# CURATING VISUAL INFORMATION IN SEQUENTIAL DECISION MAKING PROBLEMS

AARON WALSMAN

*A dissertation*
*submitted in partial fulfillment of the*
*requirements for the degree of*

Doctor of Philosophy

University of Washington
2023

*Reading Committee:*
Dieter Fox, Chair
Ali Farhadi
Gilbert Bernstein

*Program Authorized to Offer Degree:*
Paul G. Allen School of Computer Science & Engineering

University of Washington

**ABSTRACT**

# CURATING VISUAL INFORMATION IN SEQUENTIAL DECISION MAKING PROBLEMS

Aaron Walsman

*Chair of the Supervisory Sommittee:*
Dieter Fox
Paul G. Allen School of Computer Science & Engineering

Images often contain both too much and not enough information. When an image is cluttered and contains many irrelevant details, this extra information can make it challenging to learn which parts are necessary, and how to act accordingly. At the same time, an image is a narrow 2D slice of a 3D world. This means that factors such as object occlusions and a limited field of view necessarily limit the amount of information available at any given time. These two problems, informational clutter and partial observability fundamentally concern the agent's access to information. When the agent has too much information, it must learn to discard that which is irrelevant. When it does not have enough, the agent must come up with information gathering actions to acquire the information that it lacks. We explore these issues in a variety of sequential decision making problems in which an agent must repeatedly interact with an environment in order to accomplish some objective. These settings include goal-conditioned object retrieval in a simulated kitchen, part-based assembly problems using LEGO bricks, and escaping giant monsters in the first-person video game DOOM. In all of these settings, we show that by carefully considering an agent's access to information, we can significantly improve its performance.

To my wife Zoey,
my little bird,
my nuclear reactor,
my love.

# ACKNOWLEDGEMENTS

I first must thank my family, my wife Zoey, my parents Sam and John, my brother Arlo. Producing the work described in this dissertation would not have been possible without their love and emotional support.

Secondly, I must thank the Graduate Christian Fellowship at the University of Washington, specifically Geoff and Ashley Van Dragt, as well as Calvin Chen who tirelessly embodied Jesus' call to lead through service.

## RESEARCH

Next I must thank my advisors Dieter Fox and Ali Farhadi, who pushed me through many challenges. The research here would not be possible without their inspiration and constructive engagement. I must also thank Siddhartha Srinivasa, my masters advisor for inspiring me to pursue a PhD in the first place.

I would also like to thank my other committee members, Adriana Schultz, Gilbert Bernstein and Ashis Banerjee. Their suggestions and fruitful discussion have greatly improved this work.

I must also thank my other collaborators on these projects, Yonatan Bisk, Saadia Gabriel, Dependra Misra, Yoav Artzi, Yejin Choi, Klemen Kotar, Muru Zhang, Adam Fishman, Karthik Desingh and Sanjiban Choudhury.

The many members of the Robotics and State Estimation Lab at the University of Washington not only helped shape and refine these ideas over the years, but also provided support and camaraderie through the ups and downs of graduate studies. Specifically Richard Newcombe, Tanner Schmidt, Arunkumar Byravan, Connor Schenck, Daniel Gordon, Chris Xie, Amnon Horowitz, Junha Roh, Karthik Desingh, Xiangyun Meng, Yu Xiang, Adam Fishman, Yi Li, Mohit Shridhar,

Zoey Chen, Vinitha Ranganeni, Wentao Yuan, Jiafei Duan, Marius Memmel and Hellen Wang.

Similarly, the RAIVN lab at the University of Washington was an invaluable source of wisdom and inspiration, especially our weekly reading group, which was a constant source of new ideas and inspiration. Specifically Mark Yatskar, Junyuan Xie, Nancy Wang, Minjoon Seo, Hessam Bagherinezhad, Keivan Alizadeh-Vahid, Kiana Ehsani, Joe Redmon, Keunhong Park, Max Horton, Rowan Zellers, Kuo-Hao Zheng, Mitchell Wortsman, Sarah Pratt, Vivek Ramanujan, Reza Salehi, Matt Wallingford, James Park, Aditya Kusupati, Gabriel Ilharco, Ainaz Eftekhar and Matt Dietke.

I must also thank several members of the Personal Robotics Lab, including Pras Velagapudi, Michael Koval, Jen King, Laura Herlant, Anka Dragan, Chris Dellin, Gilwoo Lee and Brian Hou.

I would also like to thank the many members of AI2 and Apple who greatly contributed to discussions in the RAIVN group over the years, specifically Aniruddha Kembhavi, Mohammad Rastegari, Roozbeh Mottaghi and Luca Weihs.

I must also thank several members of the larger robotics and computer science community at the University of Washington who contributed fruitful discussions to this research. Specifically Abhishek Gupta, Kevin Jamieson, Sham Kakade, Emo Todorov, Maya Cakmak, Byron Boots, Ari Holtzman, Jesse Thomason, John Thickstun, Leah Perlmutter, Kira Goldner, Kay Ke, Robbie Weber, Max Forbes, Andrew Wagenmaker Roy Or-El, Xuan Luo, Chuning Zhu, Kendall Lowery, Vikash Kumar and Svetsoslav Kolev.

Finally I have to thank the fantastic undergraduate and masters students who worked with me on various projects over the years, Weilin Wang, Klemen Kotar and Muru Zhang. They provided an abundant source of inspiration and were able to breathe new life and enthusiasm into these projects at critical junctures.

Finally I would like to thank the many contributors to the LDraw project, the Open Model Repository and the LDCad metadata who have made their data freely available online under generous licensing agreements. These online resources have been created and are maintained by a large group of volunteers. These data sources were vital to the construction of the LTRON LEGO simulator which is an integral component of this dissertation.

The following authors have contributed parts or parts of parts to the primary LDraw database. This database contains the geometric shape of over 13,000 individual brick types, and is used in LTRON as the primary source of renderable geometry.

*Philippe Hurbain [Philo], Magnus Forsberg [MagFors], James Jessiman, Steffen [Steffen], Chris Dee [cwdee], Michael Heidemann [mikeheide], J.C. Tchang [tchang], Gerald Lasser [GeraldLasser], Alex Taylor [anathema], Guy Vivan [guyvivan], Tore Eriksson [Tore_Eriksson], Willy Tschager [Holly-Wood], Max Martin Richter [MMR1988], Damien Roux [Darats], Andy Westrate [westrate], Christian Neumann [Wesley], Ulrich R der [UR], Steve Bliss [sbliss], Franklin W. Cain [fwcain], Rolf Osterthun [Rolf], Massimo Maso [Sirio], Santeri Piippo [arezey], Marc Klein [marckl], Niels Karsdorp [nielsk], Evert-Jan Boer [ejboer], Johann Eisner [technicbasics], Donald Sutter [technog], Vincent Messenet [Cheenzo], Joerg Sommerer [Brickaneer], Paul Easter [pneaster], Daniel Goerner [TK-949], Bernd Broich [bbroich], Nils Schmidt [BlackBrick89], Stan Isachenko [angmarec], William Howard [WilliamH], John Van Zwieten [jvan], Howard Lande [HowardLande], Thomas Burger [grapeape], Owen Burgoyne [C3POwen], Orion Pobursky [OrionP], Takeshi Takahashi [RainbowDolphin], Mikkel Bech Jensen [gaia], Arne Hackstein, John Riley [jriley], Tim Gould [timgould], Mark Kennedy [mkennedy], Tony Hafner [hafhead], Kevin Roach [KROACH], Tim Lampmann [L4mpi], Greg Teft [gregteft], Christophe Mitillo [Christophe_Mitillo], Leonardo Zide, Sylvain Sauvage [SLS], Merlijn Wissink [legolijntje], Mark Chittenden [mdublade], Carsten Schmitz [Deckard], Jaco van der Molen [Jaco], Luis E. Fernandez*

*[lfernand], Miklos Hosszu [hmick], Sascha Broich, Lutz Uhlmann [El-Lutzo], Ronald Vallenduuk [Duq], Matt Schild [mschild], Thomas Woelk [t.woelk], Manfred Moolhuysen, Ross Crawford [rosco], Bertrand Lequy [Berth], Imre Papp [ampi], Remco Braak [remco1974], George Barnes [glbarnes], Jude Parrill [theJudeAbides], Marc Giraudet [Mad_Marc], Ludo Soete [ludo], El'dar Ismagilov [Eldar], Bjoern Sigve Storesund [Storesund], Tomas Kralicek [RabbiT_CZ], John Troxler [Gargan], Jonathan Wilson [jonwil], Jeff Boen [onyx], Lutz Uhlmann, Peter Watts [FrozenPea], Niels Bugge [SirBugge], Marc Schickele [samrotule], Alexandre Bourdais [x-or], Jeff Boen, Ingolf Weisheit [stahlwollschaf], Andrew Ananjev [woozle], Damien Guichard [BrickCaster], Heiko Jelnikar [KlotzKiste], Yann Bouzon [Zaghor], Stephan Meisinger [smr], Lee Gaiteri [LummoxJR], Marek Idec [Maras], Lars C. Hassing [larschassing], Nathan Wright, Remco Braak, Kevin Clague [kclague], Larry Pieniazek [lar], Matthew J. Chiles [mchiles], Guus-Jan Wijnhoven [guus], Victor Di Rienzo [tatubias], Shimpei Ohsumi [Shimpei-Ohsumi], Christian M. Angele [cma_1971], Sven Moritz Hein [smhltec], Ildefonso Zanette [izanette], Ronald Scott Moody [rmoody], Paul Izquierdo Rojas [pir], Joseph H. Cardana, Taylor Bangs [DoomTay], Peter Lind [peterlinddk], Adriano Aicardi, Jonathan P. Brown, Karim Nassar, Dee Earley [DeannaEarley], Brent Jackson [bjackson], Lance Hopenwasser [cavehop], Bram Lambrecht, Ishino Keiichiro, Joachim Probst, Yu Zhang [ishkafel], David Manley [djm], Joshua Delahunty [dulcaoin], Ignacio Fernandez Galvan [Jellby], Heather Patey, Adam Howard [Whist], N. W. Perry [Plastikean], Matthew Morrison [cuddlyogre], Dennis Osborn, Robert Sexton [rhsexton], Sybrand Bonsma [Sybrand], Remco Canten [rempie], R. M. Rodinsky [dublar], Paul Schelleman [schellie], Bob LeVan [kyphurious], Dave Schuler, Jens Bauer [rockford], Jan Folkersma [Stinky], Graham Wilkes [remorse], Damien Duquennoy, Svend Eisenhardt [eisenhardt], Enzo Silvestri [ienzisolves], Bert Van Raemdonck [BEAVeR], Andreas Weissenburg [grubaluk], Amnon Silverstein [Amnon], Frits Blankenzee, Zoltan Keri [kzoltan82], Alex Forencich [aforencich], Daniele Benedettelli [benedettelli], Rafael Skibicki [Rola], Carlos Arbesu [NXTbesu], Philip Peickert [mr51flip], Sam Roberts [sroberts], Gene Welborn [dtaax], Allister Mclaren [amclaren], Joao Almeida [TullariS], Derrick Chiu [LordAdmiral], Niklas Buchmann [NiklasB], Alex Seeley [alex], Paolo*

*Campagnaro [pcampagn], Ryan Dennett, Maciej Kowalik [Madmaks], Ian Reid [Ian_Reid], Jeff Stembel, C.L.Rasmussen [johnny-thunder], Don Heyse, Chris Moseley, Steve Chisnall [StevieC], Edwin Pilobello [gypsy_fly], John Boozer [jediknight219], Ben Lyttle [legotrek], Reinhard "Ben" Beneke [Ben_aus_BS], Jim DeVona [anoved], Jason Mantor [Xanthra47], James Mastros [theorbtwo], Jeffery MacEachern [legonerd], Axel Poque, John Jensen, Douglas Taylor, Jr. [djcool905], Stig-Erik Blomqvist [stigge72], Jeroen Ottens, Bernd Munding, Steve Demlow [demlow], Bert J. Giesen, Reuben Pearse [ReubenPearse], Robert Paciorek [bercik], Richard Baxter [rbaxter], Dan Boger [dan], Duane Hess, Earnest J. Banbury [Banbury], Martin G Cormier, Jack Hawk [jhawk], Arthur Sigg [Pendulum], James Shields [lostcarpark], Richard Finegold, Martyn Boogaarts, Antony Caparica [antonyc], Christopher Bulliner [CMB27], Christopher Pedersen [pedersen], Troy [peloquin], Travis Cobbs [tcobbs].*

The following authors have contributed full models to the LDraw Open Model Repository. This repository contains 1,727 high quality models assembled from the bricks in the LDraw parts library.

*Robert Paciorek [bercik], Philippe Hurbain [Philo], Marc Giraudet [Mad_Marc], Massimo Maso [Sirio], Damien Roux [Darats], Merlijn Wissink [Legolijntje], Willy Tschager [Holly-Wood], Orion Pobursky [OrionP], Max Martin Richter [MMR1998], Stefan Frenz [smf], Tomas Kralicek [RabbiT_CZ], Victor Di Rienzo [tatubias], Ken Drew [Ken], Johann Eisner [technicbasics], Bert Van Raemdonck [BEAVeR], Evert-Jan Boer [ejboer], Charles Farmer [farmer], Marc Belanger [MonsieurPoulet], Roland Dahl [RolandD], Oh-Seong KWON, Jude Parill [theJudeAbides], Ignacio Fernandez Galvan [jellby], Faramond Florent [Makou], Daniel Goerner [TK-949], Zoltán Kéri [Zoltank82], Takeshi Takahashi [RainbowDolphin], Stan Isachenko [angmarec], Jaco van der Molen [Jaco], Rijk van Voorst [Rijkjavik], Christian Maglekær, Christian Neumann [Wesley], Steffen [Steffen], N. W. Perry [Plastikean], Michael Heidemann [mikeheide], Michal Oravec [Bloodybeast], Magnus Forsberg [MagFors], Lasse Deleuran [Lasse Deleuran], [juraj3579], Allard van Efferen [aefferen], Rafael Skibicki [Rola], Adrien Pennamen [AdrienPennamen], Greg Teft [gregteft], Christophe Mitillo [Christophe_Mitillo], René Frijhoff, Sean*

*Burke [Leftmost], Antony Lodge, Mirjan Lipovcan [LegoZG], Florian Schüller [schuellerf], Tim Singer [tsinger], [EdmanZA], Guido Mauro, Casey Puyleart, Peter Bartfai, Oliver Damman [Nautilus], Steffen Altenburg [SteffenA], Ulrich Röder [UR], Alexey [folkoluck], Kevin Hendirckx [Gebruiker].*

LTRON uses LDraw metadata bundled with the free LDCAD software to find connection points between bricks. *Roland Melkert* is the primary author of LDCAD, but others have contributed to this metadata.

*Roland Melkert [roland], Philippe Hurbain [Philo], Milan Vancura [MilanV], Alex Taylor [anathema], Jason McReynolds [Jason McReynolds].*

I also wish to thank *Toby Nelson*, the author of the open-source *ImportLDraw* plugin for Blender. This was used to convert the LDraw parts library to wavefront OBJ files for use in the LTRON rendering system.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# 1

# INTRODUCTION

In humans, vision is by far the most information-rich sensory modality with estimates that around ninety percent of the information flowing into the brain is visual[73], and that around half of the cerebral cortex is involved in processing visual information of some kind [46]. Despite the fact that so much information comes in through our eyes, not all of this information is relevant. The world is a visually noisy place, and our brains constantly throw away information in order to help us achieve our objectives without drowning us in unnecessary details.

However, discarding irrelevant details is not the only way that our brain shapes our understanding of the visual world. When some important information is missing, we take actions in order to gather the information we need. These information gathering actions can be as simple as glancing in the direction of some object we wish to pick up, or as complex as building a particle accelerator to inspect the behavior of the fundamental building blocks of matter. These actions are necessary because vision only provides a partial observation of the world. We have a limited field of view and cannot see through solid objects, so we must move ourselves and the world around us to see the full picture.

These two techniques, discarding informational clutter and taking information gathering actions, operate together to meticulously curate the visual information that makes its way to our higher reasoning systems. This curation is an interactive process. In order to reason about what information is currently available and make plans for what to seek out in the future, we must have some concept of time and memory for keeping track of what information we have collected so far. Sequential decision making provides us with the mathetmati-

cal and machine learning tools necessary to build and reason about these systems.

In this dissertation, we will explore the curation of visual information across several sequential decision making problems.

In Part I we show that in goal-conditioned environments, providing the goal information early in the visual processing stream can allow an agent to discard irrelevant information more quickly, which allows it to make better decisions with smaller networks.

In Part II, we discuss a new problem that we have built called Break and Make in which an agent must disassemble a previously unseen LEGO assembly in order to gather enough information about it to be able to build it again from scratch. This problem is challenging because it requires an agent to learn how to inspect and gather information about a complex object, and also perform precise building actions for accurate reconstruction. We have developed a model that uses a stack-based memory structure to build a set of instructions for itself to use during reassembly. We show that this model, combined with various training improvements, outperforms baseline approaches that use RNNs and Transformers to keep track of past observations, and is able to reconstruct assemblies with more than forty bricks.

In Part III we explore the challenges related to providing supervision for agents operating from partial observations. We show that experts operating in the same environment, but under a different observation space can provide counter-productive demonstrations that cause the learning agent to drastically underperform relative to the best-in-class policy under the agent's observation space. To address this we have developed a new learning algorithm that combines reinforcement learning with imitation learning to seek out missing information that an expert may be using to make decisions.

# 2

# NOTATION AND FORMAL DEFINITIONS

In this chapter we will write down the formal definitions we will use throughout this document.

## 2.1 MARKOV DECISION PROCESS

A Markov Decision Process (MDP) is a basic description of an interactive problem. Formally an MDP is a discrete-time process consisting of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a transition function $T$ and a reward function $R$. At each time $t$, the agent receives a state $s_t \in \mathcal{S}$ and must take an action $a_t \in \mathcal{A}$. The system then transitions to a new state $s_{t+1} = T(s_t, a_t)$ and the agent receives a scalar reward $r_t = R(s_t, a_t, s_{t+1})$. Both the transition function and reward function may be stochastic. We will primarily deal with finite-horizon episodic MPDs. In these settings the agent is initialized with $s_0$ drawn from a fixed starting distribution $S^{init}$ and continues until a terminal state $s^{term}$ is reached.

A crucial assumption of an MDP is that the transition and reward functions are Markovian. This means that they only depend on the current state $s_t$ and action $a_t$ (and possibly the following state $s_{t+1}$ in the case of the reward function) but not on any of the prior states or actions. The distribution of future states and rewards is independent of past states, given the present state $s_t$.

## 2.2 PARTIALLY OBSERVABLE MARKOV DECISION PROCESS

A Partially Observable Markov Decision Process consists of a Markov Decision Process along with an additional set of observations $\Omega$ and an observation function $\mathcal{O}$. In this setting an agent does not have access to the state $s_t$, but instead receives a possibly stochastic and lossy

observation $o_t = O(s_t)$ with $o_t \in \Omega$. The fact that these observations can be lossy means that we may be required to consider not just the most recent obsevation, but also several past observations in order to act optimally in these settings. See [56] for a thorough treatment of POMDPs.

## 2.3 TRAJECTORY

A trajectory is a history of states actions and rewards that have been collected for some number of time steps after interacting with an environment.

$$\tau_i = \{(s_1, a_1, r_1) \dots (s_{i-1}, a_{i-1}, r_{i-1}), s_i\}$$

In a POMDP, a trajectory will contain observations instead of states.

$$\tau_i = \{(o_1, a_1, r_1) \dots (o_{i-1}, a_{i-1}, r_{i-1}), o_i\}$$

## 2.4 POLICY

A policy is a possibly stochastic function $\pi$ that computes actions after making observations in an environment. A policy could be a learning agent $\pi_{\mathcal{L}}$ or an expert $\pi_{\mathcal{E}}$ or any other function that produces actions. Due to the Markov property of MDPs, a policy in an MDP can make a fully informed decision based only on the current state $a_t = \pi(s_t)$. Because this property does not hold for observations in a POMDP, it is not enough for a policy with partial observations to operate on a single observation, but must instead consider either a distribution over states (belief) if the state space is finite and known, or as is most often the case, a trajectory of observations up to the present time $a_t = \pi(\tau_t)$.

## 2.5 RETURN AND VALUE

The return of a trajectory $G(\tau) = \sum_{i=1}^{|\tau|} r_i$ is the sum of all rewards encountered during that trajectory. The value function of a policy $V_\pi(s) = \mathbb{E}_{\tau\sim\pi|s_1=s} G(\tau)$ is the expected value of the return of a trajectory that starts in state $s$ and continues by sampling actions according to the policy $\pi$. We write $V_\pi^{init}$ to be $\mathbb{E}_{s\sim S^{init}} V_\pi(s)$, the expected value when acting according to $\pi$ after starting from states sampled from the initial state distribution. The goal of a training algorithm is to produce a policy $\pi_\mathcal{L}$ that maximizes $V_{\pi_\mathcal{L}}^{init}$.

# Part I

# IMAGES CONTAIN TOO MUCH INFORMATION

In Part I we show that when an image contains too much information, finding out what part of it is irrelevant and quickly throwing it away allows us to be more efficient and use smaller models to achieve similar task performance. We show this in a goal-directed environment where an agent in a simulated kitchen receives an image of the scene and a list of items it should pick up. We show that considering the list of items early in the visual processing stream allows the model to discard unnecessary information, which in turn reduces the number of network parameters that are necessary to complete the task. This contains material originally published as "Early Fusion for Goal Directed Robotic Vision" at IROS '19[130].

# 3

# EARLY FUSION FOR EFFICIENT DECISION MAKING

## 3.1 INTRODUCTION

Robotics has benefited greatly from advances in computer vision, but sometimes the objectives of these fields have been misaligned. While the goal of a computer vision researcher is often “tell me what you see," the roboticist’s is “do what I say." In goal directed tasks, most of the scene is a distraction, extra information that can and should be discarded as soon as possible. When grabbing an apple, an agent only needs to care about the table or chairs if they interfere with accomplishing the goal. Additionally, when a robot learns through grounded interactions, architectures must be sample efficient in order to learn visual representations quickly for new environments. In this work we show how inverting the traditional perception pipeline: Vision $\rightarrow$ Scene Representation + Goal $\rightarrow$ Action to incorporate goal information early into the visual stream allows agents to jointly reason and perceive: Vision + Goal $\rightarrow$ Action, yielding faster and more robust learning.

We focus on retrieving objects in a 3D environment as an example domain for testing our vision architectures. This task includes vocabulary learning, navigation, and scene understanding. Task completion requires computing action trajectories and resolving 3D occlusions from a 2D image which satisfy the user’s requests. Fast and efficient planners work well in the presence of ground-truth knowledge of the world [110]. However, in practice, this ground-truth knowledge is difficult to obtain, and we must often settle for noisy estimates. Additionally, when many objects need to be collected or moved, the planning problem search space grows rapidly.

Unlike computationally expensive modern vision algorithms, we are interested in training perception algorithms with a more natural source of supervision, example demonstrations and imitation learning, in lieu of expensive large scale collections of labeled images. This is particularly important for developing agents that learn new object classes on the fly (e.g. when being integrated into a new environment). Our work is most closely related to recent advances in instruction following and visual attention [77, 79], but we do not provide explicit supervision for object detections or classifications. Finally, we will make the assumption that goals are specified by a simple list of object IDs, so as to avoid the ambiguity introduced by natural language commands.

## 3.2 RELATED WORK

Processing strategies for goal-directed visual search have been an important area of study in psychology, neuroscience and computer vision for many years [124, 140]. Early work in this area drew on the observation that human and primate vision seems to be at least partially driven by goal-directed top-down signals [29, 30].

More recently there has been a proliferation of works examining goal directed visual learning in simulated worlds [2, 17, 19, 35, 43, 152] which each aim to bring different amounts of language, vision and interaction to the task of navigating a 3D environment. This has also been attempted in real 3D environments [39]. Importantly, in contrast to our work, these approaches often pretrain as much of their networks as possible. [43] do not pretrain for their RL based language learning. Their work does not address learning with occulusion or larger vocabularies. In parallel, the robotics literature has investigated grounding instructions directly to robotic control [7, 8, 74, 78, 121, 152], a domain where data is expensive to collect.

Training end-to-end visual and control networks [66], has proven difficult due to long roll outs and large action spaces. Within reinforcement learning, several approaches for mapping natural language instructions to actions rely on reward shapping [77, 79] and imitation

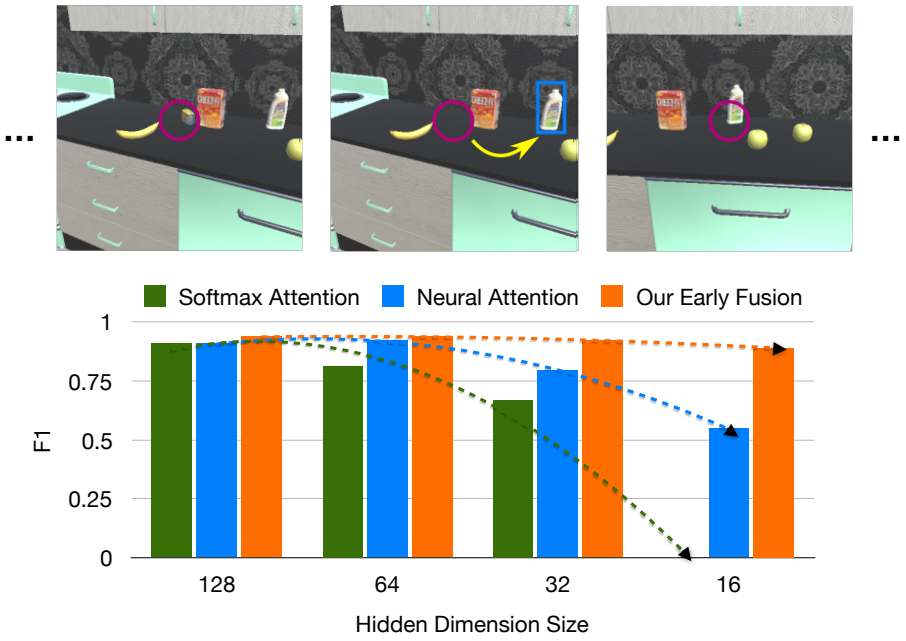


Figure 1: We demonstrate a novel neural architecture for goal directed object detection which we demonstrate in a simulated table clearing task shown in the top row. We demonstrate that unlike conventional approaches, this structure is stable under extreme parameter budgets as seen in the bottom row.

learning [7, 8]. Imitation learning has also proven effective for fine grained activities like grasping [24], leading to state-of-the-art results on a broad set of tasks [25]. The difficulty encountered in these scenarios emphasizes the need to explore new methods for efficient learning of multimodal representations. [108] explored attention model architectures, but do not include early fusion techniques. Early fusion of goal information has shown promise with small observation spaces [120], but our work begins to explore this method for high-dimensional visual domains. In this chapter, we hope to provide the community with a missing analysis and insights into this approach and its power in interactive settings.

## 3.3 TASK DEFINITION

In order to test the performance of Early Fusion we built a simulated robotic task in which the objective is to collect objects in a 3D scene as efficiently as possible. The agent is presented with a cluttered scene


Figure 2: The 16 object types used in the kitchen environment.

and a list of requested objects. Often there are multiple instances of the same object, and there can be unrequested objects blocking the agent's ability to reach a target. This forces the agent to reason about which object is closest and remove obstructions as necessary. The list of requested objects that remain in the scene is presented to the agent at every time step, to avoid conflating scene understanding performance with issues of memory. The goal (Fig. 1 and 3) is to train an agent to optimally collect a list of objects from a cluttered counter.

### 3.3.1 *Simulation Environment: CHALET*

Our environment consists of a tabletop setting with randomly placed objects, within a kitchen from the CHALET [146] house environment. Every episode consists of a randomly sampled environment which determines the set of objects (number, position, orientation and type) in addition to which subset will be requested. When there is more than one instance of a particular object, collecting any instance will satisfy the collection criteria, but one may be closer and require fewer steps to reach. Fig. 2 shows the sixteen object types that we use for this task (six from CHALET and ten from the YCB dataset).

The objects are chosen randomly and placed at a random location (x,y) on the table with a random upright orientation (θ). Positions and orientations are sampled until a non-colliding configuration is found. A random subset of the instances on the table are used for the list of requested objects. This process allows the same object type to

be requested multiple times if multiple of those objects exist in the scene. Additionally, random sampling means an object may serve as a target in one episode and a distractor in the next. The agent receives 128x128 pixel images of the world and has a 60° horizontal field of view, requiring some exploration if a requested object is not in view.

Our agent consists of a first-person camera that can tilt up and down and pan left and right with additional `collect`, `remove` and `idle` actions. Each of the pan and tilt actions deterministically rotate the camera 2° in the specified direction. The `collect` action removes the nearest object that is within 3° of the center axis of the camera and registers the object as having been collected for the purposes of calculating the agent's score. This region is visualized in Fig. 3 as a magenta circle in the center of the frame. The `remove` action does the same thing as `collect`, but does not register the item as having been collected. This is used to remove superfluous items occluding the requested target. Finally, the `idle` action performs no action and should only be used once all requested items have been collected. All actions require one time step, therefore objects which are physically closer to the center of the camera may take more time steps to reach if they are occluded. For example, in Fig. 3 the peach (orange box) requires fewer steps to collect than the Jello box (blue box) because the banana and Rubik's cube must be removed first. The precision required to successfully collect an object makes this a difficult task to master from visual data alone.

Figure 3: Collecting the Jello (blue box) requires more steps than the peach (orange box) due to occluding objects. An object may be collected if it is within the magenta circle.

## 3.4 MODELS

In our task, models must learn to ground visual representations of the world to the description of what to collect. How to best combine this information is a crucial modelling decision. Most multimodal approaches compute a visual feature map representing the contents of the entire image before selectively filtering based on the goal. This is commonly achieved using soft attention mechanisms developed in the language [4] and vision [3, 80, 108] communities.

Attention re-weights the image representation and leads to more informative gradients, helping models learn quickly and efficiently. Despite its successes, attention has important limitations. Most notably, because task specific knowledge is only incorporated late in the visual processing pipeline, the model must first build dense image representations that encode anything the attention might want to extract for all possible future goals. In complex scenes and tasks, this places a heavy burden on the initial stages of the vision system. In contrast, we present a technique that injects goal information early into the visual pipeline in order to build a task specific representation of the image from the bottom up. Our approach avoids the traditional bottleneck imposed on perception systems, and allows the model to discard irrelevant information immediately. Our system may still need to reason about multiple objects in the case of clutter and occlusion (e.g. target vs. distractor), but its perception can ignore all objects and details that are not relevant for the current task.

Below, we briefly describe the three models (Figure 4) we compare: Traditional approaches with delayed goal information (Late Fusion & Attention Map) versus our goal conditioned Early Fusion architecture.

### 3.4.1 *Late Fusion*

Late Fusion constructs a single holistic representation of the entire image via a stack of convolution and pooling layers before concatenating an embedding of the requested objects in order to predict an

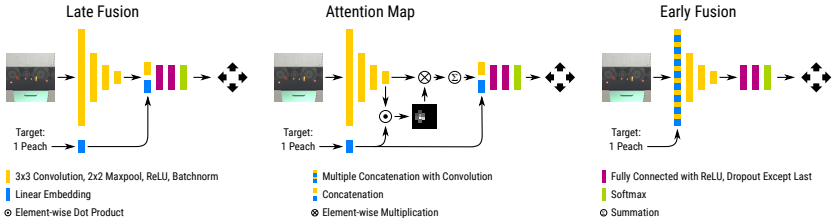


Figure 4: We compare a simple concatenation of visual and goal representations (Late Fusion), against two variations of the attention mechanism above, and Early Fusion to isolate the effects of when multimodal representations are formed.

action. An object embedding is computed using a simple linear layer designed to turn a one-hot encoding of the object into a dense representation. The complete request for multiple objects is computed as a sum of these individual object embeddings. This design forces the vision module to store semantic and spatial information about every object in the scene so the final fully connected layers can ground target objects and reason about actions.

### 3.4.2 *Attention Map*

We test traditional attention mechanisms over image regions. As with Late Fusion, the first step of this model is to pass the image through a stack of convolution layers. Rather than concatenate the request embedding directly onto the resulting representation, these models first compute an attention map over the spatial dimensions of the convolution output. This is accomplished by comparing the embedded target vector with each region of the convolutional feature map via a simple dot product. This provides a weight to each region which can then be used to form the final image representation $I = \sum_i \frac{\alpha_i}{Z} h_i$. Next, $I$ is concatenated to the request to make an action decision. We test two attention models: Softmax Attention Map which is defined above and Attention Map which is unnormalized. Using a softmax leads to a peakier distribution which focuses the model on fewer regions of the image (see Fig. 8).

In contrast to the Late Fusion model, the attention mechanism provides a filter on extraneous aspects of the image to simplify the con-

trol processing. In these models the grounding from image features to goal objects is done with a direct comparison operator (the dot product). These models are widely used for Visual Question Answering (VQA) problems on static images. We also explored more complex models [108] for computing attention maps, but found this traditional version worked the best in our setting and provided a strong baseline for comparison. In our results, we follow [72] and append spatial grids to the first layer of this network to encode spatial knowledge. This extra information proved necessary for the attention models to compete with Early Fusion. We found that these spatial grids did not aid nor hinder Late Fusion.

### 3.4.3 *Early Fusion*

Finally, our Early Fusion approach concatenates the request embedding to every region of a convolutional filter map. This feature is then processed normally by a set of convolution kernels that have been augmented to account for the extra channels. Fig. 5 shows this process. All further processing in the network is computed normally. The model's subsequent convolution and fully connected layers may filter the visual information according to the goal description that is now combined with the visual input. This results in an image representation which contains only the necessary information for deciding the next action, effectively gaining the benefits of a bottleneck while dispersing the logic throughout the network. Critically, this means that the network does not have to build a semantic representation of the entire image (See Section A.1 in the Appendix for details).

Two important results of this architecture are: 1. Because the goal information is incorporated early, the network can learn to ground the image features to the goal objects at any point in the model without additional machinery (like attention); and 2. The model can compute and retain the spatial information needed for its next action without requiring the addition of a spatial grid. These benefits allow us to obviate the complexity of other approaches, minimize parameters, and outperform other approaches on our task. One counter-argument

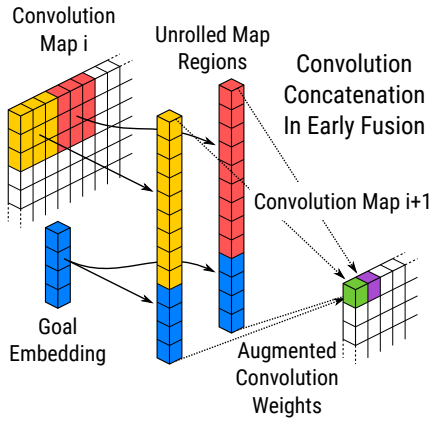


Figure 5: A diagram Early Fusion of goal information with visual data. The goal is concatenated with each block of visual data.

against this approach is that it is task-specific and therefore requires a large dataset of images paired with goal information. This runs against current trends in deep learning that leverage large unlabeled datasets to pretrain foundation models which can be used on downstream tasks. Training a goal-directed foundation model is a remaining open problem, given the scarcity of goal-directed data available in the wild. There is, however some overlap with very recent large vision and language models[1] that make predictions about images and text using more modern Transformer architectures.

### 3.4.4 *Imitation Learning*

All models are trained with imitation learning using an oracle with direct access to the simulator's state. Similar to DAgger [95] and Scheduled Sampling [6] we use an alternating two-step training process. In the first step, we roll out trajectories using the current model while collecting supervision from the expert. In the second step we use batches of recent trajectories to train the model for a small number of epochs. We then repeat this process and collect more data with the improved policy[7]. We found that for our item retrieval problem this was faster to train than a more faithful implementation of DAgger which would train a new policy on all previous data at each step, and offered significant improvements over behavior cloning (training

on trajectories demonstrated by the expert policy). We found that collecting 50 trajectories in each roll-out step, and then training on the most recent 150 trajectories for three epochs in each training step produced the best results.

Rather than teach our agents to find the shortest path to multiple objects, which is intractable in general, we design our expert policy to behave greedily and move to collect the requested object that would take the fewest steps to reach (including the time necessary to remove occluding objects).

### 3.4.5 *Implementation Details*

Since our goal is to construct a lightweight network that is fast to train and evaluate, we use a simple image processing stack of four convolution layers. While this is small relative to models trained for state-of-the-art performance on real images, it is consistent with other approaches in simple simulated settings [101]. All convolutions have 3×3 kernels with a padding of one, followed by 2×2 max-pooling, a ReLU nonlinearity [33] and batch normalization [48]. This means each layer produces a feature map with half the spatial dimensions of the input. The convolution layers are followed by two fully connected layers, the first of which uses a ReLU nonlinearity and Dropout [111] and the second of which uses a softmax to produce output controls. The number of convolution channels and hidden dimensions in the fully connected layers vary by experiment (see Section 3.5.2). All of our models are optimized with Adam [59] with a learning rate of 1e-4, and trained with dropout [111]. The training loss was computed with cross-entropy over the action space.

IMAGES Our images are RGB and 128x128 pixels, but as is common practice in visual episodic settings [81] we found our models performed best when we concatenated the most recent three frames to create a 9x128x128 input. We used black frames as padding in the first two frames when prior frames were not available.

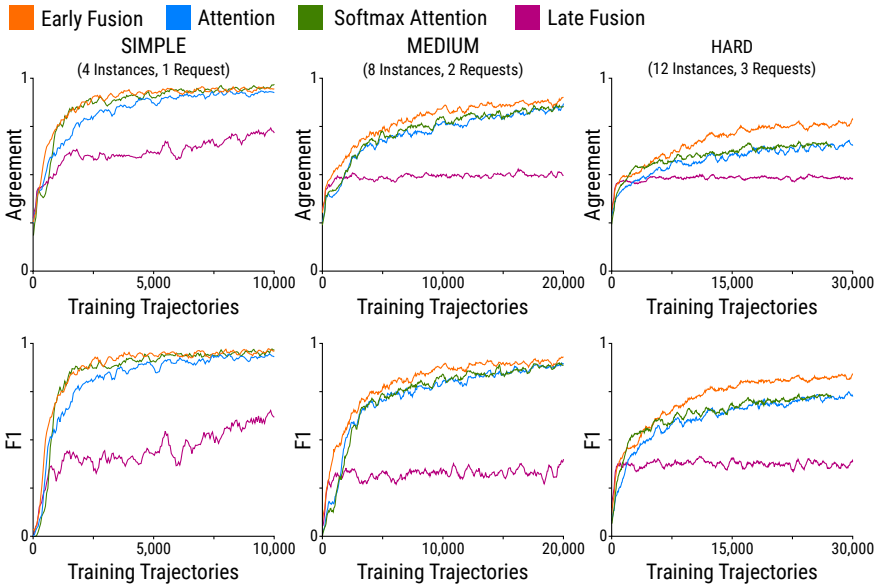


Figure 6: The performance of various models on SIMPLE, MEDIUM, and HARD kitchen environments.

REQUESTS Models are provided the remaining items to collect as a list of one-hot vectors. Each of these items is passed through a learned embedding (linear) layer to produce an encoding. These are then summed to produce a single dense vector (Target). Because the sequence order is not important to our task, we found no benefit from RNN based encodings, though the use of an embedding layer, rather than a count vector, proved essential to model performance.

## 3.5 EXPERIMENTS

We tested all four models on a series of increasingly cluttered and difficult problems. We also tested these models with varying network capacity by reducing the number of convolution channels and features in the fully connected layers. In all of these experiments, our Early Fusion model performs as well or better than the others, while typically training faster and with fewer parameters.

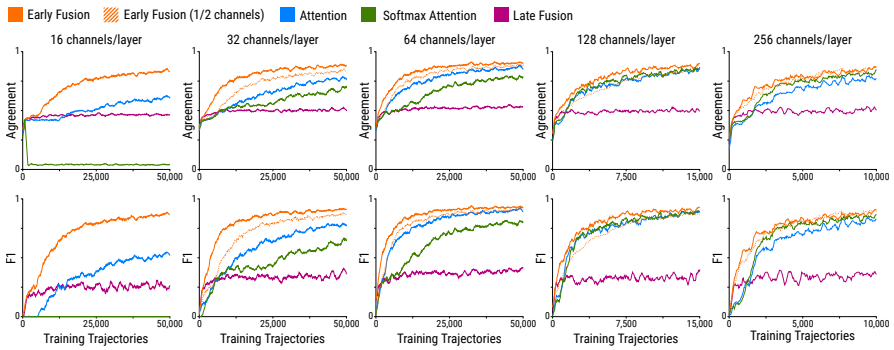


Figure 7: Ablation analysis showing the effect of the number of convolution channels and fully-connected hidden units on network performance. Note that the scale of the x-axes in these plots varies due to longer training times for smaller networks to converge. Dashed Early Fusion lines plot the performance of the model with half the reported number of filters to include a comparison ensure where model has fewer parameters than the attention based approaches.

### 3.5.1 *Varying Problem Difficulty*

To test models on problems of increasing difficulty, we built three variations of the basic task by varying clutter and the number of requested items. In the simplest task (SIMPLE), each episode starts with four instances randomly placed on the table and one object type is requested. Next, for MEDIUM eight instances are placed and two are requested. Finally, for HARD twelve instances are placed and three are requested. Note that as the clutter increases, the agent is presented with not only a more complicated visual environment, but must also work in a more complex action domain where it is increasingly important to use the `remove` action to deal with occluding objects. The agent's goal is to collect only the requested items in the allotted time. To evaluate peak performance for these experiments we fixed the number of convolutions and hidden layer dimensions in the fully connected layers to 128.

Each episode runs for forty-eight steps, during which it is possible for the agent to both successfully collect requested objects and erroneously collect items that were not requested. We therefore measure task completion using an F1 score. Precision is the percentage of collected objects that were actually requested, and recall is the percent-

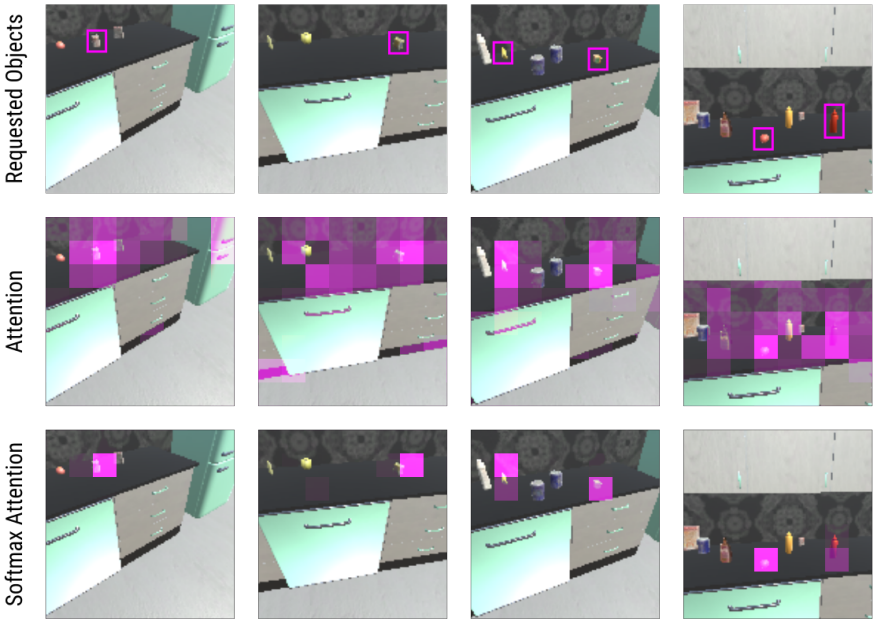


Figure 8: Attention visualizations for Attention Map and Softmax Attention Map models in the kitchen environment. Targets are indicated here with magenta boxes in the top row.

age of requested objects that were collected. The F1 score is computed at the end of each episode. In addition, we report overall agreement between the model and the expert's actions over the entire episode. Figure 6 plots the results of all four models on each of these problems as a function of training time.

SIMPLE Except for the Late Fusion model, which performs poorly in all scenarios, all models are able to master the easiest task. The Early Fusion and Softmax Attention Map models learn quickly, but Attention Map eventually catches up to them. The failure of the Late Fusion baseline on this task shows that even the simplest version of this problem is non-trivial.

MEDIUM The intermediate problem formulation is clearly more difficult, as no models are able to perform as well on it as the easiest problem. The Early Fusion model gains a small but significant improvement in performance while Softmax Attention Map and Attention Map are slightly worse, but comparable to each other.

HARD This case contains more cluttered images and more complex goal descriptions. The Early Fusion model is clearly superior; it learns significantly faster than the other models and results in higher peak performance.

It is also worth comparing the Attention Map and Softmax Attention Map models. While these models perform similarly on these tasks, the Softmax Attention Map model learns faster than the Attention Map model on the easiest task, but slightly slower on the more difficult ones. We posit that the softmax focuses the attention heavily on only a few regions, which is useful for sparse uncluttered environments, but less appropriate when the network must reason about multiple objects in different regions.

Fig. 8 provides a comparison of attention maps. Unsurprisingly, the Softmax Attention Map model produces a sharper distribution around the requested objects, but both methods correctly highlight the objects of interest. In this work, we have limited our definition of clutter to 12 items per scene, in part for ease of visualization and compute time.

### 3.5.2 *Varying Network Capacity*

Having demonstrated that Early Fusion is at least as powerful as attention based approaches while being simpler (no grid information or attention logic), we explore how these approaches perform on varying parameter budgets. Real-time and embedded systems require efficiency both when training and during inference. Since Early Fusion removes irrelevant information early in the processing pipeline, we expect it to require less network capacity than the other methods. To test this claim, we re-run our MEDIUM difficulty setting (because attention models performed well) and compare performance when models have access to 256, 128, 64, 32, or only 16 channel convolutions and fully connected layers, reducing our model sizes by several orders of magnitude.

In Fig. 7, training time increases for small networks, but Early Fusion is able to quickly achieve around the same final performance

regardless of the extremely small network capacity. This allows for dramatically more efficient inference and parameter/memory usage. In contrast, other models degrade substantially as the network size decreases. Note that after 50,000 trajectories it appears that attention based models are still slowly improving, but there is a stark contrast in learning rates. In particular, for the smallest models (16) we see that Attention Map, even after training for twice as long as Early Fusion, still has half the performance.

Because attention mechanisms collapse their final representations, they have a smaller fully connected layer and therefore fewer parameters for the same number of channels. To account for this, we have also included a dashed orange line in Fig. 7 labeled early fusion (1/2 channels) which shows the performance of Early Fusion with half the channels as the other models and fewer parameters. Again smaller Early Fusion networks outperform and learn faster than the other approaches.

### 3.5.3 *Generalization*

To measure generalization we conduct experiments in which the agent is trained on a subset of the possible request combinations and then tested on unseen requests. Here the agent is trained with 128 different two-item combinations, and then tested on a held out 128 two-item combinations (Rows 1 and 2 below). In this setting, the agent generalizes to unseen item pairs, indicating that the agent is not merely memorizing these combinations, but learning to recognize the structure of requests composed of individual objects.

| | Agreement | F1 |
|---|---|---|
| Two-Item Train | 0.8918 | 0.9215 |
| Two-Item Test | 0.8695 | 0.8938 |
| Three-Item Test | 0.8140 | 0.8243 |

In the second experiment, the same agent was tested on a random collection of three-item combinations to determine if the agent can

generalize to higher counts than during training (Row 3). The agent is surprisingly robust to this variation.

## 3.6 CONCLUSION

Goal directed computer vision is an important area for robotics research and efficient training of high performing models with minimal footprints are essential for in situ learning. We take one step in this direction by showing how Early Fusion is ideal for the simulated robotic object retrieval task, and preferable to traditional attention based approaches. Future work should investigate how our approach and analysis can be generalized to on-device learning.

It is also worth noting that in the time since this work was originally published, Transformers[125] have become a dominant architecture across a variety of applications. Transformers are able to consume a heterogeneous mix of data and are well suited to the kinds of goal-driven visual problems discussed here. Transformers neatly avoid the question of early vs. late fusion by allowing the data processing stream to pay attention to all information at each layer. In this setting, the goal information could be passed to a visual transformer as an additional token which allows the tokens representing visual information to pay attention to this goal information whenever is appropriate. This allows the network to automatically learn when and how to fuse information while processing sensory information.

# Part II

# IMAGES DO NOT CONTAIN ENOUGH INFORMATION

In Part II we move from the cluttered kitchen environments, where each image contained too much information, to a new LEGO assembly problem where a single image is not enough to describe an entire structure. This is fundamentally a problem of partial observability. The agent, when looking at a LEGO assembly from the outside has no way of knowing its internal structure. The agent must therefore take steps to disassemble and inspect the assembly's internal components in order to gain a better understanding of it. This allows us to study in detail an important strategy in decision making under partial observations: compensating for the lack of information in a single observation by combining the information from a number of subsequent observations. In other words using memory. In the following chapters we will discuss the Break and Make problem in more detail, then see two different approaches to memory on this problem: sequence modelling using LSTMs and Transformers to keep information about an entire sequence of actions and observations, and a more tailored approach called InstructioNet that uses a stack-based memory structure to more carefully present the model with a series of goals to achieve. This contains material originally published as "Break and Make: Interactive Structural Understanding Using LEGO Bricks" at ECCV '22[132] and additional unpublished material that is currently in submission.

# 4

# BREAK AND MAKE IN LTRON

## 4.1 INTRODUCTION

The physical world is made out of objects and parts. Buildings are made out of roofs, rooms and walls, chairs are made out of seats, backs and legs, and cars have doors, wheels and windshields. The ability to reason about these parts and the structural relationships between them are a key component of our ability to build tools and shelters, solve complex organizational problems and manipulate the world around us. Building part-based reasoning capability into intelligent agents has been a long-standing goal of the computer vision, robotics and broader AI communities.

Due to the complexities of physical structures, part-based reasoning often allows for only partial observation of a particular structure or mechanism. While a single image may tell you a lot about a car, it does not tell you about it's internal construction and wiring. In order to gain that information, you must physically inspect it, possibly even removing panels or components in order to get a better view.

We have recently proposed Break and Make, a challenging new problem designed to investigate this kind of interactive structural reasoning using LEGO bricks. This problem is designed to simulate the process of reverse engineering: taking apart a complex object to learn more about its structure, and then using this newfound knowledge to put it back together again. This task is naturally divided into two phases. In the first Break phase, a learning agent is presented with a previously unseen LEGO model and has the opportunity to disassemble and inspect it in order to observe its internal structural and hidden components. After this, in the second Make phase, the agent is presented with an empty scene and must use the information gathered during the Break phase to rebuild the model from scratch. Both

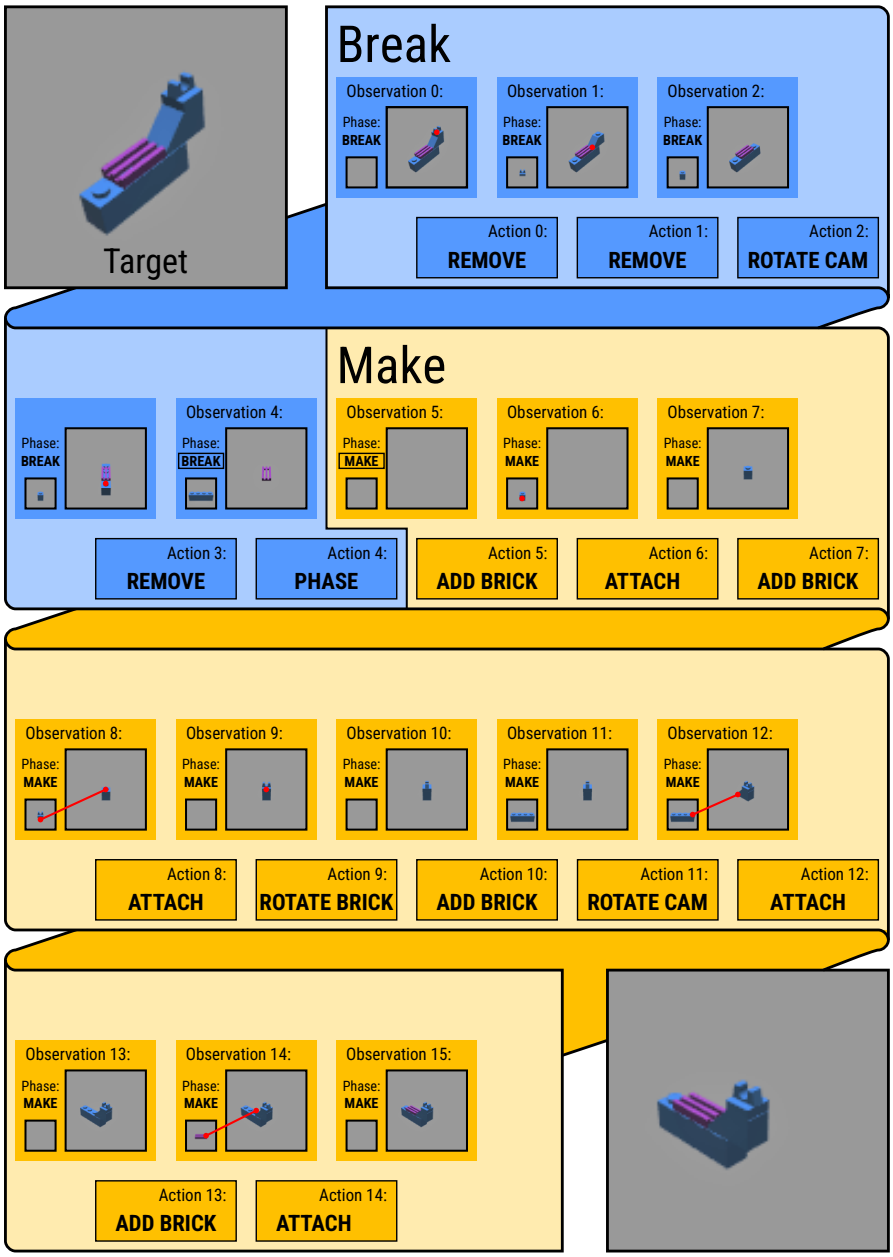


Figure 9: A training example of the Break and Make task on a four-brick model in LTRON. During **Break** phase, the agent must learn to disassemble removable parts based on RGB images to understand the underlying structure. During the second **Make** phase, the agent must learn to pick bricks and reassemble the scene based on all past observations.

phases must be completed using visual action primitives designed to simulate the LEGO construction process. These actions require an agent to reason not only about individual bricks, but also the connection points between them. Figure 9 shows an example of an expert completing this task on a four brick model.

In order to facilitate research on this challenging problem, we provide a dataset of 1727 ethically-sourced fan-made LEGO models with generous public licensing. These models range in size from 5 to 7302 individual bricks and use a library of 1790 distinct brick shapes. We also include a set of augmentations and a random model generator in order to provide more examples for large-scale training. Finally, we provide a 3D simulator and interactive learning environment with an OpenAI gym interface designed to train agents on this problem. Our simulator is compatible with a file format commonly used in the LEGO fan community, and is therefore capable of displaying and manipulating a wide range of models found online.

The Break and Make problem presents a difficult challenge for a number of reasons. First, the interchangeable nature of LEGO bricks and the large number of distinct brick shapes results in a very large state and action space. Second, this problem requires precise memory in order to bridge the long-term temporal distance between observations in the Break phase and reconstruction actions that must be taken in the Make phase. Third, this problem also requires precise spatial reasoning in order to carefully place bricks in the correct location using a visual observation and action space. Finally, it is difficult to provide direct supervision for this problem, even with accurate information from the simulator. This stems from the fact that it is not possible to directly compute which observations are necessary to capture the structural details of a model. We are however able to provide noisy supervision using a custom planner that reasons over visual observations. This planner generates a series of actions and observations that will feasibly disassemble and reassemble the model, but it comes with no guarantee that an agent equipped with only the visual observations from the sequence would have enough information to make the necessary decisions.

## 4.2 RELATED WORK

### 4.2.1 *Understanding Compositional Structures*

Interactive scene understanding and reasoning about compositional structure has origins in the early days of AI. An early example is Winograd's SHRDLU system [139] that used language instructions to interactively stack virtual blocks and answer questions posed by a human operator.

More recently researchers have introduced a number of interactive environments such as RoboThor[21], Chalet[147], iGibson[106], Habitat[103, 119] and MultiON[134] designed to simulate indoor environments for embodied learning agents. Many tasks have been proposed for these environments, such as goal-directed navigation [151], interactive question answering [20, 36] and instruction following [107]. While many of these tasks and environments offer some degree of object manipulation, most of these interactions involve only a small number of object classes, and do not require the agent to reason about complex compositional structures. In contrast LTRON requires an agent to reason in detail about these structures and how to build them from a library of 1790 unique parts.

In the non-interactive domain, researchers have released a number of simulated tasks and datasets [10, 69, 83] designed to provide access to a diverse set of objects with increasing detail, part structure and complexity. Others such as CLEVR [52], CLEVERER [148], and CATER [32] are designed around answering questions about object relationships in images and videos. In these settings, it is easy to procedurally generate a large dataset using randomization, but it has been difficult to generate datasets with large object and relationship vocabularies. Researchers have also taken great effort to annotate natural images and videos with detailed attributes [27], parts [129] and relationships [61].

Scene understanding via active or interactive perception is a classic way for robots and embodied agents to explore and model their environment. Researchers have investigated varying levels of detail

and semantics in this space[75, 93, 100, 114, 115, 127]. Previously it has been difficult to explore objects with fine-grained part structure in these settings due to the difficulty in collecting and annotating this data. LTRON provides complex models in an interactive environment, allowing agents to collect large amounts of data for researching complex cluttered environments with compositional structures. Another recent line of work explores learning physical properties of the world either from observations of rigid body interactions [141–143] or unsupervised physical interaction with a robot [28]. While we do not provide explicit rigid body dynamics in LTRON, we allow agents to explore extremely detailed physical structures with complex part-interactions at a scope that has not been practical in the past.

### 4.2.2 *Building 3D Structures*

In robotics, there has been a long-standing interest in enabling robots to build or assemble structured objects. Several authors have explored assembling IKEA furniture [63, 70, 112, 150]. Others [47, 102] have used Deep Reinforcement Learning and Learning from Demonstration methods to teach robots high precision assembly tasks using a real robot. While LTRON does not offer the realistic dynamics necessary to support traditional robotic manipulation, it does offer a high degree of scene complexity and compositionality which allows researchers to explore fine-grained spatial reasoning.

Recently construction and object-centric reasoning have become important topics in the reinforcement learning and AI community. Multiple datasets [53, 137] have been developed to train agents to build and reason about geometric forms using CAD software. While they support a small number of primitive-based modelling tools, our building environment supports constructing models from over one thousand discrete brick types. Other recent works [5, 31] have used reinforcement learning for block-stacking problems, and to create structures designed to achieve goals such as connecting or covering other blocks. Finally MineDojo[26] provides an open world exploration and building platform in Minecraft.

Researchers have also investigated the task of generating programs to describe and/or assemble shapes out of low level primitives [54, 84] and reason about the relationships between them [45].

LEGO bricks are popular construction toys that are often an early entry point for children to learn about building. They are also an excellent abstraction for real-world construction problems, which has led other researchers to explore using LEGO for various construction problems. Several approaches have been proposed to automatically construct LEGO assemblies from a reference 3D body [58]. For example, multiple authors [62, 88] have suggested methods for automated reconstruction based on genetic and evolutionary algorithms. Duplo bricks have also been used for tracking human demonstrations and assembly [38]. Lego bricks have also been used for reconstruction on mobile phones[68].

In contrast to these approaches, recent works have suggested data-driven deep learning approaches for LEGO problems based on generative models of graphs [122], and image to voxel reconstruction [64]. Similar to Break and Make, Chung et al. [16] propose a method for assembling LEGO structures from a reference image using interactive learning. Unlike LTRON these approaches use a use only a limited number of bricks, and do not support the large variety of bricks in the LEGO universe. Most similar to our work, Wang et al.[133] build LEGO structures from existing instructions. Our setting is more challenging because the agent must interactively learn about a previously unseen LEGO assembly rather than assuming instructions are already provided. Furthermore, the action space in LTRON is more difficult as it requires the agent to use a 2D cursor to interact with the scene and contains assemblies with bricks attached to the sides of objects, that cannot be described using simple stacking.

## 4.3 TASK AND DATA

The Break and Make task requires an agent to learn how to inspect a LEGO assembly using rendered images, and then use the information gathered in this way to rebuild the assembly from scratch. Both

the inspection phase and the construction phase are inherently interactive problems that require multi-step reasoning due to the ambiguities resulting from occlusions and the iterative nature of the building process. Many LEGO bricks have groups of similar neighbors which may appear identical under partial occlusion. Furthermore, complex structures often contain interior bricks that are not visible at all unless outer bricks are removed. These two factors mean that for many assemblies, there is no single viewpoint that completely captures an entire structure. Therefore in order to solve this problem an agent must often consider multiple viewpoints and take apart the assembly in order to fully understand it.

### 4.3.1 *LEGO Bricks*

A LEGO **brick** describes the shape and connection-point structure of a single LEGO part. While most LEGO bricks are a single rigid shape, some such as ropes and connector hoses are flexible. LTRON currently does not support these flexible components, so they are removed from all models before training. Some other bricks have moving parts, but in this case we break each of these into a separate brick shape for each moving component. We use polygon meshes extracted from the LDraw [50] package to represent all bricks. The **color** of a brick is represented as a single integer that refers to a specific RGB color value in a lookup table, which is consistent with LDraw conventions.

Each brick also contains a number of **connection points**. These describe how bricks may be connected to each other. The prototypical connection point is the short cylindrical stud that covers the top of many bricks in a rectangular grid, and the corresponding holes that cover the bottom. However, there are a large number of additional connection point types that exist in the LEGO universe, including technic pins, axles, clips, poles and ball/socket joints. In developing LTRON we have tried to faithfully represent as many of these as possible in order to provide a rich action space for interactive learning. Each of these connection points has a number of attributes related to

its physical dimensions and compatibility with other bricks. One important attribute of all connection points is **polarity**, which describes whether the connection point is an extrusion (positive polarity) or cavity (negative polarity). We use part metadata from the LDCAD[76] software package in order to detect these connection points on bricks and provide manipulation actions for them.

We refer to a collection of multiple bricks and their 3D locations as an **assembly**. Mathematically, this can be modelled as a set tuples $a = \{b_1, b_2, \dots b_n\}$ where each tuple $b_i = (s_i, c_i, R_i, t_i)$ represents an **instance** of a single brick. Each of these instances $b_i$ contains a brick shape index $s_i \in N_{shapes}$, a color index $c_i \in N_{colors}$, a 3D rotation $R_i \in SO(3)$ and a 3D translation $t_i \in \mathbb{R}^3$. The relative placement of the instances, combined with their shapes and the connection points associated with those shapes allow us to construct a set of **connections** describing a pair of connection points that are in very close proximity to each other and are mutually compatible.

### 4.3.2 *Break and Make*

The break and make task requires an agent to fully inspect a previously unseen LEGO assembly, then use its history of observations to build it again from scratch. This is presented as a two phase problem. In the first "Break" phase, the agent must disassemble and gather detailed information about a new assembly. Then when it takes a special "Phase" action, it the scene is cleared and the agent must rebuild the assembly from scratch. At each time step the agent receives a visual observation of the scene and must choose a high-level manipulation action, for example "Remove Brick" or "Rotate Brick," as well as low-level action parameters such as the rotation direction, and a 2D cursor location that the agent can use to specify which brick it wants to interact with. See Section 4.3.3 for details on the high level actions. Access to this fine-grained control makes this an exceptionally challenging interactive problem, as spatial information from the break phase must be remembered with enough detail to produce extremely precise actions during the make phase. In order for a neural network

to produce actions for this environment, it must produce two discrete distributions over the high level actions and action parameters, and also a heatmap over the image to predict the cursor location. See Sections 5.3.1 and 5.5.1 for details on how models can be built for this environment.

### 4.3.3 *Interface*

There are two versions of the LTRON environment interface. The first, LTRON V1 was introduced along with the sequential prediction models in Section 5.3. LTRON V2 was introduced with the InstructioNet model in Section and offers some simplifications and improvements over the original interface. LTRON V1 provides two virtual work spaces. The first, which we refer to as the **table** work space contains the agent's work in progress towards inspecting or assembling a model. The second work space, which we refer to as the **hand** contains only a single brick that the agent is about to place, or has just removed from the table workspace. Each workspace provides a 2D image rendered from a camera viewpoint that can be controlled by the agent. The table is rendered at $256 \times 256$ pixels and the hand is rendered at $96 \times 96$ pixels. When updating the interface in LTRON V2, we removed the **hand** workspace and instead opted for a single workspace containing the in-progress model. We also found that $128 \times 128$ pixels was sufficient when rendering this workspace, and drastically improved training efficiency. In addition, LTRON V2 introduces a number of technical improvements over LTRON V1. These updates provide better collision checking and support for a larger number of connection point styles than was available previously. Below we describe the actions available in the LTRON environment and describe the differences between V1 and V2. Figure 10 shows the components of the LTRON V1 action space, while 11 shows how the manipulation components have been updated in LTRON V2.

Many of the actions below require the agent to select one or more connection points on bricks in the hand or table workspace. To do this the agent must specify a 2D location in screen space, and the

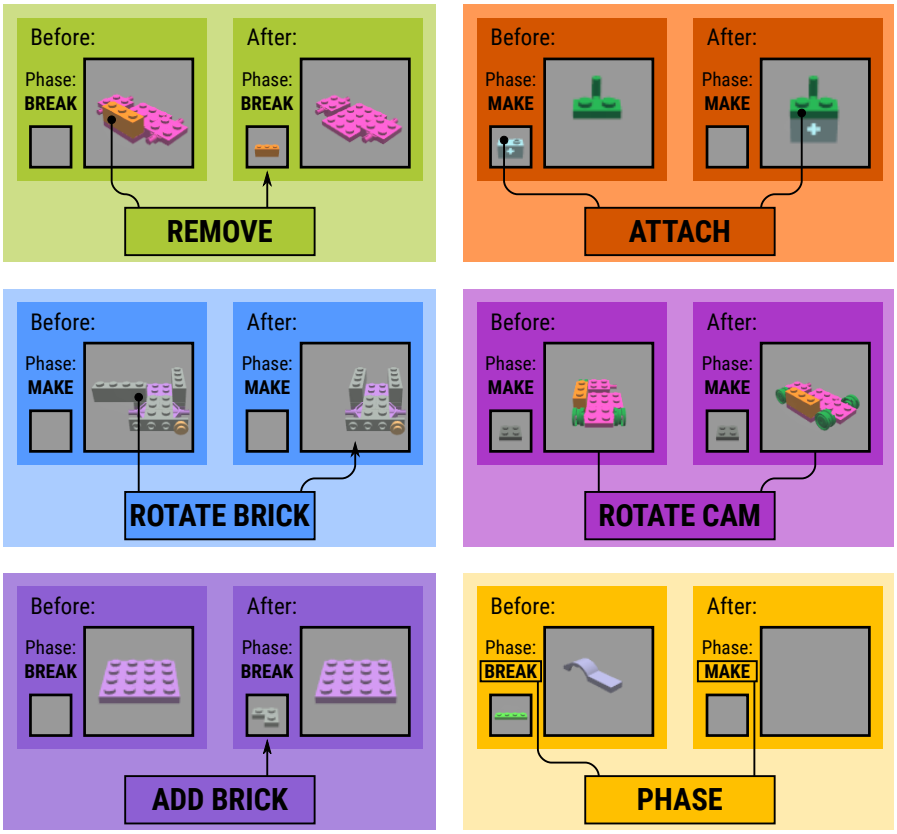


Figure 10: The six different high-level actions in LTRON V1.

polarity of the connection point it wishes to select. This is similar to the Alfred dataset[107] and AI2 THOR 2.0[60] which allow interaction with objects using pixel-based selection. LTRON V1 reduces the size of this action space by downsampling the resolution of this selection space by 4 to $64 \times 64$ for the table and $24 \times 24$ for the hand. LTRON V2, on the other hand uses a full resolution cursor space of $128 \times 128$.

**Remove**: The agent must select a valid connection point in the table workspace. If the brick can be removed without causing collision, the associated brick instance is removed from the table work space. In LTRON V1, the removed brick replaces any brick instance currently in the hand workspace.

**Attach**: In LTRON V1, the agent must specify valid and compatible connection points on one brick in the hand workspace and another in the table workspace. If the brick may be placed without collision, the brick in the hand is removed and placed into the table workspace attached to the specified connection point. If the table workspace is empty and there is no destination connection point to select, the agent

may select a valid connection point in the hand workspace alone. This will remove the brick from the hand workspace and add it to the table workspace by placing the specified connection point at the origin. In LTRON V2, the agent must specify both connection points in the table workspace, since there is no hand workspace.

**Add Brick**: The agent specifies a shape id and color id that it wishes to add to the assembly in progress. In LTRON V1, a new brick with the specified shape and color replaces any brick instance currently in the hand work space. In LTRON V2, the hand work space no longer exists, so the new brick is placed in the table work space, in a floating location above the existing assembly that is computed using the bounding box of existing bricks, or at the origin if no bricks are present.

**Rotate**: The agent must select a valid connection point on a brick in the table workspace. If rotating the brick will not cause collision, the brick is rotated by the a specified discrete angle about the primary axis of the connection point.

**Translate**: In LTRON V2, we added a Translate action mode which allows the agent to shift a brick by a fixed amount, using the cursor to select a connection point and specifying a discrete direction and offset value. We found this useful for helping the agent recover from small mistakes.

**Rotate Camera Left/Right/Up/Down/Frame**: In some cases it may be necessary to view an assembly from different viewpoints in order to effectively manipulate it, so we provide five actions for each workspace that the agent can use to manipulate the camera. The first four rotate the camera up, down, left or right about a fixed center point. Rotating left and right rotates by 45 degrees about the scene's up-axis, while rotating up and down alternate between a downward viewing angle 30 degrees above the center point and an upward viewing angle 30 degrees below the center point. The fifth **Frame** camera action moves the camera's fixed center point to the centroid of the current brick assembly.

**Switch Phase**: Finally there are two additional actions that switch from the Break phase to the Make phase, and that end the episode

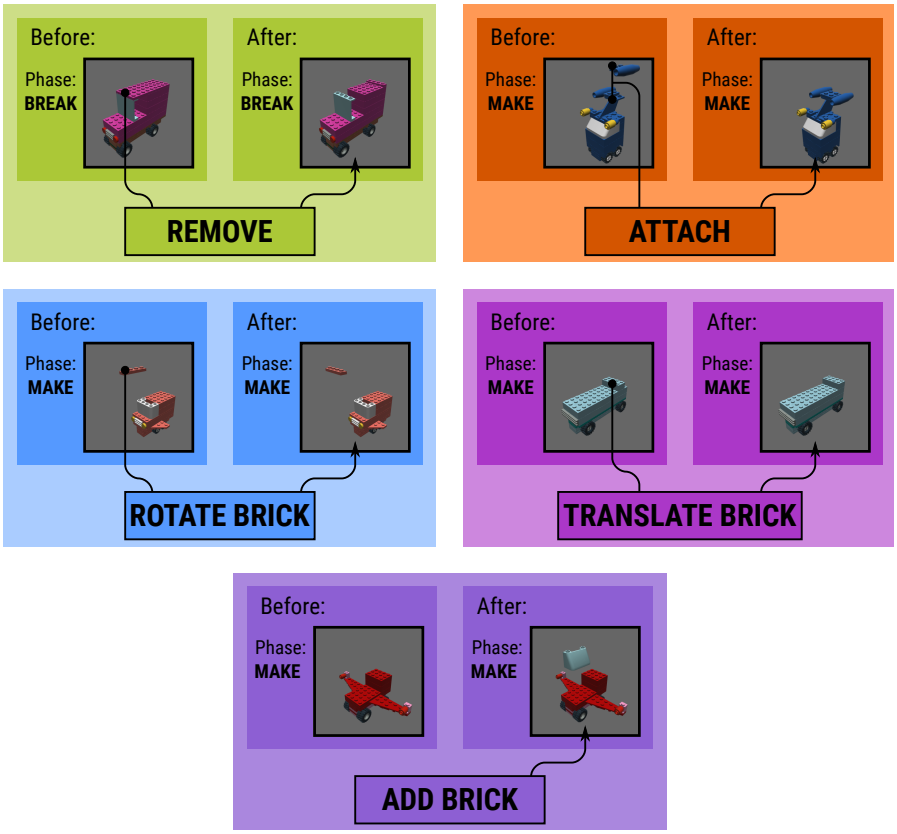


Figure 11: The manipulation actions in LTRON V2 without the extra hand viewport. This does not show the camera rotation and done actions which are unchanged from LTRON V1.

when the agent is finished building. Switching from the Break phase to the Make phase clears both workspaces.

### 4.3.4 *Dataset*

We provide three sources of scene files to train and evaluate agents on these tasks. The first is a set of fan-made reproductions of official LEGO sets that have been uploaded to the Open Model Repository (OMR) [51]. The second is a set of randomly constructed (RC) assemblies that we have generated with the LTRON simulator by attaching bricks together at random, designed as a large source of unstructured assemblies from a common distribution. Note that there are two versions of this data, one corresponding to LTRON v1 and another corresponding to LTRON V2. The difference is that an update in LTRON V2 it necessary to regenerate the randomly constructed data to ac-

count for a more precise collision checker. The final set of scene files are randomly constructed vehicles (RC-V). Like the randomly constructed assemblies, these were designed to use a smaller set of bricks from a common distribution, but are much larger models and contain a more recognizable structure.

The OMR contains 1727 files that are incredibly diverse, ranging in size from 5 to 7302 bricks. The sets come from over fifty distinct product categories such as "City," "Castle," and "Star Wars" that have been released over a span of several decades and use 1790 distinct brick shapes. These files have many properties in common with other naturally occurring data sources such as a long tail of increasingly rare bricks, and edge-cases that are difficult to model. This is a blessing to researchers who are interested in building models that can handle complex data distributions, and a curse to those looking for quick progress. In general these models are much larger than we are presently able to train on. Both the mean and the median number of bricks in a scene is more than one hundred, which is too large for the methods we consider in the next chapter. In order to generate a large amount of training data with smaller scenes, we have sliced these models into compact connected components using the connection points to find groups of connected bricks. In all cases we have used a master train/test split on the original files to inform the train/test on all slices of those files. Table 1 shows the train test splits for these slices. See the supplementary material for more details on the statistics, slicing procedure and cleaning process of this data.

In contrast to the OMR data above, our randomly constructed (RC) models are built by iteratively selecting brick shapes and colors at random and attempting to connect them to other bricks using randomly selected compatible connection points. This provides a much larger source of data that is in many ways easier to use for training, but unfortunately has many qualitative differences from the more natural OMR data. For example OMR scenes with a similar number of bricks tend to be much more compact than our randomly generated files as a byproduct of the human designers' preferences for tightly fitting configurations. Similarly, the OMR scenes exhibit more

Figure 12: Examples of the RC-Vehicles dataset.

symmetry, and more high-level structure such as clearly identifiable walls and branching structures. Despite these issues, this randomly constructed data is still very useful as a way to explore how the problem becomes easier as we reduce the number of brick shapes.

Our randomly constructed vehicles (RC-V) were generated with a series of scripted rules defining distributions over the vehicle dimensions, swappable components such as tires and windshield shapes, and optional features such as wings, headlights and helicopter blades. When combined, these distributions have over 21 bits of entropy defining the shape of the vehicle and 29 bits of entropy defining its color combinations. These models vary in size from 19 to 73 bricks, making them substantially larger and more complex than the random construction data. Examples of these vehicles can be seen in Figure 12.

## 4.4 EVALUATION

The Break and Make task requires a learning agent to visually inspect a LEGO assembly in order to gather enough information to then build it again from scratch. In order to assess the capability of a learned model, it is necessary to compare the generated assembly that it builds with the target assembly it is trying to copy. We provide four different metrics that attempt to estimate various aspects of the agent's success.

**$F1_b$ score:** The first metric is an F1 score over bricks in the two assemblies which we refer to as $F1_b$. This metric ignores pose and simply measures whether the agent was able to add the correct bricks to

| OMR | Train Scenes | Test Scenes | Total Scenes |
|---|---|---|---|
| Original Scenes | 1360 | 367 | 1727 |
| 2 Brick Slices | 136072 | 2000 | 138072 |
| 4 Brick Slices | 61514 | 2000 | 63514 |
| 8 Brick Slices | 28094 | 2000 | 30094 |
| RC V1 | Train Scenes | Test Scenes | Total Scenes |
| 2 Bricks | 50000 | 2000 | 52000 |
| 4 Bricks | 50000 | 2000 | 52000 |
| 8 Bricks | 50000 | 2000 | 52000 |
| RC V2 | Train Scenes | Test Scenes | Total Scenes |
| 2 Bricks | 100000 | 1000 | 101000 |
| 4 Bricks | 100000 | 1000 | 101000 |
| 8 Bricks | 100000 | 1000 | 101000 |
| RC-Vehicles | Train Scenes | Test Scenes | Total Scenes |
| Variable Size | 100000 | 1000 | 101000 |

Table 1: Train/test split sizes for the Open Model Repository and our Randomly Generated Data.

its estimated assembly regardless of how they are connected together. For this metric, we first remove pose information from the generated assembly $\hat{a}$ and the target assembly $a^*$ to produce a multi-set of brick shape and colors $m^* = \{(s_0^*, c_0^*) \dots (s_n^*, c_n^*)\}$ for the target assembly and another $\hat{m} = \{(\hat{s}_0, \hat{c}_0) \dots (\hat{s}_n, \hat{c}_n)\}$ for the assembly the agent generated. We can then compute true positives, false positives and false negatives as:

$$TP_b = m^* \cap \hat{m}, \quad FP_b = \hat{m} - m^*, \quad FN_b = m^* - \hat{m}$$

We then use these three quantities to compute an F1 score. An example showing the true positives, false positives and false negatives in an example scene are shown in Figure 13. Getting a score of 1.0 on this problem is necessary to rebuilding the assembly correctly, but it is not sufficient. This metric is still useful though because it allows us to categorize errors. If the agent was not able to rebuild the structure, but was able to identify the necessary bricks for that structure, then it may give us guidance for which aspect of the system needs the most improvement.

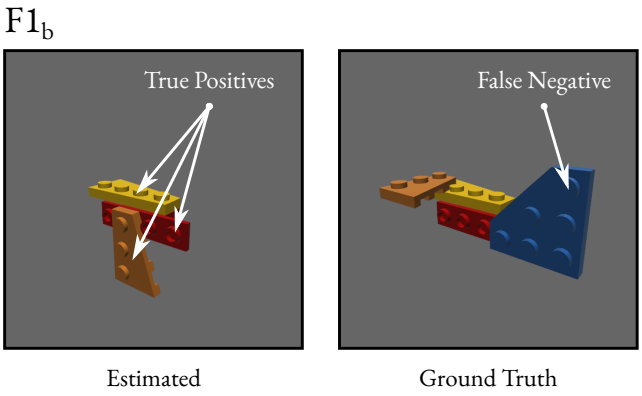


Figure 13: True positives, false positives and false negatives under the $F1_b$ metric.

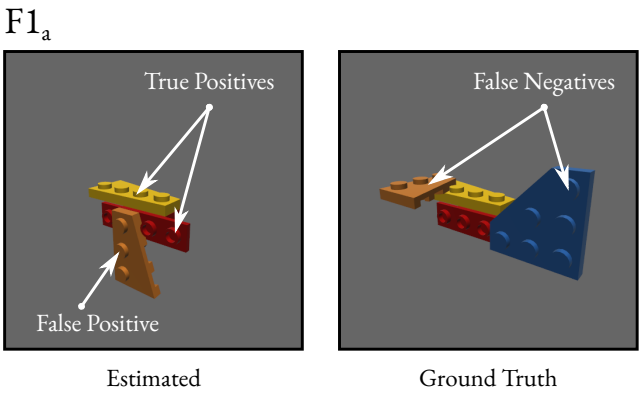


Figure 14: True positives, false positives and false negatives under the $F1_a$ metric.

**$F1_a$ score:** Unlike $F1_b$, $F1_a$ includes pose and is designed to measure the accuracy if the entire assembly. In this metric, we first define a rotation threshold $\theta_\epsilon$ and a distance threshold $d_\epsilon$ and say that two bricks $i$ and $j$ are *aligned* iff they have the same shape $s_i = s_j$ and color $c_i = c_j$ and their centers are close $\|t_i - t_j\| < d_\epsilon$ and the geodesic distance between their orientations is close $G(R_i, R_j) < \theta_\epsilon$. An example is shown in Figure 14. Note that while our action space is discrete, there are many bricks that contain connection points at odd angles. Connecting bricks to these connection points results in a wide variety of possible orientations that a brick can assume. Rather than dealing with this large, but discrete space of orientations directly, we instead use these thresholds with fairly tight tolerances for computing similarity.

Given that we care more about the *relative* position of bricks to each other, than their *absolute* position in the scene, we first compute a single rotation $R_0$ and translation $t_0$ that bring as many bricks in $a^*$ into alignment with $\hat{a}$ as possible. We then consider each brick in $\hat{a}$ to be a true positive if it is aligned with another brick in $a^*$ and consider it to be a false negative otherwise. Any brick in $a^*$ that is not aligned to a brick in $\hat{a}$ is a false negative. We then use these quantities to compute $F1_a$.

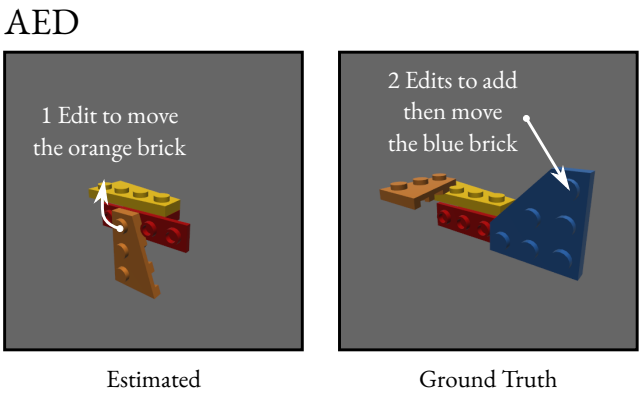


Figure 15: Example of edits considered by the AED metric.

**Assembly Edit Distance (AED):** While this $F1_a$ metric gives us a useful measure of similarity between two assemblies, it is possible that it may over-penalize some small mistakes. Consider the case where a long chain of bricks has been reconstructed correctly except for a single mistake in the middle. Because of the single rigid transform $R_0$ and $t_0$, we can only align either the top half or the bottom half of the reconstruction $\hat{a}$ with the target assembly $a^*$, and will incur a massive penalty for this single mistake. To mitigate this, we introduce Assembly Edit Distance (AED): we compute $R_0$ and $t_0$ as before, but once this is done, we mark all bricks that are aligned under this transformation and remove them from their respective assemblies. We then repeat this process with the remaining bricks and count how many rigid alignments must be computed until either the scene is empty, or the remaining bricks cannot be aligned because their shapes or colors do not match. We then add an additional edit penalty of 1, representing a single edit to remove the brick, for each brick in $\hat{a}$ left at the end of this process, and a penalty of 2, repre-

senting an edit to add the brick to the assembly and an edit to move it into place, for each brick in $a^*$ left at the end of the sequence. An example is shown in Figure 15.

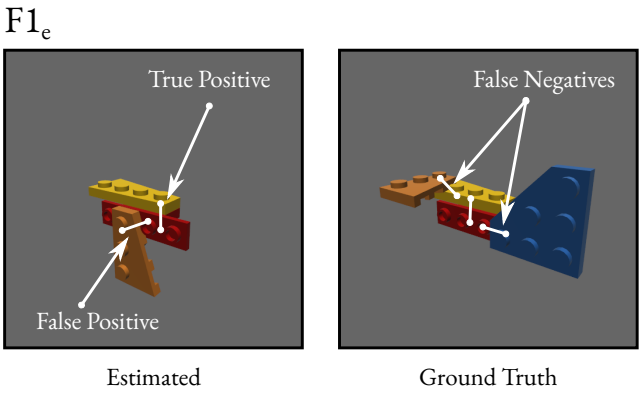


Figure 16: True positives, false positives and false negatives under the $F1_e$ metric.

**$F1_e$ score:** An added bonus of the **AED** metric is that it can be used to compute a matching between each brick in the generated assembly $\hat{a}$ and the target assembly $a^*$. This matching allows us to compute one final metric: an F1 score over edges ($F1_e$), or connections between two bricks. We consider every pair of bricks that are connected to each other in the generated assembly $\hat{a}$ to be a true positive edge if both of those bricks have been matched to a brick in the target assembly $a^*$ and the matching bricks in the target assembly are also connected to each other. Otherwise the connected pair is a false positive. Any connected pair in the target assembly $a^*$ that is not matched in this way is a false negative. Like $F1_b$ this metric can be considered necessary but insufficient, but again it is useful because it lets us characterize the errors made during the build process. An example is shown in Figure 16. If the agent was not able to determine the correct spatial alignment of the bricks, but is able to connect the right bricks together, then it may tell us the agent is struggling with the precise placement necessary to align bricks correctly. This is similar to a metric used in Visual Genome [61], but uses our iterative matching edit distance to compute assignment and has no action/attribute labels on individual edges.

## 4.5 CONCLUSION

LTRON and the Break and Make challenge offer an ideal environment to study a number of important technical problems in Machine Learning and Artificial Intelligence. First, LTRON provides an ideal environment for studying partially observable conditions that arise from complex physical structures. Second, the LTRON simulator offers an environment to explore interactive building and construction problems at a level of detail and granularity that has not previously been possible. Finally Break and Make provides an ideal setting to explore interactive learning algorithms designed for long-term credit assignment, as agents must connect low-level actions taken during disassembly and inspection with reward signals collected in the distant future during reassembly.

# 5

# SEQUENTIAL PREDICTION MODELS AND INSTRUCTIONET

## 5.1 INTRODUCTION

The Break and Make problem introduced in Chapter 4 is designed to train agents to understand and build complex structures using LEGO assemblies. In this problem, an agent must learn to build a previously unseen assembly by actively inspecting it. To do this the agent is given access to an interactive simulator that allows it to disassemble the structure in order to reveal hidden components and see how everything fits together. Once it is confident that it knows the structure, the agent is presented with an empty scene and must build the model again from scratch. This problem is designed to simulate a reverse engineering problem. The agent must take apart a complex structure in order to learn how to make it. By training agents to effectively rebuild these systems, we can discover new tools for understanding and building intelligent systems that can reason about complex structures.

The Break and Make problem is quite challenging, as it requires long-term memory, and interaction with a complex visual environment using a 2D cursor-based action space. In this chapter we describe progress that has been made on this problem. We first show that approaches using sequential prediction models based on Transformers and LSTMs can achieve modest success on small assemblies. These models were trained using LTRON V1.

We then discuss a new model InstructioNet for this problem that uses an explicit memory to store a stack of self-curated instruction images. InstructioNet slowly adds to this memory by iteratively disassembling part of the model, then saving its most recent observation to the top of this instruction stack. This stack provides the model with

a series of short-term visual targets to use when reassembling the model later on, just like a real-world LEGO instruction book. When rebuilding the model the agent only has to reason about its current observation and the page of the instruction stack that it is currently working on. Once the agent's current assembly matches this instruction, it can turn the page and get a new short-term target to work towards. Figure 17 shows a successful example of this on the RC-Vehicles dataset. InstructioNet was trained using LTRON V2. While this slightly complicates direct comparison between the sequential prediction models and InstructioNet, the relative improvement of InstructioNet is larger than can be plausibly explained by the differences between LTRON V1 and V2.

In addition to this memory-based model, we also detail several practical components that are either necessary or improve training performance in this space. These include online imitation learning, conditional action heads and new loss functions for the dense 2D cursor-based action space.

## 5.2 RELATED WORK

### 5.2.1 *Memory*

Memory structures for interactive problems have been studied for decades. Early attempts at implicit (problem-agnostic) memory structures include simple RNNs[55, 98] and more complex variants such as LSTMs[44] and GRUs[14]. These methods propagate information forward in time using specialized network architecture. Neural Turing Machines [37] use a learned external memory module to read and write to long term storage. Recently attention-based transformers[126] have become one of the most popular ways to build neural networks that rely on past information to make future decisions. While transformers are quite effective, they are computationally expensive for large sequences of observations, which limits their use for very long-term memory.


Figure 17: The break phase of an example of InstructioNet completing the Break and Make task on a previously unseen example from RC-Vehicles. Our model saves 34 distinct images to the instruction stack over the break phase. The make phase for this sequence is shown in Figure 18.

In contrast to implicit memory structures, explicit memory uses some knowledge of the environment or task in order to build a structure more appropriate to the problem. This can come in the form of geometric or topological map building such as in SLAM[15, 42, 65], semantic maps in embodied navigation[34] or more complex struc-

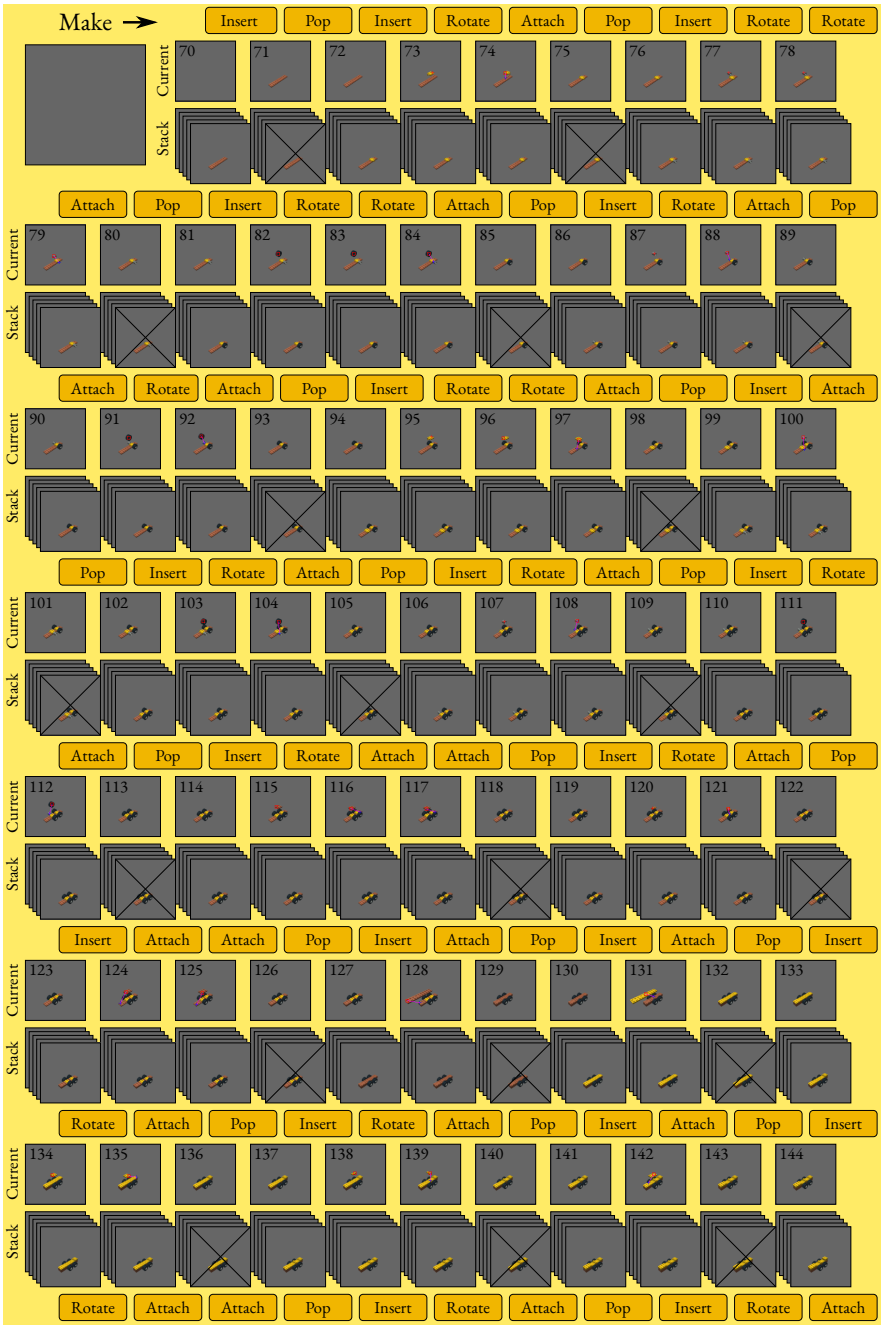


Figure 18: The first half of the make phase of InstructioNet completing the Break and Make task on a previously unseen example from RC-Vehicles. The make phase of this sequence is shown in Figure 17. The second half of the make phase is shown in Figure 19.

tures that combine multiple components[11]. Our InstructioNet uses an explicit memory structure designed around the intuitive understanding that assembling a structure can be completed in the reverse order of disassembling it. This motivates the use of a stack that allows

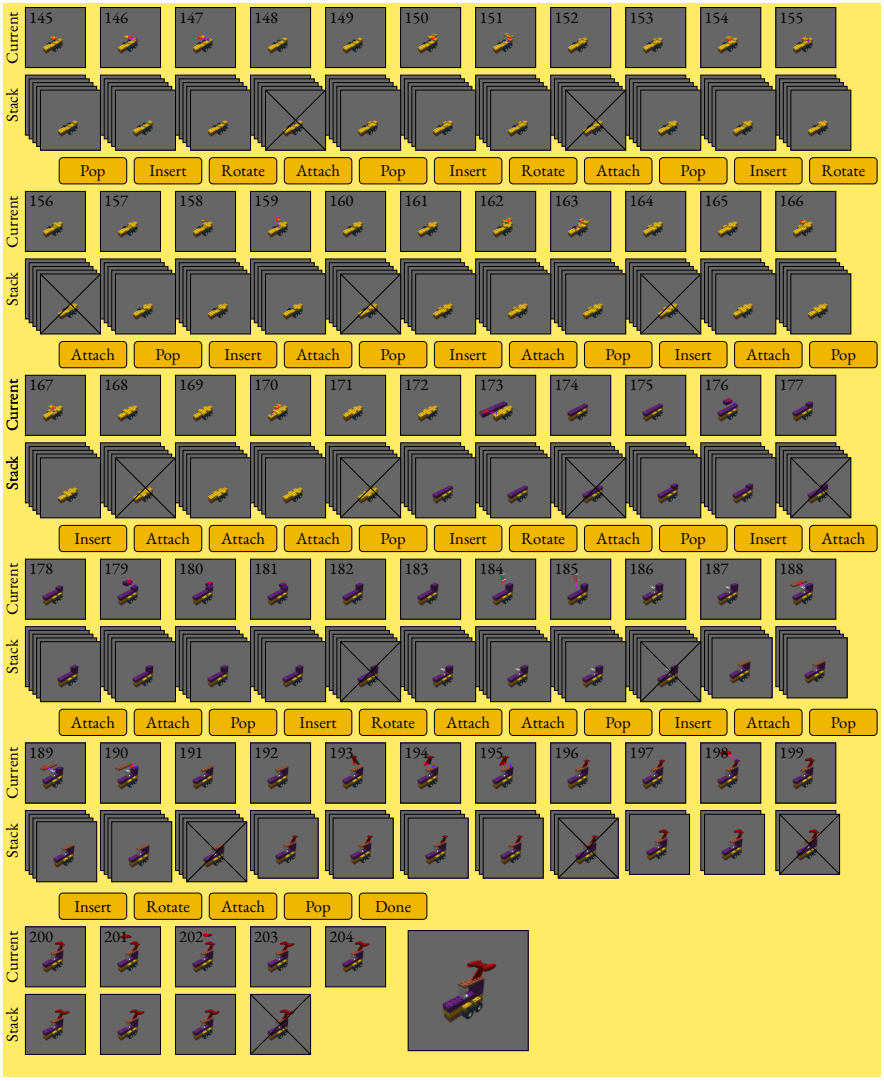


Figure 19: The second half of the make phase of InstructioNet completing the Break and Make task on a previously unseen example from RC-Vehicles. The first half of this sequence is shown in Figure 18. The break phase is shown in Figure 17.

the agent to sequentially build up a series of experiences, and then pop them off one-by-one in reverse order later.

### 5.2.2 *Inverse-Graphics*

Reconstructing geometry and reasoning about 3D structures from images has been an important issue in research fields such as computer-aided design[23, 145] and robotics [12, 63, 113]. In particular, prior works such as [67, 138] use 3D shapes while [82, 87, 149] use images to guide the inverse inference process. However, when building com-

plex structures such as LEGO models, it is challenging to generate a set of sufficient visual images to predict the reconstruction without dynamically interacting with the object or the environment.

## 5.3 SEQUENTIAL PREDICTION MODELS

### 5.3.1 *Architecture*

Our sequential prediction models use LSTM and Transformer sequence-based models to keep track of the entire sequence of observations in order to make informed decisions later. These models were built using the LTRON V1 interface.

The most competitive of these models, which we refer to as StudNet are based on the popular Transformer[125] architecture. In this model, the input images are first broken into $16 \times 16$ pixel tiles similar to the VIT architecture[22]. The model then extracts features from each tile using a learned linear layer and two positional embeddings, one that encodes the tile's XY coordinates in the image and another that encodes the tile's frame id in the temporal sequence. We unroll the XY coordinates of the image into a single one-dimensional coordinate space and concatenate the coordinates of the table image and the hand image so that a single index can be used to determine which image the tile belongs to and its 2D location. These tile features are then fed into a transformer that uses GPT-style[89] causal masking to prevent tokens that occur early in the sequence from paying attention to later tokens.

Transformer models notoriously require very large memory due to the $N^2$ attention mechanism that allows for long-range connectivity between tokens in the sequence. In order to make this architecture tractable on the long sequences of tokens produced by LTRON, we employ a simple but effective data compression technique: at each step we only include image tiles which have changed since the previous frame. In the first frame, we also remove all tiles that contain only the solid background color. Given that manipulating a single brick usually only changes a small portion of the image, this results

in substantial savings. In addition to the image tiles, we provide a token that specifies the current phase (Break or Make).

The Break and Make task requires an agent to take both discrete high-level actions as well as select low-level pixel locations to assemble and disassemble bricks. We model this using five separate heads: a mode head that selects one of the primary action types (see Figure 10) to take at each step, a shape selector head and color selector head that are used when picking up a new brick, and a table location and hand location heatmap that is used to select pixel locations for brick interaction. The shape and color heads are linear layers that project from the transformer hidden dimension to the number of shapes and colors used in a particular experiment. Unfortunately we cannot decode a dense heatmap for the pixel locations directly from the tokens coming out of the transformer encoder because our compression strategy throws many of these tokens away. We experiment with two different decoder styles to address this issue.

The first, which we refer to as StudNet-A uses a separate transformer decoder layer. This layer receives a dense positional encoding as the query tokens, and the output of the encoder as the key and value tokens resulting in a dense output. Although some details differ, this is similar to the Perceiver IO[49] and MAE[40] models that do primary computation at a lower resolution and use cross-attention to expand to dense output when necessary. We decode at $16 \times 16$ resolution and upsample to $64 \times 64$.

The second decoder, which we refer to as StudNet-B, feeds the input images through a small convolutional network to produce a $64 \times 64$ feature map for the table and a $24 \times 24$ feature map for the hand. In our experiments we use the first layer of a Resnet-18[41] for this. Two additional heads, one for the table workspace and another for the hand workspace, compute a single feature from a per-frame readout token, and use dot-product attention with the convolutional feature map to produce a heatmap of click locations.

We also use an LSTM baseline. This model takes guidance from the ALFRED Dataset [107] which similarly requires an agent to reason about high level actions as well as pixel-based selection. In this

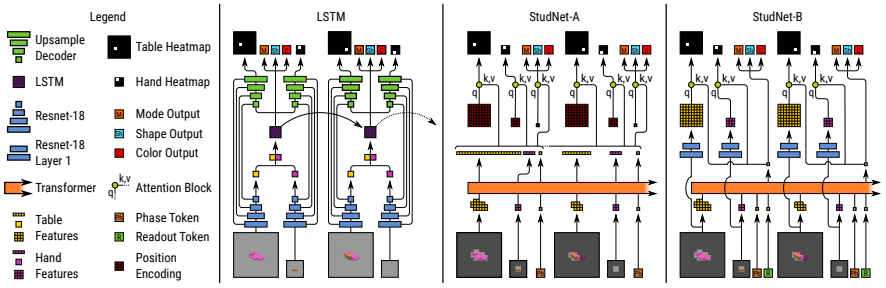


Figure 20: Sequential Preidction Model network architectures.

network, the images from both the table and hand workspaces are fed through a Resnet-18 backbone [41], and are then concatenated and passed to an LSTM. The output of this LSTM is then decoded using five heads. The first three produce the mode, shape and color actions. The second and third heads tile the LSTM feature to match the shape of the table and hand resnet features, then upsample these with UNET-style [92]/FPN [71]-style lateral connections from the image encoder to produce a dense feature that is used to select cursor locations. In experiments we use two versions of this model, one trained from scratch, and another where the Resnet-18 backbone has been pretrained on a pixel-labeling task designed to densely predict brick shapes and colors. The network structure for these models can be found in Figure 20

### 5.3.2 *Training*

We train the sequential prediction models above using behavior cloning on offline sequences. In order to generate these sequences, we have developed a visual planner that interfaces directly with LTRON. This planner uses hidden state information combined with rendered occlusion maps to reason about which bricks are currently visible in the scene and plan assembly and disassembly sequences accordingly. While this information allows the planner to determine which bricks can be manipulated, it does not strictly guarantee that the visual information acquired during the planning process is enough to unambiguously resolve the full 3D structure of the scene, or correctly identify the shapes of every brick. This is due to the fact that many brick

shapes look identical to others when viewed from certain angles or under partial occlusion, and so it may be important to change the camera viewpoint or disassembly order to resolve these ambiguities. Due to the large number of brick shapes, we have not attempted to exhaustively catalogue when and how these ambiguities arise for every combination of brick shapes. Therefore the planner currently has no way of knowing when these conditions occur.

The visual planner can be quite slow and uses a two-stage process that requires reasoning over groups of individual actions. Both of these issues make it difficult to use the planner as an expert for methods such as DAgger[96] that require the expert to produce labels for sequences generated by the model. We therefore do not attempt to solve Break-and-Make using these approaches at the present time, and limit ourselves to methods that can train on a static dataset. Building improved planners with the ability to quickly provide high-quality actions would be beneficial for this problem.

## 5.4 SEQUENTIAL PREDICTION EXPERIMENTS

### 5.4.1 *Break and Make Results*

We evaluate the baseline models on the Random Construction and Sliced OMR datasets at three fixed scene sizes: two bricks, four bricks and eight bricks. While these scenes are quite small compared to the complete models in the Open Model Repository, they often require dozens of interaction steps to complete and present a challenging problem.

On the random construction data with six brick types and six colors, all models make substantial progress on small scenes. Table 2 shows the models' performance on each of these tasks under the four metrics described in Section 4.4. Note that performance drops substantially as the scenes get larger.

The Sliced OMR dataset contains 1790 brick shapes and 98 colors making it structurally and visually significantly more challenging than the random construction dataset. Table 3 illustrates that all of the

Random Construction (RC)

| | $F1_b \uparrow$ | $F1_e \uparrow$ | $F1_a \uparrow$ | AED ↓ |
|---|---|---|---|---|
| RC-2 | | | | |
| LSTM | 0.61 | 0.38 | 0.43 | 2.16 |
| Pretr. LSTM | 0.70 | 0.51 | 0.45 | 1.89 |
| **StudNet-A** | **0.90** | **0.86** | **0.58** | **1.11** |
| StudNet-B | 0.87 | 0.77 | 0.57 | 1.30 |
| RC-4 | | | | |
| LSTM | 0.41 | 0.09 | 0.13 | 7.25 |
| Pretr. LSTM | 0.25 | 0.01 | 0.08 | 8.46 |
| StudNet-A | 0.56 | 0.29 | 0.24 | 5.80 |
| **StudNet-B** | **0.64** | **0.34** | **0.25** | **5.48** |
| RC-8 | | | | |
| LSTM | 0.02 | 0.00 | 0.02 | 16.05 |
| Pretr. LSTM | 0.03 | 0.00 | 0.02 | 16.09 |
| StudNet-A | 0.02 | 0.01 | 0.01 | 15.87 |
| **StudNet-B** | **0.38** | **0.14** | **0.12** | **13.90** |

Table 2: Test results of our four models on randomly constructed assemblies across three scene sizes. See Section 4.4 for details on metrics.

models we tested score significantly lower on this dataset. In particular our StudNet-A transformer architecture fails to correctly learn to switch from disassembling to rebuilding the LEGO models and thus scores very poorly across all of our metrics. Our StudNet-B architecture shows the best overall performance, demonstrating that progress can be made even on the most challenging 8 brick dataset. This illustrates not only that Break-and-Make is a fundamentally hard problem, but also that its difficulty can be regulated by the dataset selection while maintaining the same action space and problem structure. This allows future work to make meaningful progress on simple datasets like Random Construction and then progress to ever more difficult datasets.

Given the relatively low performance of the models presented here on the break and make task, we also conducted several experiments designed to discover which part of this problem is most difficult.

Open Model Repository (OMR)

| | $F1_b \uparrow$ | $F1_e \uparrow$ | $F1_a \uparrow$ | AED ↓ |
|---|---|---|---|---|
| RC-2 | | | | |
| **LSTM** | 0.43 | 0.33 | 0.31 | **2.76** |
| **Pretr. LSTM** | **0.45** | **0.34** | **0.33** | 2.86 |
| StudNet-A | 0.00 | 0.00 | 0.00 | 3.99 |
| StudNet-B | 0.36 | 0.18 | 0.29 | 3.74 |
| RC-4 | | | | |
| **LSTM** | 0.10 | **0.03** | 0.07 | **7.67** |
| Pretr. LSTM | 0.04 | 0.01 | 0.03 | 8.16 |
| StudNet-A | 0.00 | 0.00 | 0.00 | 8.08 |
| **StudNet-B** | **0.14** | 0.02 | **0.12** | 8.30 |
| RC-8 | | | | |
| LSTM | 0.01 | 0.00 | 0.01 | 16.01 |
| **Pretr. LSTM** | 0.00 | 0.00 | 0.00 | **15.97** |
| StudNet-A | 0.00 | 0.00 | 0.00 | 16.01 |
| **StudNet-B** | **0.05** | 0.00 | **0.04** | 16.05 |

Table 3: Test results of our four models on sliced Open Model Repository assemblies across three scene sizes. See Section 4.4 for details on metrics.

### 5.4.2 *Automatic Brick Selection*

In order to test various components of this system, we first modified the environment to provide the correct "Add Brick" actions (selection of brick shape and color to insert into the scene) automatically when necessary. If numbers on this experiment improved dramatically, this would indicate that the models were having difficulty remembering the brick shapes and colors that were observed during the Break phase. For the sake of space, we ran this experiment on the randomly constructed 2-brick assemblies using the StudNet-B model. As shown in Table 4 this yields a very small performance gain. This result, taken together with the fact that the $F1_b$ score strongly outperforms the $F1_a$ score in almost all experiments indicates that remembering the brick shape and color, and learning when to insert them is not a primary source of error.

| **Ground Truth Insertion** | | | | |
|---|---|---|---|---|
| Random 2 Brick | $F1_b$ | $F1_e$ | $F1_a$ | AED |
| StudNet-B Original | 0.87 | 0.77 | 0.57 | 1.30 |
| StudNet-B GT-Insert | 0.88 | 0.84 | 0.57 | 1.12 |

Table 4: Test results when providing ground truth brick insertion operations (bottom) compared to original performance (top).

### 5.4.3 *Detection*

We also extracted frames from the random 2-brick assemblies and trained a single-frame FCOS[123] detection model with additional heads to predict 3D position and orientation. This is designed to determine if estimating the 3D pose of the bricks in a single frame is a possible point of failure. To evaluate this model, we use an AP score where we consider an estimated brick to be a true-positive match with a ground truth brick if their shapes and colors match, and their poses are within 30 degrees and 8mm of each other. In this setting a ResNet50 backbone scores 0.97 AP, a ResNet18 backbone scores 0.96 AP and a transformer backbone scores 0.81 AP. These results indicate that estimating the identities and poses of the bricks in a single frame is also not a major challenge.

Taken together, these automatic brick selection and detection results provide indirect evidence for the hypothesis that the most challenging part of this problem is the need for spatially-precise long term memory, and a large interactive action space.

### 5.4.4 *Fine Tuning*

We also conducted an experiment to test whether a network trained on random construction assemblies could be fine-tuned to the OMR assemblies. Due to the mismatched number of brick shapes and colors, we trained new color and shape heads for the OMR data. As shown in Table 5 performance actually gets slightly worse, except for edit distance, which improves slightly.

### 5.4.5 *Human Baseline*

In order to make sure the Break and Make task is possible using the interface provided, we also conducted a small user-experiment. Using a rudimentary interface, we were able to perfectly reconstruct 8 out 10 randomly sampled scenes from the Random Construction 2 dataset. The two failures contained a single placement mistake each which could not be fixed within the maximum episode length. This would yield F1b: 1.0, F1e: 1.0, F1a: 0.90 and an AED: 0.2, which is far better than any of the baseline models. This shows that the task is feasible, and that there is substantial room for improvement in the approaches here.

### 5.4.6 *Qualitative Examples*

Figure 21 shows ten randomly chosen target assemblies and assemblies predicted by the Studnet-B model on each dataset. Very few examples are completed precisely correct. In the random 2-brick, 4-brick and the OMR 2-brick examples, the agent is able to build an assembly using the correct bricks but fails to connect them correctly. The model largely fails to build anything resembling the target for the random 8-brick and OMR 4-brick and 8-brick scenes. This clearly demonstrates the difficulty of this problem even in simple settings, and shows how the problem becomes more difficult as the number of bricks per scene increases, and the total number of bricks used in the dataset increases. See the supplemental video for an example of a successful episode on a 2 brick model generated with the Studnet-B model.

**Pretrain/Fine Tune**

| OMR 2 Brick | $F1_b$ | $F1_e$ | $F1_a$ | AED |
|---|---|---|---|---|
| StudNet-B Original | 0.36 | 0.18 | 0.29 | 3.74 |
| StudNet-B Finetune | 0.29 | 0.11 | 0.25 | 3.32 |

Table 5: Test results when using a model pretrained on the Random-2 dataset and fine-tuning on OMR-2 (bottom) compared to training from scratch on OMR (top).

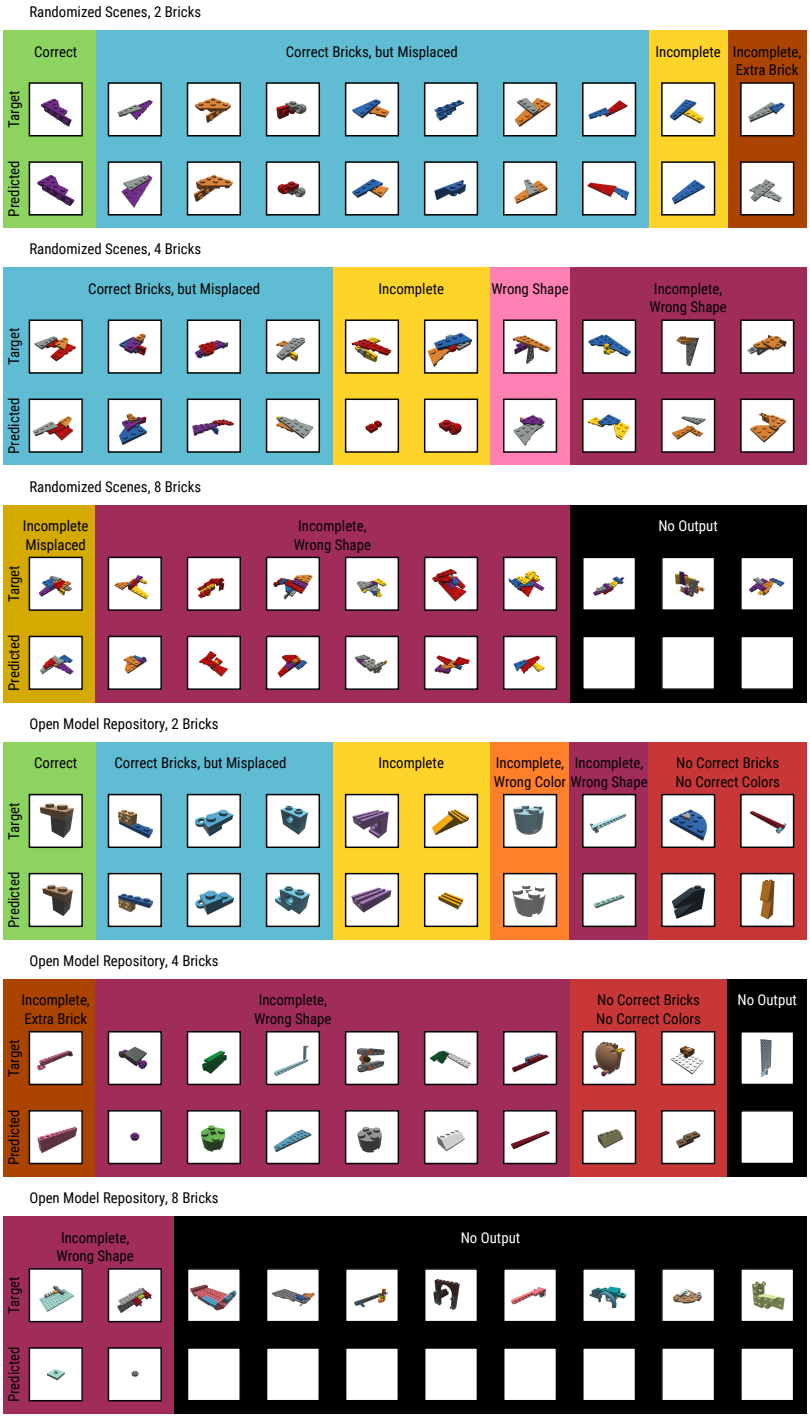


Figure 21: Qualitative examples of full reconstruction using the StudNet-B model.

## 5.5 INSTRUCTIONET MODELS

### 5.5.1 *Architecture*

Our new model InstructioNet uses the LTRON V2 interface, and works by storing an explicit stack of instruction images in order to

remember the structure of an assembly at various stages of deconstruction. To do this, we augment the action space discussed in the previous section with two additional **Push Instruction** and **Pop Instruction** actions. Push takes the current image from the simulator and adds it to the top of the instruction stack, while Pop removes the top image from the instruction stack.

Our learned policy takes in the current image from the simulator as well as the top image of the instruction stack. During the Break phase when the agent is trying to gather more information about the LEGO assembly, the agent can compare these two images and see if they are similar. If they are, then the agent should disassemble the model further. Otherwise, the agent should take the Push action to store the new information that has just been gathered. After completing this process several times, the agent should have an instruction stack with an image of the fully completed assembly on the bottom, and increasingly disassembled images as you move closer to the top. At the start of the Make phase, the agent will be presented with an empty scene, so the current image from the environment will be empty, while the top of the instruction stack will contain the last brick the model saw during the disassembly process. The agent must then build until the assembly in the current image matches the assembly in the top instruction image. When they match, the agent can take the Pop action and will then reveal a new instruction image with slightly more of the original assembly remaining. In this way, the model only needs to reason about two images at a time, which greatly reduces the complexity of the policy.

The learned policy is implemented using a modified vision transformer[22] with multiple heads that are responsible for different components of the action space. Like the StudNet baselines, this model tokenizes the workspace and the instruction images into $16 \times 16$ pixel patches. Given that the LTRON V2 environment produces images that are $128 \times 128$ pixels, this results in 64 tokens per image. The patches from both images are passed through a single linear layer and added to a learned positional encoding. The model then concatenates a single decoder token and a binary embedding of the current phase

(Break or Make) for 130 total tokens. The transformer consists of 12 blocks with 512 channels and 8 heads each. Note that because the model only needs to consider two images at the same time, it is not necessary to employ the compression trick that discards unchanged tiles in this model.

To compute an action, the output of the decoder token is then passed through a set of decoder heads to predict distributions for the action mode such as Remove, Attach, or Rotate Brick and mode-specific parameters such as Rotate Direction when performing a Rotate action or Brick Shape and Color when inserting a new brick. In order to predict 2D cursor click and release locations, the 64 output tokens of transformer blocks 3,6,9 and 12 that correspond to the current image are combined using two separate DPT[91] decoders to produce dense feature maps at the resolution of the original input image. We found it beneficial to condition these click locations on the high level action and parameters sampled from the initial decoder heads. This is accomplished by passing the sampled high level actions to an embedding layer and adding the resulting feature to the output of the decoder token. This value is then used to compute a distribution over click locations using dot product attention over the dense features computed by the DPT decoder. This conditional structure is similar to models used in game AI with complex action spaces [128]. Figure 22 shows our policy model and how these components fit together.

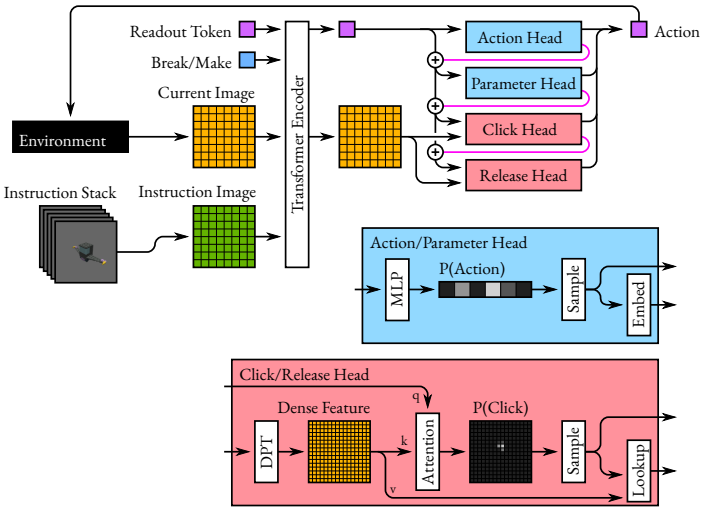


Figure 22: Architecture of the InstructioNet model.

### 5.5.2 *Training*

When developing LTRON V2, we have built a fast expert in LTRON V2 that can provide online supervision for trajectories generated by the learning model during training. This allows us to train the InstructioNet model using online imitation learning similar to DAgger[96]. This is in contrast to the planner used to train the baselines in LTRON V1, which uses an expert that is too slow for online labeling and can therefore only be used to generate a single static dataset. Note that while our new expert is much faster, it is weaker and is not able to construct plans in cases where the agent makes too many mistakes or deviates too far from the target assembly. In these cases, we simply terminate the training episode early. In many cases, there are multiple possible best actions that the expert could suggest. When this happens, one of them is selected at random.

The online training algorithm alternates between generating new data by acting in the environment according to either the expert or the learning model, and then training on a randomized subset of the data generated over the past several iterations. When generating data, a fixed percentage of the environment steps are generated by sampling actions according to the expert, and the rest are sampled by acting according to the agent. For our main experiments, we found that a mix of 75% expert-generated and 25% model-generated data worked well. Note that regardless of which model is controlling the simulator, the expert's actions are always used for supervision. Incorporating trajectories generated by the agent in this way allows the model to learn to recover from its mistakes: when the model takes an inappropriate action it will reach part of the state space that would not have been encountered if acting according to the expert, yet seeing the expert's advice in these states shows the model how to correctly recover from this behavior.

As noted by Czarnecki et al. [18], this method of using the model to generate data with direct supervision from the expert can be unstable. This instability occurs frequently when the data distribution shifts as the model gets better at some parts of the state space and learns to

take different actions. This problem is exacerbated in settings with long trajectories where many nearby frames appear similar to each other. If proper care is not taken, the model can quickly overfit to the data it has most recently seen and catastrophically forget prior examples. The original DAgger algorithm addresses this by periodically retraining the entire model on all data collected so far. Unfortunately, this is somewhat impractical for long training runs with millions of steps. To mitigate this issue we maintain a replay buffer and train on data randomly sampled from this experience. Algorithm 1 provides the pseudo-code we use to train our model.

**Algorithm 1** Online Training

**Require:** LTRON Environment $Env$
**Require:** Expert $\pi^*$
**Require:** Total Steps $N_{steps}$
**Require:** Epoch Rollout Steps $N_{steps/epoch}$
**Require:** Epoch Train Steps $N_{train/epoch}$
**Require:** Max Dataset Size $C$
**Require:** Expert-guided percentage $\alpha$

```
π_θ = InitializePolicy()                    ▷ Initialize the Policy
D = {}                                      ▷ Initialize an empty dataset
o = ResetEnv(Env)                           ▷ Get the first observation
while i < N_steps do
    for j = 1 to N_steps/epoch do           ▷ Collect data
        a* ~ π*(o)                          ▷ Sample action from expert
        a ~ π_θ(o)                          ▷ Sample action from Learner
        if j/N_steps/epoch < α then
            o = StepEnv(Env, a*)            ▷ Execute expert's action
        else
            o = StepEnv(Env, a)             ▷ Execute learner's action
        end if
        Append(D, (o, a*))                  ▷ Add experience to D
        if |D| > C then
            Evict(D)                        ▷ Evict the oldest entry from D
        end if
    end for
    for j = 1 to N_train/epoch do           ▷ Train
        (o, a*) ~ D                         ▷ Sample a batch
        Train(π_θ(o), a*)                   ▷ Train learner with expert action
    end for
    i = i + N_steps/epoch
end while
```

### 5.5.3 *Cursor Losses*

We also experimented with several approaches to computing losses for the cursor click and release locations in the InstructioNet model. During online training, we would like the click-and-release decoders to each produce a distribution over locations that can be sampled in order to generate a variety of training data. To compute the probability of clicking on a particular pixel $p(i, j)$, the dense raw value $x_{i,j}$ predicted at that pixel location is normalized using a standard softmax:

$$p(i, j) = \frac{\exp(x_{i,j})}{\sum_{i',j'} \exp(x_{i',j'})}$$

When the expert's action suggests clicking on a particular LEGO connection point in the scene, there are usually multiple "acceptable" pixels that correspond to the same connection point which complicates the choice of loss function that we use.

One option is to use binary cross-entropy loss using the mask of acceptable pixel locations as a target. This assumes a different probabilistic interpretation of the output, one where multiple pixels can be chosen at the same time instead of just one, but still may be a useful way to encourage the model to put a high probability on the acceptable pixels. This loss function encourages the model to increase the probability of all acceptable pixels, without considering the cross-pixel relationships.

Another option commonly used in keypoint detection is to construct target heatmaps using a small Gaussian distribution around correct locations and supervise the output values using a mean-squared-error loss [9]. We opted against this as it does not lend itself well to softmax sampling, and may add probability mass outside the desired pixel boundaries. However, we can use the mean-squared-error loss to simply push all acceptable pixels toward a large positive constant, and all unacceptable pixels towards a large negative constant. Here we used a constant such that if only one pixel in the image assumed the positive constant, and all others assumed the negative constant,

the probability of selecting the single pixel in the softmax would be 0.999. For $128 \times 128$ pixel images, these constants are $\pm 8.3$.

Finally, we consider a loss function in which the probability of acceptable pixels is summed and the probability of unacceptable pixels is summed forming a new two-way distribution. We then supervise this new distribution to maximize the probability of choosing an acceptable pixel using cross-entropy. This loss can be expressed as:

$$L = -\log \sum_{i,j} y_{i,j} \frac{\exp x_{i,j}}{\sum_{i',j'} \exp x_{i',j'}}$$

This allows the model to place probability mass on any of the acceptable pixels while decreasing the probability mass of all unacceptable pixels. We find that this loss function empirically outperforms the others and discuss these findings in Section 5.6.3.

## 5.6 INSTRUCTIONET EXPERIMENTS

### 5.6.1 *Break and Make Results*

To evaluate the effectiveness of the InstructioNet model, we trained it on our modified version of the Break and Make task using the randomly constructed assemblies and vehicles discussed in Section 4.3.4. In order to focus on construction ability and avoid the confounding issues of long-tailed part distributions, we did not train on the Open Model Repository (OMR) data. Table 6 shows the performance of our model on these datasets compared to the reported numbers of the LSTM, Studnet-A, and Studnet-B baselines from [132]. Note that due to the updated observation space and the small changes to the random construction data due to the improved collision checker, these models should not be considered to have been trained in the same environment, and so the comparisons are only approximate. Also note that while both LTRON V1 and LTRON V2 allow for camera rotation, we found that rotating the camera was not necessary to successfully reassemble the models in the datasets considered here, so we used a fixed camera angle for all InstructioNeg experiments. Regardless of

these differences, the InstructioNet model is able to reconstruct large models with much greater accuracy than was previously possible, and the large performance gap clearly demonstrates a new level of capability. Note that we were not able to train the LSTM and Studnet methods on the new RC-Vehicles data as they require the entire history of past frames to make each decision. The RC-Vehicles assemblies can take over 150 steps to correctly disassemble and reassemble, which are much larger sequences than we could effectively train on available hardware.

| | $F1_b \uparrow$ | $F1_e \uparrow$ | $F1_a \uparrow$ | AED ↓ |
|---|---|---|---|---|
| RC-2* | | | | |
| **InstructioNet** | **0.98** | **0.95** | **0.93** | **0.18** |
| LSTM | 0.61 | 0.38 | 0.43 | 2.16 |
| Studnet-A | 0.90 | 0.86 | 0.58 | 1.11 |
| Studnet-B | 0.87 | 0.77 | 0.57 | 1.30 |
| RC-4* | | | | |
| **InstructioNet** | **0.80** | **0.69** | **0.71** | **2.39** |
| LSTM | 0.41 | 0.09 | 0.13 | 7.25 |
| Studnet-A | 0.56 | 0.29 | 0.24 | 5.80 |
| Studnet-B | 0.64 | 0.34 | 0.25 | 5.48 |
| RC-8* | | | | |
| **InstructioNet** | **0.68** | **0.62** | **0.63** | **6.30** |
| LSTM | 0.02 | 0.00 | 0.02 | 16.05 |
| Studnet-A | 0.02 | 0.01 | 0.01 | 15.87 |
| Studnet-B | 0.38 | 0.14 | 0.12 | 13.90 |
| RC-Vehicles | | | | |
| **InstructioNet** | **0.59** | **0.51** | **0.53** | **43.36** |

Table 6: InstructioNet compared against the LSTM and Studnet baselines. See Section 4.4 for details on these metrics. Note that these methods were not trained on exactly the same data, and are therefore not directly comparable. See 5.6 for details on the direct comparability of these methods.

In addition to the average F1 scores shown in Table 6, we also show a histogram of the distribution of F1 scores for the InstructioNet model on the RC-Vehicles dataset in Figure 23. This data shows that around five percent of the examples are reconstructed perfectly according to the $F1_a$ score, while over 20 percent achieve an $F1_a$ score

of over 0.8. As shown in Figure 24, a score of 0.8 corresponds to only a few small mistakes.

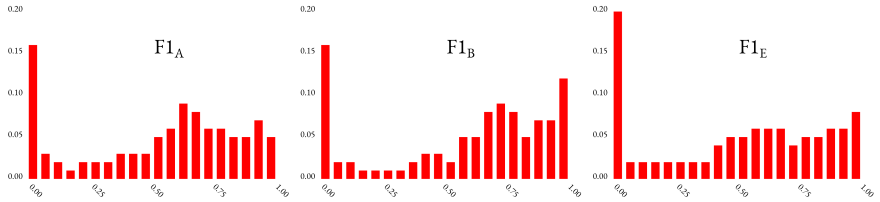


Figure 23: Histogram of InstructioNet F1 scores on the RC Vehicles Data.

### 5.6.2 *Online Training*

In order to show the effectiveness of online training using sequences of actions and observations generated by the learning model, we also train a model on sequences generated only by the expert on the RC-2 and RC-4 datasets. The results are shown in Table 7. While these models make significant progress on these tasks, they consistently underperform relative to the default mixture.

| | $F1_b \uparrow$ | $F1_e \uparrow$ | $F1_a \uparrow$ | AED $\downarrow$ |
|---|---|---|---|---|
| RC-2 | | | | |
| $\boldsymbol{\alpha}$ **0.75** | **0.98** | **0.95** | **0.93** | **0.18** |
| $\alpha = 1.0$ | 0.97 | 0.93 | 0.90 | 0.29 |
| RC-4 | | | | |
| $\boldsymbol{\alpha}$ **0.75** | **0.80** | **0.69** | **0.71** | **2.39** |
| $\alpha = 1.0$ | 0.77 | 0.68 | 0.66 | 2.88 |

Table 7: InstructioNet trained with the default split of expert-guided and model-guided data ($\alpha = 0.75$) compared with a separate model trained with only expert-guided data ($\alpha = 1.0$). See Section 4.4 for details on metrics.

### 5.6.3 *Loss Functions*

We evaluate the effectiveness of our cursor loss function by comparing it against the binary cross entropy and constant regression methods discussed in Section 5.5.3. We find that even on the two-brick models, the summed-probability loss outperforms these other tech-

niques. The results are shown in Table 8. Note that due to the difference in magnitude between these losses, we adjusted the learning rates in an attempt to achieve the best results possible. We found that both benefited from a higher learning rate of $5 \times 10^{-4}$ rather than the default $5 \times 10^{-5}$ used for the summed-probability loss.

| | $F1_b \uparrow$ | $F1_e \uparrow$ | $F1_a \uparrow$ | AED $\downarrow$ |
|---|---|---|---|---|
| RC-2 | | | | |
| **Summed CE** | **0.98** | **0.95** | **0.93** | **0.18** |
| BCE | 0.91 | 0.72 | 0.72 | 0.92 |
| MSE | 0.91 | 0.50 | 0.66 | 1.00 |

Table 8: InstructioNet trained with the default summed cross-entropy loss compared with Binary Cross Entropy (BCE) and large constant regression (MSE). See Section 4.4 for details on metrics and Section 5.5.3 for loss function details.

### 5.6.4 *Conditional Action Generation*

We also test the importance of sequentially conditioning the action heads on one another as discussed in Section 5.5.1 by training a new model where these conditional connections are cut. This corresponds to cutting the magenta connections coming out of the Action Head, Parameter Head and Click Head in Figure 22. We found that cutting these connections leads to training instability where after a certain point the model loses its ability to effectively use the cursor to connect bricks together. In light of this, we report results after 1.7M frames, right before the instability occurs in addition to the default 2.6M frames after the instability occurs. We also report an evaluation of the default model at 1.7M frames for comparison. Note that even before training became unstable, the model without the conditional connections was significantly underperforming the default model. Table 9 shows these results.

| | $F1_b \uparrow$ | $F1_e \uparrow$ | $F1_a \uparrow$ | AED $\downarrow$ |
|---|---|---|---|---|
| RC-2 | | | | |
| 1.7M | **0.98** | 0.94 | 0.92 | 0.22 |
| **2.6M** | **0.98** | **0.95** | **0.93** | **0.18** |
| Cut 1.7M | 0.97 | 0.89 | 0.84 | 0.43 |
| Cut 2.6M | **0.98** | 0.00 | 0.50 | 1.09 |

Table 9: The default InstructioNet model compared to a version where the conditional connections between the model heads have been cut. See Section 4.4 for details on metrics.

### 5.6.5 *Camera Motion*

We also test the extent to which direct image comparison enables our results. To evaluate this, we retrained our model on the RC-V dataset, but with the camera in each frame rotated about the center of the assembly by $\pm 0.1$ radians, and translated by $\pm 10$ LDU in X, Y and Z. This means that when comparing the current simulator image with the top image of the construction stack, they will be viewed from slightly different viewpoints, and will not be aligned pixel-by-pixel. For reference, we also report numbers for the original model that was trained without camera motion evaluated on data with camera motion. The results of this experiment is shown in Table 10. Clearly these shifts in viewpoint negatively impact performance in a substantial way even when retrained on this data (see row Motion(T)), though the model is still able to achieve some success. We hypothesize that this decrease in performance is due to the difficulty of accurately comparing small offsets at low resolution. The problem becomes much worse when evaluating the model that was not trained with camera motion on data that contains camera motion (see row Motion(Unt)). In this case the model completely fails given that it was only ever trained on neatly aligned images.

### 5.6.6 *Expert Instruction Images*

Given that our model only requires the current simulator image, the top image of the instruction stack and the current phase, we also ex-

| | $F1_b \uparrow$ | $F1_e \uparrow$ | $F1_a \uparrow$ | AED $\downarrow$ |
|---|---|---|---|---|
| RC-V | | | | |
| **No Motion** | **0.59** | **0.51** | **0.53** | **46.36** |
| Motion (Trained) | 0.45 | 0.29 | 0.33 | 56.37 |
| Motion (Untrained) | 0.00 | 0.00 | 0.00 | 78.75 |

Table 10: InstructioNet with the default training approach (No Motion) compared against a model trained under small camera motion (Motion(Trained)) and the model trained on no motion, but evaluated under camera motion (Motion (Untrained)).

plore the success of the model in a setting where we use the expert to generate the instruction stack during the break phase, but then do assembly with a learned agent using the expert's instruction stack as input. In this case, the task is more similar to a standard construction task where the agent is given a book of instructions up front. This is similar to building IKEA furniture from paper instructions. Note that the models used here are not retrained, but use the same checkpoints from our main results. The improvement in performance when using the expert instructions aligns with the fact that the model sometimes fails to complete the Break phase on its own. Having access to the expert instructions and starting each episode in the Make phase removes these failure cases. Unsurprising the gap shrinks for smaller models where the overall performance is higher. While it is theoretically possible that the distribution of expert instructions differs from that which is typically produced by the agent, to the extent that this distribution shift exists, it does not appear to be hindering the results here. The result of this experiment is shown in Table 11

### 5.6.7 *Hyperparameters*

Unless otherwise mentioned, we used AdamW with a learning rate of $5 \times 10^{-5}$, $\beta_1 = 0.9$, $\beta_2 = 0.95$ and weight decay of 0.1. For RC-8 and RC-V, the learning rate was cut to $1 \times 10^{-5}$ after 15.8M frames. Table 12 shows the training hyperparameters for Algorithm 1 used to train the models for each dataset.

| | $F1_b \uparrow$ | $F1_e \uparrow$ | $F1_a \uparrow$ | AED ↓ |
|---|---|---|---|---|
| RC-2 | | | | |
| Model | 0.98 | 0.95 | 0.93 | 0.18 |
| Expert | **0.99** | **0.96** | **0.96** | **0.17** |
| RC-4 | | | | |
| Model | 0.80 | 0.69 | 0.71 | 2.39 |
| **Expert** | **0.86** | **0.77** | **0.76** | **1.94** |
| RC-8 | | | | |
| Model | 0.68 | 0.62 | 0.63 | 6.30 |
| **Expert** | **0.83** | **0.75** | **0.75** | **4.39** |
| RC-V | | | | |
| Model | 0.59 | 0.51 | 0.53 | 46.36 |
| **Expert** | **0.71** | **0.63** | **0.65** | **33.81** |

Table 11: InstructioNet evaluated on both Break and Make, and on Make alone using instructions generated by the expert.

### 5.6.8 *Qualitative Evaluation*

Figure 24 shows twelve representative failure and success cases of our model on the RC-Vehicles dataset sorted by their F1a score. Example A shows a case where the model fails to complete the Break phase due to a small ornament on top of the car that the model does not realize it needs to remove. In examples B, C, and D the model successfully completes the Break phase, but then struggles to complete the early part of the model. In example E, the model initially placed the wings correctly, but then misclicked as it was placing a later piece and inadvertently moved one of the wings to the wrong location and it was not able to recover. In example F, the model incorrectly built

| | RC-2 | RC-4 | RC-8 | RC-V |
|---|---|---|---|---|
| $N_{steps}$ | 2.6M | 7.9M | 19.7M | 19.7M |
| $N_{steps/epoch}$ | 8K | 8K | 8K | 32K |
| $N_{train/epoch}$ | 16K | 16K | 16K | 65K |
| C | 32K | 32K | 32K | 131K |
| $\alpha$ | 0.75 | 0.75 | 0.75 | 0.75 |

Table 12: Hyperparameters used to train InstructioNet on different datasets.


Figure 24: Examples of reconstructions from our InstructioNet model trained on the RC-Vehicles dataset. The top right overlay shows the target assembly, while Break or Make indicates which phase the sequence ended in. These examples were chosen to present a diverse array of failure and success cases. See Section 5.6.8 for descriptions of these failure cases and Section 4.4 for an explanation of the evaluation metrics.

the front grille and fails to either undo its mistakes or move on. In example G, the model misplaced one brick in the back of the car and was also not able to correct its mistake. Examples H and I show cases where the model almost completely reconstructs the assembly, but gets hung up on the small ornamental details on the roof. Example J

shows a case where the model accidentally inserted two of the same brick with different colors. Finally, examples K and L show an almost perfect reconstructions where one brick in the middle of the model is incorrect (K) or misplaced (L).

## 5.7 CONCLUSION

Interactive scene understanding and construction domains remain a challenging problem for AI agents. We have demonstrated substantially improved performance on the Break and Make problem, using a model with explicit instruction memory. The failure modes of this approach suggest that performance could be improved by working on solutions that avoid getting stuck while disassembling and reassembling. While our approach is successful, it requires an online expert which can provide explicit instructions not only for the inspection and reconstruction process but also for the process of storing and retrieving memory. This limits the utility of this method in real-world settings, where an online expert may not be available. The InstructioNet approach may not be appropriate for problems that do not follow our assumption that assembly can be completed by approximately reversing the disassembly process. Nevertheless the advantage of this method over prior approaches which considered the entire observation history points to the effectiveness of considering only a portion of memory at a time when making decisions.

This approach also shows the value of curating the visual information, and presenting the model with only the data necessary to solve immediate problems. This is in contrast to the sequential model baselines which require the full history of observations at each time step and therefore need to sift through much more data. In addition to the gains from curating the available information, this has important practical computational benefits as well. The InstructioNet model was able to train and evaluate on much larger models than was previously possible precisely because it only accepts two images at a time and does not require simultaneously processing an entire sequence's worth of data.

# Part III

# IMAGES CONTAIN UNCERTAIN INFORMATION

While Part I and II considered cases where an agent received either too much, or not enough information, in Part III we instead deal with uncertainty and the question of what to do when you are unsure of whether or not your agent has enough information to accomplish some objective given the data or expert you are using to train it. To explore this we consider the sequential decision making problem of learning from an expert that has access to more information than the learner. For many problems this extra information will enable the expert to achieve greater long term reward than any policy without this privileged information access. We call these experts "Impossibly Good" because no learning algorithm will be able to reproduce their behavior. However, in these settings it is reasonable to attempt to recover the best policy possible given the agent's restricted access to information. We provide a set of necessary criteria on the expert that will allow a learner to recover the optimal policy in the reduced information space from the expert's advice alone. We also discuss our approach ELF-Distill that can be used in cases where these criteria are not met and environmental rewards must be taken into account. This contains material originally published as "Impossibly Good Experts and How to Follow Them" at ICLR '23[131]

# 6

# IMPOSSIBLY GOOD EXPERTS AND HOW TO FOLLOW THEM

## 6.1 INTRODUCTION

Sequential decision making is one of the most important problems in modern machine learning theory and practice. Reinforcement learning from an environmental reward signal is a powerful but unwieldy tool for attacking these problems. In contrast, imitation learning can be much more sample efficient and empirically easier to train than reinforcement learning, but requires a powerful expert that can either provide an offline dataset of instructions or online supervision. In many practical settings, these experts have access to more information than the learning agent. This can occur when using human demonstrators to train robots that have inferior sensors, or in simulated environments where a synthetic expert uses hidden simulator information to train an agent. In these settings, it is possible that the expert's additional information makes it more powerful than any learning agent that does not have access to the hidden information. We call these experts "Impossibly Good" and show that learning from them using techniques that do not incorporate environmental rewards can cause the agent to drastically under perform the optimal policy in the reduced information space.

For example, consider a simulated robot tasked with retrieving a cell phone in an unknown apartment consisting of multiple rooms. The robot observes the world using a camera and is not given the location of the phone in advance, so it must explore each room in order to find it. Because this is a simulated environment, we can construct an expert that knows not only the location of the phone, but also the exact layout of each room and can compute the shortest path from the robot to the phone. We can then use this expert to construct a

large corpus of training data across any number of apartments and phone locations. While these demonstrations may be optimal according to the expert that knows the phone's location, they crucially do not provide any demonstrations of the exploratory behavior that is necessary for the robot which must rely on its more limited sensors. At test time, the robot may need to explore many empty rooms before finding the one that contains the phone, but the expert has always walked directly to the goal and so it has never shown the robot what to do when encountering an empty room. In this case the expert is impossibly good because on average, it can reach the phone much faster than any agent that does not have access to the map, but must explore each room one by one. While we may be able to learn some important skills from this expert, we are crucially missing demonstrations of other necessary behavior, and so learning from this expert's advice alone may cause the robot to fail.

Our goal in these settings is to find an algorithm that retains the efficiency of imitation learning, while incorporating just enough reward feedback from the environment to achieve success. To address this, we introduce a new technique called ELF-Distillation (**E**xplorer **L**earning from **F**ollower). The key insight of this approach is to train one **follower** policy using the advice of the impossibly good expert alone, and then use the estimated long-term value of this policy to drive exploration of a second **explorer** policy using reward shaping. These two policies are trained jointly so that the explorer policy can be used to inform the distribution of states from which the follower must learn.

In order to study these problems, we have constructed a suite of Minigrid[13] and Vizdoom[144] environments that clearly demonstrate the challenges of learning from impossibly good experts. While these are toy problems, they are quite challenging for many strong baselines and related approaches, and allow us to clearly demonstrate the necessary concepts in a setting that avoids confounding implementation details.

## 6.2 IMPOSSIBLY GOOD EXPERTS

In order to study learning from an expert with more information than the learning agent, we use the framework of Partially Observable Markov Decision Processes (POMDPs), [56], a generalization of Markov Decision Processes (MDPs) [117] to situations where an agent must make decisions using limited observations. The goal in these settings is to sequentially make decisions in discrete time based on feedback from the environment in order to maximize a reward signal. See Chapter 2 for details, and make particular note of the non-Markovian nature of POMDP observations, which requires policies to act over trajectories instead of single observations.

In our setting, a learning agent's policy $\pi_{\mathcal{L}}$ is a differentiable function mapping a trajectory to a normalized distribution over actions. We define $\pi_{\mathcal{L}}(\tau)$ as the agent's action distribution after witnessing the history $\tau$. We also define the model class $\Pi_{\mathcal{L}}$ to be the set of all possible learnable policies. During training we assume access to an expert policy $\pi_{\mathcal{E}}$ which is a non-differentiable function that also maps a trajectory to a normalized distribution over actions.

In order to reason about agents and experts that have different information, we consider a POMDP with two separate observation functions. The first $o_{\mathcal{L}} = O_{\mathcal{L}}(s)$ produces an observation that the agent sees, while the second $o_{\mathcal{E}} = O_{\mathcal{E}}(s)$ produces an observation that the expert sees. This allows us to reason about a trajectory in the underlying state space $\tau_S$ and map it to a corresponding trajectory of observations for the agent $\tau_{\mathcal{L}}$ and a separate trajectory of observations for the expert $\tau_{\mathcal{E}}$.

$$\tau_S = \{(s_1, a_1, r_1) \dots (s_N, a_N, r_N)\}$$

$$\tau_{\mathcal{L}} = \{(O_{\mathcal{L}}(s_1), a_1, r_1) \dots (O_{\mathcal{L}}(s_N), a_N, r_N)\}$$

$$\tau_{\mathcal{E}} = \{(O_{\mathcal{E}}(s_1), a_1, r_1) \dots (O_{\mathcal{E}}(s_N), a_N, r_N)\}$$

The observational trajectory $\tau_{\mathcal{L}}$ represents what the learning agent observes while interacting with the environment, while the state tra-

jectory $\tau_S$ represents the ground truth state of the world which is unknown. For non-deterministic observation functions we use $O_{\mathcal{L}}(o_{\mathcal{L}}|s)$ and $O_{\mathcal{L}}(\tau_{\mathcal{L}}|\tau_S)$ to represent the probability of a state $s$ or state space trajectory $\tau_S$ producing an observation $o_{\mathcal{L}}$ or observation trajectory $\tau_{\mathcal{L}}$.

We assume a model-free setting in which the states, observation functions, transition dynamics and reward dynamics are all unknown to the learning algorithm. Instead during training, the algorithm interacts with the environment, and receives observations and a scalar reward in the form $\{o_{\mathcal{L}}, o_{\mathcal{E}}, r\}$ with the assumption that $o_{\mathcal{L}}$ and $o_{\mathcal{E}}$ were generated from the same unknown state $s$. The learning algorithm also has access to an expert $\pi_{\mathcal{E}}$ that can produce advice in the form of a suggested action $a_{\mathcal{E}} \sim \pi_{\mathcal{E}}(\tau_{\mathcal{E}})$. At test time the agent will not receive the expert's observations $o_{\mathcal{E}}$ or have access to the expert's advice $a_{\mathcal{E}}$, and so it must learn to make decisions using trajectories of $o_{\mathcal{L}}$ alone.

We consider the episodic setting with a set of terminal states $s^{term}$. When a terminal state is reached, the environment informs the agent that the episode has ended using a special observation $o^{term}$ and resets $s$ using an unknown initial state distribution $S^{init}$. We write $\tau_{\mathcal{L}} \sim \pi_{\mathcal{L}}$ to refer to a trajectory generated by initializing according to $S^{init}$ and repeatedly sampling actions from $\pi_{\mathcal{L}}$ until a terminal state is reached. We also write $\rho_{\pi_{\mathcal{L}}}(o_{\mathcal{L}})$ and $\rho_{\pi_{\mathcal{L}}}(\tau_{\mathcal{L}})$ as the probability of observing $o_{\mathcal{L}}$ and $\tau_{\mathcal{L}}$ respectively when acting according to $\pi_{\mathcal{L}}$ from the initial state distribution. We will also use $P_{\pi_{\mathcal{L}}}$ to refer to the set of trajectories $\tau$ that have nonzero probability when acting according to $\pi_{\mathcal{L}}$ from the initial state distribution.

In our setting, the goal of the learning algorithm is to learn a policy $\pi_{\mathcal{L}}(\tau_{\mathcal{L}})$ using the reward signal $r$ and expert advice $a_{\mathcal{E}}$ during training in order to maximize $V^{init}_{\pi_{\mathcal{L}}}$ during test time. An agent-optimal policy $\pi^*_{\mathcal{L}}$ is one that achieves the greatest value when starting from the initial state distribution compared to all other policies in the model class $V^{init}_{\pi^*_{\mathcal{L}}} \geqslant V^{init}_{\pi_{\mathcal{L}}} \; \forall \; \pi_{\mathcal{L}} \in \Pi_{\mathcal{L}}$. Note that while this agent-optimal policy is the best policy achievable with limited information, it may still underperform the expert $\pi_{\mathcal{E}}$ that has more information.

We now have the tools to formally state the impossibly good criterion. We say an expert $\pi_{\mathcal{E}}$ is impossibly good iff

$$V^{init}_{\pi_{\mathcal{E}}} > V^{init}_{\pi^*_{\mathcal{L}}} \tag{1}$$

This means that an expert is impossibly good if in expectation it achieves a greater value from the initial state distribution than the best policy in the model class $\Pi_{\mathcal{L}}$.

## 6.3 HOW NOT TO FOLLOW THEM

In this section we consider learning from an impossibly good expert using **Behavior Cloning** and **DAgger**, two common techniques that use expert demonstrations as a learning signal. We first describe these methods, then show examples designed to demonstrate their success and failure modes when learning from impossibly good experts. We then provide formal criteria for determining when these methods are capable of finding the agent-optimal policy $\pi^*_{\mathcal{L}}$ from demonstrations provided by $\pi_{\mathcal{E}}$. Note that Swamy et al. [118] have also recently provided conditions under which an agent with restricted information may achieve expert level performance from demonstrations. Rather than considering cases where expert performance may be achieved, our goal is instead to give conditions for learning the best policy possible in cases where achieving expert performance is impossible. Finally we show that in some environments, the agent-optimal policy $\pi^*_{\mathcal{L}}$ cannot be recovered from expert advice alone.

**Behavior Cloning** is an off-policy technique in which the expert $\pi_{\mathcal{E}}$ generates a static dataset of trajectories prior to training. The algorithm then iteratively samples batches of state-action transitions from this dataset and trains the learning agent to match the expert's demonstrations. Behavior Cloning is known to suffer from covariate shift [97], a condition in which errors made by the model during test time can quickly take the agent outside of the distribution of states encountered during training. Despite this limitation, Behavior Cloning

enjoys widespread popularity due to its simplicity and ability to perform well on some problems [109].

**DAgger** Ross, Gordon, and Bagnell [97] is an on-policy method designed to overcome the covariate shift of Behavior Cloning. Recently Swamy et al. [118] has shown that on-policy methods such as DAgger can improve an agent's performance when learning from an expert with more information. Instead of generating a static dataset by acting according to the expert $\pi_{\mathcal{E}}$, DAgger iteratively collects batches of data by acting according to the learning agent $\pi_{\mathcal{L}}$ and simultaneously recording the expert's advice in each visited state. After collecting a batch of data, the agent is trained to replicate the expert's advice on all data collected so far. This allows the agent to train on states that may not be visited by the expert, but are necessary to recovering from mistakes made by the learner.

Diagrams of the examples in this section are found in Figure 25. In these environments, the state space, action space and state-action transition function consist of a directed graph of nodes representing the location of the agent. The state space also contains a random variable $X$ that is uniformly sampled from $\{0, 1\}$ at the start of each episode and remains fixed until termination. The expert observes the value of $X$ at the start of the episode, while the agent only observes $X$ after reaching certain nodes in the graph. Transitions between nodes are deterministic, and both expert and agent observe the node they currently occupy. In all examples there are two terminal nodes, one which produces a reward of 1 when $X = 0$ and the other which produces a reward of 1 when $X = 1$. The agent also receives a small reward penalty $-\epsilon$ at each step to encourage faster solutions.

Due to the deterministic transition function and the fact that $X$ is fixed for each episode, we can simplify $\pi_{\mathcal{L}}$ to be a function of the current node and the remembered value of $X \in \{0, 1, ?\}$ rather than an entire trajectory of observations. We also assume the expert $\pi_{\mathcal{E}}$ acts optimally according to its observations.

First consider Example I. Here, the agent sees the value of $X$ as soon as it visits B. This allows the agent-optimal policy $\pi^*_{\mathcal{L}}$ to always know which terminal reward to transition to from B. An agent in this

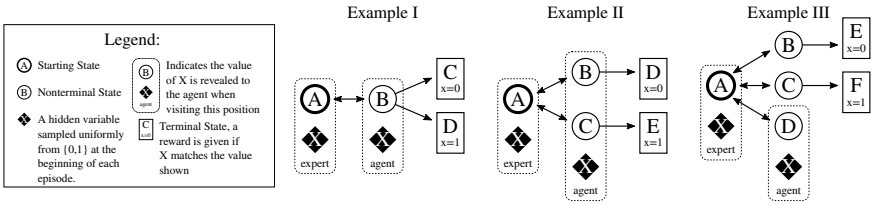


Figure 25: Three example environments featuring information asymmetry between the learning agent and the expert. An agent may travel through states by taking actions as indicated by the arrows. The reward when reaching a terminal state depends on a hidden variable X that is revealed to the agent if it visits the indicated locations.

example must learn distributions for observations $\pi_{\mathcal{L}}(A, ?)$, $\pi_{\mathcal{L}}(B, 0)$ and $\pi_{\mathcal{L}}(B, 1)$. Because the expert $\pi_{\mathcal{E}}$ takes the same path as the agent-optimal policy, it will generate correct training examples for these distributions. Using Behavior Cloning to match these examples will recover $\pi^*_{\mathcal{L}}$.

Example II in Figure 25 shows a case where Behavior Cloning fails. To see this, note that when generating data, the expert will travel from A to B when $X = 0$ and from A to C when $X = 1$. Under the agent's observation function $O_{\mathcal{L}}$, these correspond to $(A, ?)$, $(B, 0)$ and $(C, 1)$. Now consider the agent-optimal behavior in this example. Because of its limited information, the agent must take the first step without knowing the value of X. Sometimes it will reach $(B, 0)$ or $(C, 1)$ which it has seen in the training data, but other times it will reach $(B, 1)$ or $(C, 0)$ which it has never seen. In these cases, the optimal behavior is to backtrack to A where it will encounter either $(A, 0)$ or $(A, 1)$, remembering the value of X. From these observations, the agent now has enough information to guarantee reaching $(B, 0)$ or $(C, 1)$ and achieve the terminal reward. Note though that data generated by acting according to the expert did not provide demonstrations for what to do when observing $(B, 1)$ or $(C, 0)$ so we cannot hope to recover this optimal behavior.

On the other hand, we can show that DAgger can recover $\pi^*_{\mathcal{L}}$ in this example. Now data is generated by the learning agent $\pi_{\mathcal{L}}$, and as we just saw, it has no way to decide whether to transition to B or C at the first step. This means that unlike Behavior Cloning, the observations $(B, 1)$ and $(C, 0)$ will exist in the training set. For these observations,

the optimal behavior according to the expert $\pi_{\mathcal{E}}$ is to backtrack to A and continue down the opposite branch so these labels will be provided along with the corresponding observations. These labels are also optimal according to $\pi^*_{\mathcal{L}}$, so this extra data will allow DAgger to recover the agent-optimal policy.

Finally Example III in Figure 25 shows a case where DAgger will fail to recover $\pi^*_{\mathcal{L}}$. Now the problem is not only data coverage, but also the instructions received by the expert. Similar to Example II, the expert will always tell the learner to transition to B or C from the initial observation $(A, ?)$. However, due to the information asymmetry, the agent-optimal policy $\pi^*_{\mathcal{L}}$ must first visit D in order to discover the value of X before backtracking to A and continuing to B or C as appropriate. Even if some uncertainty in the agent's policy causes it to visit D while generating data, the labels for $(A, ?)$ will never tell the agent that this is the correct behavior.

These examples have given us two intuitive conditions that must be met in order for these methods to recover $\pi^*_{\mathcal{L}}$. The first is that all observations that are encountered when acting according to the agent-optimal policy $\pi^*_{\mathcal{L}}$ must be encountered during training. The second condition is that the expert $\pi_{\mathcal{E}}$ must instruct the agent to take correct actions according to $\pi^*_{\mathcal{L}}$. We now generalize these intuitions into a formal set of conditions that must be met in order for these methods to recover the agent-optimal policy $\pi^*_{\mathcal{L}}$ in a simplified learning model.

Consider a dataset D generated by acting according to an arbitrary generation policy $\pi_D$ and sampling labels $a_{\mathcal{E}}$ from $\pi_{\mathcal{E}}$. Also consider a learning policy $\widetilde{\pi}_{\mathcal{L}}$ that memorizes the empirical distribution of expert recommendations for each trajectory $\tau_{\mathcal{L}}$ in the dataset D.

$$\widetilde{\pi}_{\mathcal{L}}(a|\tau_{\mathcal{L}}) = \frac{\sum_i^{|D|} \mathbb{1}_{\tau_{\mathcal{L}i}=\tau_{\mathcal{L}}, a_{\mathcal{E}i}=a}}{\sum_i^{|D|} \mathbb{1}_{\tau_{\mathcal{L}i}=\tau_{\mathcal{L}}}} \tag{2}$$

**Theorem 1.** *The empirical policy $\widetilde{\pi}_{\mathcal{L}}$ will recover $\pi^*_{\mathcal{L}}$ as $|D| \to \infty$ iff the following conditions hold:*

1. *(Coverage)* $\rho_{\pi_D}(\tau^*_{\mathcal{L}}) \neq 0 \; \forall \; \tau^*_{\mathcal{L}} \in P_{\pi^*_{\mathcal{L}}}$

2. *(Correctness)* $\mathbb{E}_{\tau_S \sim \pi_D} \pi_\mathcal{E}(O_\mathcal{E}(\tau_S)) O_\mathcal{L}(\tau^*_\mathcal{L} | \tau_S) = \pi^*_\mathcal{L}(\tau^*_\mathcal{L}) \; \forall \; \tau^*_\mathcal{L} \in P_{\pi^*_\mathcal{L}}$

The proof is in Appendix D.1.

The first condition states that all policy input trajectories that have nonzero probability of being visited under the agent-optimal policy $\pi^*_\mathcal{L}$ must also have non-zero probability under the dataset sampling policy $\pi_D$. This corresponds to the intuition developed from the examples above that during training we must visit all states that the agent-optimal policy needs to learn about.

The second condition states that the distribution of expert advice for the state trajectories $\tau_S$ which map to agent observation trajectories $\tau^*_\mathcal{L}$ must be equal to $\pi^*_\mathcal{L}(\tau^*_\mathcal{L})$ for all trajectories with nonzero probability of being visited under the agent-optimal policy $\pi^*_\mathcal{L}$. This corresponds to the intuition developed earlier that an expert must tell the agent to act according to $\pi^*_\mathcal{L}$ in order to recover $\pi^*_\mathcal{L}$.

Next we show it may not be possible to learn the agent-optimal policy $\pi^*_\mathcal{L}$ from expert demonstrations alone regardless of the dataset generation policy $\pi_D$.

**Theorem 2.** *There exist environments with impossibly good experts that violate the correctness condition in Theorem 1 regardless of the dataset distribution policy* $\pi_D$.

*Proof.* Example III in Figure 25 is a proof by example.

In this environment $\pi^*_\mathcal{L}(A \rightarrow D | (A, ?)) = 1.0$. However, $\pi_\mathcal{E}(A \rightarrow D | \tau_\mathcal{E}) = 0 \; \forall \; \tau_\mathcal{E}$ so there is no $\pi_D$ that can sample states mapping to $\tau_\mathcal{E}$ that will cause $\pi_\mathcal{E}(\tau_\mathcal{E})$ to produce the labels required to learn $\pi^*_\mathcal{L}$. ■

This result shows that in some cases it is impossible to learn the agent-optimal policy $\pi^*_\mathcal{L}$ from the labels generated by an impossibly good expert alone, regardless of the rollout policy $\pi_D$ that generates the dataset.

## 6.4 HOW TO FOLLOW THEM

In the previous section, we have shown that it is possible to construct an environment with an impossibly good expert such that the learner cannot recover the agent-optimal policy when learning from the expert's advice, regardless of the dataset generating distribution $\pi_D$. Paradoxically, this means we must treat advice from these experts as sub-optimal from the agent's perspective, even though they come from an expert that can achieve higher reward than any policy we can learn. In light of this result, it is natural to ask if we can improve this situation by incorporating environmental reward into our learning algorithm.

The literature on imitation learning [90] and policy distillation [99] provides many useful tools that can be used to learn from sub-optimal experts. We describe these methods using the unifying distillation framework of [18]. In this setting, distillation algorithms consist of two steps. The first step rolls out a set of N state-action-reward transitions $D = \{(\tau_1, a_1, r_1) \dots (\tau_N, a_N, r_N)\}$ using a dataset generation policy $\pi_D$ ($q_\theta$ in Czarnecki et al. [18]). In the second step, the parameters $\theta$ of the learner's policy $\pi_\mathcal{L}$ are then updated in proportion to

$$\mathbb{E}_{\pi_D}\left[\sum_{i=1}^{|D|} -\nabla_\theta \log \pi_\mathcal{L}(a_i|\tau_i)\widehat{R}_t + \nabla_\theta l(\pi_\mathcal{L}(a_i|\tau_i), \pi_\mathcal{E}(a_i|\tau_i))\right] \quad (3)$$

These steps are repeated until performance converges or some other stopping criterion is met. The update rule in Equation 3 can be thought of as mixing a reinforcement learning objective based on a non-differentiable multi-step return $\widehat{R}_t = \sum_{i=t}^{|\tau|} \hat{r}_i$ with a single-step differentiable loss function $l$. The reward term $\hat{r}_i$ can be the environmental reward $r_i$, or any other term we choose that assigns greater value to some behavior that we want to encourage. The loss $l$ is more restrictive because it must be differentiable, but provides a powerful way to inject direct supervision into the training process. By varying our choice of $\pi_D$, $\hat{r}_i$ and $l$ we can arrive at several different distillation algorithms as shown in Table 13. Here the functions $H(\pi_\mathcal{E}, \pi_\mathcal{L})_i$ and $KL(\pi_\mathcal{E}, \pi_\mathcal{L})_i$ re-

spectively refer to the cross entropy and KL divergence between the expert and agent for the current observational history $\tau_i$.

Table 13: Algorithms.

| Algorithm | $\pi_D$ | $l$ | $\widehat{r}_i$ |
|---|---|---|---|
| Policy Gradient | $\pi_{\mathcal{L}}$ | 0 | $r_i$ |
| Teacher Distill | $\pi_{\mathcal{E}}$ | $H(\pi_{\mathcal{E}}, \pi_{\mathcal{L}})_i$ | 0 |
| On-Policy Distill | $\pi_{\mathcal{L}}$ | $H(\pi_{\mathcal{E}}, \pi_{\mathcal{L}})_i$ | 0 |
| On-Policy Distill+R | $\pi_{\mathcal{L}}$ | $H(\pi_{\mathcal{E}}, \pi_{\mathcal{L}})_i$ | $r_i$ |
| N-Distill +R | $\pi_{\mathcal{L}}$ | $H(\pi_{\mathcal{E}}, \pi_{\mathcal{L}})_i$ | $-H(\pi_{\mathcal{E}}, \pi_{\mathcal{L}})_{i+1} + r_i$ |
| Exp. Match. Rew. +R | $\pi_{\mathcal{L}}$ | 0 | $\alpha \mathbb{1}_{a_{\mathcal{E}i}} - \beta \mathbb{1}_{\neg a_{\mathcal{E}i}} + r_i$ |
| ADVISOR | $\pi_{\mathcal{L}}$ | $H(\pi_{\mathcal{E}}, \pi_{\mathcal{L}})_i w_i$ | $r_i(1 - w_i)$ |
| ELF | $\pi_{\mathcal{L}}$ | 0 | $r_i + v_{\mathcal{F}}(\tau_{i+1}) - v_{\mathcal{F}}(\tau_i)$ |

### 6.4.1 *Policy Gradient*

Policy Gradient is a general class of reinforcement learning algorithms [57, 104, 105, 136]. Applying policy gradient methods to our problems essentially throws away the expert advice and learns from environmental rewards alone.

### 6.4.2 *Teacher Distill and On-Policy Distill*

Teacher Distill and On-Policy Distill are the names used by Czarnecki et al. [18] for versions of the Behavior Cloning and DAgger [97] algorithms adapted for the distillation framework where data is continually generated online. These algorithms do not include an $\hat{r}_i$ term, and so will not be able to recover $\pi^*_{\mathcal{L}}$ in all cases.

### 6.4.3 *On-Policy Distill+R*

On-Policy Distill+R combines the single step loss term from On-Policy Distill with the environmental reward used in Policy Gradient. The motivation is that if On-Policy Distill attempts to follow the expert,

and Policy Gradient seeks high environmental reward, then combining them seeks to do both.

### 6.4.4 *N-Distill+R*

N-Distill+R is also from Czarnecki et al. [18] and is similar to COSIL [86], but adapted to the distillation framework. Unlike COSIL, N-Distill+R uses cross entropy rather than the KL Divergence, but these are identical for deterministic experts, which is the setting we assume in our experiments. The other difference is that N-Distill+R uses the cross entropy term from the first time step as a differentiable loss rather than rolling it into the reward signal, which improves performance. This method is also similar to On-Policy Distill+R except that it augments the reward using future agreement with the expert. This encourages the model to not only take actions that the expert immediately agrees with, but also to seek out states in which it is easy to agree with the expert in the future.

### 6.4.5 *Expert Matching Reward+R*

Expert Matching Reward+R removes the single-step loss term and instead assigns a fixed scalar $\alpha$ to the reward $r_i$ when the agent's actions agree with the expert and negative reward $-\beta$ when the actions disagree.

### 6.4.6 *Teacher Distill + PPO*

Teacher Distill + PPO is similar to DAPG [90] and first trains an agent with Teacher Distill for a fixed number of time steps, then refines this policy using reinforcement learning. This is a popular technique used to learn robot behavior from human demonstrations.

All methods mentioned so far that attempt to learn from both the expert advice and the reward signal suffer from a common problem: they all attempt to balance reward seeking and expert following be-

havior uniformly across all sequences of observations, and do not attempt to explicitly discover where following the expert yields high reward and where it does not.

### 6.4.7 *ADVISOR*

ADVISOR from Weihs et al. [135] attempts to address this by dynamically interpolating between imitation and reinforcement learning signals using a weighting factor $w_i = e^{-\alpha \mathrm{KL}(\pi_\mathcal{E}, \pi_{aux})_i}$, where $\pi_{aux}$ is an auxiliary policy trained to follow the expert in every state using the information available to the agent. This weighting factor estimates how closely the agent can follow the expert given the information it has at the current time step. Unfortunately, the ability to follow the expert at a given state is not strictly indicative of states that require exploration. See Example IV in Figure 26 for a demonstration of a failure case for ADVISOR. In this example, nothing prevents the auxiliary policy $\pi_{aux}$ from replicating expert behavior at A, meaning the ADVISOR loss will strongly favor the imitation learning signal at this location which encourages the transition from A to B. Unfortunately, however, the agent-optimal action is to first travel from A to C in order to learn the value of X, then backtrack to A and continue to the goal. Because ADVISOR is able to reproduce the expert's behavior at A, it will ignore the transition to C, and will be unable to gather the necessary information.

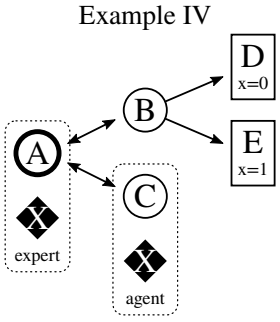


Figure 26: An example environment where ADVISOR cannot recover the agent-optimal policy $\pi^*_\mathcal{L}$.

### 6.4.8 *ELF-Distill*

ELF-Distill is our new technique which overcomes the limitations of the previous methods. Pseudocode is shown in Algorithm 2. The key insight is to train two policies jointly: a *follower* $\pi_{\mathcal{F}}$ which attempts to learn how to follow the expert, and an *explorer* $\pi_{\mathcal{L}}$ that attempts to maximize environmental reward using the follower's value function as reward shaping.

**Algorithm 2** ELF-Distill

**Require:** $N_{steps}$
**Require:** Expert $\pi_{\mathcal{E}}$
**Require:** Horizon $T$
**Require:** Target value $v$
**Require:** Dataset Sizes $N_{D\mathcal{F}}$, $N_{D\mathcal{L}}$ and training Times $N_{T\mathcal{F}}$, $N_{T\mathcal{L}}$
  $\pi_{\mathcal{F}} = \mathrm{InitializePolicy}()$ ▷ Initialize the Follower
  $v_{\mathcal{F}} = \mathrm{InitializeValue}()$ ▷ Initialize Follower's Value Estimate
  $\pi_{\mathcal{L}} = \mathrm{InitializePolicy}()$ ▷ Initialize the Explorer
  **for** $i = 1$ **to** $N_{steps}$ **do**
    $D_{\mathcal{F}} = \{\}$ ▷ Initialize the Follower Dataset
    **for** $j = 1$ **to** $N_{D\mathcal{F}}$ **do** ▷ Collect $N_{D\mathcal{F}}$ trajectories
      $t \sim \{1, \ldots, T\}$ ▷ Choose a random switching Time
      $\tau_{1\ldots t} \sim \pi_{\mathcal{L}}$ ▷ Sample a new trajectory using $\pi_{\mathcal{L}}$ until t...
      $\tau_{t+1\ldots T} \sim \pi_{\mathcal{F}}$ ▷ ...and $\pi_{\mathcal{F}}$ afterward
      $a \sim \pi_{\mathcal{E}}(\tau)$ ▷ Sample expert actions for the trajectory
      $\mathrm{Append}(D_{\mathcal{F}}, (\tau, a, t))$ ▷ Add trajectory to $D_{\mathcal{F}}$
    **end for**
    **for** $j = 1$ **to** $N_{T\mathcal{F}}$ **do** ▷ Train $\pi_{\mathcal{F}}$ for $N_{T\mathcal{F}}$ steps
      $\tau, a, t \sim D_{\mathcal{F}}$ ▷ Sample a trajectory from $D_{\mathcal{F}}$
      $\mathrm{SupervisePolicy}(\pi_{\mathcal{F}}, \tau, a)$ ▷ Train follower from expert
      $\mathrm{SuperviseValue}(v_{\mathcal{F}}, \tau, t)$ ▷ Train value from rewards after t
    **end for**
    Initialize $D_{\mathcal{L}} = \{\}$. ▷ Initialize the Explorer Dataset
    **for** $j = 1$ **to** $N_{D\mathcal{L}}$ **do** ▷ Collect $N_{D\mathcal{L}}$ trajectories
      $\tau \sim \pi_{\mathcal{L}}$ ▷ Sample a new trajectory using $\pi_{\mathcal{L}}$
      $a \sim \pi_{\mathcal{E}}(\tau)$ ▷ Sample expert actions for the trajectory
      $\hat{r}_t = r_{\tau t} + v_{\mathcal{F}}(\tau_{t+1}) - v_{\mathcal{F}}(\tau_t) \forall t \in \{1 \ldots T\}$ ▷ Reshape rewards
      $\mathrm{Append}(D_{\mathcal{L}}, (\tau, a, \hat{r}))$ ▷ Add trajectory to the dataset
    **end for**
    **for** $j = 1$ **to** $N_{T\mathcal{F}}$ **do** ▷ Train $\pi_{\mathcal{L}}$ for $N_{T\mathcal{L}}$ steps
      $\tau, a, \hat{r} \sim D_{\mathcal{L}}$ ▷ Sample a trajectory from $D_{\mathcal{L}}$
      $\mathrm{ReinforcePolicy}(\pi_{\mathcal{L}}, \tau, \hat{r})$ where $v_{\mathcal{F}}(\tau) < v$
      $\mathrm{SupervisePolicy}(\pi_{\mathcal{L}}, \tau, a)$ where $v_{\mathcal{F}}(\tau) >= v$
    **end for**
  **end for**

The follower is trained using a distillation that samples data according to a switching policy that rolls out experience according to the explorer $\pi_{\mathcal{L}}$ for a random number of time steps, then allows the follower $\pi_{\mathcal{F}}$ to take over. This rollout behavior is also used in AggreVaTe [94] and allows us to learn to follow the expert from states visited by the explorer that would not normally be visited by the follower. We also train a value function $v_{\mathcal{F}}$ on the second half of the trajectory that is rolled out according to the follower. The follower's distillation uses cross entropy to the expert's advice as the single-step loss function $l$ with no reward term.

The explorer is trained using policy gradients with reward shaping that encourages the explorer to visit states with high value according to the follower's learned value function. The reward term $\hat{r}_i = r_i + v_{\mathcal{F}}(\tau_{i+1}) - v_{\mathcal{F}}(\tau_i)$ uses the follower's learned value function $v_{\mathcal{F}}$ as potential-based reward shaping [85]. This is referred to as Teacher V Reward in Czarnecki et al. [18] and is related to Sun, Bagnell, and Boots [116]. The single-step loss function is 0 unless the follower's value function $v_{\mathcal{F}}(\tau)$ is greater than a target value $v$ in which case it uses cross-entropy to the follower's distribution. This accelerates learning in states where the follower has already learned to reach high returns.

Training proceeds in an alternating fashion, first training the follower on new data generated by the switching policy, then training the explorer on new data generated using the explorer (on-policy). It is necessary to keep training the follower because the explorer may reach states in the middle of training that would not be visited otherwise, and it is important to build accurate value estimates for them as well.

At test time, the follower is no longer necessary and can be discarded. The only purpose of the follower is to discover where the expert is reliable and where it isn't in order to overcome the limitation of trying to find good global mixing rules for reward seeking and expert following behavior. In this way the follower policy is similar to the auxiliary policy in ADVISOR, but it is used by the algorithm in a much different way. Rather than guiding exploration using the

follower's ability to replicate the expert in a given state, we instead use the follower's value estimate, which encourages the learner to visit states where following the expert leads to high long-term reward. This avoids important failure cases such as the one shown in Section 6.4.7.

## 6.5 MINIGRID ENVIRONMENTS AND TRAINING DETAILS

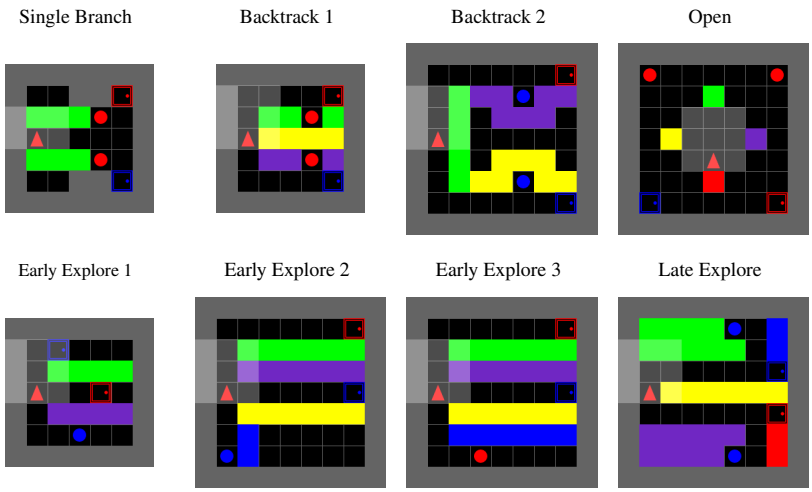


Figure 27: Minigrid Maps

### 6.5.1 *Environment*

The goal of the minigrid environments is to reach the door that matches the color of one or more balls placed in the scene. At the beginning of each episode all balls in the scene set to be either blue or red, and they remain that color for the entire episode. There is one door per color and it's location is fixed for all time. The expert knows in advance the color of the balls and provides the agent with direct supervision to reach the correct door, but does not consider the agent's ignorance of the ball color. Without taking further exploratory action to find the balls first, the agent cannot determine which door to go to and will always underperform compared to the expert. In order to succeed, the agent must deviate from the instructions provided by the expert in order to gather the information necessary to complete the task.

The agent gets a reward of $1 - 0.01|\tau|$ for entering the correct door for an episode. During training the expert demonstrations are provided by computing a shortest line path from the agent to the correct door without bothering to visit locations where the balls are located.

Visibility is strictly limited and the agent can only view the $3 \times 3$ tiles directly in front of them. While these environments are small, they are quite challenging due to the sparse reward and the need to automatically learn the color-matching behavior.

### 6.5.2 *Model*

All minigrid methods train the same small model that takes a 3x3 grid with two channels representing object type (wall, door, etc) and a single color. The model automatically remembers the color of balls that it has seen in the past. The field of view of the agent is a 3x3 grid of tiles. The input to the network is provided as integer indices, so we use three embedding layers to construct a 3x3 16-channel representation of the observed object type, another 3x3 16-channel representation of the observed object color and a 1x1 16-channel representation of the remembered ball color (a third "grey" color is used when the ball has not been observed yet). The feature for the remembered ball color is tiled to 3x3 and these features are added together and flattened. The model then uses two fully connected layers with 256 channels each and ReLU activations. A policy head consists of another 256-channel linear layer followed by a Tanh activation and a linear projection to the number of actions 7. A value head is the same except it projects to a single value. ELF distill trains two policies, so it has one network for each. ADVISOR has an additional auxilliary head.

### 6.5.3 *Training*

Each method was trained on $2^{20} \approx 1$M frames. When training ELF-Distill, both the follower and explorer were trained on $2^{19} \approx 500$K frames, so that the total frames observed during training was equal

to the other methods. All methods were trained with 10 different random seeds in each environment. PPO [105] was used as the loss function to maximize reward in all algorithms with a $\widehat{r}_i$ term.

## 6.6 VIZDOOM ENVIRONMENTS AND TRAINING DETAILS

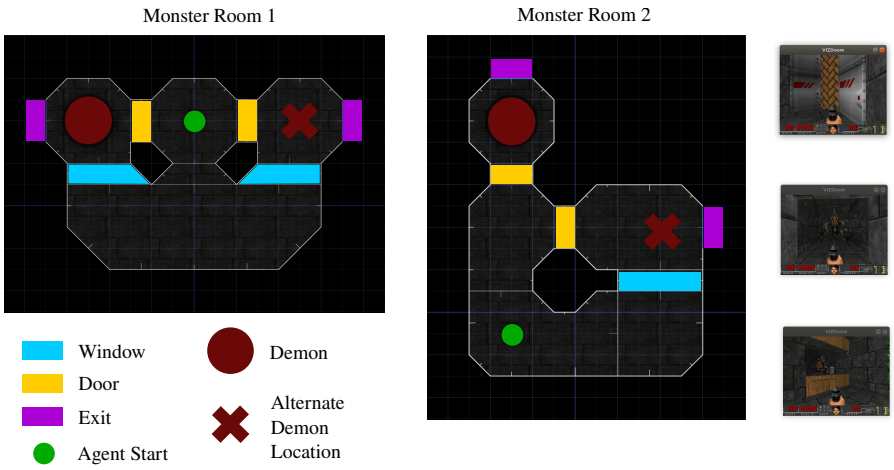


Figure 28: Vizdoom Maps

### 6.6.1 *Environments*

Our two VizDoom [144] were built with the free SLADE Doom editor. The goal of these environments is to escape the room without getting destroyed by a cyborg demon. The agent must act using first-person visual data, and memory of the past. The native resolution of the environment is $320 \times 240$ RGB pixels which we rescale to 84x84 grayscale channels. The game engine runs at a very high frequency, so we use a frame skip of 8 to avoid making decisions at too fine of a granularity. The agent has access to four actions: WALK FORWARD, TURN LEFT, TURN RIGHT and USE which both opens doors and pushes the switch to end the level. The damage settings have been tuned so that a single hit from the monster guarantees instant death. Although the agent can be seen carying a pistol, it is purely decorative, and the agent has no ability to fight back against the monster.

The agent gets a reward of 2 for completing the level successfully, a reward of -2 for getting blown up, and a reward of 0 if neither happens before the maximum number of 72 frames. We also give the agent an exploration bonus based on how far it moves away from the nearest location it has been before, and a small negative reward of -0.001 at each time step. We also use early-termination, stopping an episode if the agent tries to walk forward, but makes no progress (hitting forward while pointing directly at a wall), switches between looking left and looking right three times in a row, or if the agent pushes the USE button twice in a row when not in front of a door or switch. These help avoid long trajectories of meaningless behavior in early episodes. If any of these early termination conditions are triggered, the episode ends with a small penalty of -0.05 for the agent.

### 6.6.2 *Training*

Each method was trained on $2^{21} \approx 2M$ frames. As with Minigrid, we train each component of ELF-Distill with half that number so that the total number of training frames are comparable. All methods were trained with 3 different random seeds in each environment. As with Minigrid, PPO was used as the loss function to maximize reward in all algorithms with a $\hat{r}_i$ term.

## 6.7 EXPERIMENTS

In order to evaluate ELF-Distill, we compare it against the baselines in Section 6.4 on several challenging Minigrid [13] and Vizdoom [144] environments with partial information. Figure 29 shows the results with diagrams of the environments.

The goal of the Minigrid environments is to reach a door that has the same color as a set of balls hidden in various locations in the environment. Observations are provided as a restricted top-down view of the gridworld. In this setting memory is handled for the agent by au-

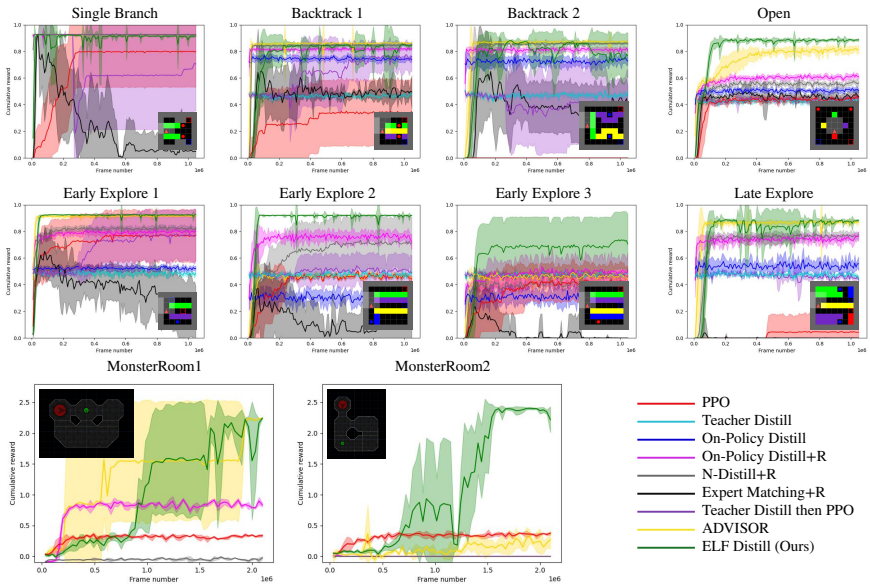


Figure 29: ELF Distill compared against seven baselines on eight Minigrid and two Vizdoom environments designed to require various levels of deviation from an impossibly good expert's advice. The inset image of each plot shows a diagram of the environment.

tomatically remembering the color of the last ball seen and presenting as an additional observation variable. See Section 6.5 for details.

The goal of the Vizdoom environments is to navigate safely to an exit point without getting blown up by a cyborg demon. Each map has two exits, one of which is randomly guarded by the monster. The agent must find a window that allows it to see which exit the monster is guarding and take the alternate route. Unlike Minigrid, agents in these Vizdoom environments are provided with a first-person view from the player's perspective, and must remember relevant past observations using a recurrent network. See Section 6.6 for details.

From the experiments shown here, we can see Elf-Distill performing comparably or better than all other baselines. Note that ADVISOR also performs quite well, except for a few environments (Early Explore 2, Early Explore 3, Monster Room 2) where the issues pointed out in Section 6.4.7 severely impact performance.

## 6.8 LIMITATIONS

Because the explorer policy relies on value estimates from the follower policy, it is possible for biases in those estimates to cause poor explorer performance. This is especially true early in the training process before the follower's value function has had time to learn to differentiate good and bad states. This can be exacerbated by the fact that the follower's state coverage is determined by the explorer due to the switching policy used to collect training data for the follower. This can lead to situations where the follower underestimates the value of a particular state, which causes the explorer to visit it less often, which then causes the follower to get less training data for that state and therefore fail to recover from its initial error.

This suggests that it may be beneficial to use a training schedule that devotes more samples to the follower early on in order to provide better estimates of the value function, and more samples to the explorer later once this value function is well modelled.

Taken to an extreme, this would suggest training the follower alone for half of the training step budget, then freezing it and training the explorer. Unfortunately, this will not be ideal in many circumstances, as the explorer's policy is used to determine which states are necessary for the follower to learn. However, it may be possible to automatically decide when to allocate training resources to the follower or the explorer if the quality of the follower's value function can be estimated online.

## 6.9 CONCLUSION

Imitation learning remains a powerful tool for sequential decision making problems. Unfortunately when experts have more information than the policies that learn from them, the experts may be impossibly good, and their behavior and performance cannot be replicated by any learning algorithm. We have shown that in these settings, it may still be possible to recover the optimal policy in the limited information space using only the expert's advice, but that this

is not guaranteed as we have shown by providing counter-examples. To address this, we have introduced ELF-Distill, a training method that uses the estimated value function of a policy trained from expert demonstrations to guide exploration for a second reinforcement learning agent. We have shown that this algorithm outperforms several distillation baselines that incorporate both environmental reward and expert demonstrations on a set of challenging Minigrid and Vizdoom environments.

The success of ELF-Distill shows progress in an area where we, as the problem designers, may not be sure if the agent has enough information to follow the supervision we are giving it. Unfortunately this comes at the high price of mixing in reinforcement learning, which is typically much more computationally expensive, and challenging to implement than imitation learning. However, we have also shown that this requirement is unavoidable, and that there are environments in which learning from the advice of an impossibly good expert will always lead to suboptimal behavior.

# 7

# CONCLUSION

Vision is a rich source of information for agents operating in sequential decision making problems, but it does not come cheaply. It is often incredibly noisy and computationally expensive to process. We have shown that using contextual goal information to filter clutter early in the visual processing stream can save precious computational resources and allow an agent to do more with less.

We have also seen that visual information is often incomplete. When working with complex assemblies in the LTRON environment, no single view of the structure is sufficient to fully understand all of its parts. The only way forward with these partial observations is to use them to accumulate information over time to form a complete picture of the task at hand. This requires memory, and we have shown that structuring our memory so that it is tailored to the task at hand can greatly improve performance in these settings.

Finally we have seen that providing supervision in partially observable problems also requires caution and careful analysis of potential differences between the agent and expert's observation spaces. We have shown that when the expert knows more than the agent it's advice can be counterproductive and lead the agent to drastically underperform the best in class model. We have developed ELF-Distill, which combines imitation learning and reinforcement learning to address these issues.

Taken together these findings demonstrate the importance of carefully curating the information available to a learning agent, and training agents to be able to curate this information themselves. In this dissertation, several of the solutions we have proposed for these issues have been tailored to the specific environment that agent was operating in. A general solution to episodic memory, learning what sensory data to keep and how to access it over long-horizon tasks, is

an important open problem that is of crucial importance to long-term autonomy. Solving this general problem would begin to unlock our capability to build agents that operate in changing environments for months or years on end. I hope that this work is a productive step in this direction.

# BIBLIOGRAPHY


[1] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. "Flamingo: a visual language model for few-shot learning." In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 23716–23736.

[2] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian D. Reid, Stephen Gould, and Anton van den Hengel. "Vision-and-Language Navigation: Interpreting visually-grounded navigation instructions in real environments." In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[3] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. "Multiple Object Recognition with Visual Attention." In: *ICLR 2015*. Dec. 2015.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate." In: *ICLR 2015*. Sept. 2015.

[5] Victor Bapst, Alvaro Sanchez-Gonzalez, Carl Doersch, Kimberly Stachenfeld, Pushmeet Kohli, Peter Battaglia, and Jessica Hamrick. "Structured agents for physical construction." In: *International Conference on Machine Learning*. PMLR. 2019, pp. 464–474.

[6] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. "Scheduled sampling for sequence prediction with recurrent neural networks." In: *Advances in Neural Information Processing Systems*. 2015, pp. 1171–1179.

[7] Valts Blukis, Nataly Brukhim, Andrew Bennett, Ross A. Knepper, and Yoav Artzi. "Following High-level Navigation Instruc-

tions on a Simulated Quadcopter with Imitation Learning." In: *Proceedings of the Robotics: Science and Systems Conference*. 2018.

[8] Valts Blukis, Dipendra Misra, Ross A. Knepper, and Yoav Artzi. "Mapping Navigation Instructions to Continuous Control Actions with Position Visitation Prediction." In: *Proceedings of the Conference on Robot Learning*. Zurich, Switzerland, 2018.

[9] Adrian Bulat and Georgios Tzimiropoulos. "Human pose estimation via convolutional part heatmap regression." In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part VII 14*. Springer. 2016, pp. 717–732.

[10] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. "Shapenet: An information-rich 3d model repository." In: *arXiv preprint arXiv:1512.03012* (2015).

[11] Matthew Chang, Theophile Gervet, Mukul Khanna, Sriram Yenamandra, Dhruv Shah, So Yeon Min, Kavit Shah, Chris Paxton, Saurabh Gupta, Dhruv Batra, et al. "GOAT: GO to Any Thing." In: *arXiv preprint arXiv:2311.06430* (2023).

[12] Qiuyu Chen, Marius Memmel, Alex Fang, Aaron Walsman, Dieter Fox, and Abhishek Gupta. "URDFormer: Constructing interactive Realistic Scenes from Real Images via Simulation and Generative Modeling." In: *Towards Generalist Robots: Learning Paradigms for Scalable Skill Acquisition@ CoRL2023*. 2023.

[13] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. *Minimalistic Gridworld Environment for Gymnasium*. https://github.com/Farama-Foundation/MiniGrid. 2018.

[14] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. "On the properties of neural machine translation: Encoder-decoder approaches." In: *arXiv preprint arXiv:1409.1259* (2014).

[15] Howie Choset and Keiji Nagatani. "Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization." In: *IEEE Transactions on robotics and automation* 17.2 (2001), pp. 125–137.

[16] Hyunsoo Chung, Jungtaek Kim, Boris Knyazev, Jinhwi Lee, Graham W Taylor, Jaesik Park, and Minsu Cho. "Brick-by-Brick: Combinatorial construction with deep reinforcement learning." In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 5745–5757.

[17] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. "End-to-end driving via conditional imitation learning." In: *ICRA*. 2017.

[18] Wojciech M Czarnecki, Razvan Pascanu, Simon Osindero, Siddhant Jayakumar, Grzegorz Swirszcz, and Max Jaderberg. "Distilling policy distillation." In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR. 2019, pp. 1331–1340.

[19] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. "Embodied Question Answering." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[20] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. "Embodied question answering." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 1–10.

[21] Matt Deitke, Winson Han, Alvaro Herrasti, Aniruddha Kembhavi, Eric Kolve, Roozbeh Mottaghi, Jordi Salvador, Dustin Schwenk, Eli VanderBilt, Matthew Wallingford, et al. "Robothor: An open simulation-to-real embodied ai platform." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 3164–3174.

[22] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et

al. "An image is worth 16x16 words: Transformers for image recognition at scale." In: *arXiv preprint arXiv:2010.11929* (2020).

[23] Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. "Inversecsg: Automatic conversion of 3d models to csg trees." In: *ACM Transactions on Graphics (TOG)* 37.6 (2018), pp. 1–16.

[24] C. Eppner, J. Sturm, M. Bennewitz, C. Stachniss, and W. Burgard. "Imitation learning with generalized task descriptions." In: *2009 IEEE International Conference on Robotics and Automation*. 2009, pp. 3968–3974. DOI: 10.1109/ROBOT.2009.5152466.

[25] Clemens Eppner, Sebastian Höfer, Rico Jonschkowski, Roberto Martín-Martín, Arne Sieverling, Vincent Wall, and Oliver Brock. "Lessons from the Amazon Picking Challenge: Four Aspects of Building Robotic Systems." In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 2017, pp. 4831–4835.

[26] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. "MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge." In: *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. 2022. URL: https://openreview.net/forum?id=rc8o_j8I8PX.

[27] Ali Farhadi, Ian Endres, Derek Hoiem, and David Forsyth. "Describing objects by their attributes." In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2009, pp. 1778–1785.

[28] Chelsea Finn, Ian Goodfellow, and Sergey Levine. "Unsupervised learning for physical interaction through video prediction." In: *arXiv preprint arXiv:1605.07157* (2016).

[29] Simone Frintrop, Gerriet Backer, and Erich Rome. "Goal-directed Search with a Top-down Modulated Computational Attention System." In: *Proceedings of the 27th DAGM Conference on Pattern*

*Recognition*. PR'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 117–124.

[30] Simone Frintrop, Erich Rome, and Henrik I. Christensen. "Computational Visual Attention Systems and Their Cognitive Foundations: A Survey." In: *ACM Trans. Appl. Percept.* 7.1 (Jan. 2010), 6:1–6:39.

[31] Seyed Kamyar Seyed Ghasemipour, Satoshi Kataoka, Byron David, Daniel Freeman, Shixiang Shane Gu, and Igor Mordatch. "Blocks assemble! learning to assemble with large-scale structured reinforcement learning." In: *International Conference on Machine Learning*. PMLR. 2022, pp. 7435–7469.

[32] Rohit Girdhar and D. Ramanan. "CATER: A diagnostic dataset for Compositional Actions and TEmporal Reasoning." In: *ArXiv* abs/1910.04744 (2020).

[33] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks." In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 2011, pp. 315–323.

[34] Daniel Gordon, Dieter Fox, and Ali Farhadi. "What should i do now? marrying reinforcement learning and symbolic planning." In: *arXiv preprint arXiv:1901.01492* (2019).

[35] Daniel Gordon, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, Dieter Fox, and Ali Farhadi. "IQA: Visual Question Answering in Interactive Environments." In: *Computer Vision and Pattern Recognition*. 2018.

[36] Daniel Gordon, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, Dieter Fox, and Ali Farhadi. "Iqa: Visual question answering in interactive environments." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4089–4098.

[37] Alex Graves, Greg Wayne, and Ivo Danihelka. "Neural turing machines." In: *arXiv preprint arXiv:1410.5401* (2014).

[38] Ankit Gupta, Dieter Fox, Brian Curless, and Michael Cohen. "DuploTrack: A Reatime System for Authoring and Guiding Duplo Model Assembly." In: *Proceedings of the 25th annual ACM symposium adjunct on User interface software and technology*. Cambridge, Massachusetts, USA: ACM, 2012.

[39] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. "Cognitive mapping and planning for visual navigation." In: vol. 3. 2017.

[40] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. "Masked autoencoders are scalable vision learners." In: *arXiv preprint arXiv:2111.06377* (2021).

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[42] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. "RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments." In: *The international journal of Robotics Research* 31.5 (2012), pp. 647–663.

[43] Karl Moritz Hermann et al. *Grounded Language Learning in a Simulated 3D World*. 2017. URL: http://arxiv.org/abs/1706.06551.

[44] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[45] Jiani Huang, Calvin Smith, Osbert Bastani, Rishabh Singh, Aws Albarghouthi, and Mayur Naik. "Generating Programmatic Referring Expressions via Program Synthesis." In: *International Conference on Machine Learning*. PMLR. 2020, pp. 4495–4506.

[46] David H Hubel and Torsten N Wiesel. "Brain mechanisms of vision." In: *Scientific American* 241.3 (1979), pp. 150–163.

[47] Tadanobu Inoue, Giovanni De Magistris, Asim Munawar, Tsuyoshi Yokoya, and Ryuki Tachibana. "Deep reinforcement learning for high precision assembly tasks." In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 819–825.

[48] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning*. Feb. 2015.

[49] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al. "Perceiver io: A general architecture for structured inputs & outputs." In: *arXiv preprint arXiv:2107.14795* (2021).

[50] James Jessiman et al. *LDraw*. http://www.ldraw.org. 2022.

[51] James Jessiman et al. *Open Model Repository*. https://omr.ldraw.org. 2022.

[52] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. "CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning." In: *CVPR*. 2017.

[53] Benjamin Jones, Dalton Hildreth, Duowen Chen, Ilya Baran, Vladimir G Kim, and Adriana Schulz. "Automate: A dataset and learning approach for automatic mating of cad assemblies." In: *ACM Transactions on Graphics (TOG)* 40.6 (2021), pp. 1–18.

[54] R Kenny Jones, Theresa Barton, Xianghao Xu, Kai Wang, Ellen Jiang, Paul Guerrero, Niloy J Mitra, and Daniel Ritchie. "Shapeassembly: Learning to generate programs for 3d shape structure synthesis." In: *ACM Transactions on Graphics (TOG)* 39.6 (2020), pp. 1–20.

[55] Michael I Jordan. "Serial order: A parallel distributed processing approach." In: *Advances in psychology*. Vol. 121. Elsevier, 1997, pp. 471–495.

[56] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. "Planning and acting in partially observable stochastic domains." In: *Artificial intelligence* 101.1-2 (1998), pp. 99–134.

[57] Sham M Kakade. "A natural policy gradient." In: *Advances in neural information processing systems* 14 (2001).

[58] J. Kim. "Survey on Automated LEGO Assembly Construction." In: 2015.

[59] Diederik P Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In: *3rd International Conference for Learning Representations*. 2015.

[60] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. "Ai2-thor: An interactive 3d environment for visual ai." In: *arXiv preprint arXiv:1712.05474* (2017).

[61] Ranjay Krishna et al. "Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations." In: 2016. URL: https://arxiv.org/abs/1602.07332.

[62] Sangyeop Lee, Jinhyun Kim, Jae Woo Kim, and Byung-Ro Moon. "Finding an optimal LEGO brick layout of voxelized 3D object using a genetic algorithm." In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. 2015, pp. 1215–1222.

[63] Youngwoon Lee, Edward S Hu, and Joseph J Lim. "IKEA furniture assembly environment for long-horizon complex manipulation tasks." In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 6343–6349.

[64] Kyle Lennon, Katharina Fransen, Alexander O'Brien, Yumeng Cao, Matthew Beveridge, Yamin Arefeen, Nikhil Singh, and

Iddo Drori. "Image2Lego: Customized LEGO Set Generation from Images." In: *arXiv preprint arXiv:2108.08477* (2021).

[65] John J Leonard and Hugh F Durrant-Whyte. "Simultaneous map building and localization for an autonomous mobile robot." In: *IROS*. Vol. 3. 1991, pp. 1442–1447.

[66] Sergey Levin, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. "End-to-end training of deep visuomotor policies." In: *Journal of Machine Learning Research*. 2017.

[67] Changjian Li, Hao Pan, Adrien Bousseau, and Niloy J Mitra. "Sketch2cad: Sequential cad modeling by sketching in context." In: *ACM Transactions on Graphics (TOG)* 39.6 (2020), pp. 1–14.

[68] Kejie Li, Jia-Wang Bian, Robert Castle, Philip HS Torr, and Victor Adrian Prisacariu. "MobileBrick: Building LEGO for 3D Reconstruction on Mobile Devices." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 4892–4901.

[69] Yichen Li, Kaichun Mo, Lin Shao, Minhyuk Sung, and Leonidas Guibas. "Learning 3d part assembly from a single image." In: *European Conference on Computer Vision*. Springer. 2020, pp. 664–682.

[70] Joseph J Lim, Hamed Pirsiavash, and Antonio Torralba. "Parsing ikea objects: Fine pose estimation." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 2992–2999.

[71] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. "Feature pyramid networks for object detection." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2117–2125.

[72] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. "An intriguing failing of convolutional neural networks and the coordconv

solution." In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 2018.

[73] George Markowsky. "s.v. Science, Information Theory, Physiology." In: *Encycolopedia Britannica*. 2023.

[74] Cynthia Matuszek, Dieter Fox, and Karl Koscher. "Following directions using statistical machine translation." In: *Proceedings of the international conference on Human-robot interaction*. 2010.

[75] John McCormac, Ronald Clark, Michael Bloesch, Andrew Davison, and Stefan Leutenegger. "Fusion++: Volumetric object-level slam." In: *2018 international conference on 3D vision (3DV)*. IEEE. 2018, pp. 32–41.

[76] Roland Melkert. *LDCad*. http://www.melkert.net/LDCad. 2017.

[77] Dipendra K Misra, John Langford, and Yoav Artzi. "Mapping Instructions and Visual Observations to Actions with Reinforcement Learning." In: (Apr. 2017).

[78] Dipendra K Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. "Tell Me Dave: Context-Sensitive Grounding of Natural Language to Mobile Manipulation Instructions." In: *Robotics: Science and Systems*. RSS. 2014.

[79] Dipendra Misra, Andrew Bennett, Valts Blukis, Eyvind Niklasson, Max Shatkhin, and Yoav Artzi. "Mapping Instructions to Actions in 3D Environments with Visual Goal Prediction." In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium, 2018.

[80] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. "Recurrent Models of Visual Attention." In: *NIPS*. June 2014.

[81] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing atari with deep reinforcement learning." In: *NIPS Deep Learning Workshop 2013*. 2013.

[82] Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy Mitra, and Leonidas J Guibas. "Structurenet: Hierarchical graph networks for 3d shape generation." In: *arXiv preprint arXiv:1908.00575* (2019).

[83] Kaichun Mo, Shilin Zhu, Angel X Chang, Li Yi, Subarna Tripathi, Leonidas J Guibas, and Hao Su. "Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 909–918.

[84] Chandrakana Nandi, Max Willsey, Adam Anderson, James R Wilcox, Eva Darulova, Dan Grossman, and Zachary Tatlock. "Synthesizing structured CAD models with equality saturation and inverse transformations." In: *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2020, pp. 31–44.

[85] Andrew Y Ng, Daishi Harada, and Stuart Russell. "Policy invariance under reward transformations: Theory and application to reward shaping." In: *Icml*. Vol. 99. 1999, pp. 278–287.

[86] Hai Nguyen, Andrea Baisero, Dian Wang, Christopher Amato, and Robert Platt. "Leveraging Fully Observable Policies for Learning under Partial Observability." In: *arXiv preprint arXiv:2211.01991* (2022).

[87] Chengjie Niu, Jun Li, and Kai Xu. "Im2struct: Recovering 3d shape structure from a single rgb image." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4521–4529.

[88] Maxim Peysakhov and W. Regli. "Using assembly representations to enable evolutionary design of Lego structures." In: *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 17 (2003), pp. 155 –168.

[89] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. "Improving language understanding by generative pre-training." In: (2018).

[90] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations." In: *arXiv preprint arXiv:1709.10087* (2017).

[91] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. "Vision transformers for dense prediction." In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 12179–12188.

[92] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[93] Antoni Rosinol, Arjun Gupta, Marcus Abate, Jingnan Shi, and Luca Carlone. *3D Dynamic Scene Graphs: Actionable Spatial Perception with Places, Objects, and Humans*. 2020. arXiv: 2002.06289 [cs.RO].

[94] Stephane Ross and J Andrew Bagnell. "Reinforcement and imitation learning via interactive no-regret learning." In: *arXiv preprint arXiv:1406.5979* (2014).

[95] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning." In: *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*. Nov. 2011.

[96] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. "A reduction of imitation learning and structured prediction to no-regret online learning." In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 627–635.

[97] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. "A reduction of imitation learning and structured prediction to no-regret online learning." In: *Proceedings of the fourteenth inter-*

*national conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 627–635.

[98] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. *Learning internal representations by error propagation*. 1985.

[99] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. "Policy distillation." In: *arXiv preprint arXiv:1511.06295* (2015).

[100] Renato F Salas-Moreno, Richard A Newcombe, Hauke Strasdat, Paul HJ Kelly, and Andrew J Davison. "Slam++: Simultaneous localisation and mapping at the level of objects." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2013, pp. 1352–1359.

[101] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. "A simple neural network module for relational reasoning." In: *Advances in neural information processing systems*. 2017, pp. 4967–4976.

[102] Thiusius Rajeeth Savarimuthu, Anders Glent Buch, Christian Schlette, Nils Wantia, Jürgen Roßmann, David Martínez, Guillem Alenyà, Carme Torras, Ales Ude, Bojan Nemec, et al. "Teaching a robot the semantics of assembly tasks." In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 48.5 (2017), pp. 670–692.

[103] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. "Habitat: A platform for embodied ai research." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 9339–9347.

[104] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust region policy optimization." In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[105] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms." In: *arXiv preprint arXiv:1707.06347* (2017).

[106] Bokui Shen, Fei Xia, Chengshu Li, Roberto Martín-Martín, Linxi Fan, Guanzhi Wang, Shyamal Buch, Claudia D'Arpino, Sanjana Srivastava, Lyne P Tchapmi, et al. "iGibson, a Simulation Environment for Interactive Tasks in Large RealisticScenes." In: *arXiv preprint arXiv:2012.02924* (2020).

[107] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, R. Mottaghi, Luke Zettlemoyer, and D. Fox. "ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks." In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 10737–10746.

[108] Jasdeep Singh, Vincent Ying, and Alex Nutkiewicz. "Attention on Attention: Architectures for Visual Question Answering (VQA)." In: *arXiv preprint arXiv:1803.07724* (2018).

[109] Jonathan Spencer, Sanjiban Choudhury, Arun Venkatraman, Brian Ziebart, and J. Andrew Bagnell. *Feedback in Imitation Learning: The Three Regimes of Covariate Shift*. 2021. DOI: 10.48550/ARXIV.2102.02872. URL: https://arxiv.org/abs/2102.02872.

[110] S.S. Srinivasa, G. Johnson A.and Lee, M. Koval, S. Choudhury, J. King, C. Dellin, M. Harding, D. Butterworth, P. Velagapudi, and A. Thackston. "A System for Multi-Step Mobile Manipulation: Architecture, Algorithms, and Experiments." In: *International Symposium on Experimental Robotics*. 2016.

[111] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." In: *The Journal of Machine Learning Research* 15 (Jan. 2014), pp. 1929–1958.

[112] Francisco Suárez-Ruiz, Xian Zhou, and Quang-Cuong Pham. "Can robots assemble an IKEA chair?" In: *Science Robotics* 3.17 (2018), eaat6385.

[113] Francisco Suárez-Ruiz, Xian Zhou, and Quang-Cuong Pham. "Can robots assemble an IKEA chair?" In: *Science Robotics* 3.17 (2018), eaat6385.

[114] Edgar Sucar, Kentaro Wada, and Andrew Davison. "NodeSLAM: Neural object descriptors for multi-view shape reconstruction." In: *2020 International Conference on 3D Vision (3DV)*. IEEE. 2020, pp. 949–958.

[115] Zhiqiang Sui, Haonan Chang, Ning Xu, and Odest Chadwicke Jenkins. "Geofusion: geometric consistency informed scene estimation in dense clutter." In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 5913–5920.

[116] Wen Sun, J Andrew Bagnell, and Byron Boots. "Truncated horizon policy search: Combining reinforcement learning & imitation learning." In: *arXiv preprint arXiv:1805.11240* (2018).

[117] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[118] Gokul Swamy, Sanjiban Choudhury, J Andrew Bagnell, and Zhiwei Steven Wu. "Sequence Model Imitation Learning with Unobserved Contexts." In: *arXiv preprint arXiv:2208.02225* (2022).

[119] Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, et al. "Habitat 2.0: Training home assistants to rearrange their habitat." In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 251–266.

[120] Lei Tai, Giuseppe Paolo, and Ming Liu. "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation." In: *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE. 2017, pp. 31–36.

[121] Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R Walter, Ashis Gopal Banerjee, Seth Teller, and Nicholas Roy. "Understanding natural language commands for robotic nav-

igation and mobile manipulation." In: *Proceedings of the National Conference on Artificial Intelligence*. 2011.

[122] Rylee Thompson, Elahe Ghalebi, Terrance DeVries, and Graham W Taylor. "Building lego using deep generative models of graphs." In: *arXiv preprint arXiv:2012.11543* (2020).

[123] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. "Fcos: Fully convolutional one-stage object detection." In: *ICCV*. 2019.

[124] John K. Tsotsos, Scan M. Culhane, Winky Yan Kei Wai, Yuzhong Lai, Neal Davis, and Fernando Nuflo. "Modeling visual attention via selective tuning." In: *Artificial Intelligence* 78.1 (1995), pp. 507 –545.

[125] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention is all you need." In: *arXiv preprint arXiv:1706.03762* (2017).

[126] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." In: *Advances in neural information processing systems* 30 (2017).

[127] Vibhav Vineet, Ondrej Miksik, Morten Lidegaard, Matthias Nießner, Stuart Golodetz, Victor A Prisacariu, Olaf Kähler, David W Murray, Shahram Izadi, Patrick Pérez, et al. "Incremental dense semantic stereo fusion for large-scale semantic scene reconstruction." In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 75–82.

[128] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning." In: *Nature* 575.7782 (2019), pp. 350–354.

[129] Catherine Wah, Steve Branson, Pietro Perona, and Serge Belongie. "Multiclass recognition and part localization with hu-

mans in the loop." In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 2524–2531.

[130] Aaron Walsman, Yonatan Bisk, Saadia Gabriel, Dipendra Misra, Yoav Artzi, Yejin Choi, and Dieter Fox. "Early fusion for goal directed robotic vision." In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 1025–1031.

[131] Aaron Walsman, Muru Zhang, Sanjiban Choudhury, Dieter Fox, and Ali Farhadi. "Impossibly Good Experts and How to Follow Them." In: *The Eleventh International Conference on Learning Representations*. 2022.

[132] Aaron Walsman, Muru Zhang, Klemen Kotar, Karthik Desingh, Ali Farhadi, and Dieter Fox. "Break and make: Interactive structural understanding using lego bricks." In: *European Conference on Computer Vision*. Springer. 2022, pp. 90–107.

[133] Ruocheng Wang, Yunzhi Zhang, Jiayuan Mao, Chin-Yi Cheng, and Jiajun Wu. "Translating a visual lego manual to a machine-executable plan." In: *European Conference on Computer Vision*. Springer. 2022, pp. 677–694.

[134] Saim Wani, Shivansh Patel, Unnat Jain, Angel X. Chang, and Manolis Savva. "Multi-ON: Benchmarking Semantic Map Memory using Multi-Object Navigation." In: *Neural Information Processing Systems (NeurIPS)*. 2020.

[135] Luca Weihs, Unnat Jain, Iou-Jen Liu, Jordi Salvador, Svetlana Lazebnik, Aniruddha Kembhavi, and Alex Schwing. "Bridging the imitation gap by adaptive insubordination." In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 19134–19146.

[136] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." In: *Machine learning* 8.3 (1992), pp. 229–256.

[137] Karl DD Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G Lambourne, Armando Solar-Lezama, and Wojciech

Matusik. "Fusion 360 Gallery: A Dataset and Environment for Programmatic CAD Reconstruction." In: *arXiv preprint arXiv:2010.02392* (2020).

[138] Karl DD Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G Lambourne, Armando Solar-Lezama, and Wojciech Matusik. "Fusion 360 gallery: A dataset and environment for programmatic cad construction from human design sequences." In: *ACM Transactions on Graphics (TOG)* 40.4 (2021), pp. 1–24.

[139] Terry Winograd. "SHRDLU: A System for Dialog." In: (1972).

[140] Jeremy M. Wolfe. "Guided Search 2.0 A revised model of visual search." In: *Psychonomic Bulletin & Review* 1.2 (1994), pp. 202–238.

[141] Jiajun Wu, Joseph J Lim, Hongyi Zhang, Joshua B Tenenbaum, and William T Freeman. "Physics 101: Learning Physical Object Properties from Unlabeled Videos." In: *BMVC*. Vol. 2. 6. 2016, p. 7.

[142] Jiajun Wu, Erika Lu, Pushmeet Kohli, Bill Freeman, and Josh Tenenbaum. "Learning to See Physics via Visual De-animation." In: *NIPS*. 2017, pp. 153–164.

[143] Jiajun Wu, Ilker Yildirim, Joseph J Lim, Bill Freeman, and Josh Tenenbaum. "Galileo: Perceiving physical object properties by integrating a physics engine with deep learning." In: *Advances in neural information processing systems* 28 (2015), pp. 127–135.

[144] Marek Wydmuch, Michał Kempka, and Wojciech Jaśkowski. "ViZDoom Competitions: Playing Doom from Pixels." In: *IEEE Transactions on Games* 11.3 (2019). The 2022 IEEE Transactions on Games Outstanding Paper Award, pp. 248–259. DOI: 10.1109/TG.2018.2877047.

[145] Xianghao Xu, Wenzhe Peng, Chin-Yi Cheng, Karl DD Willis, and Daniel Ritchie. "Inferring cad modeling sequences using zone graphs." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 6062–6070.

[146] Claudia Yan, Dipendra Misra, Andrew Bennett, Aaron Walsman, Yonatan Bisk, and Yoav Artzi. "CHALET: Cornell House Agent Learning Environment." 2018. URL: https://arxiv.org/abs/1801.07357.

[147] Claudia Yan, Dipendra Misra, Andrew Bennnett, Aaron Walsman, Yonatan Bisk, and Yoav Artzi. "Chalet: Cornell house agent learning environment." In: *arXiv preprint arXiv:1801.07357* (2018).

[148] Kexin Yi, Chuang Gan, Yunzhu Li, Pushmeet Kohli, Jiajun Wu, Antonio Torralba, and Joshua B. Tenenbaum. *CLEVRER: CoLlision Events for Video REpresentation and Reasoning*. 2020. arXiv: 1910.01442 [cs.CV].

[149] Guanqi Zhan, Qingnan Fan, Kaichun Mo, Lin Shao, Baoquan Chen, Leonidas J Guibas, Hao Dong, et al. "Generative 3d part assembly via dynamic graph learning." In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 6315–6326.

[150] Jiahao Zhang, Anoop Cherian, Yanbin Liu, Yizhak Ben-Shabat, Cristian Rodriguez, and Stephen Gould. "Aligning Step-by-Step Instructional Diagrams to Video Demonstrations." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 2483–2492.

[151] Xiaoming Zhao, Harsh Agrawal, Dhruv Batra, and Alexander Schwing. "The Surprising Effectiveness of Visual Odometry Techniques for Embodied PointGoal Navigation." In: *Proc. ICCV*. 2021.

[152] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. "Target-driven visual navigation in indoor scenes using deep reinforcement learning." In: *ICRA*. 2017.

# APPENDIX

# A

# EARLY FUSION FOR EFFICIENT DECISION MAKING

## A.1 INFORMATION RETENTION

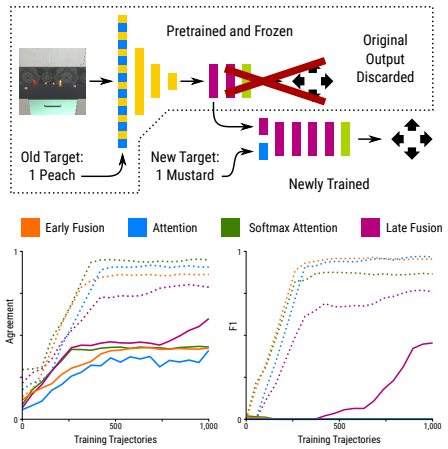


Figure 30: A frozen Early Fusion network with an new trainable branch for collecting an alternate test object. On bottom is the performance of this approach when the old target and the new target are aligned (dotted lines) and performance for when they are different (solid lines). Note that 40% agreement is the majority class baseline as movement is more common than collection.

We have argued above that knowing the request allows the network to discard information about irrelevant objects in the scene. To investigate how much information is retained in the intermediate stages of the network we use the hidden states from models trained on the SIMPLE task and assess whether they can be used to predict the correct action for a new query that is different than the one they were conditioned on. This is implemented by freezing the original model, and training a new set of final layers with a second conditional (Figure 30). In this experiment, we use the Late Fusion model as a proxy for the layer prior to attention in those models.

We find that if the same request is fed to both the original network and the new branch, we achieve performance comparable to the orig-

inal model (dotted lines). On the other hand if mismatched requests are fed into the two branches all models suffer a substantial degradation of performance, with most unable to collect a single object (solid lines). Both Early Fusion and the attention models have completely removed irrelevant information, while Late Fusion approaches appear to only retain much of the irrelevant information.

# B

# BREAK AND MAKE IN LTRON

## B.1 SIMULATOR DETAILS AND PERFORMANCE

The LTRON simulator uses native Python along with the Splendor-Render rendering package that provides a python interface for OpenGL rendering. The speed of the simulator is dependent on the size and complexity of the scene, but for the scenes used in the experiments here, the simulator runs at over 100fps on an Nvidia 2080ti graphics card, and can be parallelized. The simulator also supports headless EGL rendering for use on clusters that do not have graphical sessions. LTRON is not as heavily optimized as some other 3D environments such as Habitat 2.0 [119], but the simulator speed was fast enough not to be a bottleneck during experiments.

## B.2 STATISTICS

We source brick shapes from the LDRAW[50] database, which contains over ten thousand official bricks. However, many bricks in the official parts list are not used in any of the models in the OMR dataset. Thus while LTRON supports over ten thousand individual bricks, only around four thousand are used in the files that we slice to produce our training data.

As one might expect, the distribution of part usage has a long tail of increasingly rare brick shapes. To illustrate this, we plot the log of the individual brick usage counts against the log of the rank of these counts. These diagrams are frequently used in language applications to demonstrate long-tail distributional statistics. Figure 31 illustrates the usage distribution.

In order to measure how common various combinations are we also compute statistics of brick *bigrams* which consist of two bricks that

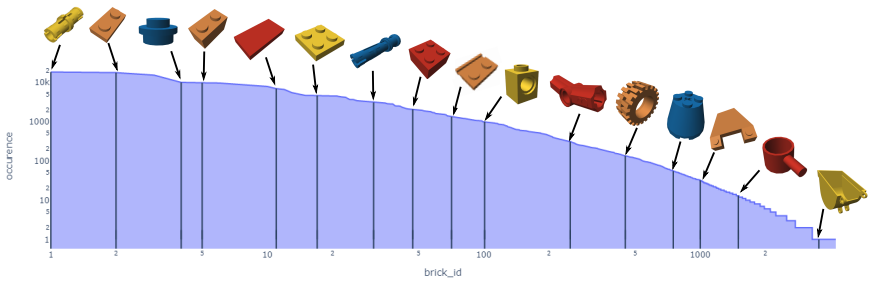


Figure 31: Distribution of brick frequency in the OMR data with examples of various common and rare bricks. The x-axis is the log-rank of each brick shape sorted by frequency with the most common brick on the left and the least common on the right. The y-axis is the log-frequency of each class. The most common brick is a simple connector pin which is used over 18,000 times throughout the dataset. There are around one hundred classes with one thousand or more examples, and over five hundred classes with one hundred or more examples. There is also a long tail of rare bricks, with more than half of all classes occurring less than ten times.

are attached to each other with the same relative transform between them. In total we have 222,679 bigrams where the most common one occurs 1,471 times in the data. The bigrams also have a long tail of increasingly rare combinations as seen in Figure 32.

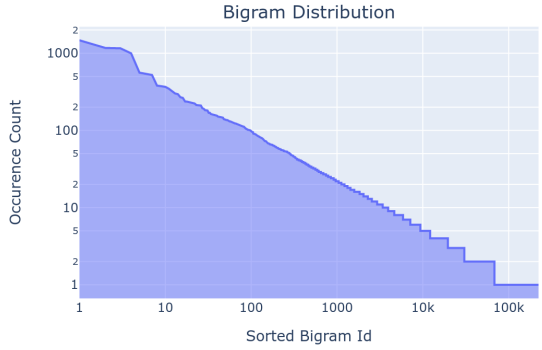


Figure 32: The distribution of brick bigrams–two brick types and an associated transform–in the data included in LTRON.

## B.3 DATA PREPROCESSING

When working with data from the Open Model Repository, we have taken several steps to make the data more manageable for the learning agent.

First we have blacklisted several bricks that are very large that would not fit into the default screen viewport.

Second, many of the brick shapes in the LDRAW repository represent small changes and variations that have been made to different bricks over the years. Where possible, we have collapsed multiple brick shapes that represent only minor variations into a single class by replacing variations with a single canonical version of a part. This further reduced the number of shapes in our dataset from over four thousand to 1790. Figure 33 shows examples of bricks bricks that have been collapsed into a single category.

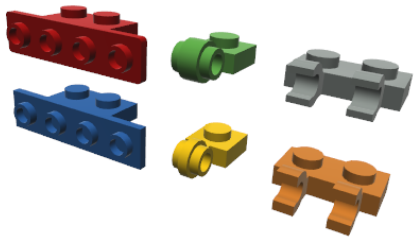

Figure 33: Examples of brick variants in the LDRAW repository. Note the red brick has slightly rounded corner compared to the blue brick. The thick section between the two connection points is slightly thicker in the green brick than the yellow. The shape of the clips is slightly thinner in the grey brick than the orange. As a preprocessing step we replaced each of these categories with a single canonical version when using these bricks for Break and Make.

Third, for each model in the Open Model Repository, we have computed all connected components in the model and split out each as a separate file. These connected components represent groups of bricks that are all attached to each other using the connection points that LTRON currently supports. This is important as many official LEGO sets contain multiple detached components, for example a single racing set may contain two separate cars.

Finally, once we have these connected components, we slice them into smaller components in order to generate many examples of small scenes for training. When slicing we aim to keep the models as small and compact as possible, so we use a simple greedy algorithm that selects the first brick in the scene, then looks at other bricks it is connected to and selects the one that minimizes the largest axis of the bounding box of the new model, and repeats this process until

the desired number of bricks has been selected. These bricks are then broken out as their own new model file and removed from the scene. Then this process is repeated until nothing remains of the original model. This process ensures that the resulting sliced models have similar distributional statistics as the original data, not only in terms of individual brick usage, but also local neighborhood structure.

## B.4 SYMMETRY

Many LEGO bricks exhibit rotational symmetry about a one of the three primary X, Y, Z axes. We have generated a table describing these symmetries by automatically analyzing the shape of each brick in order to take symmetry into account when scoring the final models. In order to compute this table, we render a depth map of each brick from six canonical directions (+/- X,Y,Z). We then rotate each brick by 90, 180 and 270 degrees about each of the X, Y and Z axes and re-render these depth maps. We mark any transformation that produces approximately identical depth maps as a symmetry.

Figure 34: Several examples of models from the Open Model Repository. Each model in the bottom row has over eight hundred bricks, each of which can be individually manipulated within LTRON.

## B.5 ETHICAL RESEARCH AND INTELLECTUAL PROPERTY

The LTRON environment makes substantial use of data collected from the internet. Whenever working with data that comes from external sources, it is important to protect individuals' privacy and respect the intellectual property rights of the original authors.

To our knowledge, LTRON does not pose a substantial privacy risk to individuals. The only personally identifiable information contained in LTRON is a name or pseudonym of the individual author of a particular file from the Open Model Repository. These authors have chosen to make their names public, and have requested attribution through Creative Commons licensing when using their work, so the inclusion of their name is not a privacy concern.

Intellectual Property is an important consideration for this project as each of the scene files included in LTRON represent a substantial investment of time and effort by the original authors. We have therefore only included files that have clear and unambiguous open licensing terms. Fortunately the Open Model Repository contains over a thousand high-quality files with generous Creative Commons licensing. LTRON is inter operable with other user-generated content that can be found scattered throughout various community forums. In most cases these do not contain explicit licensing terms, so we have not included this data for distribution and have not used these files in our experiments. In the future, we may augment LTRON with more data when and if we are able to secure open licensing agreements with individual authors.

LEGO is an official trademark of the LEGO Group which is not affiliated with this dissertation or the authors, and has not endorsed or sponsored this work. To our knowledge, all LEGO material in LTRON has been generated either by the fan community or this paper's authors, and are therefore not under copyright or other intellectual property protection by the LEGO Group.

# C

# SEQUENTIAL PREDICTION MODELS AND INSTRUCTIONET

## C.1 ONLINE EXPERT DETAILS

Here we include details on the logical procedure used to generate expert actions. At each time step, the expert has access to the current assembly $\hat{A}$ and the target assembly $A$. We also keep track of a list of assemblies $\bar{A}$ that correspond to each time an instruction image was pushed onto the instruction stack. Algorithm 3 shows the procedure for generating expert actions. We use the shorthand $|A|$ to refer to the number of bricks in an assembly. The subroutine matching$(A, B)$ computes the single transform that best aligns the assemblies $A$ and $B$ and creates a lookup table between bricks that match under the alignment. The subroutine matching_statistics$(m, A, B)$ takes a matching and a lookup table $m$ and two assemblies and computes a set of true positives $t_p$, false positives $f_p$ and false negatives $f_n$. It also returns the disconnected true positives $d_p$ which are bricks in the estimated assembly that match the shape and color of a brick in the ground truth assembly, but is not in the correct pose and is not connected properly, as well as connected true positives $c_p$ which are bricks that match the shape and color of a brick in the ground truth assembly, but are not in the correct pose, but do have at least one connection point connected correctly. The **REMOVE**, **ROTATE**, and **ASSEMBLE** require that an appropriate connection point is visible for cursor selection. If one is not, **TERMINATE EARLY** is returned instead.

During the Break phase, this expert provides advice that removes one brick at a time, then pushes a new instruction image until no more bricks remain. It then switches to the make phase. During the Make phase, the expert will add a new brick and move it into place until the current assembly matches the assembly that was stored with

**Algorithm 3** Online Expert

**Require:** Current Assembly $\hat{A}$
**Require:** Target Assembly $A$
**Require:** Stack of Instruction Assemblies $\bar{A}$
**Require:** Current Phase $p$
  Compute assembly matching $m, T = \text{match}(\bar{A}_{top}, \hat{A})$
  $t_p, d_p, c_p, f_p, f_n = \text{match\_statistics}(m, \bar{A}_{top}, \hat{A})$
  $m_p = d_p \cup c_p$
  **if** $|f_n| > 1$ **or** $|m_p| > 1$ **or** ($|m_p|$ **and** $|A| \neq |\bar{A}_{top}|$) **then**
    Return **TERMINATE EARLY**
  **end if**
  **if** $p = \text{Break}$ **then**
    $r = |\bar{A}_{top}| - |\hat{A}|$
    **if** $r > 1$ **or** $r < 0$ **then**
      Return **TERMINATE EARLY**
    **else if** $r = 1$ **then**
      Return **PUSH INSTRUCTION**
    **else if** $|\hat{A}| = 0$ **then**
      Return **SWITCH TO MAKE PHASE**
    **end if**
  **else**
    **if** $\hat{A} = \bar{A}_{top}$ **then**
      Return **POP INSTRUCTION**
    **else if** $\hat{A} = A$ **then**
      Return **DONE**
    **end if**
  **end if**
  **if** $|f_p| > 0$ **then**
    Return **REMOVE(**$f_p$**)**
  **else if** $|f_n| > 0$ **then**
    Return **INSERT(**$f_n$**)**
  **else if** $|c_p| > 1$ **then**
    Return **ROTATE(**$c_p$**)**
  **else if** $|d_p| > 1$ **then**
    **if** orientation_correct($d_p$) **then**
      Return **ASSEMBLE(**$d_p$**)**
    **else**
      Return **ROTATE(**$d_p$**)**
    **end if**
  **end if**

the top instruction image, then pop that image off the stack. When rolling out using the learning agent, the expert is capable of recovering from incorrectly inserted bricks (by removing them), incorrectly placed bricks (by moving them). However it is not able to handle situations where the current assembly differs from the assembly corresponding to the top of the instruction stack by two or more bricks.

# D

# IMPOSSIBLY GOOD EXPERTS AND HOW TO FOLLOW THEM

## D.1 PROOF OF THEOREM 1

Recall that the policy $\widetilde{\pi}_{\mathcal{L}}$ is computed using the empirical distribution of a dataset D sampled from a rollout policy $\pi_D$:

$$\widetilde{\pi}_{\mathcal{L}}(a|\tau_{\mathcal{L}}) = \frac{\sum_i^{|D|} \mathbb{1}_{\tau_{\mathcal{L}i}=\tau_{\mathcal{L}}, a_{\mathcal{E}i}=a}}{\sum_i^{|D|} \mathbb{1}_{\tau_{\mathcal{L}i}=\tau_{\mathcal{L}}}} \tag{4}$$

We want to show:

*The empirical policy $\widetilde{\pi}_{\mathcal{L}}$ will recover $\pi^*_{\mathcal{L}}$ as $|D| \to \infty$ iff the following conditions hold:*

1. *(Coverage)* $\rho_{\pi_D}(\tau^*_{\mathcal{L}}) \neq 0 \; \forall \; \tau^*_{\mathcal{L}} \in P_{\pi^*_{\mathcal{L}}}$
2. *(Correctness)* $\mathbb{E}_{\tau_S \sim \pi_D} \pi_{\mathcal{E}}(O_{\mathcal{E}}(\tau_S)) O_{\mathcal{L}}(\tau^*_{\mathcal{L}}|\tau_S) = \pi^*_{\mathcal{L}}(\tau^*_{\mathcal{L}}) \; \forall \; \tau^*_{\mathcal{L}} \in P_{\pi^*_{\mathcal{L}}}$

*Proof.* We first show that the coverage condition is necessary, and then show that the correctness condition is necessary and sufficient when the coverage condition holds.

*The coverage condition is necessary:* If the coverage condition does not hold, for at least some observation histories $\tau_{\mathcal{L}}$, the denominator of Equation 4 will be:

$$\sum_i^{|D|} \mathbb{1}_{\tau_{\mathcal{L}i}=\tau_{\mathcal{L}}} = 0$$

This will result in an undefined distribution for $\widetilde{\pi}_{\mathcal{L}}$ at some trajectories that are encountered while acting according to $\pi^*_{\mathcal{L}}$. Undefined behavior is not optimal according $\pi^*_{\mathcal{L}}$ so this shows that the coverage condition is necessary to recover $\pi^*_{\mathcal{L}}$.

*The correctness condition is necessary and sufficient when the coverage condition holds:* Because the dataset D is sampled according to $\pi_D$, as $|D| \to \infty$, the empirical distribution of expert recommendations in Equation 4 becomes:

$$\widetilde{\pi}_{\mathcal{L}}(\tau_{\mathcal{L}}) = \mathbb{E}_{\tau_S \sim \pi_D} \pi_{\mathcal{E}}(O_{\mathcal{E}}(\tau_S)) O_{\mathcal{L}}(\tau_{\mathcal{L}}|\tau_S) \; \forall \; \tau_{\mathcal{L}} \in P_{\pi_D}$$

If the coverage condition holds, then $P_{\pi^*_{\mathcal{L}}} \subseteq P_{\pi_D}$. This allows us to extend the expression above to $P_{\pi^*_{\mathcal{L}}}$, the set of trajectories with nonzero probability under $\pi^*_{\mathcal{L}}$

$$\widetilde{\pi}_{\mathcal{L}}(\tau_{\mathcal{L}}) = \mathbb{E}_{\tau_S \sim \pi_D} \pi_{\mathcal{E}}(O_{\mathcal{E}}(\tau_S)) O_{\mathcal{L}}(\tau^*_{\mathcal{L}}|\tau_S) \; \forall \; \tau^*_{\mathcal{L}} \in P_{\pi^*_{\mathcal{L}}}$$

This means that if the correctness condition holds:

$$\widetilde{\pi}_{\mathcal{L}}(\tau_{\mathcal{L}}) = \mathbb{E}_{\tau_S \sim \pi_D} \pi_{\mathcal{E}}(O_{\mathcal{E}}(\tau_S)) O_{\mathcal{L}}(\tau^*_{\mathcal{L}}|\tau_S) = \pi^*_{\mathcal{L}}(\tau^*_{\mathcal{L}}) \; \forall \; \tau^*_{\mathcal{L}} \in P_{\pi^*_{\mathcal{L}}}$$

■

This shows that the agent-optimal policy can only be recovered if the distribution over labels generated by the expert $\pi_{\mathcal{E}}$ under the sampling distribution of the state space, match the agent-optimal policy $\pi^*_{\mathcal{L}}$ wherever the states map to $\tau^*_{\mathcal{L}}$.

## D.2 DISCUSSION OF ELF-DISTILL EXPERIMENTS

### D.2.1 *Minigrid*

Our eight MiniGrid Environments belong to five categories. In all environments we colored the walls so that the agents with their very limited field of view could find their way around.

**Single Branch** was designed to mimic Example I in Figure 25. In order to reach any door, the agent must pass by the balls. This is the easiest case, and the only one in which Teacher Distill can do better than 0.5. PPO is often successful here, but often takes longer to reach the goal than methods that use the expert. Here we see a common fail-

ure mode of Expert Matching+R in that the agent learns a state-action loop for which the agent agrees with most of the expert's actions, but disagrees with a few of them. This allows the agent to continuously cycle and gain infinite reward according to it's reshaped reward objective that attempts to maximize agreement with the expert while failing to complete the task.

**Backtrack** are two environments, one larger than the other designed to mimic Example II in Figure 25. This time a ball is placed near each door so that if the agent heads to the wrong one initially, it can backtrack to the other. On-Policy Distill often reaches the correct goal, but does not perform as well as reward seeking approaches due to ambiguous expert advice before the balls have been observed. We also see Teacher Distill then PPO perform well in the smaller version, but it takes a long time to learn. PPO performs much worse on this task. ELF-Distill, ADVISOR, On-Policy Distill+R and N-Distill+R all perform very well on these problems, with ELF-Distill underperforming the others in a few cases.

**Early Explore** are three environments of different sizes designed to mimic Example III in Figure 25. Here the ball is placed on a separate hallway that is not on the path to any door. An algorithm must incorporate environmental rewards to be successful here. Here ELF-Distill clearly dominates all other methods, although the problem becomes more difficult the futher the agent has to explore from states suggested by the expert. ADVISOR performs well on the first environment, but fails at the other two due to the issues pointed out in Section 6.4.7.

**Late Explore** is similar to Backtrack in that the balls are near the doors. However in this case, they are far enough away that they will not be observed by an agent that walks directly to the door. Here ELF-Distill and ADVISOR dominate, with On-Policy Distill+R and N-Distill+R also making progress.

**Open** places the balls and doors in opposite corners of the room with most of the space freely navigable. ELF-Distill outperforms all others, while ADVISOR also makes progress.

### D.2.2 *Vizdoom*

Our Vizdoom environments were designed to provide a more challenging visual scenario for training agents from impossibly good experts.

**Monster Room 1** Here the agent must exit a small room with two doors to enter a large room with windows where the agent can check for the existence of monsters. The agent can then exit the level by going back to the original room and opening the door to the empty room and proceeding to the exit. Only ELF-Distill and ADVISOR are able to reliably complete this task.

**Monster Room 2** In the second Monster Room, the agent starts outside of the room with two doors. Because the expert knows where the monster is, it will always tell the agent to proceed to the room with two doors. This causes problems for ADVISOR, which will latch onto this predictable expert signal without knowing what to do once it has to make a decision about which door to enter. ELF-Distill is the only method to reliably make progress on this task.

## COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

*Final Version* as of December 20, 2023 (`classicthesis` version 4.2).