (i.e. label), our sub-goal planner in the hierarchical structure is able to integrate different trajectories for various picking and placing positions. And it is able to learn the expert's planning strategy. In carrying out related sub-goal actions via final joint actions, we build an action planner that directly learns actions by modeling expert preference through a simple neural network. This is unlike [36] in which they used neural networks to learn the state transition function and thus guide the action learning of Deep Reinforcement Learning (DRL). Our work enables more data efficiency without the need for exploration. It can also produce adaptive collision-free plans for both unseen picking and placing positions.

2.3. TAMP architecture

Efforts have been made to integrate the aforementioned discrete plan planning and continuous trajectory learning methods within the same architecture. Such methods are defined as integrated Task and Motion Planning (TAMP) [37]. Nevertheless, the main challenge in a TAMP architecture is the integration of a discrete task planner with a continuous motion planner. Sampling-based methodologies are one of the common approaches used to do this. A sampling-based approach merges the discrete task planning and continuous motion into one common search space and uses sampling-based probabilistic search to navigate and find solutions in the search space [2]. In [38], they sampled valid sub-goals and actions through the use of cascaded variations inference with a user-specified reward function. The application of a conditional sampler with domain knowledge to sample actions in large solution space is another approach that was used in [39]. However, in mPTO cases, the sampling-based method is inefficient to handle the complex search at the task level. Furthermore, if the hierarchical character of trajectories were considered, the sampling-based methods may yield no solutions [2].

In order to solve this drawback, [2] suggests the use of a technique called Procedural Attachment. In Procedural Attachment, a high-level task planner is followed by an external motion planner [2]. There are several works that apply Procedural Attachment in addressing flexible production with robots. Examples include prioritizing the object sequence through the use of mixed linear programming (MILP) [40] as well as using Ordered Visiting Constraints (OVC) with constraint optimization [41]. However, these techniques require that a description of the task structure via symbolic representation should be known in advance so that the workpieces can be manipulated sequentially towards achieving the specific goal configuration. The need for such a task description means that they are not able to deal with variations in the environment. Thus, it becomes necessary to redefine the task description and retrain their model if a customer order changes. Furthermore, in these implementations, they only consider high-level plans while the low-level collision-free motion generation is left to an existing motion planner such as Rapidly exploring Random Tree (RRT). This could be time-consuming.

In order to deal with variations in observed objects, current advances in GNN [6] have shown the ability to directly use the graph-encoded representations to learn the high-level policy instead of using symbolic representations. For example, [6] showed that such an approach can learn task-specific rules and generalize to variant geometric goal configurations that make use of the objects' observation during demonstrations. In other words, their work only considers cases in which the task structure is static meaning that their approach cannot deal with redundant objects in a task structure as well as in between tasks. Furthermore, their approach cannot deal with variations in the task sequences of objects which are required for the human varying element in an HRC scenario.

Consequently, from the above discussions, we raise some questions that we aim to address for TAMP architecture in mPTO problem:

- Q1: In the mPTO scenario, how can a high-level decision-making module generate an accurate sequential plan while dealing with redundant objects in a task structure? It should be noted that the redundant objects are necessary as they might be needed for the next customer order which requires a different goal configuration. In manufacturing systems, this approach would reduce or potentially eliminate the downtime required during changeovers.
- Q2: How do we translate the change in sequential plan to the lower level motion planner for subsequent rapid motion generation? Also, how do we build a motion planner that can produce faster collision-free trajectories than conventional motion planners as well as handle variant pick and place positions?
- Q3: Furthermore, assume a human worker gets involved and decides not to follow the demonstrated task structure, how do we build a TAMP architecture that can rapidly detect this variation and adapt the sequential plan?

The next section discusses the methodology and framework that was applied in addressing these questions.

3. Methodology

In this section, we will describe the problem setup as well as the framework and methodology that were designed and applied to address the questions raised at the end of the previous section. We aim to provide an end-to-end solution learnt from expert demonstrations for a mPTO scenario. We follow a Procedural Attachment style in TAMP architecture that can hierarchically decide the useful features to different levels of planning.

Suppose we can obtain demonstrated observations as a tuple $\Pi = \{g, o, p, r, \mu, a, I\}$ from environment scenes. We have m products with position information $o = \{o_m\}_{m=1}^{m}$ from the pending area. According to customer demand, there will be n selected products with goal position information $g = \{g_n\}_{n=1}^{n}$ at the packaging box p ($n \leq m$).

The cobot should manipulate the selected products to the packaging area sequentially while considering two different cases under the same framework. The cases are:

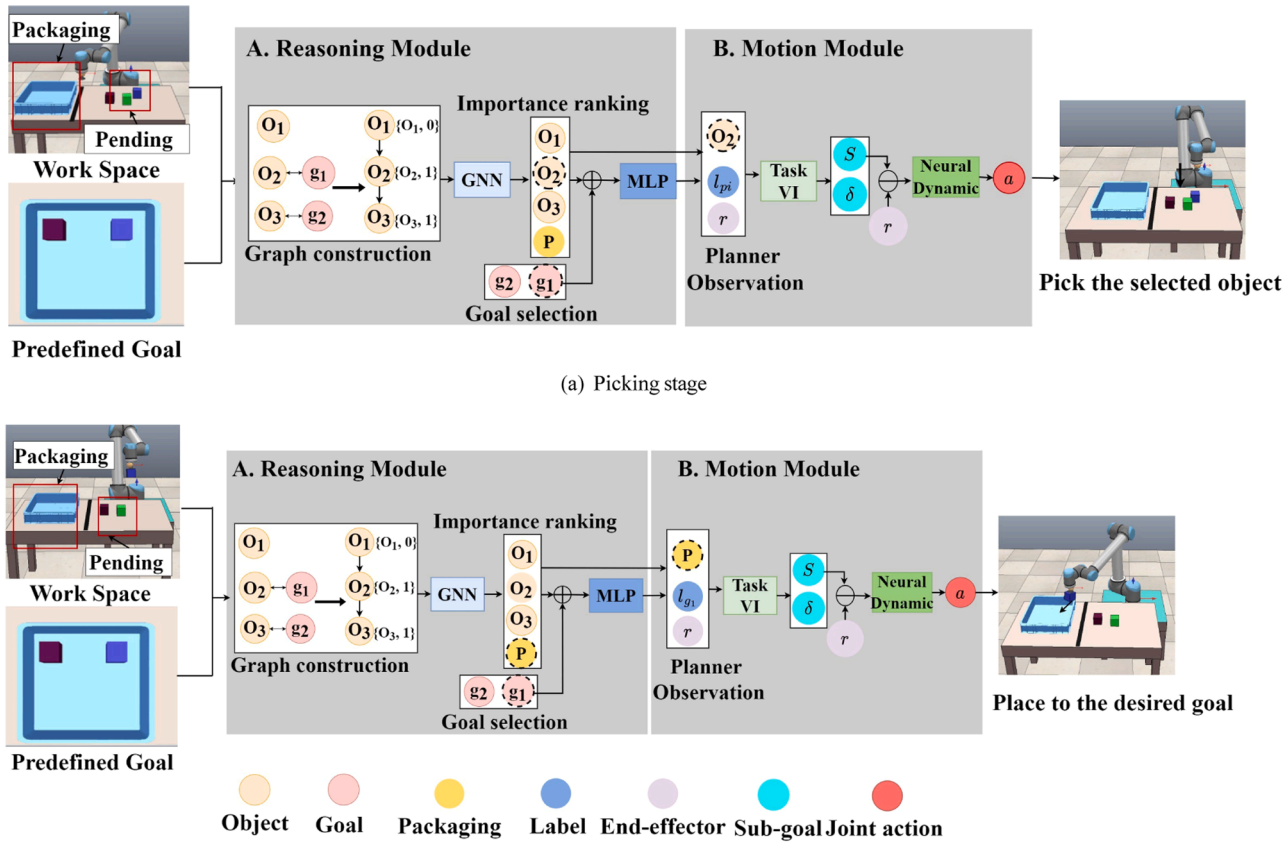


Fig. 2. Graphic representation of proposed framework at **picking** and **placing** stage. At **picking** stage, the **reasoning module** will always focus on the selected objects in the pending area while at the **placing** stage, it will always focus on the packaging box observations.

a) CASE 1: cobot should accomplish this sequential task alone by following the same task structure that was learnt from demonstration. In this case, the cobot picks and places the first product followed by the next one until the goal configuration is reached.
b) CASE 2: In the second case, the cobot should handle different task structures. This is particularly true when responsive Human-Robot-Collaboration is considered. In this case, human performance can be random, for instance, different arm movement trajectories could be used to pick and place objects. Thereafter, we only monitor the final states of human actions such as the objects' final positions.

Fig. 2 illustrates the proposed framework which was inspired by the natural way by which a human decides to manipulate an object. We consider that she/he often first focuses on a specific object and then proceeds to grab it. Once the object has been grabbed, the human then focuses on a specific goal pose to achieve.

To produce an efficient plan during manipulation, we separate the observed human demonstrations into two different task stages for each selected object and its subsequent manipulation:

a) Cobot needs to first focus on the most important object within o and pick it up. This is called the **picking** stage.
b) Once the object has been picked up, the cobot should carry the object to the specified goal pose. In this stage, we let the most important feature be the packaging box position p and infer the specific goal position based on the task label l. This is called the **placing** stage.

At each stage, we decompose the task into task planning and motion planning through the use of:

a) A high-level **reasoning module** that focuses on the objects observations and thus reduces them to the most important observations with tasks labels l
b) A low-level **motion module** that generates adaptive motions based on the selected observations and task information (e.g. labels) from **reasoning module**.

3.1. *Reasoning module*

The high-level reasoning module is built on a Graph Neural Network (GNN) with an additional Neural Network (NN) classifier. It plans the raw objects' sequential manipulation based on the customer's specifications. It should be noted that there could exist extraneous objects ($n \leq m$) where only a subset of raw objects needs to be manipulated to achieve the final goal configuration. We aim to model this decision-making problem as which observations should be focused at what stages during picking and placing. For instance, at the initial picking stage, the cobot should focus on the selected object o and its corresponding goal pose, once it has been picked, the cobot needs to focus on the packaging p area to achieve the required goal position.

The previous study provides both objects and goals nodes with extra properties defining the object type and its fulfillment in the graph. Although the goal positions for the objects can be variants (for example, when working on different geometric shapes of final goal configurations), such a setting will cause the reasoning module to only follow a specified task structure. In previous works, extra if-else statements were used to ascertain if the goal has been fulfilled or not in every step. However, in our study, our aim is to allow the agent to first distinguish the important observations and stages only through the object position

information with necessary features and secondly to become more generalizable when task structure varies.

To achieve these goals, the main idea is to reduce the high-dimensional observations through importance ranking. we first assign the necessary goal positions to each selected object. This will be selected by the importance score at the beginning of the **picking** stage. We treat each single object manipulation task as a graph classification problem. The output of the graph neural network will be a m +1 dimensional probabilistic distribution $P^o_{pred} = \{p^o_{pl}, p^o_1, ..., p^o_m\}$ where $\{p^o_1, ..p^o_m\}$ depicting the object importance within o at the **picking** stage and an extra p^o_{pl} suggests the importance of packaging area p at **placing** stage.

The distribution P^o_{pred} is combined with the selected goal position. This will then be further fed to a fully connected NN classifier for classifying $n+1$ dimensional probability distribution $P^g_{pred} = \{p^g_{pi}, p^g_1, ..., p^g_n\}$, where $\{p^g_1, ..p^g_n\}$ describes the goal labels at **placing** stage and an extra p^g_{pi} represents the **picking** stage. Note that each p^g_n in $\{p^g_1, ..p^g_n\}$ represents a specific goal position, which is demonstrated by the predefined goal configuration. We refer to it as a position-specified label. Such a design is to better infer the low-level **motion module** about the task stages.

Thereafter, the important features selected by P^o_{pred} and the one-hot encoded label $l \in \{l_{pi}, l_{g_1} ... l_{g_n}\}$ converted from P^g_{pred} are combined together as the inputs for the **motion module** (See Fig. 2).

Fig. 1 describes our neural network architecture for **reasoning module**. We first introduce a GNN to operate the graph classification. We encode states as graphs. Let there be n out of m ($n \leq m$) objects that need to be manipulated. There will be m nodes $V = \{v_m\}^m_{m=1}$ and each node contains 4-dimensional features $\phi(v_m)$, including the 3-dimensional objects positions o and an extra binary property $I = 0$ *or* 1 describing whether such an object has been selected or not according to the predefined $g = \{g_n\}^n_{n=1}$. We therefore define the directed linking edges $E = \{e_{i,j}\}$ for $i = 1, ..m-1$ and $j = 2, ...m$, where each node is only connected with its neighbors nodes.

We mainly adopt a GraphSAGE (Sage) [42] layer for this study. It holds the advantage of being generalizable to unseen nodes by sampling and aggregating the target node's neighbor nodes instead of weighting the whole neighbor nodes like Graph Convolution Network (GCN) [43].

Assume the initial node embedding is $h^0_i = \phi(v_m)$ and there will be K message passing iterations or K layers. We can thus aggregate its neighbor nodes h^{k-1}_j from the previous layer (i.e. $K-1$) and to form a single vector representation as Eq. (1). In this study, f_{agg} is the aggregator that aggregates the neighbours' features with an averaging function $\frac{1}{N}\sum_{j \in \mathcal{N}(i)} h^{k-1}_j$. This aggregated representation $h^k_{\mathcal{N}(i)}$ will be concatenated with the target node's embedding from the previous layer h^{k-1}_i and further multiplied by a weight matrix W_k. Thus, the node embedding of K_{th} layer can be represented as Eq. (2), where σ is the ReLu activation function. $f_{\theta_{gnn_1}}$ and $f_{\theta_{gnn_2}}$ are the trainable functions with parameters θ_{gnn_1} and θ_{gnn_2} for each layer. In order to prevent gradient explosion, we normalize the obtained node embedding as $h^k_i \leftarrow \frac{h^k_i}{\|h^k_i\|_2}$.

$$h^k_{\mathcal{N}(i)} = f_{agg}\left(h^{k-1}_j, j \in \mathcal{N}(i)\right) \tag{1}$$

$$h^k_i = \sigma(W_k \cdot \left(f_{\theta_{gnn_{k-1}}}\left(h^{k-1}_i\right) + f_{\theta_{gnn_k}}\left(h^k_{\mathcal{N}(i)}\right)\right)) \tag{2}$$

Afterwards, for graph classification, there will be an additional readout layer that aggregates the node embeddings into a graph embedding as Eq. (3). A final output layer accepts the graph embedding and produces $m+1$ final categorical distribution P^o_{pred}.

$$G_k = \frac{1}{\mathcal{N}(v)} \sum_{i \in \mathcal{N}(v)} h^k_i \tag{3}$$

We further build a classifier with three layers. It takes $m+1$ dimensional distribution P^o_{pred} and a 3D selected goal position g as inputs, which yields total $m+4$ dimensional features with the final outputs P^g_{pred}.

For training this module, we consider it as a supervised learning model with the ground truth distribution P^o_{goal} and P^g_{goal}. For both GNN and NN classification, we use cross-entropy loss as Eqs. (4) and (5). These two cost functions are jointly optimized as a linear combined cost function in Eq. (6).

$$loss_1 = -\sum_{m=1}^{m} \left[p^o_{goal}\right]_m log\left(p^0_{pred}\right)_m \tag{4}$$

$$loss_2 = -\sum_{n=1}^{n} \left[p^g_{goal}\right]_n log\left(p^g_{goal}\right)_n \tag{5}$$

$$\mathscr{L}_{re} = (loss_1) + (loss_2) \tag{6}$$

3.2. Motion module

In this section, we focus on building a **motion module** for a cobot to generate actions based on the information provided by the **reasoning module**. In mPTO scenarios, collision-free motions need to be considered in order to avoid any damage to the products. Thus, a cobot should avoid any collisions with itself and with objects at the **picking** stage. It should also avoid collisions between various objects already packed during the **placing** stage. Thus, the aim of this module is to learn the expert's planning strategies and integrate trajectories from different task demonstrations. In order to achieve this, we decompose the motion planning problem into two-level steps consisting of:

a) The generation of an effective collision-free plan as sub-goals conditioned by task labels.
b) The generation of joint actions based on the current end-effector position and the predicted sub-goal. This can be achieved by modeling the demonstrator's preference.

We build a conditional sub-goal planner with variational inference, which takes the most important observation from the **reasoning module** and cobot end-effector positions as inputs. This is also conditioned on the task stage label l. By accessing the task information (e.g. label), it is able to provide 3-dimensional collision-free sub-goals for different target poses even when similar observations are perceived. Lastly, the neural dynamic planner is built with simple neural networks to provide dynamic transition models of the expert preference based on the current end-effector position and predicted sub-goal.

3.2.1. Task-conditioned sub-goal planner

For sub-goal planning, if the same object was picked for different locations, similar observations can confuse the planner. Thus, we provide task parameters (i.e. labels) to differentiate between different tasks. The sub-goal plan s is based on the current end-effector pose r and different features obtained from **reasoning module** at different stages. In **picking** stage, the **reasoning module** will always provide the selected object position $o_{selected}$ and label indicating the task stage l_{pi}, and thus lead to total observations $\mathscr{O} = \{o_{selected}, l_{pi}, r\}$. During the **placing** stage, the **reasoning module** will focus on the packaging box position p with a label describing different goal positions l_{g_i}, and lead to $\mathscr{O} = \{p, l_{g_i}, r\}$.

Moreover, instead of using a deterministic model that directly generates the categorical distribution $p(s|\mathscr{O})$, we use a variational inference-based probabilistic regressor with additional uncertainty output δ (e.g. standard deviation(std)). We formulate latent parameters $Q(z) \sim \mathcal{N}(\mu, \delta)$ to approximate the ground truth sub-goal s as $p(z|\mathscr{O})$. The μ and δ can be parameterized with neural network as dependency of $\mathscr{O}$, $\mu = f_{\theta_{sub_\mu}}(\mathscr{O})$, $\delta = f_{\theta_{sub_\delta}}(\mathscr{O})$. According to Bayes rule, the posterior $p(z|\mathscr{O})$ can

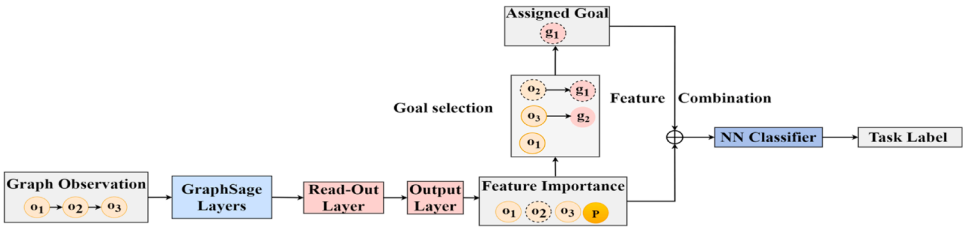


Fig. 3. Neural Network Architecture for **reasoning module**. We first construct a graph representation of the objects states. With three GraphSAGE layers and ReLu activation function, we obtain the most important observation as the probability distribution and thus select the corresponding goal from goal sets. Thereafter, the task stage can be classified by combining these two features through 3-layer neural networks with ReLu as activation function.

be expressed as Eq. (7). The integral form of marginal likelihood $\int p(\mathscr{O}|z)p(z)dz$ is often computationally intractable. In variational inference, it tries to find the optimal distribution $p^*(z)$ that approximates the posterior distribution. It equivalent to performing optimization by maximizing the Evidence Lower Bound function (ELBO) in Eq. (8), where $E_{z\sim Q}[logp(s|z)]$ is the likelihood term and D_{kl} is the Kullback-Leibler (KL) divergence that regulates the predicted variational probability $q(z)$ with a prior distribution $p(z)$. The KL divergence can be rewritten as the expectation form of z and finally, the cost function can be expressed as Eq. (9). Assuming the likelihood and variational distribution are Gaussian, we can thus replace them with a negative Gaussian log-likelihood function in Eq. (10). For prior probability, we assume the ground truth sub-goal has a unit Gaussian distribution $p(s) \sim \mathcal{N}(0,1)$.

Fig. 2 illustrates our network architecture, we use Stochastic Gradient Variational Bayes (SVGB) estimator with reparametrization trick to train the model [44]. By accessing task labels, the sub-goal planner is able to produce adaptive sub-goals among different tasks. The variational inference concepts can lead to high likelihood while penalizing over-fitting when estimated $q(z)$ is far away from the true prior $p(z)$.

$$p(z|\mathscr{O}) = p(z)\frac{p(\mathscr{O}|z)}{\int p(\mathscr{O}|z)p(z)dz} \tag{7}$$

$$argmax\mathscr{O}_z = E_{z\sim Q}[logp(s|z)] - D_{kl}[q(z)\|\, p(z)] \tag{8}$$

$$\mathscr{L}_{sub} = E_{z\sim Q}[logp(s|z)] - E_{z\sim Q}[logq(z) - logp(z)] \tag{9}$$

$$\mathscr{L}_{Gaussian} = -\frac{N}{2}\left(2\pi\sigma_{\theta_\sigma}^2\right) - \frac{1}{2\sigma_{\theta_h}^2}\sum_{i=1}^{N}\left(p'_i - \mu_{\theta_\mu}\right) \tag{10}$$

3.2.2. Neural dynamic planner

For the final action planer, since we are dealing with a large number of observations and continuous actions, a small amount of demonstrations are inefficient to directly map the observations to optimal actions required to achieve the sub-goal.

Instead, we model the expert preference as a dynamic transition function. Assume experts always prefer to minimize the distance between current end-effector position r_t and sub-goal position s_t at every time step t: $\Delta_t = s_t - r_t$ in a consistent way (i.e. they will first minimize Δ_t in the x-y plane, and thus approach the final goal vertically). Meanwhile, the joint actions $a_t = [a_1, a_2, a_3, ..., a_n]$ will lead to different posi-

Algorithm 1
Proposed TAMP architecture for adaptive packaging problem.

Initial observations $\Pi = \{g, o, p, r, \mu, a\}$
Trained **reasoning module** $\mathcal{R}$, and **motion module** with **task-conditioned sub-goal planner** $\mathcal{S}$,**neural dynamic planner** $\mathcal{D}$
In picking stage
Construct graph G based on objects o and their assigned goals g
Produce the important object $o_{selected}$ and select its goal $g_{selected}$ from GNN, and thus obtain the task label for picking l_{pi} as $\{o_{selected}, l_{pi}\} = \mathcal{R}(G, g_{selected})$
For *every sub-goal planning step in* ***picking*** **do**
  Produce sub-goal $s_t = S(p, r, l_{g_i})$
  For *every action planning step* **do**
    Produce joint actions $a_t = D(\Delta t)$, where Δt is the distance between end-effector position and sub-goal.
    $\Delta_t = s_t - r_t$.
In placing stage
Construct graph G based on objects o and their assigned goals g
Produce the packaging box position p from GNN and the placing goal label l_{g_i} as $\{p,\ l_{g_i}\} = \mathcal{R}\ (G, g_{selected})$.
For *every sub-goal planning step in* ***placing*** *stage* **do**
  Produce sub-goal $s_t = S(p, r, l_{g_i})$
  For *every action planning step* **do**
    Produce joint actions $a_t = D(\Delta t)$, where Δ_t is the distance between end-effector position and sub-goal
    $\Delta_t = s_t - r_t$.

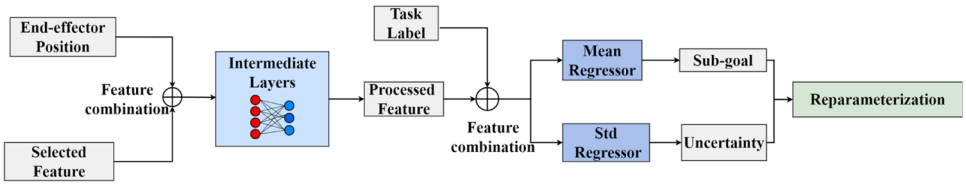


Fig. 4. Neural Network architecture for our Task-conditioned sub-goal planner with inputs from the **reasoning module**. We have two fully connected intermediate layers for pre-processing the observations after which we feed the processed features with task labels to variational inference. We apply a reparameterization process to train the network with the predicted probability distribution. We adopt Tanh as our activation function.

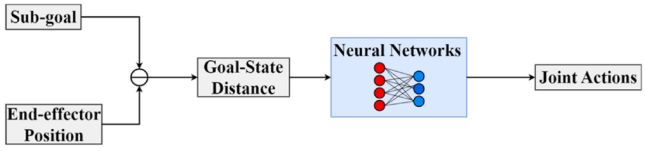


Fig. 5. Neural Network architecture of neural dynamic planner. We first calculate the distance between the predicted sub-goal and the current end-effector position. This will be fed into a sample three -layers neural network to finally produce the joint actions. The activation function between each layer is ReLu.

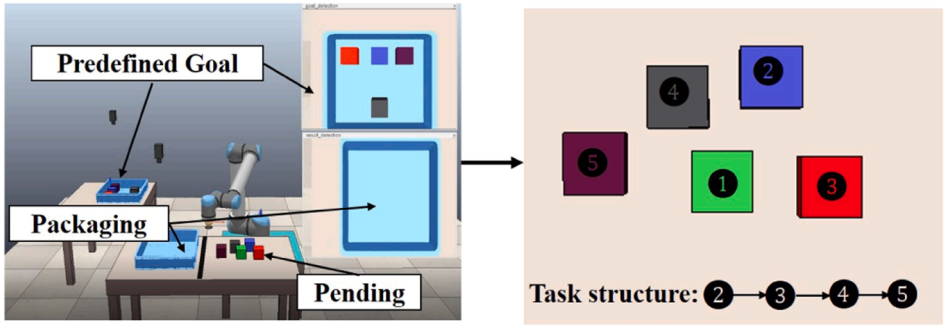
tion r_t through forward kinematic in robot arm. Thus, we formulate a dynamic transition function containing a continuous state space as Δ_t with action space $a_t = [a_1, a_2, a_3, ..., a_n]$. We aim to obtain the actions from the expert preference $\Delta_t = s_t - r_t$ inversely through a simple neural network $a_t = D_{\theta_D}(\Delta_t)$ as shown in Fig. 3. It is trained with a supervised loss function as Mean Square Error (MSE), which minimizes the loss between ground truth action a_t and prediction as shown in Eq. (11), where T is the training batch size.

$$\mathscr{L}_{act} = \frac{1}{T} \sum_{(a_t,\Delta_t)\in\tau} \frac{1}{2} ||a_t - D_{\theta_D}(\Delta_t)||^2 \tag{11}$$

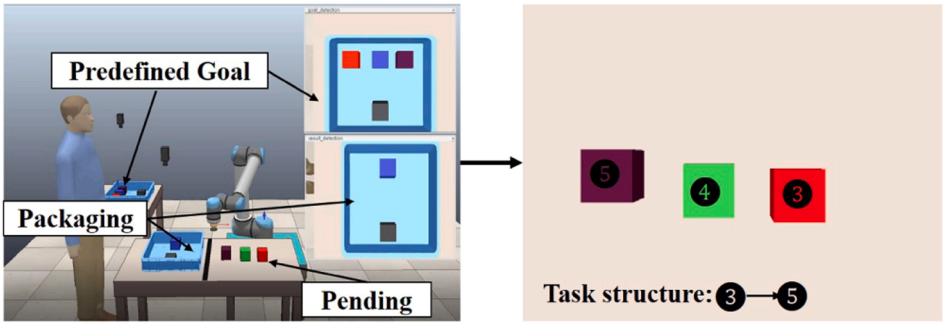
Finally, during testing, at the high-level planning, the **reasoning module** first identifies the task label and the most important observation. In **motion module**, the task-conditioned sub-goal planner will use the information provided by **reasoning module** to propose conditioned mean sub-goals while the neural dynamic planner tries to achieve the sub-goal during the low-level action execution steps as shown in Algorithm 1.

4. Experiments, results and discussion

In this study, we design our use cases to replicate a mPTO. However, due to the limitations related to the lack of massive customer orders, we scoped down the use cases to an adaptive packaging problem while preserving the nature of the changes that happen between batches of massive orders.

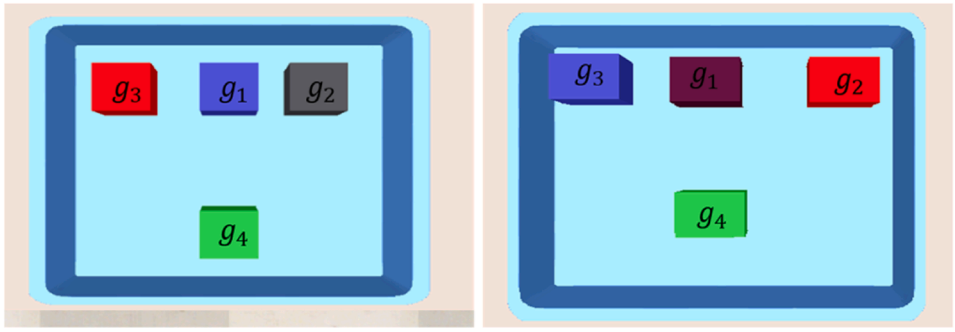


(a) **CASE 1**: The simulation experiment in 4 out of 5 experiment with the task structure followed by the demonstrator

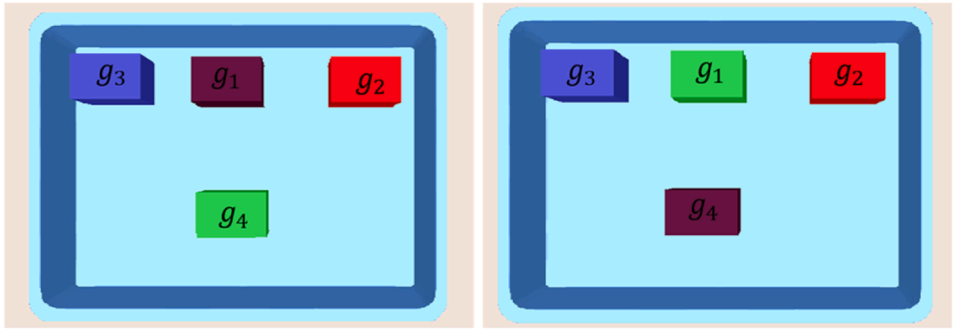


(b) **CASE 2**: human manipulate objects randomly, and the cobot should generalize to different task structures to carry out the rest of the task.

Fig. 6. This figure shows **CASE 1:** in which a static task structure is handled by cobot and **CASE 2:** in which the cobot works with a human on a varying and dynamic task structure. In these examples, the cobot needs to pick and place the target objects into the blue packaging box from the pending area. The human model in the figures is for illustrative purposes only.



(a) Different combinations in goal configurations when selecting 4 objects among 5 objects



(b) Different permutations under the same combination

Fig. 7. These figures show the different goal configurations including possible combinations and permutations with position-specified labels.

We design n out of m ($n \leq m$) adaptive packaging experiments with Fig. 5 describing an example 4 out of 5 experiment. According to a customer order, there will be n objects required to be packaged among total m objects.

We consider diverse goal configurations in every experiment as shown in Fig. 6 as every object has the potential of being selected. There will be ${}_mC_n = \frac{m!}{(m-n)!n!}$ possible combinations. Moreover, although the positions in the goal configuration are fixed, same objects may occupy different positions in the goal configuration. Thus, for each combination, there will be $P_n = n!$ possible permutations. Therefore, for every experiment, there will be a total of ${}_mC_n \times P_n$ different goal configurations.

We aim to solve two different cases in every experiment under the same proposed architecture. In **CASE 1**, the cobot handles the diverse goal configurations solely as shown in Fig. 6a. In this case, cobot should follow the underlying task structure learnt from demonstration. For example, if the selected objects are [2,3,4,5]. The cobot will always manipulate the selected object with a smaller number until the final goal is achieved.

In **CASE 2**, we present the generalization ability of our approach to unseen task structures as a result of human performance with diverse goal configurations as stated above. Unlike other previous work (e.g. [6]), it should be noted that the cobot in our work is free to compute and use another task structure depending on the perceived and observed current state of the task structure. This means that if the perceived task structure changes due to interference by a human, our architecture are able to support the cobot in understanding the change and responding accordingly. For example, as shown in Fig. 6b, the human picks the first and third selected objects while the cobot needs to manipulate the second and fourth selected objects sequentially.

To train our architecture, the high-level **reasoning module** and the low-level **motion module** are programmed separately using different expert demonstrations.

For the high-level **reasoning module**, we first demonstrate the desired goal configuration with position-specified labels as Fig. 7 shows. We thus allow the expert to manipulate the objects by following the same task structure as shown in Fig. 6a. We only collect two graph-based observations with the selected goal pose for each single object manipulation at the beginning of **picking** and **placing** stages as shown in Fig. 10. The ground truth distribution P^o_{goal} and P^g_{goal} are given as one-hot vector label ensuring that only the probability selected by expert will be 1 and the rest will be 0. The agent is then left to learn the underlying structures from graph-encoded observations without any hand-coded task descriptions or rules.

For the low-level motion module demonstration, we manually control the cobot to pick and place objects. We design 3 sub-goals each for different stages. For the picking, firstly, the cobot moves above the object followed by an approach to the object's surface and finally grabbing the object up. For placing, the cobot will approach a certain point according to the goal position followed by moving above the goal and then finally placing the object. For every sub-goal, we sampled 20 actions. The collected actions contain three most effective joint actions of UR10 including $[q'_{base}, q'_{shoulder}, q'_{elbow},]$.

In the following sections, we evaluate each module step by step with comparisons against alternative approaches. We train each module with expert demonstrations and test the trained models with our designed n out of m experiments in both simulation and the real world. For each single object manipulation during testing, the **reasoning module** will produce important features and task labels at beginning of **picking** and **placing** stages. For **motion module,** the sub-goal planner will produce three sub-goals at each stage. We set up a maximum of 30 steps for the neural dynamic planner to approach the predicted sub-goal. In the following sections, we present the average testing results over 5 seeds with standard deviation.

For **reasoning module**, we aim to demonstrate the adaptability and the generalizability of such a data-driven and learning-based model on variations in goal configurations and task structures. For **motion module,** we describe its flexibility of trajectory production while dealing with variations in position changes during both **picking** and **placing**. Thus, we show that our architecture holds the advantages of

being faster and more accurate than other methodologies. Finally, we validate our trained architecture on physical experiments.

For both simulation and physical experiments, we adopt a 6-DoF Universal Robot 10 (UR10) robot arm with a suction cup to interact with objects. In the simulation, we conducted experiments in the physical simulator CoppeliaSim [45] with cubes, while we use real-world products with similar shapes in the physical experiment. To detect them, we utilize a RealSense Depth Camera with a convolutional neural network (CNN)-based computer vision detection framework YOLOV3 [46]. All GNN-based models are written in PyG [43]. And the neural networks including our architectures and comparison methodologies are written in Pytorch [47] and trained on NVIDIA GeForce RTX 2060 GPU.

4.1. Reasoning module

In this section, we aim to show the ability of our GNN-based **reasoning module** is able to handle a large amount of goal configurations with redundant objects. We also examine the generalizability of our module regarding unseen goal configurations and task structures during different task stages. We compare our **reasoning module** against the different methods using the mPTO use case:

1) **GNN** [6]:We implemented the GNN-based high-level policy from the work with GraphSAGE Layers. The observations are encoded as a fully connected graph. This graph provides both objects and goals nodes with spatial 3D positions and extra properties regarding object types and fulfillment in the graph. performs the graph classification to directly output the two probability distribution of the target object P^o_{pred} and goal P^g_{pred}. In order to meet the needs of our experiment, we added an extra binary feature as I describing whether an object has been selected or not in every node. Their work can handle various positions for each object's goal by producing the same distribution P^g_{pred}. Hence, this ground truth P^g_{goal} label does not reflect the specific goal position. Instead, it only represents the order of goals. Furthermore, we assign the same ground truth for **picking** and **placing** stages as their work only considers the initial observation at the beginning of each object manipulation.
2) **GNN-task**: A design like **GNN** cannot distinguish between different goal positions. Hence the low-level **motion module** will always follow the same static trajectory regardless of the goal positions. In the comparisons that follow, we use their approach with our ground truth $P^a_{pred}$and P^g_{goal}
3) **MLP**: In comparing our approach with a traditional Multi-Layer Perceptron (**MLP**), we used flat 1D observations instead of graphs. And we replace the GraphSAGE Layers with a three-layer fully connected neural network. This was trained with our expert demonstration data.
4) **RF**: We also compare our approach with a Random Forest Classifier (**RF**) which is a non Neural Network based traditional approach. This approach infers the target predictions by using an ensemble of decision trees with each tree containing a sub-sample of data features. Each tree has branches that use Boolean-type logic to reach a decision. The **RF** reaches a decision through a majority vote from an ensemble of trees. We replace the GraphSAGE Layers with the **RF** in our comparisons. This was trained with our expert demonstration data.

We trained these methods with 5-fold cross-validation. The learning rates were set to 1×10^{-3}. During the test stages, we assess their performance by using a Success Rate (SR) metric. SR refers to the percentage of successful trials among the total attempts. We obtain the classification result for P^o_{pred} and P^g_{pred} at both picking and placing stages respectively. This is computed for every single object manipulation and one success trial is counted when all predictions for n objects are correct. For instance, in one n out of m experiment, there will be total $n \times 2$ predictions for both P^o_{pred} and P^g_{pred}.

Table 1
Success rates of different methodologies on n out of m objects reasoning in simulation.

n out of m	2 out of 3	3 out of 3	3 out of 5	4 out of 5
Ours	1 ± 0.000	1 ± 0.000	1 ± 0.000	1 ± 0.000
GNN	1 ± 0.000	1 ± 0.000	1 ± 0.000	1 ± 0.000
GNN-task	0.83 ± 0.015	0.72 ± 0.017	0.58 ± 0.024	0.24 ± 0.031
MLP	1 ± 0.000	1 ± 0.000	0.96 ± 0.013	0.86 ± 0.025
RF	1 ± 0.000	1 ± 0.000	1 ± 0.000	1 ± 0.000

Table 1 describes the performance on CASE 1. Each model is trained with the full demo data. The objects' positions are initialized randomly within the pending area and their performance with the trained scenarios are assessed. As shown in Table 1, our approach has a competitive performance when compared with **GNN** achieving a 100% SR among a large amount of diverse scenarios. For example, there are 120 different scenarios in 4 out of 5 experiment. **RF** method also shows comparable performance. However, the **GNN-task** performs the worst as it suffers from directly producing the positions specified labels from the extracted **GNN** features. This indicates our design of using an extra classifier is efficient to infer different task stages, especially when producing different position-specified labels at placing stage. The **MLP** method decreases in SR because of the challenge of dealing with variations in positions.

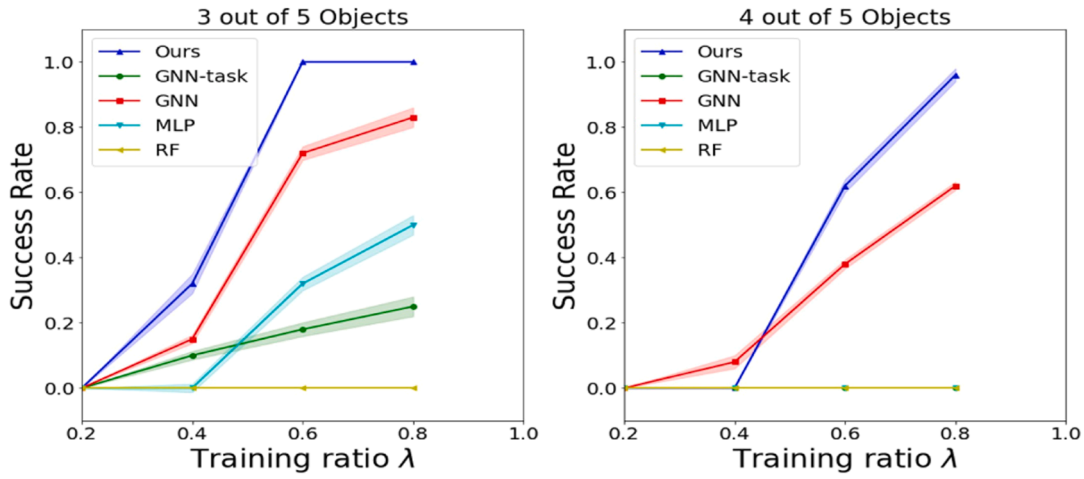
Next, we highlight the generalization ability of our module for dealing with an observation distribution outside the training data. In this study, we train the modules with only partial goal configurations using a proportion (which we call the training ratio λ) of the dataset.

In **CASE 1**, we randomly select $\lambda \times_m C_n$ combinations with their possible permutations P_n from the total goal configurations to train the models. Each goal configuration is trained only once. And the initial objects' positions are also randomized. We assess their SR on the unseen $(1-\lambda) \times_m C_n \times P_n$ tasks. This means that during testing, the module needs to produce task structures on previously unseen goal configurations as well as in the presence of previously unseen redundant objects in the observed scene.

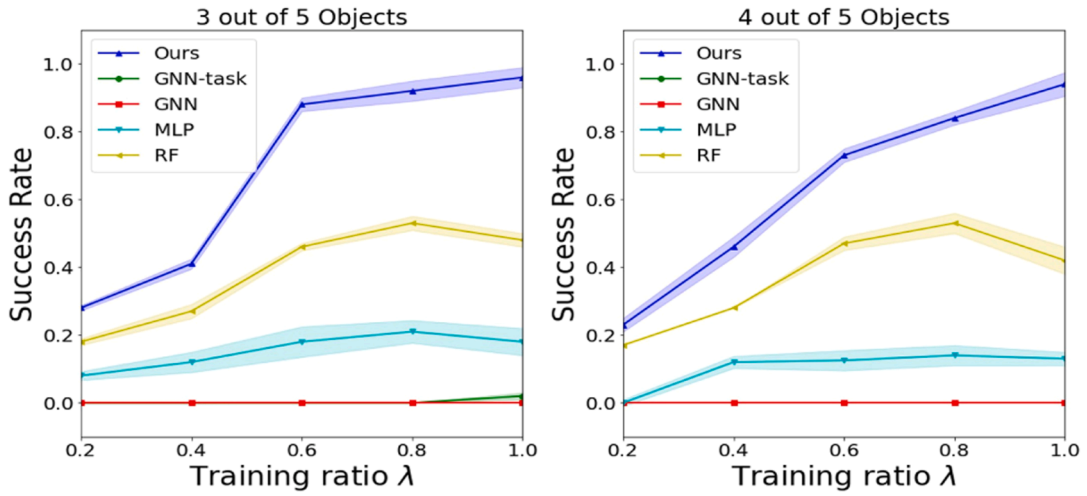
Fig. 8a shows the results on **CASE 1**. Our model can achieve 100% success on unseen goal configurations with $\lambda = 0.6$ in 3 out of 5 experiment and is able to handle 24 unseen goal configurations with 94 trained demos in 4 out of 5 experiments. **GNN** and **GNN-task** fail to generalize to unseen goal configurations with redundant objects. **RF** suffers to generalize to unseen goal configurations.

In **CASE 2**, similar to the study in **CASE 1**, we train the models with partial goal configurations $\lambda \times_m C_n \times P_n$. During testing, we augment testing data that perform the task in different task structures with randomly selected goal configurations (see Fig. 6b as an example). We have 300 testing scenarios for each n out of m experiment. We aim to prove that if our module is able to handle unseen task structures under unseen goal configurations.

CASE 2 as shown in Fig. 8b, is challenging due to the increase in the amount of unseen scenarios. Our module can achieve the best average of 96.3% and 93.7% success rates in unseen task structures for 3 out of 5 and 4 out of 5 experiments respectively when trained with full goal configurations ($\lambda = 1$). The **GNN** cannot handle different task structures. It fails to produce correct importance P^a_{pred}. We also notice that **MLP** even has better performance than **GNN** and **GNN-task**. The reason may be that, by using a fully connected graph, it may aggregate irrelevant neighbor node features and hence affect the prediction accuracy. Compared to **RF**, it indeed can solve part of the problem, while it still suffers from generalizing to varying task structures. We show our **reasoning module**'s results for generalization on unseen goal configurations and task structures as Fig. 9.



(a) Generalization study on handling unseen goal configurations in **CASE 1**



(b) Generalization study on handling unseen task task structures and goal configurations in **CASE 2**

Fig. 8. A study of the generalization over training ratio in simulation. A successful trial is defined when all predictions are correct for every single object.

Finally, we interpret the learnt importance ranking results with GNNExplanier [48] as Fig. 10. GNNExplanier aims to explain the trained graph model by determining the most important features and edges. As shown in Fig. 10, the feature I is necessary in our case as it allows the GNN to determine which subsets of objects need to be focused. Our reasoning module is able to produce efficient object importance with position information and its neighbor nodes at the **picking** stage. The spatial feature of the object height z infers the **placing** stages. Meanwhile, our module is robust to handle different task structures as we notice that it has identical masks in both **CASE 1** and **CASE 2**. This indicates that our graph construction enables the trained GNN to efficiently identify if some of the objects with $I = 1$ have been packaged (i.e. human performance) through only position information. And thus, it is still able to correctly plan the rest selected objects with the learnt task structure as shown in Fig. 9.

To conclude, our **reasoning module** can handle large amounts of different goal configurations and is data-efficient because it can generalize to unseen goal configurations under the same task during different task stages for cobot manipulation without any hand-coded task descriptions. Furthermore, our work is capable of handling different task structures and as a result, can support human-robot collaborative work.

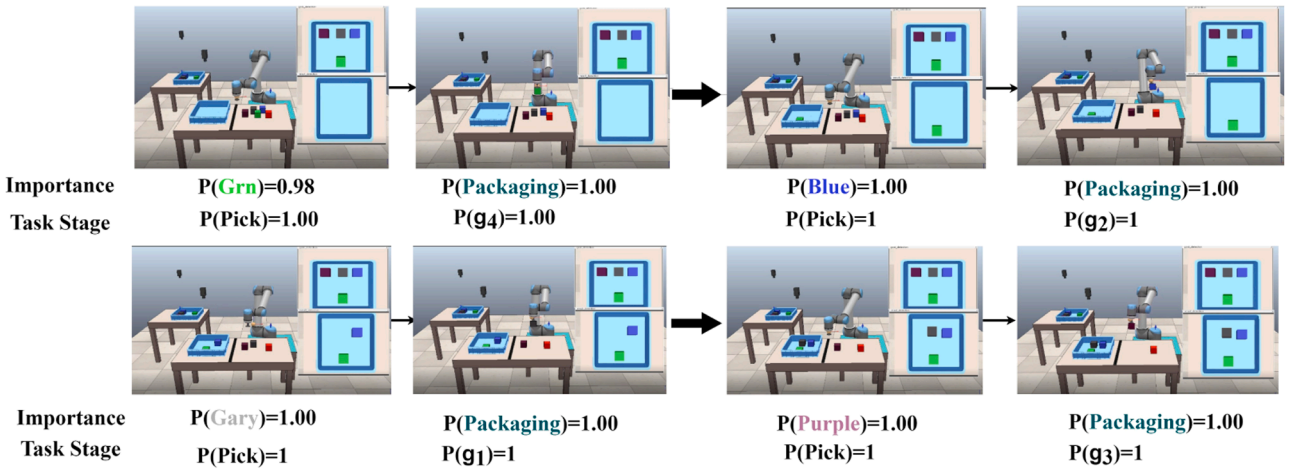
4.2. Motion module

In this section, we present the **motion module** 's performance. Since we have two separate neural network-based models (Task-Conditioned Sub-goal planner and Neural Dynamic Planner) in this module, we assess them individually.

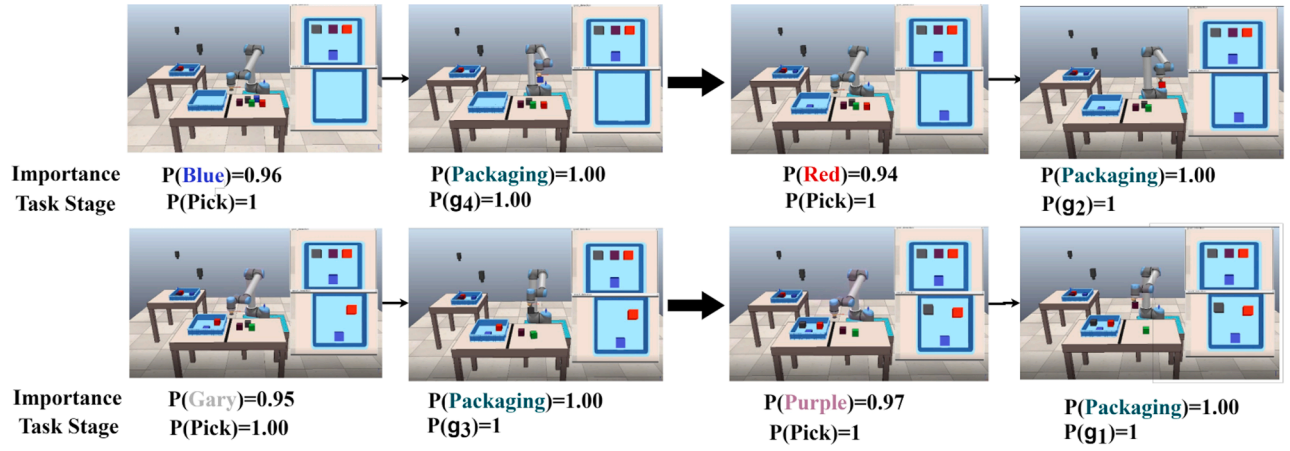
4.2.1. Task-conditioned sub-goal planner

In this section, we aim to justify the importance of providing task information from the high-level module and the effectiveness of our sub-goal planner in dealing with variations in object positions. Since we used a regressor in our work, we compare our sub-goal planner with different regression methodologies as discussed below. All the models are trained with expert demonstration data.

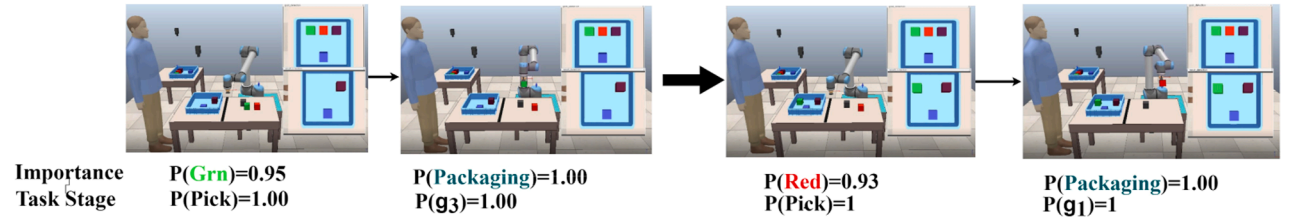
1) **VI**: This is a variational inference regressor without any task label.
2) **GPR** [49,50]: Gaussian Process Regression (GPR) is a non-parametric regression method based on the Bayes method. We provide the task label and train it with Radial basis function (RBF) kernel.



(a) Demonstrated goal configuration



(b) Generalization on unseen goal configuration



(c) Generalization on unseen task structures

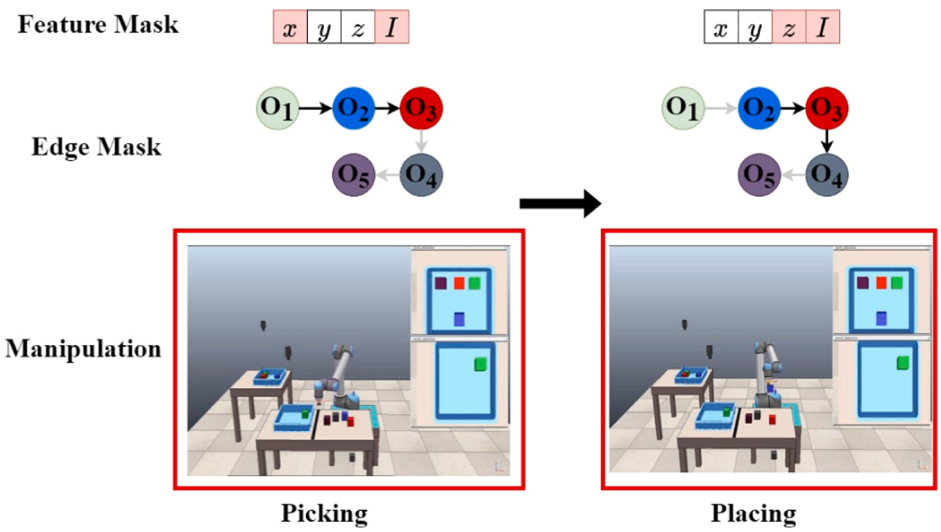
Fig. 9. We present the simulation results in 4 out 5 experiment. Our **reasoning module** produces the feature importance and task labels at the beginning of **picking** and **placing** stages for each object manipulation, where P (Pick) stands for picking stage, *P* (packaging) is the feature importance of packaging box at **placing** stage. Our model can handle unseen tasks in terms of goal configurations and task structures that are different from demonstrations.

3) **MLP**: We also build a simple neural network-based deterministic regression model with task labels (i.e. it does not measure the uncertainty of $p(s|\mathscr{O})$.

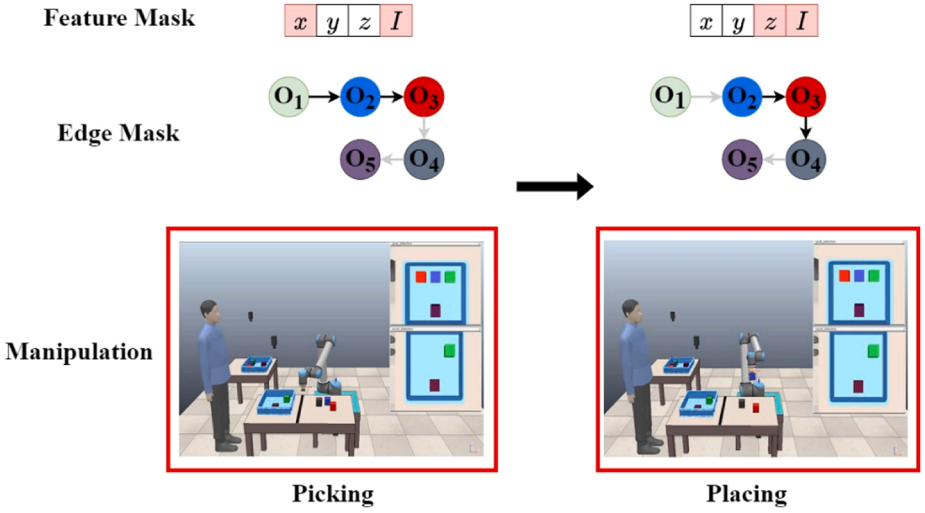
The learning rates were set to 1×10^{-3} and we measure the regression results based on the 3-fold cross-validation with R2 score, the Coefficient of determination. A higher R2 score indicates that the model can better explain the variability of data. Afterwards, we apply different models as sub-goal planners in our architecture and evaluate them in the simulation experiments. Furthermore, one object is only allowed to be manipulated to one goal during demonstration (training phase), while in testing, we studied the generalization ability of our approach.

We thus investigate if the sub-goal planner can produce an efficient plan when the same object needs to be used in different goal positions. For example, when considering the permutations under the same combination as shown in Figs. 7a and 13b. In this situation, an object might need to be manipulated into different goal positions starting from the same position. We illustrate one successful example as shown in Fig. 11.

During testing, we test 100 trials which are randomly selected from **CASE 1** and **CASE 2** as discussed in the previous section. One success in an object's manipulation is defined as when the Euclidean Distance between object pose and goal position is under a certain threshold $\delta_r = 0.06m$ (See Eq. (12)). One success trial is one in which all target objects have been fulfilled to the predefined goal without any collision during the execution. We still use Success Rate (SR) to evaluate their performances.



(a) Case 1: Showing the cobot moving towards the objects for subsequent **picking** and **placing** into a box



(b) Case 2: Showing the handling of different task structures when collaborating with a human

Fig. 10. We interpret the learnt **reasoning module** at **picking** and **placing** in two different cases. For each sub-figure above, the first row (Feature mask) describes the most important features and the second row (Edge mask) indicates the most important edges as solid lines. The third row describes the manipulation scenes from the simulation.

$$success = \begin{cases} 1 & |Pose_{obj} - Pose_{goal}| < \delta_r \\ 0 & otherwise \end{cases} \tag{12}$$

Our task-conditioned sub-goal planner can handle up to 4 different goal positions as shown in Table 2. The **MLP** has the worst performance, and we also notice it can only handle up to 2 different goal positions. In comparison, the **VI** has a better $R2$ score in the training data set. Nevertheless, we notice that it is unable to produce a stable sub-goal without accessing the task information thereby indicating the importance of task labels.

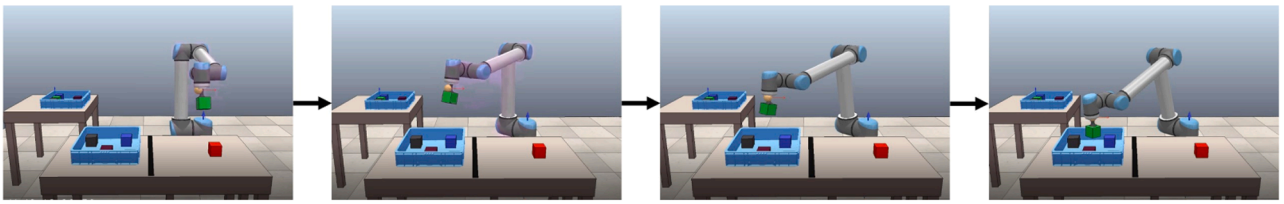
For **GPR**, it uses the whole training sample information to perform predictions of sub-goals. Although it has task information, it is still unable to handle the unseen goal positions which are different from training distribution. Our task-conditioned variational-inference-based sub-goal planner shows better performance when dealing with variations in the testing experiment.

As shown in Fig. 11, our sub-goal planner can imitate the planning strategy from the demonstration. It is also generalizable when different goal positions are provided while guaranteeing safety between objects. We also found that it can produce sub-goals when the cobot needs to pick an object at previously unseen positions as shown in Fig. 13a. However, it should be noted that these positions should remain within a particular range (see Section 4.3). The main failure case in our sub-goal planner happens when the predicted sub-goal is not efficient in catching the object at its surface.

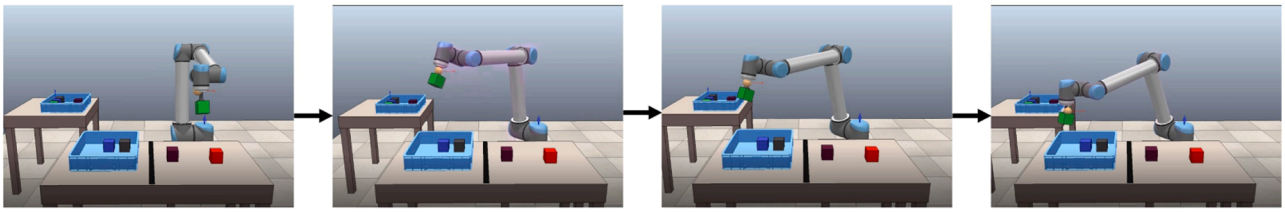
4.2.2. *Neural dynamic planner*

In this section, we examine the efficiency of only modeling the expert preference (i.e. state-sub-goal distance) to produce effective actions in cobot joints.

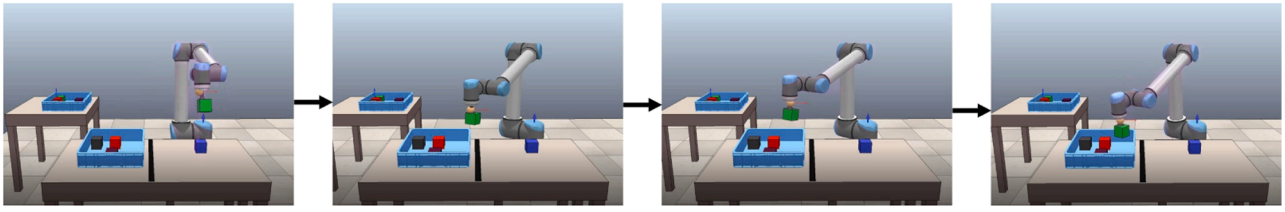
We compare different inputs as shown in Fig. 12. If the inputs become full observation including current end-effector position and sub-goal as $\pi(r_t, s_t)$. Such a model will become a simple actor-network in actor-critic based reinforcement learning like Deep Deterministic Policy



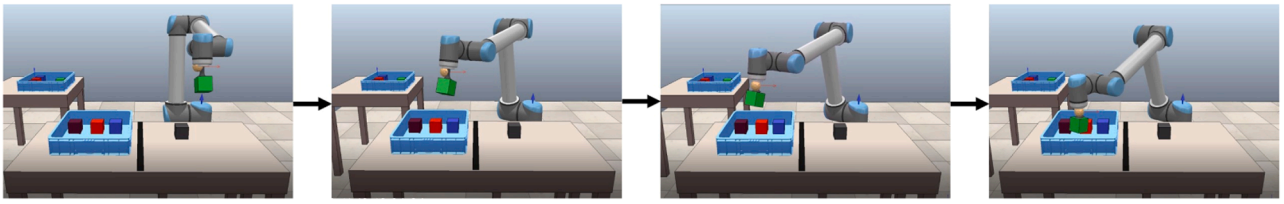
(a) Trained sub-goals from demonstration for g_1



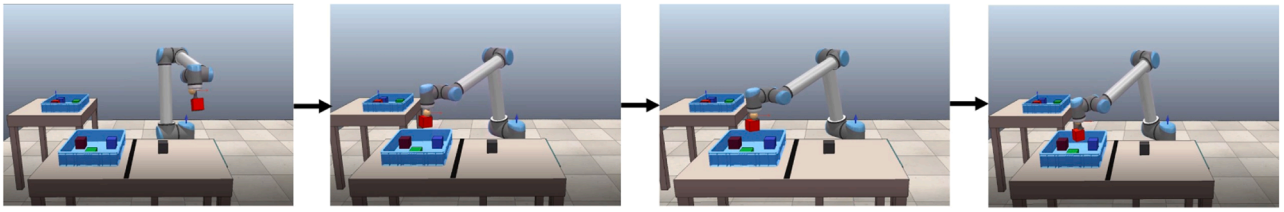
(b) Generalized sub-goals for g_2 with similar observation.



(c) Generalized sub-goals for g_3 with similar observation.



(d) Generalized sub-goals for g_4 with similar observation.



(e) Generalized sub-goals for g_1 with different observation.

Fig. 11. We show the simulation results of our sub-goal planner at **placing stage** in 4 out of 5 experiment. The first graph of each figure shows the initial state of the stage. The sub-goal planner will predict three collision-free sub-goals followed by actions from Neural Dynamics Planner. As the figure shows, our sub-goal planner is adaptable according to different task information (i.e. task labels) without any collisions.

Table 2
Different sub-goal regression methods' results on demonstration data, and success rates on the adaptive packaging experiments. We perform 100 testing trials in each experiment.

	2 out of 3		3 out of 3		3 out of 5		4 out of 5	
Model	R2	SR	R2	SR	R2	SR	R2	SR
Ours	0.98	0.96±0.012	0.98	0.94±0.008	0.97	0.92±0.012	0.97	0.89±0.017
VI	0.96	0	0.94	0	0.95	0	0.97	0
GPR	0.98	0.35±0.046	0.99	0.43±0.076	0.99	0.32±0.047	0.99	0.21±0.031
MLP	0.87	0.26±0.041	0.73	0.12±0.023	0.621	0.08±0.019	0.43	0.06±0.016

Gradient (DDPG) [51]. We train these models with 5-fold cross-validation with the learning rates as 1×10^{-3}. We allow the cobot to approach to the predicted sub-goal within maximum 30 steps and record the absolute mean distance at each time step.

As Fig. 12 shows, full observations $\pi(r_t, s_t)$ fail for producing optimal actions, and the predicted actions remain always the same as the red

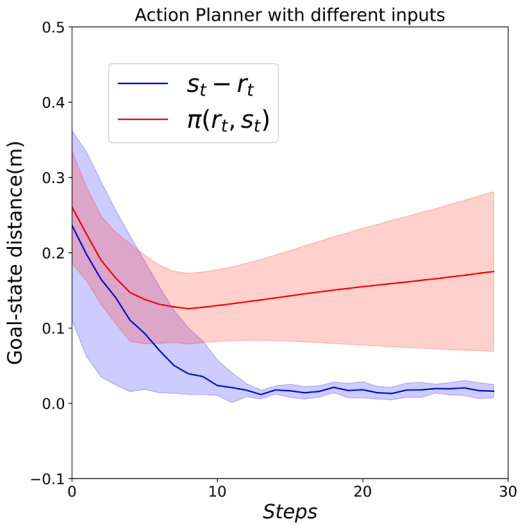


Fig. 12. We compare different inputs. The figure shows the absolute mean distance between the state and the goal with standard deviation over steps. We perform 20 trials with the random initial and sub-goal positions. The blue curve represents the performance of our neural dynamic planner. While the red curve is a simple actor-network.

curve shows. With inefficient state-action pairs, the actor-network may fail by getting trapped in local optimal. Also, the similarity between observations may also cause difficulty during training. Our neural dynamic planner as shown by the blue line in Fig. 12 can converge to a distance under 0.02 m at around 15 steps. That shows the data-efficient and adaptability of our proposed method.

Fig. 13 shows the 3D trajectories produced by our **motion module**. In the **picking stage** of Fig. 13a, the **motion module** follows the expert planning strategy by first moving above the object and thus picking the object. As seen in Fig. 13b, our module is able to deal with various object positions in the **picking** stage and capable of producing effective trajectories to various goal positions even with the same starting positions during the **placing stage**.

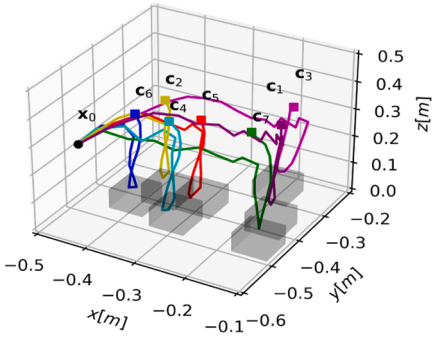
4.3. Overall performance

In this section, we discuss our whole architecture performance in every experiment. We randomly select scenarios in both **CASE 1** and **CASE 2**. The performance is assessed using the approach discussed in Section 4.2.1.

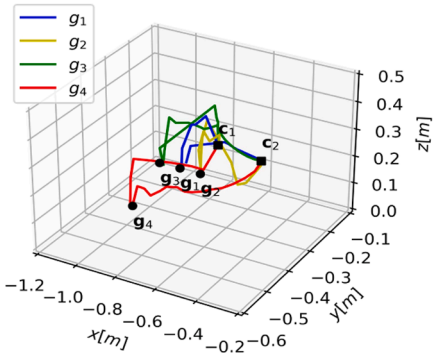
We compare different baseline methods including:

1) **DRL**: A Deep Reinforcement Learning (DRL) method uses a flat structure to generate actions directly from full observations. The observations contain objects positions $o = \{o_1, o_2, o_3 ... o_m\}$ and end-effector positions r. The action space is the same as our neural dynamic planner. For every trial, the goal also contains m objects positions including $g = \{g_1, g_2, \ldots, g_n\}$ for the selected objects while the rest objects' positions remain the same. We implement DDPG and Hindsight Experience Replay (HER) Buffer [52] with sparse reward. We allow the DRL agent to be trained for 5000 iterations for every experiment.
2) **RM+RRT-Connect** [53]: **RRT-Connect** refers to a conventional path planning algorithm that searches a configuration space with two rapidly expanding random trees growing from both the initial start point and target point. Since it requires both starting and target position, we use the **reasoning module** (**RM**) to produce the target object's position and its goal. We then replace our **motion module** with **RRT-Connect**. Moreover, in order to produce collision-free motions in mPTO problems, we provide environmental information in simulation via OMPL library [54].

In regards to object position variations in the environment, our proposed system can handle variations of 0.3 m on the x-axis and 0.15 m



(a) 3D representation of cobot end-effector's trajectory at **picking** stage for various object positions, where x_0 stands for the initial position, the cobot need to approach to the surface of the objects and catch them up to the picking points as c

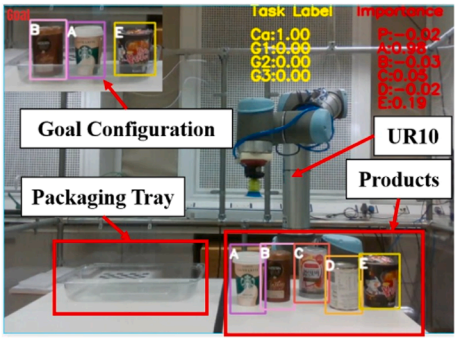


(b) 3D representation of end-effector's trajectory at **placing** stage starting at different picking point c and place the objects to the predefined goal positions as g

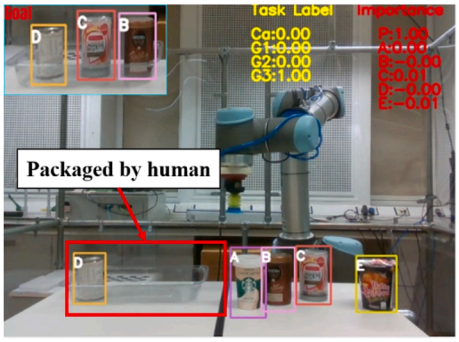
Fig. 13. The 3D reproduction trajectory from our **motion module** trained from 4 out 5 objects experiments.

Table 3
We compare the overall performance of different methods over two different cases (**CASE 1** and **CASE 2**). We trained the DRL agent with both **CASE 1** as well as **CASE 2**. We only trained our approach on **CASE 1** and show the average planning and execution time for our method and **RRT-Connect** in 4 out of 5 experiment.

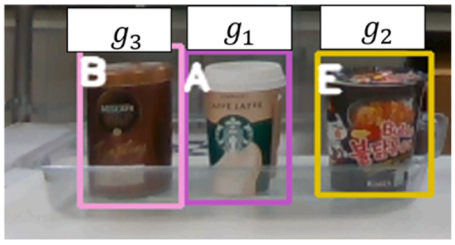
Model	2 out of 3 SR	3 out of 3 SR	3 out of 5 SR	4 out of 5 SR	Avg. planning time	Avg. execution time
Ours	0.96±0.012	0.94±0.008	0.92±0.012	0.89±0.017	7.8ms±0.32ms	14.25 s ± 2.9s
DRL	0.41±0.032	0.31±0.023	0.18 ± 0.018	0.07±0.011	–	–
RM+RRT-Connect	0.94±0.008	0.93±0.012	0.89±0.01	0.88 ± 0.012	9.6 s ± 3.32s	21.75 s ± 2.2s



(a) Case 1:Handling the same task structure



(b) Case 2: Handling different task structure caused by human



(c) Goal configuration with label

Fig. 14. The physical experiment in 3 out of 5 experiments. We utilize a RealSense 3D camera in front of the workspace as shown in Fig. 14a and b. Similar to the simulation, we first feed the predefined goal positions. The objects placed in the right pending area need to be manipulated to the left fixed packaging tray. The texts at the right top of the first two pictures indicate the real-time high-level task plan produced by our **reasoning module**. The yellow texts "Task Label" include the task label classification as different goals and *Ca* indicates the picking probability. The red texts "Importance" is the result of the feature importance of objects and P are packaging importance.

on the y-axis respectively regarding the object picking positions. It can also handle variations of 0.35 m on the x-axis and 0.2 m on the y-axis respectively regarding the goal positions. Also, the Euclidean Distance between objects and objects' goals can vary from 0.38 m to 0.82 m.

As shown in Table 3, the **DRL** algorithm struggles in dealing with high dimensional observations and goal spaces. Meanwhile, it is also inefficient in data, meaning that it needs a large amount of exploration. The hierarchical property of our proposed TAMP system is able to produce an efficient plan by reducing the high dimensional observations into the informative features for each module. Meanwhile, our **motion module** has a competitive performance with collision-free **RRT-Connect**.

We also compare the planning and execution time for our proposed **motion module** with the **RRT-Connect**. As mentioned before, for each object manipulation in our proposed system, our system will plan three sub-goals sequentially to achieve specific positions during each stage. We allow a maximum of 30 action steps to achieve each sub-goal. During testing, the planning time for our NN-based planner is around 1.3 ms for one of the sub-goals. During manipulation, the joint actions are constrained within [−5, 5] degrees. As a result, the total action steps for one single object manipulation can vary between 27 and 47 steps while the execution time can vary between 10.51 s and 17.94 s.

For comparing our **motion module** with **RRT-Connect**, we adopt **RRT-Connect** in both **picking** and **placing** stages. An inverse kinematic solver is used to iteratively find the optimized joint configurations. We also provide environmental information at the beginning of each object manipulation in order to achieve collision-free motions. This means that the objects that do not need to be manipulated are considered as obstacles. It was discovered that **RRT-Connect** often requires longer average planning time (9.6 s). And the execution time varies between 18.3 s to 25.2 s. Also, the configuration space becomes more complex as the number of objects increases and this affects the SR values. Moreover, we noticed that the inverse kinematic solver may produce dangerous joint configurations due to redundancy.

Table 4
Physical experiment results in both cases, we also show the total time of each object manipulation.

	2 out of 3	3 out of 3	3 out of 5
SR in Case 1	0.98±0.002	0.95±0.01	0.93±0.008
SR in Case 2	0.91±0.012	0.91±0.007	0.87±0.034
Avg. time	44.3s±1.45	43.2s±2.3	43.5s±1.8

4.4. Physical experiment

We conduct the physical experiments with a similar setting as in simulations. As long as the distribution of positions remains similar as in simulation, our architecture can directly accomplish the physical experiments without extra training. We utilize a RealSense 3D camera in front of the workspace to sense the environment as shown in Fig. 14. To obtain the real-world products' information with respect to the cobot base, we collect and train the object detection with CNN-based detection network YoloV3 and perform coordinate transformation through Robot Operating System (ROS) [55]. The suction cap is activated or deactivated by controlling the digital I/O output on the UR10 during a single object manipulation cycle. The desired joint actions produced from our system control the cobot through ROS and MoveIt [56]. One drawback of our neural dynamic planner is that it cannot adjust the orientation of the end-effector in real-time. Nevertheless, we adjust the orientation with MoveIt for picking objects with predicted sub-goal during **picking stage** and the final step of **placing stage** for better detection of objects.

Fig. 18 illustrates one example of our experiment in 3 out 5 experiment in **CASE 1**. As shown in Fig. 18, our architecture still follows the same learnt task structure as in simulation, (i.e. from A to B … until the final goal configuration is achieved). In **CASE 2**, we allow the worker to first package the products with his own preference. Afterwards, the cobot gets involved and starts to plan and package the rest of the selected products through our architecture as shown in Figs. 16 and 17. Note that in achieving these results, our architecture was not trained by **CASE 2** scenarios. We also show the performance of our **motion module** in the physical experiment as shown in Fig. 15. This demonstrates the generalizability of our approach. We conduct 20 runs for each case in every experiment including 2 out of 3, 3 out of 3 and 3 out of 5. The success rate results in both cases have shown that our proposed system

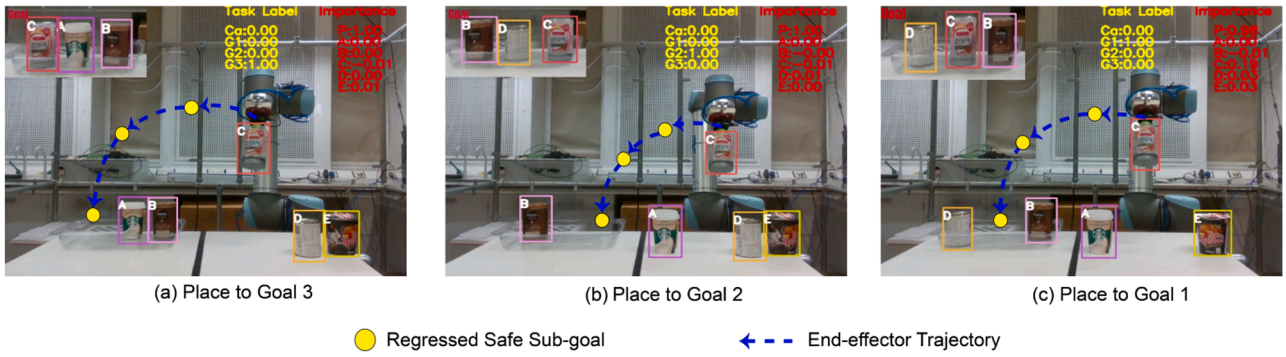


Fig. 15. The performance of our **motion module** in adapting to various goal positions during physical experiments. For example, the **motion module** was trained to put object *C* at Goal 3 as in Fig.15a. However, the **motion module** can adaptively produce trajectories for other unseen task goals as seen in Fig.15b and Fig.15c.

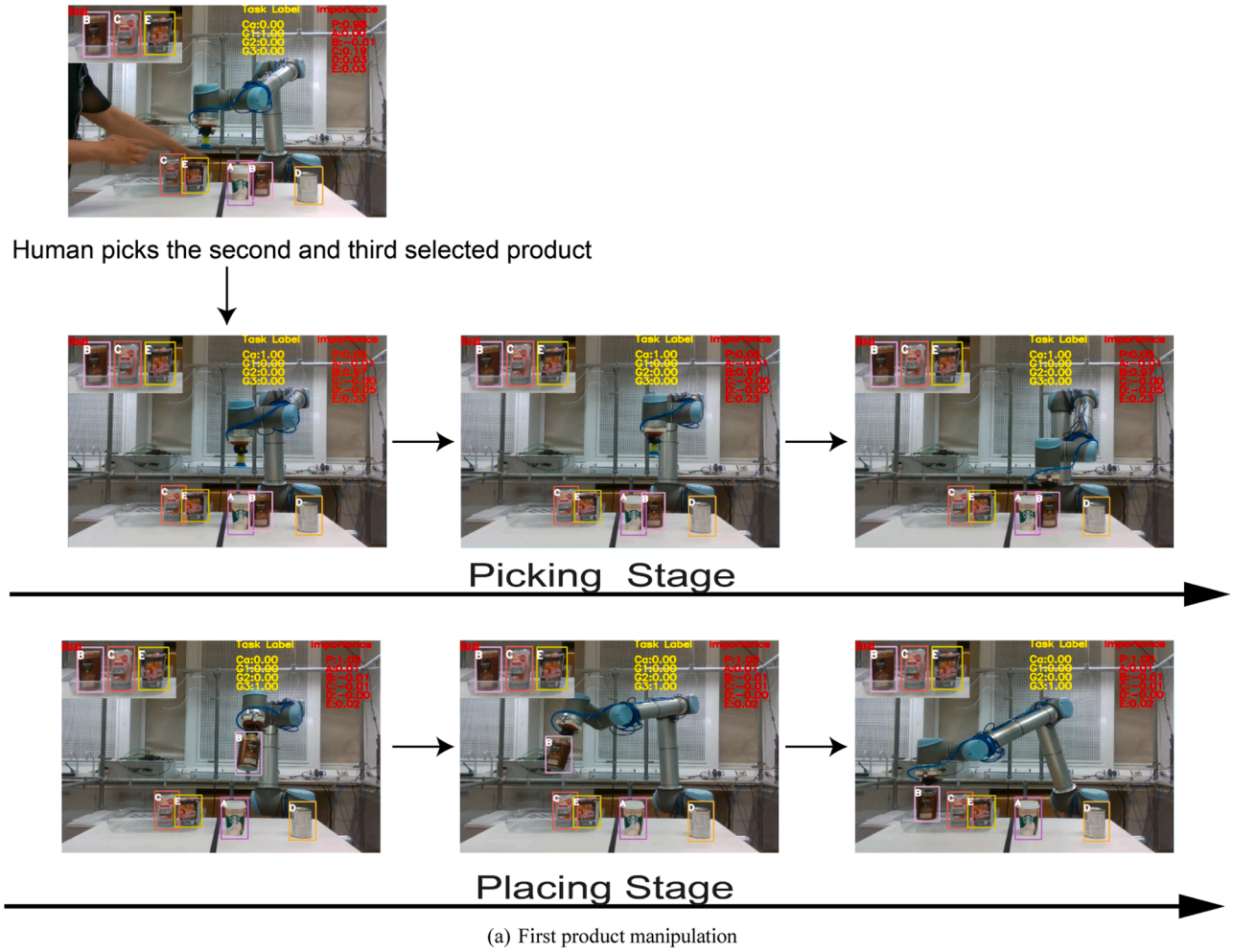
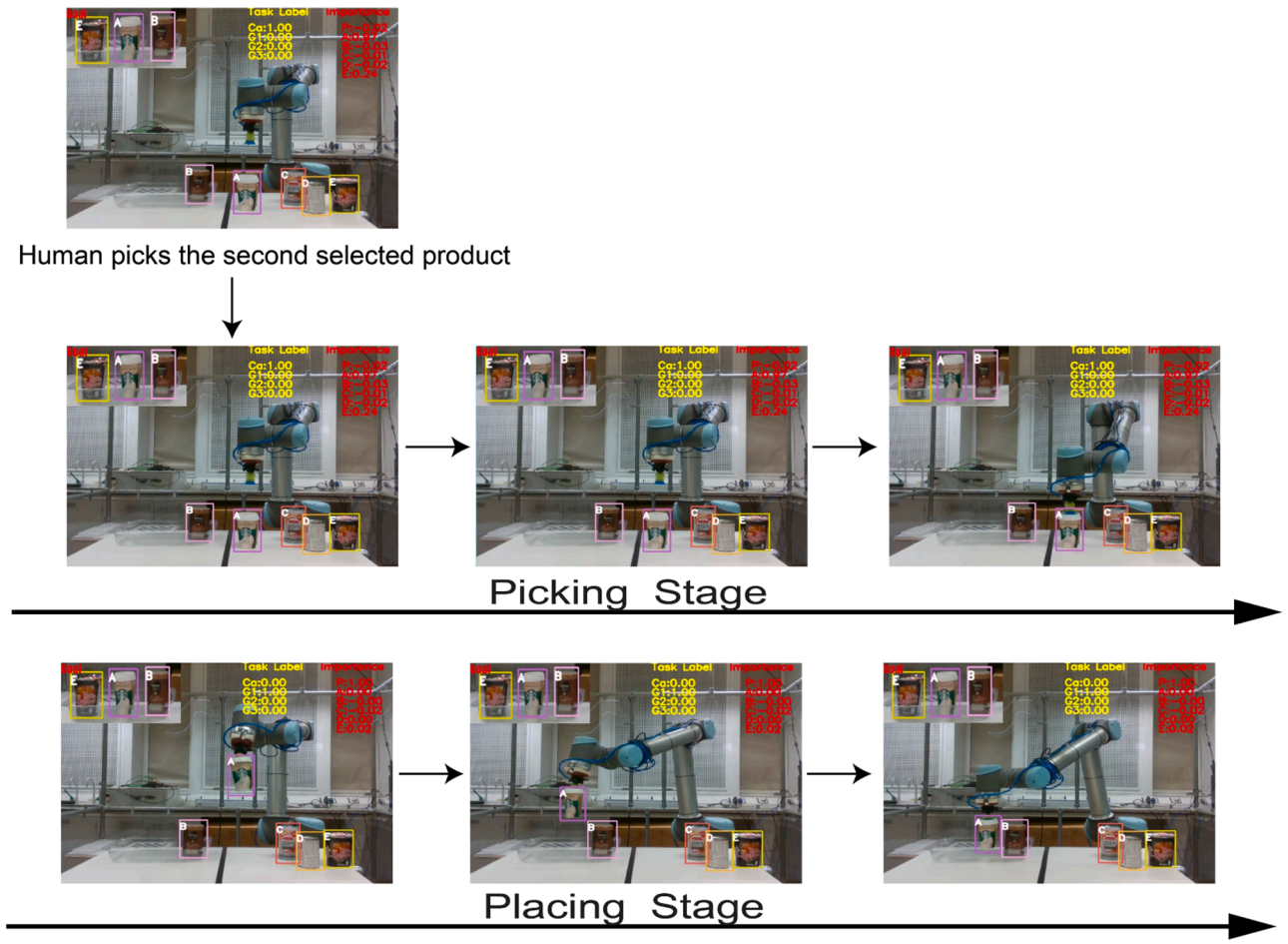


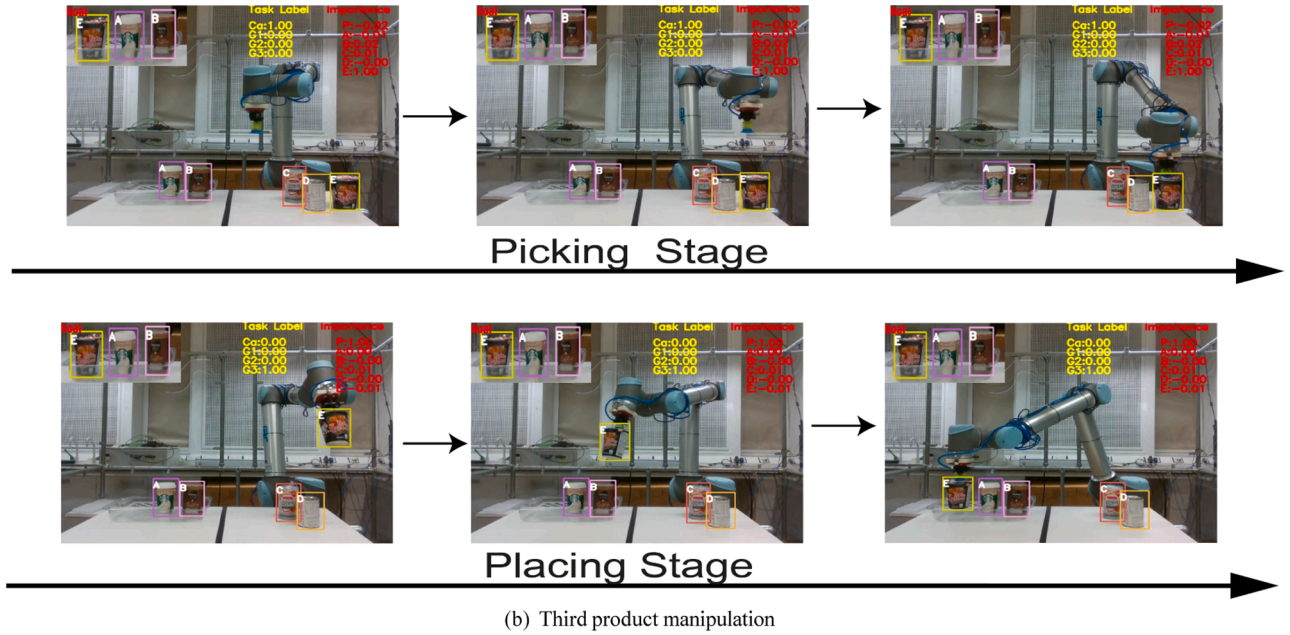
Fig. 16. Physical experiment for 3 out of 5 in **CASE 2**, where the human picks the second and third objects and lets the cobot do the rest.

trained on simulation can efficiently generalize to hardware for both **reasoning module** and **motion module** as shown in Table 4. It will take around 5 ms for **reasoning module** to construct graph and make predictions. The planning time for the sub-goal planner is similar to the simulation. The Euclidean Distance between the object and the goal varies from 0.25 m to 0.83 m. Thus, the steps taken for each object manipulation vary from 20 to 47 steps. We set the velocity scaling factor for UR10 to 0.05 to ensure safety between cobot and human. Therefore, the average time for one step taken will be around 0.7 s. Furthermore, the extra orientation correction will take an average of 10.2 s at each stage. We did not implement collision-free RRT in real-time, as it needs rich environmental information.

Nevertheless, the main cause of failure is the unpredictable dynamics of the real products during manipulation, which may cause unexpected motions during the **placing** stage and collide with other products. The misclassification of the products in YOLOV3 detection may also decrease the success rate during the experiments. Furthermore, due to the extra orientation correction with inverse kinematics at task stages, the average time for one single object manipulation increases. Nevertheless, as seen from the results above, our architecture can still achieve faster



(a) First product manipulation



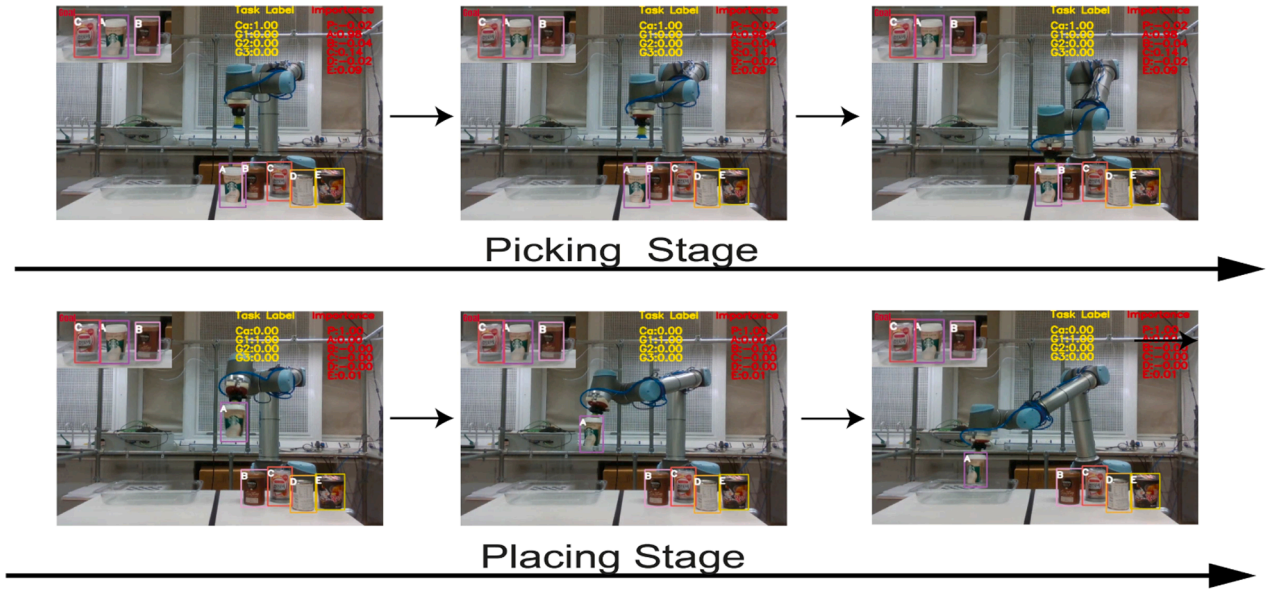
(b) Third product manipulation

Fig. 17. Physical experiment for 3 out of 5 in **CASE 2**, where the human picked the second object and lets the cobot do the rest.

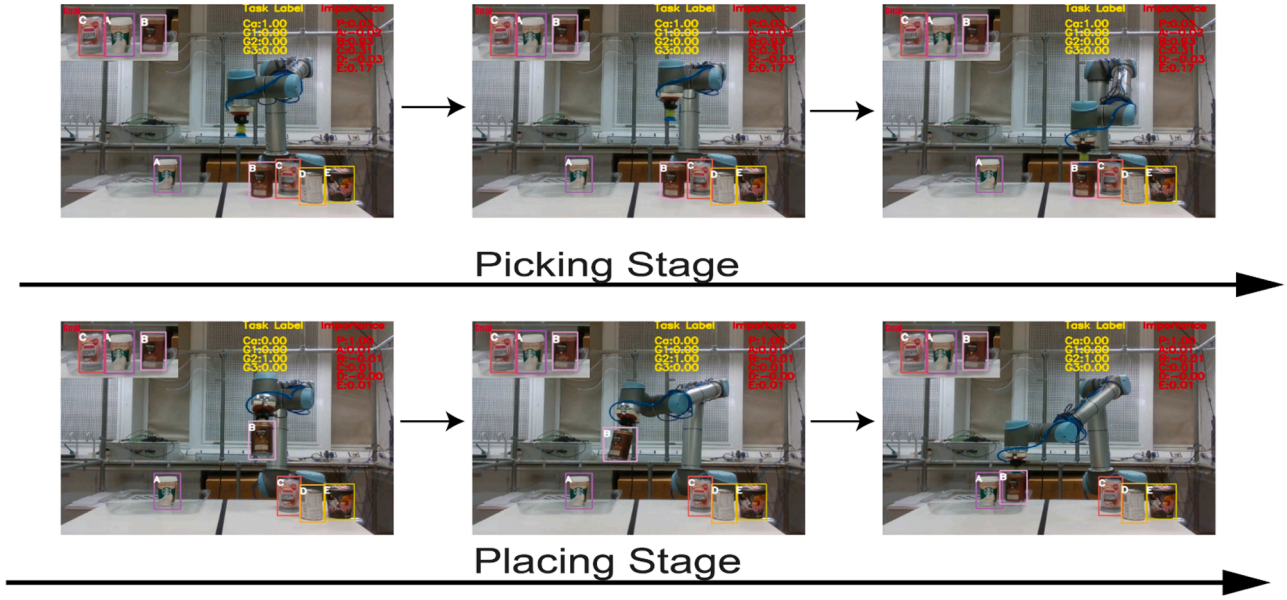
planning and execution time than the conventional motion planner.

4.5. Practical scenarios

In this section, we further explore the ability of our proposed method of dealing with variations in practical scenarios. Towards this, we



(a) First product manipulation



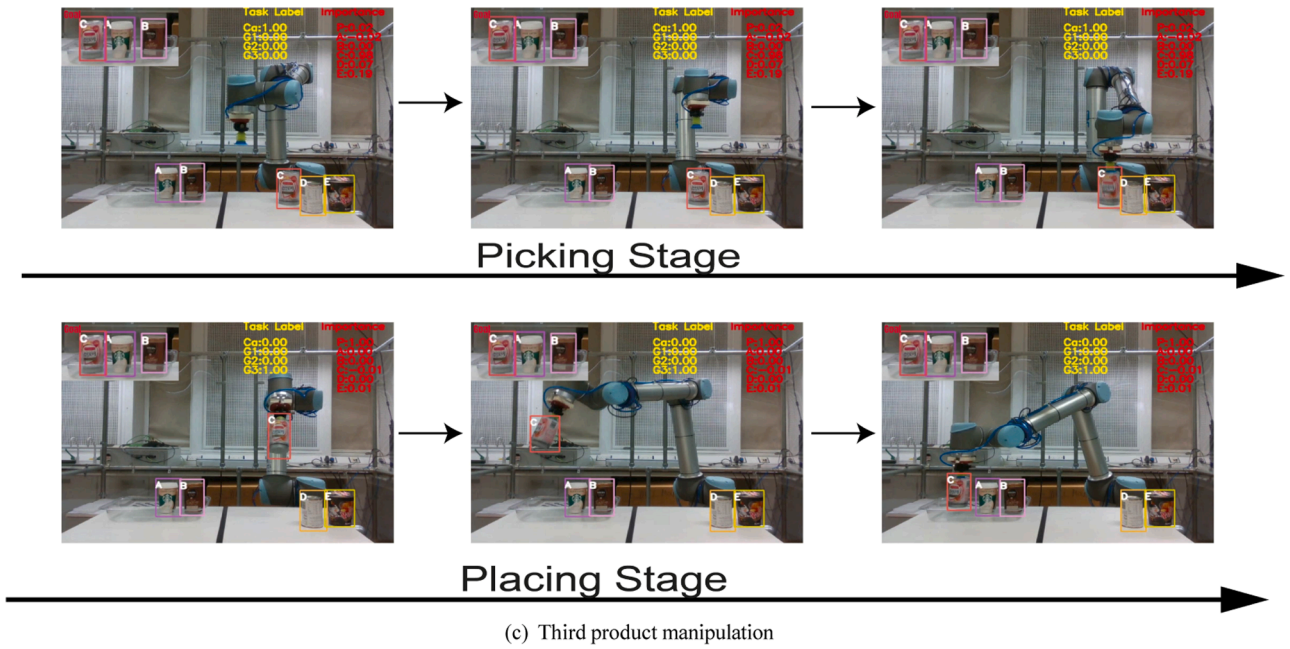
(b) Second product manipulation

Fig. 18. Physical experiment for 3 out of 5 in **CASE 1**, where cobot packages the customer order solely.

present two practical problems that may exist in mPTO scenarios.

Firstly, we consider one practical problem in which some items are lacking in the inventory of the pending area and waiting to be replenished. In this scenario, in the interest of meeting a customer order fulfillment time, the cobot needs to first pack the available products to the corresponding goals while waiting for the replenishment of the missing products. Thus, such a problem will need a different task structure rather than the previously learnt task structure. This scenario can be recognized as a partial observable problem, where the dimensions of observations can vary. When the **MLP** and **RF** approaches are applied in the **reasoning module**, they are unable to handle this problem because they require the observations to have the same dimensions during both training and testing. On the other hand, a GNN-based approach is more powerful for handling such a problem as it can process different numbers of nodes. In this situation, our reasoning module requires partial retraining with extra augmented data of 50 randomly chosen partial observable expert demonstrations from 3 out of 5 experiment. This step allows our reasoning module to perform better inferences based on each object's position information. To test our **reasoning module**, we conducted 180 experiments five times with 3 out of 5 scenario and with 1 or 2 products lacking. Our reasoning module can achieve an average SR of 0.97. We also conducted 20 physical experiments with our whole system as shown in Fig. 19. Our approach was able to achieve an average SR of 0.92 among 5 test sets.

Secondly, we considered that customers can choose different product combinations (for instance: **2 out of 5, 3 out of 5** and **4 out of 5**). This



(c) Third product manipulation

Fig. 18. (*continued*).

requires the cobot to handle various combinations and different task lengths at the same time. We train **reasoning module** in **2 out of 5** with expert demonstration in **CASE 1**. The end goal position for each object can be randomly selected as described in Fig. 7. We then analyze our approach's generalizability to handle **3 out of 5**, and **4 out of 5** in both **CASE 1** and **CASE 2** as shown in Fig. 20. For each experiment, we randomly select 100 test sets with our proposed system. We compared our approach with **MLP** and **RF**. They were unable to generalize to variations in customer choices. However, our approach trained with a simple scenario (**2 out of 5**), can adapt to more complex unseen tasks (i.e. **3 out of 5** and **4 out of 5**).

The above experiments show that our graph-based **reasoning module** is efficient in directly learning the underlying task structure without any specific hand-coding rules. It can also generalize a learnt task structure to more complex problems. Compared to other baseline methods, our approach is more capable of processing varying lengths of observations making it suitable for practical industrial mPTO problems.

4.6. Discussion and limitation

In this section, we discuss our proposed architecture and revisit the research questions raised in Section 2.3 based on the results from the experiments conducted.

For the high-level decision-making problem, we have shown that our **reasoning module** is capable of directly learning the underlying task structure from observations without any specific design of task structures or extra effort that is without the need to use specific *if-else* rules. This answers **Q1** raised in Section 2.3.

Towards addressing **Q2**, our architecture efficiently translates the changes in the high-level plan to low-level motion generations by separating a single object manipulation into **picking** and **placing** task stages. This allows the **reasoning module** to identify different necessary features via P^{o}_{pred} for **motion module** at different manipulation stages. Furthermore, we use the classified labels P^{g}_{pred} in order to help the **motion module** to differentiate between sub-tasks. Furthermore, our **motion module** can efficiently learn an expert's planning strategy and can be adaptive when different observations are obtained. For the sub-goal planner, our results have shown that a neural-network-based probabilistic model (i.e. Variational Inference) can handle larger variations than a deterministic model or a non-parametric approach. We have also shown that our neural dynamic planner based on expert preference is also data-efficient.

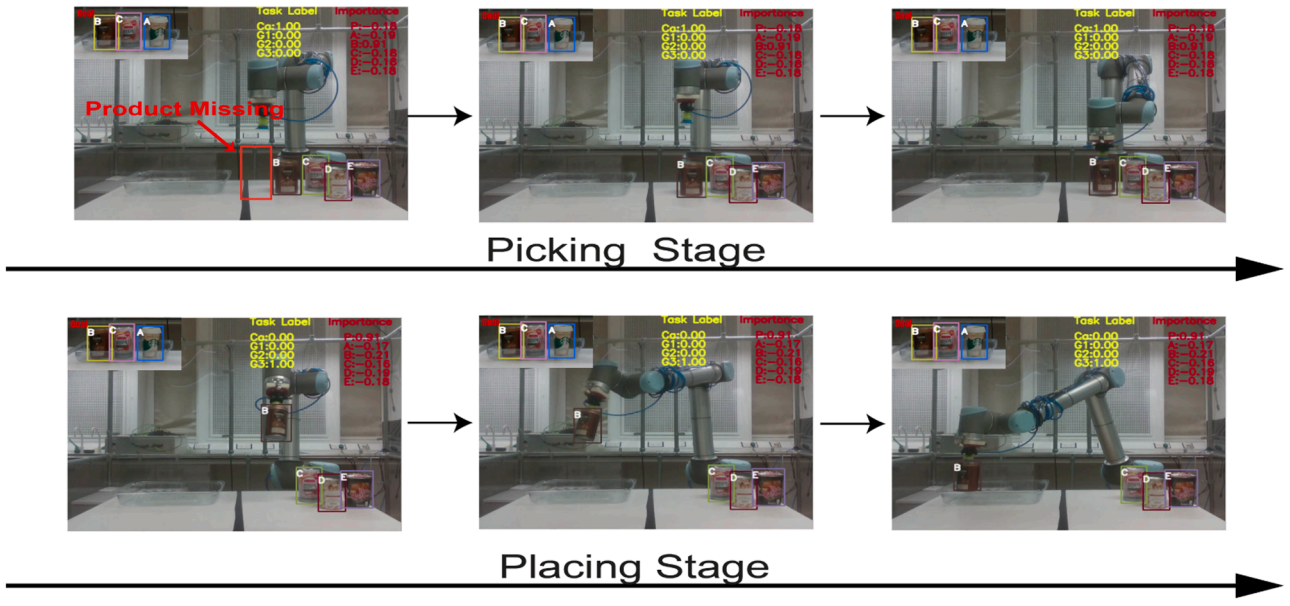
In addressing **Q3**, we have also shown that our approach can deal with observed redundant objects through embedding extra property I in our GNN and thus enabling GNN to focus on a specific subset of products during a mPTO task. Furthermore, our results highlight that our architecture has zero-shot generalization ability regarding different task structures caused by human performance. Our **reasoning module** is robust to handle these variations, as it is able to first distinguish if some of the selected objects have already been packaged through position information, and thus plan the rest of selected objects via learnt task structure (i.e. from smaller numbered object to the bigger one as described in Fig. 6b). Finally, based on the above, our architecture has enabled a cobot to meet the mPTO requirements raised in the introduction section of this manuscript.

In regards to limitations of our work, when comparing our work with [6], their work holds the advantages of being generalizable to the diverse geometric shapes of final goals while ours is constrained to the pose-specified labels in order to enable the use of our low-level **motion module**. Nevertheless, our **reasoning module** is more capable of handling different task structures and thus it is more possible to cater for the varying and different preferences of humans in a Human-Robot Collaboration scenario.

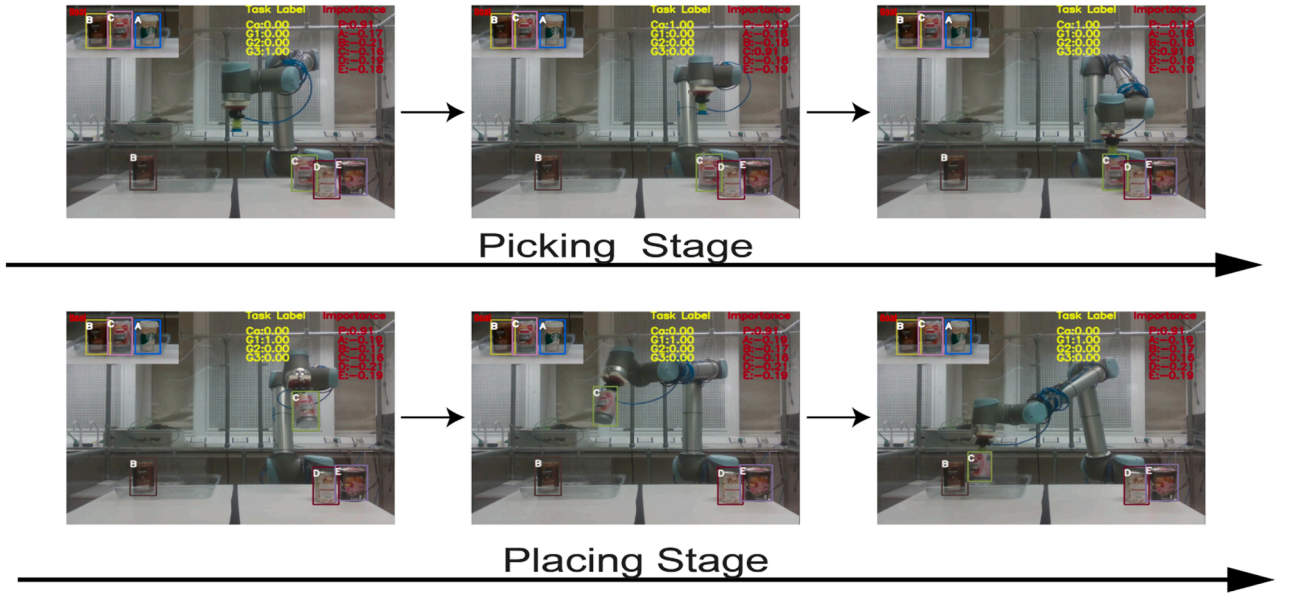
Another limitation for our **motion module** is that the sub-goal planner is unable to handle too large variations in observations as they are multivariate Gaussian distributions and may lead to an inefficient plan to pick the object precisely at its surface. Moreover, the neural dynamic planner only considers the 3D positions of the robot's end-effector. This is less important when dealing with rigid objects but becomes more important if the objects to be manipulated are semi-rigid or soft.

5. Conclusion

In this work, we designed and developed a novel integrated task and motion planning (TAMP) architecture to tackle the adaptive mPTO problem with expert demonstrations. By taking inspiration from the way humans manipulate objects, we separate an expert demonstration into **picking** and **placing** stages.



(a) Second product manipulation



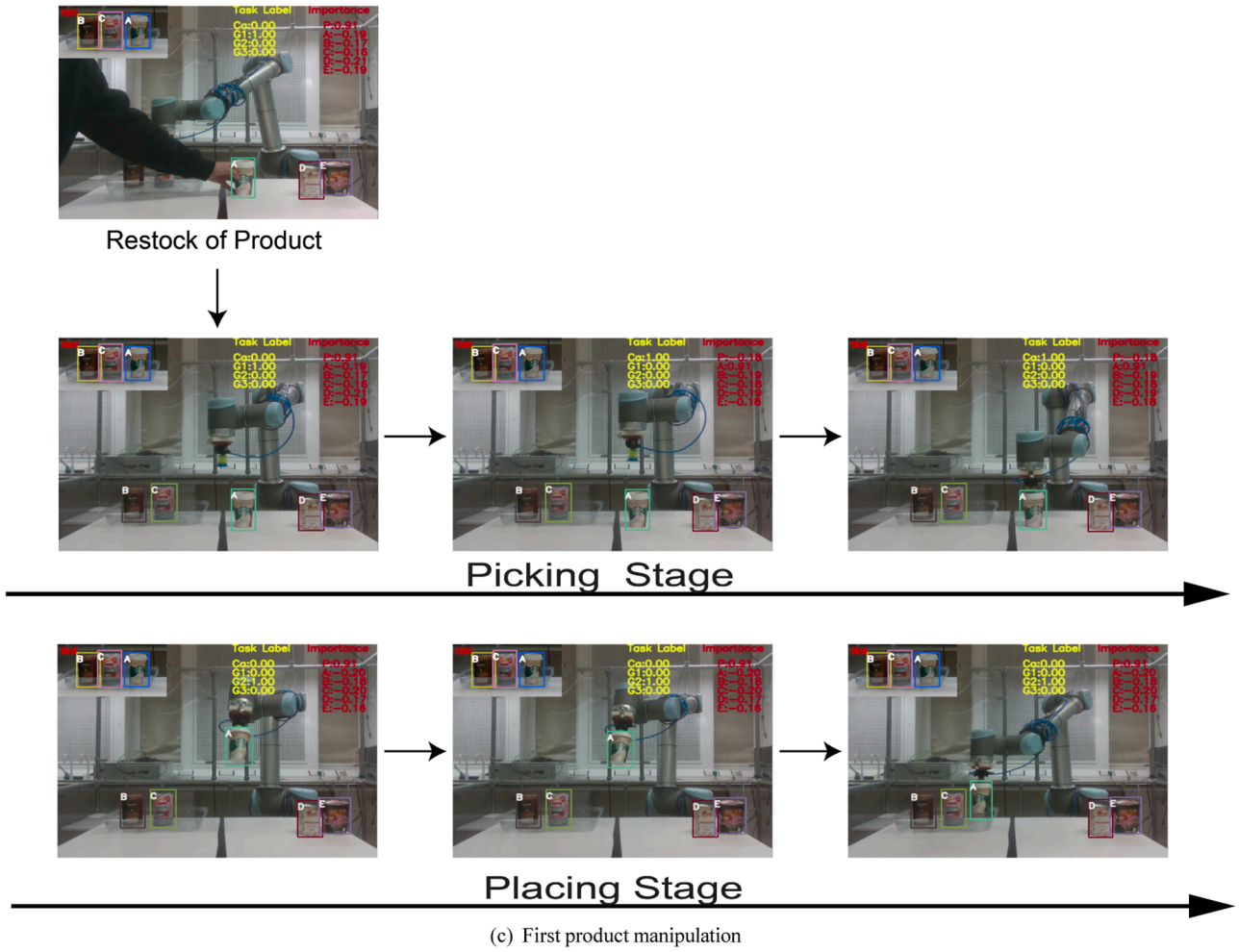
(b) Third product manipulation

Fig. 19. Physical experiment for 3 out of 5 in **CASE 1**, where some products are out of stock while cobot needs to firstly pack the rests and come back to pack the restocked product.

We build a GNN-based high-level **reasoning module** that can identify the important features and task stages. Compare to previous works [6,22], our **reasoning module** does not need scaffolding code in the form of if-else structures to support the distinguishing of variations in both diverse goal configurations and task stages. It also shows zero-shot generalization to handle variant task structures without any additional training. This suggests its potential for use in HRC scenarios where different preferences of human workers need to be taken into consideration. From the **reasoning module**, we can efficiently identify the necessary features and task stages for use by a **motion module** to generate low-level motions.

Our **motion module** utilizes a two-step structure that can first produce sub-goals through conditioned variational inference as well as produce final joint actions by modeling expert preference through a simple neural network. By accessing the task labels, our sub-goal planner has also shown the ability to efficiently integrate diverse demonstrations for different picking and placing stages. It is also able to generalize trajectory production regarding variations in positions for both initial and goal objects' positions. Lastly, our TAMP architecture also has the advantage of being sample-efficient during training and testing in both simulation and physical experiments.

In future work, we mainly want to address the problem of when a product's orientation needs to be considered during its manipulation. This is because our current approach only focuses on the 3D position information in both modules. Also, we would like to investigate synchronous and simultaneous random movements during human



(c) First product manipulation

Fig. 19. (*continued*).

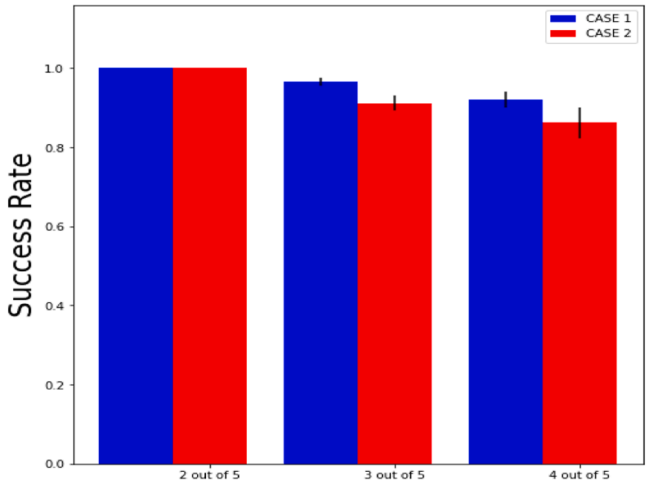


Fig. 20. We consider a practical problem where customers can have multiple choices regarding the number of selected products. We train our approach with **2 out of 5** products. It can generalize to **3 out of 5** products with an average SR of 0.97 and 0.91 as well as generalize to **4 out of 5** with an average SR of 0.92 and 0.86 in both **CASE 1** and **CASE 2**.

collaboration in an HRC scenario as well as what effects this will introduce to the final product configurations. This will lead to safe and adaptive solutions in HRC scenarios (Fig. 4).

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

We would like to acknowledge the support of the Engineering and Physical Sciences Research Council (EPSRC) funding: DigiCORTEX (EP/W014688/1), for the work carried out in this manuscript.

References

[1] S. Benjaafar and M. Elhafsi, "Production and inventory control of a single product assemble-to-order system with multiple customer classes," Manage. Sci., vol. 52, no. 12, pp. 1896–1912, 2006.
[2] M. Mansouri, F. Pecora, P. Schüller, Combining Task and Motion Planning: Challenges and Guidelines, Front. Robot. AI 8 (2021) 1–12. May.
[3] International F. of Robotics, Demystifying Collaborative Industrial Robots, Int. Feder. Robot. (2019) 2–3. no. October.
[4] L.P. Kaelbling, T. Lozano-Pérez, Hierarchical task and motion planning in the now, in: 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 1470–1477, may.

[5] D. McDermott, M. Ghallab, A.E. Howe, C.A. Knoblock, A. Ram, M.M. Veloso, D.S. Weld, and D.E. Wilkins, "PDDL-the planning domain definition language," 1998.

[6] Y. Lin, A.S. Wang, E. Undersander, A. Rai, Efficient and Interpretable Robot Manipulation with Graph Neural Networks, IEEE Robot. Autom. Lett. 7 (2) (2022) 2740–2747.

[7] H. Ravichandar, A.S. Polydoros, S. Chernova, A. Billard, Recent Advances in Robot Learning from Demonstration, Annu. Rev. Control. Robot. Auton. Syst. 3 (1) (2020) 1–34.

[8] E. Erdem, V. Patoglu, P. Schüller, T. Mancini, M. Maratea, F. Ricca, A Systematic Analysis of Levels of Integration between High-Level Task Planning and Low-Level Feasibility Checks, AI Commun. 29 (2016) 319–349, jan.

[9] Maria Fox, Derek Long, PDDL2.1: An extension to PDDL for expressing temporal planning domains, J. Artif. Intellig. Res. 20 (2003) 1–48.

[10] D. Holler, G. Behnke, P. Bercher, S. Biundo, H. Fiorino, D. Pellier, R. Alford, HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems, in: AAAI 2020 - 34th AAAI Conference on Artificial Intelligence, 2020, pp. 9883–9891.

[11] M. Gharbi, R. Lallement, R. Alami, Combining symbolic and geometric planning to synthesize human-aware plans: Toward more efficient combined search, in: IEEE International Conference on Intelligent Robots and Systems, 2015, pp. 6360–6365, vol. 2015-December.

[12] C. Yu, Y. Ji, G. Qi, X. Gu, L. Tao, Group-based production scheduling for make-to-order production, J. Intell. Manuf. 26 (3) (2015) 585–600.

[13] L.S. de Mello, A.C. Sanderson, AND/OR graph representation of assembly plans, IEEE Trans. Rob. Autom. 6 (1990) 188–199, apr.

[14] R. Zhang, J. Lv, J. Li, J. Bao, P. Zheng, T. Peng, A graph-based reinforcement learning-enabled approach for adaptive human-robot collaborative assembly operations, J. Manuf. Syst. 63 (2022) 491–503. April.

[15] S. Pirk, K. Hausman, A. Toshev, M. Khansari, Modeling Long-horizon Tasks As Sequential Interaction Landscapes, CoRL, 2020, pp. 1–14.

[16] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, G. Monfardini, The Graph Neural Network Model, IEEE Trans. Neural Netw. 20 (1) (2009) 61–80.

[17] B. Hayes, B. Scassellati, Autonomously constructing hierarchical task networks for planning and human-robot collaboration, in: 2016 IEEE International Conference on Robotics and Automation (ICRA), 2016, pp. 5469–5476, may.

[18] Z. Su, O. Kroemer, G.E. Loeb, G.S. Sukhatme, S. Schaal, Learning Manipulation Graphs from Demonstrations Using Multimodal Sensory Signals, in: 2018 IEEE International Conference on Robotics and Automation (ICRA), 2018, pp. 2758–2765, may.

[19] D.A. Huang, S. Nair, D. Xu, Y. Zhu, A. Garg, L. Fei-Fei, S. Savarese, J.C. Niebles, Neural task graphs: Generalizing to unseen tasks from a single video demonstration, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2019, pp. 8557–8566. -June2019.

[20] Y. Ye, D. Gandhi, A. Gupta, S. Tulsiani, Object-centric Forward Modeling For Model Predictive Control, CoRL, 2019, pp. 1–13.

[21] P. Battaglia, R. Pascanu, M. Lai, D. Rezende, K. Kavukcuoglu, Interaction Networks For Learning About objects, Relations and Physics, Advances in Neural Information Processing Systems, 2016, pp. 4509–4517.

[22] T. Silver, R. Chitnis, A. Curtis, J. Tenenbaum, T. Lozano-Pérez, L.P. Kaelbling, Planning with Learned Object Importance in Large Problem Instances using Graph Neural Networks, in: 35th AAAI Conference on Artificial Intelligence, AAAI 2021 13B, 2021, pp. 11962–11971.

[23] S. Calinon, F. Guenter, A. Billard, On Learning the Statistical Representation of a Task and Generalizing it to Various Contexts, Proc. 2006 IEEE Int. Conf. Robot. Automat. (2006) 2978–2983. May.

[24] L. Rozo, S. Calinon, D.G. Caldwell, P. Jim, Learning Physical Collaborative Robot Behaviors From Human Demonstrations, IEEE Trans. Rob. 32 (3) (2016) 513–527.

[25] D. Vogt, S. Stepputtis, S. Grehl, B. Jung, H. Ben Amor, A system for learning continuous human-robot interactions from human-human demonstrations, in: Proceedings - IEEE International Conference on Robotics and Automation, 2017, pp. 2882–2889.

[26] S. Calinon, A tutorial on task-parameterized movement learning and retrieval, Intell. Serv. Robot. 9 (1) (2016) 1–29.

[27] Y.Q. Wang, Y.D. Hu, S.E. Zaatari, W.D. Li, Y. Zhou, Optimised Learning from Demonstrations for Collaborative Robots, Rob. Comput. Integr. Manuf. 71 (2021) no. March.

[28] M. Gharbi, R. Lallement, R. Alami, Combining symbolic and geometric planning to synthesize human-aware plans: toward more efficient combined search, in: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015, pp. 6360–6365.

[29] A. Orthey, M. Toussaint, N. Jetchev, Optimizing motion primitives to make symbolic models more predictive, in: Proceedings - IEEE International Conference on Robotics and Automation, 2013, pp. 2868–2873.

[30] O. Kroemer, C. Daniel, G. Neumann, H. Van Hoof, J. Peters, Towards learning hierarchical skills for multi-phase manipulation tasks, in: Proceedings - IEEE International Conference on Robotics and Automation, 2015, pp. 1503–1510, 2015-June, no. June.

[31] R. Lioutikov, G. Maeda, F. Veiga, K. Kersting, J. Peters, Inducing Probabilistic Context-Free Grammars for the Sequencing of Movement Primitives, in: 2018 IEEE International Conference on Robotics and Automation (ICRA), 2018, pp. 5651–5658, may.

[32] G. Konidaris, S. Kuindersma, R. Grupen, A. Barto, Robot learning from demonstration by constructing skill trees, Int. J. Rob. Res. 31 (3) (2012) 360–375.

[33] S. Paul, J. van Baar, A.K. Roy-Chowdhury, Learning from trajectories via subgoal discovery, Adv. Neural. Inf. Process. Syst. 32 (2019) 1–11. NeurIPS.

[34] T. Jurgenson, E. Groshev, and A. Tamar, "Sub-Goal Trees – a Framework for Goal-Directed Trajectory Prediction and Optimization," 2019.

[35] X. Pan, Y. Shen, Human-interactive subgoal supervision for efficient inverse reinforcement learning, in: Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems 2, AAMAS, 2018, pp. 1380–1387.

[36] A. Nagabandi, G. Kahn, R.S. Fearing, S. Levine, Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning, in: Proceedings - IEEE International Conference on Robotics and Automation, 2018, pp. 7579–7586.

[37] C.R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L.P. Kaelbling, and T. Lozano-Perez, "Integrated Task and Motion Planning," 2021.

[38] K. Fang, Y. Zhu, A. Garg, S. Savarese, L. Fei-Fei, Dynamics Learning with Cascaded Variational Inference for Multi-Step Manipulation, in: Conference on Robot Learning (CoRL), 2019.

[39] C.R. Garrett, T. Lozano-Pérez, L.P. Kaelbling, Sampling-based methods for factored task and motion planning, Int. J. Rob. Res. 37 (13-14) (2018) 1796–1825.

[40] J. Kurosu, A. Yorozu, M. Takahashi, Simultaneous dual-arm motion planning for minimizing operation time, Appl. Sci. (Switzerland) 7 (12) (2017).

[41] J.K. Behrens, R. Lange, M. Mansouri, A constraint programming approach to simultaneous task allocation and motion scheduling for industrial dual-arm manipulation tasks, in: Proceedings - IEEE International Conference on Robotics and Automation, 2019, pp. 8705–8711, vol. 2019-May.

[42] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," in Advances in Neural Information Processing Systems (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[43] M. Fey, J.E. Lenssen, Fast Graph Representation Learning with PyTorch Geometric. ICLR Workshop on Representation Learning On Graphs and Manifolds, 2019, pp. 1–9.

[44] D.P. Kingma, M. Welling, Auto-Encoding Variational Bayes, 6, CoRR, 2014 vol. abs/1312.

[45] E. Rohmer, S.P. Singh, M. Freese, V-REP: A versatile and scalable robot simulation framework, in: IEEE International Conference on Intelligent Robots and Systems, 2013, pp. 1321–1326.

[46] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," 2018.

[47] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: An imperative style, high-performance deep learning library, Adv. Neural. Inf. Process. Syst. 32 (2019).

[48] R. Ying, D. Bourgeois, J. You, M. Zitnik, J. Leskovec, GNNExplainer: Generating explanations for graph neural networks, in: Advances in Neural Information Processing Systems 32, 2019.

[49] J. Wang, An Intuitive Tutorial to Gaussian processes Regression, arXiv preprint, 2020.

[50] V. Joukov, D. Kulic, Gaussian process based model predictive controller for imitation learning, in: 2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids), 2017, pp. 850–855.

[51] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, in: 4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings, 2016.

[52] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, W. Zaremba, Hindsight experience replay, in: Advances in Neural Information Processing Systems 2017, 2017, pp. 5049–5059. -Decem, no. Nips.

[53] J.J. Kuffner, S.M. La Valle, RRT-connect: an efficient approach to single-query path planning, in: Proceedings - IEEE International Conference on Robotics and Automation 2, 2000, pp. 995–1001, no. April.

[54] I.A. Sucan, M. Moll, L.E. Kavraki, The Open Motion Planning Library, IEEE Robot. Automat. Mag. 19 (4) (2012) 72–82.

[55] A. Koubaa, Robot Operating System (ROS): The Complete Reference, 1st ed., 2, Springer Publishing Company, Incorporated, 2017. Volume.

[56] M. Gorner, R. Haschke, H. Ritter, J. Zhang, Moveit! task constructor for task-level motion planning, in: Proceedings - IEEE International Conference on Robotics and Automation 2019-May, 2019, pp. 190–196.