# Blocks Assemble! Learning to Assemble with Large-Scale Structured Reinforcement Learning

Seyed **Kamyar** Seyed Ghasemipour [1]   **Daniel Freeman** [1]   **Byron David** [1]   **Shixiang Shane Gu** [1]
**Satoshi Kataoka** [1] [*]   **Igor Mordatch** [1] [*]

## Abstract

Assembly of multi-part physical structures is both a valuable end product for autonomous robotics, as well as a valuable diagnostic task for open-ended training of embodied intelligent agents. We introduce a naturalistic physics-based environment with a set of connectable magnet blocks inspired by children's toy kits. The objective is to assemble blocks into a succession of target blueprints. Despite the simplicity of this objective, the compositional nature of building diverse blueprints from a set of blocks leads to an explosion of complexity in structures that agents encounter. Furthermore, assembly stresses agents' multi-step planning, physical reasoning, and bimanual coordination. We find that the combination of large-scale reinforcement learning and graph-based policies – surprisingly without any additional complexity – is an effective recipe for training agents that not only generalize to complex unseen blueprints in a zero-shot manner, but even operate in a reset-free setting without being trained to do so. Through extensive experiments, we highlight the importance of large-scale training, structured representations, contributions of multi-task vs. single-task learning, as well as the effects of curriculums, and discuss qualitative behaviors of trained agents. Our accompanying project webpage can be found at: sites.google.com/view/learning-direct-assembly

## 1. Introduction

Robotic assembly of objects from a given set of parts is an incredibly intriguing avenue for research in artificial intelligence (AI). Not only is it a valuable capability we would like autonomous robots to possess, but it is a challenging problem statement with open-ended complexity, touching upon many fruitful avenues of AI research. Agents that can assemble structures can reshape their surroundings, which creates dynamic environments with more possibilities for open-ended learning (Baker et al., 2019; Wang et al., 2019; Co-Reyes et al., 2020). A key feature of assembly is that due to its compositional and modular nature, given a set of parts, one can create objects on a broad spectrum of complexity. Furthermore, in order to solve the assembly problem, agents must acquire a diverse set of skills and capabilities. They must learn to grasp and attach components in an order that is amenable to successful completion. They must develop physical reasoning capabilities to avoid collision, and they need to learn bi-manual coordination. In addition, once agents have acquired such skills, it is expected that they can rapidly learn to create new desired objects.

To study assembly, we create a simulated but naturalistic environment which consists of blocks of varying shapes that can be magnetically attached to one-another. Agents are then tasked with constructing a desired structure from one of almost 200 pre-designed blueprints (although generative models can be used to automate this process (Thompson et al., 2020)). In this environment, in lieu of grasping robotic arms, we enable agents to directly move desired blocks. Such direct manipulation and the use of magnetic connections (instead of more complex joining mechanisms) abstracts away some details of the assembly problem while retaining many of the challenges that make the problem hard and interesting. Compared robot manipulation tasks which emphasize rearrangement and stacking (Li et al., 2020; OpenAI et al., 2021), the compositional nature of assembly from a set of blocks leads to an agent continually encountering new structures of varying complexities, which necessitates developing a richer representation of what constitutes an "object" (Spelke, 1990). In addition, our magnetic assembly benchmark stresses multi-step planning, physical reasoning, and bimanual coordination.

Despite the complexity of this problem, we find that it is possible to train a single agent that can simultaneously assemble all given blueprint tasks, generalize to complex un-

---

[*]Equal supervision  [1]Google Research. Correspondence to: Seyed Kamyar Seyed Ghasemipour <kamyar@google.com>.
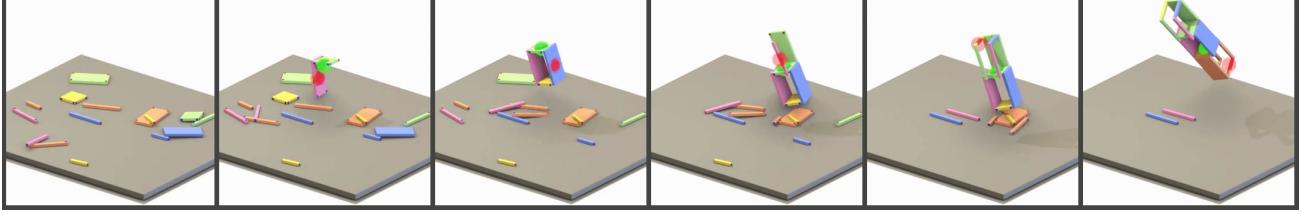
*Figure 1.* Visualization of trained agent building a blueprint that was not seen during training. The blueprint corresponds to a complex structure requiring 14 of the available 16 blocks.



*Figure 2.* Examples of target blueprints we consider. We train on variety of target structures, ranging from structures of 2 to 16 blocks.

seen blueprints in a zero-shot manner, and even operate in a reset-free manner despite being trained in an episodic fashion. Our solution relies on a combination of large-scale reinforcement learning, structured (graph-based) agent representations, and simultaneous multi-blueprint training. Simultaneously training on diverse blueprints of varying complexity scaffolds the agent's learning by enabling it to first make progress of simpler tasks (such as simply joining two blocks), while structured policy representations enable the agent to generalize and transfer its solutions towards solving more complex – and even unseen – blueprints. We empirically observe a progression of learning increasingly large blueprints - many of which were not solvable with single-blueprint training. Surprisingly, we find other components such as planning or hierarchical approaches to be unnecessary for this task. Through experiments, we highlight the contributions of various components such as structured policies, episodic initial state distribution, curriculum that emphasize training on harder blueprints, and discuss qualitative behaviors and maneuvers discovered by the trained agents.

Our contributions are as follows:

- We introduce an assembly domain that allows for a controlled study of generalization in reinforcement learning (RL).

- We demonstrate a single agent that can simultaneously solve all seen assembly tasks and generalize to unseen tasks.

- We demonstrate the importance of combining large-scale RL, structured policies, and multi-task training as a route to arrive at generally capable agents.
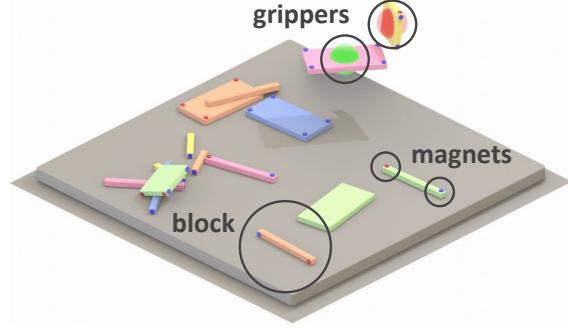


*Figure 3.* Our magnetic assembly domain. Two virtual grippers directly manipulate the available blocks to magnetically assemble a desired blueprint structure.

We hope this work further encourages study of assembly as an open-ended means to develop and evaluate embodied agent learning.

## 2. Magnetic Block Assembly Environment

Our goal is to design a minimal tractable assembly environment to study generalization in a naturalistic, multi-step, combinatorial, dynamic problem requiring bi-hand coordination. We construct a three-dimensional environment containing a fixed set of 16 cuboid blocks of 6 different types. Blocks contain positive and negative magnet points, rendered as red and blue respectively, positioned on the block surface. Positive and negative magnets "snap" together when sufficiently close, and disconnect when adequate pulling force is applied. Magnets enable creation of arbitrarily complex composed structures from the given building blocks. Additionally, magnets can be implemented in

the real-world, unlike more abstract locking constraints (e.g. instantaneous weld constraints), yet are tolerant enough to join objects without tackling the problem of high-precision insertion that would be required for other connection mechanisms such as pegs or screws. To simplify the problem, in lieu of robotic arms we opt for the use of virtual grippers which can directly manipulate desired blocks. More specifically, each gripper can decide which block to move, and set its positional and rotational velocities[1]. The use of direct manipulation abstracts away the challenges of grasping and manipulation with a robotic arm, and enables us to focus on research questions concerning higher-level assembly behaviors such as planning and generalization to unseen structures. While the number of grippers is parameterizable, unless otherwise specified, throughout this work we will use 2 virtual grippers.

To specify the assembly task, we designed 165 blueprints (split into 141 train, 24 test) describing interesting structures to be built, although the blueprints can potentially be procedurally generated (Thompson et al., 2020). The complexity of the created blueprints range from requiring only a single magnetic connection, up to challenging structures that make use of all 16 available blocks. The problem statement in our magnetic assembly environment is simple to describe: In each episode, the agent must assemble the blocks to create the desired blueprint. Each episode begins with either all blocks randomly scattered around the environment, or from a randomly sampled pre-constructed blueprint – with unused blocks dispersed on the ground. Episodes are 100 environment steps long, translating to a length of 10 seconds in the real world. In each step agents receive rewards based on how close blocks are to their intended configurations, as well as correct and incorrect magnetic connections. Episodes terminate when exactly correct magnetic connections are made and blueprint blocks are in the correct relative position and orientation, or when 100 steps has passed. Our simulated assembly task is implemented in the open-source Mujoco (Todorov et al., 2012) physics engine. A detailed description of observation space, action space, rewards, and success criterion used can be found in Appendix B.

## 3. Methodology

While solving the magnetic assembly task can be approached through a variety of solutions such as hierarchical reinforcement learning and geometric planning algorithms (e.g. RRT (LaValle et al., 2001)), in past work it has been demonstrated that many tasks which intuitively require complex planning strategies can be solved through large-scale application of reinforcement learning algorithms using the

---

[1]We restrict velocity magnitudes to a range that would be realistically achievable if the blocks were being manipulated by real robot arms.
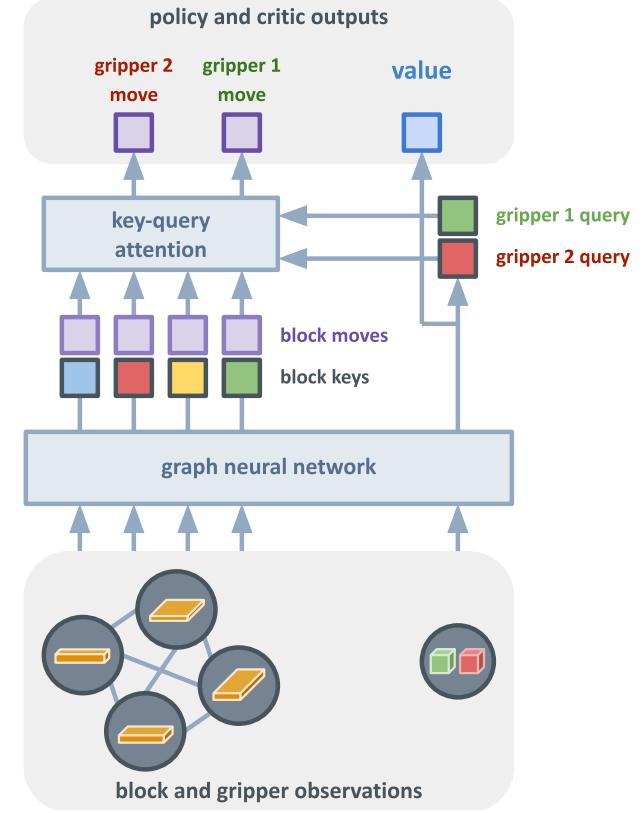


*Figure 4.* Diagram depicting our structured agent. Inputs to the agent are graph-structured block observations as well as gripper observations. A graph neural network processes the observations and produces: (1) per block moves and per block attention keys, (2) per gripper attention queries, (3) a global latent representation. Using dot-product attention between the keys and queries, the grippers decide which block to hold, and output its proposed move. The global latent representation is used to predict a baseline value for the PPO (Schulman et al., 2017) algorithm.

right training setups, and appropriate neural network architectures and inductive biases (Silver et al., 2017; Berner et al., 2019; Vinyals et al., 2019; Baker et al., 2019). Motivated by such results, in this work we explore the ingredients necessary to train effective agents for magnetic assembly through RL. In the subsequent sections we describe the main ingredients of training successful agents and study the contribution of each component.

## 4. Agents

In this section we describe our structured agents, including observations and action spaces, and graph-based network architecture. The Python code describing our agent architecture can be found in Appendix C.

### 4.1. Structured Observations

The observations provided to the agent can be divided into two broad categories, those concerning the blocks, and those concerning the grippers. When designing observations to provide to agents, we have taken the effort to ensure observations are invariant to the global position and orientation. This is a valuable inductive bias that provides agents with the flexibility build desired blueprint anywhere and in any rotation.

**Block Observations** In the assembly task, observations pertaining to the blocks can be naturally organized into a directed graph, with each node containing information about a particular block, and each directed edge representing relative information about the two blocks. The information contained in each node is very minimal: the z height of the block from the ground, and whether it was being held by each gripper in the previous timestep. The majority of observations are placed on the directed edges. An edge connecting two blocks contains the information regarding: relative position and orientation of their magnets that need to be connected, change in relative position and orientation of the blocks needed to match the blueprint, relative position of center of mass of the blocks, whether the blocks are magnetically attached, and whether the blocks should be magnetically attached according to the blueprint. All these observations can be automatically extracted from the simulator state and the target blueprint configuration, and can realistically be computed in a real-world setting as well by simply obtaining each blocks position and orientation. Detailed information regarding exact observations can be found in Appendix B.

**Gripper Observations** For each gripper we include its orientation, positional and rotational velocities, and which block the gripper was holding in the previous timestep.

### 4.2. Graph Neural Network Encoder

Given that our magnetic assembly task can be naturally set up using graph-based observations, prior to extracting actions and critic values, we first encode inputs using a graph neural network architecture (Battaglia et al., 2018), specifically graph attention networks (Veličković et al., 2017). The two inputs to our encoder are (1) a directed graph containing all block observations (2) a "global node" containing gripper observations. After linearly embedding all input features, they are passed through $N = 3$ graph attention layers whose design is inspired by Transformers (Vaswani et al., 2017) and Graph Attention Networks (Veličković et al., 2017). Concisely, in each layer, each node aggregates information by attending to incoming edge and node features, and subsequently the global node features are updated by aggregating information from the graph nodes. An intuitive diagram

describing the architecture used can bee seen in Figure 4.

### 4.3. Policy

Through experimentation, we have discovered that in addition to a graph neural network encoder, a key design choice is how to extract policy actions from the encoded inputs. The outputs of the graph neural network encoder are hidden features per node corresponding to the blocks, and hidden features corresponding to the global node. Using linear layers, from each block node we obtain 2 vectors: (1) a vector representing how the block would like to be moved if a gripper chooses to move it, and (2) a vector representing a key vector for the block. From the global hidden features, using linear layers we obtain one query vector per gripper. To obtain logits representing which gripper decides to move which block, we use dot-product attention between the block keys and the gripper query vectors, akin to the popular attention mechanism used in Transformers (Vaswani et al., 2017). In addition to the use of graph neural networks, using this form of decoding from the graph encoder has been a key enabler in training effective policies.

### 4.4. Critic

To train our RL agents we also require critic value estimates, which we obtain by passing global features obtained from the graph encoder to a 3 layer MLP, with 512 dimensional hidden layers, and relu activation function.

## 5. Training and Evaluation

**Large-Scale PPO** We train our agents using Proximal Policy Optimization (PPO) (Schulman et al., 2017) and Generalized Advantage Estimation (GAE) (Schulman et al., 2015), and follow the practical PPO training advice of (Andrychowicz et al., 2020a). As will be shown below, one of the most key ingredient in enabling the training of our magnetic assembly agents is the scale of training. Unless otherwise specified, our agents are trained for 1 Billion environment timesteps, using 1 Nvidia V100 GPU for training, and 3000 preemptible CPUs for generating rollouts in the environment. 1 Billion steps in our setup amounts to about 48 hours of training. The key libraries used for training are Jax (Bradbury et al., 2018), Jraph (Godwin* et al., 2020), Haiku (Hennigan et al., 2020), and Acme (Hoffman et al., 2020).

**Multi-Task Training** The blueprints that we have designed range from very simple 2 block structures, up to complex blueprints containing all blocks. To train assembly agents, we have split blueprints into training and testing structures, and unless otherwise specified, agents are trained on the full training set of blueprints; in each episode, we sample a training blueprint and task the agent with creating

that structure.

**Initial State**   Episodes start from either (1) all the blocks randomly dispersed on the ground, or (2) a randomly chosen preconstructed blueprint structure with unused blocks randomly arranged on the ground. Resetting from blueprints increases the diversity of initial states, forces the agent to learn how to disassemble structures, and as we found enables a reset-free mode of operation where the agent can continually construct, deconstruct, and reconstruct different blueprints. Unless otherwise specified, we reset from training blueprints with probability 0.2.

**Curriculum**   We have observed that throughout training, some blueprints can be quickly learned while others can be much more challenging. We believe an interesting feature of the assembly problem is that even if a blueprint is currently unsolvable, due to the modular aspect of building complex structures, agents can learn more effectively if we emphasize focus on more challenging blueprints rather than allocating resources to rolling out policies on blueprints that they are already capable of solving. For this reason, in each episode we sample goal blueprints based on a curriculum whose detailed description can be found in Appendix D.

**Performance Evaluation**   During training, we evaluate trained policies continuously approximately every 10 minutes by freezing the policy and computing average success rate over 40 episodes. This continuous evaluation is executed on both training and test environments. Also, in each evaluation cycle, we generate a video to visualize the agents' behavior. Such visualizations have been a valuable asset in iterating over the design of our agents, observations, reward functions, and training setups.

## 6. Experiments

### 6.1. Importance of Large-Scale Training

We begin by verifying that our training procedure leads to capable assembly agents. To this end, we train our structured agents (Section 4), using the training procedure described in Section 5, for 2.5 billion environment steps to observe training patterns that may arise over a long period of training. Figure 5 presents the success rates of our agent (averaged across two runs) throughout training, on blueprints the agent was trained on as well as held-out structures (per blueprint success rates presented in Figure 12 in the appendix).

The first key observation is the compute scale necessary for effectively training our structured agents using PPO (Schulman et al., 2017). The simplest 2 block structures can take up to 100 million steps to be reliably solved, while it can take up to 500 million environment steps until the first time some of the most complex blueprints are solved. The

second observation is that after a long period of training, not only can agents reliably solve all training blueprints, but they can also generalize well to complex held-out blueprints.

### 6.2. Multi-Task vs. Single-Task

As noted in Section 5, agents are simultaneously trained to construct all blueprints in the training split. To understand the contribution of this "Multi-Task" training, we train three agents in a "Single-Task" setting: one for learning to construct a particular 6 block blueprint, one for constructing a particular 12 block blueprint, and one for constructing a particular 16 block blueprint. The success rates for these three agents can be found in Figures 15, 16, and 17 respectively. Our key observations are the following: (1) While the 6 and 12 block blueprints are eventually learned, the 16 block blueprint is not learned, (2) In the single-task setting, the 12 block blueprint requires approximately 500 million environment steps to be learned, while in the multi-task setting (Figure 12) it is learned within 300 million steps, (3) the single-task agents can transfer to some blueprints of equal or lower complexity than they were trained on, but mostly fail to transfer to any blueprints they were not trained to solve. This is in sharp contrast to the multi-task agents which can even transfer to complex held-out blueprints. These results highlight the necessity of multi-task training, not only for generalization to unseen blueprints, but for quickly and reliably solving complex tasks, despite the fact that agent architectures are well-matched to the problem domain.

### 6.3. Structured Agent Architecture

As discussed in section 4, given that state information for the assembly task can be naturally organized into a graph representation, the use of graph neural networks imbues agents with an inductive bias that is well-matched to the domain. Indeed, prior work (Bapst et al., 2019; Li et al., 2020) have observed that the use of agents with relational structures is a key ingredient in solving object-oriented tasks. In this section we aim to understand the contribution of various components of our agents' architecture.

**Removing Attention in Graph Layers**   Figure 11 demonstrates the effect of removing the attention mechanism in the graph neural network layers, meaning that while our agents continue have a graph inductive bias, the hidden representations for each block are updated by treating other blocks equally, rather than deciding which blocks to attend to. The results in Table 2 and Figure 11 clearly demonstrate the necessity of the attention mechanism.

**Removing Relational Inductive Biases**   We also attempt to train agents without relational inductive biases. Instead, we flatten the environment observations and use a residual network (He et al., 2016) encoder, with a similar action
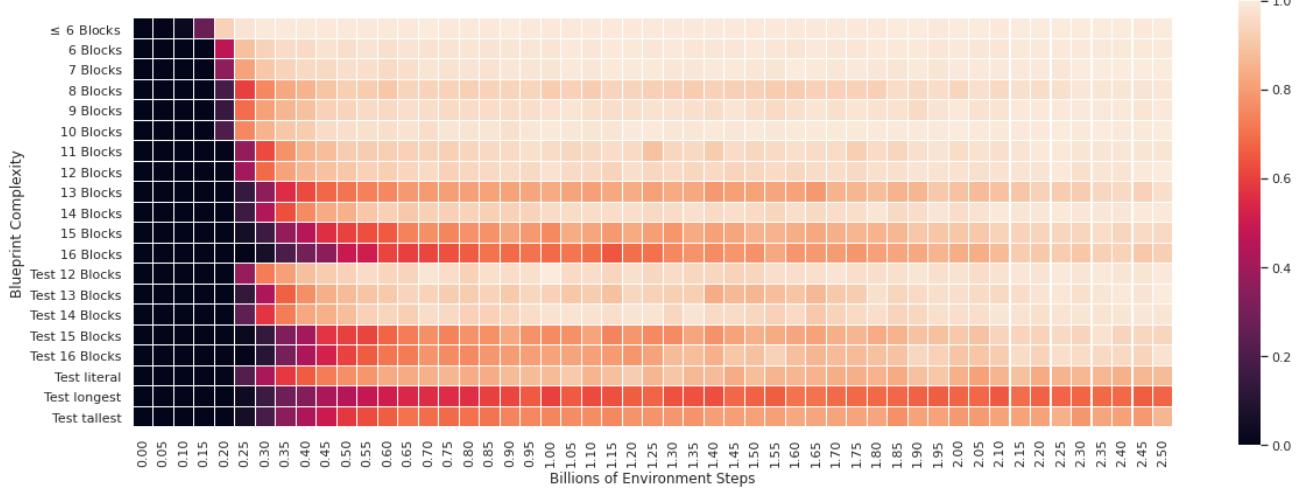
*Figure 5.* Plot presenting success rates for different groups of blueprints throughout training. Each square represents the success rate on 40 episodes, evaluated at that point in training, averaged across two training runs. Our results demonstrate (1) the key role of scale in training successful agents, and that (2) after a long period of training, agents generalize well to complex held-out blueprints. Per blueprint results presented in Figure 12.

| Blueprint Size | Single-Blueprint Training | | Multi-Blueprint Training (ours) | |
|---|---|---|---|---|
| | **Success Rate** | **Steps Until Success** | **Success Rate** | **Steps Until Success** |
| 6 Blocks | 99.6% | 100M | 100% | 180M |
| 12 Blocks | 99.9% | 480M | 98.8% | 220M |
| 16 Blocks | 0% | – | 90.9% | 240M |

*Table 1.* Comparison of Single-Blueprint vs. Multi-Blueprint training for blueprints of various complexities. Success rates are calculated after 1 billion steps of training for the respective agents, and "Steps Until Success" denotes approximately the first timestep at which the respective agent was able to successfully create the blueprint. While for very small structures single-blueprint training can be effective, for more complex blueprints, single-blueprint agents take significantly longer or are entirely unable to learn the task within 1 billion timesteps.

and value decoder as described in Section 4. Details of the residual network architecture are described in Appendix E. We train three variants of the residual network agents: (1) trained on the full training set of blueprints, (2) trained on a subset of the training blueprints requiring $\leq 6$ blocks, and (3) trained on a single training blueprint requiring 6 blocks. Figures 22, 23, and 24 demonstrate that in all three scenarios, removing the relational inductive bias of graph neural networks is catastrophic. After 1 billion steps, all agents have a 0% success rate on all blueprints, and never accomplish higher than 2.5% success rate on any train or held-out blueprint.

### 6.4. Bimanual Manipulation

Compared to most robotics tasks and benchmarks (Levine et al., 2018; Kalashnikov et al., 2018; Andrychowicz et al., 2020b; Chen et al., 2021; Huang et al., 2021; Yu et al., 2020; Li et al., 2020; Batra et al., 2020; OpenAI et al., 2021), part-based assembly stresses the bimanual coordination of agents. To verify that our magnetic assembly domain

stresses this skill, we compare success rates between bimanual and single-gripper agents in Table 2 and Figure 10. We find that while our single-gripper agents finds unique strategies to complete some of the structures, its overall success rate is lower than that of a dual-gripper agent, particularly on the more complex blueprints. This indicates the necessity of using two grippers in our domain.

### 6.5. Reset-Free Evaluation

As described in Section 5, with probability 0.2, the initial state of an episode is set to be a randomly selected pre-constructed blueprint, with remaining blocks dispersed on the ground. This choice has two advantages: (1) it provides an opportunity for agents to learn how to disassemble incorrect constructions, and (2) it enables the evaluation of our agents in a reset-free manner, where we continually task agents with constructing new blueprints without resetting the environment to an initial state.

To evaluate the contribution of this choice, we compare

| Agent Type | Success Rate | | | | |
|---|---|---|---|---|---|
| | **Train 2 - 6 Blocks** | **Train 7 - 11 Blocks** | **Train 12 - 16 Blocks** | **Test 12 - 16 Blocks** | **Test Special** |
| **Default** | **99.9%** | **97.9%** | **87.3%** | **89.6%** | **77.5%** |
| **No Graph Attention** | 95.3% | 43.1% | 4.1% | 3.6% | 4.8% |
| **Non-Graph Network** | 0% | 0% | 0% | 0% | 0% |
| **Single-Gripper** | 94.6% | 77.9% | 51.7% | 55.4% | 42.8% |

*Table 2.* Comparing the success rates of agent ablations, discussed in Sections 6.3 and 6.4, on the various subsets of blueprints. Results are reported after 1 billion steps of training the respective agents.

the success rate of two agents, one with and one without blueprint resets, in a reset-free setting. Specifically, within one reset-free episode we ask agents to build 10 consecutive blueprints without resetting to an initial state: once an agent successfully constructs a blueprint, or the maximum of 100 steps has elapsed, we change the target blueprint. As an additional challenge, we sample blueprints from the training set structures requiring a minimum of 12 blocks. We report the success rate aggregated across 50 reset-free episodes (i.e. $50 \times 10$ total episodes).

When resetting from blueprints is disabled, our agent achieves a sucess rate of $69.4\% \pm 17.0\%$. In constrast, with blueprint resets, the success rate increases to $93.1\% \pm 7.5\%$. This is an exciting finding as it demonstrates a scenario where episodic training enables agents to be deployed in the practically-relevant reset-free scenario.

### 6.6. Curriculum

As described in Section 5, throughout training we make use of a curriculum that increases the likelihood of sampling more challenging blueprints. To analyze its contribution, we compare two runs of agents with curriculum, to an agent trained without curriculum. Our results in Figures 18 and 19 indicate that our curriculums do not have a clear-cut benefit, but may be leading to improvements in generalization to blueprints in the held-out test set.

### 6.7. Curbing Rapid Switching Between Blocks

Due to the use of direct manipulation, agents can rapidly switch which block they are holding, which can result in sometimes unrealistic maneuvers not achievable by physical robot grippers. Thus, in the event we wanted to transfer the success from our direct manipulation environment to more realistic settings using robotic arms, it is important to understand how one can mitigate this unrealistic behaviors. To this end, after training an agent using the default training procedure described in Section 5, we modify the environment as follows: Whenever a gripper chooses to change the object it is holding, we disable that gripper for 2 steps. With this change in place, we continue to train our agent.

Figure 20 demonstrates that while initially the agent's success rate drops very significantly, within less than 100 mil-

lion environment steps the agent recovers its strong performance. This is a small amount of steps compared to the 2.5 billion environment steps used to train the initial agent.

Given this result, one might ask whether we could have started training our agents using this environment modification from the beginning. Figure 21 compares this approach to our default training setup. As can be seen, training agents from scratch using gripper transition delays is a significantly more challenging problem, and even after 1.5 billion environment steps, the agent is still unable to make significant progress on many of the blueprints in the training set. These results demonstrate that an efficient approach towards training practical agents is to first train agents in the simplest settings, and continue to finetune those agents in more realistic scenarios. Videos demonstrating behaviors of agents discussed in this section can be found in the accompanying project webpage.

### 6.8. Analyzing Learned Solutions

In this section our goal is to obtain a qualitative understanding of the strategies learned by our trained agents.

**Learned Attention Patterns** As shown in Section 6.3, the attention mechanism is a key ingredient in the graph neural network architecture we used. To understand what the attention heads in the different layers have learned to focus on, we visualize agents' attention patterns throughout different episodes. We observe the following interpretable patterns: (1) Some attention heads focus on the blocks that should be connected according to the blueprint, (2) Some keep account of which blocks are currently connected, and (3) Some focus on which blocks are currently being held by the two grippers. Other attention heads, particularly in the later layers, are more challenging to interpret, but appear to contain a combination of the previously described attention motifs. Videos visualizing learned attention patterns can be found in the accompanying website.

**Qualitative Behaviours** Rolling out trained agents, we observe a number of interesting learned behaviors. Examples of such behaviors include: (1) Despite environment observations not providing fine-grained detail about free-space, agents appear to have learned robust collision avoid-

ance skills. (2) When building complex structures, agents appear to first build separate smaller substructures, and subsequently attach the substructures to construct the full blueprint.

## 7. Related Work

**Assembly and Construction** Bapst et al. (2019) previously studied two-dimensional construction environments, training an agent to assemble a structure for an open-ended goal, such as a connecting or a covering structure. Certain details of assembly are abstracted away, as the agent has the ability to directly summon a block of choice anywhere in the scene and weld blocks via an explicit action. By contrast, our environment contains a fixed set of blocks that must be moved – or reassembled from a previous structure – in three-dimensional space, and where block connections are made via magnet forces. Such a design makes the environment more easily implementable as a real-world robot setup. Lee et al. (2019; 2021) also introduce a three-dimensional assembly environment for furniture design from a blueprint. By contrast, we use generic blocks, which leads to combinatorial complexity in the space of structures, and enables a more controlled study of generalization. A key point of differentiation from the above works is that we use assembly as a domain for the study of generalization, and train a single agent to solve all – seen and unseen – assembly tasks simultaneously. Additionally, we introduce the use of multiple grippers in assembly which allows us to evaluate the bimanual coordination of trained agents. Chung et al. (2021) present a task where given side view images of a desired structure, Lego blocks must be stacked to create a structure with a similar silhouette. At each step, the state of the Lego structure is encoded using graph neural networks, and through deep RL, a policy learns where the next Lego block must be placed. In contrast, in our work we focus on the dynamic task of assembly, where discrete decisions and continuous control are solved simultaneously using large-scale deep RL. Funk et al. (2022) introduce a 3D task where blocks must be re-arranged to create stable structures that occupy randomly sampled target regions. They present a long-horizon manipulation algorithm combining deep reinforcement learning and Monte-Carlo tree search, that at each step decides which block should be moved to which location. Similar to our work, multi-head attention graph neural networks are used, which enable generalization to new settings with larger number of blocks. By comparison, we train a single policy to jointly consider the low-level manipulation of the blocks and block selection. Suárez-Ruiz et al. (2018) studied assembly of a single chair with real-world bimanual robots using offline planning methods. Kim & Seo (2019); Cabi et al. (2019) studied real-world insertion problem, which is an operation in the broader assembly process. Hartmann et al. (2021) present a planning system to solve long-horizon multi-robot construction problems, consisting of stacking parts to create architectural structures. In contrast, in this work we focused on the assembly problem and were motivated to understand whether deep reinforcement learning can be used so simultaneously solve discrete planning and continuous control.

**Generalization in Robotic Manipulation** Much recent work in robot manipulation focused on the tasks of object grasping (Pinto & Gupta, 2016; Mahler et al., 2017; Levine et al., 2018; Kalashnikov et al., 2018), in-hand object manipulation (Andrychowicz et al., 2020b; Chen et al., 2021; Huang et al., 2021), or execution of a motor skill (Yu et al., 2020), where variation comes from diversity of object shapes and arrangements involved. By contrast, while assembly uses a fixed set of blocks, the compound structures that must be manipulated have a combinatorial diversity of shape that dynamicaly changes during the episode. Li et al. (2020); Batra et al. (2020); OpenAI et al. (2021) propose scene re-arrangement as a universal task for embodied AI. While re-arrangement is general, we caution that many instances of rearrangement can be solved as a sequence of largely independent sub-tasks that do not influence each other, and can be performed in any order. By contrast, assembly steps are coupled and must be performed in a specific order that must be discovered by the agent. Ahmed et al. (2020); Funk et al. (2022) also present robotics benchmarks based on block rearrangement, with stronger emphasis on long-horizon planning, learning from structured representations, and generalization to unseen scenarios. Gupta et al. (2019) introduce a kitchen environment with an implicit dependency structure among sub-tasks (i.e. opening a container before putting something in it), but such pre-conditions are harder to scale.

**Structured and Object-Centric Policies** Besides our work, there have been numerous efforts to parameterize policies through structured models such as graph neural networks or Transformers, leveraging intrinsic objectness and invariances of the physical world (Spelke, 1990). They have been used for controlling agents of different morphologies for locomotion and manipulation tasks (Wang et al., 2018; Sanchez-Gonzalez et al., 2018; Chen et al., 2018; Pathak et al., 2019; Huang et al., 2020; Kurin et al., 2020), as well as enabling agents to learn a compositionally-challenging task like stacking (Li et al., 2020). Other relevant works include parameterizing manipulation actions, especially of 2D tasks, as object-based spatial actions, which have enabled breakthroughs in vision-based manipulation (Zeng et al., 2020; Noguchi et al., 2021; Shridhar et al., 2022). While inspired by similar motivations, our work tackles a uniquely challenging task, 3D bi-manual assembly, compared to single-arm or 2D tasks in prior work (Li et al., 2020; Zeng et al., 2020).

# 8. Limitations, Discussion, & Future Directions

We introduced a new blueprint assembly environment for studying bimanual assembly of multi-part physical structures, and demonstrated training of a single agent that can simultaneously solve all seen and unseen assembly tasks via a combination of large-scale RL, structured policies, and multi-task training. While our work showed that a solution to our problem exists, it is by no means efficient - requiring billions of training episodes. It is likely that by incorporating planning or hierarchical methods, the training time can be significantly shortened. Additionally, upon maturity of accelerated simulation engines (Liang et al., 2018; Freeman et al., 2021; Makoviychuk et al., 2021), our agents may be trained at a similar compute scale using much more modest hardware infrastructures. Beyond more efficient training, in this work we chose to abstract away complexities of manipulation and perception. A more detailed treatment of these elements, such as constraints that prevent overly aggressive behaviors, can bring this work closer to robotics applications. And lastly, while we focused on blueprint assembly in this work, more open-ended assembly goals as well as multi-agent interaction present further research opportunities for developing agents of increasing complexity.

# 9. Author Contributions

Seyed Kamyar Seyed Ghasemipour: first-author; ran all experiments; wrote manuscript draft; Developed the agent architectures (e.g. Graph Neural Network architectures, policy action representations, etc.); contributed to the environment creation (improved and added observations, rewards, success conditions, moved observations from node-based towards edge-based structure, bug fixes).

Byron David, Daniel Freeman, Shixiang Shane Gu: helped experiment implementations; created blueprints used for magnetic parts assembly; engaged in research discussions.

Satoshi Kataoka: co-supervising author; created majority of robust infrastructure (evaluation infrastructure, and backup mechanisms); contributed to the environment creation (improved robustness, usability, and reliability); ceviewed all code modifications.

Igor Mordatch: co-supervising author; initiated the research direction; created the environment as well as the benchmark task definition; created first proof of concept results motivating more focused investigation; guided project direction; rendered figures and videos.

# References

Ahmed, O., Träuble, F., Goyal, A., Neitz, A., Bengio, Y., Schölkopf, B., Wüthrich, M., and Bauer, S. Causalworld: A robotic manipulation benchmark for causal structure and transfer learning. *arXiv preprint arXiv:2010.04296*, 2020.

Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., et al. What matters for on-policy deep actor-critic methods? a large-scale study. In *International Conference on Learning Representations*, 2020a.

Andrychowicz, O. M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020b.

Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

Bapst, V., Sanchez-Gonzalez, A., Doersch, C., Stachenfeld, K., Kohli, P., Battaglia, P., and Hamrick, J. Structured agents for physical construction. In *International Conference on Machine Learning*, pp. 464–474. PMLR, 2019.

Batra, D., Chang, A. X., Chernova, S., Davison, A. J., Deng, J., Koltun, V., Levine, S., Malik, J., Mordatch, I., Mottaghi, R., et al. Rearrangement: A challenge for embodied ai. *arXiv preprint arXiv:2011.01975*, 2020.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Cabi, S., Colmenarejo, S. G., Novikov, A., Konyushkova, K., Reed, S., Jeong, R., Zolna, K., Aytar, Y., Budden, D., Vecerik, M., et al. Scaling data-driven robotics with reward sketching and batch reinforcement learning. *arXiv preprint arXiv:1909.12200*, 2019.

Chen, T., Murali, A., and Gupta, A. Hardware conditioned policies for multi-robot transfer learning. *arXiv preprint arXiv:1811.09864*, 2018.

Chen, T., Xu, J., and Agrawal, P. A system for general in-hand object re-orientation. *Conference on Robot Learning*, 2021.

Chung, H., Kim, J., Knyazev, B., Lee, J., Taylor, G. W., Park, J., and Cho, M. Brick-by-brick: Combinatorial construction with deep reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Co-Reyes, J. D., Sanjeev, S., Berseth, G., Gupta, A., and Levine, S. Ecological reinforcement learning. *arXiv preprint arXiv:2006.12478*, 2020.

Freeman, C. D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., and Bachem, O. Brax - a differentiable physics engine for large scale rigid body simulation, 2021. URL http://github.com/google/brax.

Funk, N., Chalvatzaki, G., Belousov, B., and Peters, J. Learn2assemble with structured representations and search for robotic architectural construction. In *Conference on Robot Learning*, pp. 1401–1411. PMLR, 2022.

Godwin*, J., Keck*, T., Battaglia, P., Bapst, V., Kipf, T., Li, Y., Stachenfeld, K., Veličković, P., and Sanchez-Gonzalez, A. Jraph: A library for graph neural networks in jax., 2020. URL http://github.com/deepmind/jraph.

Gupta, A., Kumar, V., Lynch, C., Levine, S., and Hausman, K. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019.

Hartmann, V. N., Orthey, A., Driess, D., Oguz, O. S., and Toussaint, M. Long-horizon multi-robot rearrangement planning for construction assembly. *arXiv preprint arXiv:2106.02489*, 2021.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hennigan, T., Cai, T., Norman, T., and Babuschkin, I. Haiku: Sonnet for JAX, 2020. URL http://github.com/deepmind/dm-haiku.

Hoffman, M., Shahriari, B., Aslanides, J., Barth-Maron, G., Behbahani, F., Norman, T., Abdolmaleki, A., Cassirer, A., Yang, F., Baumli, K., Henderson, S., Novikov, A., Colmenarejo, S. G., Cabi, S., Gulcehre, C., Paine, T. L., Cowie, A., Wang, Z., Piot, B., and de Freitas, N. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020. URL https://arxiv.org/abs/2006.00979.

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

Huang, W., Mordatch, I., Abbeel, P., and Pathak, D. Generalization in dexterous manipulation via geometry-aware multi-task learning. *arXiv preprint arXiv:2111.03062*, 2021.

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

Kim, C. H. and Seo, J. Shallow-depth insertion: Peg in shallow hole through robotic in-hand manipulation. *IEEE Robotics and Automation Letters*, 4(2):383–390, 2019.

Kurin, V., Igl, M., Rocktäschel, T., Boehmer, W., and Whiteson, S. My body is a cage: the role of morphology in graph-based incompatible control. *arXiv preprint arXiv:2010.01856*, 2020.

LaValle, S. M., Kuffner, J. J., Donald, B., et al. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and computational robotics: new directions*, 5: 293–308, 2001.

Lee, Y., Hu, E. S., Yang, Z., Yin, A., and Lim, J. J. Ikea furniture assembly environment for long-horizon complex manipulation tasks. *arXiv preprint arXiv:1911.07246*, 2019.

Lee, Y., Lim, J. J., Anandkumar, A., and Zhu, Y. Adversarial skill chaining for long-horizon robot manipulation via terminal state regularization. *arXiv preprint arXiv:2111.07999*, 2021.

Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., and Quillen, D. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

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

Liang, J., Makoviychuk, V., Handa, A., Chentanez, N., Macklin, M., and Fox, D. Gpu-accelerated robotic simulation for distributed reinforcement learning. In *Conference on Robot Learning*, pp. 270–282. PMLR, 2018.

Mahler, J., Liang, J., Niyaz, S., Laskey, M., Doan, R., Liu, X., Ojea, J. A., and Goldberg, K. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*, 2017.

Makoviychuk, V., Wawrzyniak, L., Guo, Y., Lu, M., Storey, K., Macklin, M., Hoeller, D., Rudin, N., Allshire, A., Handa, A., et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

Noguchi, Y., Matsushima, T., Matsuo, Y., and Gu, S. S. Tool as embodiment for recursive manipulation. *arXiv preprint arXiv:2112.00359*, 2021.

OpenAI, O., Plappert, M., Sampedro, R., Xu, T., Akkaya, I., Kosaraju, V., Welinder, P., D'Sa, R., Petron, A., Pinto, H. P. d. O., et al. Asymmetric self-play for automatic goal discovery in robotic manipulation. *arXiv preprint arXiv:2101.04882*, 2021.

Pathak, D., Lu, C., Darrell, T., Isola, P., and Efros, A. A. Learning to control self-assembling morphologies: a study of generalization via modularity. *arXiv preprint arXiv:1902.05546*, 2019.

Pinto, L. and Gupta, A. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pp. 3406–3413. IEEE, 2016.

Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R., and Battaglia, P. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pp. 4470–4479. PMLR, 2018.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

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

Shridhar, M., Manuelli, L., and Fox, D. Cliport: What and where pathways for robotic manipulation. In *Conference on Robot Learning*, pp. 894–906. PMLR, 2022.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Spelke, E. S. Principles of object perception. *Cognitive science*, 14(1):29–56, 1990.

Suárez-Ruiz, F., Zhou, X., and Pham, Q.-C. Can robots assemble an ikea chair? *Science Robotics*, 3(17):eaat6385, 2018.

Thompson, R., Ghalebi, E., DeVries, T., and Taylor, G. W. Building lego using deep generative models of graphs. *arXiv preprint arXiv:2012.11543*, 2020.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, 2019.

Wang, R., Lehman, J., Clune, J., and Stanley, K. O. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019.

Wang, T., Liao, R., Ba, J., and Fidler, S. Nervenet: Learning structured policy with graph neural networks. In *International Conference on Learning Representations*, 2018.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pp. 1094–1100. PMLR, 2020.

Zeng, A., Florence, P., Tompson, J., Welker, S., Chien, J., Attarian, M., Armstrong, T., Krasin, I., Duong, D., Sindhwani, V., et al. Transporter networks: Rearranging the visual world for robotic manipulation. *arXiv preprint arXiv:2010.14406*, 2020.

# A. Blueprints

In this section, we describe details of all blueprints. Figure 6 and 7 are used for training while figure 8 and 9 are used for testing. Table 3 shows distribution for block sizes for different types.

| Number of blocks | Types | | | | |
|:---:|:---:|:---:|:---:|:---:|:---:|
| | Training | Test | Test Literal | Test Longest | Test Tallest |
| 2 | 14 | | | | |
| 3 | 31 | | | | |
| 4 | 36 | | | | |
| 5 | 5 | | | | |
| 6 | 9 | | | | |
| 7 | 6 | | | | |
| 8 | 6 | | | | |
| 9 | 6 | | | | |
| 10 | 5 | | 1 | | |
| 11 | 4 | | | | |
| 12 | 4 | 2 | 1 | 1 | 1 |
| 13 | 3 | 2 | | 1 | 1 |
| 14 | 5 | 2 | 1 | 1 | 1 |
| 15 | 4 | 2 | | 1 | 1 |
| 16 | 4 | 2 | 1 | 1 | 1 |

*Table 3.* Number of blueprint block distributions
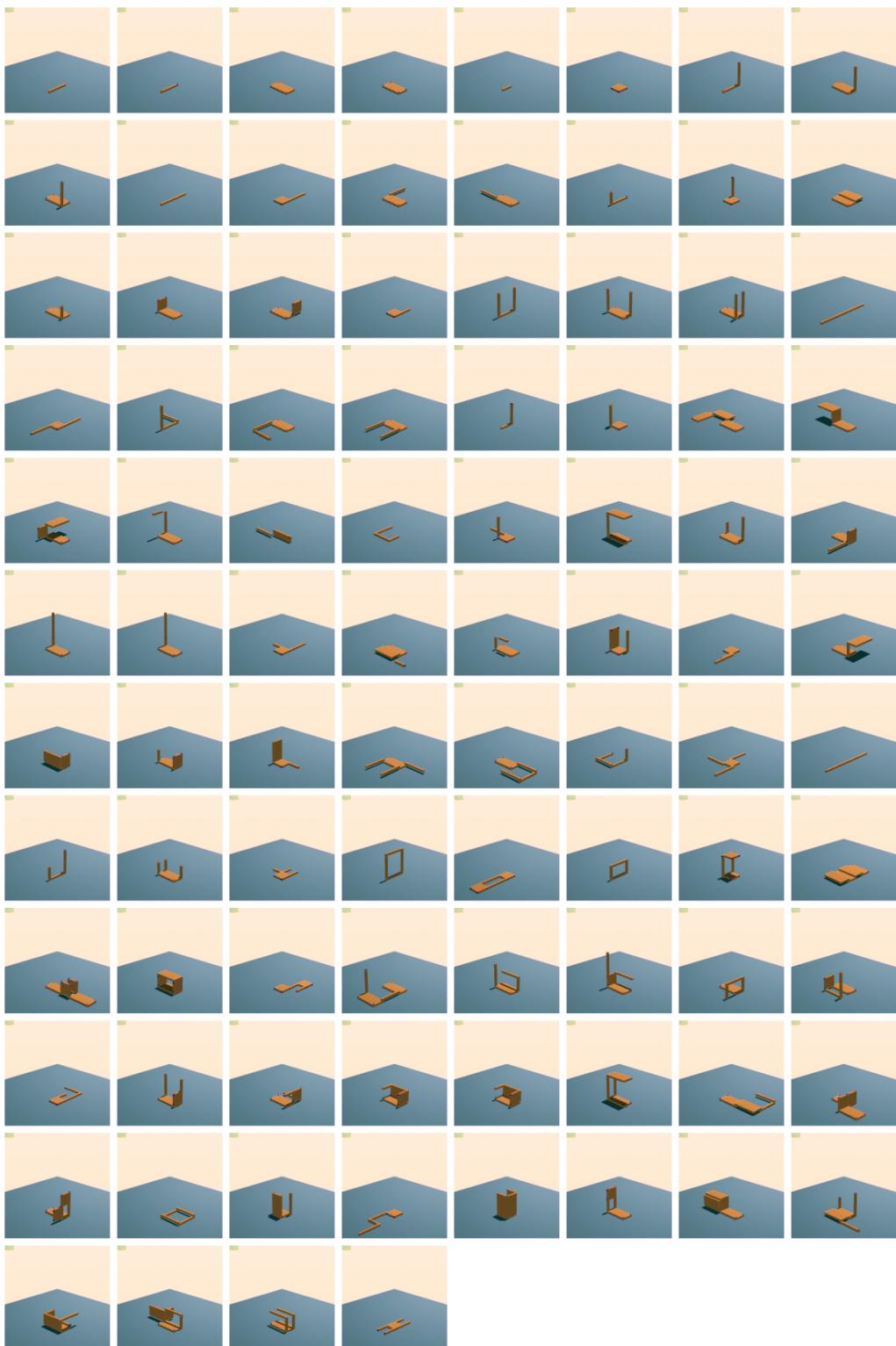
*Figure 6.* All blueprints used for training (of size 1 - 5).

*Figure 7.* All blueprints used for training (of size 6 - 16).
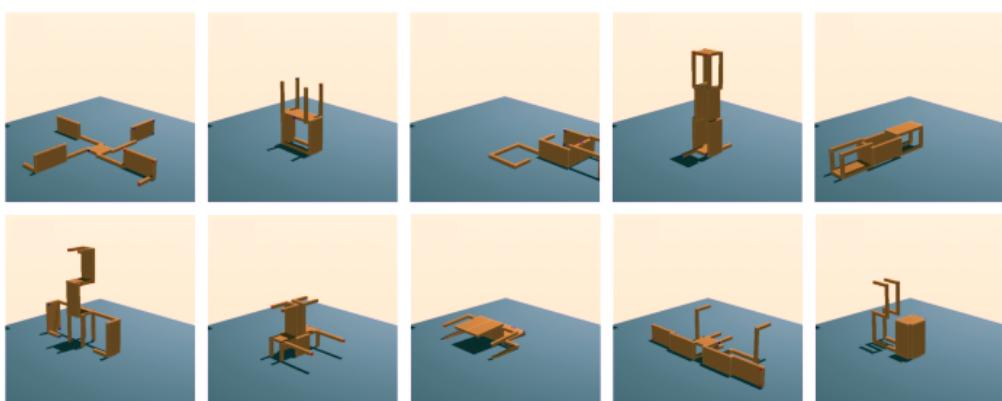


*Figure 8.* All blueprints used for testing (of size 12 - 16).
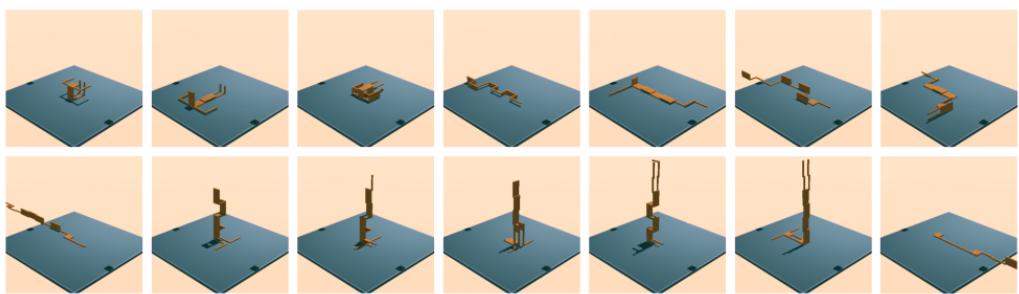
*Figure 9.* Blueprints used for literal, tallest, longest testing.

## B. Descriptions of Observation Space, Action Space, Rewards, Success Criterion

### B.1. Observation Space

**Per block observations on graph nodes**

- $z$ height from ground

- binary indicator for whether the block was being held in the previous timestep by some gripper

**Observations on directed graph edge from block A to block B**

- change in position for desired magnets to align

- change in orientation for desired magnets to align

- the difference between current block positional delta, and block positional delta in the blueprint

- the difference between current block orientation delta, and block orientation delta in the blueprint

- positional delta

- 0-1 indicator for whether blocks should be connected according to the blueprint

- 0-1 indicator for whether blocks should are currently connected

**Per gripper observations**

- orientation

- positional velocity

- rotational velocity

- 16-dimensional 0-1 vector showing which block the gripper was holding in the previous timestep

### B.2. Action Space

- one 16-dimensional one-hot vector per gripper, representing which block the gripper wants to manipulate

- a $16 \times 6$ dimensional matrix, representing desired position and rotational velocity per block, were it to be chosen by a gripper to be manipulated

### B.3. Rewards

In each step, the following rewards are computed and added together. The cumulative reward is then subtracted from the cumulative reward in the prior timestep, and the difference in cumulative rewards is returned as the environment reward for the current timestep.

- force penalty if two blocks, or a block and the ground, make heavy contact

- -1 for each magnetic connection that should not be connected

- dense reward based on position and orientation of each two magnets that should be connected

- +1 if two magnets that should be connected are connected

- dense reward based on position and orientation of each two blocks that should be attached to one-another

### B.4. Success Criterion

An episode is deemed successful when in some timestep all of the following success criterion are satisfied.

- magnetic connections required by the blueprint are made

- no extra magnets outside the blueprint definition are connected

- all blocks in the blueprint structure are in the correct relative position and orientation

## C. Graph Neural Network Architecture

```
1  import jax
2  import jax.numpy as jnp
3  import haiku as hk
4  import jraph
5
6  def build_GraphNetGAT_ppo_network(environment_spec):
7    action_dim = np.prod(environment_spec.actions.shape, dtype=int)
8
9    #### PPO NETWORK PARAMS
10   NUM_GN_GAT_LAYERS = 3
11   VALUE_MLP_LAYER_SIZES = [512, 512, 512]
12   #### GN_GAT LAYER PARAMS
13   NUM_HEADS = 4
14   NUM_READOUT_HEADS = 4
15   # NUM_HEADS = 8
16   QK_DIM = 64
17   M_DIM = 64
18   NODE_EMBED_DIM = 64
19   EDGE_EMBED_DIM = 64
20   GLOBAL_EMBED_DIM = 64
21   NODE_MLP_HID = 256
22   NODE_MLP_NUM_LAYERS = 1
23   GLOBAL_MLP_HID = 256
24   GLOBAL_MLP_NUM_LAYERS = 1
25
26   def update_edge_fn(edge_features, sender_features, receiver_features, global_features):
27     feats = jnp.concatenate([sender_features, edge_features], axis=-1)
28     m = hk.Linear(NUM_HEADS * M_DIM, with_bias=False, name='linear_message')(feats)
29     return (jnp.reshape(m, list(m.shape[:-1]) + [NUM_HEADS, M_DIM]), edge_features)
30
31   def update_node_fn(node_features, sender_features, receiver_features, global_features):
32     receiver_features = jnp.reshape(receiver_features, list(receiver_features.shape[:-2])
         + [-1])
33     x = jnp.concatenate([receiver_features], axis=-1)
34     residual = hk.Linear(node_features.shape[-1], with_bias=False, name='linear_node')(x)
35     y = node_features + residual
36     y = hk.LayerNorm(axis=-1, create_scale=True, create_offset=True,)(y)
37
38     h = hk.nets.MLP(
39         output_sizes=[NODE_MLP_HID]*NODE_MLP_NUM_LAYERS + [y.shape[-1]],
40         activation=jax.nn.relu,
41         name='mlp_node')(jnp.concatenate([y, global_features], axis=-1))
42     h = h + y
43     h = hk.LayerNorm(axis=-1, create_scale=True, create_offset=True,)(h)
44     return h
45
46   def update_global_fn(node_features, edge_features, global_features):
47     x = jnp.concatenate([global_features, node_features], axis=-1)
48     h = hk.nets.MLP(
49         output_sizes=[GLOBAL_MLP_HID]*GLOBAL_MLP_NUM_LAYERS + [GLOBAL_EMBED_DIM],
50         activation=jax.nn.relu,
51         name='mlp_global')(x)
```

```python
52      h = h + global_features
53      h = hk.LayerNorm(axis=-1, create_scale=True, create_offset=True,)(h)
54      return h
55
56  def attention_logit_fn(edge_features, sender_features, receiver_features,
        global_features):
57      edge_features = edge_features[1]
58      q = hk.Linear(NUM_HEADS * QK_DIM, with_bias=False, name='linear_query')(
59          jnp.concatenate([sender_features, edge_features], axis=-1))
60      q = jnp.reshape(q, list(q.shape[:-1]) + [NUM_HEADS, QK_DIM])
61      k = hk.Linear(NUM_HEADS * QK_DIM, with_bias=False, name='linear_key')(
        receiver_features)
62      k = jnp.reshape(k, list(k.shape[:-1]) + [NUM_HEADS, QK_DIM])
63      return jnp.sum(q * k, axis=-1, keepdims=True) / (QK_DIM**0.5)
64
65  def attention_reduce_fn(edge_features, weights):
66      return edge_features[0] * weights
67
68  def encode_graph(g):
69      return jraph.GraphMapFeatures(
70          embed_edge_fn=hk.Linear(EDGE_EMBED_DIM, name='linear_embed_edge'),
71          embed_node_fn=hk.Linear(NODE_EMBED_DIM, name='linear_embed_node'),
72          embed_global_fn=hk.Linear(GLOBAL_EMBED_DIM, name='linear_embed_global')
73      )(g)
74
75  @jax.vmap
76  def _ppo_graph_part(g):
77      g = encode_graph(g)
78      enc_edges = g.edges
79
80      for _ in range(NUM_GN_GAT_LAYERS):
81          g = jraph.GraphNetGAT(
82              update_edge_fn=update_edge_fn,
83              update_node_fn=update_node_fn,
84              attention_logit_fn=attention_logit_fn,
85              attention_reduce_fn=attention_reduce_fn,
86              update_global_fn=update_global_fn,
87          )(g)
88          g = g._replace(edges=enc_edges)
89
90      return g
91
92  def _ppo_graph_network(g):
93      input_g = g
94      input_globals = jnp.squeeze(g.globals, axis=1)
95      input_globals = hk.Linear(GLOBAL_EMBED_DIM, with_bias=False)(input_globals)
96
97      g = _ppo_graph_part(g)
98      output_g = g
99      g_globals = jnp.squeeze(g.globals, axis=1)
100     g_nodes = g.nodes # B x n_nodes x NODE_DIM
101
102     ######### The policy part
103     # first predict an action distribution per block
104     # for now lets try without further processing of the g_nodes
105     node_feats = g_nodes
106
107     BLOCK_ACT_DIM = 6
108     block_act_locs = hk.Linear(
109         BLOCK_ACT_DIM,
110         w_init=hk.initializers.VarianceScaling(1e-4),
111         b_init=hk.initializers.Constant(0.))(node_feats)
112     block_act_scales = hk.Linear(
113         BLOCK_ACT_DIM,
114         w_init=hk.initializers.VarianceScaling(1e-4),
```

```
115          b_init=hk.initializers.Constant(0.))(node_feats)
116      block_act_scales = jax.nn.softplus(block_act_scales) + 1e-6
117
118      # now for each gipper generate logits for which block it wants to move
119      # for now lets try without further processing of g_globals
120      global_feats = jnp.concatenate([input_globals, g_globals], axis=-1)
121
122      NUM_GRIPS = 2
123      grip_keys = hk.Linear(NUM_GRIPS * QK_DIM, with_bias=False)(global_feats)
124      grip_keys = jnp.reshape(grip_keys, [grip_keys.shape[0], NUM_GRIPS, 1, QK_DIM]) # B x
     G x 1 x QK_DIM
125
126      block_active = jnp.reshape(input_g.globals, [input_g.globals.shape[0], NUM_GRIPS,
     -1])
127      block_active = block_active[:, :, -node_feats.shape[1]:]
128      block_active = jnp.transpose(block_active, [0, 2, 1])
129      block_keys = hk.Linear(QK_DIM, with_bias=False)(
130          jnp.concatenate([node_feats, block_active], axis=-1)) # B x n_nodes x QK_DIM
131
132      block_keys = jnp.reshape(block_keys, [block_keys.shape[0], 1, block_keys.shape[1],
     QK_DIM]) # B x 1 x n_nodes x QK_DIM
133
134      grip_logits = jnp.sum(grip_keys * block_keys, axis=-1) / (QK_DIM**0.5) # B x G x
     n_nodes
135
136      policy_output = PolicyOutput(
137          block_act_locs=block_act_locs,
138          block_act_scales=block_act_scales,
139          grip_logits=grip_logits,)
140
141      ######### The baseline value part
142      # for now just doing it based on the globals
143
144      # get the baseline value
145      input_globals = jnp.squeeze(input_g.globals, axis=1)
146      input_globals = hk.Linear(GLOBAL_EMBED_DIM, with_bias=False)(input_globals)
147      g_globals = jnp.squeeze(output_g.globals, axis=1)
148      g_globals = hk.Linear(GLOBAL_EMBED_DIM, with_bias=False)(g_globals)
149      all_globals = jnp.concatenate([input_globals, g_globals], axis=-1)
150
151
152      # trying something
153      grip_logits_softmax = jax.lax.stop_gradient(jax.nn.softmax(grip_logits, axis=-1)) #
     dont want the value loss to influence gripper choice
154      grip_logits_softmax = jnp.reshape(grip_logits_softmax, [grip_logits_softmax.shape[0],
      -1])
155      all_globals = jnp.concatenate([all_globals, grip_logits_softmax], axis=-1)
156
157
158      mlp_inputs = all_globals
159
160      value_network = hk.nets.MLP(
161          output_sizes=VALUE_MLP_LAYER_SIZES + [1],
162          activation=jax.nn.relu)
163      value = value_network(mlp_inputs)
164      value = jnp.squeeze(value, axis=-1)
165
166      return (policy_output, value)
167
168  return _ppo_graph_network
```

## D. Curriculum

The following python script represents how our training curriculum is defined.

```python
import numpy as np
from scipy import special

class Curriculum():
  def __init__(
      self,
      tau: float = 0.2,
      temp: float = 0.5,
      decay: float = 0.99,
      curriculum_logdir: str = ''):
    self.tau = tau
    self.temp = temp
    self.decay = decay
    self.blueprint_indices = blueprint_indices

    self.success_rates = np.zeros((len(blueprint_indices),), dtype=np.float32)

    self.probs = np.ones((len(blueprint_indices),), dtype=np.float32)
    self.probs *= 1./float(len(blueprint_indices))
    self.probs_history = self.probs[None, :].copy()

    self._steps = 0

  def update(self, batch):
    """Updates the curriculum."""
    successes = batch['success']
    blueprint_indices = batch['blueprint_indices']

    for s, b_i in zip(successes, blueprint_indices):
      self.success_rates[i] = (1. - self.tau) * self.success_rates[i] + self.tau * s

    self.success_rates *= self.decay # ensures there are not things that never get
    sampled
    self.probs = special.softmax((1. - self.success_rates) / self.temp)

  def sample(self):
    return int(np.random.choice(np.arange(len(blueprint_indices)), p=self.probs))
```

## E. ResNet Baseline Details

When ablating the role of relation inductive biases in the agent architecture, we replace the graph neural network encoder in our default agent (described in Section 4) with a residual network encoder defined as follows:

```python
import jax
import haiku as hk

NUM_RESNET_BLOCKS = 4
HIDDEN_DIM = 1024

uniform_initializer = hk.initializers.VarianceScaling(
    scale=0.333, mode='fan_out', distribution='uniform')

def residual_block(x):
    h = x
    h = hk.Linear(HIDDEN_DIM, w_init=uniform_initializer)(h)
    h = jax.nn.relu(h)
    h = hk.Linear(HIDDEN_DIM, w_init=uniform_initializer)(h)
    x = x + h
    return hk.LayerNorm(axis=-1, create_scale=True, create_offset=True)(x)

def resnet_encoder(x):
    for _ in range(NUM_RESNET_BLOCKS):
        x = block(x)
```

# F. Additional Figures
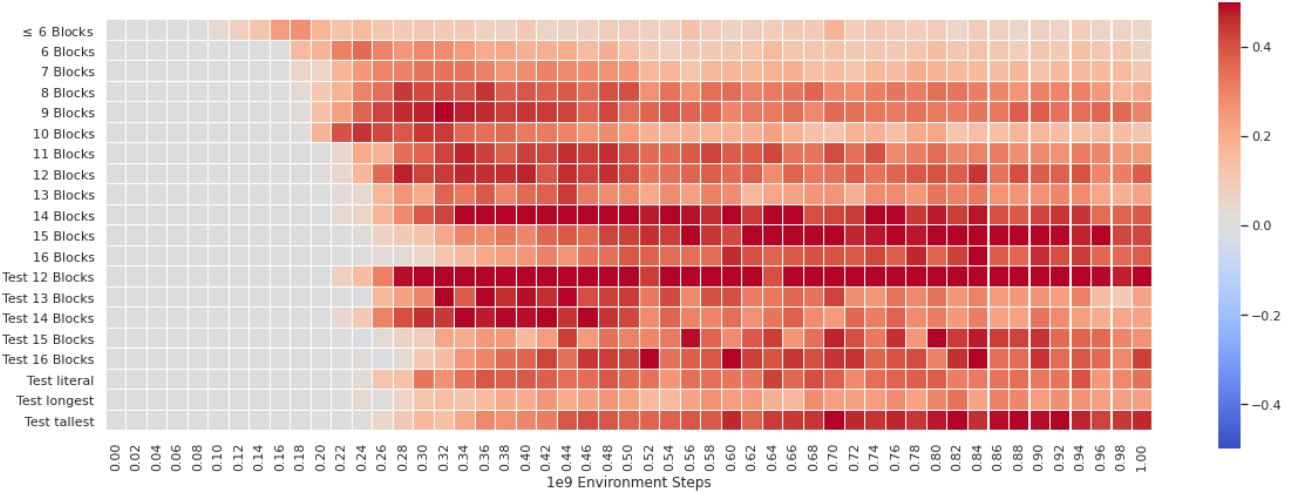


*Figure 10.* Plot showing `bimanual success rate` minus `single gripper success rate`. In our proposed magnetic assembly tasks, agents benefit strongly from having access to 2 grippers compared to 1. Full details in Figure 13.
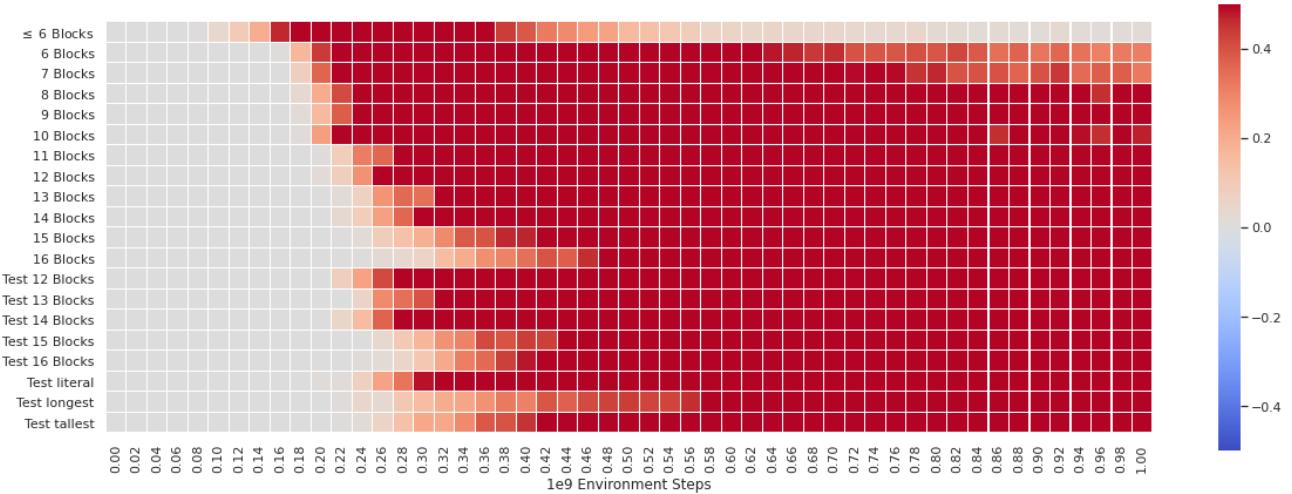


*Figure 11.* Plot showing `default agent success rate` minus `no attention agent success rate`, clearly demonstrating the necessity of the graph attention mechanism. Full details in Figure 14.

## G. Full Size Version of All Plots

Due to the large size of the plots, figures appear starting from the next page.
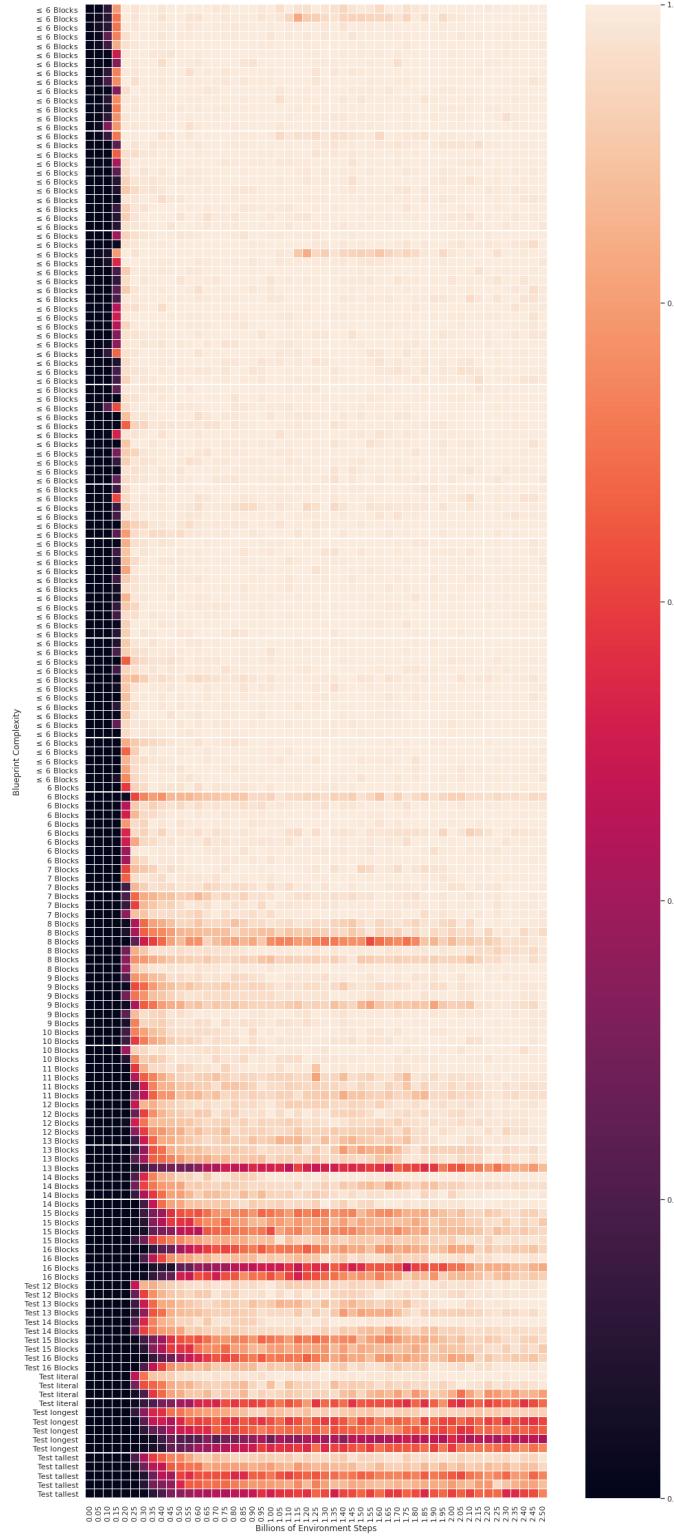
*Figure 12.* Plot presenting success rates for each blueprints throughout training, for our default agents (Section 4) and training procedure (Section 5). Each square represents the success rate on 40 episodes, evaluated at that point in training, averaged across two training runs. Our results demonstrate (1) the key role of scale in training successful agents, and that (2) after a long period of training, agents generalize well to complex held-out blueprints.
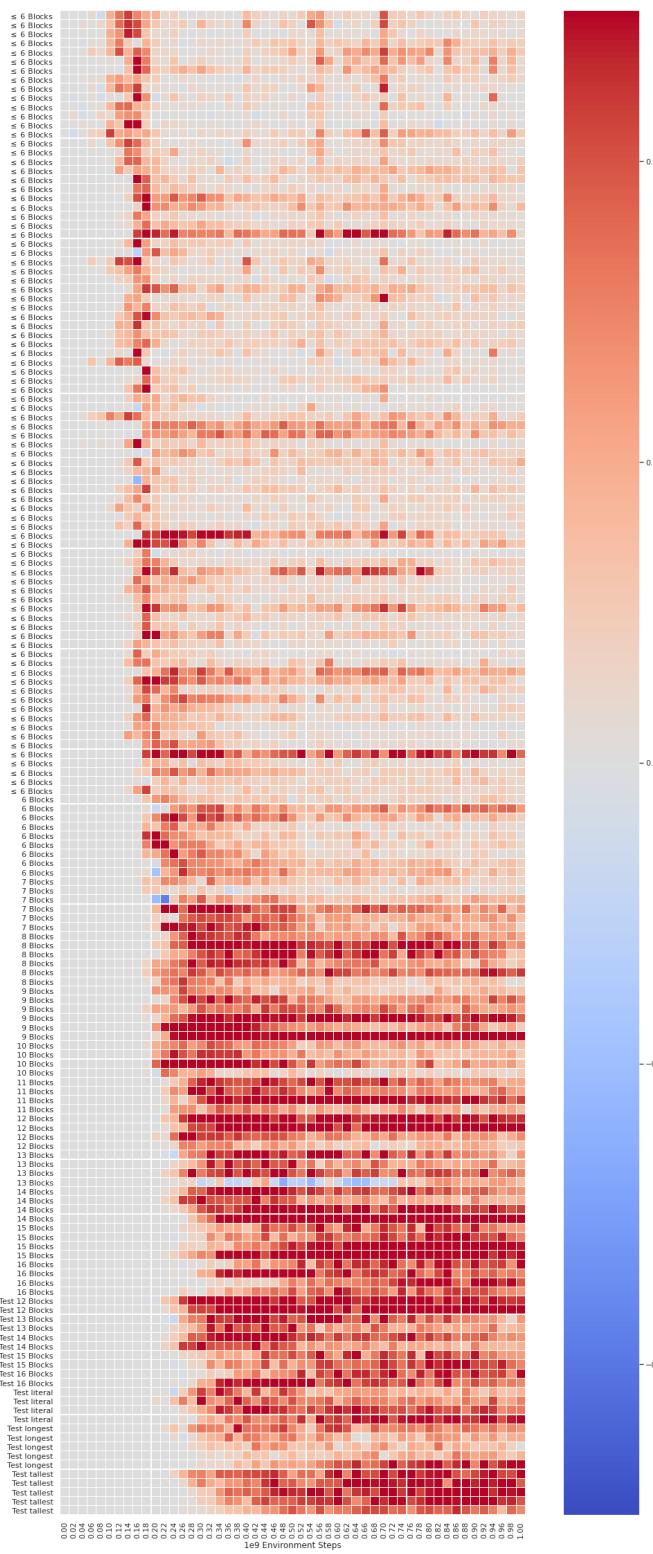
*Figure 13.* Plot showing `bimanual success rate` minus `single gripper success rate`. In our proposed magnetic assembly tasks, agents benefit strongly from having access to 2 grippers compared to 1.
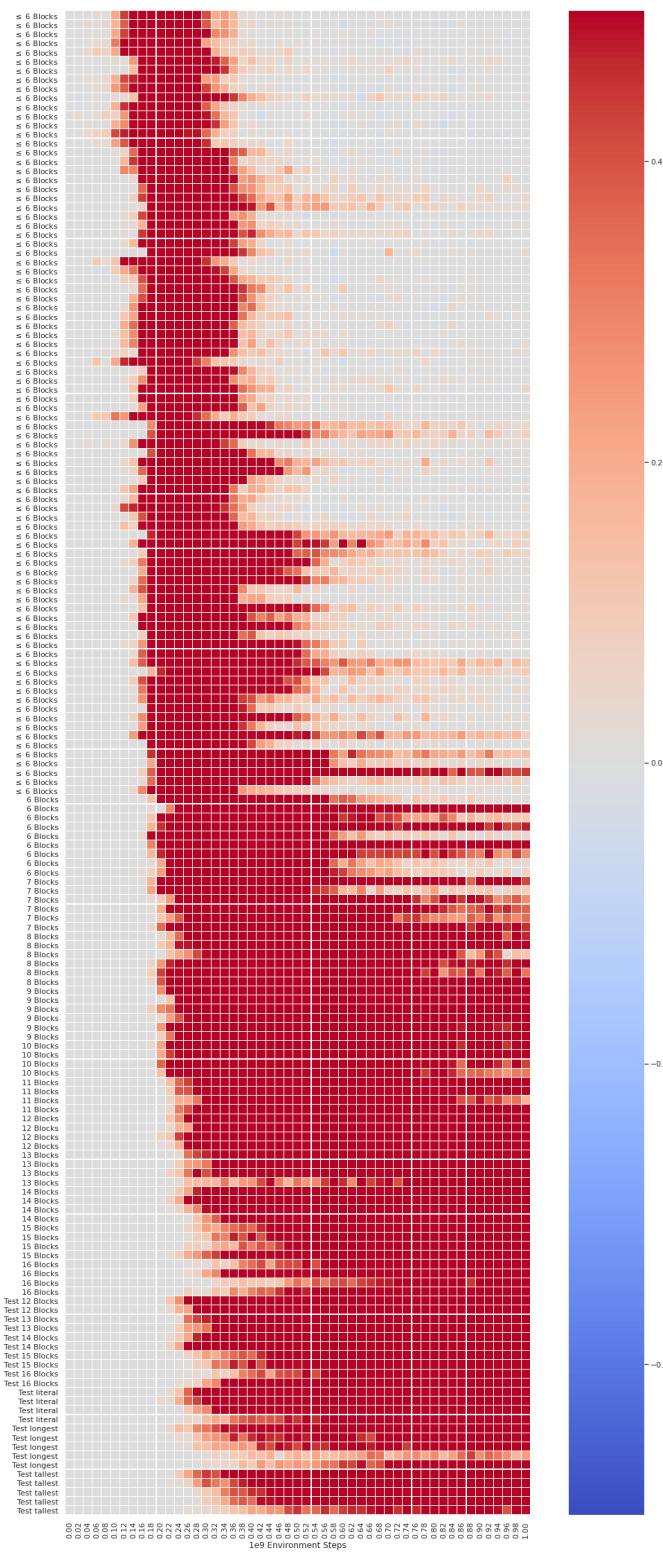
*Figure 14.* Plot showing `default agent success rate` minus `no attention agent success rate`, clearly demonstrating the necessity of the graph attention mechanism.
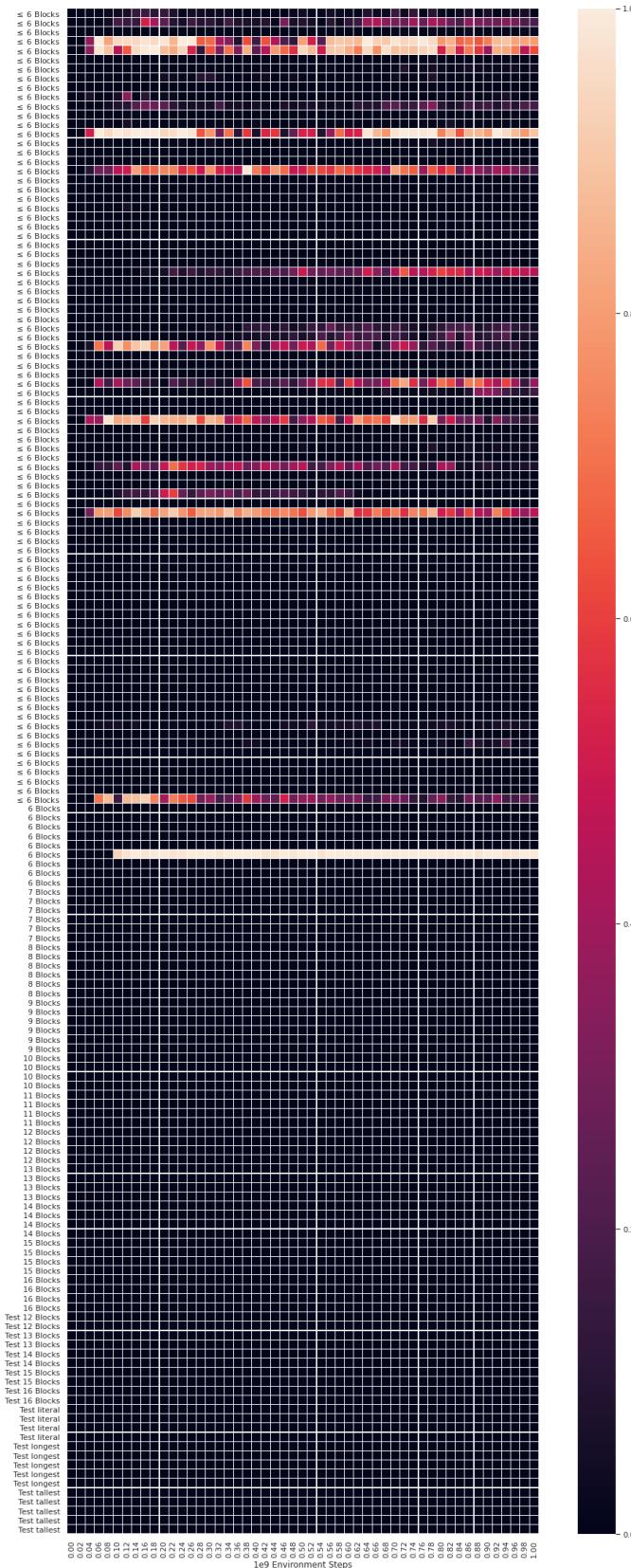
*Figure 15.* Plot showing success rates when training on only blueprint 6_005.
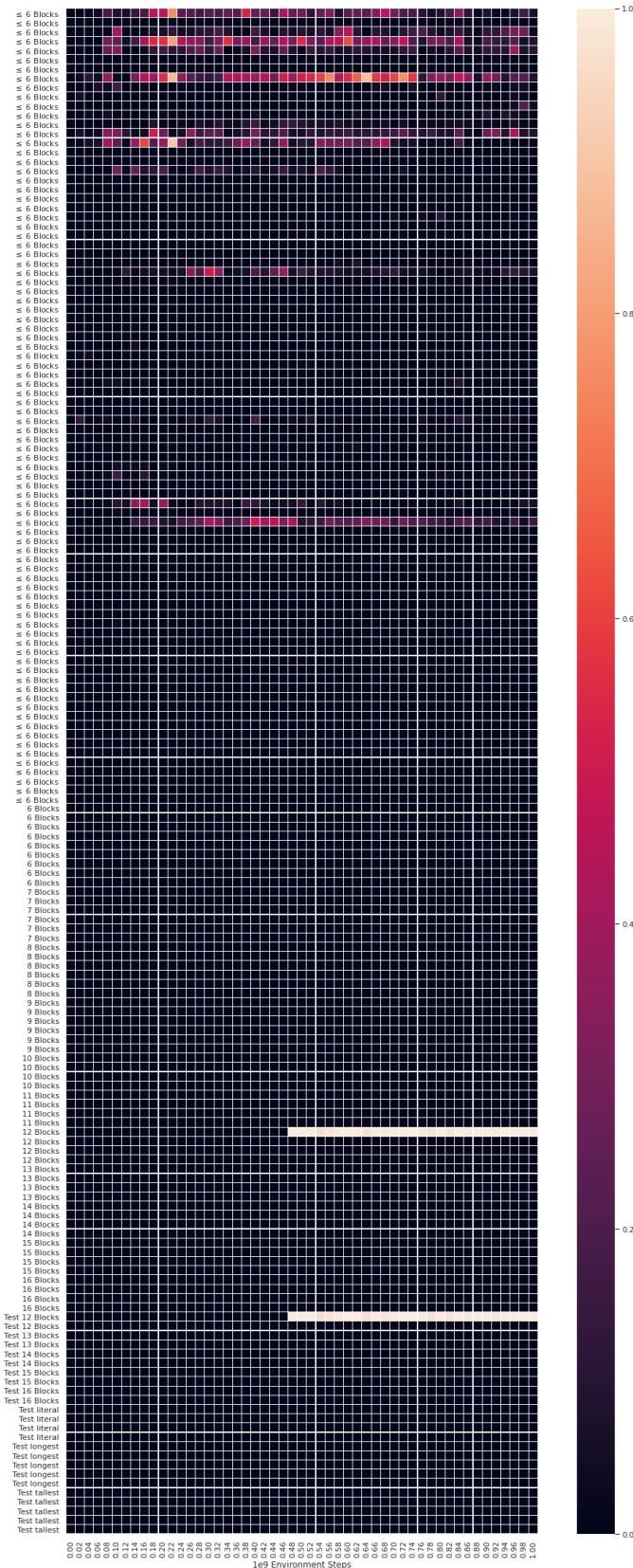
*Figure 16.* Plot showing success rates when training on only blueprint 12_000.
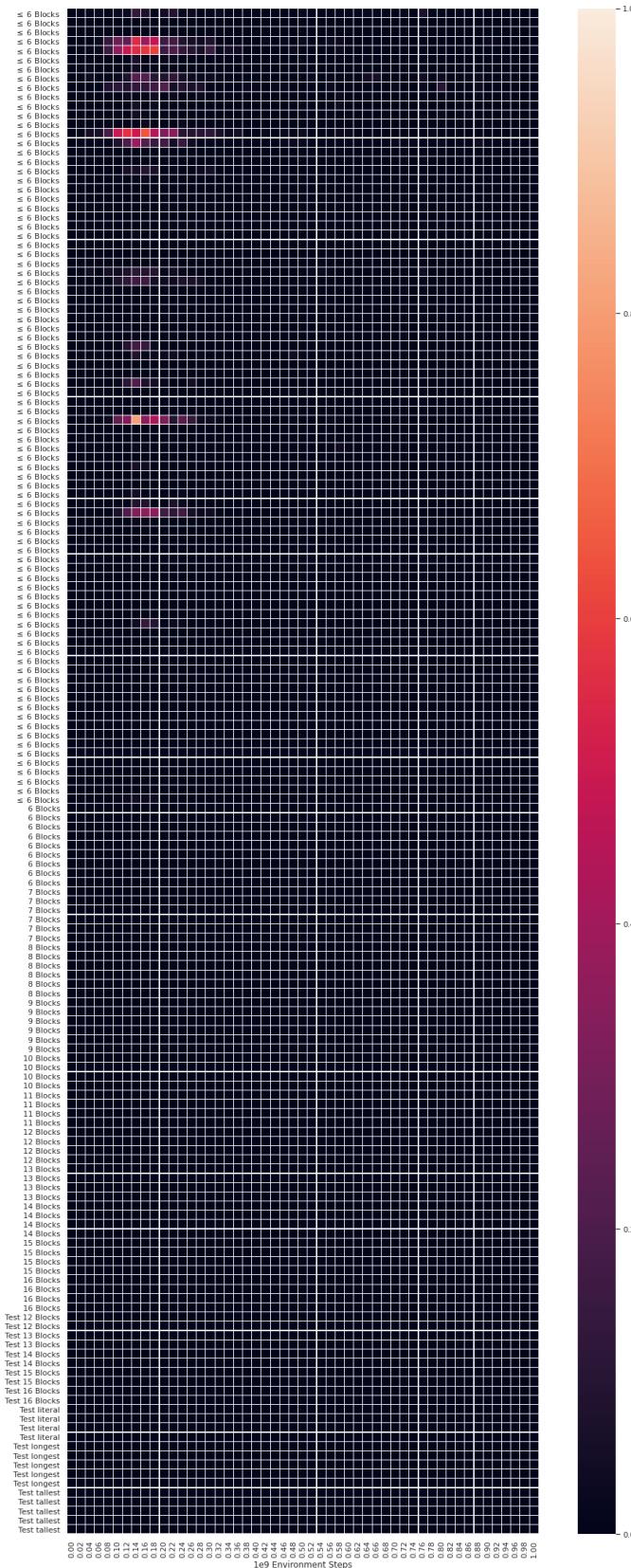
*Figure 17.* Plot showing success rates when training on only blueprint 16_001.
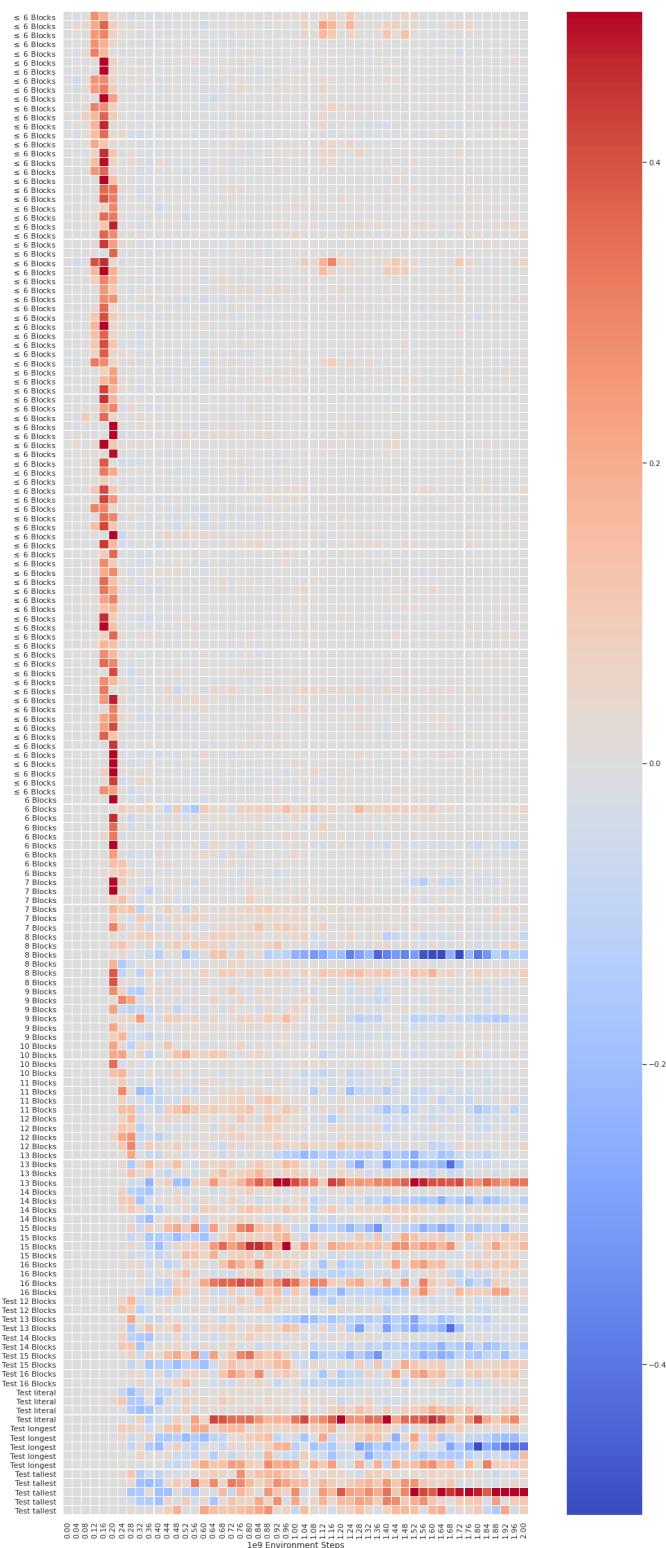
*Figure 18.* Plot showing `with curriculum agent (seed 0) success rate` **minus** `no curriculum agent success rate`.
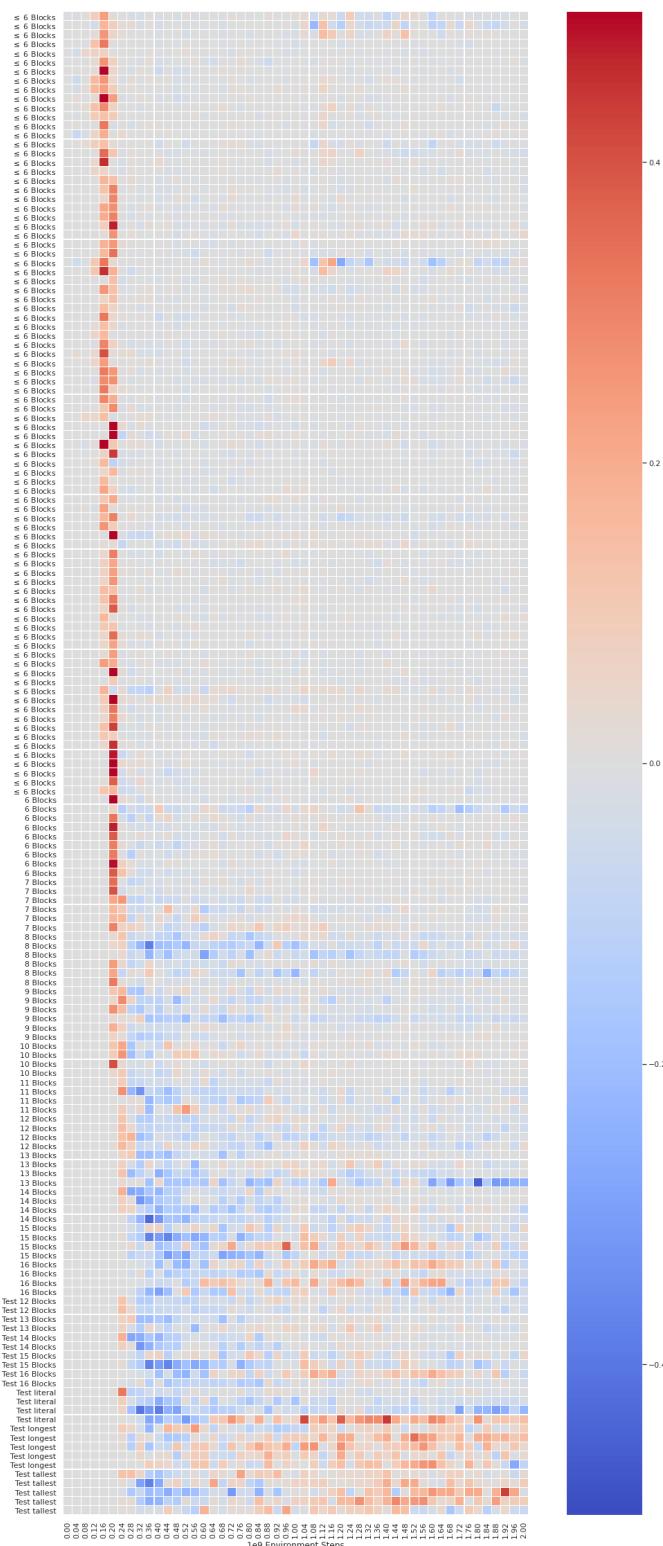
*Figure 19.* Plot showing `with curriculum agent (seed 1) success rate` minus `no curriculum agent success rate.`
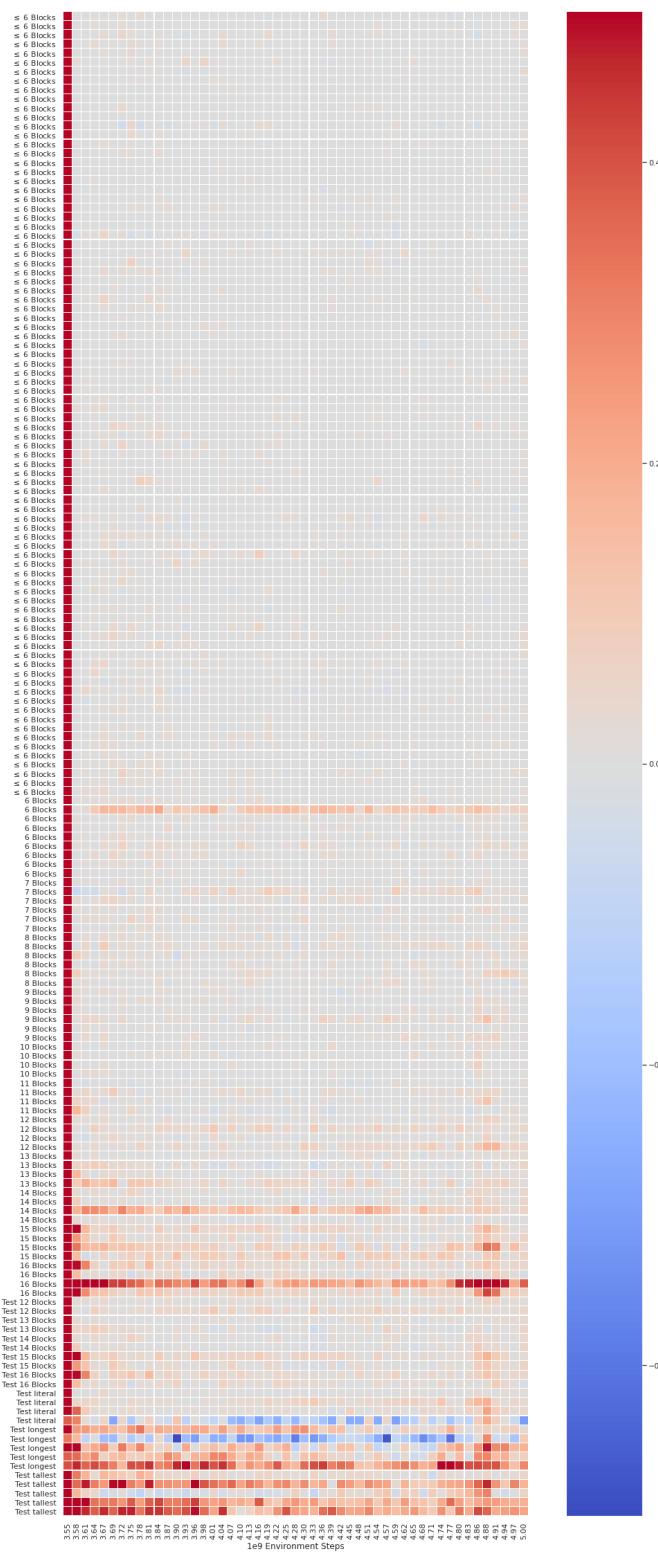
*Figure 20.* Plot showing `default agent success rate` minus `forked agent fine-tuned with gripper transition delay 2 success rate`.
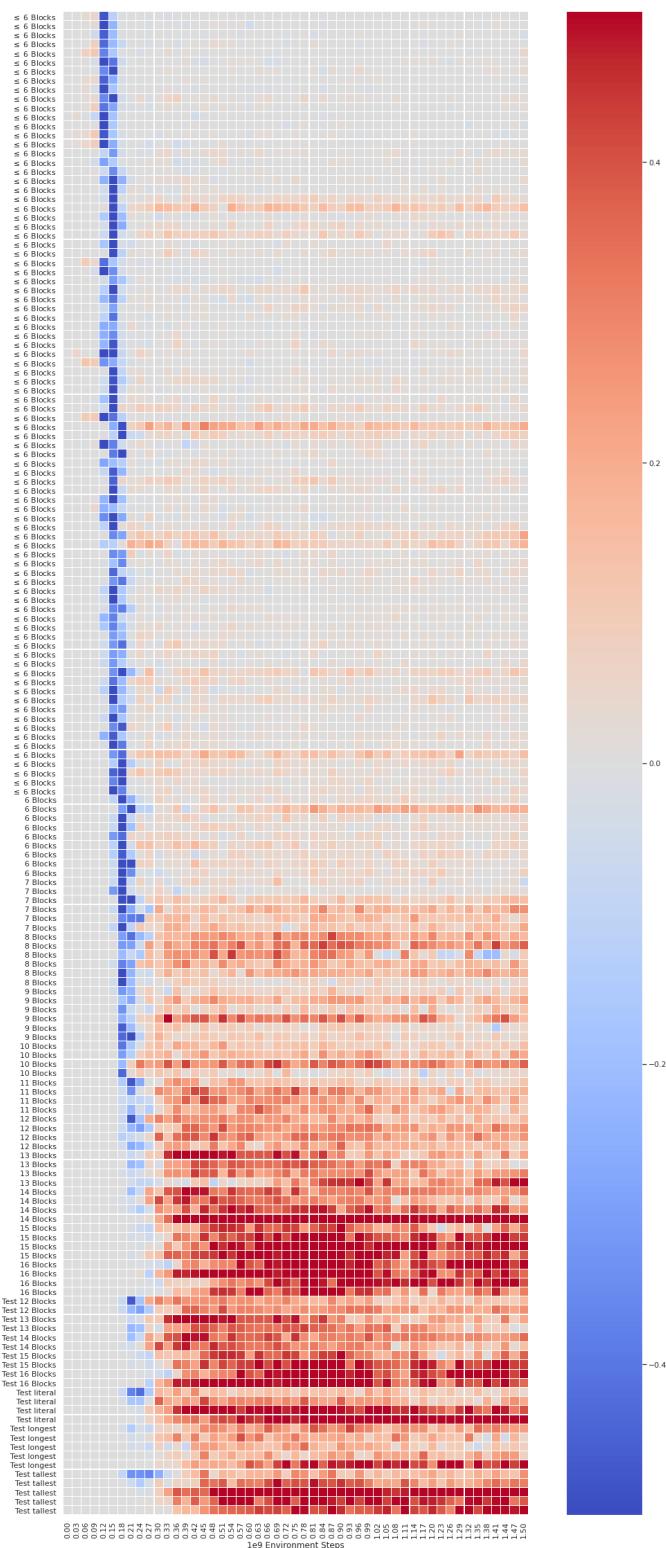
*Figure 21.* Plot showing `default agent success rate` minus `agent trained with gripper transition delay 2 from scratch success rate`.
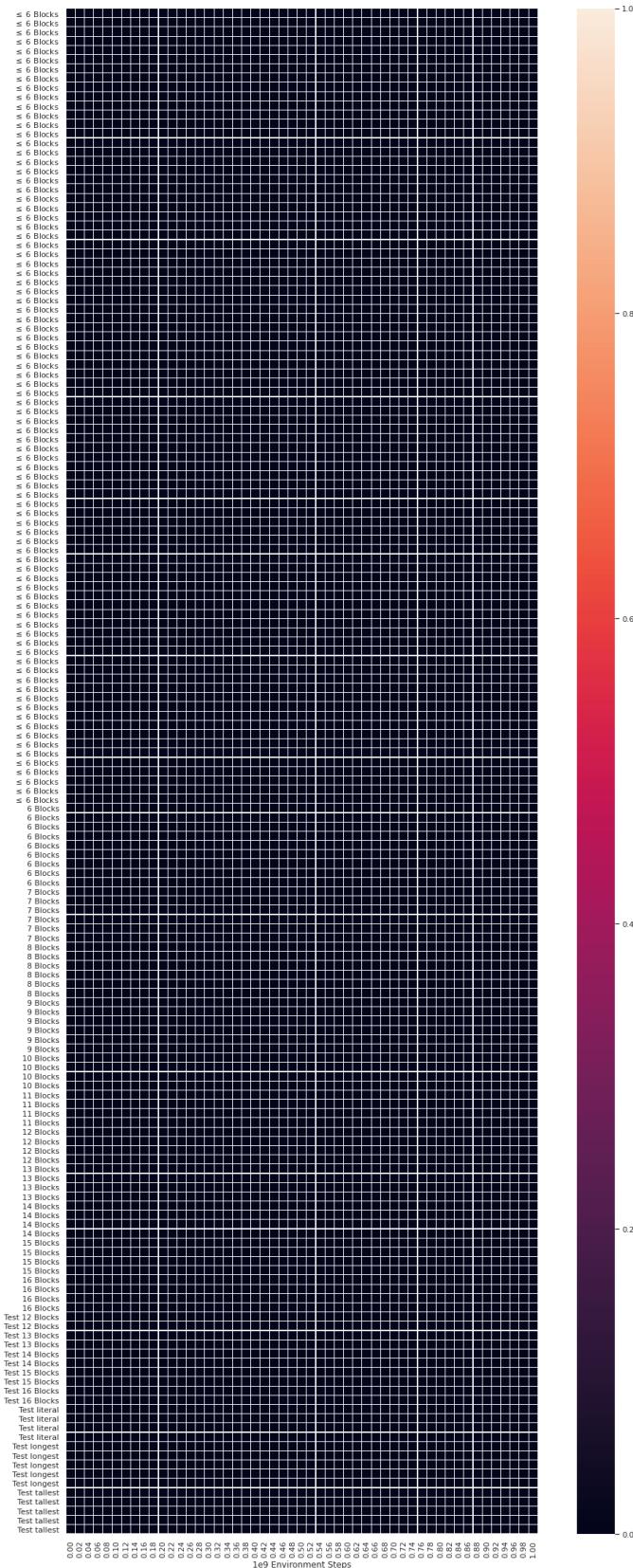
*Figure 22.* Plot showing the success rate of a residual network architecture trained on the full training set of blueprints.
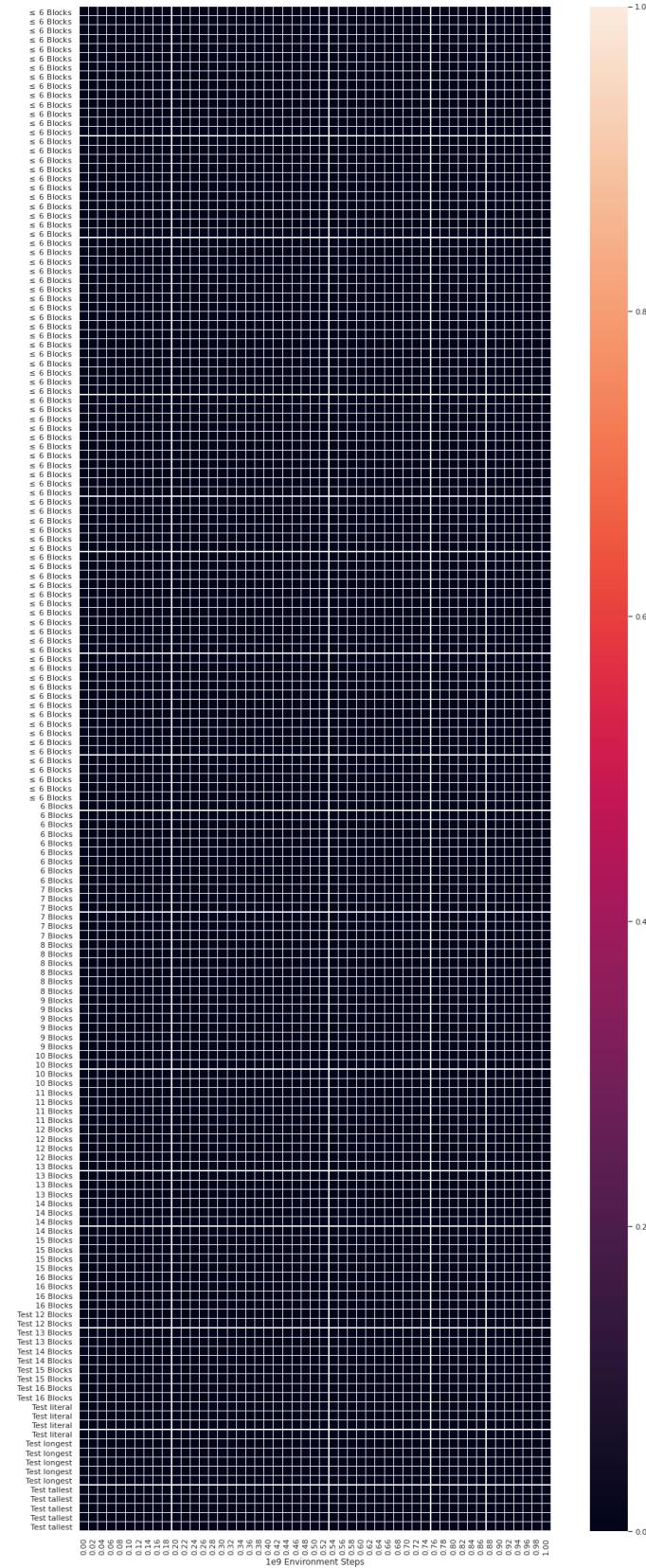
*Figure 23.* Plot showing the success rate of a residual network architecture trained on the subset of the training set of blueprints with ≤ 6 blocks.
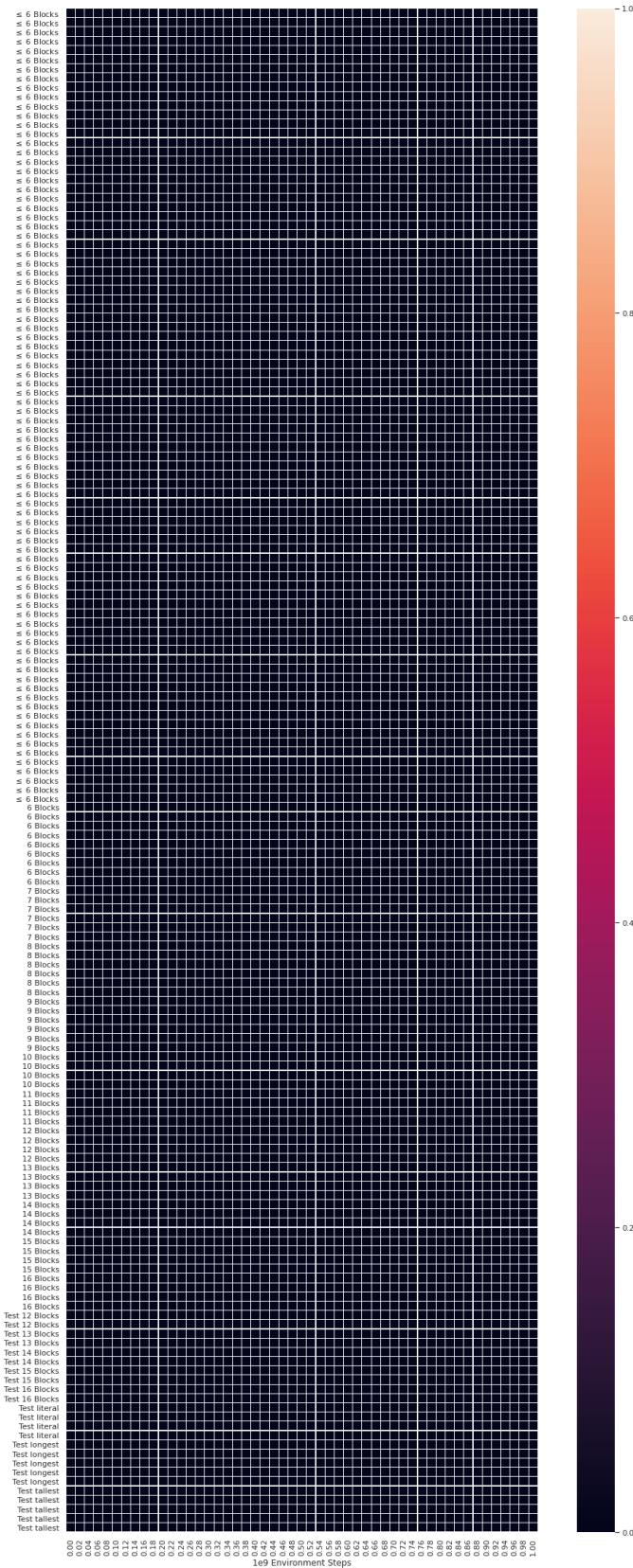
*Figure 24.* Plot showing the success rate of a residual network architecture trained only on a single blueprint requiring 6 blocks.