北京大学 人工智能研究院
INSTITUTE FOR ARTIFICIAL INTELLIGENCE, PEKING UNIVERSITY

PKU-IAI Technical Report: TR-PKU-IAI-2024-0001

Towards a Unified Probabilistic PDDL Solver in the Block Stacking Domain

Penglin Cai[†] **Shanglin Wu**[†] **Xingping Yu**[†] **Chenhao Zhou**[†]
[†] Yuanpei College, Peking University
`{cpl, jasonwu1017, 2100017812, zhouch}@stu.pku.edu.cn`

Abstract

One-shot imitation learning has aroused great interest in the efficiency of data and demonstrations. However, some problem settings such as block stacking require precise planning and thus exhibiting extremely low tolerance for errors. To tackle such a challenge, we use a probabilistic continuous planning method as a relaxation of the symbolic state representation. We implement a pipeline consisting of a Symbol Grounding Network (SGN) module and a Continuous Planner (CP) module, building a probabilistic PDDL solver in the domain of block stacking problems. By constructing an agent based on large language models, we demonstrate that our probabilistic PDDL solver greatly outperforms the baseline. We also provide variants of the solver, from which we show the effectiveness of our SGN module and CP module respectively through an ablation study. All these methods are tested on our hand-crafted lightweight environment, which is convenient for transplantation. We open-source our environments, models, as well as codes, and hope to be of use for further research. Our code is made available at https://github.com/Intellouis/Probabilistic-PDDL-Solver.

1 Introduction

Abstract reasoning is definitely an important ability that an intelligent agent must possess to solve diverse tasks in the real world. In a typical setting, agents receive the instructions of the tasks, make plans, and conduct executions while interacting with the external environment. Among these steps, the process of planning greatly relies on the ability of abstract reasoning. Hence, it is essential to build intelligent agents that can reason with abstract concepts.

Symbol grounding [8] is always a significant problem in artificial intelligence research. It is vital to make intelligent agents able to associate abstract symbols with their actual references and logical meanings. In recent studies, symbolic planning has been widely adopted to test such capabilities of abstract reasoning, especially in the robotics domain. Planning Domain Definition Language (PDDL) [14] was proposed to decompose the planning problem into two major parts - domain description and problem description. CoSTAR [10] was benchmarked as a dataset solving learning problems with workspace constraints in the domain of block stacking. Li *et al.* [12] used a reinforcement learning system to tackle the challenges of block stacking problems without demonstrations to imitate. Huang *et al.* [9] proposed a novel method in which a Symbol Grounding Network (SGN) and a Continuous Planner (CP) were combined in the cases where probabilistic representations of the symbol groundings were utilized for planning.

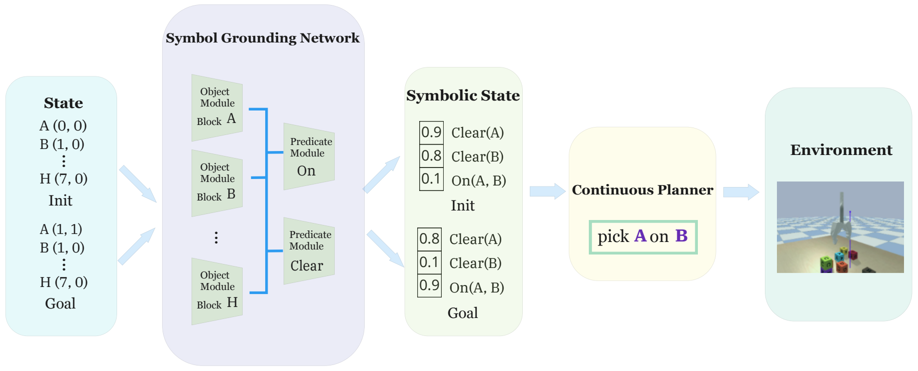


Figure 1: Our pipeline of the probabilistic PDDL solver. We use an upstream Symbol Grounding Network (SGN) to convert the initial state into a continuous embedding representation, which is the grounded symbolic state. Then the downstream Continuous Planner (CP) conducts planning over the continuous representation and gives the next action. The predicted actions are executed in our hand-crafted lightweight environment to finish the block stacking task.

Following Huang *et al*. [9], we build up a lightweight environment, implement a pipeline of computational framework, and examine various approaches to solve block stacking tasks in the setting of one-shot imitation learning. In this paper, we formulate the one-shot imitation learning (also referred to as meta-learning) of long-horizon tasks into a planning problem and implement a hierarchical model with two components to tackle such challenges. Specifically, the upstream Symbol Grounding Network (SGN) converts an observation or absolute state into a probabilistic distribution over the symbolic states, while the downstream Continuous Planner (CP) takes in the probabilistic representation and outputs a sequence of atomic actions as the plan.

The necessity of the continuous planner lies in that discrete representations may result in failed planning procedures. According to Huang *et al*. [9], the discrete symbol grounding can be prone to error with limited training data in the setting of one-shot imitation learning, since the representation in the discrete form can be contradictory itself. Subsequently, wrong representations will lead to symbolic planning failures. To tackle this challenge, we follow Huang *et al*. [9] and adopt SGN to construct probabilistic representations of symbolic states, followed by a continuous relaxation of the classical symbolic planner to plan with the probabilistic representations. Through this framework, we manage to address one-shot imitation learning problems in the block stacking domain, which is extremely data efficiency.

Since we are required to reproduce the paper of Huang *et al*. [9], it is necessary to distinguish the parts borrowed from the original paper and the parts of our own design. Details are listed in Appendix A.

Our main contributions are threefold:

- We build up a lightweight environment, which simplifies the state and action space, aiming to focus on the approaches in planning problems.
- We implement the whole pipeline of the hierarchical planning model, which contains a high-level Symbol Grounding Network and a low-level Continuous Planner. Overall, we manage to reproduce the original paper [9].
- We explore multiple approaches to tackle the symbolic planning challenges and compare their performances via solid experiments. By open-sourcing our environment, models, and codes, we hope to be of inspiration for further research.

2 Preliminaries

2.1 Problem Formulation

2.1.1 Block Stacking Problems

In the problem setting, we have N blocks numbered from 1 to N. Each block is placed on the desktop initially, and the goal is to make the state satisfy a set of stacking constraints by moving the blocks sequentially.

Therefore, we formulate the block stacking tasks as a Constraint Satisfaction Problem (CSP):

- Given a set of N blocks $B = \{b_1, b_2, ..., b_N\}$.
- Each state consists of the locations of the blocks: $s_t = \{X_1, X_2, ..., X_N\}, 0 \leq t \leq T$.
- Each action a_t is a function mapping from previous state to the next state: $s_{t+1} = a_t(s_t), \forall t$.
- Given a set of m constraints $C = \{c_1, c_2, ..., c_m\}$.
- Objective: find a sequence of actions $[a_0, a_1, ..., a_{T-1}]$, such that the final state $s_T = a_{T-1}(s_{T-1}) = ... = (a_{T-1} \circ a_{T-2} \circ ... \circ a_0)(s_0)$ satisfies the constraints $C = \{c_1, c_2, ..., c_m\}$.

2.1.2 One-Shot Imitation Learning

One-Shot imitation learning [4] is an important policy of imitation learning. The goal is to optimize the performance of the learned strategy when faced with a new and unknown task. By observing the current state and a successful demonstration, the model learns the strategy to solve the problem, and we expect the strategy to achieve good performance without any additional interaction.

We formulate the setting of one-shot imitation learning in a certain domain following Huang *et al.* [9]. Let $\mathcal{T}$ be the set of tasks in this domain. For a certain task $\tau \in \mathcal{T}$, there is a successful T-step demonstration $d^\tau = [d^\tau_1, d^\tau_2, ..., d^\tau_T]$, where each d^τ_i denotes a state-action pair. The aim is to train a model $\psi(\cdot)$ which outputs a policy $\pi^\tau = \psi(d^\tau)$ to complete new tasks in this domain.

We divide the tasks $\mathcal{T}$ into $\mathcal{T}_{train}$ and $\mathcal{T}_{test}$, and use $\mathcal{T}_{train}$ to train our model $\psi(\cdot)$. In the hypothesis, we assume $\mathcal{T}_{train}$ is sufficient so that $\psi(\cdot)$ can generalize to new tasks in $\mathcal{T}_{test}$.

2.2 Symbolic Planning

Symbolic planners have been widely employed in traditional AI literature, for their exceptional interpretability and remarkable performance in solving complex reasoning tasks by planning within a pre-defined specific symbolic space. In this paper, we retained the approach from the original literature [9], using the PDDL [14] to specify the block stacking problem.

A PDDL task generally includes two main components: a domain file and a problem file. The domain file precisely defines the objects involved in a planning problem, the predicates expressing attributes of these objects, and a set of operators affecting changes in the states of these objects. The problem file illustrates the initial and goal states, both defined through a set of true ground atoms, which entail the combinations of predicates and objects.

Specifically addressing the block stacking tasks, we formulate it as a simplified PDDL planning problem (S_0, S_G, O), given the initial state S_0, the goal state S_G, and a set of operators O. Each state S is defined by a set of ground atoms that are true (*e.g.*, {`On(A,B)`, `Clear(A)`}), and each operator $o \in O$ is defined as $o = ($`name(`o`)`, `precondition(`o`)`, `effect(`o`)`$)$, where both the `precondition` and `effect` are formally a set of ground atoms. The `name` is employed to denote the solution to the planning problem, which is an action plan $\Pi = [$`name(`o_1`)`, ..., `name(`o_N`)`$]$.

2.3 Related Work

One-shot imitation learning. In recent years, imitation learning has aroused great interest in the research field, and especially for tackling challenges in reinforcement learning and robotics manipulation. Recent work has utilized imitation learning to play video games [1, 19], to master dexterous grasping[11, 20], as well as to produce household robots [6]. Delving further into this field, one-shot imitation learning [4] is proposed as a more challenging problem, since the efficiency of limited data and demonstrations. Finn *et al.* [5] studied on visual meta-learning with raw images as inputs, and can be adapted to new tasks from a single vision demonstration. To further simplify the acquisition of high-quality demonstrations, Bonardi *et al.* [2] leveraged sim-to-real transfer on humans, utilizing simulated data to conduct one-shot imitation learning to train the control policies. Different from traditional imitation learning, we divide the one-shot demonstrations into states to train the SGN module, instead of a policy. Our training aims at obtaining a plausible probabilistic representation of the state, rather than the action of the next step.

Planning domain definition language (PDDL). Planning domain definition language (PDDL) [14] is widely applied in simulated environments for its convenience of representations. PDDL combines the understanding of natural language (for humans) and the structural readability (for machines and models), which makes it easy to use in the domain of planning and decision making. In recent years, a variety of environments and simulators relevant to PDDL have been developed. PDDLGym [18] was proposed as a framework to automatically construct environments like OpenAI Gym from PDDL domains and problems. ALFWorld [17] used PDDL to describe the scenes and states in the environment, based on which a unification and alignment of ALFRED [16] and the proposed TextWorld was realized.

Embodied agents and robots in the domain of block stacking tasks. Embodied AI and robots in real life have aroused another wave of artificial intelligence research, becoming a promising area of development. We focus on the embodied agents in block stacking domains. Macias *et al*. [13] introduced a robust system of image-based block pick-up and stacking, from the aspect of vision. Cannizzaro *et al*. [3] combined causal inference with block stacking tasks, reasoning about the optimal selection of next action over multiple candidates. Zhu *et al*. [22] trained an end-to-end policy with raw images as inputs in the block stacking tasks, which outperformed various baselines trained with deep reinforcement learning or behavior cloning alone. In this paper, we study the block stacking tasks under the setting of one-shot imitation learning, and explore the use of probabilistic representation and planning.

3 Methods

Our hierarchical probabilistic PDDL solver framework contains a high-level Symbol Grounding Network (SGN) and a low-level Continuous Planner (CP), as shown in Figure 1. In this section, we introduce the structures and algorithms used in these two components. Specifically, we introduce our representation of continuous symbolic states in Sec. 3.1, the SGN module in Sec. 3.2, the CP module in Sec. 3.3, and the training and inference of our pipeline in Sec. 3.4.

3.1 Continuous Symbolic State Representation

In our block stacking environment, there are eight blocks in total, numbered from `A` to `H`. According to the relationship of locations and arrangements, we classify the predicates into two categories - `On(A,B)` and `Clear(A)`. Therefore, we can describe a state within 64 atomic propositions - 56 of which are used to depict `On(i,j)` and the remaining 8 propositions are used to depict `Clear(i)`. To sum up, we represent the state using a 64-dimensional vector, each element of which depicts the probability that the corresponding proposition is true. Under this representation, a 64-dimensional vector, 8 elements of which are 1 and the other 56 elements are 0, can determine a state uniquely.

Different from the discrete vector, the continuous representation provides a perspective of probabilistic planning. In the following sections, we present the effectiveness of continuous planning compared to discretization.

3.2 Symbol Grounding Network (SGN)

Following Huang *et al*. [9], our Symbol Grounding Network consists of two categories of modules - object module and predicate module, and both modules are 2-layer MLPs. We use 8 object modules to represent 8 blocks, and use 2 predicate modules to represent 2 predicates.

For each input state, the state will pass the 8 object modules simultaneously to get 8 block embeddings. Then these 8 embeddings will form 56 pairwise combinations and pass the predicate module depicting `On(i,j)`, to get the former 56 elements in the symbolic state representation vector. These 8 block embeddings will also pass the other predicate module depicting `Clear(i)`, to get the latter 8 elements in the vector. More details about the structure of the SGN module can be found in Appendix C.1.

Following the setting of one-shot imitation learning, we disassemble the 2,000 demonstrations (episodes) into over 8,000 states, and label these states with the ground truth of the vectors (all ground truth vectors only contain 0-1 elements; 0 indicating `False` for the corresponding proposition and 1 indicating `True`). Then these state-label pairs are used to train our SGN module.

3.3 Continuous Planner

Through the Symbol Grounding Network (SGN), we have obtained a probability distribution of the corresponding symbolic states from observation (object states). It is necessary to adapt the classic symbolic planner, which focuses on handling discrete symbolic states (usually the set-theoretic representation [7]), to accommodate current continuous symbolic states. A straightforward approach involves discretizing the current output, for instance, by truncating it with a threshold. However, due to the grounding errors of SGN under low-sample conditions of one-shot imitation learning, simply discretizing output may result in the generation of invalid symbolic states before the planning phase.

Handling invalid states can be achieved through the introduction of manual rules, yet this may lead to the loss of the symbolic planner's generalization across different tasks. Following Huang *et al.* [9], we implement the continuous relaxation method to achieve a continuous planner, which successfully accommodates the continuous representation of symbolic states. In addition, we simultaneously optimize some specific details of the continuous planner to enhance planning efficiency in the block stacking task.

In essence, we apply continuous relaxation and some specific adjustments to a series of the traditional discrete planner to handle continuous representation of the symbolic states outputted from the SGN module. Our iterative planning process contains:

1. Representing current state;
2. Determine a set of applicable actions;
3. Select an applicable action;
4. Apply the action and change the current state;
5. Stop when arriving at the goal state.

Continuous State Representation The definition of the state representation for the continuous planner is delineated as expounded in Sec. 3.1. We substitute previous set-theoretic representation with the probability distribution that SGN outputs. Specifically, what SGN outputs can be seen as the probability of the each ground atom g being true. Assuming conditional independence among the ground atoms, the probability for each symbolic state S can be simply computed through the product of the probability for all the ground atoms. Formally, for current distribution Z over all symbolic states, the probability of one symbolic state S_0 can be calculated as follows:

$$P(S_0) = \prod_{g \in S_0} P_Z(g), \tag{1}$$

where P_Z denotes the continuous embedding representation of the current state, which is a 64-dimensional vector. Since g is a ground atom, $P_Z(g)$ is an element of the embedding vector.

Action Applicability Under continuous condition, the applicability of an action a is decided by the probability of its `precondition(`a`)`, defined in Sec. 2.2. The action is "more applicable" if its applicability is higher. Formally, for current distribution Z over all symbolic states, the applicable probability App of an action a can be calculated as:

$$App(a) = \prod_{g \in pre(a)} P_Z(g), \tag{2}$$

where $pre(a)$ denotes the set of preconditions of action a.

Action Selection The selection of action involves not only its applicability but also its potential positive contribution towards attaining the goal state. A correct action should be more applicable under current state, and its application results in another modified state which, in comparison to current state, should be closer to the goal state. We define the contribution score of action a as follows:

$$Score(a) = \gamma App(a) + (1-\gamma)\|Z_a - Z_g\|, \tag{3}$$

where $\gamma \in (0,1)$, Z_a denotes the shifted distribution after applying a (more details in the next paragraph), and Z_g denotes the goal distribution. Thus we can select action by ranking the contribution score of all available actions. $\|Z_a - Z_g\| = \sum_g \|P_{Z_a}(g) - P_{Z_g}(g)\|$ denotes the norm distance between two vectors, and we choose to use the Euclidean distance. The inclusion of the distance term differs from the original literature [9].

Action Application In contrast to discrete planner, which simply modify the set of ground atoms based on the `effect` defined in Sec. 2.2 for state transition, continuous planner involves the shift of distribution over states (from Z to Z'). Specifically, we modify probabilities associated with ground atoms. Given a selected action a and the corresponding `effect(`a`)`, the ground atom g can be categorized into three distinct classes: (i) activated, (ii) deactivated, and (iii) neutral. Each category of ground atoms is associated with distinct updating equations for their probabilities. In the context of a given action a:

(i) g is activated. This signifies that the application of a causes a positive effect on the transition from $False$ to $True$ of the ground atom g. The new probability of g in the new distribution Z' is calculated as follows:

$$P_{Z'}(g) = P_Z(g) + (1 - P_Z(g)) \prod_{\hat{g} \in pre(a)} P_Z(\hat{g}). \tag{4}$$

(ii) g is deactivated. This indicates that the application of a leads to a negative transition of the ground atom g, which is from $True$ to $False$. The new probability of g in the new distribution Z' is calculated as:

$$P_{Z'}(g) = P_Z(g) - \prod_{\hat{g} \in pre(a)} P_Z(\hat{g}). \tag{5}$$

(iii) g is neutral. This means that the application of a has no effect on g, the probability of which remains unchanged:

$$P_{Z'}(g) = P_Z(g). \tag{6}$$

Goal Matching For continuous representation of symbolic states, we opt to find an objective distribution that matches the goal. For the 64-dimensional vector that represent the symbolic state (in Sec. 3.1), any metric describing vector distance can be utilized to assess the match of two symbolic states, and we choose to use the Euclidean distance.

In block stacking tasks, the action space contains 56 actions named "put i on j", where i, j denotes the label of 8 blocks. The `precondition` of each action is the set `{Clear(`i`), Clear(`j`)}`. The `effect` of each action would be `{not(Clear(`j`)), On(`i,j`)}`. By iteratively implementing the aforementioned five steps and continuously adjusting the 64-dimensional vector, the continuous planner seeks to identify a sequence of actions that can lead to a state distribution closest to the goal distribution.

3.4 Training and Inference

In this section, we list the pseudo codes of training and inference phases, which are borrowed from the replicated paper [9].

Training Phase The continuous planner does not require training, and our focus lies solely on training the SGN module. We represent demonstrations as d_t, and the function $ContinuousSymbolicStates()$ computes the symbolic state of d_t. For training, we employ the binary cross entropy with logits as the loss function and utilize the Adam optimizer. Specifics and details are shown in Alg. 1. We provide the loss curves during the training phase in Appendix D.1.

Algorithm 1 Training Phase

```
Input: $D_{train} = \{(d_t, a_t)\}$ including demonstration $d_t$ and action $a_t$, operators $O$.
  for $(d_t, a_t)$ in $D_{train}$ do
    $S_t \leftarrow ContinuousSymbolicStates(d_t, a_t, O)$
    for $(d_{t,i}, S_{t,i}) \in d_t, S_t$ do
      $S'_{t,i} \leftarrow SGN(d_{t,i})$
      $L \leftarrow CrossEntropy(S'_{t,i}, S_{t,i})$
      $Adam(L, SGN(\cdot))$
    end for
  end for
```

Inference Phase As elucidated in Sec. 3.3, the continuous planner can generate a series of actions Π based on the initial state S_0 and the desired goal state S_G. Subsequently, the $Step()$ function is employed to execute the actions Π in the environment E. After the execution, the initial state S_0 undergoes an update and transitions to the next state. The entire process operates within a loop, with further details listed in Alg. 2.

Algorithm 2 Inference Phase

```
Input: $D_{inference} = \{d_t\}$, operators $O$, environment $E$, maximum iteration number $N$.
  for $d_t$ in $D_{inference}$ do
    $s_0 \leftarrow Init(E)$
    $S_0 \leftarrow SGN(s_0), S_G \leftarrow SGN(d_t)$
    while $iteration < N$ do
      $\Pi \leftarrow CP(S_0, S_G, O)$
      $s_{cur} \leftarrow Step(E, \Pi)$
      $S_0 \leftarrow SGN(s_{cur})$
    end while
  end for
```

4 Environment and Experiments

4.1 Block Stacking Tasks and Environment

In our block stacking environment, we study one-shot imitation learning using 8 basic blocks as the test-bed. For simplification, we ignore the way-points, traces, and rotations of the end-effector, but focus on the orders and locations of the object manipulation, formulating this process as a planning problem, which is consistent with the problem formulation (Sec. 2.1).

The basic elements of our environment are listed here:

- **Object space:** We have 8 blocks, numbered 1 to 8. These blocks are totally same except for numbers. All the blocks are placed on the desktop initially without stacking.
- **State space:** We provide two versions of state description - both the precise locations of blocks (the coordinate space) and the PDDL descriptions. These coordinate-based state representations form the observation space of SGN, while those language-based descriptions form the observation space of LLM-based agents in Sec. 4.2. We also provide the ground truth embeddings of each state during interactions and rollouts, which are used to determine whether a task is finished (whether the goal is reached).
- **Embedding space:** We use a 64-dimensional vector to represent a state, as figured out in Sec. 3.1. Each element of the vector can serve as the probability of the corresponding proposition (ground atom) being true.
- **Action space:** We use the format of "put i on j" to represent an action, where $i \in \{1, 2, ..., 8\}$ is the source block, and $j \in \{1, 2, ..., 8\}$ is the target position. In the domain of block stacking, we only allow agents placing a block exactly on the top of another block. This representation is consistent with the "grasp" and "place" in Huang *et al.* [9].
- **Goal space:** We also use a 64-dimensional vector to represent the goal state, in which 8 elements are 1 exactly. In each step of a trajectory, the goal vector is used to determine whether the goal has been reached (whether the task is finished).
- **Tasks:** We provide 2,000 different tasks, which are obtained from Xu *et al.* [21]. Each task has a unique goal, forming a diverse set of task cases. 1,900 of these tasks are utilized to train the SGN, while the remaining 100 tasks form the test set.

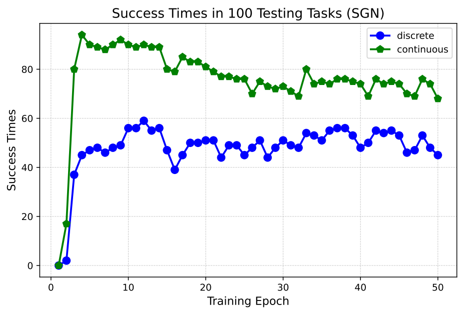
More details about the environment are listed in Appendix B, with a screenshot of our environment showcase.

4.2 Implementations

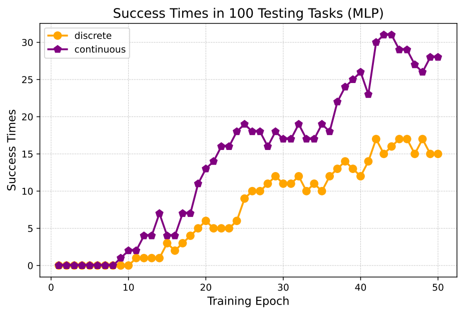
In the block stacking domain, we make various attempts and implement different models to tackle the challenges in the block stacking problems. To be specific, we implement the SGN module and CP module following Huang *et al.* [9], which is the main pipeline. We also provide another method, using a large language model as the agent to directly interact with the environment, which serves as the baseline. Besides, we have tried an RL-based method using DQN algorithm [15], yet it failed on solving these tasks in the setting of sparse reward. In the following, we present the implementation of the former two methods.

Table 1: The success rate of different methods. "SGN" denotes "SGN + CP", "SGN (discrete)" denotes "SGN (discrete) + SP", and the same applies to others. We trained the SGN and the MLP for 50 epochs using over 8,000 states, and combine both networks with or without discretization. The LLM-agent uses GPT-4-1106-preview for planning and decision making.

	SGN	SGN (discrete)	MLP	MLP (discrete)	LLM-agent
Success Rate	68%	45%	28%	15%	45%



(a) Number of success times using SGN.



(b) Number of success times using MLP.

Figure 2: Times of success over the 100 testing tasks. Using SGN (Fig. 2a) and MLP (Fig. 2b) respectively. The results show that SGN surpasses vanilla MLP, and discretization makes no benefit of the performance.

As to the main pipeline, **SGN + CP**, these two modules are respectively introduced in Sec. 3.2 and Sec. 3.3. In the experiments, our SGN module takes the coordinate state as input (Sec. 4.1) and embeds the state into continuous symbolic representations. Then the CP module takes the symbolic state for planning, and outputs the next actions. As a planning algorithm, the CP module can independently solve all the tasks given certain vectors of state representation. Therefore, we collect expert demonstrations only using the CP module. Each trajectory has 4 steps on average, and we divide these 2,000 trajectories into over 8,000 states. The states decomposed from the former 1,900 tasks are used to train the SGN module, while the latter 100 tasks form our testing set. On the other hand, the CP module does not require training, and it is used for inference in downstream tasks, cooperating with the SGN module.

In the meanwhile, we also provide three variants of the main pipeline:

- **SGN (discrete) + Symbolic Planner (SP)**: We discretize the output of the SGN with a threshold of 0.5 and make down-stream planning using a classical symbolic planner.
- **MLP + Continuous Planner (CP)**: A MLP is used to replace the SGN module, both having the same number of layers. This ablation is meant to verify the usefulness of SGN, which is specially designed in the architecture.
- **MLP (discrete) + Symbolic Planner (SP)**: We also discretize the output of the MLP and plan with a classical symbolic planner with the same discretization method.

In terms of the baseline, the LLM-based agent, we utilize the ability of abstract reasoning of existing LLMs. Specifically, we adopt GPT-4-1106-preview, which can stand for the most outstanding one in the existing LLMs. In each step, we provide GPT-4 with the current state and the history information within this episode in the prompt, written in PDDL. We ask the LLM to give the next action, and execute it in the environment. For more details, we provide the prompts used in the LLM-agent in Appendix C.2.

4.3 Results and Analyses

We use over 8,000 states to train the SGN module, as well as the MLP, for 50 epochs. We have 4 variants - SGN + CP, SGN (discrete) + SP, MLP + CP, and MLP (discrete) + SP. Together with the LLM-based agent, we test all the five methods over the 100 testing tasks and calculate the success rate. The results are listed in Tab. 1. For additional results, such as the loss curves during the training phase and the visualization of the predicted embedding vectors, please refer to Appendix D.

The experimental results show that our probabilistic PDDL solver, SGN + CP, can solve most of the tasks, exhibiting the great effectiveness of one-shot imitation learning. Comparing the SGN-based methods and the LLM-agent, it is shown that our PDDL solver outperforms the most outstanding large language model in the domain of block stacking planning tasks. Additionally, the success rate of those methods with MLP are relatively low, indicating the effectiveness of the designed SGN architecture.

4.4 Ablation Study

In the ablation study, we focus on the former four methods (except for baseline). We test all 50 models corresponding to 50 training epochs for the four methods and plot the curve of success times, as shown in Fig. 2a (SGN) and Fig. 2b (MLP). According to the results, we claim that the SGN module surpasses vanilla MLP, and discretization makes no benefit of the performance.

Compared to the two methods using vanilla MLP, the SGN-based methods reached more success times within fewer training epochs, demonstrating the capacity of the designed SGN architecture. Considering the fact that SGN equips each block and each predicate with an MLP module, each MLP is only used for the corresponding block or predicate. Such architecture can effectively capture the features and the hidden information of the input state, thus resulting in better performance.

On the other hand, comparing each single network with or without discretization, we find that discretization seems to have a negative effect on the performance of the whole PDDL solver. This can owe to the ambiguity of the continuous embedding vector. Since we have made each state represented in a continuous embedding, each ground atom has a certain probability to be true. In this case, simplistic discretization may generate invalid symbolic states in the environment. For instance, in the domain of block stacking, the two ground atoms `Clear(B)` and `On(A,B)` cannot be true simultaneously. However, the discretization process cannot guarantee the validity of the continuous embedding representation, and may result in planning failure.

5 Limitations and Future Work

In this work, we reproduce the original paper [9] overall, on the basis of which we also propose our own designs and improvements. Besides, there are also several limitations with our work, remaining to be solved in future work.

On the one hand, we have not delved into diverse combinations of algorithms and methods. For instance, learning from the failure of the attempt of the DQN algorithm, we may try initializing the Q-Network with values learned from imitation learning, which is a combination of imitation pre-training and RL fine-tuning methods.

On the other hand, many various mechanisms and functions are constrained by the hand-crafted lightweight environment. For instance, our current action space is not fine-grained enough, with the absence of detailed waypoints, traces, and rotations of the end-effector. We may consider implementing an embodied environment with physically simulated blocks and robot arms in future work. In this case, we can also use the visualized image observation instead of coordinate values as input to train SGN and to predict the state embedding, which is more consistent with the real world. In that case, the important role and effectiveness of such combinations of SGN and CP will be further highlighted.

Another limitation is the time efficiency. In our block stacking environment, it takes about 10 min to conduct planning on the 100 tasks in the test set. However, since our continuous planner is a search-based algorithm, the time consumed can greatly rely on the state space and action space. Besides, other than the time for planning, we have not considered the time for robotics moving yet. In the real-world robot deployment, the time-consuming of motion can be a bottleneck, making this method much less efficient.

Moreover, it is worth noting that the method of SGN + CP can not only work in the block stacking domain, but can also be extended to other planning problems with continuous state representation. In the future, we may extend this method to other potential applications, such as block sorting, navigation, as well as motion planning.

6 Conclusion

In this paper, we implement a probabilistic PDDL solver within the framework of one-shot imitation learning, incorporating an upstream SGN module and a downstream CP module. We design a lightweight environment to assess the PDDL solver's performance, and compare it with another agent based on LLMs. The experimental results show that our probabilistic PDDL solver outperforms the LLM-agent baseline, tackling the challenges posed by complex planning problems in the block stacking domain. By open-sourcing our environment, models, and codes, we lay a foundation for the research in the area of one-shot imitation learning and probabilistic PDDL solvers, which we hope to be of inspiration for future work.

References

[1] Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654, 2022. 3

[2] Alessandro Bonardi, Stephen James, and Andrew J Davison. Learning one-shot imitation from humans without humans. *IEEE Robotics and Automation Letters*, 5(2):3533–3539, 2020. 3

[3] Ricardo Cannizzaro, Jonathan Routley, and Lars Kunze. Towards a causal probabilistic framework for prediction, action-selection & explanations for robot block-stacking tasks. *arXiv preprint arXiv:2308.06203*, 2023. 4

[4] Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. *Advances in neural information processing systems*, 30, 2017. 3

[5] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. In *Conference on robot learning*, pages 357–368. PMLR, 2017. 3

[6] Zipeng Fu, Tony Z Zhao, and Chelsea Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. *arXiv preprint arXiv:2401.02117*, 2024. 3

[7] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. The Morgan Kaufmann Series in Artificial Intelligence. Elsevier Science, 2004. ISBN 9781558608566. 5

[8] Stevan Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1-3): 335–346, 1990. 1

[9] De-An Huang, Danfei Xu, Yuke Zhu, Animesh Garg, Silvio Savarese, Li Fei-Fei, and Juan Carlos Niebles. Continuous relaxation of symbolic planner for one-shot imitation learning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2635–2642, 2019. 1, 2, 3, 4, 5, 6, 7, 9, 12

[10] Andrew Hundt, Varun Jain, Chia-Hung Lin, Chris Paxton, and Gregory D Hager. The costar block stacking dataset: Learning with workspace constraints. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1797–1804. IEEE, 2019. 1

[11] Liyiming Ke, Jingqiang Wang, Tapomayukh Bhattacharjee, Byron Boots, and Siddhartha Srinivasa. Grasping with chopsticks: Combating covariate shift in model-free imitation learning for fine manipulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6185–6191. IEEE, 2021. 3

[12] Richard Li, Allan Jabri, Trevor Darrell, and Pulkit Agrawal. Towards practical multi-object manipulation using relational reinforcement learning. In *2020 ieee international conference on robotics and automation (ICRA)*, pages 4051–4058. IEEE, 2020. 1

[13] Nathanael Macias and John Wen. Vision guided robotic block stacking. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 779–784. IEEE, 2014. 4

[14] Drew McDermott, Malik Ghallab, Adele E. Howe, Craig A. Knoblock, Ashwin Ram, Manuela M. Veloso, Daniel S. Weld, and David E. Wilkins. Pddl-the planning domain definition language. 1998. 1, 3, 4

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015. 7

[16] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749, 2020. 4

[17] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2020. 4

[18] Tom Silver and Rohan Chitnis. Pddlgym: Gym environments from pddl problems. *arXiv preprint arXiv:2002.06432*, 2020. 4

[19] Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*, 2023. 3

[20] Yueh-Hua Wu, Jiashun Wang, and Xiaolong Wang. Learning generalizable dexterous manipulation from human grasp affordance. In *Conference on Robot Learning (CoRL)*, pages 618–629. PMLR, 2023. 3

[21] Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Neural task programming: Learning to generalize across hierarchical tasks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3795–3802, 2018. 7, 12

[22] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, et al. Reinforcement and imitation learning for diverse visuomotor skills. *arXiv preprint arXiv:1802.09564*, 2018. 4

A Author Contribution and Acknowledgement

All our teammates are greatly devoted to this project. Penglin established the environment and collaborated with Xingping to implement the SGN module. Meanwhile, Shanglin and Chenhao implemented the continuous planner. All the four team members actively participated in conducting experiments and contributed to the writing and revision of this paper.

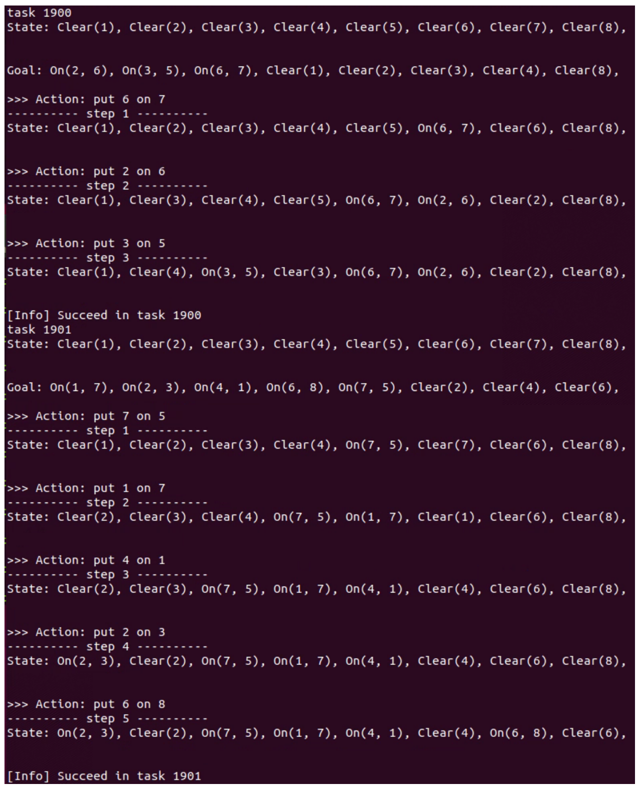
Since this project is required to replicate Huang *et al*. [9], it is essential to distinguish the parts reproduced from the original paper and those designed by ourselves. Specifically:

- The environment is our original design, to simplify the action space and state representation.
- The architecture SGN module and the procedures of the CP module are borrowed from the original paper, but the implementation details (not covered in the original paper) and the extra improvements are designed by ourselves.
- We obtain the 2,000 block stacking tasks from Xu *et al*. [21], and sample the demonstrations as datasets using our continuous planner as an expert.
- For experiments, the LLM-agent baseline and the ablation studies are our original design.

We extend our sincere appreciation to Prof. Zhu and the TAs, whose guidance and support have been instrumental in our progress and have significantly enhanced our learning experience. Some basic ideas in cognitive reasoning, including communications, tool-use, intentionality, and abstract reasoning, have provided great inspirations in our learning and research.

B The Details of the Environment

Our lightweight environment contains the basic elements of an online learning environment. In this section, we introduce more details about the environment and the corresponding block stacking tasks. A screenshot of command line is also attached (Fig. 3) for ease of understanding our environment.



```
task 1900
State: Clear(1), Clear(2), Clear(3), Clear(4), Clear(5), Clear(6), Clear(7), Clear(8),


Goal: On(2, 6), On(3, 5), On(6, 7), Clear(1), Clear(2), Clear(3), Clear(4), Clear(8),

>>> Action: put 6 on 7
---------- step 1 ----------
State: Clear(1), Clear(2), Clear(3), Clear(4), Clear(5), On(6, 7), Clear(6), Clear(8),


>>> Action: put 2 on 6
---------- step 2 ----------
State: Clear(1), Clear(3), Clear(4), Clear(5), On(6, 7), On(2, 6), Clear(2), Clear(8),


>>> Action: put 3 on 5
---------- step 3 ----------
State: Clear(1), Clear(4), On(3, 5), Clear(3), On(6, 7), On(2, 6), Clear(2), Clear(8),


[Info] Succeed in task 1900
task 1901
State: Clear(1), Clear(2), Clear(3), Clear(4), Clear(5), Clear(6), Clear(7), Clear(8),


Goal: On(1, 7), On(2, 3), On(4, 1), On(6, 8), On(7, 5), Clear(2), Clear(4), Clear(6),

>>> Action: put 7 on 5
---------- step 1 ----------
State: Clear(1), Clear(2), Clear(3), Clear(4), On(7, 5), Clear(7), Clear(6), Clear(8),


>>> Action: put 1 on 7
---------- step 2 ----------
State: Clear(2), Clear(3), Clear(4), On(7, 5), On(1, 7), Clear(1), Clear(6), Clear(8),


>>> Action: put 4 on 1
---------- step 3 ----------
State: Clear(2), Clear(3), On(7, 5), On(1, 7), On(4, 1), Clear(4), Clear(6), Clear(8),


>>> Action: put 2 on 3
---------- step 4 ----------
State: On(2, 3), Clear(2), On(7, 5), On(1, 7), On(4, 1), Clear(4), Clear(6), Clear(8),


>>> Action: put 6 on 8
---------- step 5 ----------
State: On(2, 3), Clear(2), On(7, 5), On(1, 7), On(4, 1), Clear(4), On(6, 8), Clear(6),


[Info] Succeed in task 1901
```

Figure 3: A screenshot of command line of our environment. This is a demonstration of finishing the 1,900 and 1,901 tasks in the testing set.

B.1 Coordinate System

For simplification, we use an 8×8 matrix to represent the coordinate system of the environment. In the initial state, each block (numbered i) is placed at $(i, 0)$. Suppose block i is at (x_i, y_i) at a certain time, then after placing block j on the top of block i, the coordinate of block j will be $(x_i, y_i + 1)$.

B.2 PDDL Description and Ground Atoms

We provide two kinds of ground atoms - `Clear(A)` and `On(A,B)`. `Clear(A)` means that there are no other blocks on the top of block A, while `On(A,B)` indicates that block A is on the top of block B. It is worth noting that we do not use transitivity of two atoms. For instance, if both `On(A,B)` and `On(B,C)` are true, we do not have `On(A,C)`. In other words, `On(`i`,`j`)` describes the case where block i is just on the top surface of block j.

B.3 Action Space

We use a string `"put A on B"` to denote an action, which is a combination of the source block and target place. Such action will result in putting block A on the top of block B. Any illogical statements are invalid, such as `"put A on A"` or `"put A on C"`, when `Clear(C)` is false.

C Implementation Details

C.1 Details of SGN

There are 10 MLPs in our SGN module, 8 of which are correlated with the 8 blocks, while the other two are corresponding with the two predicates.

Specifically, the 8 block-MLPs take in the current state (in the form of coordinate values) and output the embeddings of the each block respectively. Then every two block-embeddings combine together (by concatenation) to act as the input of the MLP related to predicate `On(`$\cdot, \cdot$`)`. There are 56 combinations in total, corresponding with the former 56 elements of the final continuous embedding vector. In the next step, each of the 8 block-embeddings will also act as the input of the MLP related to predicate `Clear(`$\cdot$`)`. The 8 outputted scalar become the latter 8 elements of the final continuous embedding vector. In the end, we concatenate the former 56 elements with the latter 8 element to construct the continuous embedding representation, which is 64-dimensional in total.

Some structure-parameters of the SGN module are listed in Tab. 2, and the hyper-parameters in Tab. 3.

Table 2: Some structure-parameters of the SGN module. "On-MLP" denotes the MLP related to the predicate `On(`$\cdot, \cdot$`)`, and "Clear-MLP" denotes the MLP related to the predicate `Clear(`$\cdot$`)`.

Structure-parameter	Value
input dimension of block-MLP	16
hidden dimension of block-MLP	128
output dimension of block-MLP	64
input dimension of On-MLP	128
hidden dimension of On-MLP	128
output dimension of On-MLP	1
input dimension of Clear-MLP	64
hidden dimension of Clear-MLP	128
output dimension of Clear-MLP	1

C.2 Details of LLM-Agent

In this section, we provide the prompts for the LLM.

```
You should complete a task of block stacking.
I will explain the PDDL language first:
```

Table 3: Some hyper-parameters of the SGN module.

Hyper-parameter	Value
loss function	binary cross entropy with logits
optimizer	Adam
learning rate	0.001

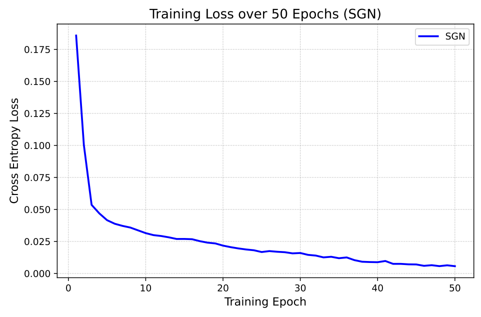
```
There are 8 blocks, numbered from 1 to 8.
We use 2 predicates to describe a state: On(i, j) and Clear (i).
On(i, j) means block i is on the top of block j (1 <= i, j <= 8).
Clear (i) means there are no other blocks on the top of block i (1 <= i <= 8).
Given a state described by several predicates, you should output the next action.
The action is in the form of 'put i on j', where 1 <= i, j <= 8.
The action means putting block i on the top of block j.
After execution, the next state will be notified to you. Please note that if you
    take an invalid action, the state will not change.
The history of actions and states will be shown to you:

<History Information>

The goal state is:

<Goal PDDL Description>

Please give your next action in exactly one sentence, in the format of "put <i> on <
    j>":
```

The history contains multiple pairs up to current step T: $(s_0, a_1, s_1, a_2, s_2, ..., a_T, s_T)$, and the PDDL description of goal s_N contains eight ground atoms. We use this prompt to ask GPT-4 to generate the next action a_{T+1}.

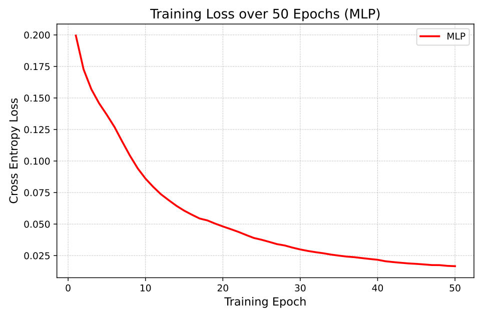
D Additional Results

D.1 Loss Curve of Training Phase

We provide the training loss curves in Fig. 4a (SGN) and Fig. 4b (MLP).



(a) Loss curve of SGN.

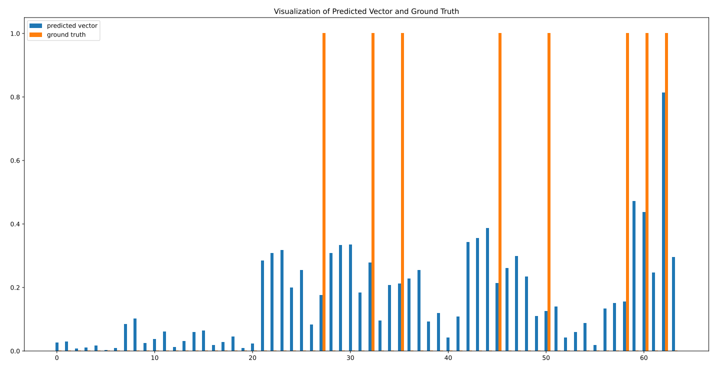


(b) Loss curve of MLP.

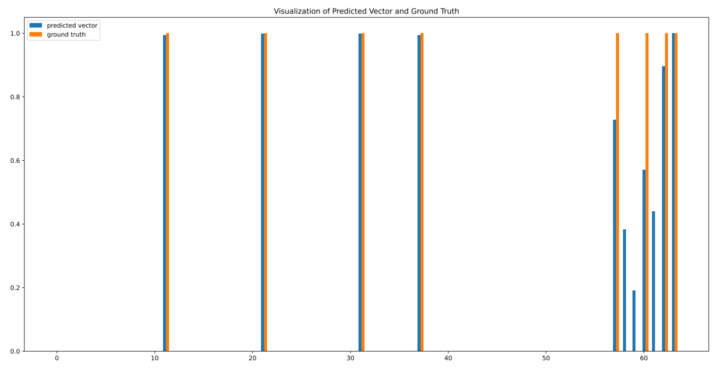
Figure 4: Loss curve over 50 training epochs.

D.2 Visualization of the Predicted Embedding Vectors

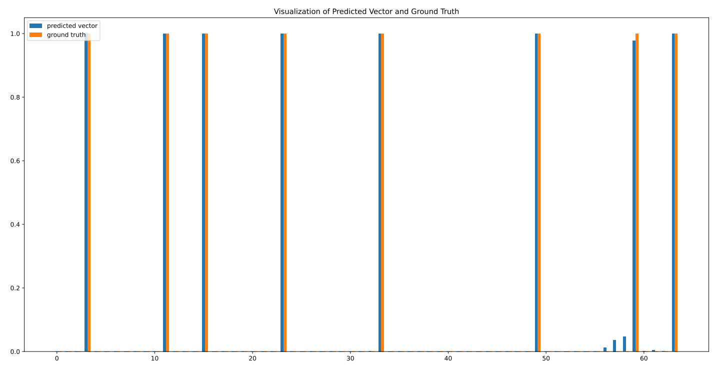
To intuitively compare the predicted embedding vector with the ground truth, we visualize the relative values of the predicted embedding vector and the ground truth, as shown in Fig. 5.



(a) Visualization of the predicted embedding vector and the ground truth using the epoch-1 model of SGN.



(b) Visualization of the predicted embedding vector and the ground truth using the epoch-10 model of SGN.



(c) Visualization of the predicted embedding vector and the ground truth using the epoch-50 model of SGN.

Figure 5: Visualization of the predicted embedding vector and the ground truth using models of SGN from different training epochs. This figure shows the great convergence of our SGN module.