Compositional Visual Reasoning and Generalization with Neural Networks

Doctoral Dissertation submitted to the
Faculty of Informatics of the Università della Svizzera italiana
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

presented by
Aleksandar Stanić

under the supervision of
Prof. Jürgen Schmidhuber

February 2024

Dissertation Committee

Prof. Cesare Alippi	Università della Svizzera italiana, Switzerland
Prof. Rolf Krause	Università della Svizzera italiana, Switzerland
Prof. Michael Mozer	Univeristy of Colorado, USA
Prof. Ivan Vulić	University of Cambridge, England

Dissertation accepted on 19 February 2024

Research Advisor	PhD Program Director
Prof. Jürgen Schmidhuber	**Prof. Walter Binder/ Prof. Stefan Wolf**

Aleksandar Stanić
Lugano, 19 February 2024

Abstract

Deep neural networks (NNs) recently revolutionized the field of Artificial Intelligence, making great progress in computer vision, natural language processing, complex game play, generative AI, and scientific disciplines. However, these connectionist models are still far from human-level performance in terms of their robustness to domain changes, their ability for compositional reasoning, and combinatorial generalization to unseen data.

In this thesis, we hypothesize that this is a consequence of the binding problem, namely, the inability of NNs to discover discrete representations from raw data and flexibly combine them to solve tasks. We first discuss three promising paths going forward: learning an object- and relation-centric world model, scaling NNs, and employing task decomposition through modular NNs, e.g. by using a large language model (LLM) as a controller. We then present several contributions on learning object representations, as well as on visual reasoning with LLMs as controllers.

Firstly, we propose a novel approach to common-sense relational reasoning by learning to discover objects and relations from raw pixels and modeling their interactions in a parallel rather than sequential manner, which improves prediction and systematic generalization. We then introduce a model that can not only learn objects and relations from raw visual data, but also discover a hierarchical part-whole structure between them. Our approach distinguishes multiple levels of abstraction and improves over strong baselines in modeling synthetic and real-world videos. Since (hierarchical) decomposition into objects is generally task dependent, it is sometimes infeasible and undesirable to decompose a scene into all hierarchy levels. For these reasons, it might be more beneficial to modulate objects with task information, e.g. via actions/goals in a reinforcement learning setting. In this context, we introduce object-centric agents that greatly improve generalization and robustness to unseen data. We then introduce a novel synchrony-based method that, for the first time, is capable of discovering objects in an unsupervised manner in multi-object color datasets and simultaneously representing more than three objects.

Our final contribution is on visual reasoning with LLMs as controllers that has the potential to “sidestep” the binding problem, by decomposing the task into subtasks and then solving the subtasks by orchestrating a set of (visual) tools. We introduce a framework that makes use of spatially and temporally abstract routines and leverages a small number of labeled examples to automatically generate in-context examples, thereby avoiding human-created in-context examples and making the LLMs as controllers setup more robust.

By comparing these models with standard approaches in the literature, we confirm that object-centric approaches are promising for endowing NNs with human-level compositional reasoning and generalization capabilities.

Acknowledgements

Completing this thesis and my PhD would have been impossible without so many great people that helped me along the way. First of all, I would like to thank my advisor Jürgen Schmidhuber for giving me the opportunity to do my PhD in his group, for all the discussions that have fundamentally shaped the way I think about AI, but also the more "philosophical" discussions over our numerous lunches. I will never forget questions such as "Can all cattle of the world fit into that hill across the river?". Thank you also for believing in me and giving me basically unlimited freedom to research the topics in this dissertation.

Special thanks also go to my committee members Cesare Alippi, Rolf Krause, Mike Mozer, and Ivan Vulić. Thank you all for agreeing to serve on my committee and for your valuable feedback on this thesis. In particular, thank you Mike for numerous insightful discussions and for being so supportive whenever I reached out to you. You are a perfect example of how to selflessly help others and a great motivation to do good. I hope to be able to do for others what you have done for me.

Going a bit earlier in my academic "family tree" I would like to thank Helmut Bölcskei, my master thesis advisor at ETH Zurich. It was you who advised me to take the machine learning course early in my studies, because you thought that "this will be a very important field in the future". Thank you especially for your pedagogical approach to research and teaching, for taking a chance on me and even personally guiding me through theory-heavy projects no matter how "lost" I was.

PhD is a very solitary journey, but luckily I had some great companions in my labmates along the way who shared the struggle, the happiness, the frustration, the good, and the bad moments. Thank you to Robert Csordas, Francesco Faccio, Sjoerd van Steenkiste, Imanol Schlag, Louis Kirsch, Anand Gopalakrishnan, Klaus Greff, Aditya Ramesh, Vicent Herrmann, Kazuki Irie, Dylan Ashley, Mikhail Andronov, Miroslav Mirek Strupl, Michael Wand, Krsto Prorokovic, Xingdong Zuo, Paulo Rauber, Filipe Mutz, Julius Kunze, Qinhan Hou and Stefano van Gogh. Thank you all for creating a stimulating research environment and for helping me find moments of joy outside of work in Lugano. Thank you Robert and Francesco for being unforgettable officemates in the early years. We created a unique and fun atmosphere (to say the least), which was often challenging to get some focused work done, but the jokes kept us in good spirits and eased our way through the difficult early years. Thank you, Robert, in particular, for your critical thinking and for being unwilling to accept any idea before you fully understand it. Thank you also for being our internal IT support and, above all, for being a true friend. Thank you Francesco for acting as the group's "event manager", for organizing all the boardgames and the hikes, and for all the entropy you generated with your unusual life stories. Thank you Sjoerd for the collaboration, your critical thinking, and all the help and advice regarding papers, internships, and projects in general. Working with you has definitely sharpened my way of thinking about the scientific process. Thank you Imanol for all the interesting discussions and for generously offering to collaborate on the languini project. Thank you Lous for being an example of where motivated and focused work can bring us to and for categorically rejecting to spend energy on anything that is not constructive and positive. If it were not for you, we would not have been encouraged to apply for internships so early in our PhDs. Thank you Anand for so often being my go-to person, my closest collaborator, and for the friendship that was always fueled by jokes. Your sense of humor and wittiness so often brought light into the day. Thank you Kazuki for the collaboration, for your help and for being the right example of what it means to "do science for the right reasons". Thank you Klaus for all the discussions and help. Even though we overlapped only a few months in our PhDs (and a few more during my visit to Google), the interaction with you probably had the largest influence on my research in these last few years, since it were your papers through which I became familiar with the binding problem. If it weren't for you, this thesis would have looked completely different. Thank you also to Aditya and Vincent, the "young" group members that have profoundly shaped it through their positivity and friendship, especially when it comes to social activities outside of work.

During the course of my PhD I was lucky to have had several opportunities to do internships and to collaborate with amazing people and researchers. First of all, I would like to thank David Ha and Yujin Tang for the collaboration during my student researcher time at Google Brain. Thank you David for taking a chance on me for my first engagement at Google and for your support throughout the project. Thank you Yujin for being such a caring person, for your help with the project, and the discussions we have had since then.

Secondly, I would like to thank many people who I met and worked with

during my internship at DeepMind. Thank you to Alex Lerchner, Loic Matthey, Matko Bosnjak, Jovana Mitrovic, and Andre Saraiva for collaboration. In particular, thank you Alex for the opportunity to join and work in your team. Thank you Loic for being an amazing host. I will forever be grateful for your selfless help, daily discussions about research and technical problems, and for being there for me to even code up some parts when things were getting tough. Thank you Matko for your help on the project, for daily discussions about research and life, and for providing a constant stream of jokes and good humor. One often needs a tough "Balkan" skin for your humor, but you are one of the people with the biggest heart that I know, so everything else becomes irrelevant. Thank you Jovana for your dedication and willingness to collaborate with me. Your way of thinking has definitely shaped mine, especially in terms of prioritizing things in research and the focus. Thank you to the entire Concepts team for making DeepMind feel like home. I would also like to thank many other people that I met at DeepMind for the interesting interactions and discussions that have shaped the way I think: Marin Vlastelica, Nemanja Rakicevic, Michael Chang, Cristina Pinneri, Sami Alabed, Ivana Balazevic, Petar Velickovic, Hamza Merzic, Anastasija Ilic, Daniel Zoran, Rishabh Kabra, Joe Marino, Martin Engelcke, Goker Erdogan, Chris Knutsen, Matt Botvinick, Olivier Henaff, Jean-Baptiste Alayrac, Drew Jaegle, Antoine Miech, Anirudh Goyal, Adam Kosiorek, Geoff Hinton, Antonia Creswell, Murray Shanahan, Andreea-Ioana Deac, Vitaly Kurin, Wilka Carvalho, and Aishwarya Kamath.

Thirdly, I would like to thank my hosts Sergi Caelles and Michael Tschannen for a very fruitful collaboration during my internship at Google. Thank you both for numerous brainstorming sessions, all the help with the infrastructure, advice on the project, and especially on guidance in terms of prioritization when I would become overly optimistic of what can be done in the short time frame. Thank you also to numerous people who welcomed me in the Berlin office: Klaus Greff, Mehdi S. M. Sajjadi, Tomasz Dobrowolski, Étienne Pot, Alexey Dosovitsky, Aravindh Mahendran, Daniel Duckworth, Adam Paszke, Thomas Unterthiner, Klaus-Robert Müller, Danny Dries, and also fellow interns Max Dax, Robert Lange, and Niklas Schmitz. Thank you for all the lunches, board game evenings, research lunches, table foosball matches, and in general for lighting up my Berlin days. Finally, I would also like to thank Sjoerd van Steenkiste, Mike Mozer, Thomas Kipf, Rodrigo Benenson, Jordi Pont-Tuset, Mario Lucic, and Filip Pavetic for interesting discussions and their help.

Throughout the course of this PhD, I was lucky to have had many friends supporting me from the other, more important, nonbusiness side of life. First, I would like to thank my Swiss family Bénédicte Amphimaque, Jana Jankovic,

Masa Davidovic, Ceca Todorovic, Mihajlo Milenkovic, and Slobodan Boricic. Thank you for the numerous dinners, summer and winter trips, and for being such a supportive group in some of the most difficult times I've been through. I can't wait to see what the future holds for us. Thank you to the boys "gang" from my masters studies: Rajko Kisdobranski, Djordje Miladinovic, Berkin Kalay, Baris Atakan and Tanju Gülsever. As Baris wrote in his PhD thesis, with you guys I know I have friends for life. Thank you to my many other friends from Zurich, Lausanne and all over Switzerland: Andjelo Martinovic, Marko Ristin, Jovan Nikolic, Suzan Çelik, Sudarshan Ravi, Amrollah Seifoddini, Nevena Saponjic, Miroslav Veljovic, Irina Subotic, as well as the uniqFEED (alumni) Michele Dell'Ambrogio, Peter Entschev, Ulrich Müller and Niko Stefanoski. Thank you to my childhood friends Pavle Ilic and Vladimir Ivanovic for making me remember those days and act as if nothing changed whenever we are together. Thank you also to my Belgrade friends from bachelor studies, Hana Gostimir, Marija Boljanac, Jovana Vranic, Igor Bajovic, Vladimir Radosevic, Ivan Simeunovic and Nikola Bozic. Finally, I would like to thank all the people who made my stay in Lugano more fun and easier to go through: Camila Alvarez Triviño, Kana Hirayama, Taimoor Tariq, Slobodan Lukovic, Beatrice Granaroli, Marilena Palomba, Alex Bortolotti and Bobby Ventikal.

Special and the greatest thanks go to my family: my parents Mirjana and Dragan Stanic, my sister Sonja and her family Darko, Hana, and Luka Pipic for being a great team, for always being there for me and for being my biggest fans. In particular, I thank my parents for their never-ending unconditional love, the single most important thing that parents could give to their child. Hvala vam svima za sve.

Finally, thank you Béné for lighting up my every day. Merci pour tout.

Contents

Chapter 1

Introduction

Artificial intelligence (AI) has recently made great progress in many areas, such as image recognition [Ciresan et al., 2012; Krizhevsky et al., 2012], natural language processing [Devlin et al., 2018], game playing [Silver et al., 2018], and scientific disciplines [Jumper et al., 2021; Degrave et al., 2022]. In particular, generative models for images [Ramesh et al., 2021; Betker et al., 2023], videos [Villegas et al., 2022], and text [OpenAI, 2022, 2023c; Anil et al., 2023] have made notable breakthroughs. Lately, generative models for text based on large language models (LLMs) have also been combined with visual modality into multimodal vision and language models (VLMs) [OpenAI, 2023d; Team et al., 2023]. Central to this success have been deep neural networks (NNs) [Schmidhuber, 2015] trained on massive amounts of data. However, even the most advanced state-of-the-art (SotA) models struggle in tasks that require *compositional* reasoning, the ability to generalize, fine-grained spatial reasoning, and counting [Bugliarello et al., 2023; Paiss et al., 2023; Hsieh et al., 2023; Yuksekgonul et al., 2022; Zhao et al., 2022; Hendricks and Nematzadeh, 2021].

The notion of compositionality can be tracked as far as Gottlob Frege's work [Frege, 1892a,b, 1923], where the separability of sentences and thoughts into substructures is first noted. However, the first time the term compositionality was used was probably by Katz and Fodor [Katz and Fodor, 1963]. Compositionality has since been studied in fields such as linguistics, logic, neuroscience, and dynamic NNs [Werning et al., 2005; Schmidhuber, 1990a]. A concise formulation of the term "compositionality" was provided by Werning et al. [2005]: *"An interpreted representational system R is compositional if and only if for every complex representation r of R, the meaning of r is determined by the structure of r and the meaning of the constituents of r"*. This is a central theme of the symbolic paradigm, where intelligence results from the manipulation of abstract compositional representations of objects and their relations. There, complex models are decomposed into objects using a compositional representation by using relations. This results in a hierarchy of recursive computations, where we end up with generic atomic parts that can be combinatorially reused to generate novel representations. Here, by combinatorial generalization, we mean "making infinite use of finite means" [von Humboldt et al., 1999; Chomsky, 1969]. In linguistics, for example, the set of primitives is the letters in written language, and the structure is defined by orthographic, syntactic, and grammatical rules. Compositionality is present in vision as well. Biederman [1987] interprets compositionality in vision by introducing shape primitives, called *geons*, which have distinctive properties invariant under different viewpoints and are composed into entities via a hierarchical structure. Object detection is then achieved by analyzing the spatial relationships (the structure) of the individual components.

An example of a compositional task would be to answer the following question: "Could the cookies on the table be evenly distributed among children?". To solve this task, the model needs to detect the cookies in the image, filter out the ones that are not on the table, detect children, count cookies and children, and check if cookies count is divisible by the children count. Questions like these are difficult for current end-to-end VLMs. Scaling VLMs further makes them even more data and compute hungry [Villalobos et al., 2022], which means that scale alone seems unlikely to solve these tasks, especially due to the infinitely long tail of compositional tasks.

On the other hand, humans effortlessly answer such questions: decomposing the question into the respective subquestions (e.g. "how many cookies are there on the table?"), parsing the raw visual input into salient entities (*symbols*), using them to answer subquestions, and then composing those to answer the initial question. Humans can also deal with previously unseen data or situations by making use of analogies and relying on previously learned knowledge, rules, and laws that govern our world. The ability to generalize beyond our experience has long been thought to be based on our ability to think in terms of *symbols*. We seem to be able to extract symbols from raw visual, auditory, and tactile input signals, which are then used to relate unseen concepts and generalize to unseen situations using previously learned knowledge about related concepts. This facilitates compositional reasoning in domains such as language, vision, and planning. This ability to compositionally solve tasks and to generalize seems to be missing in today's NNs. Our hypothesis of why NNs fail in tasks that humans effortlessly solve is that they suffer from the so-called *binding problem* [Von Der Malsburg, 1994; Roskies, 1999; Greff et al., 2020].

We argue that the inability to flexibly bind features hampers both the gener-

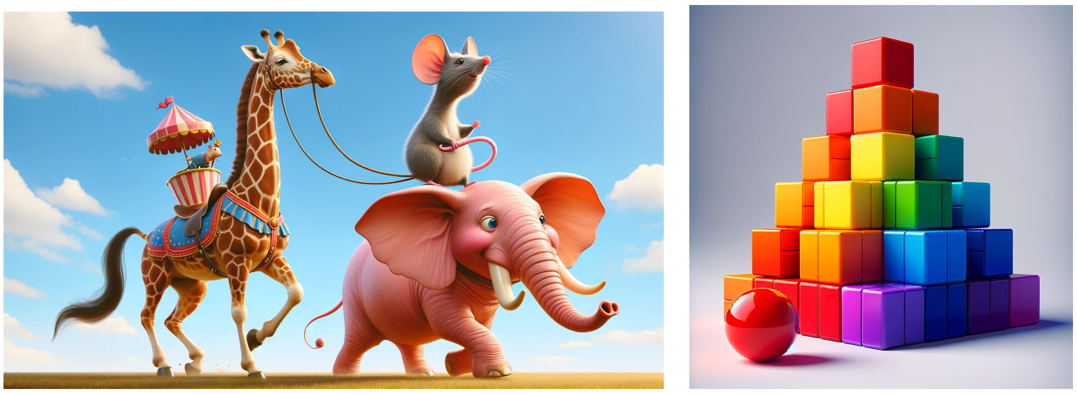

(a) (b)

Figure 1.1: Images generated by OpenAI's DALL-E 3 model [OpenAI, 2023a] for the following prompts: (a) "An image of a giraffe riding a horse. On their left side there is an elephant. A mouse is riding the elephant. The elephant is pink."; (b) "An image of 7 cubes stacked on top of each other in rainbow colors. On the left side of this stack, there is a red sphere and a violet cube".

ative and the recognition models that are based on NNs. For concrete evidence, consider the images in Figure 1.1, generated by OpenAI's DALL-E3 model [OpenAI, 2023a], one of the state-of-the-art image generation models. In Figure 1.1a we can see how the model struggles with complex generalization, binding attributes to features, and relational reasoning between objects. For example, instead of generating a giraffe riding a horse, the network generates a mouse riding a giraffe. We can also see that certain features of the giraffe have horse-like properties (e.g. the tail and the legs). The elephant is correctly generated in pink, but on the right side of the giraffe, instead of the left side, as instructed. In Figure 1.1b we can see how the model struggles with counting and again with relational reasoning. From the instruction, it is intuitively clear that the model should simply generate a stack of 7 cubes. Instead, the model generates a pyramid with 22 cubes, many more than the 7 requested. The model correctly generates a red sphere next to the cubes, but forgets to draw a violet cube. In other works [Wang et al., 2023b], it was found that generative image models handle material, color, and size attributes better than count and shape attributes.

To show that the binding problem is present not only in the generative models but also in the recognition models, we query GPT4-V [OpenAI, 2023d] to answer the following question about Figure 1.1b: "How many cubes are there in the image?". GPT4-V replies: "There are 21 cubes arranged in a pyramid-like structure, with 1 cube at the top, followed by layers with 4, 6, and 10 cubes,

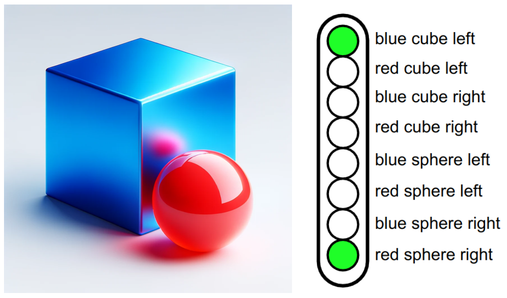


Figure 1.2: A visually simple scene illustrating the difficulty of representing multiple objects simultaneously in neural networks. This naive representational format assumes that individual neurons encode different patterns in the data. This results in the so-called "grandma neurons".

respectively, from the top to the bottom.". Here, we can see that the models also have limited counting and fine-grained scene understanding capabilities. Instead of 22, the model detects 21 cubes. Also, instead of 6 layers, it reports that there are 4 layers, and it further wrongly counts (or hallucinates) the number of cubes in each layer. This further supports our hypothesis that NNs suffer from the binding problem.

1.1 The Binding Problem

The binding problem is a class of problems concerned with how complex patterns in perceptual input are represented in the human brain, more specifically how individual attributes "bind" to a particular object. Visual binding problem [Treisman, 1996; Roskies, 1999] is of great importance for human perception [Spelke, 1990] and cognition [Wertheimer, 1923; Koffka, 1935; Köhler, 1967]. It is fundamental to our visual capabilities to integrate several features together, such as color, shape, texture, etc., into a unified whole [Kahneman et al., 1992]. This problem becomes even more prominent in a multi-object setting when multiple distributed representations need to be simultaneously represented in the network.

Consider a scene containing a blue cube and/or a red sphere, such as the one in Figure 1.2. In the case where only the blue cube is present, the NN representation should contain the "blue" and the "cube" information. Similarly, it would contain the "red" and "sphere" for the scene with only a red sphere. However, if both cube and sphere are present, then the NN representation would contain all attributes: "blue", "red", "cube" and "sphere". The binding problem

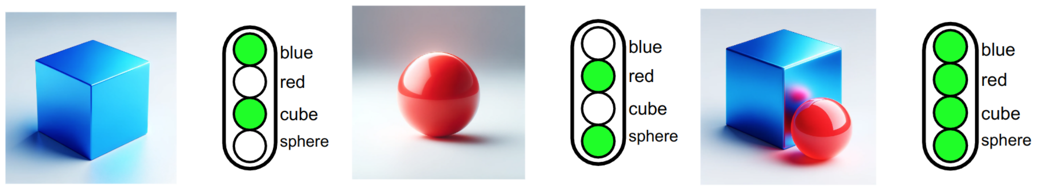


Figure 1.3: An illustration of the binding problem in the fully distributed representational format with disentangled features, also known as the superposition catastrophe. From the representation of the image on the right, it is impossible to know which shape binds to which color.

deals with the question of how to decompose and keep track of all individual attributes simultaneously, but correctly knowing which ones bind to each other.

A naive theory of how the brain could achieve such binding is that individual neurons encode different patterns in the data Figure 1.2, resulting in so-called “grandma neurons”. However, in practice, this would be infeasible, as there would need to be exponentially many neurons to account for the complexity of the real-world patterns, and for each novel situation (e.g. a new object or a known object in a different context), a new neuron would need to be added.

An alternative solution could be the distributed representation as in Figure 1.3, where each concept is represented by a different (group of) neuron(s). The difficulty of this representational format arises when multiple objects are present, as in the image on the right in Figure 1.3. From the encoding of this image, we cannot know which attribute binds to which, for example, if the cube is blue or red.

Finally, the temporal correlation hypothesis [Milner, 1974; Von Der Malsburg, 1994] posits that “the neurons that fire together, wire together” by temporal synchronization of the phase (frequency) of their firing patterns. However, the hypothesis that the brain can achieve binding through the temporal synchrony of firing patterns is not yet unanimously accepted by the research community.

In the following, we describe the particular aspects of the binding problem through the lens of the framework introduced in Greff et al. [2020], which was inspired by Treisman [1999]. Note, however, that we do not always follow closely Greff et al. [2020], but only consider parts which are relevant for our work, and at times there could be differences in our interpretations. Greff et al. [2020] proposes to decompose the analysis of the binding problem into three subproblems: representation, segregation, and composition, visually depicted in Figure 1.4. The representation problem deals with the ways in which the information is represented in NNs such that it is symbolic and compositional.

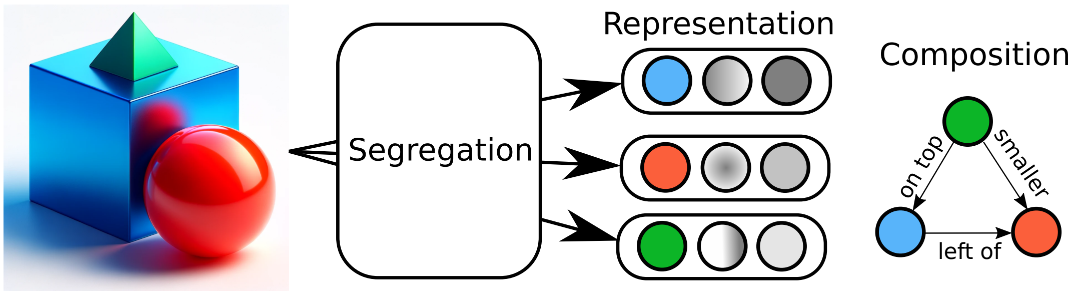


Figure 1.4: The three subproblems of the binding problem according to Greff et al. [2020]: segregation, representation and composition.

The segregation problem is concerned with the process of mapping the raw input to the symbolic (object) representations. The composition problem deals with how representations are used for inferences such as prediction, reasoning, and planning.

Representation. The representation problem deals with how the information that is extracted from the raw visual input is represented in structured, symbol-like entities, called objects. Defining what an object is is a daunting task and would likely necessitate a philosophical debate. For the purposes of discussing the binding problem and generalization in NNs, by an "object" we assume something that can be used as a node in a graph, which can then be used to perform certain inferences. Objects are, in this sense, building blocks that facilitate symbolic functions of NNs. They represent a single whole separate from the rest of the input that can be related or associated with other objects of similar type. From the previous example of "a blue cube and a red sphere" it is obvious that the incorporation of object-centric representations in NNs faces the "superposition catastrophe" [Von Der Malsburg, 1986], suggesting that fully connected NNs suffer from "inherent trade-off between distributed representations and systematic bindings among units of knowledge" [Hummel and Holyoak, 1993].

In terms of the representation format, it should allow objects to be distinguished while retaining the advantages of distributed learned representations. The key properties of this representation are separation, common format, and disentanglement. Separation refers to the fact that the information about individual objects should remain separate at the representation level, i.e., their features should not interfere with one another and should be able to ensure that objects can be formed from novel feature combinations. The common format of object representations is required to be able to compare and relate them (e.g. rela-

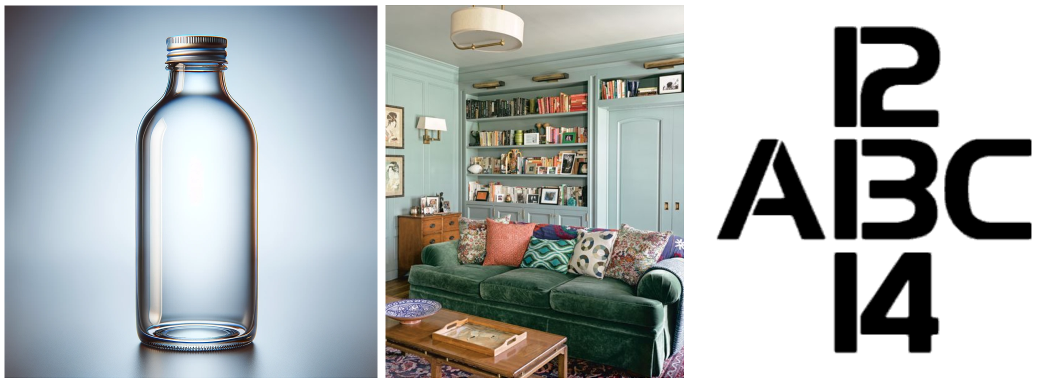


Figure 1.5: Illustration of the difficulty of the segregation problem. What an object is is task- and context-dependent, e.g. the bottle and the cap in the left image as a single object or separated into two objects. Similarly, the meaning of the middle symbol in the image on the right depends on the context in which we read it.

tionships such as "same as", "bigger than", "brighter than", etc.). This means that object representations should be inferred with the same set of parameters (weights) from the raw input. Disentanglement [Schmidhuber, 1991b; Schmidhuber et al., 1996; Bengio et al., 2013] means that the individual attributes in the object representation should be represented separately and allowed to vary independently (e.g. color of an object). This also facilitates representing objects with unseen combinations of attributes and, therefore, robustness and generalization; e.g., though we have never seen a pink elephant, we can easily imagine, draw, and describe it. Apart from the format requirements, Greff et al. [2020] identify temporal dynamics as an important aspect of the representation: object representation should be updated recurrently, taking into account the current input and its previous state. The temporal consistency aspect should intuitively aid in learning the representation (e.g. in the case of an occlusion or temporary absence from the field of view).

Segregation. The segregation problem deals with how the raw input is mapped onto the structured (object-centric) representation. In a way, the representation problem describes *what* properties the object-centric representations should have and the segregation problem *how* to infer them. The difficulty of segregation lies in the often ill-defined notion of what an object is. For example, consider the image on the left in Figure 1.5. By default, e.g. if the task is to move the bottle, you would likely consider it to be one object. However, if you were to drink the water from it, you would open the cap and then you would have two objects. At what point does the cap stop being part of the "bottle" object? Similar can be said for objects in the center image in Figure 1.5. If the task is to move the sofa, then it would be considered an object, together with the pillows on it. If we want to move the pillows, then each pillow would be a separate object. For a more fine-grained task of counting the number of rhomboid patterns on the green pillow, we would separate the pillow into multiple independent rhomboids to count them. Finally, in the image on the right in Figure 1.5 we can see an example of how object meaning is also context dependent. If we consider the middle symbol as part of the horizontal text, we would likely read it as a letter "B". However, if we read it as part of the vertical text, the same symbol would probably now be read as the number "13". Furthermore, even concepts without clear boundaries can be considered an object: a cloud, a lake, a pile of sand, a rainbow, or even a hole! Hopefully, this exemplifies that the notion of what an object is cannot be solved purely through supervision but must be learned mainly in an unsupervised manner and yet be context/task dependent.

Therefore, objects might be easier to define in terms of their defining properties: modularity, hierarchy, and multi-stability through task-/context-dependent modulation. Objects should be modular in the sense that they are self-contained and reusable, independent of the context. Objects are often hierarchical and can be decomposed into parts that can be observed as individual objects (the bottle and its cap). In some way, this also connects to the "task-dependent" definition of an object. Hierarchical decomposition may not be unique; for example, a page of text can be decomposed into lines or sentences [Greff et al., 2020]. This brings us to the important aspect of "multi-stability" that a part of the perceptual input can belong to a number of different object groupings. Moreover, often the number of possible decompositions is so large that it would be impossible and, more importantly, undesirable to keep track of the lowest level of decomposition. Humans solve this by having a multi-stable perception that allows dynamical segregation [Attneave, 1971]. Furthermore, the hierarchical or task-dependent notion of objects may be modulated by top-down feedback from the task at hand. Therefore, the segregation mechanism should be able to modulate the objects by the task information (e.g. a goal, an action, or text), which hints at a reinforcement learning (RL) setup or a weak supervision via image-text pairs. In particular, the weakly supervised case of image and text pairs is related to the symbol grounding problem [Harnad, 1990]. In Harnard's own words, the symbol grounding problem is the problem of how to make the "semantic interpretation of a formal symbol system ... intrinsic to the system, rather than

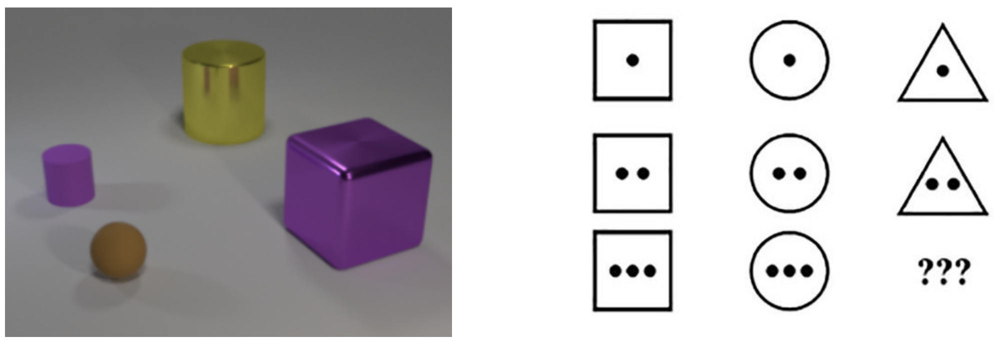


Figure 1.6: Composition facilitates solving relational reasoning tasks such as multi-hop compositional visual question answering (e.g. "Are there any rubber things that have the same size as the yellow metallic cylinder?") or Raven's progressive matrices [Raven and Court, 1938].

just parasitic on the meanings in our heads ... in anything but other meaningless symbols". It deals with how symbols, such as words, are connected to objects or concepts in the real world that they refer to. In this sense, the symbol grounding problem can be seen as a special case of the binding problem. Apart from image-text binding, the binding problem deals with other modalities such as video and audio, but also with how perceptual grouping is accomplished in the purely unimodal case of visual only input.

Composition. The composition problem deals with how segregated objects are connected to each other and used for inference, such as planning and reasoning. In order to combine object representations and relations, we need a variable binding mechanism, similar to having variables that are bound to placeholder symbols in mathematical expressions or particular objects to input arguments in a program function. Composition refers to both the process (function body) and the variable binding mechanism. The compositionality in these mechanisms should mimic similar aspects of human cognition, which allow us to systematically generalize beyond the situations we have encountered.

Examples of tasks where such compositional reasoning is required are presented in Figure 1.6. In the image on the left, the task could be to answer: "Are there any rubber things that have the same size as the yellow metallic cylinder?". Solving this task should be trivial if we decompose the image into individual objects, use them as nodes in the graph, and reason by performing inferences over this graph. For example, one operation that we would use is "compare two objects", an operation that takes two variables as input. The first variable would be the yellow cylinder, and for the second variable, we would perform dynamic binding of all the other objects present in the scene. From this example, we can see how answering such compositional queries should be possible for any combination of objects (even if they have not been observed during training), provided that we can decompose the scene into objects and that the "compare" operation has been learned correctly. Similarly, to solve the problem on the right, we should first decompose the image into eight objects, recognize their individual shapes, count the dots inside of them, and then reason about how the patterns change along the horizontal and vertical axes by comparing the individual nodes in the graph (objects). The described reasoning process is at the core of the composition problem.

The key aspect for compositionality to arise is a structured model, namely an NN that organizes its computation in terms of objects and their relations. Separate representation of relations allows for representing graphs of different structures and different edge types. Relations can be represented in a similar way to objects, as distributed representations (of graph "edges") that can encode causal ("triggered movement"), hierarchical ("is part of") or comparative ("smaller than") relations [Greff et al., 2020]. Combining objects and relations into different structures requires a variable binding mechanism, as there could be many different relations between the same objects, e.g. "smaller than", "left of", or even abstract ones such as "older than". By connecting multiple object-relation pathways, one can solve multihop compositional reasoning tasks, such as the introductory example: "Could the cookies on the table be evenly distributed among children?". It should also be possible to infer a task-dependent structure, so relational inference should be dynamic.

Concluding thoughts. Finally, to give a complete overview of the field, note that it is not unanimously accepted that the binding problem exists in the human brain (or NNs for that matter) [Riesenhuber and Poggio, 1999a,b; Ghose and Maunsell, 2002, 2008]. They argue that the classical neural coding scheme [Hubel and Wiesel, 1962] can cope with the perceptual combinatorial complexity. However, the debate whether there is a binding problem in the brain is beyond the scope of this thesis. Here, we look at the binding problem as a useful framework for thinking about ways to improve current state-of-the-art NNs. Importantly, the failure modes of the NNs with which we are concerned are well agreed upon, and for which there exists extensive empirical evidence [Greff et al., 2020; Bugliarello et al., 2023; Paiss et al., 2023; Hsieh et al., 2023; Yuksekgonul et al., 2022; Zhao et al., 2022].

1.2 Prior work and possible ways of going forward

As discussed above, neural networks struggle to generalize out-of-distribution, adapt quickly to novel scenes, and solve compositional reasoning tasks. The perspective of the binding problem gives us a framework to think about solutions and potential ways of moving forward. In our view, there are three high-level directions worth pursuing to improve NNs:

1. Object-centric inductive biases,
2. Scaling network parameters, data, and compute and
3. Modular NNs and, in particular, "tools use" through LLMs that act as controllers and orchestrate a number of models ("tools").

In the following, we describe each of these directions and give an overview of prior work in each of the fields.

1.2.1 Object-centric Inductive Biases

If a neural network does not have an inductive bias that would enforce learning compositional representations, the gradient descent method will most likely produce distributed representations. Even if the data-generating process is compositional, unless there is enough variation of all possible combinations of generating factors, the network will tend to learn an entangled representation [Bengio et al., 2013; Garnelo and Shanahan, 2019].

Object-centric representations facilitate relational reasoning and generalization, leading to better performance on downstream tasks such as visual question answering [Ding et al., 2021; Wu et al., 2023], video game playing [Zambaldi et al., 2019; Kulkarni et al., 2019; Gopalakrishnan et al., 2021; Stanić et al., 2023; Yoon et al., 2023], and robotics [Mandikal and Grauman, 2021; Wu et al., 2021; Sharma and Kroemer, 2021], compared to other monolithic representations of visual input. Here, we discuss the most common object-centric approaches: slot-based representations that enforce strict independence between inferred objects, methods to infer (hierarchical) relations between learned objects, approaches to learning a "dynamic" object representations, and approaches to learning object groupings through temporal synchrony of phases of complex-valued NNs.

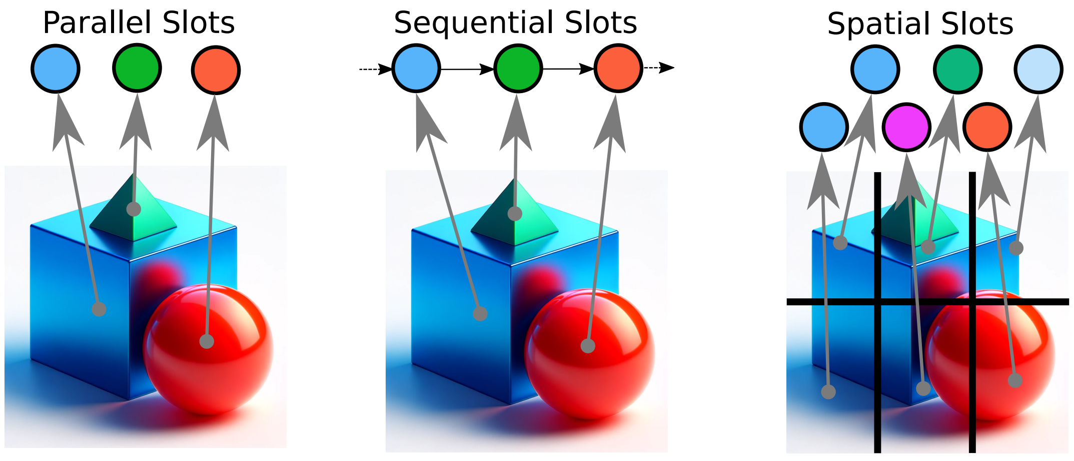


Figure 1.7: Slot-based object-centric representations can be divided into parallel slots, sequential slots and spatial slots.

Slot-based representations. Neural networks with slot-based representations infer objects from visual input, often through a shared set of weights, which guarantees a common format. They can be divided into parallel slots, sequential slots, and spatial slots, as shown in Figure 1.7.

Parallel slots (Figure 1.7, left) are perhaps the most extensively studied kind of slot-based methods [Greff et al., 2015, 2016, 2017, 2019; Prémont-Schwarz et al., 2017; van Steenkiste et al., 2018; Veerapaneni et al., 2020; Goyal et al., 2019; Kipf et al., 2020; Löwe et al., 2020; Kipf et al., 2021; Zoran et al., 2021; Kabra et al., 2021; Ding et al., 2021; Singh et al., 2022; Elsayed et al., 2022; Wu et al., 2023]. Here, all objects are inferred in parallel through a shared set of weights. Since inference is parallel, there is a need for a symmetry-breaking mechanism such that not all slots infer the same information. This line of work was initiated by Greff et al. [2015] where binding on simple toy datasets was obtained with reconstruction clustering in a denoising autoencoder. Many other methods followed, such as NEM [Greff et al., 2017] which learns object representations using a spatial mixture model, IODINE [Greff et al., 2019] that extends NEM by iteratively refining inferred objects, SlotAttention [Locatello et al., 2020] that further simplifies the inference by iterating in the latent instead of the input space, SAVi [Kipf et al., 2021] that extends it to videos and SAVi++ [Elsayed et al., 2022] that uses depth information to scale slot-based methods to real-world data. Another line of work applied SlotAttention-like iterative mechanisms to reconstruct features instead of pixels for images (DINOSAUR [Seitzer et al., 2022]) and videos (VideoSAUR [Zadaianchuk et al., 2023]). To extract features for grouping, they use a powerful visual encoder pre-trained in a self-supervised manner [Radford et al., 2021; Caron et al., 2021]. Using this approach, they showed that by optimizing in the highly semantic space of these visual encoders, it is possible to discover objects on complex real-world image datasets.

Sequential slots (Figure 1.7, middle), on the other hand, infer objects one by one, usually through an attention "glimpse" [Schmidhuber and Huber, 1991; Schmidhuber, 1993a] that is guided by a recurrent NN [Eslami et al., 2016; Kosiorek et al., 2018; Stanić and Schmidhuber, 2019; Yuan et al., 2019; Stelzner et al., 2019; Burgess et al., 2019; Engelcke et al., 2020]. The sequential inference mechanism removes the need for symmetry breaking that parallel slots have. However, since the objects are ordered now, they are no longer permutation-invariant, so this complicates the relational inference process. The earliest method AIR [Eslami et al., 2016] learns to infer one object per iteration with a hard-attention window on a given image, followed by SQAIR [Kosiorek et al., 2018] which extends it to the sequential setting, R-SQAIR [Stanić and Schmidhuber, 2019] which improves the relational reasoning ability of SQAIR, and MoNET [Burgess et al., 2019] which uses a VAE and a soft-attention mechanism. Due to recurrence, these slots may not always be fully independent, which limits their modularity as building blocks. GENESIS [Engelcke et al., 2020] alleviates this by not using recurrence for object representations, but only for the information routing.

Spatial slots (Figure 1.7, right) infer each object from a particular part (e.g. patch) of the image, which hinders their ability to represent arbitrarily shaped objects. Approaches such as Relation Networks [Santoro et al., 2017; Zambaldi et al., 2019; Stanić et al., 2021] consider slices of a feature tensor of a CNN as a slot representation, whereas SPAIR [Crawford and Pineau, 2019], SPACE [Lin et al., 2020] and SCALOR [Jiang* et al., 2020] use an explicit feature to indicate presence of an object. Although these methods offer simplicity in the segregation process due to the fixed "attention pattern", the composition modules need to implicitly segregate objects to solve the task.

Relational Inference. Once the objects are inferred, we need to infer relations [Battaglia et al., 2018] between them to make inferences such as prediction, planning, and reasoning (the "composition problem" discussed in Section 1.1). The early object-centric methods [Battaglia et al., 2016; Chang et al., 2017; van Steenkiste et al., 2018] computed relations by learning a neural network that "passes messages" between the representations of objects (here acting as nodes in the graph) and outputs the edge (relation).

In their most general form, such relational networks are called message-passing NNs (MPNNs) [Gilmer et al., 2017; Battaglia et al., 2018]. They update object representations by computing and then aggregating incoming edge representations. This update process can be repeated iteratively. MPNNs were first introduced as a generalization of RNNs to graph-structured inputs [Sperduti and Starita, 1997; Gori et al., 2005], after which they were also explored in a deep learning setting [Li et al., 2015]. MPNNs have been shown to lead to improved generalization on tasks such as common sense physical reasoning [Battaglia et al., 2016; Chang et al., 2017], hierarchical reasoning [Mrowca et al., 2018; Stanić et al., 2021], visual question answering [Santoro et al., 2017], and physical construction [Hamrick et al., 2018].

Graph Neural Networks (GNNs) [Scarselli et al., 2009; Pollack, 1990] are a special kind of MPNN that have both a *node embedding* and an *edge embedding* NNs. Edges determine how messages are passed between nodes (objects). They have been studied in diverse contexts, such as modeling dynamical systems, learning intuitive physics, multi-object scenes, motion capture data, and multi-agent systems [Goller and Kuchler, 1996; Küchler and Goller, 1996; Küchler, 2000; Scarselli et al., 2009; Bronstein et al., 2017; Watters et al., 2017; Raposo et al., 2017; Santoro et al., 2017; Gilmer et al., 2017; Wang et al., 2018; van Steenkiste et al., 2018; Kipf et al., 2018; Battaglia et al., 2018; Wang et al., 2018].

Graph Convolutional Networks (GCNs) [Kipf and Welling, 2016; Bronstein et al., 2017; Hamilton et al., 2017] are GNNs that generalize convolutional NNs. CNNs operate on a fixed grid, whereas GCNs operate on arbitrary graph structures and use graph convolution operators to update node representations. GCNs excel in tasks with graph-structured data, such as social networks [Hamilton et al., 2017], citation networks [Kipf and Welling, 2016], knowledge base completion tasks [Schlichtkrull et al., 2018], and biochemical modeling [Atwood and Towsley, 2016]. However, GCNs require the knowledge of the interaction graph, which may not be known apriori, e.g. when discovering the interaction structure of molecules or when learning from raw visual data. Finally, Battaglia et al. [2018] introduced a general framework that generalizes MPNNs and GCNs, and additionally includes a global graph representation that interacts with all nodes and edges.

Approaches based on self-attention, such as Transformers [Vaswani et al., 2017] (which extend the unnormalized linear Transformers of 1991 [Schmidhuber, 1992; Katharopoulos et al., 2020; Marcin Choromanski et al., 2021; Schlag et al., 2021]), are a special type of MPNNs (and GNNs) [Battaglia et al., 2018] that use self-attention to compute a weighted sum of the aggregated messages to update the node representations. This facilitates dynamic information routing through relational inference in a "soft" manner, in the sense that attention weights can be interpreted as representing relations, but they need to obey certain constraints, such as adding one. For example, non-local networks were applied to images [Wang et al., 2018] where the attention coefficients model relations between spatial slots. However, they use only a single attention "head", which limits the number of different interactions that these networks can learn. Several works have explored the use of "multihead" attention transformers and showed performance gains for relational reasoning about objects [Zambaldi et al., 2019; van Steenkiste et al., 2018; Goyal et al., 2019; Santoro et al., 2018], citation networks [Veličković et al., 2017], question answering [Dehghani et al., 2018], and language modeling [Devlin et al., 2018; Brown et al., 2020].

Transformers can, however, be computationally inefficient, since they assume a fully connected graph and require computing all possible messages. Several approaches try to learn a sparse graph. For example, RIMs [Goyal et al., 2019] (a special kind of RMC [Santoro et al., 2018]) infer a sparse graph and can be seen as a GNN where the nodes are individual mechanisms that communicate sparsely between each other.

Transformers, RIMs, and RMC learn relations *implicitly* through soft attention coefficients. In some cases, it is desirable to dynamically infer explicit edge representations, such as in NRI [Kipf et al., 2018], which infers edges between moving particles (given their coordinates) as latent variables in a VAE [Kingma and Welling, 2013; Rezende et al., 2014] setup. HRI [Stanić et al., 2021] further extends the NRI to learn objects and edges directly from visual input and uses a hierarchical inductive bias to infer a hierarchical graph.

Dynamic object representations. In general, the notion of what an object is is ill-defined, as in the example of a bottle and a cap in Figure 1.5. Thus, ideally, object-centric representations in NNs should have task-based top-down modulation, e.g. through text in a weakly supervised setup or through actions or goals in an RL setup.

Early approaches that could incorporate top-down feedback were based on attractor networks such as Amari-Hopfield networks [Amari, 1972; Hopfield, 1982] (see also Grossberg's work on biological networks [Grossberg, 1969, 2013]), Boltzmann machines [Ackley et al., 1985], and associative memory [Kohonen, 2012] where an object representation is inferred by running a dynamical system until it converges to a stable attractor state. Since ambiguous inputs can have multiple competing interpretations, top-down feedback can be used to guide the system to a different attractor state. Attractor networks can be difficult to train, so they have received little attention lately, with a few notable exceptions [Mozer et al., 2018; Iuzzolino et al., 2019]. Since each weight participates in each attractor, spurious (unintended) attractors and ill-conditioned attraction basins may appear [Neto and Fontanari, 1997].

An alternative class of methods connects object discovery and text through contrastive learning [Hénaff et al., 2022; Xu et al., 2022b]. Although they deal with real-world images, they are only able to discover *semantic* groupings and future work is necessary to enable *instance/object-level* groupings.

In an RL setup, S3A [Mott et al., 2019] uses attention to learn action-modulated object-like representations, while OP3 [Veerapaneni et al., 2020] extends IODINE to the sequential setting and uses actions to modulate object representations in a model-based RL setup. C-SWM [Kipf et al., 2020] learns to factor input into objects using actions and predicting the dynamics of the environment, and Biza et al. [2022] proposes attention mechanisms for binding actions to objects. Another line of work [Tang et al., 2020; Tang and Ha, 2021; Stanić et al., 2023] shows that agents with object-centric representations have improved generalization and interpretability capabilities in (open-world) games.

Synchrony-based representations. Slot-based approaches have several conceptual limitations. First, the binding information (i.e. addresses) about object instances is maintained only by the constant number of slots, a hard-wired component which cannot be adapted through learning. This restricts the ability of slot-based models to flexibly represent a variable number of objects with variable precision without tuning the slot size, number of slots, number of iterations, etc. Second, the inductive bias used by the grouping module strongly enforces the independence among all pairs of slots. This restricts individual slots to store relational features at the object level and requires additional processing of slots using a relational module, e.g., Graph Neural Networks [Battaglia et al., 2016; Gilmer et al., 2017] or Transformer models [Vaswani et al., 2017; Schmidhuber, 1992; Schlag et al., 2021]. Third, iterative attention-based binding is computationally very demanding to train [Löwe et al., 2022]. Additionally, the spatial broadcast decoder [Watters et al., 2019b] (a necessary component in these models) requires multiple forward/backward passes to render the slot-wise reconstruction, resulting in a large memory overhead. Synchrony-based methods can in principle address these limitations.

In synchrony-based approaches (Figure 1.8), each object shares the same features, which are augmented by additional information (such as the phase of a complex number). The additional information is then responsible for group-

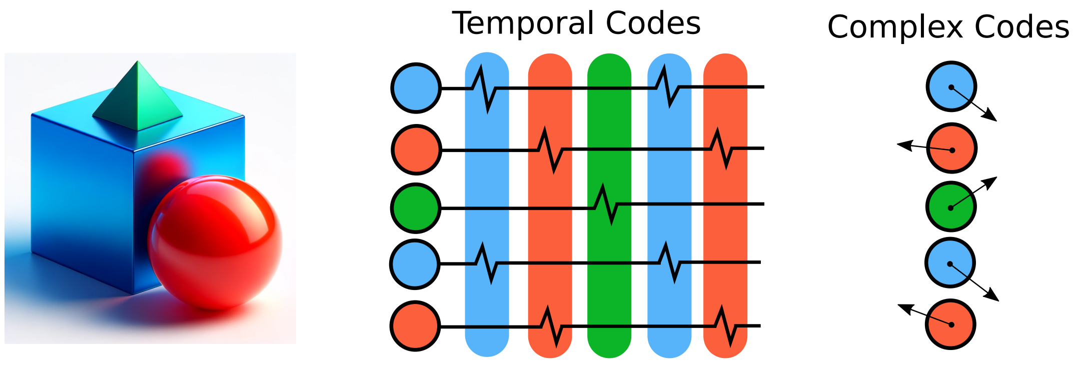


Figure 1.8: Illustration of synchrony-based approaches: in temporal codes (middle) neurons that belong together are synchronized (i.e. they spike at the same time), whereas in complex codes (right) the neurons that group together have similar phase values.

ing. The grouping is usually continuous, which may facilitate the computation of relational information, without requiring an additional relational mechanism. However, extracting the grouping information necessitates an additional step, such as clustering based on some similarity metric. Such grouping information may help encode uncertainty about objects, while complicating situations when a feature is active in multiple objects. Early approaches determined synchrony through the firing patterns of spiking neurons [Singer, 2009] (the middle image in Figure 1.8). Here, neurons that fire in synchrony are considered to represent the same object [Milner, 1974; Von Der Malsburg, 1994; Singer, 1999]. The drawback of these networks is that they are non-differentiable, since the temporal codes rely on spiking neurons.

An alternative to NNs based on temporal codes are complex-valued NNs (right image in Figure 1.8). Recently, [Löwe et al., 2022; Stanić et al., 2023b] revived this class of neural object binding models [Mozer et al., 1991; Mozer, 1998; Reichert and Serre, 2014]. Complex-valued NNs are conceptually very promising. In principle, they address most of the conceptual challenges faced by slot-based models. The binding mechanism is implemented through constructive or destructive phase interference caused by the addition of complex-valued activations. They store and process information about object instances in the phases of complex activations that are more amenable to adaptation through gradient-based learning. Furthermore, they can, in principle, store a variable number of objects with variable precision by partitioning the phase components of complex activations at varying levels of granularity. Additionally, synchrony-based models can represent relational information directly in their distributed representation, that is, distance in phase space yields an implicit relational metric between object instances (e.g., inferring part-whole hierarchy from distance in "tag" space [Mozer, 1998]). Lastly, training synchrony-based models is computationally more efficient by two orders of magnitude [Löwe et al., 2022]. Building atop of CAE [Löwe et al., 2022], CtCAE [Stanić et al., 2023b] introduced a novel contrastive learning method that increases separability in phase values of pixels belonging to different objects and facilitates scaling synchrony-based methods to multi-object color datasets.

1.2.2 Scaling Parameters, Data and Compute

Lately, a very active area of exploration has been on ways of scaling NNs in the hope of solving all outstanding issues of NNs (including compositional reasoning and out-of-distribution generalization) in this manner. The SotA on real-world visual question answering (VQA) tasks has been largely obtained by scaling end-to-end vision and language models (VLMs) in terms of their size, training data and compute [Alayrac et al., 2022; Chen et al., 2022b; Li et al., 2023; Driess et al., 2023; Chen et al., 2023b,a].

One of the first VLMs Flamingo [Alayrac et al., 2022] used a frozen pretrained language model of up to 70B parameters and a frozen pretrained image encoder with 435M parameters and trained only a "cross-attention" module that served as an interface between them. Since then, effort has been put mainly into scaling both the image and the language models. GIT [Wang et al., 2022a] used a 300M language model and scaled the image encoder to 4.8B parameters. PaLI [Chen et al., 2022b] jointly scaled both components, the language model to 17B and the image encoder to 4B parameters. PaLI-X [Chen et al., 2023b] continued this trend of scaling the total number of parameters to 55B using an image encoder with 22B parameters. PaLM-E scaled the number of total parameters in the VLM to 562B by integrating the 540B PaLM [Chowdhery et al., 2022] and the 22B Vision Transformer [Dosovitskiy et al., 2020; Dehghani et al., 2023]. On the other hand, BLIP-2 [Li et al., 2023] achieved SotA performance in various tasks with a 12B VLM and PaLI-3 [Chen et al., 2023b] introduced a significantly smaller VLM with 5B total parameters that achieves competitive performance with SotA models on various VLM benchmarks.

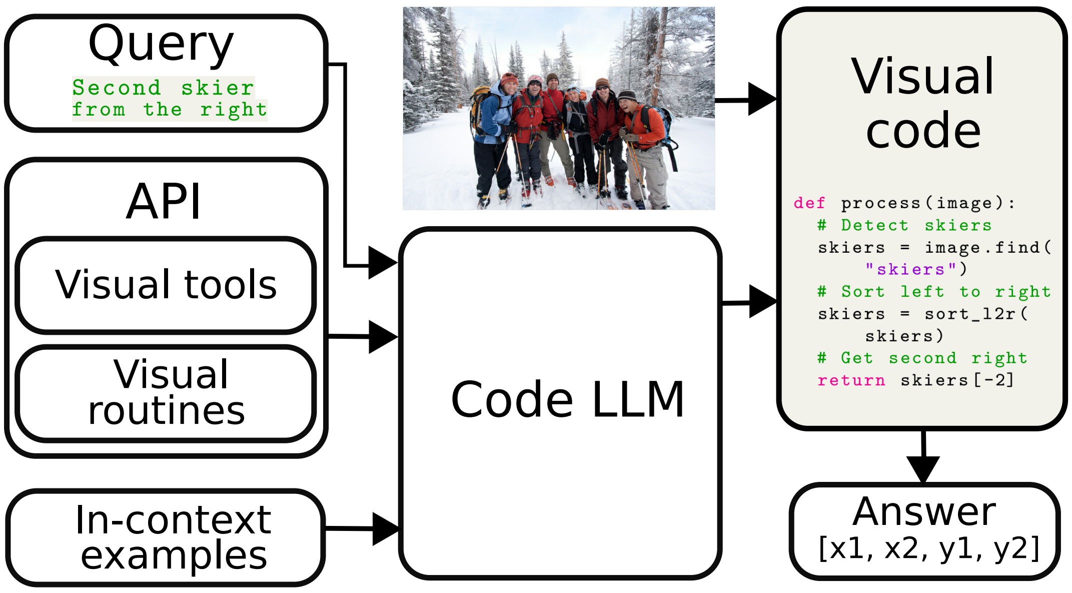
These models are typically pretrained on large amounts of data and then fine-tuned for the best performance on the downstream tasks. However, even the largest VLMs struggle with tasks that require compositional reasoning, the ability to generalize, fine-grained spatial capabilities, and counting [Bugliarello et al., 2023; Paiss et al., 2023; Hsieh et al., 2023; Yuksekgonul et al., 2022; Zhao et al., 2022; Hendricks and Nematzadeh, 2021]. Further scaling makes them even more data- and compute-hungry; therefore, it is unclear whether scaling alone can solve these tasks. For these reasons, we remain highly skeptical about the potential of this hypothesis and do not pursue these directions in this thesis.

1.2.3 Modular Networks and LLMs as controllers

An alternative approach to solving tasks that require compositional reasoning and out-of-distribution generalization is to decompose tasks into subtasks, solve them individually, and then combine the answers to solve the original task. This is reminiscent of the way humans approach compositional problems. According to Daniel Kahneman's framework [Kahneman, 2017], our thought process can be thought of as consisting of two mechanisms: System 1 and System 2. System 2 is the "slow", "analytical" system that can decompose the task into subtasks, while System 1 is the "fast", "reactive" system that solves individual tasks such as recognizing patterns. The early work on task decomposition was pioneered by Neural Module Networks (NMNs) [Andreas et al., 2016; Johnson et al., 2017; Hu et al., 2017]. NMNs are trained end-to-end, using supervised or reinforcement learning. NMNs are designed to have several modules and the hope is that during training, each module will learn a different functionality, which will then be reusable across tasks. However, these models have a number of drawbacks: program generation requires hand-tuned parsers, and they require optimization through reinforcement learning (e.g., REINFORCE [Williams, 1992]), which is often unstable. Learning all modules end-to-end hinders their ability to generalize [Bahdanau et al., 2018] and sometimes leads to a mode "collapse", where some modules take over all the work and other modules are never activated, or modules that do not learn intended functionalities [Subramanian et al., 2020]. Additionally, they sometimes require supervision for program learning, which is difficult to obtain at scale.

Recently, an alternative approach became prominent in the literature, the one based on "tool use" [Parisi et al., 2022; Schick et al., 2023], in particular for structured reasoning in the natural language domain [Madaan et al., 2022; Wang et al., 2023c; Gao et al., 2023; Chen et al., 2022a]. In tool use (see Figure 1.9 for an example of tool use for visual reasoning), an LLM is used as a controller (akin to System 2) to solve the task by orchestrating a set of tools (such as an object detector, akin to System 1).

In the domain of using LLMs as controllers for visual reasoning, PICa [Yang et al., 2022] solves a knowledge-based VQA task by first extracting an object and captions from the image and then querying GPT-3 with this information



```
def process(image):
  # Detect skiers
  skiers = image.find(
      "skiers")
  # Sort left to right
  skiers = sort_l2r(
      skiers)
  # Get second right
  return skiers[-2]
```

Figure 1.9: An example framework of tool use for visual reasoning [Surís et al., 2023; Stanić et al., 2024]. Here, a code-generating LLM takes as input a query (e.g. "second skier from the right), an API for tool use (e.g. Python documentation for visual tools) and (optionally) a number of query-code pairs as in-context examples in the prompt. The generated (visual) code takes the image as input, executes and returns the answer to the query (here a bounding box coordinates).

and in-context examples to answer a question. Socratic Models [Zeng et al., 2022], HuggingGPT [Shen et al., 2023], and Societies of Mind [Zhuge et al., 2023] compose vision and language models to "communicate" in a fixed "protocol" and solve tasks such as image captioning, visual question answering, image generation, and robot planning.

On the other hand, models such as VisProg [Gupta and Kembhavi, 2023], ViperGPT [Surís et al., 2023], and CodeVQA [Subramanian et al., 2023] show great promise in solving visual question answering by using an LLM orchestrating tool use by writing a (Python) program. During execution, the program calls individual vision modules (such as object detector, depth estimator) through an *API* that is provided in the prompt. For example, to answer "what is the color of the shirt of the second person to the left?", the program would detect all persons, sort them horizontally, take the second person, detect their shirt, and then query its color. These models showed great performance and achieved SotA on tasks such as compositional visual question answering, visual grounding, and video temporal reasoning. Furthermore, by their construction, they are interpretable, offer strong generalization, mathematical and reasoning skills, compositional, adaptable (individual models can be simply swapped with better models) and do not require gradient-based training.

However, in their current form, these models heavily rely on human engineering of in-context examples (ICEs) in the prompt. Moreover, ICEs are often dataset- and task-specific. To generate them, significant labor by highly skilled workers is required. For this reason, we argue that these methods *should not be called zero-shot* in their current form. Some promising ways of going beyond human prompt engineering are automatizing the generation of in-context examples, or enabling LLMs to "self-correct" their own output [Stanić et al., 2024]. Below, we provide an overview of the related work on automatizing prompt engineering and "self-correcting" LLMs.

Automatizing prompt engineering. Vast literature shows that prompt format and contents are often important for achieving good performance with an LLM [Reynolds and McDonell, 2021; Zhao et al., 2021; Lu et al., 2021; Moradi and Samwald, 2021; Madaan and Yazdanbakhsh, 2022; Wei et al., 2023]. A prompt typically consists of a task description (in natural language), in-context examples (e.g. query-code in ViperGPT [Surís et al., 2023]) and an API (in the case where LLMs write a program).

Various prompting techniques have been engineered, such as Chain-of-Thought prompting [Wei et al., 2022], Self-Consistency [Wang et al., 2022c], Tree of Thoughts [Yao et al., 2023], Graph of Thoughts [Besta et al., 2023], Plan-and-Solve Prompting [Wang et al., 2023a], Least-to-Most Prompting [Zhou et al., 2022a], etc. All these techniques rely on human prompt engineering, in particular, on in-context examples.

On the other hand, some methods try to automatize prompt engineering. They sometimes use gradient-based optimization [Shin et al., 2020; Gao et al., 2020; Wen et al., 2023] and some approaches require only API access to the model [Xu et al., 2022a; Prasad et al., 2022].

Other works use LLMs for prompt optimization. APE [Zhou et al., 2022b] first generates instructions with an LLM, then selects instructions with the highest accuracy, and uses them for future LLM prompts. APO [Pryzant et al., 2023] generates feedback with an LLM that informs how to update the previous instruction. OPRO [Yang et al., 2023a] uses an LLM to generate new instructions in each optimization step, asking the LLM to improve task accuracy by changing task instructions, which requires determining the score on a small set of labeled examples and providing it in the meta-prompt.

Promptbreeder [Fernando et al., 2023] goes a step further and proposes a self-referential self-improvement LLM using a meta-prompt that controls the generation of the main (task) prompt and evolves both via mutation. More importantly, Promptbreeder shows some surprising results such that a simple prompt "SOLUTION" outperforms all previous approaches. This further demonstrates the sensitivity of LLMs and the importance of automatizing the prompt engineering process. Common to all above frameworks for automatizing prompting is that they automatize the "task description" part of the prompt. Other approaches [Stanić et al., 2024] automatize the generation of in-context examples, which might have an even greater influence on the performance of the LLM.

LLMs and self-correction. In the LLM literature, there have been mixed findings on the ability of LLMs to critique and self-correct their own reasoning and outputs. Self-Refine [Madaan et al., 2023] provides feedback to the LLM of the previously generated output, which is then refined. Several other approaches show benefits of providing feedback to LLM in improving reasoning [Shinn et al., 2023; Madaan et al., 2023], code generation [Chen et al., 2023c; Olausson et al., 2023; Chen et al., 2023], improving LLM alignment [Bai et al., 2022; Ganguli et al., 2023], etc.

On the other hand, there has been increasing evidence that LLMs cannot self-correct reasoning yet [Huang et al., 2023; Stechly et al., 2023; Valmeekam et al., 2023], unless they receive external feedback, which usually requires access to ground truth labels. Other work [Stanić et al., 2024] also found that providing the previous question and code as feedback to the model did not improve the results, but that it is possible to improve performance by tuning hyperparameters on the fly.

1.3 Contributions and organization

In this thesis, we make several contributions on learning object-centric representations from raw visual data and on using LLMs as programmers for visual reasoning (tool use). In terms of object-centric representations, we introduce novel methods for all types of slots (parallel, sequential, and spatial), as well as a novel synchrony-based method for learning object representations. We develop novel methods in both the unsupervised and RL setup.

In Chapter 2 we present a method that improves the reasoning and generalization abilities of a sequential slots model. Here, we identify a key limitation of sequential slots methods that compute interactions between objects in a sequential manner, which limits their predictive power. We address this by endowing a sequential slots model with a module with strong relational inductive bias that computes in parallel pairwise interactions between inferred objects. This model leads to better prediction on videos of interacting objects and improved combinatorial generalization.

Since many real-world objects have hierarchical structures, in Chapter 3 we introduce a method (called *Hierarchical Relational Inference* (HRI)) that is capable of inferring nodes and edges of a hierarchical graph directly from raw visual data. Our approach to physical reasoning models objects as hierarchies of parts that may locally behave separately but also act more globally as a single whole. For this, HRI uses spatial slots. Unlike prior approaches, our method learns in an unsupervised fashion directly from raw visual images to discover objects, parts, and their relations. It explicitly distinguishes multiple levels of abstraction and improves over a strong baseline in modeling synthetic and real-world videos.

As discussed above, (hierarchical) decomposition into objects is generally task dependent, and sometimes it is infeasible and undesirable to decompose a scene into all hierarchy levels. For these reasons, it might be more beneficial to modulate objects with task information, such as words (in VQA) or actions/goals (in RL). In Chapter 4 we introduce and study object-centric agents in an RL setting. Starting from the Crafter benchmark, a 2D open-world survival game, we also introduce a new set of environments that evaluate out-of-distribution generalization. In particular, they evaluate the agent's ability to generalize to previously unseen objects or their frequencies of appearance. We show that current agents struggle to generalize, whereas our novel object-centric agents improve over strong baselines. The object-centric agents that we introduce are based on both parallel slots and spatial slots. We achieve new SotA performance in these environments and provide critical insights that are of general interest for future work.

In Chapter 5 we look at synchrony-based methods, an alternative to slot-based methods that have the potential to address several of their limitations. The main limitations of slot-based methods are that the number of slots is hard-wired; all slots have equal capacity; training has high computational cost; there are no object-level relational factors within slots. Synchrony-based models can, in principle, address these limitations by using complex-valued activations that store binding information in their phase components. However, working examples of such synchrony-based models have been developed only very recently and are still limited to toy grayscale datasets and simultaneous storage of less than three objects in practice. In Chapter 5 we introduce architectural modifications and a novel contrastive learning method that greatly improve the state-of-the-art synchrony-based model. For the first time, we obtain a class of synchrony-based models capable of discovering objects in an unsupervised manner in multi-object color datasets and simultaneously representing more than three objects.

Finally, in Chapter 6 we look at "tool use" and introduce a framework for compositional visual reasoning. Visual reasoning with LLMs addresses the limitations of end-to-end models by decomposing the task and solving subtasks by orchestrating a set of (visual) tools. Recently, these models achieved great performance on tasks such as compositional visual question answering, visual grounding, and video temporal reasoning. Nevertheless, in their current form, these models heavily rely on human engineering of in-context examples in the prompt, which are often dataset- and task-specific and require significant labor by highly skilled programmers. In this work, we present a framework that mitigates these issues by introducing spatially and temporally abstract routines and by leveraging a small number of labeled examples to automatically generate in-context examples, thereby avoiding human-created in-context examples. On a number of visual reasoning tasks, we show that our framework leads to consistent gains in performance, makes LLMs as controllers setup more robust, and removes the need for human engineering of in-context examples.

Lastly, in Chapter 7 we provide a summary of our contributions and offer promising directions going forward.

Chapter 2

Improving Relational Reasoning In Sequential Slots Models

Numerous studies [Xu and Carey, 1996; Kellman and Spelke, 1983; Spelke et al., 1995; Baillargeon et al., 1985; Saxe and Carey, 2006] show that infants quickly develop an understanding of intuitive physics, objects and relations in an unsupervised manner. To facilitate the solution of real-world problems, intelligent agents should be able to acquire such knowledge [van Steenkiste et al., 2019]. However, artificial neural networks are still far from human-level understanding of intuitive physics.

Existing approaches to unsupervised learning about objects and relations from visual data can be categorized into either parallel [Greff et al., 2016, 2017, 2019], sequential [Schmidhuber and Huber, 1991; Schmidhuber, 1993a; Eslami et al., 2016; Kosiorek et al., 2018; Crawford and Pineau, 2019; Burgess et al., 2019; Yuan et al., 2019] or spatial [Santoro et al., 2017; Zambaldi et al., 2019; Crawford and Pineau, 2019; Stanić et al., 2021], depending on the core mechanism responsible for inferring object representations from a single image. One model from the former group is Tagger [Greff et al., 2016] which applies the Ladder Network [Rasmus et al., 2015] to perform perceptual grouping. RTagger [Prémont-Schwarz et al., 2017] replaces the Ladder Network by a Recurrent Ladder Network, thus extending Tagger to sequential settings. NEM [Greff et al., 2017] learns object representations using a spatial mixture model and its relational version R-NEM [van Steenkiste et al., 2018] endows it with a parallel relational mechanism. The recently proposed IODINE [Greff et al., 2019] iteratively refines inferred objects and handles multi-modal inputs.

This chapter is based on "Relational Sequential Attend, Infer, Repeat" [Stanić and Schmidhuber, 2019] paper, which was presented as a workshop paper at NeurIPS 2019.

On the other hand, the sequential attention model AIR [Eslami et al., 2016] learns to infer one object per iteration over a given image. Contrary to NEM, it extracts object glimpses through a hard attention mechanism [Schmidhuber and Huber, 1991] and processes only the corresponding glimpse. Furthermore, it builds a probabilistic representation of the scene to model uncertainty. Many recent models have AIR as the core mechanism: SQAIR [Kosiorek et al., 2018] extends AIR to sequential settings, similarly does DDPAE [Hsieh et al., 2018]. SPAIR [Crawford and Pineau, 2019] scales AIR to scenarios with many objects and SuPAIR[Stelzner et al., 2019] improves speed and robustness of learning in AIR. The recent MoNET [Burgess et al., 2019] also uses a VAE and a recurrent neural network (RNN) to decompose scenes into multiple objects. These methods usually model relations by a sequential relational mechanism such as an RNN which limits their relational reasoning capabilities[Battaglia et al., 2018].

Here we present Relational Sequential Attend, Infer, Repeat (R-SQAIR) to learn a generative model of intuitive physics from video data. R-SQAIR builds on SQAIR which we augment by a mechanism that has a strong relational inductive bias [Battaglia et al., 2016; van Steenkiste et al., 2018; Santoro et al., 2018]. Our explicit parallel model of pairwise relations between objects is conceptually simpler than a sequential RNN-based model that keeps previous interactions in its memory and cannot directly model the effects of interactions of previously considered objects. Our experiments demonstrate improved generalization performance of trained models in new environments.

2.1 Relational Sequential Attend Infer Repeat

Attend, Infer, Repeat (AIR) [Eslami et al., 2016] is a generative model that explicitly reasons about objects in a scene. It frames the problem of representing the scene as a probabilistic inference in a structured VAE. At the core of the model is an RNN that processes objects one at a time and infers latent variables $\mathbf{z} = \{\mathbf{z}^{\mathrm{i}}_{\mathrm{what}}, \mathbf{z}^{\mathrm{i}}_{\mathrm{where}}, z^{\mathrm{i}}_{\mathrm{pres}}\}_{i=1}^{n}$, where $n \in \mathbb{N}$ is the number of objects. The continuous latent variable $\mathbf{z}^{\mathrm{what}} \in \mathbb{R}^{d}$ encodes the appearance of the object in the scene (where $d \in \mathbb{N}$ is the hidden size) and $\mathbf{z}^{\mathrm{where}} \in \mathbb{R}^{4}$ encodes the coordinates according to which the object glimpse is scaled and shifted by a Spatial Transformer [Jaderberg et al., 2015]. Given an image $\mathbf{x} \in \mathbb{R}^{h \times w \times c}$, where $h, w, c \in \mathbb{N}$ are height, width, and the number of channels, the generative model of AIR is defined as follows:

$$p_\theta(\mathbf{x}) = \sum_{n=1}^{N} p_\theta(n) \int p_\theta^z(\mathbf{z}|n) p_\theta^x(x|\mathbf{z}) d\mathbf{z}, \tag{2.1}$$

where $p_\theta(n) = \mathrm{Geom}(n \mid \theta)$ represents the number of objects present in the scene, $p_\theta^z(\mathbf{z}|n)$ captures the prior assumptions about the underlying object and $p_\theta^x(x|\mathbf{z})$ defines how it is rendered in the image. In general, the inference for Equation 2.1 is intractable, so [Eslami et al., 2016] employs amortized variational inference using a sequential algorithm, where an RNN is run for N steps to infer latent representation of one object at a time. The variational posterior is then:

$$q_\phi(\mathbf{z} \mid \mathbf{x}) = q_\phi\big(z_{\mathrm{pres}}^{n+1} = 0 \mid \mathbf{z}^{1:n}, \mathbf{x}\big) \prod_{i=1}^{n} q_\phi\big(\mathbf{z}^i, z_{\mathrm{pres}}^i = 1 \mid \mathbf{z}^{1:i-1}, \mathbf{x}\big), \tag{2.2}$$

where q_ϕ is a neural network which outputs the parameters of the latent distributions: the mean and standard deviation of a Gaussian distribution for $\mathbf{z}_{\mathrm{what}}$ and $\mathbf{z}_{\mathrm{where}}$ and the probability parameter of the Bernoulli distributed $z_{\mathrm{pres}} \in []$.

Relational Sequential Attend, Infer, Repeat (R-SQAIR) augments SQAIR through a parallel relational mechanism. SQAIR extends AIR to the sequential setting by leveraging the temporal consistency of objects using a state-space model. It has two phases: Discovery (DISC) and Propagation (PROP). PROP is active from the second frame in the sequence, propagating or forgetting objects from the previous frame. It does so by combining an RNN, which learns the temporal dynamics of each object, with the AIR core which iterates over previously propagated objects (explaining away phenomena). DISC phase uses the AIR core, conditioned on propagated objects, to discover new appearances of objects. For a full description of AIR and SQAIR, we refer to previous work [Eslami et al., 2016; Kosiorek et al., 2018].

R-SQAIR retains the strengths of its predecessors and improves their relational capabilities. More specifically, SQAIR relies on the RNN core of AIR to model the relations. However, an RNN has only a weak relational inductive bias [Battaglia et al., 2018], as it needs to compute pairwise interactions between objects sequentially, iterating over them in a specific order. R-SQAIR, on the other hand, employs networks with strong relational inductive bias which can model arbitrary relations between objects in parallel. To construct conceptually simple yet powerful architectures that support combinatorial generalization, we use the following two methods: *Interaction Network* (IN) [van Steenkiste et al., 2018] and *Relational Memory Core* (RMC) [Santoro et al., 2018].

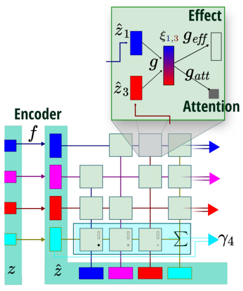


Figure 2.1: Interaction Network of R-NEM [van Steenkiste et al., 2018].

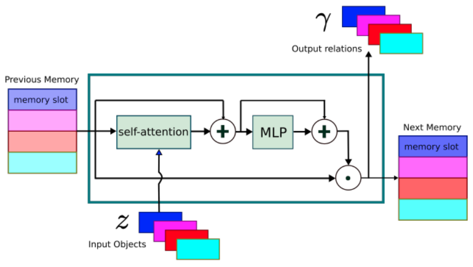


Figure 2.2: Relational Memory Core [Santoro et al., 2018].

The R-SQAIR generative model is built by extending the PROP module of SQAIR to include the relations $\gamma_t = \Gamma(\mathbf{z}_{t-1}) \in \mathbb{R}^{n \times n \times d_r}$, where Γ is the relational module, $n \in \mathbb{N}$ is the number of objects, $d_r \in \mathbb{N}$ is the dimensionality of the relations vector, and $\mathbf{z}_{t-1}$ are object representations from the previous timestep, defined as follows:

$$p(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}) = p^D(\mathbf{z}_1^{D_1}) \prod_{t=2}^{T} p^D(\mathbf{z}_t^{D_t} | \mathbf{z}_t^{P_t}) p^P(\mathbf{z}_t^{P_t} | \gamma_t) p_\theta(\mathbf{x}_t | \mathbf{z}_t), \tag{2.3}$$

The *discovery prior* $p^D(\mathbf{z}_t^{D_t} | \mathbf{z}_t^{P_t})$ samples latent variables $\mathbf{z}_t^{D_t}$ for new objects that enter the frame, by conditioning on propagated variables $\mathbf{z}_t^{P_t}$. The *propagation prior* $p^P(\mathbf{z}_t^{P_t} | \gamma_t)$ samples latent variables for objects that are propagated from the previous frame and removes those that disappear. Both priors are learned during training. We recover the original SQAIR model for $\gamma_t = \mathbf{z}_{t-1}$. The inference model is therefore as follows:

$$q_\phi(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T}) = \prod_{t=1}^{T} q_\phi^D\left(\mathbf{z}_t^{D_t} \mid \mathbf{x}_t, \mathbf{z}_t^{P_t}\right) \prod_{i \in \mathcal{O}_{t-1}} q_\phi^P\left(\mathbf{z}_t^i \mid {\gamma_t}^i, h_t^i\right), \tag{2.4}$$

where h_t^i are hidden states of the temporal and AIR core RNNs. Discovery q_ϕ^D is essentially the posterior of AIR. Again, the difference to SQAIR lies in the propagation module q_ϕ^P, which receives the relations γ_t as input.

2.1.1 Relational Module

Interaction Network Our first relational module is the Interaction Network (IN) of R-NEM [van Steenkiste et al., 2018], depicted in Figure 2.1, which is closely related to Interaction Networks [Battaglia et al., 2016; Watters et al., 2017]. Here, the effect on object k of all other objects $i \neq k$ is computed by the relational module $\gamma_t = \Gamma^{IN}(\mathbf{z}_{t-1})$, which in the case of IN is defined as follows (for simplicity we drop time indices):

$$\hat{\boldsymbol{z}}_k = f(\boldsymbol{z}_k),\ \ \boldsymbol{\xi}_{k,i} = g([\hat{\boldsymbol{z}}_k; \hat{\boldsymbol{z}}_i]),\ \ \boldsymbol{E}_k = \sum_{i \neq k} g_{\text{att}}(\boldsymbol{\xi}_{k,i}) \cdot g_{\text{eff}}(\boldsymbol{\xi}_{k,i}),\ \ \gamma_k = [\mathbf{z}_k; \mathbf{e}_k], \tag{2.5}$$

where $\mathbf{z}_i = \{\mathbf{z}^{\mathrm{i}}_{\mathrm{what}}, \mathbf{z}^{\mathrm{i}}_{\mathrm{where}}, z^{\mathrm{i}}_{\mathrm{pres}}\}$ from the previous time step. First, each object $\boldsymbol{z}_i$ is transformed using an MLP f to obtain $\hat{\boldsymbol{z}}_i$, which is equivalent to a node embedding operation in a graph neural network. Then each pair $(\hat{\boldsymbol{z}}_k, \hat{\boldsymbol{z}}_i)$ is processed by another MLP g, which corresponds to a node-to-edge operation by encoding the interaction between object k and object i in the embedding $\boldsymbol{\xi}_{k,i} \in \mathbb{R}^{d_\xi}$, where $d_\xi \in \mathbb{N}$ is the embedding dimension. Note that the computed embedding is directional. Finally, an edge-to-node operation is performed, where the effect on the object k is calculated by summing the individual effects $g_{\text{eff}}(\boldsymbol{\xi}_{k,i}) \in \mathbb{R}^{d_e}$ of all other objects i on the particular object k. Note that the sum is weighted by an attention coefficient $g_{\text{att}}(\boldsymbol{\xi}_{k,i}) \in \mathbb{R}^{d_e}$, where d_e is a positive integer, which allows each individual object to consider only particular interactions. This technique also yields better combinatorial generalization to a larger number of objects, as it controls the magnitude of the sum.

Relational Memory Core We compare the effects modeled by IN to the effects learned by a Relational Memory Core (RMC), $\gamma_t = \Gamma^{RMC}(\mathbf{z}_{t-1})$. RMC (Figure 2.2) learns to compartmentalize objects into memory slots, and can keep the state of an object and combine this information with the current object's representation $\mathbf{z}_t$. This is achieved by borrowing ideas from memory-augmented networks [Sukhbaatar et al., 2015; Graves et al., 2014, 2016] and interpreting memory slots as object representations. The interactions between objects are then computed by a multi-head self-attention mechanism [Vaswani et al., 2017]. Finally, recurrence for sequential interactions is introduced, resulting in an architecture that is similar to a 2-dimensional LSTM[Hochreiter and Schmidhuber, 1997], where rows of the memory matrix represent objects. The model parameters are shared for each object, so the number of memory slots can be changed without affecting the total number of model parameters. For a full description, we refer to previous work [Santoro et al., 2018].

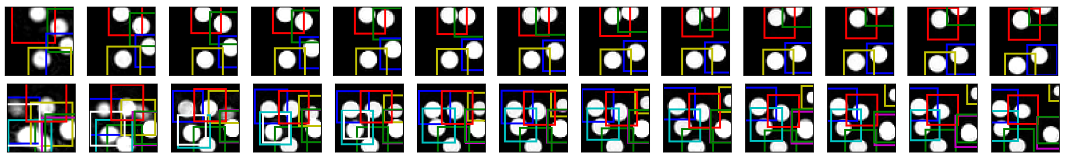


Figure 2.3: R-SQAIR trained on sequence of 4 bouncing balls (top rows) and evaluated on 6-8 bouncing balls.

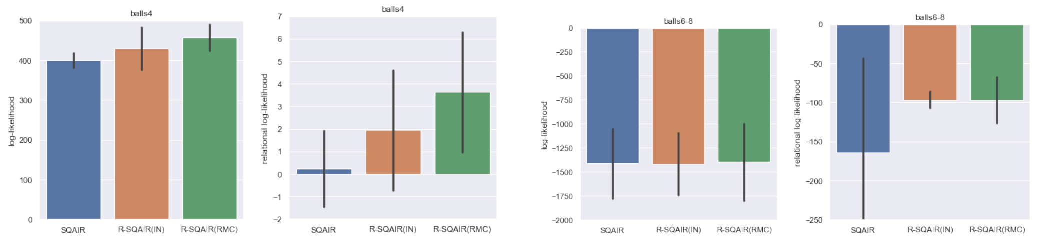


Figure 2.4: Log-likelihood and relational log-likelihood of R-SQAIR and SQAIR on the bouncing balls task.

2.2 Experiments

We analyze the physical reasoning capabilities of R-SQAIR on the *bouncing balls* dataset, which consists of video sequences of 64x64 images. As done in SQAIR experiments, we crop the central 50x50 pixels from the image, such that a ball can disappear and later reappear. Although visually simple, this data set contains highly complex physical dynamics and has previously been used for similar studies (R-NEM [van Steenkiste et al., 2018]). The method is trained in a SQAIR-like fashion by maximizing the importance-weighted evidence lower bound IWAE [Burda et al., 2015], with 5 particles and the batch size of 32. Curriculum learning starts at sequence length 3 which is increased by one every 10000 iterations, up to a maximum length of 10. Early stopping is performed when the validation score has not improved for 10 epochs.

The qualitative evaluation of R-SQAIR is present in Figure 2.3. Each column represents one time step in the video. The first row is about the R-SQAIR model trained and evaluated on videos with 4 balls, with object representations highlighted by different color bounding boxes. In the second row, the same model is evaluated on datasets with 6-8 balls. Note that R-SQAIR disentangles objects already in the first few frames and later only refines the learned representations. At each time step, it computes up to $k = 4$ object representations, by considering objects from the previous frame and the learned dynamics.

For all SQAIR hyperparameters, we use default values, except for the dimensionality of latent variable $\mathbf{z}_{\mathrm{what}}$, which is set to 5 instead of 50. This reflects the low visual complexity of individual objects in the scene. For similar reasons, the embedding dimensionality of IN that we use is also set to 5. We use a version of the IN module with attention coefficients to compute the weighted sum of the effects. In total, this adds 9 389 parameters to the 2 726 166 of the default SQAIR implementation. It also suggests that improved performance is a result of learning a better *propagation prior* rather than just increasing the number of model parameters.

RMC has more hyperparameters to choose from. We use self-attention with 4 heads, each of dimensionality 10. The number of memory slots is 4 and coincides with the total number of sequential attention steps we perform. Finally, RMC can perform several computations of attention per time step, where each corresponds to a message passing phase. As we are interested only in collisions, we computed attention only once per time step. This results in 98 880 parameters. Comparing the size of the SQAIR model, we obtain a conclusion similar to the one for the case of IN.

Note that the last frames in Figure 2.3 are sampled from the learned propagation prior. This enables us to evaluate the role of the relational module, as it is responsible for learning the object dynamics. Moreover, as the models are stochastic, we train 5 models for each architecture and sample 5 different last frames. We compare models in terms of log-likelihood of data and relational log-likelihood, which takes into account *only* the objects that are *currently colliding* (ground truth available in the dataset). The evaluation on the test set with 4 balls shows an increase in average data log-likelihood from 399.5 achieved by SQAIR (0.21 relational) to 429.2 by R-SQAIR(IN) (relational 1.95) and 457.32 by R-SQAIR(RMC) (relational 3.62). The error bars in Figure 2.4 represent the standard deviation of the stochastic samples from the trained models.

We test the generalization of R-SQAIR by evaluating the models trained on sequences with 4 balls on a test set with videos of 6-8 balls. Both qualitative (Figure 2.3 bottom row) and quantitative results show that R-SQAIR is capable of generalizing, with an increase in the relational log-likelihood from -164.1 achieved by SQAIR to -96.7 achieved by R-SQAIR(IN) and -97 achieved by R-SQAIR(RMC). Larger margins between relational losses of R-SQAIR and SQAIR in the test set with 6-8 balls suggest higher generalization capabilities of R-SQAIR.

2.3 Conclusion and Discussion

Graph neural networks are promising candidates for combinatorial generalization, a central theme of AI research [Battaglia et al., 2018; van Steenkiste et al., 2019]. We show that a sequential attention model can benefit from incorporating an explicit relational module, which infers pairwise object interactions in parallel. Without retraining, the model generalizes to scenarios with more objects. Its learned generative model is potentially useful as part of a world simulator [Schmidhuber, 1990b; Schmidhuber, 1990; Ha and Schmidhuber, 2018; Watters et al., 2019a].

At the time of writing this thesis, the significance of this chapter is more historical, rather than presenting a model that is still the state-of-the-art model on some dataset. It shows where the field of object-centric representation learning was only a few years ago. Since then, we have gone from visually simple grayscale scenes to real-world scenes [Elsayed et al., 2022; Seitzer et al., 2022; Zadaianchuk et al., 2023]. However, the algorithmic findings of our paper stood the test of time. Current SotA models for prediction infer pairwise object interactions in parallel, typically using Transformers.

Chapter 3

Learning Hierarchical Object Representations

Common-sense physical reasoning in the real world involves making predictions from complex high-dimensional observations. Humans somehow discover and represent abstract objects to compactly describe complex visual scenes in terms of 'building blocks' that can be processed separately [Spelke and Kinzler, 2007]. They model the world by reasoning about dynamics of high-level objects such as footballs and football players and the consequences of their interactions. It is natural to expect that artificial agents operating in the real world will benefit from a similar approach [Lake et al., 2015; Greff et al., 2020].

Real world objects vary greatly in terms of their properties. This complicates modelling their dynamics. Some have deformable shapes, e.g., clothes, or consist of parts that support a variety of complex behaviors, e.g., arms and fingers of a human body. Many objects can be viewed as a *hierarchy* of parts that locally behave somewhat independently, but also act more globally as a single whole [Mrowca et al., 2018; Lingelbach et al., 2020]. This suggests to simplify models of object dynamics by explicitly distinguishing multiple levels of abstraction, separating hierarchical sources of influence.

Prior approaches to common-sense physical reasoning explicitly consider objects and relations at a representational level, e.g., [Chang et al., 2017; Battaglia et al., 2016; van Steenkiste et al., 2018; Kipf et al., 2018]. They decompose complex physical interactions in the environment into pairwise interactions between objects, modelled efficiently by Graph Networks [Battaglia et al., 2018]. Here the representation of each object is updated at each time step by propagat-

This chapter is based on "Hierarchical Relational Inference" [Stanić et al., 2021] paper, which was published at AAAI 2021.

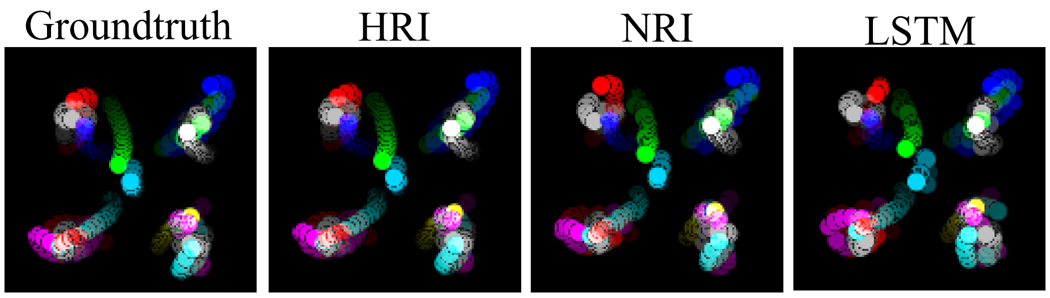


Figure 3.1: HRI outperforms baselines at modeling interacting objects that are coupled via hierarchically organized springs.

ing 'messages' through the corresponding interaction graph. While recent approaches (specifically) address the challenge of learning object representations from raw visual data [Greff et al., 2017; Kosiorek et al., 2018; van Steenkiste et al., 2018; Burgess et al., 2019; Greff et al., 2019] and of dynamically inferring relationships between objects [van Steenkiste et al., 2018; Kipf et al., 2018; Goyal et al., 2019; Veerapaneni et al., 2020], reasoning about the dynamics and interactions of complex objects remains difficult without incorporating additional structure. On the other hand, approaches that consider part-based representations of objects and hierarchical interaction graphs lack the capacity to learn from raw images and dynamically infer relationships [Mrowca et al., 2018; Lingelbach et al., 2020].

Here we propose *Hierarchical Relational Inference* (HRI), a novel approach to common-sense physical reasoning capable of learning to discover objects, parts, and their relations, directly from raw visual images in an unsupervised fashion. HRI extends *Neural Relational Inference* (NRI) [Kipf et al., 2018], which infers relations between objects and models their dynamics while assuming access to their state (e.g., obtained from a physics simulator). HRI improves upon NRI in two regards. Firstly, it considers part-based representations of objects and infers *hierarchical* interaction graphs to simplify modeling the dynamics (and interactions) of more complex objects. This necessitates a more efficient message-passing approach that leverages the hierarchical structure, which we will also introduce. Secondly, it provides a mechanism for applying NRI (and thereby HRI) to raw visual images that infers part-based object representations spanning multiple levels of abstraction.

Our main contributions are as follows: (i) We introduce HRI, an end-to-end approach for learning hierarchical object representations and their relations directly from raw visual input. (ii) It includes novel modules for extracting a hierarchical part-based object representations and for hierarchical message passing.

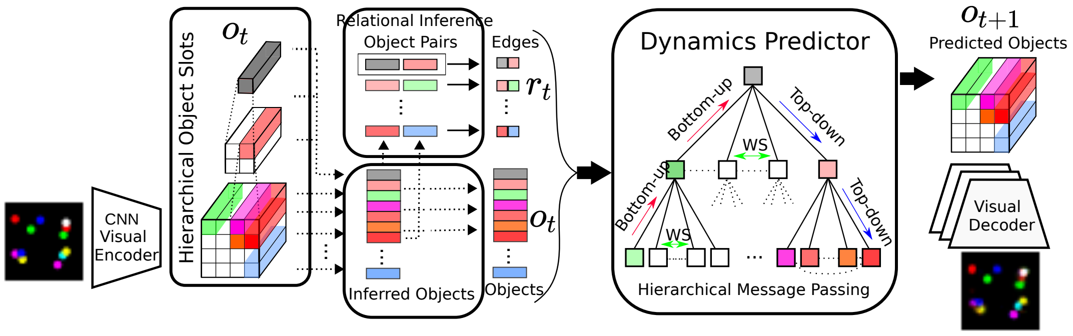


Figure 3.2: The proposed HRI model. An encoder infers part-based object representations, which are fed to a relational inference module to obtain a hierarchical interaction graph. A dynamics predictor uses hierarchical message-passing to make predictions about future object states. Their 'rendering', produced by a decoder, is compared to the next frame to train the system.

The latter can operate on a (hierarchical) interaction graph more efficiently by propagating effects between all nodes in the graph in a single message-passing phase. (iii) On a trajectory prediction task from object states, we demonstrate that the hierarchical message passing module is able to discover the latent hierarchical graph and greatly outperforms strong baselines (Fig. 3.1). (iv) We also demonstrate how HRI is able to infer objects and relations directly from raw images. (v) We apply HRI to synthetic and real-world physical prediction tasks, including real-world videos of moving human bodies, and demonstrate improvements over strong baselines.

3.1 Method

Motivated by how humans learn to perform common-sense physical reasoning, we propose Hierarchical Relational Inference (HRI). It consists of a *visual encoder*, a *relational inference module*, a *dynamics predictor*, and a *visual decoder*. All are trained end-to-end in an unsupervised manner. First, the visual encoder produces hierarchical (i.e. part-based) representations of objects that are grounded in the input image. This representation serves as input to the relational inference module, which infers pairwise relationships between objects (and parts) given by the edges in the corresponding interaction graph. The dynamics predictor performs hierarchical message-passing on this graph, using the learned representations of parts and objects for the nodes. The resulting predictions (based on the updated representations) are then decoded back to image space using the visual decoder, to facilitate an unsupervised training objective. An overview is shown in Figure 3.2. We note that HRI consists of standard building blocks (CNNs, RNNs, and GNNs) that are well understood. In this way, we add only a minimal inductive bias, which helps facilitate scaling to more complex real-world visual settings, as we will demonstrate.

3.1.1 Inferring Objects, Parts, and their Relations

To make physical predictions about a stream of complex visual observations, we will focus on the underlying *interaction graph*. It distinguishes objects or parts (corresponding to nodes) and the relations that determine interactions between them (corresponding to the edges), which must be inferred. Using this more abstract (neuro-symbolic) representation of a visual scene allows us to explicitly consider certain invariances when making predictions (e.g., the number of objects present) and reasoning about complex interactions in terms of simpler pair-wise interactions between objects.

Inferring Object/Part Representations The task of the visual encoder is to infer separate representations for each object from the input image. Intuitively, these representations contain information about its state, i.e., its position, behavior and appearance. In order to relate and compare these representations efficiently, it is important that they are described in a common format. Moreover, since we are concerned with a hierarchical (i.e. part-based) representation of objects, we also require a mechanism to relate the part representations to the corresponding object representation.

Here we address these challenges by partitioning the feature maps learned by a CNN according to their spatial coordinates to obtain object representations. This is a natural choice, since CNNs are known to excel at representation learning for images [Ciresan et al., 2012; Krizhevsky et al., 2012] and because weight-sharing across spatial locations ensures that the resulting object representations are described in a common format. Indeed, several others have proposed to learn object representations in this way [Santoro et al., 2017; Zambaldi et al., 2019]. Here, we take this insight a step further and propose to learn hierarchical object representations in a similar way. In particular, we leverage the insight that the parts belonging to real-world objects tend to be spatially close, to apply a sequence of convolutions followed by down-sampling operations to extract object-level representations from part-level representations (left side of Figure 3.2). While this leaves the network with ample freedom to develop its own internal notion of an object, we find that representational slots learn to describe physical objects (Fig. 3.4a).

The choice of kernel-size and degree of down-sampling allow us to adjust how representations at one level of abstraction are combined at the next level. Similarly, the parameters of the CNN layers that produce the initial set of feature maps determine the size of the input region captured by these 'spatial slots' [Greff et al., 2020] at the lowest level of abstraction. Note the distinction between *parts*, *objects* and *slots*. Parts refer to objects at the lowest level of the visual hierarchy, while the more general notion of an object applies to nodes at all levels. Slots are variable placeholders (of a function) at a representational level, which at each point in time are expected to contain information about a particular object/part. Therefore, an architectural hierarchy of slots reflects a hierarchy of objects. In general, we construct a 3-level part-based hierarchy, which is then fed into the *relational module*.

Neural Relational Inference To infer relations between object representations, we will make use of NRI [Kipf et al., 2018], which learns *explicit* relations. This is advantageous, since it allows one to incorporate prior beliefs about the overall connectivity of the interaction graph (e.g., a degree of sparsity) and associate a representation with each relation to distinguish between multiple different relation types. By default, NRI takes as input a set of object trajectories (states) and infers their pairwise relations (edges) using a Graph Neural Network (GNN) [Scarselli et al., 2009; Gilmer et al., 2017; Battaglia et al., 2018]. It assumes a static interaction graph, and performs relational inference by processing the entire input sequence at once, i.e., the whole sequence (length 50) of a particular object is concatenated, and only a single, static, "node embedding" is created via an MLP. In contrast, we will consider a dynamic interaction graph, since objects move across the image and may end up in different spatial slots throughout the sequence. This is achieved by inferring edges at each time step based on the ten most recent object states, concatenating the latent vectors of a particular object and using an MLP to obtain a "node embedding".

More formally, given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes (objects) $o \in \mathcal{V}$ and edges (relations) $r_{i,j} = (o_i, o_j) \in \mathcal{E}$, NRI defines a single node-to-node message passing operation in a GNN similar to [Gilmer et al., 2017]:

$$\mathbf{e}_{i,j} = f_e([\mathbf{o}_i, \mathbf{o}_j, \mathbf{r}_{i,j}]),\ \mathbf{o}'_j = f_o([\textstyle\sum_{i \in \mathcal{N}_{\mathbf{o}_j}} \mathbf{e}_{i,j}, \mathbf{o}_j]) \tag{3.1}$$

where $\mathbf{e}_{i,j} \in \mathbb{R}^{d_e}$ is an embedding (effect) of the relation $\mathbf{r}_{i,j} \in \mathbb{R}^{d_r}$ between objects $\mathbf{o}_i \in \mathbb{R}^{d_o}$ and $\mathbf{o}_j \in \mathbb{R}^{d_o}$, $\mathbf{o}'_j \in \mathbb{R}^{d_o}$ is the updated object embedding, $\mathcal{N}_j$

the set of indices of nodes connected by an incoming edge to object $\mathbf{o}_j \in \mathbb{R}^{d_o}$ and $[\cdot, \cdot]$ indicates concatenation, and d_e, d_r, d_o positive integers. Functions f_o and f_e are node- and edge-specific neural networks (MLPs in practice). By repeatedly applying equation 3.1, multiple rounds of message passing can be performed.

The NRI 'encoder' receives as input a sequence of object state trajectories $\mathbf{o} = (\mathbf{o}^1, ..., \mathbf{o}^T)$, which in our case are inferred. It consists of a GNN f_ϕ that defines a probability distribution over edges $q_\phi(\mathbf{r}_{ij}^t | \mathbf{o}^{t-k:t}) = \mathrm{softmax}(f_\phi(\mathbf{o}^{t-k:t})_{ij})$, where k is the window size, and relations are one-hot encoded. The GNN performs the following message passing operations, where the initial node representations $\mathbf{o}_i$ are obtained by concatenating the corresponding object states across the window:

$$\begin{aligned}
&\mathbf{o}'_j = f_o^1(\mathbf{o}_j),\ \mathbf{e}'_{i,j} = f_e^1([\mathbf{o}'_i, \mathbf{o}'_j]),\ \mathbf{o}''_j = f_o^2(\textstyle\sum_{i \neq j} \mathbf{e}'_{i,j}),\\
&\mathbf{e}''_{i,j} = f_e^2([\mathbf{o}''_i, \mathbf{o}''_j]),\ f_\phi(\mathbf{o}^{t-k:t})_{ij} = \mathbf{e}''_{i,j}
\end{aligned}$$

where ϕ contains the parameters of the message-passing functions, which are simple MLPs, and o', e' and o'', e'' are node- and edge-embeddings after first and second message passing operations respectively. To backpropagate through the sampling from $q_\phi(\mathbf{r}_{ij} | \mathbf{o})$, NRI uses a continuous approximation of the discrete distribution to obtain gradients via the reparameterization trick [Maddison et al., 2017; Jang et al., 2017].

3.1.2 Physical Reasoning

Physical reasoning is performed by the *dynamics predictor*, which leverages the inferred object representations and edges to predict object states at the next time step. To distinguish between representations at different levels of abstractions, HRI makes use of *hierarchical message passing*. We will also use recurrent units in the non-Markovian setting.

Hierarchical message passing The default approach to physical reasoning based on message-passing employed in NRI can only propagate effects between directly connected nodes. This is costly, as it requires several iterations for information to propagate across the whole graph, especially when the number of nodes increases (a consequence of modeling objects as hierarchies of parts). Alternatively, we can leverage the hierarchical structure of the interaction graph to propagate all effects across the entire graph in a *single* step, i.e. evaluating each relation only once. To achieve this, we introduce a hierarchical message passing module which propagates effects between objects using a three-phase sequential mechanism, loosely inspired by Mrowca et al..

Starting from the leaf nodes, the *bottom-up phase* computes the effect on parent nodes $\mathbf{o}_p$ based on messages from its children, $\mathbf{e}_p^1 = \mathbf{e}_p^0 + f_{MP}^{bu}(\{\mathbf{e}_c^0\}_{c\in\mathcal{C}_p}, \mathbf{e}_p^0, \{\mathbf{r}_{cp}\}_{c\in\mathcal{C}_p})$ where $\mathcal{C}_p$ is the set of children indices of object $\mathbf{o}_p$ and the initial effects $\mathbf{e}^0$ are simply the object embeddings. In this way, global information is propagated from every node in the hierarchy to the root node. Afterwards, the bottom-up effect $\mathbf{e}_i^1$ on node $\mathbf{o}_i$ is combined with effects from its siblings (*within-sibling* phase) $\mathbf{e}_i^2 = \mathbf{e}_i^1 + f_{MP}^{ws}(\{\mathbf{e}_s^1\}_{s\in\mathcal{S}_i}, \mathbf{e}_i^1, \{\mathbf{r}_{si}\}_{s\in\mathcal{C}_i})$, where $\mathcal{S}_i$ is the set of sibling indices of object $\mathbf{o}_i$. Starting from the root node, the *top-down* phase then propagates top-down effects that are incorporated by computing $\mathbf{e}_c^3 = \mathbf{e}_c^2 + f_{MP}^{td}(\mathbf{e}_p^2, \mathbf{e}_c^2, \mathbf{r}_{pc})$ for all children $\mathbf{o}_c$ based on its parent $\mathbf{o}_p$. Functions f_{MP}^{bu}, f_{MP}^{ws}, and f_{MP}^{td} perform a single node-to-edge and edge-to-node message passing operation as in equation 3.1 and have shared weights. Note that this mechanism is independent of the choice of object and relational inference module and can act on any hierarchical interaction graph.

Dynamics predictor Physical reasoning is performed by the dynamics predictor, which predicts future object states $p_\theta(\mathbf{o}^{t+1}|\mathbf{o}^{1:t}, \mathbf{r}^{1:t})$ from the sequence of object states and interactions. We implement this as in the NRI 'decoder' [Kipf et al., 2018], i.e. using a GNN that passes messages between objects, but with two notable differences. Firstly, we will pass messages *only* if an edge is inferred between two nodes, as opposed to also considering a separate dynamics predictor for the "non-edge" relation that causes information exchange between unconnected nodes[1]. Secondly, we will leverage the hierarchical structure of the inferred interaction graph to perform *hierarchical message-passing*.

If we assume Markovian dynamics, then we have $p_\theta(\mathbf{o}^{t+1}|\mathbf{o}^{1:t}, \mathbf{r}^{1:t}) = p_\theta(\mathbf{o}^{t+1}|\mathbf{o}^{t}, \mathbf{r}^{t})$ and can use hierarchical message passing to predict object states at the next step:

$$p_\theta(\mathbf{o}^{t+1}|\mathbf{o}^{t}, \mathbf{r}^{t}) = \mathcal{N}(\mathbf{o}^t + \Delta\mathbf{o}^t, \sigma^2\mathbf{I}), \tag{3.2}$$

where σ^2 is a fixed variance, $\Delta\mathbf{o}^t = f_O([\mathbf{o}^t, \mathbf{e}^t])$, $\mathbf{e}^t = f_H(\mathbf{o}^t, \mathbf{r}^t)$ is the effect computed by the hierarchical message passing module f_H, and f_O is an output MLP. Notice how we learn to predict the *change* in the state of an object, which is generally expected to be easier. When encoding individual images, no velocity information can be inferred to form the object state. In this non-Markovian case

[1]Although messages are passed only if an edge between two nodes exists, a "non-edge" categorical variable is used to allow the model to infer that there is no edge between two nodes.

we adapt equation 3.2 to include an LSTM [Hochreiter and Schmidhuber, 1997] that models $p_\theta(\mathbf{o}^{t+1}|\mathbf{o}^{1:t}, \mathbf{r}^{1:t})$ directly:

$$\mathbf{h}^{t+1}, \mathbf{c}^{t+1} = f_{LSTM}(\mathbf{o}^t, \mathbf{e}^t, \mathbf{c}^t), \quad \mathbf{o}^{t+1} = f_O(\mathbf{h}^{t+1}),$$
$$p(\mathbf{o}_j^{t+1}|\mathbf{o}^{1:t}, \mathbf{r}^{1:t}) = \mathcal{N}(\mathbf{o}^{t+1}, \sigma^2 \mathbf{I}),$$

where $\mathbf{c}$ and $\mathbf{h}$ are LSTM's cell and hidden state respectively, and $\mathbf{e}^t = f_H(\mathbf{h}^t, \mathbf{r}^t)$.

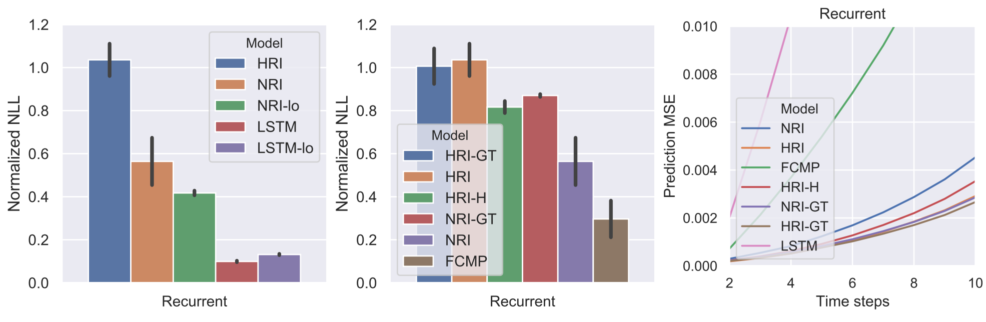
3.1.3 Learning

Standard approaches to modelling physical interactions that do not assume access to states uses a prediction objective in pixel space [van Steenkiste et al., 2018; Veerapaneni et al., 2020]. This necessitates a mechanism to 'render' the updated object representations. In this case, HRI can be viewed as a type of Variational Auto-Encoder [Kingma and Welling, 2013; Rezende et al., 2014], where the inferred edges and objects are treated as latent variables, and the ELBO can be maximized for the predicted frames:

$$\begin{aligned}\mathcal{L} =& \mathbb{E}_{q_\phi(\mathbf{r}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{r}, \mathbf{o})] - D_{\mathrm{KL}}[q_{\phi_o}(\mathbf{o}|\mathbf{x})||p_{\theta_o}(\mathbf{o})] \\ &- D_{\mathrm{KL}}[q_{\phi_r}(\mathbf{r}|\mathbf{x}, \mathbf{o})||p_{\theta_r}(\mathbf{r})].\end{aligned} \tag{3.3}$$

The relational module $q_{\phi_r}(\mathbf{r}|\mathbf{x}, \mathbf{o})$ outputs a factorized distribution over $\mathbf{r}_{ij}$, which in our case is a categorical variable that can take on two values (one-hot encoded) that indicate the presence of an edge between $\mathbf{o}_i$ and $\mathbf{o}_j$. The edge prior $p_{\theta_r}(\mathbf{r}) = \prod_{i \neq j} p_{\theta_r}(\mathbf{r}_{ij})$ is a factorized uniform distribution, which controls the sparsity of the learned graph. The object inference module $q_{\phi_o}(\mathbf{o}|\mathbf{x})$ outputs a factorized distribution over $\mathbf{o}_i$, and the object prior $p_{\theta_o}(\mathbf{o})$ is Gaussian, as in a standard VAE. Given the inferred interaction graph, the dynamics predictor and visual decoder define $p_\theta(\mathbf{x}|\mathbf{r}, \mathbf{o})$.

Visual Decoder The visual decoder renders the updated object states and we will consider two different implementations of this mapping. The first variant, which we call *SlotDec*, ensures compositionality in pixel space by decoding objects separately followed by a summation to produce the final image. In Figure 3.2 it is depicted as a set since individual object slots that are decoded separately, and decoders share weights. This implements a stronger inductive bias that encourages each slot to correspond to a particular object (since images are composed of objects) and also makes it easy to inspect the representational content of each slot. On the other hand, summation in pixel space is problematic when objects in scenes are more cluttered and occlude one another. For



(a) *(b)* *(c)*

Figure 3.3: Performance on 4-3-state-springs. We compare HRI to (a) baselines and (b) ablations in terms of the "normalized" negative log likelihood (higher is better). (c) MSE for future prediction (prediction rollouts).

this reason we implement a second variant, which we call *ParDec*, where all states are decoded together as in a standard convolutional decoder. As a result of decoding all object slots together *ParDec* is less interpretable than *SlotDec*, but computationally more efficient and potentially more scalable to real-world datasets since it does not make strong assumptions on how information about objects should be combined. This may also make it easier to handle background, although this is not explored.

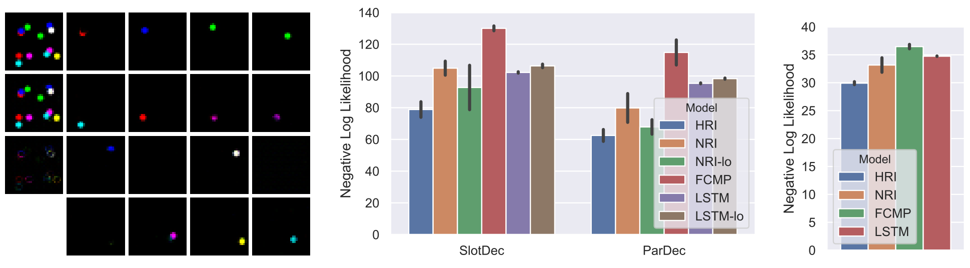
3.2 Related Work

More generic approaches to future frame prediction are typically based on RNNs, which are either optimized for next-step prediction directly [Srivastava et al., 2015; Lotter et al., 2017], e.g. using a variational approach [Babaeizadeh et al., 2018; Denton and Fergus, 2018; Kumar et al., 2020], or (additionally) require adversarial training [Lee et al., 2019; Vondrick et al., 2016]. Several such approaches were also proposed in the context of physical reasoning [Finn et al., 2016; Lerer et al., 2016; Li et al., 2016]. However, unlike our approach, they do not explicitly distinguish between objects and relations. This is known to affect their ability to accurately model physical interactions and extrapolation [van Steenkiste et al., 2018; Garnelo and Shanahan, 2019], despite their remarkable capacity for modeling more complex visually scenes.

More closely related approaches to physical prediction explicitly distinguish object representations and incorporate a corresponding relational inductive bias, typically in the form of a graph network [Battaglia et al., 2018]. In this case, reasoning is based on message-passing between object states and relations are either inferred heuristically [Chang et al., 2017; Mrowca et al., 2018; Lingelbach et al., 2020], implicitly by the message passing functions [Sukhbaatar et al., 2015; Battaglia et al., 2016; Santoro et al., 2017; Watters et al., 2017; Sanchez Gonzalez et al., 2018; Sanchez-Gonzalez et al., 2020] e.g. via attention [Hoshen, 2017; van Steenkiste et al., 2018], or explicitly as in NRI [Kipf et al., 2018]. Typically, these approaches assume access to supervised state descriptions, and only few works also infer object representations from raw visual images [van Steenkiste et al., 2018; Veerapaneni et al., 2020; Watters et al., 2019a]. RIMs [Goyal et al., 2019] impose a modular structure into an RNN, whose sparse communication can be seen as a kind of relational inference. However, neither of these additionally incorporate a hierarchical inductive bias to cope with more complex physical interactions (and corresponding objects). Arguably, this is due to the difficulty of inferring corresponding part-based object representations, and indeed prior approaches that do incorporate a hierarchical inductive bias rely on supervised state descriptions [Mrowca et al., 2018; Lingelbach et al., 2020].

Related approaches that specifically focus on the issue of learning object representations from raw visual observations typically make use of mixture models [Greff et al., 2017, 2019] or attention [Eslami et al., 2016; Kosiorek et al., 2018; Stanić and Schmidhuber, 2019; Burgess et al., 2019; Crawford and Pineau, 2019; Jiang* et al., 2020]. In our case, we adopt an approach based on spatial slots [van Steenkiste et al., 2019; Greff et al., 2020] that puts less emphasis on the exact correspondence to objects. Unlike prior work [Santoro et al., 2017; Zambaldi et al., 2019], we demonstrate how spatial slots can be extended to a hierarchical setting.

Hierarchical structure learning has previously been explored in vision. In this case, hierarchies are either learned from 3D volumetric primitives [Tulsiani et al., 2017], or using ground truth pixel-wise flow fields to guide object or object parts segmentation [Liu et al., 2019]. In concurrent work [Mittal et al., 2020], RIMs were extended to consider a hierarchy of modules, although their correspondence to perceptual objects (and parts) is unclear. A more relevant approach was recently proposed by [Deng et al., 2019], which is able to infer a hierarchy of objects and parts, directly from raw visual images. However, it was not explored how this approach can be adapted to video, and how these abstractions can be used for physics prediction.



(a) (b) (c)

Figure 3.4: (a) HRI introspection: first column contains ground-truth, prediction and their difference. The other columns show the 16 object slots decoded separately. (b) Negative log likelihood for all models on the 4-3-visual-springs and (c) Human3.6M.

3.3 Experiments

In this section we evaluate HRI on four different dynamics modelling tasks: state trajectories of objects connected via finite-length springs in a hierarchical structure (*state-springs*); corresponding rendered videos (*visual-springs*); rendered joints of human moving bodies (*Human3.6M*); and raw videos of moving humans (*KTH*). We compare HRI to NRI [Kipf et al., 2018], which performs relational inference but lacks a hierarchical inductive bias, and to an LSTM [Hochreiter and Schmidhuber, 1997] that concatenates representations from all objects and predicts them jointly, but lacks a relational inference mechanism altogether. Appendix A contains all experimental details. Reported results are mean and standard deviations over 5 seeds.

3.3.1 State Springs Dataset

We consider synthetic physical systems containing simple objects connected via finite-length springs that can be organized according to a hierarchical interaction graph (Figure 3.5, middle row). Here, an approach that attempts to model the underlying system dynamics (which are highly complex) would clearly benefit from a hierarchical inductive bias, which allows us to validate our design choices. In all configurations we consider hierarchies with 3 levels (containing a root node, intermediate nodes, and leaf nodes), whose connectivity is decided randomly while ensuring that all nodes are connected. Corresponding physical objects are initialized at random locations with the root node biased towards the center and the intermediate and leaf nodes towards a particular quadrant to reduce clutter (see also Appendix A).

We experiment with hierarchies containing 4 intermediate nodes, each having 3 or 4 leaf nodes, denoted as *4-3-state-springs* and *3-3-state-springs*, respectively. Each model receives as input the 4-dimensional state trajectories: $x(t), y(t), \Delta x(t), \Delta y(t)$.

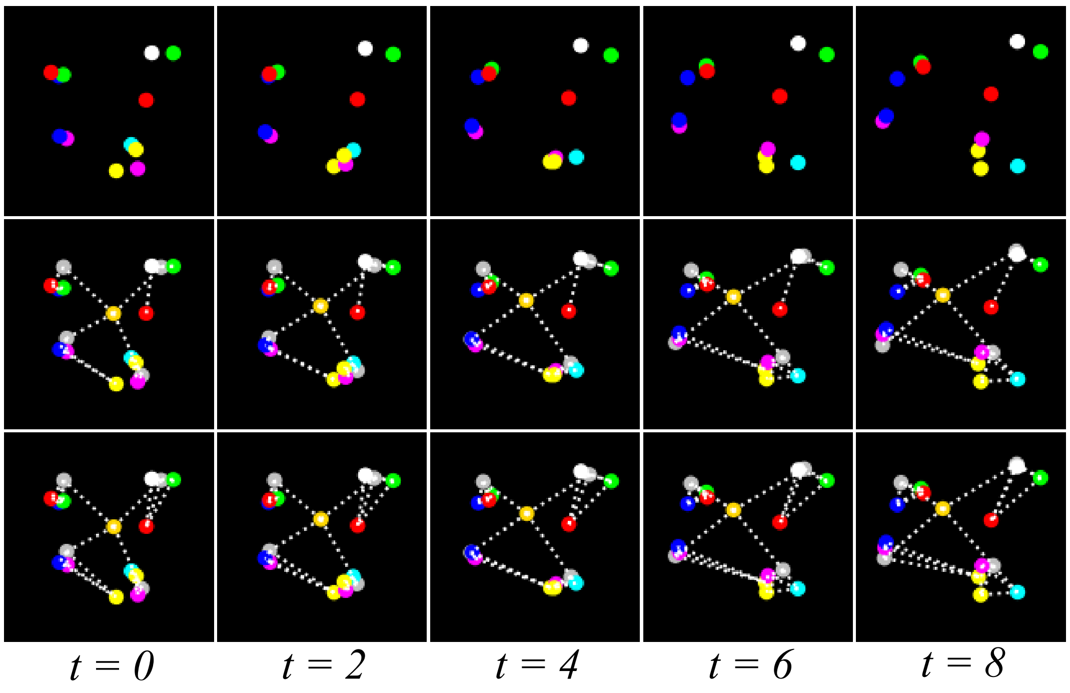


Figure 3.5: Rendered inputs for *4-3-state-springs* (leaf objects only) (top); full interaction graph, unobserved by model (middle); Predictions and inferred edges by HRI (bottom).

Comparison to baselines We compare HRI to NRI and LSTM on *4-3-state-springs* (Figure 3.3a) and *3-3-state-springs* (Figure 7a in Appendix B), in terms of the negative log likelihood inversely proportional to a version of HRI that operates on the ground-truth interaction graph (HRI-GT)[2]. In this case, values closer to 1.0 are better, although we also provide raw negative log likelihoods in Figure A.2 which offers the same conclusions. It can be observed that HRI markedly outperforms NRI on this task, and that both significantly improve over the LSTM (which was expected). These findings indicate that the hierarchical inductive bias in HRI is indeed highly beneficial for this task.

[2]This allows us to factor out the complexity of the task and make it easier to compare results *between* tasks.

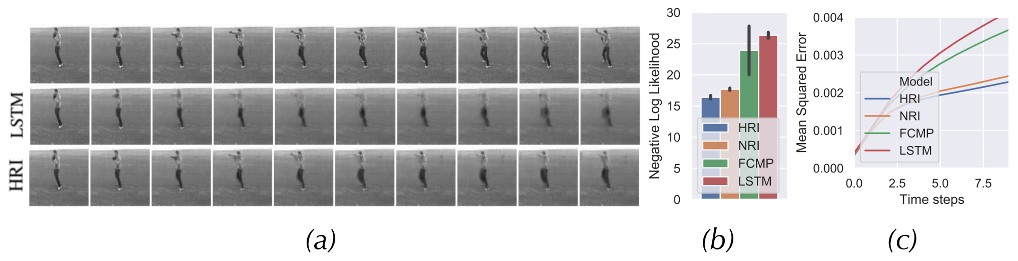


Figure 3.6: (a) Ground truth (top) and predicted 10 time steps rollout for LSTM (middle) and HRI (bottom) row on KTH. (b) Negative log likelihood for all models on KTH. (c) Rollout predictions accuracy for all models on KTH.

We also compare to baselines that focus only on the leaf objects (*NRI-lo* and *LSTM-lo*) and are therefore not encouraged to separate hierarchical sources of influence (i.e. we impose less structure). It can be observed that this does not result in better predictions and the gap between NRI-lo and NRI indicates that explicitly considering multiple levels of abstraction, as in HRI, is indeed desirable.

Ablation & Analysis We conduct an ablation of HRI, where we modify the relational inference module. We consider FCMP, a variation of NRI that does not perform relational inference but assumes a fixed fully connected graph, and HRI-H, which is given knowledge about the 'valid' edges in the ground-truth hierarchical graph to be inferred and performs relational inference only on those.

In Figure 3.3b and Figure 7b (Appendix B) it can be observed how the lack of a relational inference module further harms performance. Indeed, in this case it is more complex for FCMP to implement a similar solution, since it requires "ignoring" edges between objects that do not influence each other. It can also be observed how HRI outperforms HRI-H (but see Figure 7b where they perform the same), which is surprising since the latter essentially solves a simpler task. We speculate that training interactions could be responsible for this gap. Switching to static graph inference yielded a significant drop in performance for NRI ($\sim 74\%$ worse), while HRI remained mostly unaffected ($\sim 5\%$ worse).

We also consider the benefit of hierarchical message-passing in isolation by comparing HRI to NRI-GT, which receives the ground-truth interaction graph. In Figure 3.3b and Figure 7b (Appendix B) it can be seen how the lack of hierarchical message-passing explains part of the previously observed gap between HRI and NRI, but not all of it. It suggests that by explicitly considering multiple levels of abstraction (as in HRI), conducting relational inference generally becomes easier on these tasks. Finally, we consider the effect of using a feedforward dynamics predictor (Figures 8 and 9 in Appendix B). It can be seen that in the feedforward case, position and velocity are not enough to derive interactions between objects. For this reason we exclusively focus on the recurrent predictor from now on.

Long-term Prediction and Graph Inference We investigate if HRI is able to perform long-term physical predictions by increasing the number of rollout steps at test-time. In Figure 3.3c we report the MSE between the predicted and ground truth trajectories and compare to previous baselines and variations. It can be observed that HRI outperforms all other models, sometimes even HRI-GT, and this gap increases as we predict deeper into the future. To qualitatively evaluate the plausibility of HRI's rollout predictions, we generated videos by rendering them. As can be seen in Figure 3.5 (bottom row) both the predicted trajectories and the inferred graph connectivity closely matches the ground-truth. Similar results are obtained for *3-3-state-springs* in Appendix B.

3.3.2 Visual Datasets

To generate visual data for springs we rendered *4-3-state-springs* and *3-3-state-springs* with all balls having the same radius and one of the 7 default colors, assigned at random[3]. Additionally, we consider *Human3.6M* [Ionescu et al., 2013] (using rendered joints) and *KTH* [Schuldt et al., 2004] (using raw videos). We train each model in two stages, which acts as a curriculum: first we train the visual encoder and decoder on a reconstruction task, afterwards we optimize the dynamics parameters on the prediction task.

Visual Springs To create the visual springs dataset, we render only the *leaf nodes*. Hence the visible balls themselves are the "parts" that must be grouped in line with more abstract "objects" corresponding to the intermediate and root nodes of the interaction graph (e.g. the gold and silver balls in the bottom row of Fig. 3.5) which are not observed.

In Figure 3.4b we compare HRI to several baselines and previously explored variations (the results for *3-3-visual-springs* are available in Figure 10 in Appendix B). We are able to observe similar results, in that HRI is the best performing model, although the added complexity of inferring object states results

[3]Similar results were obtained when using homogeneous colors (Figures 11 and 12) and when varying shape (Figures 17 to 21).

in smaller differences. This is better understood by comparing the difference between HRI and NRI[4] to the difference between HRI and NRI-lo. It can be seen that NRI-lo performs slightly worse than HRI (although it is less stable) and is much better than NRI, which suggests that inferring the intermediate and root node objects is one of the main challenges in this setting. Notice, however, that HRI continues to markedly improve over the LSTM and visual NRI in this setting. We note that comparing to NRI-GT or HRI-GT is not possible, since the mapping from learned object representations to those in the graph is unknown.

Comparing SlotDec and ParDec in Figure 3.4b, we observe that better results are obtained using the latter. We suspect that this is due to the visual occlusions that occur, which makes summation in pixel space more difficult. Nonetheless, in most cases we observe a clear correspondence between spatial slots and information about individual objects (see Figure 3.4a). Finally, Figure 13 (Appendix B) demonstrates how future predictions by HRI match the ground-truth quite well. Similarly, we observe that the inferred interaction graph by HRI resembles the ground-truth much more closely compared to NRI (Figure 15 in Appendix B).

When evaluating HRI trained on *4-3-visual-springs* on *3-3-visual-springs* (i.e. when *extrapolating*) the relative ordering is preserved (Figure 22 in Appendix B).

Real-world Data We consider *Human3.6M*, which consists of 3.6 million 3D human poses composed of 32 joints and corresponding images taken from professional actors in various scenarios. Here we use the provided 2D pose projections to render 12 joints in total (3 per limb) as input to the model. This task is significantly more complex, since the underlying system dynamics are expected to vary over time (i.e. they are non-stationary). Fig. 3.4c demonstrates the performance of HRI and several baselines on this task. Note that HRI is the best performing model, although the gap to NRI and LSTM is smaller. This can be explained by the fact that many motions, such as when sitting, eating, and waiting involve relatively little motion or only motion in a single joint (and thereby lack hierarchical interactions). Example future predictions by HRI can be seen in Figure 14.

Finally, we apply HRI directly on the *raw* videos of *KTH*, which consists of people performing one of six actions (walking, jogging, running, etc.). Figure 3.6a shows a comparison between the LSTM baseline and HRI model. In the

[4]Note that NRI requires object states. Here, we consider an adaptation using the encoder and decoder proposed for HRI.

video generated by the LSTM, the shape of the human is not preserved, while HRI is able to clearly depict human limbs. We prescribe this to the ability of HRI to reason about the objects in the scene in terms of its parts and their interactions, which in turn simplifies the physics prediction. This observation is also reflected in the quantitative analysis in Figs. 3.6b and 3.6c, where large differences can be observed, especially when predicting deeper into the future.

3.4 Conclusion and Discussion

Hierarchical Relational Inference (HRI) is a novel approach to common-sense physical reasoning capable of learning to discover objects, parts, and their relations, directly from raw visual images in an unsupervised fashion. It builds on the idea that dynamics of complex objects are best modeled as hierarchies of parts that separate different sources of influence. We compare HRI's predictions to those of a strong baseline on synthetic and real-world physics prediction tasks, where it consistently outperforms. Several ablation studies validate our design choices.

A potential limitation of HRI is that it groups parts into objects based on spatial proximity, which may be suboptimal in the case of severe occlusions. Further, it would be interesting to provide for inter-object interactions, and unknown hierarchy depths. Both might be handled by increasing the hierarchy depth and relying on the relational inference module to ignore non-existent edges.

Since we published our paper on which this chapter is based, it remains one of the few that tackles the question of learning hierarchical object-centric representations. Follow-up work, such as the GLOM model [Hinton, 2023], discusses ideas only at a conceptual level, without a concrete implementation or a working model. The lack of research on hierarchical object-centric representations is partially due to the difficulty of the problem.

Moreover, since real-world scenes can be decomposed into a large number (sometimes even intractably many) of different hierarchical levels, it might not even be necessary to go beyond the two hierarchical levels that we introduced. As long as there is a hierarchical relation describing "is part of", the level at which we represent the scene could be dynamic, i.e., based on a task, word, goal, or an action. The rest of the scene can be left unmodelled, e.g. if we are interested in counting the number of screws in a car wheel, we can safely ignore the rest of the car and represent only the wheel and its parts.

Chapter 4

Learning to Generalize with Object-centric Agents

In Chapter 3 we introduced a method (HRI) that is capable of inferring nodes and edges of a hierarchical graph directly from raw visual data. HRI models objects as hierarchies of parts that may locally behave separately but also act more globally as a single whole. HRI explicitly distinguishes multiple levels of abstraction and improves over a strong baseline in modeling synthetic and real-world videos. However, (hierarchical) decomposition into objects is generally task dependent, and sometimes it is infeasible and undesirable to decompose a scene into all hierarchy levels. For these reasons, it might be more beneficial to modulate objects with task information, such as words (in VQA) or actions/goals (in RL). In this chapter we introduce object-centric agents in an RL setup, and study their properties, especially in terms of out-of-distribution generalization.

Reinforcement learning agents must generalize beyond their training experience. Prior work has focused mostly on identical training and evaluation environments. In this work, starting from the recently introduced Crafter benchmark, a 2D open world survival game, we introduce a new set of environments suitable for evaluating some agent's ability to generalize on previously unseen (numbers of) objects and to adapt quickly (meta-learning). We introduce CrafterOOD, a set of 15 new environments that evaluate OOD generalization. On CrafterOOD, we show that the current agents fail to generalize, whereas our novel object-centric agents achieve state-of-the-art OOD generalization while also being interpretable.

This chapter is based on "Learning to Generalize with Object-centric Agents in the Open World Survival Game Crafter" [Stanić et al., 2023] paper, which was published as a journal article in IEEE Transactions on Games.

Common benchmarks and solid baselines are essential for developing new models and correctly measuring progress in machine learning. Datasets for supervised learning (such as ImageNet [Deng et al., 2009]) and game-based environments for reinforcement learning (RL) (such as Atari [Bellemare et al., 2013], ProcGen [Cobbe et al., 2020] and MineRL [Guss et al., 2019, 2021]) have played a crucial role in developing new methods and improving the state of the art. To judge the improvements offered by novel methods, it is critical to compare them to solid yet simple baselines, especially in deep RL, where reproducing existing work is often hard due to a plethora of difficulties [Henderson et al., 2018].

Early benchmarks were important to show that certain deep RL methods can learn control from high-dimensional images, e.g., neuroevolution via compressed network encodings [Koutník et al., 2013] on the TORCS benchmark [Wymann et al., 2000], or the DQN agent [Mnih et al., 2015] on the Atari benchmark [Bellemare et al., 2013]. These benchmarks, however, focused on narrow task sets, and conducted training and evaluation on in-distribution environments, where the agents could (simply) memorize sequences of actions leading to high reward without *understanding* the underlying mechanics of the world. Intelligent agents should discover salient objects (entities) of the world, identify relevant properties of objects (e.g., shape is important, color is not) and ignore irrelevant parts of the environment. For better evaluation, the focus has recently shifted to studying agent performance on carefully designed benchmarks [Osband et al., 2020], with emphasis on agents that generalize to environments beyond what they are trained on (e.g., ProcGen or Crafter [Hafner, 2021]).

Here we focus on Crafter [Hafner, 2021], a recently introduced open world survival game that allows tractable investigation of new agents and their generalization, exploration, and long-term reasoning capabilities. Compared to other benchmarks, Crafter has a set of advantages that make it suitable for RL research: fast iteration speed (agents can be trained within a few hours on a standard GPU and a single CPU core), model evaluation by inspecting semantically meaningful achievements unlocked by the agents, and controllable environmental objects that facilitate systematic studies. Our experiments on the original Crafter environment offer important, previously unpublished insights into baselines and environmental workings. In addition, we introduce two sets of new environments, CrafterOODapp and CrafterOODnum, that test out-of-distribution (OOD) generalization to unseen types and numbers of objects, respectively. This also sets the stage for developing fast-adaptation algorithms in the context of meta-learning.

Our main contributions are: (1) We show that careful hyper-parameter tuning improves the PPO baseline agent by a large margin. We achieve new state-of-

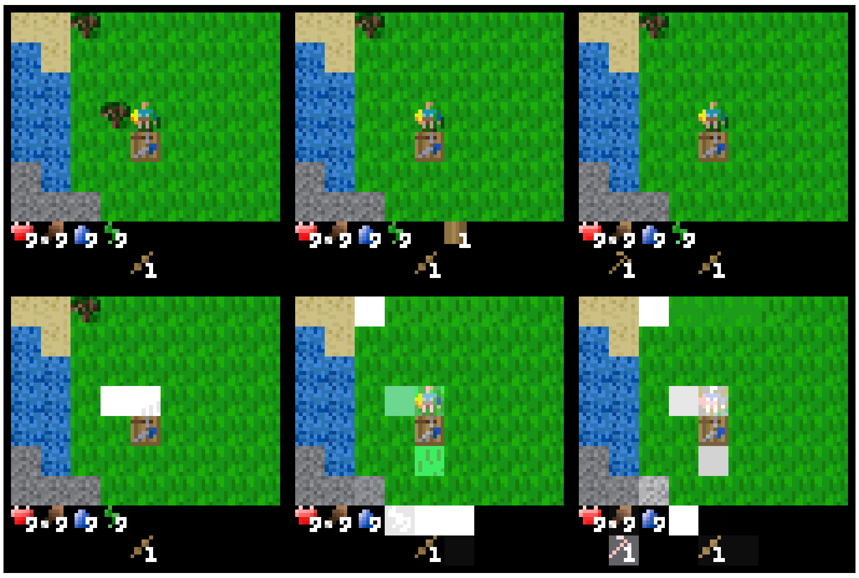


Figure 4.1: Crafter gameplay showing an agent collecting resources (wood) and crafting a weapon (wooden pickaxe), and its attention. Top row: input images. Bottom row: visualized attention. Episode steps are plotted horizontally.

the-art performance on the original Crafter environment. (2) We show that feedforward agents can unlock almost all achievements by relying on the inventory display (bottom part of the images in Figs. 4.1 and 4.2a) – a type of "scratchpad"). (3) We show that recurrent agents improve over feedforward ones, even when the inventory information is removed. (4) We show that tuned agents can unlock almost all achievements. (5) We introduce CrafterOOD, a set of 15 new environments, to evaluate OOD generalization of agents. (6) We introduce novel object-centric agents that achieve state-of-the-art OOD generalization while being interpretable.

4.1 Environments

Here we briefly summarize essential properties of the Crafter environment and the newly introduced CrafterOODapp and CrafterOODnum environments. For additional details on Crafter we refer to [Hafner, 2021].

Crafter. Crafter is an open world survival game for RL research whose game dynamics are inspired by the popular game Minecraft. The benchmark is designed to facilitate existing research challenges: procedural generation to test generalization, exploration due to a deep technology tree of achievements con-

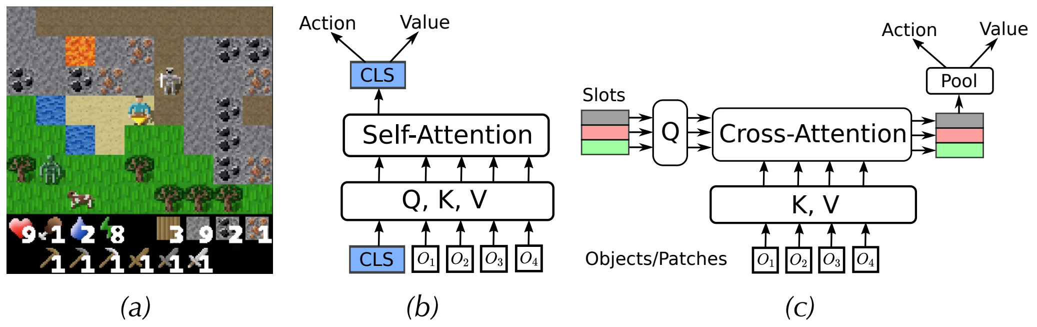


Figure 4.2: (a) Crafter gameplay. (b) Object-centric Self-Attention (OC-SA) model with a CLS token and queries (Q), keys (K) and values (V). (c) Slot-based Object-centric Cross-Attention (OC-CA) model. Pool operation averages the slots to produce the final input vector fed into the actor and critic networks.

ditioned on one another, high-dimensional image observations requiring representation learning, memory through partial observability, and finally, sparse rewards that require long-term reasoning and credit assignment. It has the right level of complexity that is challenging enough for the current agents to unlock all achievements, yet not overly complex such that it facilitates interpretability of the agent's behavior and evaluation by unlocking semantically meaningful achievements. In Crafter, a unique terrain is generated for every episode with grasslands, lakes, and mountains that contain forests, caves, ores, and lava. The world consists of 64×64 grid cells, but the agent only observes 7×9 grid cells making Crafter a partially observed environment. At each step the agent receives a $64 \times 64 \times 3$ input image as in Fig. 4.2a, and returns an action from a categorical space of 17 actions. Of those actions, four are for moving in the environment, one is for sleeping, and the rest are for crafting or interacting with specific tools or resources. For an example of gameplay, see Figure 4.2a, where you can see the player (agent) in the middle, a skeleton on its top right, a zombie bottom-left from the agent, a cow, water, trees, stones, iron and lava.

In the bottom-left part of Fig. 4.2a the agent's health status is shown. The heart represents the number of lives it has left, and next to it, the levels of food, water, and energy (that it recharges by sleeping). Next to this is the inventory display that shows the resources (here wood, stone, coal, and iron) and tools and weapons (here wooden, stone, and iron pickaxes and swords) it has collected or crafted so far. The agent must collect resources such as wood, stone, coal, and iron to craft tools. The agent needs to eat food, drink water, sleep and defend against enemies to prevent its health from reaching zero. All these agents and resources are represented as objects in the environment. Eating food, collecting resources, defeating enemies, or crafting new tools are all "achievements". The first time the agent unlocks an achievement in an episode it receives a reward of 1. On the other hand, for every lost health point the reward is -0.1, and for every regained health point $+0.1$. When the health reaches zero or when stepping into lava the agent dies, and the episode is over. Every time the agent unlocks an achievement previously unlocked in the episode, it does not receive any further reward. This should encourage the agent to explore and collect as many achievements as possible instead of exploiting the ones it already knows. Collecting the diamond is the final and the most difficult achievement. To collect a diamond, the agent must build several tools that require even more basic tools and resources to be collected. This results in a very deep technology tree (see Figure 4 in the Crafter paper [Hafner, 2021]).

In total, there are 22 achievements and the main evaluation metric is the so-called *crafter score* S. It is computed by averaging the unlocked achievements in the log space (to account for differences in their difficulties):

$$S = \exp(\frac{1}{N}\sum_{i=1}^{N}\ln(1+s_i)), \tag{4.1}$$

where $s_i \in [0, 100]$ is percentage of episodes in which the achievement i was unlocked and N is the number of achievements (here $N = 22$). This score is invariant to the number of times an achievement was unlocked in an episode (once it is achieved). Due to averaging in the log space, the score will increase more for unlocking difficult achievements in a few episodes (e.g. a diamond) than for unlocking easy ones more frequently (e.g. collecting wood). The final evaluation metrics is the geometric mean of S over all episodes.

CrafterOODapp. Building on top of Crafter, we introduce CrafterOODapp, where we develop another axis of variation: object appearance based on color. We introduce new variants of objects the agent most frequently interacts with: trees, cows, zombies, stones, coal, and skeletons. For a detailed overview of the newly introduced objects, see Figure B.2. This evaluates generalization to unseen or infrequently seen objects. The agent is trained on an environment with one distribution of such objects' appearances and then evaluated on an environment with a different distribution, possibly one with objects never seen during training. In this way, CrafterOODapp sets the stage for the development of agents based on fast-adaptation or meta-learning.

In the default Crafter environment, the agent always sees only the first variant of an object. On the other hand, in CrafterOODapp, the agent encounters one of the four object variants with varying frequencies of occurrence. For example, in the easiest scenario, the agent sees all four object variants (O_1, O_2, O_3, O_4) with equal (25%) probability in the training environment, and then in the evaluation environment it encounters only the "last" three objects O_2, O_3, O_4 with equal (but increased, 33.3%) probability. We then generate progressively more challenging scenarios where the training environment contains more instances of object variant O_1. Lastly, in the most extreme case of zero-shot generalization, we train agents on environments containing only the first object O_1 and then evaluate them on environments containing only the last three objects O_2, O_3, O_4.

CrafterOODnum. Also, building atop Crafter, we introduce CrafterOODnum, where the agent encounters different numbers of objects during the training and evaluation phases. As in CrafterOODapp, we vary the objects the agent most frequently interacts with: trees, coal, cows, zombies, and skeletons. To get a clear picture of these environments, in Table 4.6 we show the numbers of each of these objects in the respective environment variants. Starting from default object distributions in Crafter, we increase or decrease object numbers by powers of two. These variants are particularly challenging as the agent might face never-before-seen situations such as fighting against more enemies or surviving in environments with resources scarcer compared to what it was trained on. Importantly, changing the number of objects does not change the maximum score the agent can obtain, as it is still possible to unlock all the achievements, but with different difficulties. However, this changes the reward function and makes the credit assignment problem more difficult because reward is sparser with fewer objects.

4.2 Methods

In this section, we describe the agents' network architectures. We first introduce CNN feedforward baselines, followed by LSTM-based recurrent agents, and finally introduce object-centric agents. To optimize agents we use the PPO [Schulman et al., 2017] implementation in the stable baselines [Raffin et al., 2021] (for a discussion on off-policy algorithms see Appendix B.4).

4.2.1 Linear and Recurrent PPO

PPO learns to map images to actions via policy gradients. Two feedforward variants are investigated that differ by the CNN policy they use. We introduce a recurrent version based on an LSTM [Hochreiter and Schmidhuber, 1997] to account for partial observability. Note that the first application of policy gradients to LSTM dates back to 2007 [Wierstra et al., 2007]. For architecture details see Appendix B.7.

PPO with NatureCNN (PPO-CNN). This baseline is architecturally identical to the one used in [Hafner, 2021], with the CNN policy from the DQN paper [Mnih et al., 2015].

PPO with size-preserving CNN (PPO-SPCNN). Size-preserving CNN (SPCNN) differs from the PPO-CNN baseline by the CNN architecture. We introduce it as a baseline for object-centric agents that use SPCNN for "feature mixing," inspired by SlotAttention's visual encoder [Locatello et al., 2020]. SPCNN does not have pooling layers, so the resulting output tensor is of the same height and width as the input image. The policy input tensor is much larger (64x64x64 instead of 8x8x64 for CNN) making the policy also very large (134M vs. 1M parameters).

Recurrent PPO Agents. Crafter is a partially observable environment. The agent observes only a part of the world (see Figure 4.2a). To perform well, it needs to remember the locations of resources, e.g. food, mining materials, and where it placed objects for crafting new tools, e.g. table or furnace. Additionally, some achievements require the agent to perform a long chain of reasoning. For example, to collect a diamond, the agent needs to have an iron pickaxe, for which it needs to have crafted a furnace and collected coal, for which in turn it needs to have previously collected wood, placed a table, made a wooden pickaxe, and collected stone (in that particular order). For the complete overview of the technology tree, see Figure 4 in [Hafner, 2021]. For these reasons, we introduce recurrent agents (LSTM-CNN and LSTM-SPCNN) where we use LSTMs as the critic and the actor networks. These networks, in theory, entitle the agent to have memories of the world map and its inventory and can help unlock achievements.

4.2.2 Object-centric Agents

Our experiments show that the baseline agents fail to generalize to OOD environments. This is in line with the previous findings that neural networks fall short in generalization. One hypothesis for this is due to their inability to dynamically and flexibly bind information distributed throughout the network [Greff et al., 2020], also known as the binding problem [Von Der Malsburg, 1994; Roskies, 1999]. To address this issue, there has been an increased interest in designing object-centric neural networks that learn (discrete) object representations from raw visual input, which support efficient learning and generalization to novel scenarios and behaviors. Object-centric methods have successfully been used for OOD generalization in supervised and unsupervised learning [Greff et al., 2017; van Steenkiste et al., 2018; Kosiorek et al., 2018; Stanić and Schmidhuber, 2019; Greff et al., 2019; Locatello et al., 2020; Stanić et al., 2021; Kipf et al., 2021] and in RL [Watters et al., 2019a; Veerapaneni et al., 2020; Kipf et al., 2020; Carvalho et al., 2021]. Since Crafter is composed of objects, we expect these methods to facilitate learning and show stronger generalization capabilities. In this vein, we designed two object-centric agents and investigated them on CrafterOOD environments. They take as input either patches extracted from the image through a size-preserving CNN (Figure 4.4) or learn their own representation of an object by attending over the whole input tensor (Figure 4.7). Below we provide an overview of our two object-centric agents. For more details we refer to Appendix B.3.

Object-centric Self-Attention agents (OC-SA) (Figure 4.2b and Table B.3) learn a policy via a dot-product self-attention between the input patches and a learned "CLS" token (a vector initialized to unit Gaussian parameters and optimized via backprop). This is similar to using the CLS token in BERT [Devlin et al., 2018]. Let $(x_1, ..., x_k, CLS) \in \mathbb{R}^{(k+1)\times d_{in}}$ be the input sequence, where k is the number of input patches and d_{in} is their dimensionality. Dot-product self-attention is defined as:

$$Attention(Q, K, V) = softmax\Big(\frac{QK^T}{\sqrt{d}}\Big)V, \tag{4.2}$$

where $Q \in \mathbb{R}^{(k+1)\times d}$, $K \in \mathbb{R}^{(k+1)\times d}$, $V \in \mathbb{R}^{(k+1)\times d}$, are the query, key and value matrices, resulting from a linear mapping of the input sequence onto a space of dimension d. Absolute sinusoidal positional embeddings [Vaswani et al., 2017] are added to the input, enabling the learning of relative patch positions, e.g. if an enemy is nearby.

Object-centric Cross-Attention agents (OC-CA) (Figure 4.2c and Table B.4) learn a set of vectors, which we refer to as 'slots', by attending either to a patch grid or the whole input (this can be seen as an extreme case of the patch grid with patch size of one pixel). This idea was introduced in SetTransformer [Lee et al., 2019] and successfully used for object detection [Carion et al., 2020], unsupervised learning of objects [Locatello et al., 2020], learning permutation-invariant agents [Tang et al., 2020; Tang and Ha, 2021] and general perception modules [Jaegle et al., 2021b,a; Alayrac et al., 2022]. OC-CA computes attention between queries, keys and values as in self-attention Eq. (4.2), but with the queries coming from the n learned slots: $Q \in \mathbb{R}^{n\times d}$, $K \in \mathbb{R}^{k\times d}$, $V \in \mathbb{R}^{k\times d}$, where k is the number of input patches or image pixels for *learned* objects. The extreme patch size allows for higher expressiveness as each slot can attend to variable-size or more distant image regions. After cross-attention, slots are pooled by a mean operation and fed to the policy.

Method	Crafter Score
PPO*	4.6 ± 0.3
DreamerV2*	10.0 ± 0.2
PPO-CNN	10.3 ± 0.6
PPO-SPCNN	11.6 ± 0.6
LSTM-CNN	10.4 ± 0.2
LSTM-SPCNN	**12.1** ± 0.8
OC-SA	11.1 ± 0.7
OC-CA	10.0 ± 0.4
PPO-CNN (no inventory)	6.9 ± 0.4
LSTM-CNN (no inventory)	**7.7** ± 0.5

Table 4.1: Scores on Crafter for agents trained on 1M environmental steps. Reported are mean scores and standard deviations of 10 random seeds. *score from [Hafner, 2021].

4.3 Experiments on the Crafter environment

Here we present our findings on the original Crafter environment. We first investigate feedforward PPO baselines both with and without inventory display. Recurrent agents are then investigated, and we show that they outperform feedforward ones. Finally, we evaluate asymptotic performance and show that the tuned agents can learn to unlock all but the last achievement.

4.3.1 Improved baselines and hyper-parameter analysis.

By tuning a few hyper-parameters, we find that the simple PPO-CNN baseline outperforms the model-based DreamerV2 [Hafner et al., 2020] (Table 4.1). Surprisingly, even though the PPO-SPCNN model has 134M parameters, PPO is not only able to train it, but often outperforms other methods (Tables 4.1 and 4.4). This holds for both feedforward (PPO-SPCNN) and the recurrent (LSTM-SPCNN) agents. PPO hyper-parameters we searched over are shown in Table 4.2. For most hyper-parameters, in the experiments we used the default values (shown

Hyper-parameter	Sweep Values
Learning rate	[0.001, 0.0005, *0.0003*, 0.0001, 0.00005]
Batch size	[*64*, **128**, 256]
Number of rollouts	[1024, *2048*, **4096**, 8196, 16384]
Number of epochs	[3, **4**, 6, 7, *10*]
Discount factor	[0.8, 0.9, **0.95**, 0.97, *0.99*]
GAE λ	[0.5, **0.65**, 0.75, 0.85, *0.95*]
Clip Range	[0.1, *0.2*, 0.3]
Max Gradient Norm	[0.1, 0.3, *0.5*, 1.0]

Table 4.2: PPO hyper-parameters we searched over. Final used values are bolded, unless default values are used (shown in italic).

Hyper-parameter	Sweep Values
Batch size	$16, 32, 64$
Discount factor	$[0.9, 0.95, 0.99, 0.999]$
Actor entropy scale	$[0.003, 0.001, 0.0003, 0.0001, 0.00003]$
KL loss scale	$[0.1, 0.3, 1, 3]$
Reward loss scale	$[0.5, 1, 2]$
Discount loss scale	$[0.5, 1, 2]$
KL balance	$[0.5, 0.8, 1, 2]$
Discount λ	$[0.8, 0.9, 0.95]$

Table 4.3: DreamerV2 hyper-parameters we searched over.

in italic) that were tuned for Atari in previous work; the ones we changed are bolded. For a detailed hyper-parameter analysis we refer to Appendix B.2.

The best performing agent in the Crafter Benchmark paper [Hafner, 2021] was the DreamerV2 agent [Hafner et al., 2020]. Therefore, we also tried tuning its hyper-parameters: the actor entropy scale, the discount factor, the transition, and the entropy loss scales (recommended in Table B.1 of the DreamerV2 paper [Hafner et al., 2020]), as well as the batch size, KL loss scale, reward loss scale and the discount λ. However, our hyper-parameter search did not improve on results in [Hafner, 2021] (see Table 4.3).

4.3.2 Agents use inventory display as a "scratchpad."

Although purely feedforward, PPO-SPCNN can unlock almost all achievements (19/22 for 1M steps and 21/22 achievements when trained for 25M steps, Section 4.3.4). This requires the agent to remember unlocked achievements (e.g., I have an iron pick-axe now, need to mine a diamond next). However, the fact that the memory-less feedforward models unlocked most achievements leads us to hypothesize that it uses the inventory display as a "scratchpad." We evaluate this by removing the inventory display, thus making the environment much less observable. Here the agent really needs memory to know when to drink or whether it has already crafted an item or not. The results in Table 4.1 support our hypothesis, e.g., compare PPO-CNN with and without the inventory display ('PPO-CNN (no inventory)').

4.3.3 Recurrent improve over feedforward agents

The agent's reliance on the inventory display as a "scratchpad" led us to investigate whether recurrent agents could improve over feedforward ones by memorizing actions, maps, unlocked achievements and crafted tools in its inventory. With the observable inventory, LSTM-CNN does not improve upon the feedforward variant PPO-CNN (10.3 vs. 10.4 in Table 4.1). However, when the inventory is not observable, LSTM-CNN outperforms PPO-CNN (6.9 vs. 7.7). This might indicate that the recurrent agents learn to store achievements in memory, although not perfectly, as we observe a drop from the case with the observable inventory. On the other hand, LSTM-SPCNN outperforms its feedforward variant PPO-SPCNN in both in-distribution and OOD settings (Table 4.4). Here the inventory is observed, so the largest benefit must arise from the agent's ability to memorize the partially observed map layout.

4.3.4 Asymptotic Performance

As the Crafter benchmark is designed to accelerate RL research and facilitate short research cycles, the default evaluation protocol is to train agents up to 1M steps. To better understand the asymptotic performance, we train PPO-CNN and PPO-SPCNN agents for 25M steps. The resulting mean Crafter scores over ten seeds are shown in Table 4.5. Other agents do not improve significantly over these two in the asymptotic regime. For readability, we omit standard deviations (they are all below 0.6, decreasing to 0.2 at 25M steps due to the geometric mean averaging in the score). Firstly, PPO-SPCNN consistently out-

Train/Eval Dist	PPO-CNN	PPO-SPCNN	LSTM-CNN	LSTM-SPCNN	OC-SA	OC-CA
Training: $O_1 : 100\%$	10.4 ± 0.6	11.4 ± 0.5	10.5 ± 0.5	12.3 ± 0.4	11.0 ± 0.5	10.1 ± 0.6
Evaluation: $O_1 : 100\%$	10.3 ± 0.6	11.6 ± 0.6	10.4 ± 0.2	**12.1** ± 0.8	11.1 ± 0.7	10.0 ± 0.4
$O_{1-4} : 25\%$	9.2 ± 0.5	10.7 ± 0.6	10.7 ± 0.6	11.5 ± 0.4	9.7 ± 1.1	9.9 ± 0.6
$O_1 : 0\%,\ O_{2-4} : 33.3\%$	9.2 ± 0.7	11.0 ± 1.1	11.0 ± 1.0	**11.6** ± 0.6	9.7 ± 1.2	9.2 ± 0.7
$O_1 : 52\%, O_{2-4} : 16\%$	9.9 ± 0.5	11.1 ± 0.7	11.5 ± 1.4	11.1 ± 0.5	9.6 ± 0.8	9.4 ± 0.9
$O_1 : 0\%,\ O_{2-4} : 33.3\%$	10.0 ± 0.7	11.2 ± 1.1	**11.4** ± 1.6	11.0 ± 0.5	10.6 ± 0.9	9.9 ± 0.9
$O_1 : 76\%, O_{2-4} : 8\%$	9.9 ± 0.4	11.5 ± 0.6	10.7 ± 1.0	11.6 ± 0.6	9.8 ± 0.8	11.3 ± 0.5
$O_1 : 0\%,\ O_{2-4} : 33.3\%$	9.2 ± 0.6	10.5 ± 0.8	10.4 ± 1.0	**10.7** ± 0.7	9.2 ± 0.8	10.5 ± 0.7
$O_1 : 88\%, O_{2-4} : 4\%$	10.1 ± 0.6	12.2 ± 0.8	11.5 ± 1.4	11.3 ± 0.4	10.5 ± 1	11.2 ± 0.9
$O_1 : 0\%,\ O_{2-4} : 33.3\%$	9.1 ± 0.7	**10.2** ± 0.7	10.1 ± 1.3	9.8 ± 0.8	9.4 ± 1.3	9.4 ± 1.0
$O_1 : 94\%, O_{2-4} : 2\%$	10.9 ± 0.7	12.0 ± 0.8	11.4 ± 1.2	11.7 ± 0.6	10.5 ± 0.6	10.8 ± 1.1
$O_1 : 0\%,\ O_{2-4} : 33.3\%$	8.6 ± 0.7	9.2 ± 1.1	9.1 ± 1.4	9.8 ± 0.8	**9.9** ± 0.8	8.8 ± 0.9
$O_1 : 97\%, O_{2-4} : 1\%$	10.5 ± 0.6	11.8 ± 0.7	11.9 ± 1.4	12.0 ± 0.3	10.3 ± 1.1	10.8 ± 0.6
$O_1 : 0\%,\ O_{2-4} : 33.3\%$	7.3 ± 0.5	7.7 ± 1.0	8.2 ± 1.0	8.6 ± 1.0	**9.3** ± 0.8	8.8 ± 0.8
$O_1 : 100\%, O_{2-4} : 0\%$	10.5 ± 0.6	11.8 ± 0.6	10.7 ± 0.2	11.9 ± 0.8	11.1 ± 1.3	10.7 ± 0.6
$O_1 : 0\%,\ O_{2-4} : 33.3\%$	7.3 ± 0.5	7.7 ± 0.9	5.8 ± 0.2	6.8 ± 1.0	**8.0** ± 1.2	7.6 ± 0.5

Table 4.4: Scores on CrafterOODapp (mean and standard deviations over 10 random seeds) for agents trained for 1M environmental steps. Each setting has two rows, denoting training (e.g. $O_{1-4} : 25\%$) and evaluation ($O_1 : 0\%, O_{2-4} : 33.3\%$) scores.

performs the PPO-CNN baseline. Moreover, the PPO-SPCNN agent learns to unlock all but one achievement (the diamond) at 5M steps (though this is not a regular event). We suspect that the difficulty in mining the diamond is its rarity, combined with the depth and the complexity of the technology tree of skills that the agent needs to master, and tools to acquire to even have a chance at mining the diamond. Furthermore, there are only very few diamonds on the map, so finding it through random exploration is a highly unlikely event per se. Moreover, unlike humans the agents have no prior notion that a diamond is valuable, and only observe it as a different colored patch This leads us to think better better exploration strategies are needed. This challenge seems ideal for investigating novel curiosity-based agents [Schmidhuber, 1990; Schmidhuber, 1991a; Schmidhuber, 2013] or reward shaping [Justesen and Risi, 2018]. In Table 4.5 we also compare to the concurrent work [Yi et al., 2022] and two previous works (RRL [Zambaldi et al., 2019] and SMORL [Zadaianchuk et al., 2020]), results taken from [Yi et al., 2022]. We observe from Table 4.5 that other models achieve significantly lower scores compared to ours.

Method	1M	2M	5M	10M	20M	25M
PPO-CNN (Train)	10.3	15.7	18.5	27.4	29.0	29.4
PPO-CNN (Eval)	10.3	15.9	18.3	27.1	29.1	29.6
PPO-SPCNN (Train)	11.2	16.7	18.9	27.9	30.6	30.8
PPO-SPCNN (Eval)	11.6	16.5	19.4	27.8	30.5	**30.9**
OCARL [Yi et al., 2022]	-	-	-	-	-	12.3
RRL [Zambaldi et al., 2019]	-	-	-	-	-	4.22
SMORL [Zadaianchuk et al., 2020]	-	-	-	-	-	3.94

Table 4.5: Asymptotic performance of PPO-CNN and PPO-SPCNN on the Crafter environment up to 25M environmental interactions and comparison to the related work.

4.4 OOD Generalization Experiments

In this section, we evaluate agents on new CrafterOODapp and CrafterOOD-num environments, showing limited generalization of baseline agents and improved generalization of object-centric ones.

4.4.1 CrafterOODapp - out-of-distribution object appearance.

In the CrafterOODapp environment, we generate a collection of increasingly complex adaptation scenarios. Environments contain up to four object variants for trees, cows, iron, stones, zombies, and skeletons, the objects the agent interacts with most frequently (Fig. B.2 in Appendix B.1). The adaptation scenarios differ in object distributions in training and evaluation environments (Table 4.7). For example, the in-distribution training and evaluation corresponds to observing only the first object variant O_1 (first two rows in Table 4.4).

Starting from training with uniformly distributed objects ($O_{1-4} = 25\%$) we make generalization progressively more difficult by skewing the training distribution towards the first object. In this way, we create environments that contain the first object (O_1) in 52% and all the others in 16% of the time ($O_1 : 52\%, O_{2-4} : 16\%$ case in Table 4.4), then environments that contain the first object in 76%, then 88% percent of the time, etc. The evaluation environment always contains only the last three objects (we refer to them as the "evaluation" objects) uniformly distributed (each observed in 33.3% of the time). Up to the point of observing the evaluation objects in 16% of the time the agents generalize fairly well (Table 4.4, but see also Fig. B.3 in Appendix B.5). Decreasing the

Train/Eval Dist	PPO-CNN	PPO-SPCNN	LSTM-CNN	LSTM-SPCNN	OC-SA	OC-CA
Train: easy (x2)	12.4 $\pm$ 2	13.7 $\pm$ 1.2	14.2 $\pm$ 0.8	13.3 $\pm$ 1.0	15.0 $\pm$ 1.8	13.7 $\pm$ 1.8
Eval: default	10.4 $\pm$ 1.7	12.1 $\pm$ 1.0	11.5 $\pm$ 0.7	11.4 $\pm$ 0.9	**13.0** $\pm$ 1.3	11.7 $\pm$ 1.7
easy (x4)	13.1 $\pm$ 2.2	14.5 $\pm$ 1.8	15.1 $\pm$ 1.2	15.0 $\pm$ 0.7	18.6 $\pm$ 2.3	13.7 $\pm$ 1.9
default	8.8 $\pm$ 1.0	9.8 $\pm$ 1.7	9.1 $\pm$ 0.9	9.8 $\pm$ 0.9	**12.8** $\pm$ 1.7	8.8 $\pm$ 0.9
mix (x4)	13.4 $\pm$ 1.7	13.9 $\pm$ 2.1	13.0 $\pm$ 0.7	14.7 $\pm$ 1.6	15.5 $\pm$ 2.0	14.7 $\pm$ 2.3
default	9.2 $\pm$ 0.6	10.2 $\pm$ 1.7	9.0 $\pm$ 0.9	10.5 $\pm$ 0.9	**10.6** $\pm$ 1.7	9.4 $\pm$ 0.9
default	10.2 $\pm$ 0.4	11.5 $\pm$ 0.5	10.7 $\pm$ 0.6	12.1 $\pm$ 0.6	11.3 $\pm$ 0.4	10.2 $\pm$ 0.7
mix (x4)	11.2 $\pm$ 1.1	12.3 $\pm$ 1.0	11.6 $\pm$ 0.9	12.2 $\pm$ 1.3	**12.6** $\pm$ 1.0	10.3 $\pm$ 0.9
default	10.3 $\pm$ 0.4	11.2 $\pm$ 0.6	10.9 $\pm$ 0.7	12.0 $\pm$ 0.4	11.1 $\pm$ 0.5	10.1 $\pm$ 0.6
easy (x2)	10.9 $\pm$ 1.4	**13.7** $\pm$ 1.3	12.8 $\pm$ 0.9	12.9 $\pm$ 1.4	11.4 $\pm$ 1.2	10.1 $\pm$ 0.7
default	10.3 $\pm$ 0.7	11.2 $\pm$ 0.3	10.6 $\pm$ 0.3	11.8 $\pm$ 0.7	11.5 $\pm$ 0.7	10.1 $\pm$ 0.6
easy (x4)	11.3 $\pm$ 0.8	12.5 $\pm$ 0.9	12.8 $\pm$ 1.3	11.9 $\pm$ 1.0	**12.9** $\pm$ 0.8	9.9 $\pm$ 1.2
easy (x2)	11.7 $\pm$ 1.3	13.3 $\pm$ 0.4	13.7 $\pm$ 1.4	13.4 $\pm$ 0.2	15.4 $\pm$ 1.2	12.8 $\pm$ 1.6
hard (x2)	8.0 $\pm$ 0.5	8.1 $\pm$ 1.7	7.6 $\pm$ 0.7	9.2 $\pm$ 0.8	**10.5** $\pm$ 1.5	7.1 $\pm$ 0.8
easy (x4)	14.6 $\pm$ 2.2	15.4 $\pm$ 1.6	15.5 $\pm$ 1.2	15.3 $\pm$ 1.0	17.8 $\pm$ 1.5	15.2 $\pm$ 1.8
hard (x4)	3.0 $\pm$ 0.4	3.3 $\pm$ 0.4	3.0 $\pm$ 0.3	3.4 $\pm$ 0.4	**4.9** $\pm$ 0.6	4.2 $\pm$ 0.3
Train: average	12.0 $\pm$ 1.4	13.1 $\pm$ 1.1	13.0 $\pm$ 0.8	13.4 $\pm$ 0.8	14.5 $\pm$ 1.3	12.6 $\pm$ 1.4
Eval: average	9.1 $\pm$ 0.9	10.2 $\pm$ 1.1	9.7 $\pm$ 0.9	9.1 $\pm$ 1.0	**11.1** $\pm$ 1.1	8.9 $\pm$ 0.9

Table 4.6: Scores on CrafterOODnum (mean and standard deviations over 10 random seeds) for agents trained for 1M environmental steps. Each setting has two rows, denoting scores on the training and evaluation environments.

percentage of evaluation objects in the training environment further, the performance of all agents consistently drops. Finally, in the especially difficult zero-shot generalization scenario, when trained only on the first object ($O_1 = 100\%$), the agent relies on pure chance or interacting with objects that do not change (e.g., water, iron). The agent's failure to generalize is expected: to perform well on unseen evaluation objects, it must perform fast adaptation and systematic generalization in terms of composing the previously obtained knowledge about the observed objects and the inferred representations of the newly introduced evaluation objects.

Motivated by this observation, we introduce object-centric agents, which should (at least in principle) learn better representations by decomposing the input images into sets of objects and their distances via positional encodings. Our experiments show that the object-centric agents OC-SA and OC-CA (the last two columns in Table 4.4) match the vanilla PPO agents on in-distribution and easy OOD generalization. In the most difficult OOD generalization cases though

Env.	Tree	Coal	Cow	Zombie	Skeleton
Easy (x4)	764	206	100	3	2.5
Easy (x2)	380	102	46	6	4.5
Default	189	50	26	15	9.5
Hard (x2)	95	27	13	33	19
Hard (x4)	52	12.5	6	60	38
Mix (all x4)	764	206	100	60	38

Table 4.7: CrafterOODnum environment object numbers.
Default is the original Crafter environment. Easy environments have (x2 or x4) more resources and (x2 or x4) fewer enemies. In hard environments we decrease the number of resources and increase the number of enemies. Mix environment contains four times more resources and enemies.

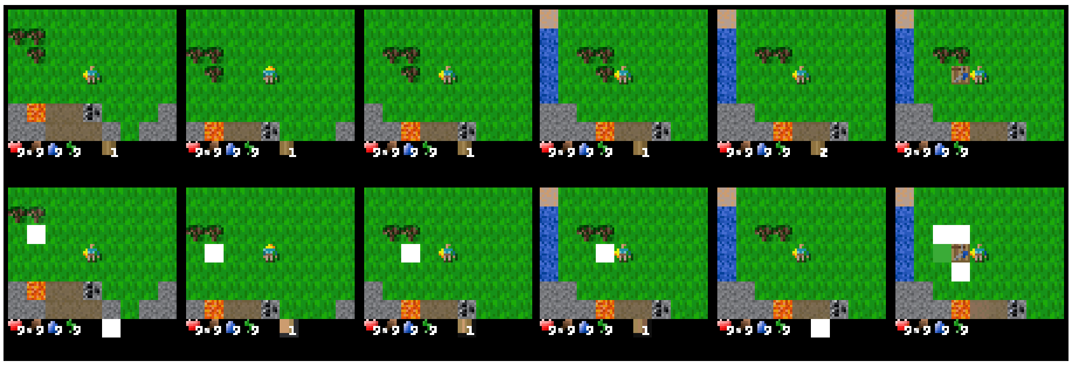


Figure 4.3: An agent building a table. Top row: input images. Bottom row: attention visualized by its intensity. Episode steps are plotted horizontally.

(the last three pairs of rows in Table 4.4), OC-SA and OC-CA generalize better compared to PPO-CNN and PPO-SPCNN (9.9 vs 8.6 and 9.2 for $O_1 = 94\%$, 9.3 vs 7.3 and 7.7 for $O_1 = 97\%$, and 8.0 vs 7.3 and 7.7 for $O_1 = 100\%$). Additionally, object-centric agents are interpretable and potentially easier to build on in future work. We visualize their attention maps in Section 4.5.2. The observed generalization gap makes environments with large differences in object distributions (requiring zero-shot adaptation) fruitful for developing and evaluating novel fast-adaptation [Schmidhuber, 1992, 1993b,c,d; Irie et al., 2022] and meta-learning [Schmidhuber, 1987] agents, such as the ones based on Fast Weight Programmers [Schmidhuber, 1991c; Schmidhuber, 1992, 1993; Schlag et al., 2021; Irie et al., 2021].

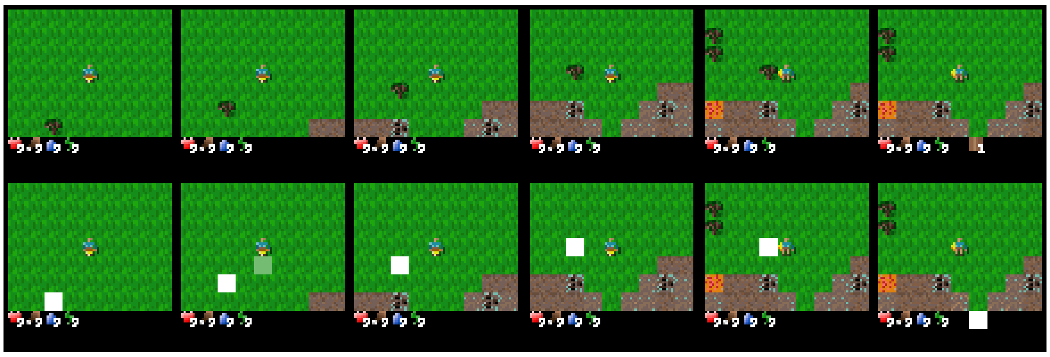


Figure 4.4: An agent collecting resources. Top row: input images. Bottom row: attention visualized by its intensity. Episode steps are plotted horizontally.

4.4.2 CrafterOODnum - out-of-distribution object numbers.

To test generalization on different numbers of objects, we introduce the CrafterOODnum environment. We vary the numbers of objects the agent interacts with most frequently: trees, coal, cows, zombies, and skeletons. These variants are particularly challenging as the agent might face never-before-seen situations such as fighting against more enemies or surviving in environments with resources scarcer compared to what it was trained on. In Table 4.6 we show the numbers of objects in the respective environments. In Table 4.6, 'Default' is the Crafter environment introduced in [Hafner, 2021]. In 'Easy' environments, we increase (double or quadruple) the number of objects representing resources and decrease the number of enemies. 'Hard' environments consist of fewer resources and more enemies. In the 'mix' environment, we quadruple the numbers of all objects—making it on the one hand easier in terms of resources, but on the other hand harder in terms of enemies.

For each pair of training and evaluation environments in Table 4.6 the agents with the best generalization score are bolded. The object-centric agent OC-SA outperforms all models in all but one case. Note how the performance decreases when transferring to harder environments (Easy to Default) and increases when transferring to easier environments (Default to Easy), confirming the intended design difficulty level. The performance drop is largest when transferring from 'Easy (x4)' to 'Hard (x4)' environments where the evaluation environment contains 16x fewer resources and 16x more enemies than the training environment. Finally, the object-centric OC-SA agent achieves the best generalization on average across all the environments (the last row in Table 4.6).

Variant	Crafter Score
OC-CA	**10.0 ± 0.4**
OC-CA + Residual MLP	7.3 ± 0.4
OC-CA + LayerNorm	4.1 ± 0.3
OC-CA + Residual MLP + LayerNorm	3.1 ± 0.6
OC-CA + Slot Competition	7.1 ± 0.3
OC-CA, Number of Slots = 1	8.2 ± 0.9
OC-CA, Number of Slots = 2	7.8 ± 0.4
OC-CA, Number of Slots = 4	8.7 ± 0.7
CA, Number of Slots = 8	**10.0 ± 0.4**
OC-CA, Number of Slots = 16	9.1 ± 0.9
OC-CA, Number of Heads = 1	6.6 ± 0.5
OC-CA, Number of Heads = 2	7.3 ± 0.9
OC-CA, Number of Heads = 4	8.0 ± 0.6
CA, Number of Heads = 8	**10.0 ± 0.4**
CA, Number of Heads = 16	7.2 ± 0.8
OC-CA, Patch Size = 1, Stride = 1	6.9 ± 0.9
OC-CA, Patch Size = 8, Stride = 8	7.2 ± 0.4
OC-CA, Patch Size = 12, Stride = 8	8.7 ± 0.7
OC-CA, Patch Size = 12, Stride = 12	8.9 ± 0.7
OC-CA, Patch Size = 16, Stride = 8	6.2 ± 0.5
OC-CA, Patch Size = 16, Stride = 16	**10.0 ± 0.4**

Table 4.8: OC-CA Agents Network Ablation.

4.5 Object-centric Agents Analysis

In this section, we first perform an ablation study of object-centric agents, inspecting various design choices. Afterwards, we visualize attention patterns of object-centric agents, and find their policies to be interpretable and to match our gameplay intuitions.

4.5.1 Object-centric Agents Ablation

To the best of our knowledge, this is the first time cross-attention-based methods (e.g. SlotAttention- and Perceiver-like methods) are used in an open world

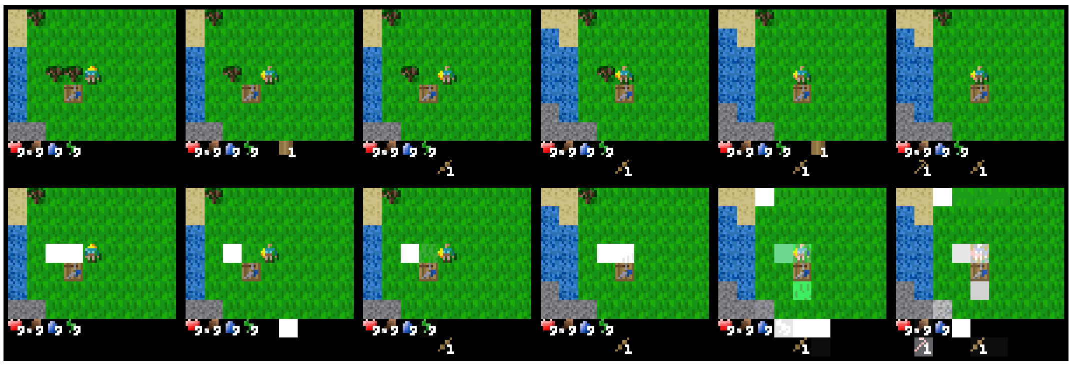


Figure 4.5: An agent crafting weapons. Top row: input images. Bottom row: attention visualized by its intensity. Episode steps are plotted horizontally.

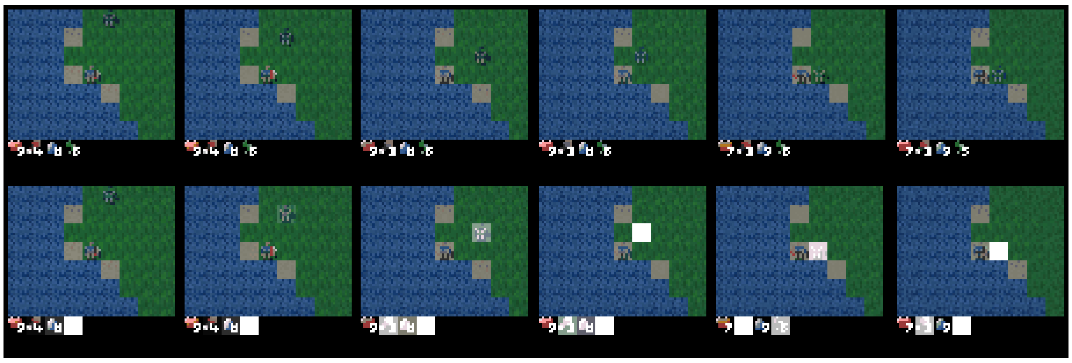


Figure 4.6: An agent defending against enemies. Top row: input images. Bottom row: attention visualized by its intensity. Episode steps are plotted horizontally.

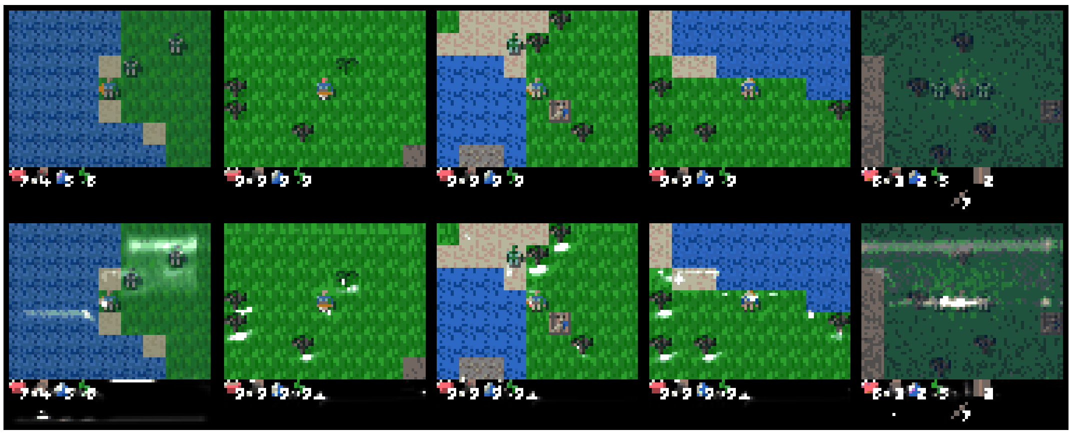


Figure 4.7: Learned attention over the whole input image instead of a fixed patch grid. Top row: input images. Bottom row: attention visualized by its intensity.

survival game. In this section, we perform an ablation analysis, compare our agent to the standard architectural choices from the literature and show that these vanilla agents underperform.

The ablation study results are shown in Table 4.8. Firstly, cross-attention-based methods typically employ LayerNorm [Ba et al., 2016] and residual MLPs. In Table 4.8 we can see that the variant using these "CA + Residual MLP + LayerNorm" underperforms. Including any of these modules individually ("CA + Residual MLP" and "CA + LayerNorm") does not improve performance. Moreover, competition over slots (a softmax over queries) "CA + Slot Competition" akin to SlotAttention also hurts the downstream performance. This led us to converge to an architecture that uses a single cross-attention over the input image, with no latent self-attention, no layer normalization, no residual connections, and no slot-wise competition.

We found the optimal number of heads and slots to be eight. We speculate that the higher number of slots lets each attend to different objects in the input, and the higher number of heads allows for operation specialization for each of the heads. Finally, we observe that a larger patch size (with the appropriate stride) improves performance. On the other hand, the agents with learned object representations "CA, Patch Size = 1, Stride = 1" do not perform as well as the attention with larger patches. Although in theory more powerful as it can attend to varying object sizes, this variant needs first to *learn* which pixels belong to each object. In Crafter, objects are of equal size, so the patches can be chosen to correspond to those. We suspect the attention over pixels would better generalize to objects of different sizes.

4.5.2 OC-SA Agents Visualization and Interpretability.

Here we visualize the CLS token attention in OC-SA. Visualizations in Figs. 4.3 to 4.6 stem from a single agent and are representative of most episodes of a trained agent. In Fig. 4.4, we can see an agent collecting resources. In the first frame (the leftmost figure) the agent notices a tree nearby and then focuses consistently on it and collects it. In the final frame, the agent attends to the newly collected wood resource.

The episode is continued in Figure 4.3, in which the agent builds a table. After collecting one tree, the agent needs to collect another one to build a table. It spots more trees in its vicinity and goes towards them, never losing sight of the tree and the position where it wants to craft a table. It collects another tree and attends again to the wood resources. Determining that it has enough wood (two pieces), it builds a table to facilitate crafting of further tools and weapons.

Once it has built the table, the agent can use it to craft tools (weapons), but it first needs to collect more resources. In Figure 4.5 it collects the two nearby trees and uses them to build a wooden sword and a wooden pickaxe (shown in the inventory). The sequence is reminiscent of what a human player would do: collect one tree, craft a sword, collect another tree and craft a pickaxe. It can use pickaxe to collect stones or coal or to defend against enemies.

In Fig. 4.6, an agent defends against a zombie during the night (the reason why the frame color is darker). Initially, the agent does not attend to the zombie, probably because it is still far away. As the zombie gets closer, it attends more and more to it and finally defeats it.

4.5.3 OC-CA Agents Visualization and Interpretability.

In Figure 4.7 we visualize the learned attention patterns when each slot can attend to any set of pixels in the input image, compared to only a predefined set of patches from a fixed grid. We find that the learned attention attends to the salient objects in the scene: zombies (1st, 3rd, and 5th columns), trees (2nd, 3rd, and 4th columns), resources (water overall), and the inventory. We observe more numerous but smaller attention patterns than the patch-based attention that is typically focused on two to five patches. Although in theory more powerful, in practice the learned attention models underperform the patch-based ones. We suspect this is because the learned attention agents first need to find out what an object actually is, whereas patch-based ones have a more "guided" learning process. We speculate that greater gains of learned attention would occur in scenarios with objects of varying sizes, in which case a fixed grid would be a too rigid representation.

4.6 Related Work

RL benchmarks have played a crucial role in developing and evaluating novel (deep) RL algorithms [Bellemare et al., 2013; Brockman et al., 2016; Kempka et al., 2016; Beattie et al., 2016; Tassa et al., 2018; Juliani et al., 2018]. An especially important role has been played by video games including arcade games [Bellemare et al., 2013], racing environments [Wymann et al., 2000], first-person shooters [Kempka et al., 2016], strategy games [Synnaeve et al., 2016; Vinyals et al., 2017], and open world games [Guss et al., 2019, 2021]. For an extensive survey of deep (reinforcement) learning for video game playing see

[Justesen et al., 2019]. Open world games have recently received special attention, e.g., Minecraft [Johnson et al., 2016; Guss et al., 2019, 2021].

However, these environments come with their drawbacks. For example, Minecraft is too complex to be solved from scratch by current methods [Milani et al., 2020]. The recently published method for mining a diamond [Baker et al., 2022] uses human-annotated data to bootstrap learning via behavioral cloning. It is also unclear by what metric agents should be evaluated. On the other hand, Atari requires large amounts of computation: training an agent with five random seeds on each game for 200M steps requires over 2000 GPU days [Castro et al., 2018; Hessel et al., 2018; Hafner, 2021], which hinders fast iteration cycles. Most importantly, many Atari games are nearly deterministic, so the agents can approximately memorize action sequences and are not required to generalize to new situations [Machado et al., 2018].

Environments like ProcGen [Cobbe et al., 2020] evaluate the generalization capabilities of agents through procedural level generation, which requires substantial compute [Hafner, 2021]. Chan et al. [Chan et al., 2022] recently investigated how RL algorithms perform in environments where feature distribution is not uniform but Zipfian, similar to the one of objects encountered in the real world, or in our CrafterOODapp environments. Generalization benchmarks based on DeepMind's Control Suite [Tassa et al., 2018; Stone et al., 2021; Hansen and Wang, 2021; Grigsby and Qi, 2020; Zhang et al., 2018] test the agent's robustness to background variations but not to variations of environmental object configurations, e.g., visual appearance, number of objects, or object compositions. For a thorough recent survey on generalization in RL, see [Kirk et al., 2021].

The Crafter game addresses these shortcomings. Crafter is an open world survival game for RL research whose dynamics are inspired by the popular game Minecraft. The benchmark is designed to address existing research challenges, such as strong generalization via procedural generation, deep exploration via achievements conditioned on one another, learning from high-dimensional image observations and sparse rewards that require long-term reasoning and credit assignment. It facilitates evaluation by combining semantically meaningful achievements and fast iteration speed. Our newly introduced CrafterOODapp and CrafterOODnum inherit all these benefits and additionally test some agent's generalization ability and robustness in presence of unseen objects, setting the stage for the development of fast-adaptation or meta-learning agents.

Research on RL neural network-based policies that generalize to OOD environments is an important research area currently tackled from several directions [Kirk et al., 2021]. One potential explanation of the poor generalization of neural networks is that they cannot dynamically and flexibly bind the information distributed throughout the network [Greff et al., 2020]. This is also known as the binding problem [Von Der Malsburg, 1994; Roskies, 1999]. Recent work addresses this through object-centric neural networks that learn (discrete) object representations from raw visual input, to support efficient learning and generalization to novel scenarios and behaviors. Object-centric methods have successfully been used for OOD generalization in both supervised and unsupervised learning [Greff et al., 2015, 2016, 2017; van Steenkiste et al., 2018; Eslami et al., 2016; Kosiorek et al., 2018; Stanić and Schmidhuber, 2019; Burgess et al., 2019; Greff et al., 2019; Engelcke et al., 2020; Locatello et al., 2020; Stanić et al., 2021; Creswell et al., 2021; Kipf et al., 2021] and in RL [Watters et al., 2019a; Veerapaneni et al., 2020; Kipf et al., 2020; Carvalho et al., 2021]. To the best of our knowledge, this is the first time they are used in open world RL survival games.

Architecturally closest to our work are methods that learn a set of vectors (slots) by attending to the input via cross-attention. This idea was introduced in SetTransformer [Lee et al., 2019] and successfully used for object detection [Carion et al., 2020], unsupervised learning of objects [Locatello et al., 2020; Kipf et al., 2021], learning permutation-invariant agents [Tang et al., 2020; Tang and Ha, 2021], and general perception modules [Jaegle et al., 2021b,a; Alayrac et al., 2022]. Most similar to our object-centric agents are the AttentionAgent [Tang et al., 2020] and the SensoryNeuron [Tang and Ha, 2021]. The former uses neuroevolution to optimize an architecture with a hard attention bottleneck, resulting in a network that only receives a fraction of the visual input and generalizes to unseen backgrounds. The latter further improves AttentionAgent's robustness to permuted orderings of its inputs. Our agents also share similarities with attention-based modular neural networks [Santoro et al., 2018; Goyal et al., 2019; Mott et al., 2019; Carvalho et al., 2021], learning a (modular) representation by attending over the input. However, unlike our object-centric work, prior work did not consider object-centric agents for OOD generalization in procedurally generated open world games.

4.7 Conclusion and Discussion

Challenging benchmarks are essential for the research on new reinforcement learning methods. However, they should be simple enough to allow for convenient systematic analysis and fast iteration cycles. Tests against strong baselines are crucial.

The paper we described in this chapter contributions to all of the above. We report important observations on the Crafter environment. Our analysis shows that PPO-based agents trained for 20M steps can unlock all but the last achievement. However, this impressive score still falls short of the human score [Hafner, 2021], indicating that Crafter remains an open research challenge, especially for sample-efficient RL agents.

We introduce CrafterOODapp and CrafterOODnum, new environments that evaluate some agent's robustness against varying object appearances and numbers of objects. Baseline agents fail to adapt in evaluation environments containing unseen objects, whereas our novel object-centric agents perform well. An additional benefit of object-centric agents is their interpretability. In the investigated environments, self-attention-based methods outperform cross-attention methods. Future work will confirm or disconfirm similar superiority in environments with varying object sizes. Further variations could also include different reward between training and test environments or different production trees (such as having more or less path leading to an achievement) in the evaluation environments.

Our OOD environments should help to evaluate fast-adaptation and meta-learning agents. For example, object-centric agents could be improved by incorporating decoding objectives, to extract as much signal from the environment as possible, and by exploring ways of training larger object-centric networks. It would also be interesting to investigate whether object-centric inductive biases can enable model-based RL agents to learn better world models. Crafter and our newly introduced CrafterOOD are good candidates for exploring all of these directions, offering control of environmental configurations, excellent visual inspection, and fast iteration cycles.

Finally, goals and actions only implicitly modulate the learning of object representations. It would be interesting to investigate ways of explicitly modulating objects either by conditioning them on the goal or the action.

Chapter 5

Synchrony-based Object Discovery with Complex-Valued Autoencoders

Current state-of-the-art object-centric models use slots and attention-based routing for binding. However, this class of models has several conceptual limitations: the number of slots is hardwired; all slots have equal capacity; training has high computational cost; there are no object-level relational factors within slots. Synchrony-based models in principle can address these limitations by using complex-valued activations which store binding information in their phase components. However, working examples of such synchrony-based models have been developed only very recently, and are still limited to toy grayscale datasets and simultaneous storage of less than three objects in practice. In this chapter, we introduce architectural modifications and a novel contrastive learning method that greatly improve the state-of-the-art synchrony-based model [Löwe et al., 2022]. For the first time, we obtain a class of synchrony-based models capable of discovering objects in an unsupervised manner in multi-object color datasets and simultaneously representing more than three objects.

Slot-based approaches have several conceptual limitations. First, the binding information (i.e. addresses) about object instances are maintained only by the constant number of slots—a hard-wired component which cannot be adapted through learning. This restricts the ability of slot-based models to flexibly represent a varying number of objects with variable precision without tuning the slot size, number of slots, number of iterations, etc. Second, the inductive bias used by the grouping module strongly enforces independence among all pairs of slots. This restricts individual slots to store relational features at the object-

This chapter is based on "Contrastive Training of Complex-Valued Autoencoders for Object Discovery" [Stanić et al., 2023b] paper, which was published at NeurIPS 2023.

level, and requires additional processing of slots using a relational module, e.g., Graph Neural Networks [Battaglia et al., 2016; Gilmer et al., 2017] or Transformer models [Vaswani et al., 2017; Schmidhuber, 1992; Schlag et al., 2021]. Third, binding based on iterative attention is in general computationally very demanding to train [Löwe et al., 2022]. Additionally, the spatial broadcast decoder [Watters et al., 2019b] (a necessary component in these models) requires multiple forward/backward passes to render the slot-wise reconstruction and alpha masks, resulting in a large memory overhead as well.

Recently, Löwe et al. [2022] revived another class of neural object binding models [Mozer et al., 1991; Mozer, 1998; Reichert and Serre, 2014] (*synchrony-based* models) which are based on complex-valued neural networks. Synchrony-based models are conceptually very promising. In principle, they address most of the conceptual challenges faced by slot-based models. The binding mechanism is implemented via constructive or destructive phase interference caused by the addition of complex-valued activations. They store and process information about object instances in the phases of complex activations which are more amenable to adaptation through gradient-based learning. Further, they can in principle store a variable number of objects with variable precision by partitioning the phase components of complex activations at varying levels of granularity. Additionally, synchrony-based models can represent relational information directly in their distributed representation, i.e., distance in phase space yields an implicit relational metric between object instances (e.g., inferring part-whole hierarchy from distance in "tag" space [Mozer, 1998]). Lastly, the training of synchrony-based models is computationally more efficient by two orders of magnitude [Löwe et al., 2022].

However, the true potential of synchrony-based models for object binding is yet to be explored; the current state-of-the-art synchrony-based model, Complex-valued AutoEncoder (CAE) [Löwe et al., 2022], still has several limitations. First, it is yet to be benchmarked on any multi-object datasets [Kabra et al., 2019] with color images (even simplisitic ones such as `Tetrominoes`) due to limitations in the evaluation method to extract discrete object identities from continuous phase maps [Löwe et al., 2022]. Second, we empirically observe that it shows low *separability* (Table 5.2) in the phase space, thereby leading to very poor (near chance-level) grouping performance on `dSprites` and `CLEVR`. Lastly, CAE can simultaenously represent at most 3 objects [Löwe et al., 2022], making it infeasible for harder benchmark datasets [Kabra et al., 2019; Greff et al., 2022].

Our goal is to improve the state-of-art synchrony models by addressing these limitations of CAE [Löwe et al., 2022]. First, we propose a few simple architectural changes to the CAE: i) remove the `1x1` convolution kernel as well as the sigmoid activation in the output layer of decoder, and ii) use convolution and upsample layers instead of transposed convolution in the decoder. These changes enable our improved CAE, which we call CAE++, to achieve good grouping performance on the `Tetrominoes` dataset—a task on which the original CAE completely fails. Further, we introduce a novel contrastive learning method to increase *separability* in phase values of pixels (regions) belonging to two different objects. The resulting model, which we call Contrastively Trained Complex-valued AutoEncoders (CtCAE), is the first kind of synchrony-based object binding models to achieve good grouping performance on multi-object color datasets with more than three objects (Figure 5.2). Our contrastive learning method yields significant gains in grouping performance over CAE++, consistently across three multi-object color datasets (`Tetrominoes`, `dSprites` and `CLEVR`). Finally, we qualitatively and quantitatively evaluate the *separability* in phase space and generalization of CtCAE w.r.t. number of objects seen at train/test time.

5.1 Background

We briefly overview the CAE architecture [Löwe et al., 2022] which forms the basis of our proposed models. CAE performs binding through complex-valued activations which transmit two types of messages: *magnitudes* of complex activations to represent the strength of a feature and *phases* to represent which features must be processed together. The constructive or destructive interference through addition of complex activations in every layer pressurizes the network to use similar phase values for all patches belonging to the same object while separating those associated with different objects. Patches of the same object contain a high amount of pointwise mutual information so their destructive interference would degrade its reconstruction.

The CAE is an autoencoder with *real-valued* weights that manipulate *complex-valued* activations. Let h and w denote positive integers. The input is a positive real-valued image $\mathbf{x}' \in \mathbb{R}^{h\times w\times 3}$ (height h and width w, with 3 channels for color images). An artificial initial phase of zero is added to each pixel of $\mathbf{x}'$ (i.e., $\boldsymbol{\phi} = \mathbf{0} \in \mathbb{R}^{h\times w\times 3}$) to obtain a complex-valued input $\mathbf{x} \in \mathbb{C}^{h\times w\times 3}$:

$$\mathbf{x} = \mathbf{x}' \odot e^{i\boldsymbol{\phi}}, \quad \text{where } \odot \text{ denotes a Hadamard product} \tag{5.1}$$

Let d_{in}, d_{out} and p denote positive integers. Every layer in the CAE transforms complex-valued input $\mathbf{x} \in \mathbb{C}^{d_{\text{in}}}$ to complex-valued output $\mathbf{z} \in \mathbb{C}^{d_{\text{out}}}$ (where we simply denote input/output sizes as d_{in} and d_{out} which typically have multiple dimensions, e.g., $h \times w \times 3$ for the input layer), using a function $f_{\mathrm{w}} : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ with real-valued trainable parameters $\mathbf{w} \in \mathbb{R}^p$. f_{w} is typically a convolutional or linear layer. First, f_{w} is applied separately to the real and imaginary components of the input:

$$\boldsymbol{\psi} = f_{\mathrm{w}}\left(\mathrm{Re}(\mathbf{x})\right) + f_{\mathrm{w}}\left(\mathrm{Im}(\mathbf{x})\right) i \quad \in \mathbb{C}^{d_{\text{out}}} \tag{5.2}$$

Note that both $\mathrm{Re}(\mathbf{x}), \mathrm{Im}(\mathbf{x}) \in \mathbb{R}^{d_{\text{in}}}$. Second, separate trainable bias vectors $\mathbf{b}_{\mathbf{m}}, \mathbf{b}_{\boldsymbol{\phi}} \in \mathbb{R}^{d_{\text{out}}}$ are applied to the magnitude and phase components of $\boldsymbol{\psi} \in \mathbb{C}^{d_{\text{out}}}$:

$$\mathbf{m}_{\boldsymbol{\psi}} = |\boldsymbol{\psi}| + \mathbf{b}_{\mathbf{m}} \quad \in \mathbb{R}^{d_{\text{out}}} \quad ; \quad \boldsymbol{\phi}_{\boldsymbol{\psi}} = \arg(\boldsymbol{\psi}) + \mathbf{b}_{\boldsymbol{\phi}} \quad \in \mathbb{R}^{d_{\text{out}}} \tag{5.3}$$

Third, the CAE uses an additional gating function proposed by Reichert and Serre [2014] to further transform this "intermediate" magnitude $\mathbf{m}_{\boldsymbol{\psi}} \in \mathbb{R}^{d_{\text{out}}}$. This gating function dampens the response of an output unit as a function of the phase difference between two inputs. It is designed such that the corresponding response curve approximates experimental recordings of the analogous curve from a Hodgkin-Huxley model of a biological neuron [Reichert and Serre, 2014]. Concretely, an intermediate activation vector $\boldsymbol{\chi} \in \mathbb{R}^{d_{\text{out}}}$ (called *classic term* [Reichert and Serre, 2014]) is computed by applying f_{w} to the magnitude of the input $\mathbf{x} \in \mathbb{C}^{d_{\text{in}}}$, and a convex combination of this *classic term* and the magnitude $\mathbf{m}_{\boldsymbol{\psi}}$ (called *synchrony term* [Reichert and Serre, 2014]) from Eq. 5.3 is computed to yield "gated magnitudes" $\mathbf{m}_{\mathbf{z}} \in \mathbb{R}^{d_{\text{out}}}$ as follows:

$$\boldsymbol{\chi} = f_{\mathrm{w}}\left(|\mathbf{x}|\right) + \mathbf{b}_{\mathbf{m}} \quad \in \mathbb{R}^{d_{\text{out}}} \quad ; \quad \mathbf{m}_{\mathbf{z}} = \frac{1}{2}\mathbf{m}_{\boldsymbol{\psi}} + \frac{1}{2}\boldsymbol{\chi} \quad \in \mathbb{R}^{d_{\text{out}}} \tag{5.4}$$

Finally, the output of the layer $\mathbf{z} \in \mathbb{C}^{d_{\text{out}}}$ is obtained by applying non-linearities to this magnitude $\mathbf{m}_{\mathbf{z}}$ (Eq. 5.4) while leaving the phase values $\boldsymbol{\phi}_{\boldsymbol{\psi}}$ (Eq. 5.3) untouched:

$$\mathbf{z} = \mathrm{ReLU}(\mathrm{BatchNorm}(\mathbf{m}_{\mathbf{z}})) \odot e^{i\boldsymbol{\phi}_{\boldsymbol{\psi}}} \quad \in \mathbb{C}^{d_{\text{out}}} \tag{5.5}$$

The ReLU activation ensures that the magnitude of $\mathbf{z}$ is positive, and any phase flips are prevented by its application solely to the magnitude component $\mathbf{m}_{\mathbf{z}}$. For more details of the CAE [Löwe et al., 2022] and gating function [Reichert and Serre, 2014] we refer the readers to the respective papers. The final object grouping in CAE is obtained through K-means clustering based on the phases at the output of the decoder; each pixel is assigned to a cluster corresponding to an object [Löwe et al., 2022].

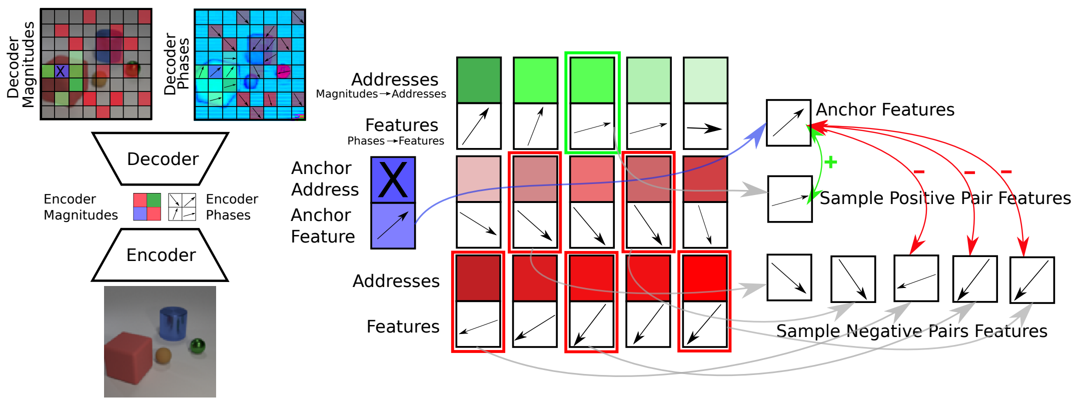


Figure 5.1: Sampling process of positive (green) and negative (red) pairs for one anchor (purple) in the CtCAE model. The sampling process here is visualized only for the decoder output. Note that we contrast the encoder output in an identical manner. Anchor address (the purple box marked with an X) corresponds to the patch of magnitude values and the feature corresponds to the phase values. See **Contrastive Training of CAEs** in Section 5.2 for more details.

5.2 Method

We describe the architectural and contrastive training details used by our proposed CAE++ and CtCAE models respectively below.

CAE++. We first propose some simple but crucial architectural modifications that enable the vanilla CAE [Löwe et al., 2022] to achieve good grouping performance on multi-object datasets such as `Tetrominoes` with color images. These architectural modifications include — i) Remove the `1x1` convolution kernel and associated sigmoid activation in the output layer ("f_{out}" in Löwe et al. [2022]) of the decoder, ii) Use convolution and upsample layers in place of transposed convolution layers in the decoder (cf. "f_{dec}" architecture in Table 3 from Löwe et al. [2022]). We term this improved CAE variant that adopts these architectural modifications as CAE++. As we will show below in Table 5.1, these modifications allow our CAE++ to consistently outperform the CAE across all 3 multi-object datasets with color images.

Contrastive Training of CAEs. Despite the improved grouping of CAE++ compared to CAE, we still empirically observe that CAE++ shows poor *separa-*

bility[1] (we also illustrate this in Section 5.3). This motivates us to introduce an auxiliary training objective that explicitly encourages higher *separability* in phase space. For that, we propose a contrastive learning method [Gutmann and Hyvärinen, 2010; Oord et al., 2018] that modulates the distance between pairs of distributed representations based on some notion of (dis)similarity between them (which we define below). This design reflects the desired behavior to drive the phase separation process between two different objects thereby facilitating better grouping performance.

Before describing our contrastive learning method mathematically below, here we explain its essential ingredients (illustrated in Figure 5.1). For setting up the contrastive objective, we first (randomly) sample "anchors" from a set of "datapoints". The main idea of contrastive learning is to "contrast" these anchors to their respective "positive" and "negative" examples in a certain representation space. This requires us to define two representation spaces: one associated with the similarity measure to define positive/negative examples given an anchor, and another one on which we effectively apply the contrastive objective, itself defined as a certain distance function to be minimized. We use the term *addresses* to refer to the representations used to measure similarity, and consequently extract positive and negative pairs w.r.t. to the anchor. We use the term *features* to refer to the representations that are contrasted. As outlined earlier, the goal of the contrastive objective is to facilitate *separability* of phase values. It is then a natural choice to use the phase components of complex-valued outputs as *features* and the magnitude components as *addresses* in the contrastive loss. This results in angular distance between phases (*features*) being modulated by the contrastive objective based on how (dis)similar their corresponding magnitude components (*addresses*) are. Since the magnitude components of complex-valued activations are used to reconstruct the image (Equation (5.7)), they capture requisite visual properties of objects. In short, the contrastive objective increases or decreases the angular distance of phase components of points (pixels/image regions) in relation to how (dis)similar their visual properties are.

Our contrastive learning method works as follows. Let h', w', d_{f}, N_A, and M denote positive integers. Here we generically denote the dimension of the output of any CAE layer as $h' \times w' \times d_{\mathrm{f}}$. This results in a set of $h' \times w'$ "datapoints" of dimension d_{f} for our contrastive learning. From this set of datapoints, we randomly sample N_A anchors. We denote this set of anchors as a matrix $\mathbf{A} \in \mathbb{R}^{N_A \times d_{\mathrm{f}}}$; each anchor is thus denoted as $\mathbf{A}_k \in \mathbb{R}^{d_{\mathrm{f}}}$ for all $k \in \{1, ..., N_A\}$. Now by

[1]We define separability as the minimum angular distance of phase values between a pair of prototypical points (centroids) that belong to different objects

Algorithm 1: Mining positive and negative pairs for a single anchor for the contrastive objective. Scalar parameters are k_{top} — the number of candidates from which to sample one positive pair and m_{bottom} — the number of candidates from which to sample $(M-1)$ negative pairs.

```
Inputs: anchor_address $\in \mathbb{R}^{d_f}$, addresses $\in \mathbb{R}^{h' \times w' \times d_f}$ , features $\in \mathbb{R}^{h' \times w' \times d_f}$.
Params: Scalars $k_{\text{top}}, m_{\text{bottom}}, M$.
 1: addresses $\leftarrow$ Flatten(addresses, dims=(0,1))
 2: features $\leftarrow$ Flatten(features, dims=(0,1))
 3: distances $\leftarrow$ CosineDistance(addresses, anchor_address)
 4: top_k_features $\leftarrow$ features[argsort(distances, dim=0)[:$k_{\text{top}}$]]
 5: bottom_m_features $\leftarrow$ features[argsort(distances,
    dim=0)[$-m_{\text{bottom}}$:]]
 6: pos_pair_idx $\sim$ Uniform [0, $k_{\text{top}}$]    ▷ sample 1 positive pair
 7: neg_pair_idxs $\sim$ Uniform [0, $m_{\text{bottom}}$]$\times(M-1)$    ▷ $M-1$ samples
    without replacement
 8: positive_pair $\leftarrow$ top_k_features[pos_pair_idx]
 9: negative_pairs $\leftarrow$ bottom_m_features[neg_pair_idxs]
10: return positive_pair, negative_pairs
```

using Algorithm 1, we extract 1 positive and $M-1$ negative examples for each anchor. We denote these examples by a matrix $\mathbf{P}^k \in \mathbb{R}^{M \times d_f}$ for each anchor $k \in \{1, ..., N_A\}$ arranged such that $\mathbf{P}^k_1 \in \mathbb{R}^{d_f}$ is the positive example and all other rows $\mathbf{P}^k_j \in \mathbb{R}^{d_f}$ for $j \in \{2, ..., M\}$ are negative ones. Finally, our contrastive loss is an adaptation of the standard InfoNCE loss [Oord et al., 2018] which is defined as follows:

$$\mathcal{L}_{\text{ct}} = \frac{1}{N_A} \sum_{k=1}^{N_A} \log \left(\frac{\exp\left(d\left(\mathbf{A}_k; \mathbf{P}^k_1\right)/\tau\right)}{\sum_{j=1}^{M} \exp\left(d\left(\mathbf{A}_k; \mathbf{P}^k_j\right)/\tau\right)} \right) \tag{5.6}$$

where N_A is the number of anchors sampled for each input, $d(\mathbf{x}_k; \mathbf{x}_l)$ refers to the cosine distance between a pair of vectors $\mathbf{x}_k, \mathbf{x}_l \in \mathbb{R}^{d_f}$ and $\tau \in \mathbb{R}_{>0}$ is the softmax temperature.

We empirically observe that applying the contrastive loss on outputs of both encoder and decoder is better than applying it on only either one (Table 5.5). We hypothesize that this is the case because it utilizes both high-level, abstract and global features (on the encoder-side) as well as low-level and local visual cues (on the decoder-side) that better capture visual (dis)similarity between positive and negative pairs. We also observe that using magnitude components of complex outputs of both the encoder and decoder as *addresses* for mining positive and negative pairs while using the phase components of complex-valued outputs as the *features* for the contrastive loss performs the best among all the other possible alternatives (Table C.8). These ablations also support our initial intuitions (described above) while designing the contrastive objective for improving *separability* in phase space.

Finally, the complete training objective function of CtCAE is:

$$\mathcal{L} = \mathcal{L}_{\text{mse}} + \beta \cdot \mathcal{L}_{\text{ct}} \quad ; \quad \mathcal{L}_{\text{mse}} = ||\mathbf{x}' - \widehat{\mathbf{x}}||_2^2 \quad ; \quad \widehat{\mathbf{x}} = |\mathbf{y}| \tag{5.7}$$

where $\mathcal{L}$ defines the loss for a single input image $\mathbf{x}' \in \mathbb{R}^{h \times w \times 3}$, and $\mathcal{L}_{\text{mse}}$ is the standard reconstruction loss used by the CAE [Löwe et al., 2022]. The reconstructed image $\widehat{\mathbf{x}} \in \mathbb{R}^{h \times w \times 3}$ is generated from the complex-valued outputs of the decoder $\mathbf{y} \in \mathbb{C}^{h \times w \times 3}$ by using its magnitude component. In practice, we train all models by minimizing the training loss $\mathcal{L}$ over a batch of images. The CAE baseline model and our proposed CAE++ variant are trained using only the reconstruction objective (i.e. $\beta = 0$) whereas our proposed CtCAE model is trained using the complete training objective.

5.3 Results

Here we provide our experimental results. We first describe details of the datasets, baseline models, training procedure and evaluation metrics. We then show results (always across 5 seeds) on grouping of our CtCAE model compared to the baselines (CAE and our variant CAE++), *separability* in phase space, generalization capabilities w.r.t to number of objects seen at train/test time and ablation studies for each of our design choices. Finally, we comment on the limitations of our proposed method.

Datasets. We evaluate the models on three datasets from the Multi-Object datasets suite [Kabra et al., 2019] namely `Tetrominoes`, `dSprites` and `CLEVR` (Figure 5.2) used by prior work in object-centric learning [Greff et al., 2019; Locatello et al., 2020; Emami et al., 2021]. For `CLEVR`, we use the filtered version [Emami et al., 2021] which consists of images containing less than seven objects. For the main evaluation, we use the same image resolution as Emami et al. [2021], i.e., 32x32 for `Tetrominoes`, 64x64 for `dSprites` and 96x96 for `CLEVR` (a center crop of 192x192 that is then resized to 96x96). For computational reasons, we perform all ablations and analysis on 32x32 resolution. Performance

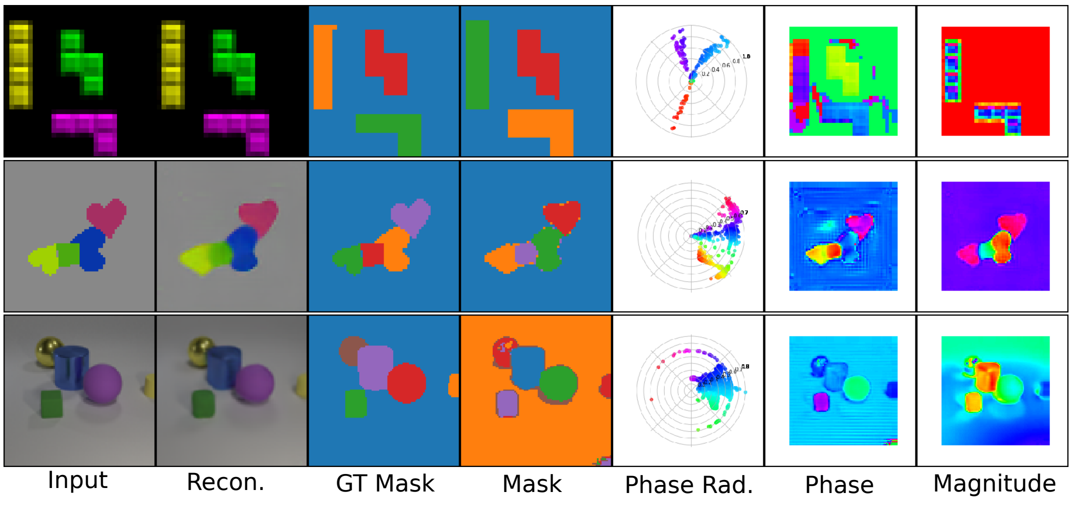


Figure 5.2: Unsupervised object discovery on `Tetrominoes`, `dSprites` and `CLEVR` with CtCAE. "Phase Rad." (col. 5) is the radial plot with the phase values from -π to π radians. "Phase" (col. 6), are phase values (in radians) averaged over the 3 output channels as a heatmap (colors correspond to those from "Phase Rad.") and "Magnitude" (col. 7) is the magnitude component of the outputs.

of all models are ordered in the same way on 32x32 resolution as the original resolution (see Table 5.1), but with significant training and evaluation speed up. In `Tetrominoes` and `dSprites` the number of training images is 60K whereas in `CLEVR` it is 50K. All three datasets have 320 test images on which we report all the evaluation metrics. For more details about the datasets and preprocessing, please refer to Appendix C.1.

Models & Training Details. We compare our CtCAE model to the state-of-the-art synchrony-based method for unsupervised object discovery (CAE [Löwe et al., 2022]) as well as to our own improved version thereof (CAE++) introduced in Section 5.2. For more details about the encoder and decoder architecture of all models see Appendix C.1. We use the same architecture as CAE [Löwe et al., 2022], except with increased number of convolution channels (same across all models). We train models for 50K steps on `Tetrominoes`, and 100K steps on `dSprites` and `CLEVR` with Adam optimizer [Kingma and Jimmy Ba, 2015] with a constant learning rate of 4e-4, i.e., no warmup schedules or annealing (all hyperparameter details are given in Appendix C.1).

Dataset	Model	MSE ↓	ARI-FG ↑	ARI-FULL ↑
Tetrominoes (32x32)	CAE	4.57e-2 ± 1.08e-3	0.00 ± 0.00	0.12 ± 0.02
	CAE++	5.07e-5 ± 2.80e-5	0.78 ± 0.07	0.84 ± 0.01
	CtCAE	9.73e-5 ± 4.64e-5	**0.84 ± 0.09**	**0.85 ± 0.01**
	SlotAttention	–	0.99 ± 0.00	–
dSprites (64x64)	CAE	8.16e-3 ± 2.54e-5	0.05 ± 0.02	0.10 ± 0.02
	CAE++	1.60e-3 ± 1.33e-3	0.51 ± 0.08	0.54 ± 0.14
	CtCAE	1.56e-3 ± 1.58e-4	**0.56 ± 0.11**	**0.90 ± 0.03**
	SlotAttention	–	0.91 ± 0.01	–
CLEVR (96x96)	CAE	1.50e-3 ± 4.53e-4	0.04 ± 0.03	0.18 ± 0.06
	CAE++	2.41e-4 ± 3.45e-5	0.27 ± 0.13	0.31 ± 0.07
	CtCAE	3.39e-4 ± 3.65e-5	**0.54 ± 0.02**	**0.68 ± 0.08**
	SlotAttention	–	0.99 ± 0.01	–
dSprites (32x32)	CAE	7.24e-3 ± 8.45e-5	0.01 ± 0.00	0.05 ± 0.00
	CAE++	8.67e-4 ± 1.92e-4	0.38 ± 0.05	0.49 ± 0.15
	CtCAE	1.10e-3 ± 2.59e-4	**0.48 ± 0.03**	**0.68 ± 0.13**
CLEVR (32x32)	CAE	1.84e-3 ± 5.68e-4	0.11 ± 0.07	0.12 ± 0.11
	CAE++	4.04e-4 ± 4.04e-4	0.22 ± 0.10	0.30 ± 0.18
	CtCAE	9.88e-4 ± 1.42e-3	**0.50 ± 0.05**	**0.69 ± 0.25**

Table 5.1: MSE and ARI scores (mean ± standard deviation across 5 seeds) for CAE, CAE++ and CtCAE models for `Tetrominoes`, `dSprites` and `CLEVR` on their respective full resolutions. For all datasets, CtCAE vastly outperforms CAE++ which in turn outperforms the CAE baseline. Results for 32x32 `dSprites` and `CLEVR` are also provided, these follow closely the scores on the full resolutions. SlotAttention results are from Emami et al. [2021].

Evaluation Metrics. We use the same evaluation protocol as prior work [Greff et al., 2019; Locatello et al., 2020; Emami et al., 2021; Löwe et al., 2022] which compares the grouping performance of models using the Adjusted Rand Index (ARI) [Rand, 1971; Hubert and Arabie, 1985]. We report two variants of the ARI score [Hubert and Arabie, 1985], i.e., ARI-FG and ARI-FULL consistent with Löwe et al. [2022]. ARI-FG measures the ARI score only for the foreground and ARI-FULL takes into account all pixels.

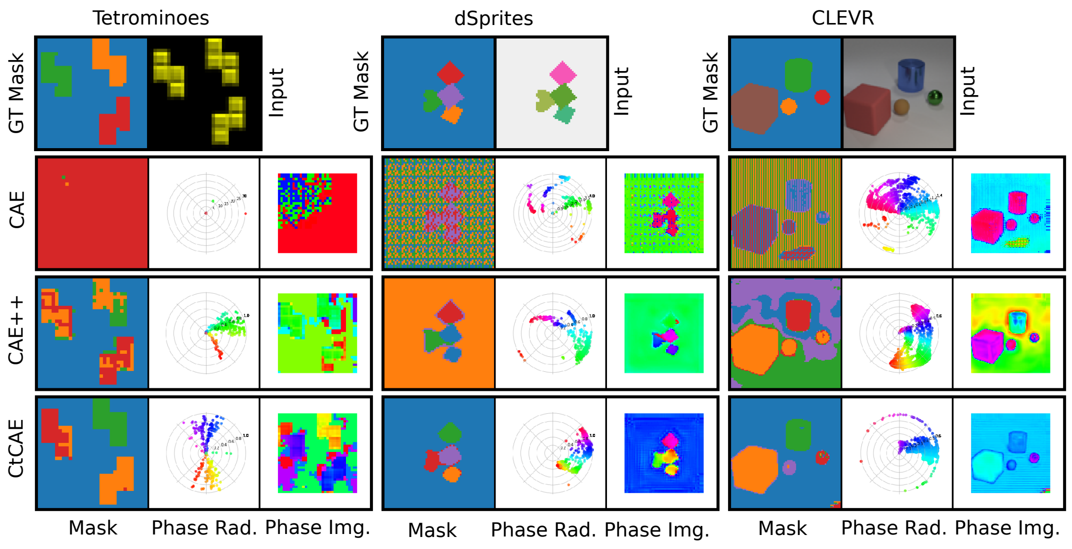


Figure 5.3: CAE, CAE++ and CtCAE comparison on `Tetrominoes` (columns 1-3), `dSprites` (columns 4-6) and `CLEVR` (columns 7-9). First row: ground truth masks and input images.

Unsupervised Object Discovery. Table 5.1 shows the performance of our CAE++, CtCAE, and the baseline CAE [Löwe et al., 2022] on `Tetrominoes`, `dSprites` and `CLEVR`. We first observe that the CAE baseline almost completely fails on all datasets as shown by its very low ARI-FG and ARI-FULL scores. The MSE values in Table 5.1 indicate that CAE even struggles to reconstruct these color images. In contrast, CAE++ achieves significantly higher ARI scores, consistently across all three datasets; this demonstrates the impact of the architectural modifications we propose. However, on the most challenging `CLEVR` dataset, CAE++ still achieves relatively low ARI scores. Its contrastive learning-augmented counterpart, CtCAE consistently outperforms CAE++ both in terms of ARI-FG and ARI-FULL metrics. Notably, CtCAE achieves more than double the ARI scores of CAE++ on the most challenging `CLEVR` dataset which highlights the benefits of our contrastive method. All these results demonstrate that CtCAE is capable of object discovery (still far from perfect) on all datasets which include color images and more than three objects per scene, unlike the exisiting state-of-the-art synchrony-based model, CAE.

Quantitative Evaluation of *Separability*. To gain further insights into why our contrastive method is beneficial, we quantitatively analyse the phase maps using two distance metrics: *inter-cluster* and *intra-cluster* distances. In fact, in all

Dataset	Model	Inter-cluster (min) ↑	Inter-cluster (mean) ↑	Intra-cluster ↓
Tetrominoes	CAE++	0.14 ± 0.00	0.30 ± 0.02	0.022 ± 0.010
	CtCAE	**0.15 ± 0.01**	**0.31 ± 0.03**	**0.020 ± 0.010**
dSprites	CAE++	**0.13 ± 0.05**	**0.51 ± 0.05**	0.034 ± 0.007
	CtCAE	**0.13 ± 0.03**	0.39 ± 0.10	**0.027 ± 0.009**
CLEVR	CAE++	0.10 ± 0.06	**0.53 ± 0.15**	0.033 ± 0.013
	CtCAE	**0.12 ± 0.05**	0.50 ± 0.12	**0.024 ± 0.005**

Table 5.2: Quantifying the *Separability* through inter- and intra-cluster metrics of the phase space. For the inter-cluster metric, we report both the minimum and mean across clusters.

CAE-family of models, final object grouping is obtained through K-means clustering based on the phases at the output of the decoder; each pixel is assigned to a cluster with the corresponding centroid. *Inter*-cluster distance measures the Euclidean distance between centroids of each pair of clusters averaged over all such pairs. Larger inter-cluster distance allows for easier discriminability during clustering to obtain object assignments from phase maps. On the other hand, *intra*-cluster distance quantifies the "concentration" of points within a cluster, and is computed as the average Euclidean distance between each point in the cluster and the cluster centroid. Smaller intra-cluster distance results in an easier clustering task as the clusters are then more condensed. We compute these distance metrics on a per-image basis before averaging over all samples in the dataset. The results in Table 5.2 show that, the mean intra-cluster distance (last column) is smaller for CtCAE than CAE++ on two (dSprites and CLEVR) of the three datasets. Also, even though the average inter-cluster distance (fourth column) is sometimes higher for CAE++, the *minimum* inter-cluster distance (third column)—which is a more relevant metric for separability—is larger for CtCAE. This confirms that compared to CAE++, CtCAE tends to have better phase map properties for object grouping, as is originally motivated by our contrastive method.

Object Storage Capacity. Löwe et al. [2022] note that the performance of CAE sharply decreases for images with more than 3 objects. We report the performance of CtCAE and CAE++ on subsets of the test set split by the number of objects, to measure how their grouping performance changes w.r.t. the num-

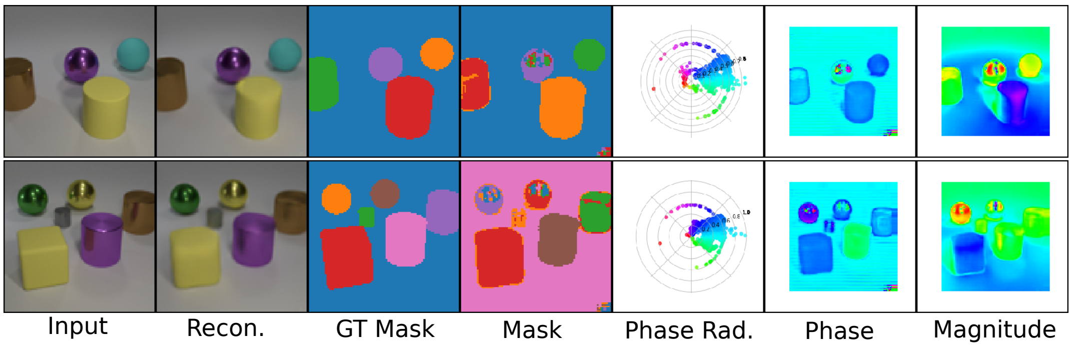


Figure 5.4: CtCAE on `CLEVR` is able to infer more than four objects, although it sometimes makes mistakes, such as specular effects or grouping together based on color (two yellow objects).

	ARI-FG	ARI-FULL	ARI-FG	ARI-FULL	ARI-FG	ARI-FULL	ARI-FG	ARI-FULL
dSprites	2 objects		3 objects		4 objects		5 objects	
CAE++	0.29	0.58	0.39	0.53	0.43	0.45	0.45	0.44
CtCAE	**0.45**	**0.76**	**0.48**	**0.72**	**0.48**	**0.65**	**0.48**	**0.61**
CLEVR	3 objects		4 objects		5 objects		6 objects	
CAE++	0.32	0.35	0.33	0.32	0.31	0.26	0.32	0.27
CtCAE	**0.53**	**0.71**	**0.52**	**0.70**	**0.47**	**0.67**	**0.45**	**0.68**

Table 5.3: Object storage capacity: training on the full `dSprites` dataset and evaluating separately on subsets containing images with 2, 3, 4 or 5 objects. Analogously, training on the full `CLEVR` dataset and evaluating separately on subsets containing images with 3, 4, 5 or 6 objects.

ber of objects. In `dSprites` the images contain $2, 3, 4$ or 5 objects, in `CLEVR` $3, 4, 5$ or 6 objects. Note that the models are trained on the entire training split of the respective datasets. Table 5.3 shows that both methods perform well on images containing more than 3 objects (their performance does not drop much on images with 4 or more objects). We also observe that the CtCAE consistently maintains a significant lead over CAE in terms of ARI scores across different numbers of objects. In another set of experiments (see Appendix C.2), we also show that CtCAE generalizes well to more objects (e.g. 5 or 6) when trained only on a subset of images containing less than this number of objects. Additionally, we observe that CtCAE generalizes well to more objects (e.g. 5 or 6) when trained only on a subset of images containing less than this number of

Model	ARI-FG ↑	ARI-FULL ↑
CAE	0.00 ± 0.00	0.12 ± 0.02
CAE-(f_{out} `1x1` conv)	0.00 ± 0.00	0.00 ± 0.00
CAE-(f_{out} sigmoid)	0.12 ± 0.12	0.35 ± 0.36
CAE-transp. + upsamp.	0.10 ± 0.21	0.10 ± 0.22
CAE++	0.78 ± 0.07	0.84 ± 0.01
CtCAE	0.84 ± 0.09	0.85 ± 0.01

Table 5.4: Architectural ablations on `Tetrominoes`.

objects (see Appendix C.2 for results).

Ablation on Architectural Modifications. Table 5.4 shows an ablation study on the proposed architectural modifications on `Tetrominoes` (for similar findings on other datasets, see Appendix C.2.2). We observe that the sigmoid activation on the output layer of the decoder significantly impedes learning on color datasets. A significant performance jump is also observed when replacing transposed convolution layers [Löwe et al., 2022] with convolution and upsample layers. By applying all these modifications, we obtain our CAE++ model that results in significantly better ARI scores on all datasets, therefore supporting our design choices.

Ablation on Feature Layers to Contrast. In CtCAE we contrast feature vectors both at the output of the encoder and the output of the decoder. Table 5.5 justifies this choice; this default setting (Enc + Dec) outperforms both other options where we apply the constrastive loss either only in the encoder or the decoder output. We hypothesize that this is because these two contrastive strategies are complementary: one uses low-level cues (dec) and the other high-level abstract features (enc).

Qualitative Evaluation of Grouping. Finally, we conduct some qualitative analyses of both successful and failed grouping modes shown by CAE++ and CtCAE models through visual inspection of representative samples. In Figure 5.3, `Tetrominoes` (columns 1-2), we observe that CtCAE (row 4) exhibits better grouping on scenes with multiple objects of the same color than CAE++ (row 3). This is reflected in the radial phase plots (column 2) which show better *sepa-*

Model	ARI-FG ↑	ARI-FULL ↑
Enc-only	0.21 ± 0.11	0.29 ± 0.15
Dec-only	0.38 ± 0.17	0.69 ± 0.18
Enc+Dec	0.50 ± 0.05	0.69 ± 0.25

Table 5.5: Contrastive loss ablation for CtCAE on `CLEVR`.

rability for CtCAE than CAE++. Further, on `dSprites` (rows 3-4, columns 4-5) and `CLEVR` (rows 3-4, columns 7-8), CtCAE handles the increased number of objects more gracefully while CAE++ struggles and groups several of them together. For the failure cases, Figure 5.4 shows an example where CtCAE still has some difficulties in segregating objects of the same color (row 2, yellow cube and ball) (also observed sometimes on `dSprites`, see Figure C.10). Further, we observe how the *specular highlights* on metallic objects (purple ball in row 1 and yellow ball in row 2) form a separate sub-part from the object (additional grouping examples in Appendix C.3).

5.4 Related Work

Slot-based binding. A wide range of unsupervised models have been introduced to perform perceptual grouping summarized well by Greff et al. [2020]. They categorize models based on the segregation (routing) mechanism used to break the symmetry in representations and infer latent representations (i.e. slots). Models that use "instance slots" cast the routing problem as inference in a mixture model whose solution is given by amortized variational inference [Greff et al., 2016, 2019], Expectation-Maximization [Greff et al., 2017] or other approximations (Soft K-means) thereof [Locatello et al., 2020]. While others [Eslami et al., 2016; Kosiorek et al., 2018; Stanić and Schmidhuber, 2019] that use "sequential slots" solve the routing problem by imposing an ordering across time. These models use recurrent neural networks and an attention mechanism to route information about a different object into the same slot at every timestep. Some models [Burgess et al., 2019; Engelcke et al., 2020] combine the above strategies and use recurrent attention only for routing but not for inferring slots. Other models break the representational symmetry based on spatial coordinates [Crawford and Pineau, 2019; Lin et al., 2020] ("spatial slots") or based on specific object types [Hinton et al., 2018] ("category slots"). All these models still maintain the "separation" of representations only at one latent layer (slot-level) but continue to "entangle" them at other layers.

Synchrony-based binding. Synchrony-based models use complex-valued activations to implement binding by relying on their constructive or destructive phase interference phenomena. This class of models have been sparsely explored with only few prior works that implement this conceptual design for object binding [Mozer et al., 1991; Mozer, 1998; Reichert and Serre, 2014; Löwe et al., 2022]. These methods differ based on whether they employ both complex-valued weights and activations [Mozer et al., 1991; Mozer, 1998] or complex-valued activations with real-valued weights and a gating mechanism [Reichert and Serre, 2014; Löwe et al., 2022]. They also differ in their reliance on explicit supervision for grouping [Mozer et al., 1991] or not [Mozer, 1998; Reichert and Serre, 2014; Löwe et al., 2022]. Synchrony-based models in contrast to slot-based ones maintain the "separation" of representations throughout the network in the phase components of their complex-valued activations. However, none of these prior methods can group objects in color images with up to 6 objects or visual realism of multi-object benchmarks in a fully unsupervised manner unlike ours. Concurrent work [Löwe et al., 2023] extends CAE by introducing new feature dimensions ("rotating features"; RF) to the complex-valued activations. However, RF cannot be directly compared to CtCAE as it relies on either depth masks to get instance grouping on simple colored `Shapes` or features from powerful vision backbones (DINO [Caron et al., 2021]) to get largely semantic grouping on real-world images such as multiple instances of cows/trains or bicycles in their Figures 2 and 20 respectively.

Binding in the brain. The temporal correlation hypothesis posits that the mammalian brain binds together information emitted from groups of neurons that fire synchronously. According to this theory [von der Malsburg, 1995; Singer and Gray, 1995], biological neurons transmit information in two ways through their spikes. The spike amplitude indicates the strength of presence of a feature while relative time between spikes indicates which neuronal responses need to bound together during further processing. It also suggests candidate rhythmic cycles in the brain such as Gamma that could play this role in binding [Jefferys et al., 1996; Fries et al., 2007]. Synchrony-based models functionally implement the same coding scheme [von der Malsburg and Schneider, 1986; von der Malsburg and Buhmann, 1992; Engel et al., 1992] using complex-valued activations where the relative phase plays the role of relative time between spikes. This abstracts away all aspects of the spiking neuron model to allow easier reproduction on digital hardware.

Contrastive learning with object-centric models. Contrastive learning for object-centric representations has not been extensively explored, with a few notable exceptions. The first method [Löwe et al., 2020] works only on toy images of up to 3 objects on a black background while the second [Baldassarre and Azizpour, 2022] shows results on more complex data, but requires complex hand-crafted data augmentation techniques to contrast samples across a batch. Our method samples positive and negative pairs *within* a single image and does not require any data augmentation. Most importantly, unlike ours, these models still use slots and attention-based routing, and thereby inherit all of its conceptual limitations. Lastly, ODIN [Hénaff et al., 2022] alternately refines the segmentation masks and representation of objects using two networks that are jointly optimized by a contrastive objective that maximizes the similarity between different views of the same object, while minimizing the similarity between different objects. To obtain the segmentation masks for objects, they simply spatially group the features using K-means. Their method relies on careful data augmentation techniques such as local or global crops, viewpoint changes, color perturbations, etc. which are applied to one object.

5.5 Conclusion and Discussion

We propose several architectural improvements and a novel contrastive learning method to address the limitations of the current state-of-the-art synchrony-based model for object binding, the complex-valued autoencoder (CAE [Löwe et al., 2022]). Our improved architecture, CAE++, is the first synchrony-based model capable of dealing with color images (e.g. `Tetrominoes`). Our contrastive learning method further boosts CAE++ by improving its phase separation process. The resulting model, CtCAE, largely outperforms CAE++ on the rather challenging `CLEVR` and `dSprites` datasets.

Admittedly, our synchrony-based models still lag behind the state-of-the-art *slot*-based models [Locatello et al., 2020; Singh et al., 2022], but this is to be expected, as research on modern synchrony-based models is still in its infancy. We hope that our work will inspire the community to invest greater effort in such promising models and alternative conceptual designs for object discovery in general.

Chapter 6

Compositional Visual Reasoning with LLMs as Programmers

Compositional visual question answering requires a model to answer questions about visual content in a compositional manner, involving multiple steps of reasoning or considering relationships between different objects or entities within an image. It is a complex task as it requires understanding both the visual information in an image and the structure of the question, and reasoning about how different visual elements relate to one another to generate the correct answer. Recently, large progress has been made on many such vision and language tasks by scaling end-to-end neural networks models in terms of size, training data, and compute [Alayrac et al., 2022; Chen et al., 2022b; Yu et al., 2022; Wang et al., 2022a; Gan et al., 2022; Lu et al., 2022; Li et al., 2023; Driess et al., 2023; Chen et al., 2023b,a]. However, even the largest state-of-the-art (SotA) models struggle in tasks that require compositional reasoning, ability to generalize, fine-grained spatial reasoning capabilities, and counting [Bugliarello et al., 2023; Paiss et al., 2023; Hsieh et al., 2023; Yuksekgonul et al., 2022; Zhao et al., 2022; Hendricks and Nematzadeh, 2021]. An example of such task is the following query: “Could the cookies on the table be equally distributed among children?” [Surís et al., 2023]. To solve this, the model needs to detect the cookies in the image, filter out the ones that are not on the table, detect children, count cookies and children, and check if the cookies count is divisible by the children count. Questions like these are difficult for current end-to-end vision and language models (VLMs). Scaling VLMs further makes them even more data- and compute-hungry [Villalobos et al., 2022], so the scale alone seems un-

This chapter is based on “Towards Truly Zero-Shot Compositional Visual Reasoning with LLMs as Programmers” [Stanić et al., 2023b] paper, that is currently under submission.

likely to solve these tasks, especially due to the exponentially-large long tail of compositional tasks.

On the other hand, it is questionable whether solving compositional tasks with a single monolithic end-to-end neural network is the optimal approach. Intuitively, it might be easier to first decompose the task into several subtasks, individually solve the subtasks, and then use the intermediate results to solve the original task. This is reminiscent of the way humans approach compositional problems. According to Daniel Kahneman's framework [Kahneman, 2017], our thought process can be thought of as consisting of two mechanisms: System 1 and System 2. System 2 is the "slow", "analytical" system that can decompose the task into subtasks, while System 1 is the "fast", "reactive" system that solves individual tasks such as recognizing patterns. In machine learning, the early work on task decomposition was pioneered by Neural Module Networks (NMNs) [Andreas et al., 2016; Johnson et al., 2017; Hu et al., 2017]. NMNs are trained end-to-end in the hope that every module will learn a separate function that will be reusable across tasks. However, these models have a number of drawbacks, namely that the program generation requires hand-tuned parsers, they are difficult to optimize (sometimes requiring reinforcement learning), and they have the issue of a module "collapse", where some modules are never activated and others take over all the work, contrary to the design intentions.

Recently, an alternative approach based on "tool use" gained popularity [Cobbe et al., 2021; Komeili et al., 2021; Thoppilan et al., 2022; Parisi et al., 2022; Zeng et al., 2022; Gao et al., 2023; Qin et al., 2023; Zhuge et al., 2023]. In "tool use", an LLM solves a task by controlling (akin to System 2) a set of tools (such as an object detector, akin to System 1) [Zeng et al., 2022; Shen et al., 2023; Zhuge et al., 2023]. In particular, VisProg [Gupta and Kembhavi, 2023], ViperGPT [Surís et al., 2023], and CodeVQA [Subramanian et al., 2023] show great promise in solving visual question answering by using an LLM to generate a program (in Python or a custom scripting language). During execution, the program calls individual vision models (such as object detector, depth estimator) through an *API* that is provided in the prompt. For example, to answer "Which color is the jacket of the second person from the left?" (Figure 6.1a), the program needs to detect people, sort them from left to right, select the second, detect their jacket, and query its color. These models achieved SotA on compositional visual question answering, visual grounding, and video temporal reasoning tasks. By their construction, they are interpretable, compositional, adaptable (tools can be upgraded on the fly), offer strong generalization, mathematical, and reasoning skills, and do not require gradient-based training. However, in their current form, they heavily rely on human engineering of in-context (program) examples

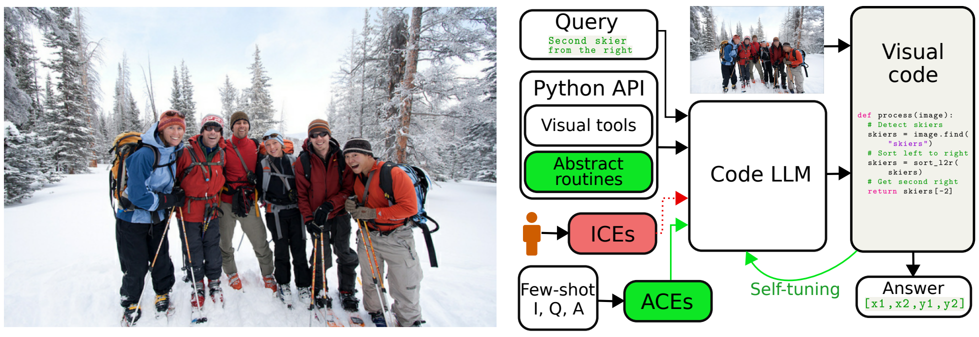
(ICEs) in the prompt. Moreover, ICEs are dataset- and task-specific. To generate them, significant labor is required by highly skilled programmers. For this reason, we argue that these methods *should not be called "zero-shot"* in their current form.

In this work, we present a framework that mitigates these issues, makes LLMs-as-programmers setup more robust, and removes the need for human engineering of ICEs. Our framework, whose effectiveness we show across a number of compositional question-answering and video temporal reasoning tasks with ViperGPT (but is universally applicable to other approaches), consists of the following:

- Firstly, instead of using a simple API with only basic routines that call individual tools, we introduce an "Abstract API". Abstract API consists of spatially and temporally abstract routines that remove the large burden on the LLM to have strong spatial and temporal reasoning.
- Second, instead of relying on a large number of dataset-specific (question, code)-pairs as ICEs, we introduce a setup that generates ICEs automatically. Using a few labeled examples (which are significantly cheaper to obtain, e.g. via crowd-sourcing), we generate query-code examples in a *zero-shot* manner and use these as ICEs. This mitigates the need for human engineering of ICEs.
- Third, we show how LLMs as controllers for visual reasoning can (to some extent) perform "self-correction" through "self-debugging" and "self-tuning" without any ground truth labels. In "self-debugging", we generate new code when the previous fails to execute, either by providing the LLM previous query-code pair and the execution error as feedback or from scratch. In "self-tuning", we show how the tool hyperparameters can be tuned automatically if execution fails due to a module.

6.1 LLMs as programmers for visual reasoning framework

In this section, we first provide a brief overview of the ViperGPT approach [Surís et al., 2023] on top of which we show the utility of our framework. We then describe each component of our framework, namely the Abstract API, automatic generation of ICEs, and self-correction.



```python
def process(image):
  # Detect skiers
  skiers = image.find(
      "skiers")
  # Sort left to right
  skiers = sort_l2r(
      skiers)
  # Get second right
  return skiers[-2]
```

(a) (b)

Figure 6.1: (a) RefCOCO [Yu et al., 2016] example image. (b) A code-generating LLM takes as input the query, the Python API (functions for "tool use" and Abstract API routines (functions) we introduce in Section 6.1.2) and a number of ICEs (we replace human-engineered ICEs by automatically-generated ACEs in Section 6.1.3). The LLM generates code that takes as input the image and outputs an answer (here a bounding box). If code fails to run, "self-tuning" (Section 6.1.4) can adjust parameters and generate new code.

6.1.1 Background

ViperGPT takes as input an image or a video and a textual query. The textual query is fed into an LLM (Codex [Chen et al., 2021]), together with the tools API and ICEs. The LLM generates a program that solves the given query using the tools without further training. The information in the prompt is crucial for good ViperGPT performance, as it is the only task-specific information provided. The prompt consists of a Python API with the necessary functions to solve the visual query, such as object detection, depth estimation, and language model queries. Additionally, ViperGPT uses several dataset-specific ICEs in the prompt. As we show in Section 6.2, performance depends heavily on these human-engineered examples.

ViperGPT API defines an `ImagePatch` and a `VideoSegment` class that contain image and video processing functions. Each function calls a pretrained model to compute the result. The API in the prompt does not contain function implementations, but it contains docstrings and query-code examples of their use. The ViperGPT API defines the following functions: `find` takes as input an image and a textual query, calls an open vocabulary detector and returns a list of image patches of detected objects; `exists` takes as input an image and a textual query and returns true if the query object exists in the image, otherwise false; `verify_property` takes as input an image, a noun representing an object and an attribute property to verify and returns a boolean whether the object has this property; `best_image_match` that takes as input a list of image patches and a textual query and returns the image patch that best matches the query; `best_text_match` that takes as input a list of queries and one image, and returns the query that best matches the image; `compute_depth` that computes the median depth of an image or image patch; `distance` which computes the pixel-distance between two images; `simple_query` which handles textual queries that are not decomposable, by calling an image captioning model; `select_answer` that given a context text describing a scene and a list of possible answers queries an LLM to select the correct answer. The `VideoSegment` class does not contain any functions that call individual models, but only the start and end point of the video segment and an iterator over the frames, which returns an `ImagePatch` object. For the full ViperGPT API, see Appendix D.4.

The code-generating LLM outputs code that attempts to solve the query. This code is executed, taking as input an image or a video (and optionally a list of possible answers) and outputs a result (e.g. a bounding box or a string). Due to generating programs in native Python code, ViperGPT avoids the need for custom interpreters and can leverage Python built-in functions (e.g. sort, if/else control flows, math functions, etc.).

6.1.2 Abstract API through visual routines

When programming, we continuously build new layers of abstraction. We start from a set of primitive functions and then abstract them away by adding new functions. These are then grouped into libraries, and additional layers of abstraction are built on top. By abstracting away the implementation details, we are able to build systems with ever-increasing capabilities and complexity. Motivated by this, we introduce a set of *spatially* and *temporally* abstract functions (routines [1]) that abstract away a number of lines of code for the same functionality and together make the *Abstract API*. From a practical perspective, we are motivated by a number of failure cases observed in the experiments (see Section 6.2). As presented in Section 6.1.1, ViperGPT's API is fairly simple (contains almost exclusively functions to call pretrained models). Although simplicity is good and often desirable, in the case of visual reasoning with LLMs as programmers, the lack of expressive visual routines requires the code-generating LLM to have strong spatial and temporal reasoning capabilities. Qualitative analysis

[1] Note that our "routines" do not correspond to the visual routines of Ullman [1987] such as tracing or scanning.

showed that this is often not the case and that the current LLMs are not yet perfect in these terms (e.g. they confuse left and right, top and bottom). For example, for the query "return the second person from the right", the program generated by the LLM correctly sorts the persons along the horizontal axis but then wrongly takes the second index in the array (instead of the second last). Similarly, they sometimes "confuse" temporal order, e.g., if a "before" event means a smaller or a larger time index.

For these reasons, we introduce a set of spatially and temporally abstract routines. We add the following spatial routines: `get_patch_left_of`, `get_patch_right_of`, `get_patch_above_of`, `get_patch_below_of` for relational retrieval relative to a patch; `get_patch_closest_to_anchor_object` that sorts patches by their distance to an anchor object and returns the one with the smallest distance; `sort_patches_left_to_right`, `sort_patches_bottom_to_top`, and `sort_patches_front_to_back` to sort the list of patches along horizontal, vertical or depth axis; `get_middle_patch` to get the middle patch from a given list of image patches; For videos, we add temporal routines for event "localization": `get_video_segment_of_event`, `get_video_segment_before_event`, `get_video_segment_after_event`, and routines to either caption a video: `caption_video` or answer a simple question about the video: `simple_query`. The routines that we introduce are *general* in the sense that they are not specific to any individual task or dataset. This facilitates their reuse across tasks and avoids engineering task and dataset-specific routines. It is an open research question what the "optimal" set of primitive routines is. Another exciting research direction is using LLM with their own abstract routines, then reusing those to come up with even more abstract routines and so on. We leave these for future work.

6.1.3 Automatic generation of in-context examples

In-context examples (ICEs) greatly influence the performance of LLMs [Brown et al., 2020; Chen et al., 2023]. For example, ViperGPT [Surís et al., 2023] uses between 10 and 16 hand-engineered dataset-specific query-code ICEs per dataset. Similarly, VisProg [Gupta and Kembhavi, 2023] uses between 16 and 31 ICEs and CodeVQA [Subramanian et al., 2023] about 50 ICEs. However, constructing these ICEs requires heavy human engineering, as they might need to be rewritten in a way that the LLM can use them to "generalize" to other examples across the dataset. Furthermore, the constructed examples are specific not only to the dataset but also to the LLM and the API. If any of those changes, they need to be written from scratch. Finally, to write good query-code ICEs, highly skilled labor is required, ideally someone familiar with the workings of LLMs and a good grasp of Python programming.

In our work, we move beyond this need for human engineering of query-code ICEs. We start from a small set of labeled examples (e.g. 16 image-question-answer tuples), as is common in few-shot transfer learning [Zhai et al., 2019; Kolesnikov et al., 2020]. We run our framework in a *zero-shot* manner (without any ICEs) on these few-shot examples, sort the results by accuracy, select the best-performing programs, pair them with the corresponding queries, and use them as ICEs during test time. We call such ICEs *automatically-generated in-context examples* (ACEs). Importantly, *no gradient-based optimization is performed on the few-shot examples*. Intuitively, this works since even if the LLM does not always generate a program that solves the task correctly, it might sometimes come up with a correct program. Since retrieval is often easier than generating programs from scratch, the reuse of the correct programs improves performance on the test set.

ACEs provide a number of benefits over manually writing ICEs. First of all, ACEs are much cheaper to obtain as they do not require highly skilled labor to write them. Second, the algorithm that generates ACEs is general: it is neither specific to the API nor the LLM. If any of these changes, ACEs can be easily generated by re-running the algorithm. Furthermore, they can be seen as a first step of the LLM "coming up" with its own abstract rules and thus creating a "rulebook" (discussed in Section 6.1.2). Finally, few-shot (image, question, answer)-labeled examples are often available in datasets typically used in machine learning. If not available, they are cheap to obtain via crowd-sourcing and can be reused for further studies as a benchmark.

6.1.4 Self-correction

One of the advantages of solving visual reasoning tasks with LLMs as programmers is that we know when code fails to execute. The failure can happen, e.g. due to a compilation error (e.g. due to hallucination), some of the models failing, or a wrong return type (e.g. a bounding-box is expected, but code returns a string). Note that to detect these types of errors, no ground-truth labels are needed.

Self-debugging. If the code execution fails, we can query the code-generating LLM to correct the previously generated code. We do this by feeding back the query, the previously generated code, and the resulting error in the prompt (see the feedback template in Appendix D.3). Moreover, if the LLM's sampling temperature is higher than zero, we can query the model with a different random seed to generate new code from scratch. There are advantages to both of these approaches. If the code-generating LLM has good "self-correction" abilities, then it should be able to correct its own mistakes based on the feedback, as we humans could. However, if the LLM is not good at self-correction or does not know how to incorporate such feedback (e.g. if the LLM is not trained to be conversational), then feeding back the previous query and code will only "bias" the model to output the same solution. In that case, generating new code from scratch could work better.

Self-tuning. In some cases, we know that code execution failed due to some components of a particular module. For example, the open vocabulary detector fails due to a too high threshold hyperparameter. When the threshold is high, we have a higher number of false negatives. For such cases, we propose to automatically change the hyperparameter of the module (e.g. reduce the threshold) and execute code again.

6.2 Experiments

Tasks. We evaluate our method on four datasets: RefCOCO, RefCOCO+ [Yu et al., 2016], GQA [Hudson and Manning, 2019] and NExT-QA [Xiao et al., 2021] used in previous work [Surís et al., 2023]. These datasets evaluate a diverse set of capabilities, namely visual grounding (RefCOCO, RefCOCO+), compositional image question answering (GQA), and video temporal reasoning (NExT-QA). In RefCOCO (example in Figure 6.1a), the task is to detect a bounding box of an object given its natural language description ("referring expression"). In compositional question answering in GQA, the task is to answer in natural language a compositional natural language query. We use the "test-dev" split of the GQA dataset, as in ViperGPT. In NExT-QA, the task is to answer a temporally compositional question by selecting one of the given multiple choice options. As in ViperGPT, we use NExT-QA "hard" split [Buch et al., 2022]. For RefCOCO and RefCOCO+, methods are evaluated in terms of Intersection over Union (IoU) between the detected and the ground truth bounding box and for GQA and NExT-QA in terms of accuracy of the predicted answer.

Vision and Language Models. For code generation, we use a code instruction-tuned version of PaLM 2 [Anil et al., 2023] `code-bison` accessible via the Google

Model	RefCOCO (IoU)	RefCOCO+ (IoU)	GQA (acc.)	NExT-QA (acc.)
Zero-Shot (ZS) SotA	53.0	57.5	44.7	38.3
Few-Shot (FS) SotA	53.3	52.5	35.7	38.3
Supervised (Sup) SotA	94.0	91.7	72.1	63.1
ViperGPT (paper)	72.0	67.0	48.1	60.0
ViperGPT (GitHub (GH) ZS)	46.7	-	-	-
ViperGPT (GH w/ DS-ICEs)	60.5	-	-	-
E2E bsl.(ZS OWLv2/PaLI-3)	33.5	31.7	40.1	58.9
E2E LLM-only baseline	-	-	-	53.3
Ours (`code-bison`, Zero-Shot)	44.4 ± 0.9	38.2 ± 0.0	32.1 ± 0.4	60.2 ± 0.3
Ours (`code-bison`)	51.2 ± 0.2	45.7 ± 0.1	33.4 ± 0.2	61.0 ± 0.1

Table 6.1: Comparison of our method against previous end-to-end and "LLMs as controllers" SotA methods. For "Ours (`code-bison`)", we report mean scores $\pm$ standard deviation across three random seeds. The reference numbers for SotA on each dataset are taken from the following publications: RefCOCO: ZS [Yang et al., 2023b], FS [Yao et al., 2021], Sup [Wang et al., 2022b]; RefCOCO+: ZS [Yang et al., 2023b], FS [Yao et al., 2021], Sup [Wang et al., 2022b]; GQA: ZS [Li et al., 2023], FS [Jin et al., 2021], Sup [Nguyen et al., 2022]; NExT-QA: ZS [Chen et al., 2023b], FS [Chen et al., 2023b], Sup [Ye et al., 2023].

Cloud API [Google, 2023]. We use the same model to select an answer for the multichoice questions in the NExT-QA dataset. Vision models we use are OWLv2 [Minderer et al., 2023] for object detection, SigLiT [Zhai et al., 2023] for text-image comparison, MiDaS [Ranftl et al., 2020] for depth estimation, and PaLI-3 [Chen et al., 2023b] for image captioning and answering visual queries. Note that all models are different from the models that ViperGPT used (see Appendix D.2).

Baselines. Strong baselines are essential for correctly measuring progress in machine learning. This is especially true in the emerging area of "tool use" [Cobbe et al., 2021; Komeili et al., 2021; Thoppilan et al., 2022; Parisi et al., 2022; Zeng et al., 2022; Gao et al., 2023; Qin et al., 2023; Zhuge et al., 2023]. When using an LLM and other pre-trained models, we must be careful to report the exact LLM version and/or API when it was accessed, and ideally report results over several random seeds to measure the statistical significance. In Table 6.1, we provide an overview of the previous Zero-Shot (ZS), Few-Shot (FS), and Supervised (Sup) SotA methods, ViperGPT, end-to-end (E2E) baselines, and our results on all datasets we used for evaluation.

Early in the project, we found it difficult to reproduce the results reported in the ViperGPT paper. Our first hypothesis is that this is due to differences in the vision and language models we use compared to the ViperGPT paper. However, when running the original ViperGPT code from the official GitHub repository on RefCOCO, we were only able to achieve an IoU of 60.5 as opposed to 72.0 reported in the paper. Note, however, that ViperGPT uses Codex, which is discontinued, so we use `GPT-3.5-turbo` [OpenAI, 2023b]. Also note that this score was obtained using 16 *dataset-specific* ICEs (DS-ICEs). These examples contain large amounts of dataset-specific human-engineered information, such as "clothing requires returning the person". In the case of truly Zero-Shot learning (without any human-engineered ICEs), the IoU score of ViperGPT's official GitHub code drops by 14 points to 46.7. Moreover, in their GitHub code we found hand-engineered improvements: if returning a bounding box fails, then return an "average" bounding box, and if code execution fails on GQA, then query the image captioner (BLIP-2 [Li et al., 2023]). These code changes lead to improved results, but make it hard to quantify the true power of LLMs as controllers approach.

In Table 6.1, we also provide Zero-Shot end-to-end baselines. On RefCOCO and RefCOCO+, we feed the query as input to the OWLv2 model and return its output. For the GQA end-to-end baseline, we query the PaLI-3 model (which was fine-tuned for multi-task inference on different captioning and VQA data sets, but not on GQA). For NExT-QA, we provide two baselines. For the first baseline, we subsample and caption with PaLI-3 one frame per second, and then feed all these captions to an LLM (`code-bison`) together with the question and multiple choice answers. As the second baseline, we simply feed the LLM the question and the possible answers and ask it to choose one. LLM-only baseline achieves an accuracy of 53.3%, which is only 5.4% lower than the baseline that also gets videos as input and significantly above the chance level accuracy of 20% (since there are 5 possible multichoice answers to each question). This suggests that NExT-QA might not fully evaluate the vision properties of the model and that new datasets are needed; for example, the Perception Test [Pătrăucean et al., 2023] was created specifically to avoid such problems.

Lastly, in Table 6.1 we report the Zero-Shot performance in our setup, as well as results for our best performing model variant (averaged over three random seeds). In the following sections, we evaluate each component, as well as their combinations, to obtain the reported results. Using `GPT-3.5-turbo` instead of `code-bison` resulted in a slight drop in performance, but as we shall see below, the same conclusions hold for both `code-bison` and `GPT-3.5-turbo` for all our suggested improvements. In the following, we present the components of our

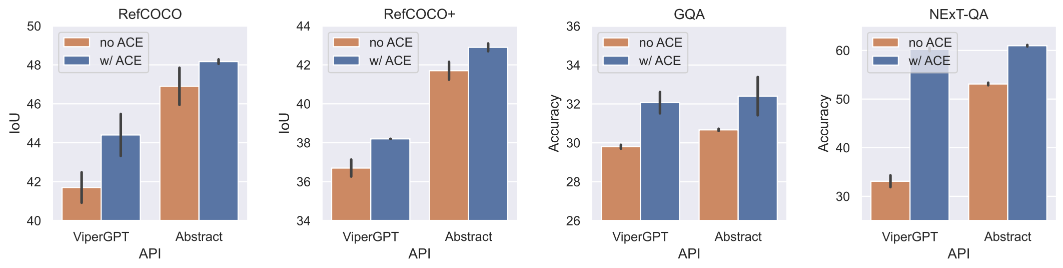


Figure 6.2: Using our Abstract API improves performance over the ViperGPT API across all datasets. Similarly, ACEs consistently improve performance, and these gains compound with the gains from the Abstract API. Uncertainty bars represent standard deviations computed over three random seeds.

framework. For an ablation study, see Table 6.2 that shows that each component contributes positively to the final scores on each dataset.

6.2.1 Zero-Shot through spatially and temporally Abstract API

In Figure 6.2, we show the effect of using our Abstract API instead of the API used in ViperGPT. The API for RefCOCO, RefCOCO+, and GQA uses the same set of image routines (see Appendix D.4), whereas the API for NExT-QA uses only video-specific (temporal) routines. For now, we focus on the brown bars in Figure 6.2, and we compare the ViperGPT API and our Abstract API. We can see that our Abstract API leads to gains both with and without ACEs across all datasets. The performance gain is most notable for the NExT-QA dataset when ACEs are not used. We suspect that this is due to the LLM's difficulty in reasoning about the temporal order of events. This confirms our hypothesis that building a more abstract API such that the LLM does not need to use low-level Python routines is a promising direction.

Finally, we investigate whether our conclusions also hold for other LLMs, namely OpenAI's `GPT-3.5-turbo`. On RefCOCO `GPT-3.5-turbo` achieves an IoU of 28.9 with the ViperGPT API and 39.8 with our Abstract API and an accuracy of 9.4 and 42.9 for the ViperGPT API and our Abstract API, respectively, on NExT-QA. This confirms that our Abstract API brings gains not only for `code-bison`, but also for other LLMs. For each sample in the evaluation, we allow only one trial of code generation (no self-correction). Compared to the results with `code-bison`, IoU of 37.7 and 40.5 on RefCOCO and accuracy of 11.5 and 46.1 on NExT-QA for the ViperGPT API and our Abstract API, respectively, the results with `GPT-3.5-turbo` are slightly worse. We suspect that the reason

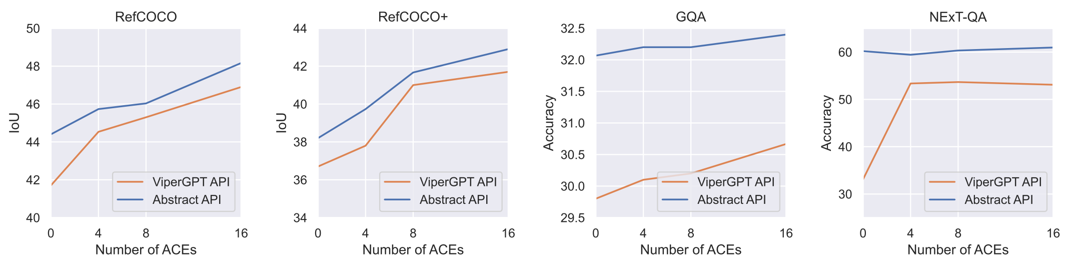


Figure 6.3: Increasing the number of ACEs in the prompt improves performance. Note that using the ViperGPT API on NExT-QA results in only three correct ACEs, so the performance plateaus after four ACEs.

for this could be that `GPT-3.5-turbo` is mainly built to be a conversational agent, while `code-bison` is specifically trained to have good coding capabilities.

6.2.2 Few-shot boostrapping via automatically generated in-context examples (ACEs)

We evaluate the effect of using automatically generated in-context examples (ACEs), described in Section 6.1.3. We can either sample few-shot examples manually or pick them at random. Both of these variants lead to good results, as we show in the following experiments. However, selecting examples manually allows for a better "quality" (in terms of diversity) given a small number of few-shot examples, so by default we use these for all experiments. For the first set of experiments, we manually pick 16 few-shot examples from the training set: image/video, question, and ground-truth answer. We try to make examples diverse to cover question "types" (e.g. left-right, front-back, closest-to, etc.).

In Figure 6.2, we show the effect of ACEs. For each dataset and for each API, the performance without ACEs is shown with the brown bar, and the performance with ACEs corresponds to the blue bar. We can see that for all datasets and all APIs, ACEs improve performance. The largest gains when using ACEs are for RefCOCO, RefCOCO+, and NExT-QA datasets when using the ViperGPT API. This indicates that ACEs are effective in dealing with complex spatial and temporal reasoning. More importantly, it can be seen in Figure 6.2 that the gains from both the Abstract API and ACEs compound for all tasks, indicating that they provide complementary strengths. Figure 6.3 shows how the performance in terms of IoU and accuracy scales with the number of few-shot examples used to generate ACEs. As expected, increasing the number of few-shot examples leads to improved performance. Note that the ViperGPT API on NExT-QA is able to correctly "solve" only 3 few-shot examples, so there are no gains beyond using 4 few-shot examples.

We evaluate the effect of using randomly sampled few-shot examples instead of manually selecting them. On RefCOCO, we sample 100 few-shot random samples from the training set, run Zero-Shot framework on them, sort the resulting programs by their IoU, and select the top 16 programs. Therefore, we end up with the same number of ACEs as with manual selection. On RefCOCO, we achieve IoU of 47.9 and 49.1 with the ViperGPT API and our Abstract API respectively. These results are slightly better than those with manually selected few-shot examples (46.9 and 48.2 IoU). This shows that the manual labor for generating ACEs can be removed altogether if we already have some labeled examples. With 50 few-shot random samples we obtain similar performance, and for 16 such samples we observe a small drop in performance (see Tables D.3 and D.4 in the Appendix D.1 for detailed results).

As in the previous section, we test whether our findings are consistent with `GPT-3.5-turbo` on RefCOCO and NExT-QA. On RefCOCO, when using `GPT-3.5-turbo`, ACEs improve IoU from 28.9 to 39.8 with the ViperGPT API and from 38.1 to 41.6 with our Abstract API. Similarly, for `GPT-3.5-turbo` on NExT-QA, ACEs improve accuracy from 9.4 to 42.9 the ViperGPT API and from 56.7 to 58.8 with our Abstract API. This confirms that the benefit of ACEs is not only limited to `code-bison` but also holds for `GPT-3.5-turbo` as well.

Another benefit of the few-shot setup when generating ACE is that it allows us to "tune" hyperparameters (HPs). For example, when sweeping over LLM temperature and object detection threshold HPs, we observed that the relative performances on the few-shot examples closely resemble the one when sweeping over the full validation dataset (the analysis is shown in Appendix D.1).

6.2.3 Self-correction

In this section, we analyze the ability of the framework to "self-correct" itself *without any external feedback* when code execution fails.

Self-debugging. When the program execution fails (due to e.g. compilation errors), we can retry by generating a new program [Chen et al., 2021]. When creating a new program, we can also feed the previously generated code and the question as part of the prompt (see Appendix D.4), a variant that we call "self-debugging". Another option would be to simply repeat the exact same prompt as in the previous trial and rely on stochasticity in the LLM with a tempera-

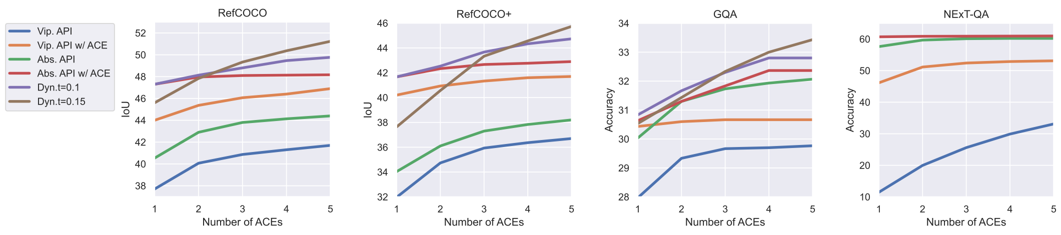


Figure 6.4: Increasing the number of "self-tuning" steps leads to improved performance. Our Abstract API (Abs. API) consistently outperforms the ViperGPT API (Vip. API). The best performance is achieved when using dynamic object detector threshold (Dyn.t) in addition to the Abstract API with ACE.

Model	RefCOCO (IoU)	RefCOCO+ (IoU)	GQA (acc.)	NExT-QA (acc.)
ViperGPT API	38.4	32.0	27.9	11.5
+ Abstract API	42.3 (+3.9)	34.0 (+2.0)	30.0 (+2.1)	57.6 (+46.1)
+ ACE	47.3 (+5.0)	41.7 (+7.7)	30.6 (+0.6)	60.7 (+3.1)
+ Self-debugging	48.2 (+0.9)	42.9 (+1.2)	32.4 (+1.8)	**61.0** (+0.3)
+ Self-tuning	**51.2** (+3.0)	**45.7** (+2.8)	**33.4** (+1.0)	-

Table 6.2: Component-wise ablations of our framework. Each component contributes positively to the final score. Their relative contributions vary for different tasks. We report mean scores across three random seeds.

ture greater than zero to generate a new correct solution. In our experiments, the "self-debugging" variant did not lead to an improvement in performance. In all cases, the performance plateaus after the first trial. This is in line with other recent findings [Huang et al., 2023; Stechly et al., 2023; Valmeekam et al., 2023]. On the other hand, the variant without any feedback in the prompt led to an increasingly better performance as the number of "trials" increases (see Figure 6.4).

Self-tuning. In some cases, we know that code fails due to some specific module. Therefore, we can then adjust the hyperparameters of the respective module and re-run code. For example, the open vocabulary detector we used (OWLv2) has a threshold hyperparameter that controls how sensitive it is. The lower this threshold, the more false positives we will have, but also the fewer false negatives. There is no global "optimal" value for this threshold that performs best for all images. Our framework allows us to adjust this hyperparameter dynamically: if the open vocabulary detector fails, we can lower the threshold and run the

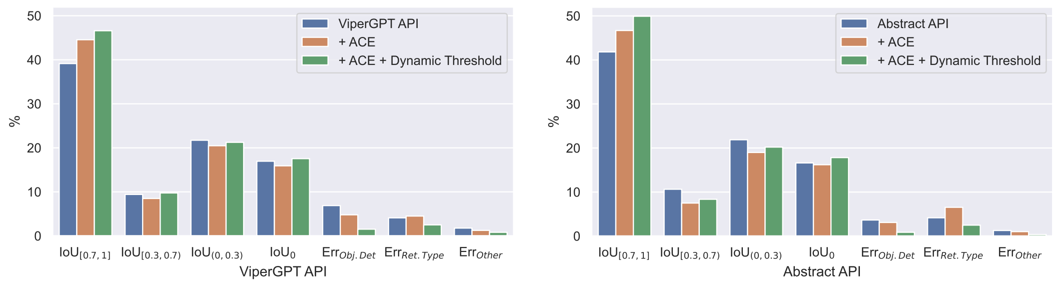


Figure 6.5: Error diagrams for the ViperGPT API and our Abstract API. We visualize the percentages of samples with IoU in certain ranges. "Err" classes are samples for which code execution failed due to either: object detection (Obj.Det), wrong return type (Ret.Type) or some other error (Other) e.g. hallucination.

visual program again. In Figure 6.4, we can see that variants with a dynamic object detection threshold outperform all other variants and achieve the best performance. Note that the variant that achieves the highest performance after five trials has a lower performance for the first trial. This happens because we start with a higher object detection threshold value of 0.15 (by default, we use 0.1). In this case, initially there will be more false negatives, but also fewer false positives. As we decrease the threshold in subsequent trials, the previously false negatives are detected and the queries are correctly answered.

6.2.4 Error analysis

Another benefit of visual reasoning with LLMs as programmers is interpretability. For example, we can get insights into the percentage of successful program executions, which can be further decomposed into the ones that resulted in correct or incorrect responses, and for the programs that failed to execute, we can provide further insights into why they failed, i.e. which module failed to return the correct result. Figure 6.5 shows one such error analysis on RefCOCO. Categories labeled with "Error" are the ones for which code failed to execute due to either object detection (Obj.Det), wrong return type (Ret.Type) or some other error (Other) e.g. hallucination. For all other cases, code executed correctly (it returned a bounding box), but sometimes it failed to detect the object ("IoU $= 0$" case). First, we notice that for both APIs the number of "correct" detections (IoU higher than 0.7) grows as we include ACEs and "self-tuning" through the dynamic object detection threshold. We can also see that the percentages of samples with high IoU are always higher for our Abstract API compared to the ViperGPT API. Finally, note that the percentage of error cases drops from 12.8%

to 4.8% for the ViperGPT API and from 9.1% to 3.8% for our Abstract API.

6.3 Discussion and future work

Although the LLMs as controllers framework is very promising for visual reasoning, there is much future work to be explored. First, the use of video-specific models (or tools) could greatly improve performance on video tasks compared to the image-specific models we used. Moreover, the code generating LLM currently only takes the question as the input, but for some questions the program that correctly solves the question can only be generated given the image or video as the input too.

The results with the Abstract API show that this is a promising path forward, but more research is needed to find the "optimal" set of visual and temporal routines. Starting from these primitive routines, the model should be able to build an ever-growing library of routines (e.g. through the ACE generation process) that it can later reuse. This growing library of routines will most likely grow larger than the size of the context window, so research is needed on an API "router" that can select routines that are relevant to a specific task at hand. Furthermore, it would be important to research ways of eliminating the need for few-shot examples when generating ACEs, e.g. by providing a natural language dataset specification (a datasheet).

Lastly, more effort should be put into creating better benchmarks for evaluating compositional visual reasoning, as current ones have a number of limitations. For example, not all samples in RefCOCO and RefCOCO+ require compositional reasoning, so the LLM should only query the open-vocabulary object detector. Similarly, many referring expressions in GQA are ambiguous in the sense that there is not a single unique answer. Finally, NExT-QA contains ambiguous questions (e.g. why someone did certain actions) or questions that can be answered by looking at the multiple choice answers only and disregarding the visual input altogether. The Perception Test [Pătrăucean et al., 2023] is a promising benchmark for future work, as it was specifically created to avoid such problems. We hope that our findings inform future research on LLMs as controllers for visual reasoning and encourage systematic evaluation and benchmarking efforts in the future.

6.4 Conclusion

In this work, we present a framework that makes LLMs as programmers for visual reasoning more robust, removes the need for human engineering of in-context examples (ICEs), and thus brings them a step closer to *truly* zero-shot visual reasoners. We introduce an "Abstract API" that consists of spatially and temporally abstract routines, which improves performance by reducing the burden on the code-generating LLM to have strong spatial and temporal reasoning. By using a few labeled examples, we show how one can generate query-code ICEs automatically (ACEs) in a zero-shot manner. When used as in-context examples, ACEs consistently improve performance, eliminating the need for human engineering of ICEs. We demonstrate how LLMs as controllers for visual reasoning can (to a certain extent) perform "self-correction" through "self-debugging" and "self-tuning" without any ground-truth labels. In self-debugging, generating new code from scratch led to consistently better results, but providing the previous query-code pair as feedback to the LLM did not improve performance. In self-tuning, we show that hyperparameters of certain modules can be tuned automatically if code execution fails due to these modules. Across a number of compositional question-answering and video temporal reasoning tasks, we demonstrate that each component of our framework consistently leads to improvement.

Chapter 7

Conclusion and Future Work

This dissertation provides several contributions towards improving compositional visual reasoning and generalization using neural networks. In our work, we hypothesized that the current NNs suffer from the binding problem, and offered three ways of going forward: learning an object- and relation-centric world model, scaling NNs, and employing task decomposition through modular NNs, e.g. by using a large language model (LLM) as a controller.

First, we argued for the importance of learning object-centric representations from raw visual data. In this context, we introduced a method that improves the reasoning and generalization abilities of a sequential slots model (Chapter 2).

We then argued for the importance of learning hierarchical object representations. In Chapter 3 we introduced a method that is capable of inferring nodes and edges of a hierarchical graph directly from raw visual data. We showed that our approach to physical reasoning models objects as hierarchies of parts that may locally behave separately but also act more globally as a single whole, which greatly improves physics reasoning capabilities on videos.

In Chapter 4 we argued that (hierarchical) decomposition into objects is generally task-dependent and that it is sometimes infeasible and undesirable to decompose a scene into all hierarchy levels. For these reasons, it might be more beneficial to modulate objects with task information, such as words, actions, or goals. In an RL setting, we first introduced a set of new environments that evaluate out-of-distribution generalization, and showed that agents based on NNs that learn distributed representations fail to generalize. We then introduced object-centric agents and showed that they perform favorably and generalize better to OOD environments.

Since slot-based methods have certain conceptual limitations, we then (Chapter 5) studied an alternative way to unsupervised object discovery. In this context, we introduced a synchrony-based method that can, for the first time, discover objects in an unsupervised manner in multi-object color datasets and simultaneously represent more than three objects.

Finally, in Chapter 6 we studied an alternative method to unsupervised object-centric representation learning, namely one based on "tool use". There, we improved a framework for compositional visual reasoning with LLMs as controllers by introducing spatially and temporally abstract routines and by leveraging a small number of labeled examples to automatically generate in-context examples, thereby avoiding human-created in-context examples.

Limitations In each chapter of this thesis, we discuss limitations of our methods and here we provide their summary.

In Chapter 2 we introduced R-SQAIR, a method that improves the reasoning and generalization abilities of a sequential slots model. However, in its current form, this method is limited to grayscale data, due to using SQAIR to infer the objects from the images. A second limitation is that this method computes all pairwise relations for every frame, which is often wasteful in the real-world where the interactions occur infrequently.

In Chapter 3 we introduced HRI, a method that learns part-object hierarchical structure and their relations purely from raw visual data. A limitation of this method is that it uses spatial slots to learn about parts and objects, which could be addressed by using parallel or sequential slots. Another limitation is that it relies on operations of the CNN to group parts into objects. This could be addressed by using a more flexible architecture with weaker inductive bias, such as Transformers. Furthermore, we evaluated HRI only on a limited number of real world datasets, so the question remains how it scales to other scenarios. Finally, HRI assumes that we can extract a fixed, static hierarchical representation of the scene, whereas this is often infeasible and undesirable in the real world, where the hierarchy level over which we operate should be task dependent.

In Chapter 4 we introduced new object-centric agents and showed that they have better out-of-distribution generalization properties. A limitation of our approach is that it does not modulate object representations with neither actions nor goals. Furthermore, the environment we used for benchmarking has fairly low visual complexity and objects are well-defined. This raises questions about the scalability of our method, mainly due to the scalability issues of object-centric methods, which are not specific to the RL setup.

In Chapter 5 we showed how synchrony-based models can be scaled to color images by using a contrastive objective. However, our method still only scales to color scenes with low visual complexity. In particular, we found it to sometimes struggle with objects that have specular highlights and similarly colored objects. Another limitation of our work is that the contrastive objective can be computationally demanding, depending on the number of patches (resolution) over which we contrast. Finally, inherits the general drawback of synchrony-based methods, which is computationally more expensive inference, due to clustering features to determine feature groupings into objects.

In Chapter 6 we introduced a framework for compositional visual reasoning with LLMs as controllers that leverages abstract routines and a small number of labeled examples to make LLMs-as-controllers setup more robust and reduce the dependence on human engineering. Perhaps the biggest limitation of our approach is that the code-generating LLM takes as input only the question and a Python API, but not the visual input. However, some questions can only be answered if we consider visual input as the context. For example, query "second from the right" for an image of people would require us to first detect persons, whereas if we are given an image of dogs, we would ask the object detector to detect the dogs in the image. Finally, in our approach, we do not create a library of already generated functions that can be reused later.

Outlook and future work Throughout this thesis, we hinted at many future research directions, but here we provide three high-level research directions: learning task-modulated object representations, scaling synchrony-based methods, and further improving the LLMs-as-controllers framework.

First, we believe that the most promising way to scale object-centric representations to real-world datasets is to study the weakly supervised setup where we are given a bag of words that represent objects or concepts in the scene, and learn to ground these in pixels in an unsupervised manner.

Second, the synchrony-based method we introduced in this thesis is only capable of handling visually simple scenes and still lags behind slot-based methods. More research is needed to close this gap. One promising direction could be investigating ways of making not only activations, but also weights to be complex-valued.

Third, in the direction of LLMs as controllers for visual reasoning in video tasks, it would be beneficial to use video-specific models. Furthermore, for some questions, the "correct" visual program can only be generated if not only the question is fed into the LLM but also the visual input. Future work should investigate this by turning the code-generating LLM into a VLM. Lastly, new ways should be investigated to make the framework truly self-referential, both in correcting its errors in generated code and in starting from the given primitive visual routines and then building an ever-increasing “library” of visual routines.

Finally, it is an open research question whether gains would be obtained by combining some of the three directions we proposed in this thesis: object-centric inductive biases, scaling NNs, and reasoning with LLMs-as-controllers. For example, in Chapter 6 we used exclusively pre-trained object detectors, but it would be interesting to see how one could incorporate models that are trained to discover objects in a completely unsupervised manner. In a way, only a part of the binding problem in end-to-end models (e.g. compositional reasoning) is circumvented by using LLMs-as-controllers, but the low-level question on how to infer discrete representations from raw visual input still remains. Similarly, the question remains on whether object-centric NNs can be scaled and what their scaling laws are. Although there are some initial results on scaling object-centric NNs [Elsayed et al., 2022], increasing their capacity is often in conflict with the “bottleneck” principle based on which they are encouraged to learn object representations. Therefore, the best way on how to make this trade-off remains an open research question.

Publications during the PhD program

- **Aleksandar Stanić** and Jürgen Schmidhuber. R-SQAIR: Relational Sequential Attend, Infer, Repeat. NeurIPS PGR workshop, 2019. [Stanić and Schmidhuber, 2019]

- **Aleksandar Stanić**, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Hierarchical Relational Inference. Proceedings of the AAAI Conference on Artificial Intelligence, 2021. [Stanić et al., 2021]

- Djordje Miladinovic, **Aleksandar Stanić**, Stefan Bauer, Jürgen Schmidhuber and Joachim Buhmann. Spatial Dependency Networks: Neural Layers for Improved Generative Image Modeling. International Conference on Learning Representations (ICLR) 2021. [Miladinović et al., 2021]

- **Aleksandar Stanić**, Yujin Tang, David Ha, and Jürgen Schmidhuber. Learning to generalize with object-centric agents in the open world survival game crafter. IEEE Transactions on Games, pages 1–20, 2023. doi: 10.1109/TG.2023.3276849. [Stanić et al., 2023]

- **Aleksandar Stanić**, Anand Gopalakrishnan, Kazuki Irie, and Jürgen Schmidhuber. Contrastive Training of Complex-valued Autoencoders for Object Discovery. Neural Information Processing Systems (NeurIPS), 2023. [Stanić et al., 2023b]

- **Aleksandar Stanić**, Sergi Caelles, and Michael Tschannen. Towards Truly Zero-shot Compositional Visual Reasoning with LLMs as Programmers. *Under submission*. arXiv:2401.01974, 2024. [Stanić et al., 2024]

- **Aleksandar Stanić**, Dylan Ashley, Oleg Serikov, Louis Kirsch, Francesco Faccio, Jürgen Schmidhuber, Thomas Hofmann, Imanol Schlag. The Languini Kitchen: Enabling Language Modelling Research at Different Scales of Compute. Under submission. arXiv:2309.11197, 2023. [Stanić et al., 2023a]

- Mingchen Zhuge, Haozhe Liu, Francesco Faccio, Dylan R. Ashley, Robert Csordas, Anand Gopalakrishnan, Abdullah Hamdi, Hasan Abed Al Kader Hammoud, Vincent Herrmann, Kazuki Irie, Louis Kirsch, Bing Li, Guohao Li, Shuming Liu, Jinjie Mai, Piotr Piękos, Aditya Ramesh, Imanol Schlag, Weimin Shi, **Aleksandar Stanić**, Wenyi Wang, Yuhui Wang, Mengmeng Xu, Deng-Ping Fan, Bernard Ghanem, Jürgen Schmidhuber. Mindstorms in Natural Language-Based Societies of Mind. R0-FoMo: Workshop on Rubstness of Few-shot and Zero-shot Learning in Foundation Models, NeurIPS 2023. arXiv:2305.17066, 2023. [Zhuge et al., 2023]

Appendix A

Additional Details for Learning Hierarchical Object Representations

In this section we decribe all datasets and the training procedure in detail. Additionally, we provide model architecture configurations for the HRI model, ablation models, NRI and LSTM baselines. Reported results in the bar plots are the mean and standard deviation obtained for each model using 5 different random seeds.

A.1 Datasets

Four main datasets are used: *state-springs*, *visual-springs* (rendered version of state-springs), and two real-world benchmark datasets: one obtained by rendering the joints of the *Human3.6M* dataset and the other using the raw videos of the *KTH* dataset.

A.1.1 Springs

We simulate a system of moving objects connected via finite-length springs starting from the open source simulator implementation of Kipf et al. [2018]. We make the following modifications: objects are connected via finite length springs (instead of ideal springs) and the sampled graphs have a hierarchical structure in terms of connectivity, initial spatial positions and the spring constants (which reflects in the speed by which objects in different layers of the hierarchy move). Objects are connected via finite springs, which makes them act according to a modified Hooke's law:

$$F_{ij} = -k_F(r_i - r_j - l \cdot \frac{r_i - r_j}{|r_i - r_j|}), \tag{A.1}$$

where r_i and r_j are (x, y)-coordinates of objects i and j, k_F is the spring constant and l is its length. The objects are connected in a hierarchical graph, and they move inside a unit box (bounce elastically from the walls). To simulate the system, we initialize the root node position randomly in a neighborhood around the center of the image by sampling its (x, y) coordinates from $\mathcal{N}(0, 0.25)$. We then initialize the intermediate and the leaf nodes randomly inside each of the four image quadrants to ensure an initial bias towards spatial grouping. In particular, we sample from Gaussian distributions with variance 0.25 and the means (for (x, y)-coordinates) being the centers of the four quadrants: $(-0.25, 0.25)$, $(0.25, 0.25)$, $(-0.25, -0.25)$ and $(0.25, -0.25)$. For each sample in the dataset, we sample a graph with *random* connectivity: we start from a full tree graph, where sibling nodes are fully connected, then drop edges at random with a probability of 0.5, but ensure that the resulting graph is connected. The spring lengths are: 0.4 between the root and intermediate nodes, 0.1 between intermediate and leaf nodes, 0.65 within intermediate node siblings and 0.2 within leaf node siblings. All springs have the same constant, except for springs between leaf node siblings, which have a value that is half the value of other constants. In total we generate a dataset of $5 \cdot 10^5$ training, 10^5 validation, and 10^5 test sequences, each 50 frames (steps) long.

We note that all objects in the springs dataset are free to move anywhere in the space, and not limited to a particular quadrant. The intermediate nodes are only initialized with a bias towards quadrants, which was done to obtain a scenario where the objects naturally group together from the beginning of the sequence. This is identical to initializing all positions at random, and then performing rollouts until the objects' positions converge (which would always happen due to the hierarchical forces that act between them).

A.1.2 Human3.6M

This dataset Ionescu et al. [2013] consists of 3.6 million 3D human poses, corresponding 2D pose projections and images. We use the provided 2D pose projections to render 12 joints in total (3 of each limb). In total there are 11 professional actors in 17 scenarios, from which subjects number 1, 5, 6, 7, and 8 are used for training, and subjects number 9 and 11 for testing. This allows us to create 10k training and 3.5k test sequences of 50 frames.

A.1.3 KTH

The KTH Action dataset Schuldt et al. [2004] consists of real-world videos of people performing one of six actions: walking, jogging, running, boxing, hand-waving and hand-clapping. We trained all models on 64×64 video sequences, by conditioning on 10 frames and training the model to predict the next 10 frames in ten frames in the sequence, as done in previous work Denton and Fergus [2018]. We split the dataset into a training dataset containing 8k sequences and test dataset containing 1.5k test sequences, consisting of subjects not present in the training dataset.

A.2 Training Details

All models are trained with Adam Kingma and Jimmy Ba [2015] using default parameters and a learning rate of $5 \cdot 10^{-4}$. We use a batch size of 32 and train models for 100 epochs. On the visual task we train each model in two stages, which acts as a curriculum: for the first 50 epochs we train the visual encoder and decoder on a reconstruction task, afterwards we optimize the relational module and dynamics parameters on the prediction task.

Optimizing only for the next step prediction task can lead to a predictor which ignores the inferred relational graph (also noted in Kipf et al. [2018]). To avoid this, in the feedforward case we predict 10 steps in the future by feeding in predicted output as the next step input, and only feed the ground truth input every 10 steps. In the recurrent case this might lead to instabilities, so we introduce a "burn-in-phase", where we feed the ground truth input at the beginning, and predict the last 10 steps of the sequence. To train the models we optimize for ELBO over such sequences, whereas we evaluate the models on the next step prediction task. The Gaussian output distribution has a fixed variance $\sigma^2 = 5 \cdot 10^{-5}$.

We evaluate the models based on the negative log likelihood loss and the "normalized" negative log likelihood (Figs. 3.3a and A.2), which is inversely proportional to a version of HRI that operates on the ground-truth interaction graph (HRI-GT) (higher value is better). The latter allows us to factor out the complexity of the task and make it easier to compare results *between* tasks. We average the negative log likelihood either over the number of objects (for the state datasets) or pixels (for the visual datasets).

A.3 HRI Architecture Details

We describe every module in terms of blocks it consists of, e.g. node-to-edge MLP, edge-to-node MLP, LSTM etc. The architectures in all tables represent the forward pass, except for Table A.1, where for the first part (leaf object encoder) we have two variants (depending on whether we use SlotDec or ParDec), whereas the hierarchy inference module is the same in both cases.

A high-level overview of the HRI model is presented in Fig. 3.2, with a high-level summary of all components. Below we describe each component in detail.

Visual Encoder The visual encoder takes as input the concatenation of $32 \times 32 \times 3$ RGB frame and a $32 \times 32 \times 2$ (x, y)-fixed coordinate channels (as in Liu et al. [2018]; Watters et al. [2019b], simplifying the object's position inference relative to the image frame), processes it with several convolutional layers with ReLU non-linearities and batch norm Ioffe and Szegedy [2015] and outputs a hierarchy of object representations. First it infers the 4×4 leaf objects (note different variants for this part in Table A.1 depending on the decoder), from which 4 intermediate nodes and 1 root node are inferred. This results in 16 leaf objects, 4 intermediate objects, and one root object, each 48-dimensional. They are all mapped with a FC layer (with shared weights) to 16-dimensional vectors and fed through another (Object-wise VAE) FC layer (μ and σ for the standard VAE reparametrization). For KTH experiments the input resolution is $64 \times 64 \times 1$, handled by repeating the first convolutional layer of the encoder.

Relational Inference Module The relational inference module takes a sequence of $K = 10$ object vectors (16-dimensional in visual and 4-dimensional in state case) as input and outputs one-hot encoding (2-dimensional) of the pairwise relations $\mathbf{e}_{(i,j)}$ (the outputs of the f_o MLP inTable A.2). The samples are drawn as $\mathbf{r}_{ij} = \mathrm{softmax}((\mathbf{e}_{(i,j)} + \mathbf{g})/\tau)$ where $\mathbf{g}$ is drawn from a $\mathrm{Gumbel}(0, 1)$ distribution and $\tau = 0.5$ is the temperature parameter.

Dynamics Predictor The dynamics predictor (Table A.3) takes as input inferred object states of dimensionality d and the inferred pairwise edges, and predicts the object states at the next time step via rounds of message passing. Hierarchical message passing functions f^{bu}_{MP}, f^{ws}_{MP}, and f^{td}_{MP} perform a single node-to-edge and edge-to-node message passing operation as in equation 3.1, where their node-to-edge and edge-to-node MLPs all share the same set of weights. The feedforward variant replaces LSTM with one layer (FC with 64 ReLU units)

Leaf Object Encoder for SlotDec
8×8 conv, 48 ReLU units, stride 8, batch norm
Leaf Object Encoder for ParDec
8×8 conv, 48 ReLU units, max pool 2, batch norm
8×8 conv, 48 ReLU units, max pool 2, batch norm
8×8 conv, 48 ReLU units, max pool 2, batch norm
Hierarchical objects inference
2×2 conv, 48 ReLU units, max pool 2, batch norm
2×2 conv, 48 ReLU units, max pool 2, batch norm
Object-wise FC, 16 ReLU units.
Object-wise VAE FC, 2×16 units (μ, σ).

Table A.1: **Visual encoder architectures**

that takes as input the concatenation of object at the current step and the effect computed by the message passing.

Visual Decoder The visual decoder takes as input a set of N vectors ($d = 16$-dimensonal object states) and produces the output image according to the architecture in Table A.4. For the SlotDec a unique float index $i \in [0, 1]$ is appended to each object state, which helps learning the visual object colors, as they are decoded separately as a (permutation invariant) set and then summed. Note that for the KTH experiments the image resolution is $64 \times 64 \times 1$, which is handled by simply adding another '4×4 convTranspose, 64 ReLU, stride 2' layer to the decoder. In a similar way we add a convolutional layer to the encoder and by doing this we are able to infer the same hierarchy regardless of the input resolution. To account for additional visual complexity of predicting As in Denton and Fergus [2018], we add skip connections between encoder and decoder to enable the model to easily generate static background features, and allow the dynamics predictor to focus on modelling the changes.

A.4 Ablations

Following are the ablation-specific configurations:

- **HRI-GT**: HRI model that gets ground truth graph as the input to the dy-

Node-embeding MLP
Concatenate K object states in a slot-wise manner FC, 64 ELU FC, 64 ELU, batch norm
Concatenate object pairs slot-wise $o_{ij} = [o_i, o_j]$
Node-to-edge MLP f_{n2e}
FC, 64 ELU FC, 64 ELU, batch norm
Edge-to-node MLP f_{e2n}
FC, 64 ELU FC, 64 ELU, batch norm
Append slot-wise the skip connection of o_{ij}
Node-to-edge MLP (shared weights with f_{n2e}**)**
FC, 64 ELU FC, 64 ELU, batch norm
Output MLP f_o
FC, 64 ELU FC, 64 ELU, batch norm FC, 2 output units

Table A.2: **Relational inference module architectures**

namics predictor (no relational inference).

- **HRI-H**: HRI model that performs relational inference on a smaller subset of edges (other edges are excluded), by considering the convolutional and pooling operations that infer the hierarchical object slots. Let $o1$ be the root object, $o2, o3, o4, o5$ intermediate objects, and $o6 - o21$ leaf objects. The subset of edges HRI-H considers are parent-child (and vice-versa child-parent) $(1-2, 1-3, 1-4, 1-5)$, $(2-6, 2-7, 2-8, 2-9)$, $(3-10, 3-11, 3-12, 3-13)$, $(4-14, 4-15, 4-16, 4-17)$ and $(5-18, 5-19, 5-20, 5-21)$ and all within-sibling edges.

- **NRI-GT**: NRI model that gets ground truth graph as the input to the dynamics predictor (no relational inference).

Bottom-up message passing round f_{MP}^{bu} (on child-parent edges)
Node-to-edge MLP f_{n2e} FC, 64 ReLU FC, 64 ReLU, batch norm **Edge-to-node MLP** f_{e2n} FC, 64 ReLU FC, 64 ReLU, batch norm
Within-siblings message passing round f_{MP}^{ws} (on sibling edges) Shared weights of f_{n2e} and f_{e2n} MLPs with f_{MP}^{bu}
Top-down message passing round f_{MP}^{td} (on parent-child edges) Shared weights of f_{n2e} and f_{e2n} MLPs with f_{MP}^{bu}
LSTM
LSTM, 64 hidden units
Output MLP f_o
FC, 64 ReLU FC, d output units.

Table A.3: **Dynamics predictor architecture**

- **FCMP**: NRI model that performs message passing in the dynamics predictor on a fully connected graph (no relational inference).

Note also that the performance gain observed in our experiments is not a consequence of an increased model capacity: in the case of visual springs dataset HRI and the NRI baseline have the same number of parameters - 292′141, whereas the LSTM baseline has more - 317′707.

A.5 NRI Baseline

To infer the object states on which NRI performs relational inference we use the visual encoder and decoder of the HRI architecture. This ensures a fair comparison between NRI and HRI in the visual setting. We emphasise that

SlotDec
FC, $4 \times d$ ReLU
4×4 convTranspose, 64 ReLU, stride 2
4×4 convTranspose, 64 ReLU, stride 2
4×4 convTranspose, 64 ReLU, stride 2
4×4 convTranspose, 3 ReLU, stride 2
ParDec
4×4 convTranspose, 64 ReLU, stride 2
4×4 convTranspose, 64 ReLU, stride 2
4×4 convTranspose, 3 ReLU, stride 2

Table A.4: **Visual decoder architectures**

standard NRI as presented in Kipf et al. [2018] did not support learning from visual images.

The dynamics predictor ('decoder' in NRI Kipf et al. [2018]) is presented in Table A.5, which uses an LSTM Hochreiter and Schmidhuber [1997] instead of the GRU Cho et al. [2014] cell.

A.6 LSTM Baseline

Similarly to the NRI baseline, the LSTM baseline uses the same (pretrained) visual encoder and decoder to map from image to object states, and vice-versa. We use an LSTM with 64 hidden units that concatenates representations from all objects and predicts their future state jointly. Essentially, the NRI baseline dynamics predictor can be viewed as extending the LSTM by adding the message passing part (functions f_{n2e} and f_{e2n}) based on the inferred interaction graph. In contrast, the LSTM baseline only explicitly considers the nodes of the graph, but not its edges (relations).

A.7 Additional Results

Below we provide additional results for the 3-3 state and visual dataset, the results for datasets containing homogeneous (white) balls instead of colored ones: 3-3-state-visual-white and 4-3-state-visual-white in Appendix A.9, present the rollouts on visual spring datasets and Huma3.6M dataset in Appendix A.10,

Node-to-edge MLP f_{n2e}
FC, 64 ReLU FC, 64 ReLU, batch norm
Edge-to-node MLP f_{e2n}
FC, 64 ReLU FC, 64 ReLU, batch norm
LSTM
LSTM, 64 hidden units
Output MLP f_o
FC, 64 ReLU FC, d output units.

Table A.5: **NRI dynamics predictor**

results on datasets containing objects of different shapes, sizes and occlusions in Appendix A.11, and finally results on generalization (extrapolation) of models trained on 4-3-visual-springs to 3-3-visual springs in Appendix A.12.

A.8 State Springs

Additional results in terms of the negative log likelihood are provided for the 4-3-state-springs dataset in Fig. A.2. By having the absolute, instead of the normalized score, we can see how markedly big the gap between the recurrent and the feedforward dynamics predictor is. Additionally, the results in Figs. A.1 and A.3 for the 3-3-state-springs show that same conclusions hold as for the 4-3-state-springs, with the exception of Fig. A.1b where HRI and HRI-H perform the same (for 4-3-state-springs HRI outperforms HRI-H, see Fig. 3.3b).

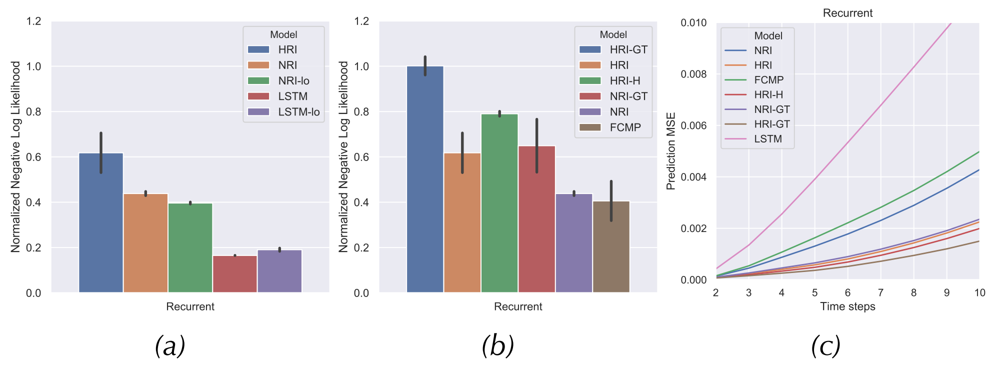


Figure A.1: Performance on the 3-3-state-springs dataset. We compare HRI to (a) baselines and (b) ablations in terms of the "normalized" negative log likelihood which is inversely proportional to HRI-GT, which receives the ground-truth graph. In this case higher is better. (c) MSE when predicting into the future (prediction rollouts).

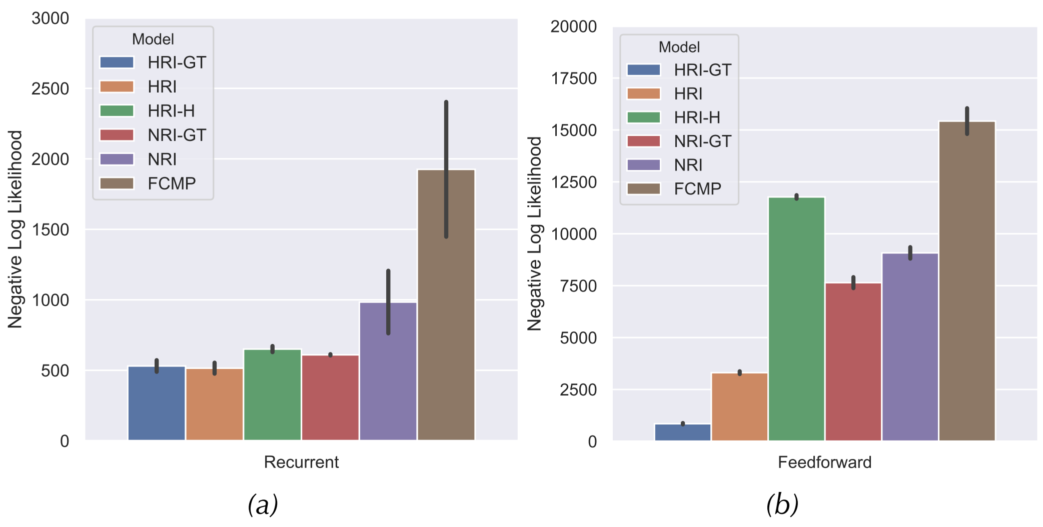
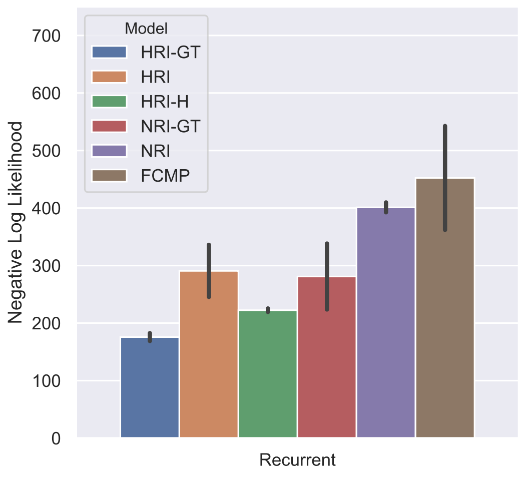
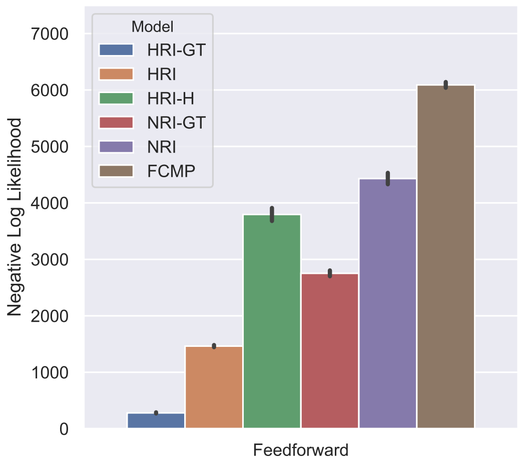


Figure A.2: Performance of HRI and ablation models on the 4-3-state-springs dataset in terms of the negative log likelihood in both recurrent and feedforward case. Note the difference in the ranges of y axis indicating the inability of the feedforward model to learn accurate dynamics prediction.



(a)

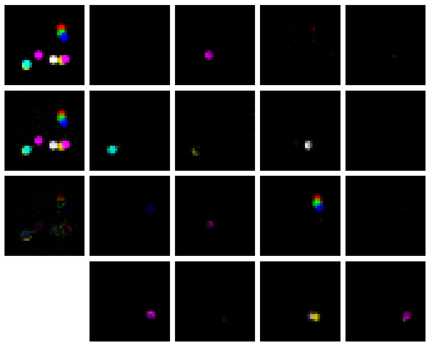


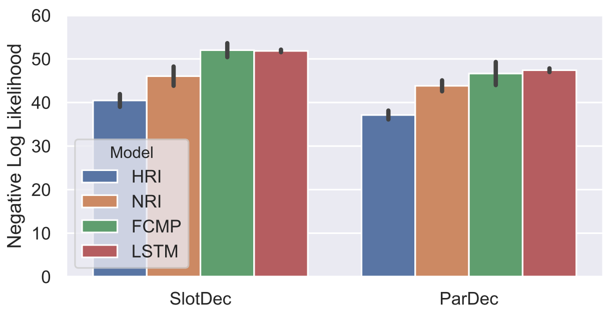
(b)

Figure A.3: Performance of HRI and ablation models on the 3-3-state-springs dataset in terms of the negative log likelihood in both recurrent and feedforward case. Note the difference in the ranges of y axis indicating the inability of the feedforward model to learn accurate dynamics prediction.

A.9 Visual Springs - Different Number of Objects and Homogeneous Colors

We provide additional results and visualizations for the 3-3-visual-springs and additional two experiments that we performed with homogeneous (white) balls: 3-3-visual-springs-white and 4-3-visual-springs-white. The result in Fig. A.4 confirms that our model is able to handle different number of balls, by learning to simply leave some slots empty (Fig. A.4a). Moreover, the results in Figs. A.5 and A.6 closely match the ones in Figs. 3.4 and A.4, thus showing that our method does not rely on color to disentangle objects into separate slots.


(a)

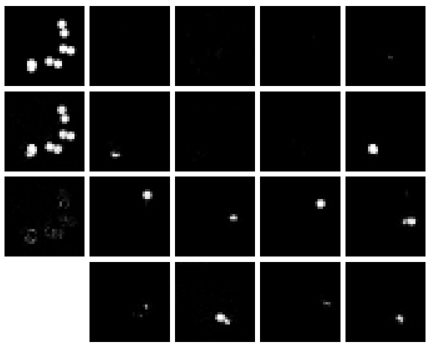


(b)

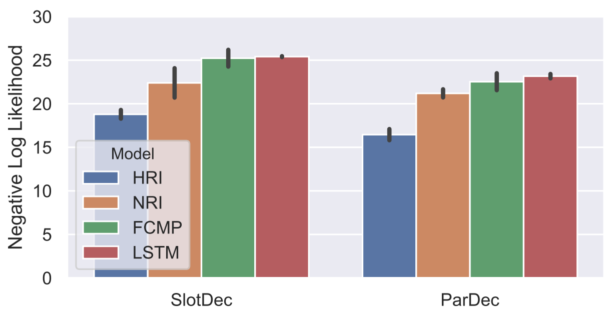
Figure A.4: (a) HRI introspection: first column contains ground-truth, predicted image and their difference. In the other 4 columns we visualize 16 object slots decoded separately. (b) Negative log likelihood for all models on the 3-3-visual-springs dataset.

A.10 Prediction Rollouts and Latent Interaction Graph Inference

We provide additional results for the 4-3-visual-springs and Human3.6M dataset. In particular, Fig. A.7 shows 10 time step prediction rollout of HRI model on 4-3-visual springs dataset, where it can be seen that the predictions closely match the ground truth sequence. The rollout sequence in Fig. A.8 shows similar performance on the Human3.6M dataset. Inferred interaction graphs on 4-3-visual springs by HRI (top row) and NRI (bottom row) are shown in Fig. A.9. The first column shows the groundtruth graphs, and the graphs to follow in the next

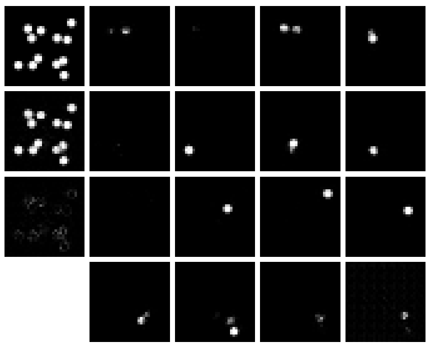


(a)

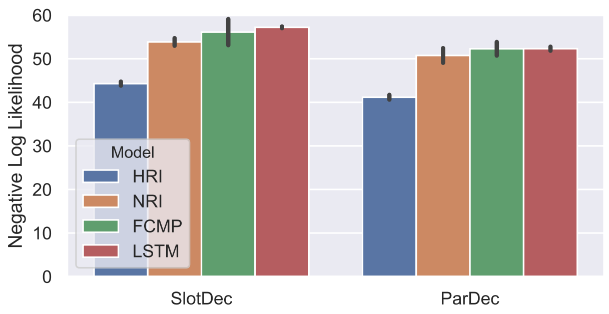


(b)

Figure A.5: (a) HRI introspection: first column contains ground-truth, predicted image and their difference. In the other 4 columns we visualize 16 object slots decoded separately. (b) Negative log likelihood for all models on 3-3-visual-springs-white.



(a)



(b)

Figure A.6: (a) HRI introspection: first column contains ground-truth, predicted image and their difference. In the other 4 columns we visualize 16 object slots decoded separately. (b) Negative log likelihood for all models on 4-3-visual-springs-white.

columns correspond to graphs inferred by models in different sequence steps. The graph inferred by HRI resembles the ground-truth much more closely compared to NRI.

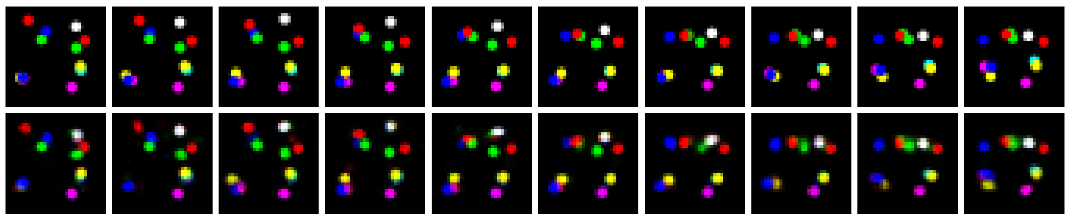


Figure A.7: Ground truth (top) and predicted (bottom) 10 time steps rollout of HRI on 4-3-visual-springs.

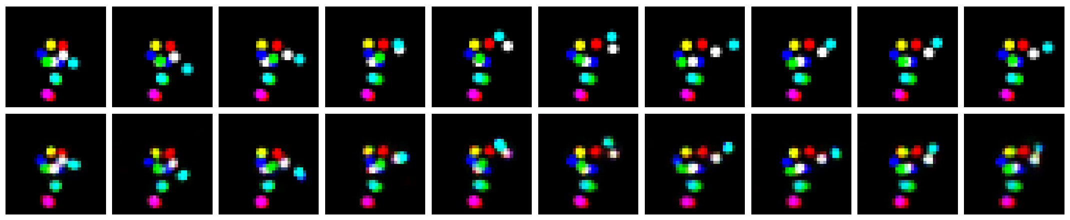


Figure A.8: Ground truth (top) and predicted (bottom) 10 time steps rollout of HRI on Human3.6M.

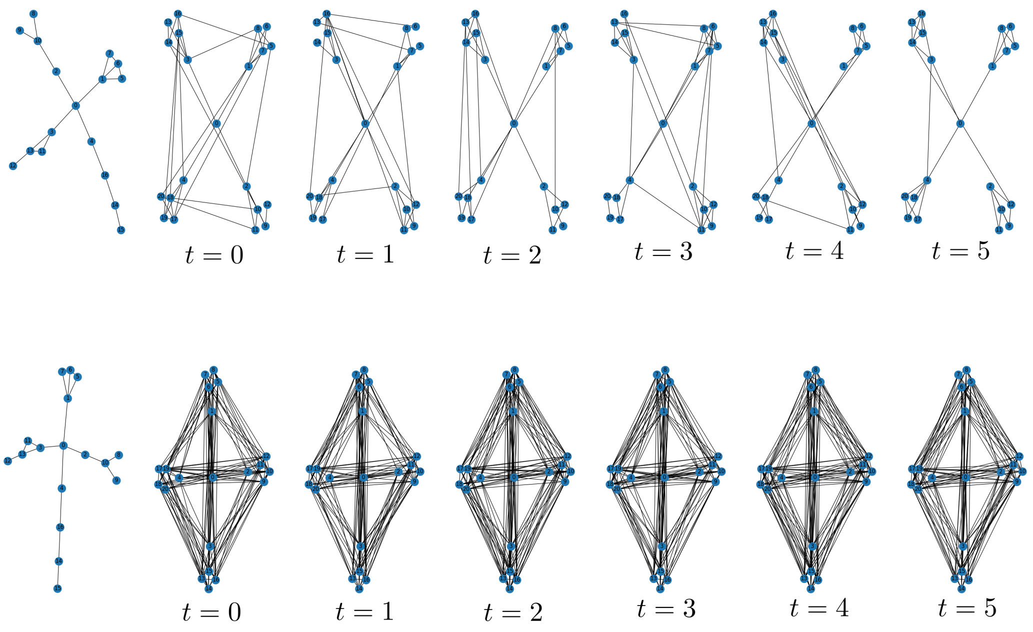


Figure A.9: Inferred interaction graphs on 4-3-visual-springs by HRI (top row) and NRI (bottom row). In the first column are groundtruth graphs, and the graphs to follow in subsequent columns correspond to graphs inferred by the respective model over the video sequence frames.

A.11 Visual Springs - Diverse Datasets

In this section we provide results on 5 additional datasets: *Triangles*, *Squares*, *Diverse-Objects* (contains balls, triangles and squares of various sizes), *Large-Objects*, and *Curtain*. Samples of a frame from each of these datasets are shown in Fig. A.10. Qualitative introspection and quantitative performance in Figs. A.11 to A.15 show that HRI is able to handle datasets containing different object shapes, sizes, and also handle occlusions, while outperforming all considered baselines in all cases. *Curtain* dataset is created form 4-3-visual-springs dataset by inserting a 12×12 pixels curtain at random position in each video sequence.

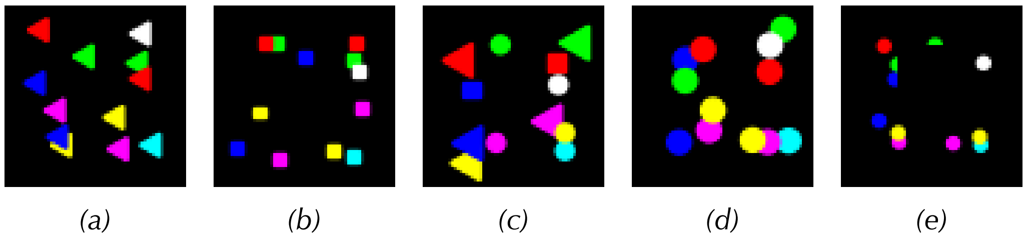


Figure A.10: Samples from: (a) *Triangles*, (b) *Squares*, (c) *Diverse-Objects*, (d) *Large-Objects*, and (e) *Curtain* datasets.

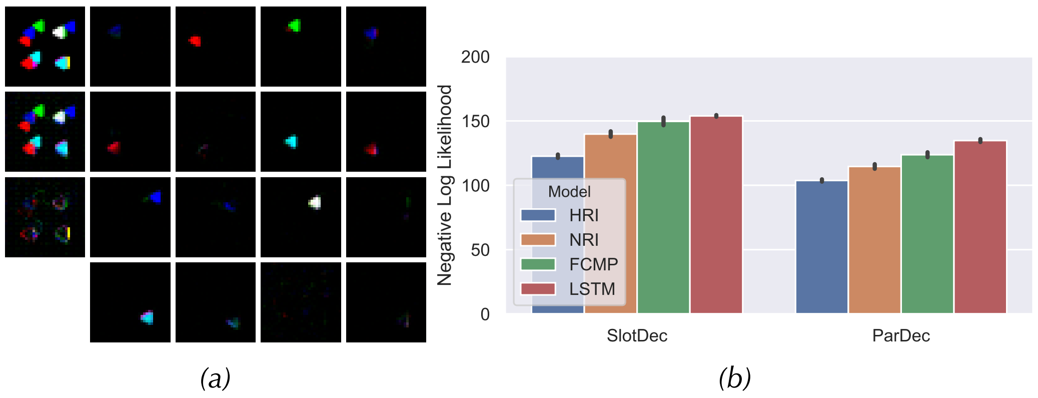
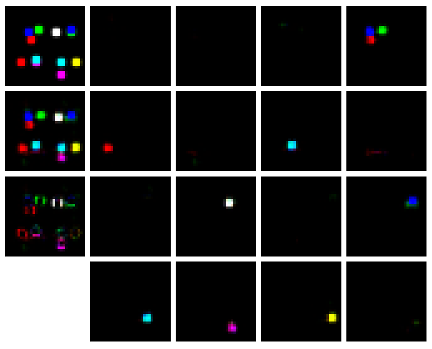
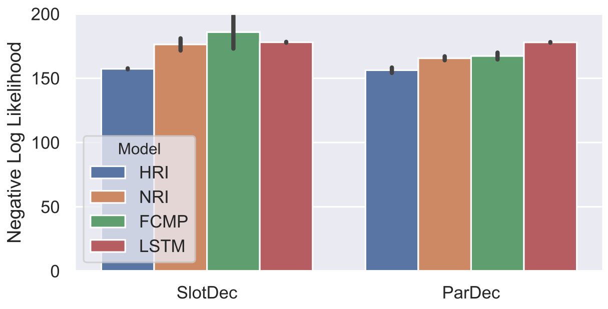


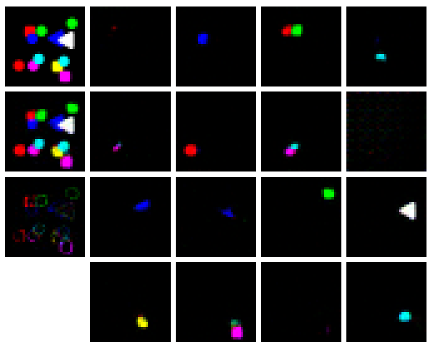
Figure A.11: (a) HRI introspection: first column contains ground-truth, prediction and their difference. The other columns show the 16 object slots decoded separately. (b) Negative log likelihood for all models on the *Triangles* dataset.

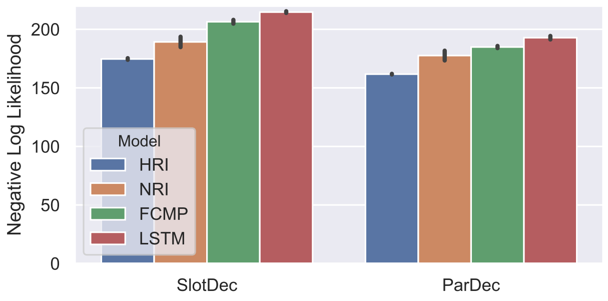

(a)



(b)

Figure A.12: (a) HRI introspection: first column contains ground-truth, prediction and their difference. The other columns show the 16 object slots decoded separately. (b) Negative log likelihood for all models on the *Squares* dataset.


(a)



(b)

Figure A.13: (a) HRI introspection: first column contains ground-truth, prediction and their difference. The other columns show the 16 object slots decoded separately. (b) Negative log likelihood for all models on the *Diverse-objects* dataset.

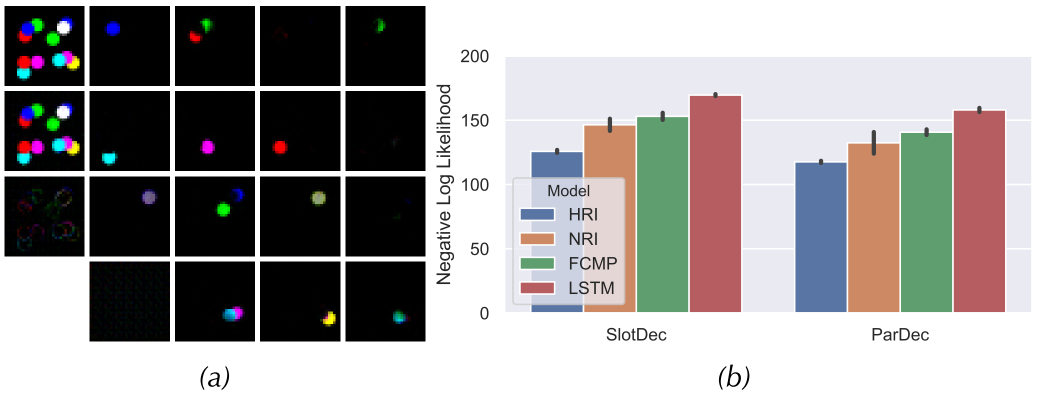


Figure A.14: (a) HRI introspection: first column contains ground-truth, prediction and their difference. The other columns show the 16 object slots decoded separately. (b) Negative log likelihood for all models on the *Large-Objects* dataset.

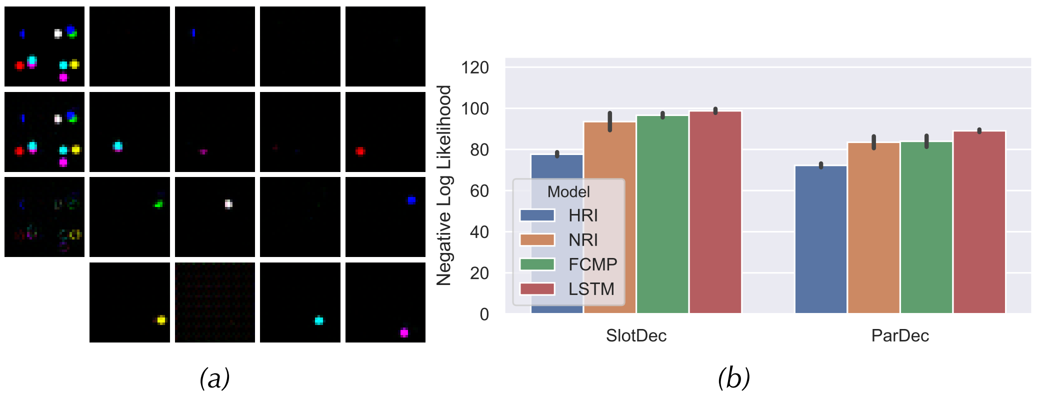
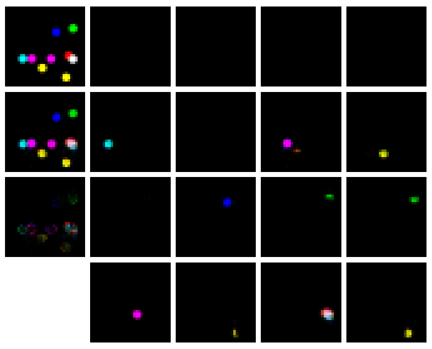


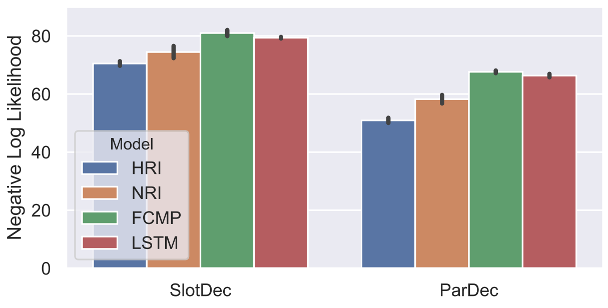
Figure A.15: (a) HRI introspection: first column contains ground-truth, prediction and their difference. The other columns show the 16 object slots decoded separately. (b) Negative log likelihood for all models on the *Curtain* dataset.

A.12 Visual Springs - Generalization to Different Number of Objects

We test whether HRI is able to generalize to scenarios with different number of objects by first training on 4-3-visual-springs and then testing on 3-3-visual springs. The qualitative results are shown in Fig. A.16a, where it can be seen that HRI is able to infer separate object representations even on a dataset it was not originally trained on. Quantitative comparison of model performance is shown in Fig. A.16b. Compared to Fig. A.4b we observe a slight drop in performance, but HRI remains the best performing model. A prediction sequence is shown in Fig. A.17. Note how HRI makes minor mistakes in the first few frames, but adapts afterwards to the setting with different number of objects from what it was trained on, and models the dynamics more accurately in subsequent frames.



(a)



(b)

Figure A.16: (a) HRI introspection: first column contains ground-truth, prediction and their difference. The other columns show the 16 object slots decoded separately. (b) Negative log likelihood for all models on trained on 4-3-visual-springs and evaluated on 3-3-visual-springs dataset.

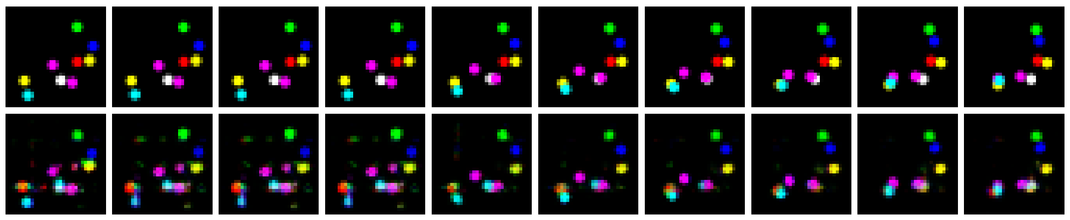


Figure A.17: Ground truth (top) and predicted (bottom) 10 time steps rollout of HRI trained on 4-3-visual springs and evaluated on 3-3-visual-springs.

Appendix B

Additional Details for Learning to Generalize with Object-centric Agents

B.1 OOD environment Objects

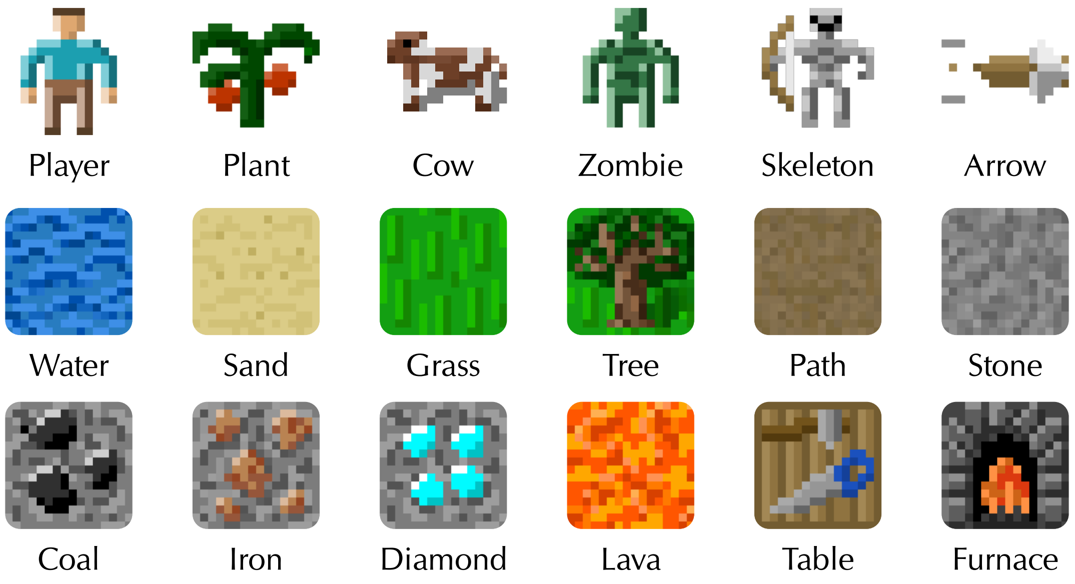


Figure B.1: Original Crafter objects. Figure from Hafner [2021].

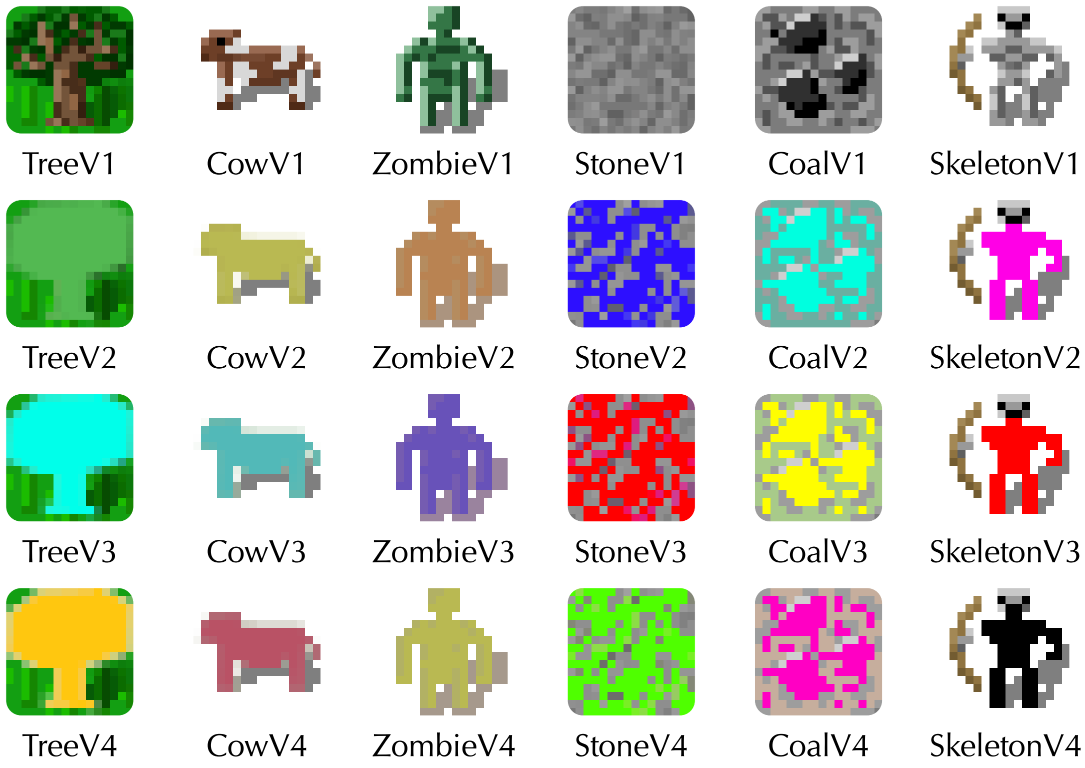


Figure B.2: In CrafterOODapp there are four variants of objects for trees, cows, zombies, stone coal and skeletons.

B.2 Network Configurations

Feature Extractor
8×8 conv, 32 ReLU units, stride 4 4×4 conv, 64 ReLU units, stride 2 3×3 conv, 64 ReLU units, stride 1 Flatten Linear, 512 ReLU units.
Action Network
Linear, 17 units.
Value Network
Linear, 1 units.

Table B.1: PPO-CNN: baseline agent with the CNN from DQN Mnih et al. [2015].

Feature Extractor
5×5 conv, 64 ReLU units, stride 1, padding 2 5×5 conv, 64 ReLU units, stride 1, padding 2 5×5 conv, 64 ReLU units, stride 1, padding 2 5×5 conv, 64 ReLU units, stride 1, padding 2 Flatten Linear, 512 ReLU units.
Action Network
Linear, 17 units.
Value Network
Linear, 1 units.

Table B.2: PPO-SPCNN: agent with the size-preserving CNN.

Feature Extractor
5×5 conv, 64 ReLU units, stride 1, padding 2 5×5 conv, 64 ReLU units, stride 1, padding 2 5×5 conv, 64 ReLU units, stride 1, padding 2 5×5 conv, 64 ReLU units, stride 1, padding 2 Split into 8×8 patches and flatten the patch grid
Self-Attention Network

Learned CLS token of 256 size.
Slot-wise projection: Linear, 256 units.
// Self-Attention layer 1:
Query map: Linear, 256 units.
Key map: Linear, 256 units.
Values map: Linear, 256 units.
// Self-Attention layer 2:
Query map: Linear, 256 units.
Key map: Linear, 256 units.
Values map: Linear, 256 units.

Action Network
Linear, 17 units.
Value Network
Linear, 1 units.

Table B.3: OC-SA: agent with the self-attention module.

Feature Extractor
5×5 conv, 64 ReLU units, stride 1, padding 2
5×5 conv, 64 ReLU units, stride 1, padding 2
5×5 conv, 64 ReLU units, stride 1, padding 2
5×5 conv, 64 ReLU units, stride 1, padding 2
Split into 16×16 patches and flatten the patch grid
Self-Attention Network
Learned slots 8×256 size.
Query map: Linear, 256 units.
Key map: Linear, 256 units.
Values map: Linear, 256 units.
Action Network
Linear, 17 units.
Value Network
Linear, 1 units.

Table B.4: OC-CA: agent with the cross-attention module.

Feature Extractor
8×8 conv, 32 ReLU units, stride 4
4×4 conv, 64 ReLU units, stride 2
3×3 conv, 64 ReLU units, stride 1
Flatten
Linear, 512 ReLU units.
Action Network
LSTM, 256 units.
Linear, 17 units.
Value Network
LSTM, 256 units.
Linear, 1 units.

Table B.5: LSTM-CNN: recurrent agent with an LSTM and a CNN from DQN Mnih et al. [2015].

Feature Extractor
5×5 conv, 64 ReLU units, stride 1, padding 2
5×5 conv, 64 ReLU units, stride 1, padding 2
5×5 conv, 64 ReLU units, stride 1, padding 2
5×5 conv, 64 ReLU units, stride 1, padding 2
Flatten
Linear, 512 ReLU units.
Action Network
LSTM, 256 units.
Linear, 17 units.
Value Network
Linear, 256 units.
Linear, 1 units.

Table B.6: LSTM-SPCNN: recurrent agent with an LSTM and a size-preserving CNN.

B.3 Object-centric agents formal details

In this section we provide a mathematical description of our object-centric agent implementations.

B.3.1 Object-centric Self-Attention (OC-SA) agents

OC-SA agent rely on simple Multi-Head Self-Attention mechanism Vaswani et al. [2017]. Formally, the inputs to the attention mechanism are N image patches (mapped through convolution layers) $\{x_i\}_{i=1}^{N} \in \mathbb{R}^{d_{in}}$ of dimension d_{in}. We treat the input patches as a set of vectors, and add one additional $CLS \in \mathbb{R}^{d_{in}}$ token Devlin et al. [2018], that is fed to the policy network after computing the attention with all other vectors from the $\{x_i\}_{i=1}^{N}$ set. Therefore, the input to the self-attention is the set of vectors $(x_1, .., x_N, CLS) \in \mathbb{R}^{N \times d_{in}}$. From each input vector, a query, key and value vectors are computed, using projection matrices W^Q, W^K, W^V. Query, key and value vectors are all of the same dimensionality d, resulting in aggregated $Q \in \mathbb{R}^{(N+1)\times d}, K \in \mathbb{R}^{(N+1)\times d}, V \in \mathbb{R}^{(N+1)\times d}$ matrices. We compute dot products of each query is computed with all the keys and scaled by $\sqrt{d}$. We then use softmax over the query dimension to obtain the attention weights over the value vectors. Formally, scaled dot-product attention is computed as:

$$Attention(Q, K, V) = softmax\Big(\frac{QK^T}{\sqrt{d}}\Big)V. \tag{B.1}$$

In practice, we use multi-head self-attention variant Vaswani et al. [2017], which allows the model to compute different mappings in each head and aggregate it in the end. Formally, multi-head self-attention is defined as:

$$MultiHeadSelfAttention(Q, K, V) = Concat(AttentionHead_1, .., AttentionHead_h), \tag{B.2}$$

where h is the number of heads and

$$AttentionHead_i = Attention(QW_i^Q, KW_i^K, VW_i^V), \tag{B.3}$$

where the projection matrices $W_i^Q \in \mathbb{R}^{d \times d_{model}}$, $W_i^K \in \mathbb{R}^{d \times d_{model}}$, $W_i^V \in \mathbb{R}^{d \times d_{model}}$, where $d_{model} = d/h$.

B.3.2 Object-centric Cross-Attention (OC-CA) agents

OC-CA agents are similar in spirit to cross-Attention (e.g. Jaegle et al. [2021b]) and slot-attention (e.g. Locatello et al. [2020]) models. Similarly to self-attention

Algorithm 2: Modified slot-attention algorithm sketch (modified from Locatello et al. [2020]). The input is a set of N_{inputs} vectors of dimension d which is mapped to a set of N_{slots} slots of dimension d. We initialize the slots by sampling their initial values as independent samples from a Gaussian distribution with shared, learnable parameters $\mu \in \mathbb{R}^d$ and $\sigma \in \mathbb{R}^d$.

```
1: Input: inputs $\in \mathbb{R}^{N_{inputs} \times d}$, slots $\sim \mathcal{N}(\mu,\ \text{diag}(\sigma)) \in \mathbb{R}^{N_{slots} \times d}$
2: Layer params: $k,\ q,\ v$: linear projections for attention; Optional: MLP and LayerNorm (x3)
3:   inputs = LayerNorm (inputs)                    ▷ # optional LayerNorm
4:   slots = LayerNorm (slots)                      ▷ # optional LayerNorm
5:   attn = $\text{softmax} \frac{1}{\sqrt{d}} k(\text{inputs}) \cdot q(\text{slots})^T$, axis=`slots/queries'
6:
  slot_updates = WeightedMean (weights=attn + $\epsilon$, values=$v$(inputs))
7:   slots += MLP (LayerNorm (slot_updates))   ▷ # optional residual MLP (per slot)
8:   return slots
```

keys and the values in slot-/cross-attention are also computed from the input which in our case are N image patches (mapped through convolution layers) $\{x_i\}_{i=1}^N \in \mathbb{R}^d$ of dimension d. However, unlike in self-attention, in cross-attention the queries come from a separate set of *learned* vectors called *slots*. In essence, this module maps from a set of N_{inputs} to a set of N_{slots}. Intuitively, each slot can learn to describe an object or an entity in the input.

A sketch of the Slot-Attention algorithm is shown in Algorithm 2 and a sketch of the Cross-Attention algorithm is shown in Algorithm 3. The main difference between slot- and cross-attention is that in the slot-attention softmax is normalized over slots (queries) dimension, and in the cross-attention softmax is normalized over keys (as in standard self-attention). It is argued Locatello et al. [2020] that softmax normalization over slots (queries) ensures that the attention coefficients sum to one for each individual input feature vector, which prevents the attention mechanism from ignoring parts of the input. In our work OC-CA uses the cross-attention mechanism. We do however experiment with slot-attention too (OC-CA + Slot Competition entry in Table 4.8). Note also the optional LayerNorm and residual MLP steps in both algorithms (these were also ablated in Table 4.8).

Lastly, note the differences of our 'modified' slot-attention with respect to the original slot-attention algorithm Locatello et al. [2020]. Firstly, we don't use a recurrent network to combine the new updates with the previous slots. More importantly, we also do not use multiple iterations of the attention. Therefore, our algorithm can be understood as a single iteration of the original slot-attention and without a recurrent network. We experimented with both using multiple iterations as well as the recurrent network, but we observed suboptimal performance.

Algorithm 3: Cross-Attention algorithm sketch (modified from Locatello et al. [2020]). The input is a set of N_{inputs} vectors of dimension d which is mapped to a set of N_{slots} slots of dimension d. We initialize the slots by sampling their initial values as independent samples from a Gaussian distribution with shared, learnable parameters $\mu \in \mathbb{R}^d$ and $\sigma \in \mathbb{R}^d$.

```
1: Input: inputs $\in \mathbb{R}^{N_{inputs} \times d}$, slots $\sim \mathcal{N}(\mu,\ \mathrm{diag}(\sigma)) \in \mathbb{R}^{N_{slots} \times d}$
2: Layer params: $k,\ q,\ v$: linear projections for attention; Optional: MLP and LayerNorm(x3)
3:   inputs = LayerNorm(inputs)                      ▷ # optional LayerNorm
4:   slots = LayerNorm(slots)                        ▷ # optional LayerNorm
5:   attn = $\mathrm{softmax}\frac{1}{\sqrt{d}}k(\texttt{inputs}) \cdot q(\texttt{slots})^T$, axis=`keys'
6:   slot_updates = attn $\cdot\, v(\texttt{inputs})$
7:   slots += MLP(LayerNorm(slot_updates))   ▷ # optional residual MLP (per slot)
8:   return slots
```

More formally, in slot-/cross-attention slots are initialized at random by sampling them from a Gaussian distribution, which allows them to generalize to different number of slots at test time. The slots play the role of 'queries' in the attention mechanism and can 'bind' to specific features by computing the dot-product with keys and attending to values (both coming from the input). Dot-product is computed as:

$$S = \frac{1}{\sqrt{d}} k(\texttt{inputs}) \cdot q(\texttt{slots})^T \in \mathbb{R}^{N_{inputs} \times N_{slots}}. \tag{B.4}$$

The main difference between slot- and cross-attention lies in how the softmax is normalized. In slot-attention, we normalize over slots (queries):

$$attn_{i,j} = \frac{e^{M_{i,j}}}{\sum_i e^{M_{i,j}}}, \tag{B.5}$$

and in cross-attention we normalize over keys:

$$attn_{i,j} = \frac{e^{M_{i,j}}}{\sum\limits_{j} e^{M_{i,j}}}. \tag{B.6}$$

Slot updates in cross-attention are then computed as a weighted sum between attention coefficients and the values:

$$\texttt{slot\_updates} = attn^T \cdot v(\texttt{inputs}), \tag{B.7}$$

whereas the slot-attention computes the weighted mean:

$$\texttt{slot\_updates} = attn\_mean^T \cdot v(\texttt{inputs}), \tag{B.8}$$

where $attn\_mean_{i,j} = \frac{attn_{i,j}}{\sum\limits_{i=1}^{N} attn_{i,j}}$.

Authors in the original slot-attention paper Locatello et al. [2020] argue that the weighted mean helps improve stability of the attention mechanism. Lastly note also that both slot- and cross-attention are permutation invariant with respect to the input (the output is independent of permutations applied to the input) and permutation equivariant with respect to the slots (output is permuted in the same way as the order of the slots is permuted). For the proof we refer to the slot-attention paper Locatello et al. [2020].

B.4 Off-policy algorithms on Crafter

We trained and evaluated an off-policy (DQN) agent on the original Crafter environment. After extensive grid search over all hyperparameters, we were unable to get good performance. The highest score we obtained was 4.5 (in the original Crafter paper a score of 4.3 was reported for Rainbow). Qualitative inspection of the agent's behavior when the score is 4.5 showed that the agent is not learning any meaningful behavior, but simply roams randomly around the map and obtains a few points if it drinks water, sleeps, picks a tree or kills an enemy.

Reasons for why the off-policy methods perform worse than on-policy ones are unclear, but we can speculate on a few things that could work in favor of on-policy algorithms. In general, on-policy learning can be better when training an agent that needs to perform a significant amount of exploration, as is the case for Crafter. Another reason could be that PPO is a policy gradient method, so as long as the learned policy is somewhat good, the value functions need not be very accurate. On the other hand DQN needs to learn a correct value function from which it then samples actions. If learning the value function is too hard to do (as it may be the case in Crafter), then the DQN will be at a disadvantage compared to the PPO.

B.5 CrafterOODapp performance

In Fig. B.3 we provide heatmap of the CrafterOODapp scores previously reported in Table 4.4.

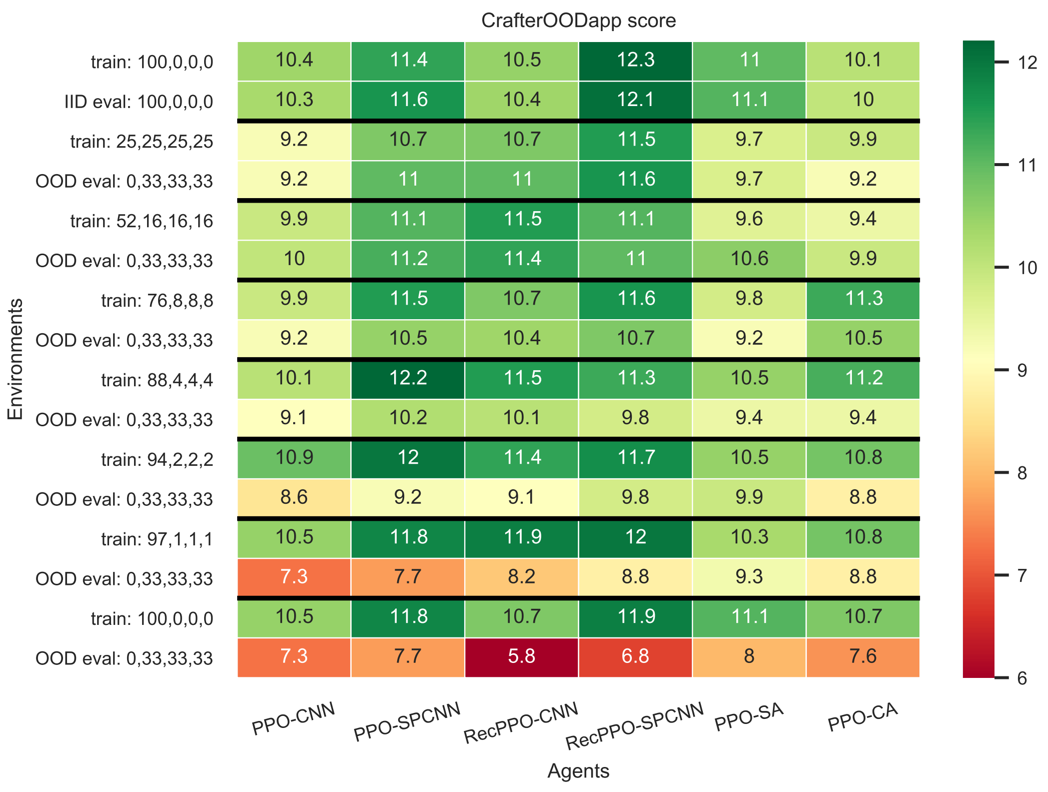
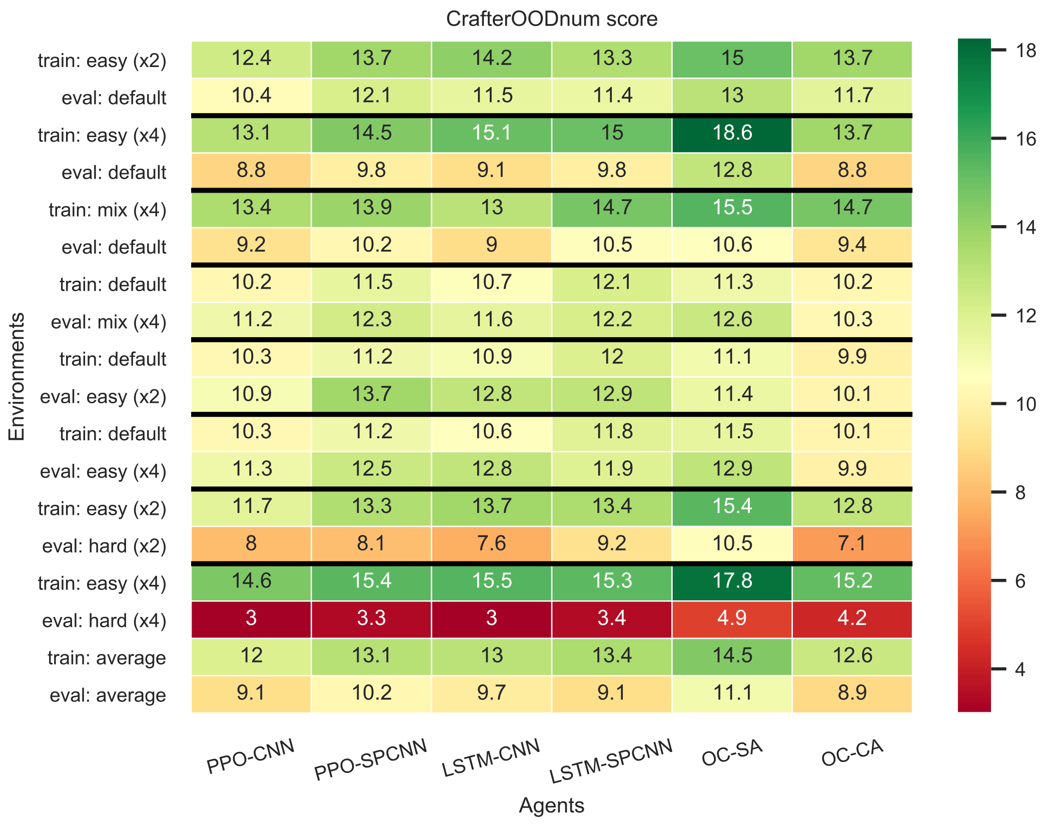


Figure B.3: Scores on CrafterOODnum for agents trained for 1M environment steps. Mean over 10 random seeds are reported. For standard deviations see Table 4.4. Each setting has two rows, denoting scores in training (e.g. $25, 25, 25, 25$ *for* $O_{1-4} : 25\%$*) and evaluation (*$0, 33, 33, 33$*, for* $O_1 : 0\%, O_{2-4} : 33.3\%$*) environments.*

B.6 CrafterOODnum performance

In Fig. B.4 we provide heatmap of the CrafterOODnum scores previously reported in Table 4.6.



Environments	PPO-CNN	PPO-SPCNN	LSTM-CNN	LSTM-SPCNN	OC-SA	OC-CA
train: easy (x2)	12.4	13.7	14.2	13.3	15	13.7
eval: default	10.4	12.1	11.5	11.4	13	11.7
train: easy (x4)	13.1	14.5	15.1	15	18.6	13.7
eval: default	8.8	9.8	9.1	9.8	12.8	8.8
train: mix (x4)	13.4	13.9	13	14.7	15.5	14.7
eval: default	9.2	10.2	9	10.5	10.6	9.4
train: default	10.2	11.5	10.7	12.1	11.3	10.2
eval: mix (x4)	11.2	12.3	11.6	12.2	12.6	10.3
train: default	10.3	11.2	10.9	12	11.1	9.9
eval: easy (x2)	10.9	13.7	12.8	12.9	11.4	10.1
train: default	10.3	11.2	10.6	11.8	11.5	10.1
eval: easy (x4)	11.3	12.5	12.8	11.9	12.9	9.9
train: easy (x2)	11.7	13.3	13.7	13.4	15.4	12.8
eval: hard (x2)	8	8.1	7.6	9.2	10.5	7.1
train: easy (x4)	14.6	15.4	15.5	15.3	17.8	15.2
eval: hard (x4)	3	3.3	3	3.4	4.9	4.2
train: average	12	13.1	13	13.4	14.5	12.6
eval: average	9.1	10.2	9.7	9.1	11.1	8.9

Figure B.4: Scores on CrafterOODapp for agents trained for 1M environment steps. Mean over 10 random seeds are reported. For standard deviations see Table 4.6. Each setting has two rows, denoting scores in training and evaluation environments.

B.7 Hyper-parameter heatmaps

In Figs. B.5 and B.6 we show the pairwise heatmaps of the hyper-parameters we tuned for the PPO-CNN. Important to note here is that once we tuned the hyper-parameters for PPO-CNN, we fixed them and used the same values for all other models. Therefore, other models might have a slight disadvantage as we did not specifically tune hyper-parameters for each model.

In Fig. B.5 we start from the default hyper-parameters from Raffin et al. [2021] that were tuned for Atari: GAE lambda 0.95, number of epochs 10, gamma 0.99, batch size 64 and number of steps 1024. From here, we sweep over individual pairs of hyper-parameters, e.g. GAE lambda and the number of epochs. Looking at the heatmaps, it is clear how each hyper-parameters contributes to the improvement (which becomes evident by looking at their combination in Fig. B.6). However, some hyper-parameters contribute much more than others, e.g. GAE lambda and the number of epochs the agent is trained on the collected rollouts. Given these plots, we settled on the following values for hyper-parameters: number of steps 4096, number of epochs 4, batch size 128, GAE lambda 0.65, and gamma 0.95. GAE-lambda can be interpreted as an extra discount factor applied after performing reward shaping transformation on the MDP Schulman et al. [2015], and we found low values to perform better. This also makes sense given that we found lower discount factors also to give better performance (0.95 compared to the most commonly used 0.99). On the other hand, we extended the horizon (number of rollout steps collected for the update) from default 1024 to 4096) and similarly increased batch size from 64 to 128, stabilizing training agents in Crafter. In general, when there are frequent rewards within an episode, the number of steps can be smaller. However, in Crafter, the rewards are sparse, which is also reflected in empirically better performance when using longer rollouts.

In Fig. B.5 we start from the best hyper-parameters we found: number of steps 4096, number of epochs 4, batch size 128, GAE lambda 0.65 and gamma 0.95. We then ablate pairs of hyper-parameters by changing them over their respective ranges. The agents are fairly robust to hyper-parameters changes. From the heatmaps, we can see that the performance drops going away from the optimal values, though not by a significant value.

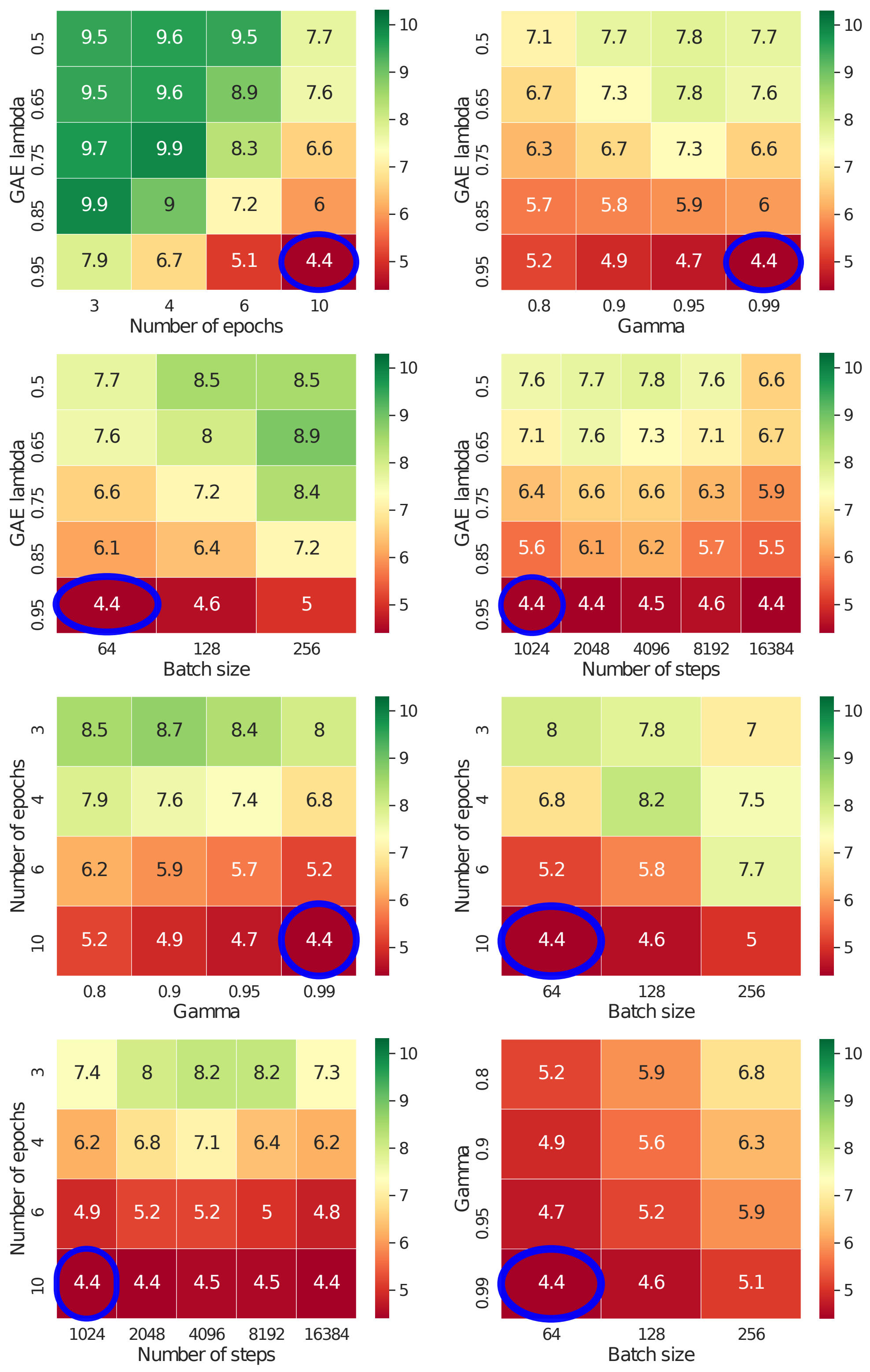


Figure B.5: Crafter scores starting from the default hyper-parameters from Raffin et al. [2021] that were tuned for Atari: GAE lambda 0.95, number of epochs 10, gamma 0.99, batch size 64 and number of steps 1024, and then ablating individual hyper-parameter pairs. For each pair of hyper-parameters, the default values are marked by a blue circle.

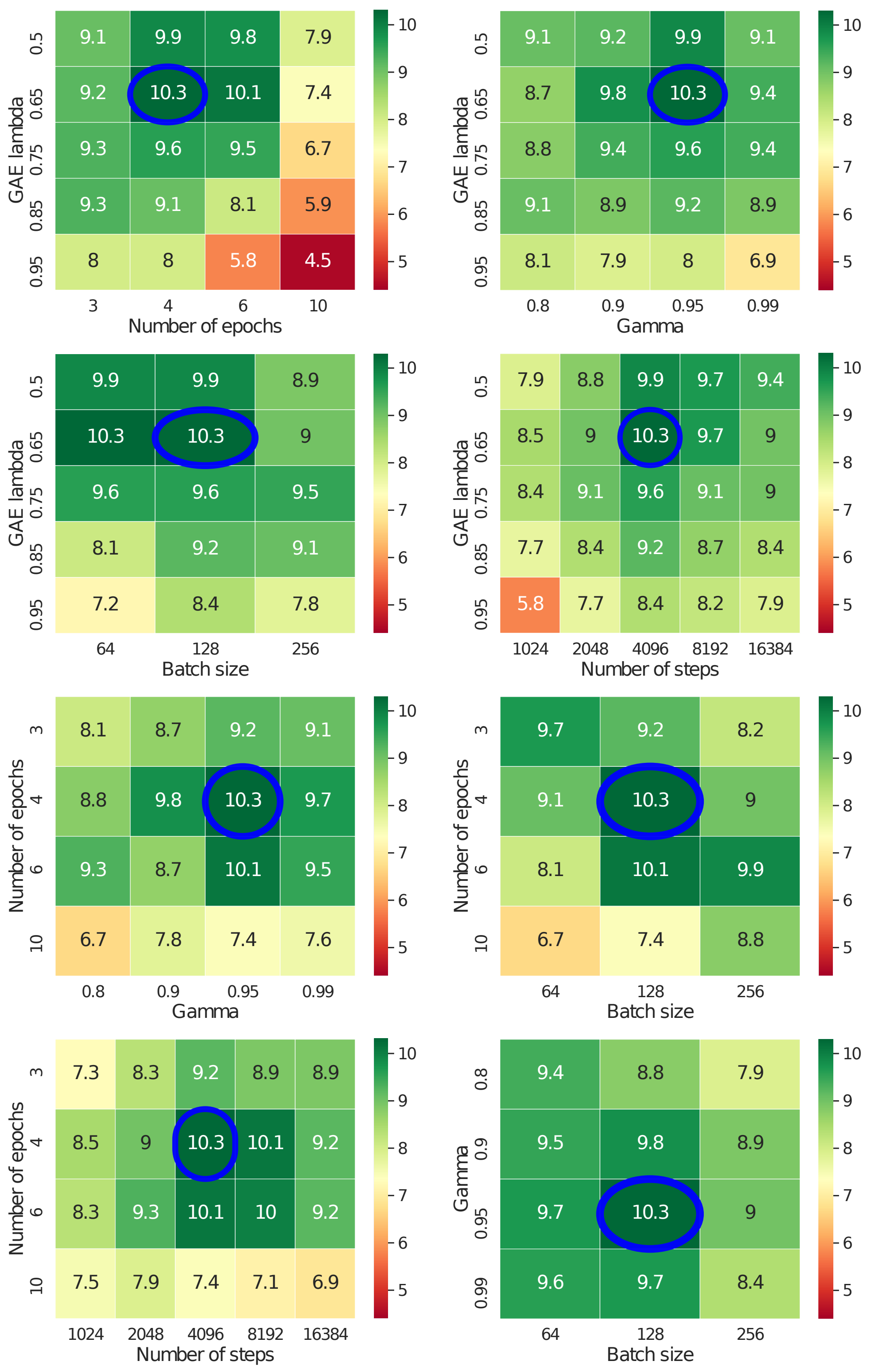


Figure B.6: Crafter scores starting from the best hyper-parameters we found: GAE lambda 0.65, number of epochs 4, gamma 0.95, batch size 128 and number of steps 4096, and then ablating individual hyper-parameter pairs. For each pair of hyper-parameters, the best values are marked by a blue circle.

Appendix C

Additional Details for Synchrony-based Object Discovery with Complex-Valued Autoencoders

C.1 Experimental Details

Datasets. We evaluate all models (i.e., CAE, CAE++ and CtCAE) on a subset of Multi-Object dataset Kabra et al. [2019] suite. We use three datasets: `Tetrominoes` which consists of colored tetris blocks on a black background, `dSprites` with colored sprites of various shapes like heart, square, oval, etc., on a grayscale background, and lastly, `CLEVR`, a dataset from a synthetic 3D environment. For `CLEVR`, we use the filtered version [Emami et al., 2021] which consists of images containing less than seven objects sometimes referred to as `CLEVR6` as in Locatello et al. [2020]. We normalize all input RGB images to have pixel values in the range $[0, 1]$ consistent with prior work [Löwe et al., 2022].

Models. Table C.1 shows the architecture specifications such as number of layers, kernel sizes, stride lengths, number of filter channels, normalization layers, activations, etc., for the convolutional neural networks used by the Encoder and Decoder modules in CAE, CAE++ and CtCAE.

Training Details. Table C.2 and Table C.3 show the hyperparameter configurations used to train CAE/CAE++ and to contrastively train the CtCAE model(s) respectively.

Encoder
3 × 3 conv, 128 channels, stride 2, ReLU, BatchNorm
3 × 3 conv, 128 channels, stride 1, ReLU, BatchNorm
3 × 3 conv, 256 channels, stride 2, ReLU, BatchNorm
3 × 3 conv, 256 channels, stride 1, ReLU, BatchNorm
3 × 3 conv, 256 channels, stride 2, ReLU, BatchNorm
———————————————————
For 64×64 and 96×96 inputs, 2 additional encoder layers:
3 × 3 conv, 256 channels, stride 1, ReLU, BatchNorm
3 × 3 conv, 256 channels, stride 2, ReLU, BatchNorm
Linear Layer
Flatten
Linear, 256 ReLU units, LayerNorm
Decoder
For 64×64 and 96×96 inputs, 2 additional decoder layers:
Bilinear upsample x2
3 × 3 conv, 256 channels, stride 1, ReLU, BatchNorm
3 × 3 conv, 256 channels, stride 1, ReLU, BatchNorm
———————————————————
Bilinear upsample x2
3 × 3 conv, 256 channels, stride 1, ReLU, BatchNorm
3 × 3 conv, 256 channels, stride 1, ReLU, BatchNorm
Bilinear upsample x2
3 × 3 conv, 256 channels, stride 1, ReLU, BatchNorm
3 × 3 conv, 128 channels, stride 1, ReLU, BatchNorm
Bilinear upsample x2
3 × 3 conv, 128 channels, stride 1, ReLU, BatchNorm
Output Layer (CAE only)
1 × 1 conv, 3 channels, stride 1, sigmoid

Table C.1: Encoder and Decoder architecture specifications for CAE, CAE++ and CtCAE models.

Computational Efficiency. We report the training and inference time (wall-clock) for our models across the various image resolutions of the 3 multi-object

Hyperparameter	Tetrominoes	dSprites	CLEVR
Training Steps	50'000	100'000	100'000
Batch size	64	64	64
Learning rate	4e-4	4e-4	4e-4

Table C.2: General training hyperparameters.

Common Hyperparameters			
Loss coefficient (in total loss sum)	1e-4		
Temperature	0.05		
Contrastive Learning Addresses	Magnitude		
Contrastive Learning Features	Phase		
Encoder Hyperparameters	**32x32**	**64x64**	**96x96**
Number of anchors	4	4	8
Number of positive pairs	1	1	1
Top-K to select positive pairs from	1	1	1
Number of negative pairs	2	2	16
Bottom-M to select negative pairs from	2	2	24
Decoder Hyperparameters	**32x32**	**64x64**	**96x96**
Patch Size	1	2	3
Number of anchors	100	100	100
Number of positive pairs	1	1	1
Top-K to select positive pairs from	5	5	5
Number of negative pairs	100	100	100
Bottom-M to select negative pairs from	500	500	500

Table C.3: Contrastive learning hyperparameters for the CtCAE model.

datasets. First, we find that inference time(s) is similar for all models, and it only depends on the image resolution and number of ground truth clusters in the input image. Inference for the test set containing 320 images of 32x32 resolution takes 25 seconds, for 64x64 images it takes 65 seconds and for the 96x96 images it takes 105 seconds. Training time(s) on the other hand differs both across models and image resolutions. We report all training time(s) using a single Nvidia V100 GPU. CAE and CAE++ have similar training times: for 32x32 images training for 100k steps takes 3.2 hours, for 64x64 images it takes 4.8 hours, and for 96x96 images it takes 7 hours. CtCAE on the other hand takes 5 hours, 9.2 hours and 10.2 hours to train on dataset of 32x32, 64x64 and 96x96 images respectively. To reproduce all the results/tables (mean and std-dev across 5 seeds) reported in this work we estimate the compute requirement to be 840 GPU hours in total. Further, we estimate that the total compute used in this project is approximately 5-10x more, including experiments with preliminary prototypes.

Object Assignments from Phase Maps. We describe the process of extracting discrete-valued object assignments from continuous-valued phase components of decoder outputs $\mathbf{y} \in \mathbb{C}^{h \times w \times 3}$. First, the phase components of decoder outputs are mapped onto a unit circle and those phase values are masked out whose magnitudes are below the threshold value of 0.1. After this normalization and filtering step, the resultant phase values are converted from polar to Cartesian form on a per-channel basis. Finally, we apply K-means clustering where the number of clusters K parameter is retrieved from the ground-truth segmentation mask for each image. This extraction methodology of object assignments from output phase maps is consistent with Löwe et al. [2022].

Dataset	Model	Threshold = 0.1		Threshold = 0.0		No threshold	
		ARI-FG ↑	ARI-FULL ↑	ARI-FG ↑	ARI-FULL ↑	ARI-FG ↑	ARI-FULL ↑
Tetrominoes	CAE++	0.78 ± 0.07	0.84 ± 0.01	0.77 ± 0.07	0.79 ± 0.02	0.54 ± 0.09	0.21 ± 0.05
	CtCAE	0.84 ± 0.09	0.85 ± 0.01	0.86 ± 0.05	0.82 ± 0.01	0.67 ± 0.11	0.26 ± 0.09
dSprites	CAE++	0.38 ± 0.05	0.49 ± 0.15	0.38 ± 0.05	0.49 ± 0.12	0.37 ± 0.05	0.49 ± 0.12
	CtCAE	0.48 ± 0.03	0.68 ± 0.13	0.46 ± 0.07	0.69 ± 0.10	0.47 ± 0.06	0.69 ± 0.10
CLEVR	CAE++	0.22 ± 0.10	0.30 ± 0.18	0.33 ± 0.04	0.32 ± 0.25	0.34 ± 0.04	0.32 ± 0.25
	CtCAE	0.50 ± 0.05	0.69 ± 0.25	0.52 ± 0.05	0.69 ± 0.20	0.52 ± 0.06	0.72 ± 0.21

Table C.4: Grouping results for CAE++ and CtCAE models with different threshold values applied to post-process the continuous output phase maps.

C.2 Additional results

We show some additional experimental results: generalization capabilities of CtCAE w.r.t. the number of objects, and ablation studies on various design choices made in CAE++ and CtCAE models.

Evaluation	ARI-FG ↑	ARI-FULL ↑
Training Subset: 4 or less objects		
3 objects	0.39 ± 0.20	0.53 ± 0.30
4 objects	**0.41 ± 0.18**	**0.55 ± 0.29**
5 objects	0.38 ± 0.15	0.54 ± 0.28
6 objects	0.38 ± 0.13	0.55 ± 0.24
All images	0.39 ± 0.04	0.54 ± 0.28
Training Subset: 5 or less objects		
3 objects	0.49 ± 0.04	**0.69 ± 0.25**
4 objects	**0.50 ± 0.03**	0.66 ± 0.24
5 objects	0.46 ± 0.02	0.65 ± 0.24
6 objects	0.45 ± 0.03	0.64 ± 0.23
All images	0.48 ± 0.02	0.66 ± 0.24

Table C.5: Generalization evaluation on the CLEVR dataset. Training only on a subset, but evaluating on all possible subsets containing images with 3, 4, 5 or 6 objects.

C.2.1 Generalization to Higher Number of Objects

Here we evaluate generalization capabilities of CtCAE by training only on a subset of images containing less than a certain number of objects, on CLEVR. We have two cases—training on subsets with either up to 4 or up to 5 objects while the original trainig split contain between 3 and 6 objects. The results in Table C.5 show that CtCAE generalizes well. The performance drops only marginally when trained on up to 4 objects and tested on 5 and 6 objects, or when trained on up to 5 objects and tested on 6 objects. We also observe that training on 5 or less objects also consistently improves ARI scores for every subset of less than 5 objects. We suspect that the reason for this, apart from having more training data, is that the network has more pressure to separate objects in the phase space when observing more objects, which in turn also helps for images with a smaller number of objects.

Dataset	Model	ARI-FG ↑	ARI-FULL ↑
Tetrominoes	CAE	0.00 ± 0.00	0.12 ± 0.02
	CAE-(f_{out} `1x1` conv)	0.00 ± 0.00	0.00 ± 0.00
	CAE-(f_{out} sigmoid)	0.12 ± 0.12	0.35 ± 0.36
	CAE-transp.+upsamp.	0.10 ± 0.21	0.10 ± 0.22
	CAE++ (above combined)	0.78 ± 0.07	0.84 ± 0.01
	CtCAE	**0.84 ± 0.09**	**0.85 ± 0.01**
dSprites	CAE	0.02 ± 0.00	0.07 ± 0.01
	CAE-(f_{out} `1x1` conv)	0.06 ± 0.01	0.07 ± 0.07
	CAE-(f_{out} sigmoid)	0.02 ± 0.01	0.07 ± 0.01
	CAE-transp.+upsamp.	0.19 ± 0.02	0.10 ± 0.04
	CAE++ (above combined)	0.38 ± 0.05	0.49 ± 0.15
	CtCAE	**0.48 ± 0.03**	**0.68 ± 0.13**
CLEVR	CAE	0.09 ± 0.05	0.08 ± 0.06
	CAE-(f_{out} `1x1` conv)	0.05 ± 0.01	0.06 ± 0.01
	CAE-(f_{out} sigmoid)	0.06 ± 0.02	0.01 ± 0.02
	CAE-transp.+upsamp.	0.19 ± 0.07	0.10 ± 0.11
	CAE++ (above combined)	0.22 ± 0.10	0.30 ± 0.18
	CtCAE	**0.50 ± 0.05**	**0.69 ± 0.25**

Table C.6: Grouping metrics achieved by various model variants from our proposed architectural modifications and resulting finally in the CAE++ model. Extends the results of Table 5.4 to all 3 multi-object datasets.

C.2.2 Architecture Modifications

Table C.6 shows the results from the ablation study that measures the effect of applying each of our proposed architectural modifications cumulatively to finally result in the CAE++ model. Across all datasets, we consistently observe that, starting from the vanilla CAE baseline which completely fails, grouping performance gradually improves as more components of the CAE++ model is added. Lastly, our proposed contrastive objective used to train CtCAE further improves CAE++ across all 3 datasets.

C.2.3 Contrastive Learning Ablations

Table C.7 shows an ablation study for the choice of layer(s) to which we apply our contrastive learning method. Please note that all variants here always use magnitude components of complex-valued activations as *addresses* and phase components as *features* to contrast. We observe that the variant which applies the contrastive objective to both the outputs of the encoder and decoder ('enc + dec') outperforms the others which apply it only to either one ('enc-only' or 'dec-only'). This behavior is consistent across `Tetrominoes`, `dSprites` and `CLEVR`. We hypothesize that this is because these two contrastive strategies are complementary: one is on low-level cues (decoder) and the other one based on high-level abstract features (encoder).

Table C.8 shows an ablation study for the choice of using magnitude or phase as *addresses* (and the other one as *features*) in our contrastive learning method. As we apply contrastive learning to both the decoder and encoder outputs, we can make this decision independently for each output (resulting in four possible combinations). We observe that the variant which uses magnitude for both the encoder and decoder ('mg + mg') outperforms other variants across all 3 multi-object datasets. These ablations support our initial intuitions when designing our contrastive objective.

Dataset	Model	ARI-FG ↑	ARI-FULL ↑
Tetrominoes	CtCAE (enc-only)	0.81 ± 0.06	0.85 ± 0.01
	CtCAE (dec-only)	0.74 ± 0.04	**0.86 ± 0.00**
	CtCAE (enc + dec)	**0.84 ± 0.09**	0.85 ± 0.01
dSprites	CtCAE (enc-only)	0.40 ± 0.06	0.58 ± 0.05
	CtCAE (dec-only)	0.48 ± 0.05	**0.72 ± 0.07**
	CtCAE (enc + dec)	**0.48 ± 0.03**	0.68 ± 0.13
CLEVR	CtCAE (enc-only)	0.21 ± 0.11	0.29 ± 0.15
	CtCAE (dec-only)	0.38 ± 0.17	0.69 ± 0.18
	CtCAE (enc + dec)	**0.50 ± 0.05**	**0.69 ± 0.25**

Table C.7: Grouping metrics achieved by CtCAE model variants that apply the contrastive loss on output features from only the encoder (enc-only) or only the decoder (dec-only) or both (enc + dec) for all 3 multi-object datasets. Extends the results of Table 5.5 to all 3 multi-object datasets.

Dataset	Model	ARI-FG ↑	ARI-FULL ↑
Tetrominoes	CtCAE w/ mg+mg	**0.84** ± **0.09**	**0.85** ± **0.01**
	CtCAE w/ ph+ph	0.85 ± 0.05	0.84 ± 0.01
	CtCAE w/ mg+ph	0.86 ± 0.03	0.85 ± 0.02
	CtCAE w/ ph+mg	0.77 ± 0.04	0.85 ± 0.01
dSprites	CtCAE w/ mg+mg	**0.48** ± **0.03**	**0.68** ± **0.13**
	CtCAE w/ ph+ph	0.38 ± 0.08	0.42 ± 0.16
	CtCAE w/ mg+ph	0.36 ± 0.07	0.57 ± 0.13
	CtCAE w/ ph+mg	0.45 ± 0.05	0.67 ± 0.18
CLEVR	CtCAE w/ mg+mg	**0.50** ± **0.05**	**0.69** ± **0.25**
	CtCAE w/ ph+ph	0.22 ± 0.06	0.22 ± 0.09
	CtCAE w/ mg+ph	0.18 ± 0.08	0.28 ± 0.15
	CtCAE w/ ph+mg	0.40 ± 0.15	0.65 ± 0.19

Table C.8: Grouping metrics on all 3 multi-object datasets achieved by CtCAE model variants that use either magnitude or phase components of the encoder/decoder outputs as the *addresses* for contrastive learning. For example, 'mg+ph' means that magnitude components used as *addresses* of the encoder outputs and phase components used as *addresses* of the decoder outputs (conversely, the phase components of the encoder outputs are used as *features* and the magnitude components of the decoder outputs are used as *features*).

C.3 Additional Visualizations

We show additional visualization samples of groupings from CAE, CAE+ + and CtCAE on `Tetrominoes`, `dSprites` and `CLEVR`, and qualitatively highlight both grouping failures and successes of CAE, CAE+ + and CtCAE.

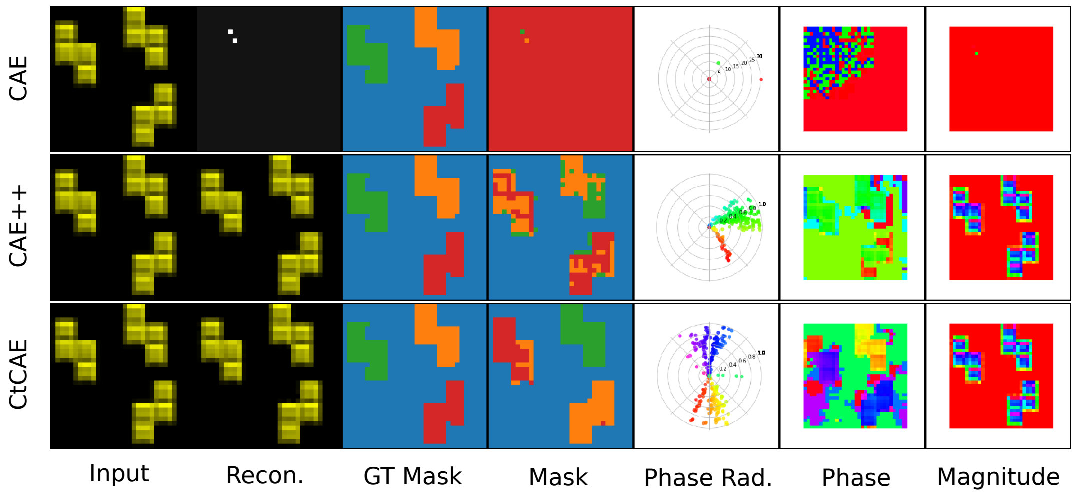


Figure C.1: Visualizations for CAE (row 1), CAE++ (row 2) and CtCAE (row 3) on the `Tetrominoes` dataset example from Figure 5.3. We see that CtCAE achieves near perfect grouping (column 4) and better *separability* in phase space (column 5) compared to our improved CAE++ variant which shows significant grouping interference (parts of all three objects modelled by same group, e.g., orange cluster). We also observe that the baseline CAE completely fails to reconstruct or group the image.

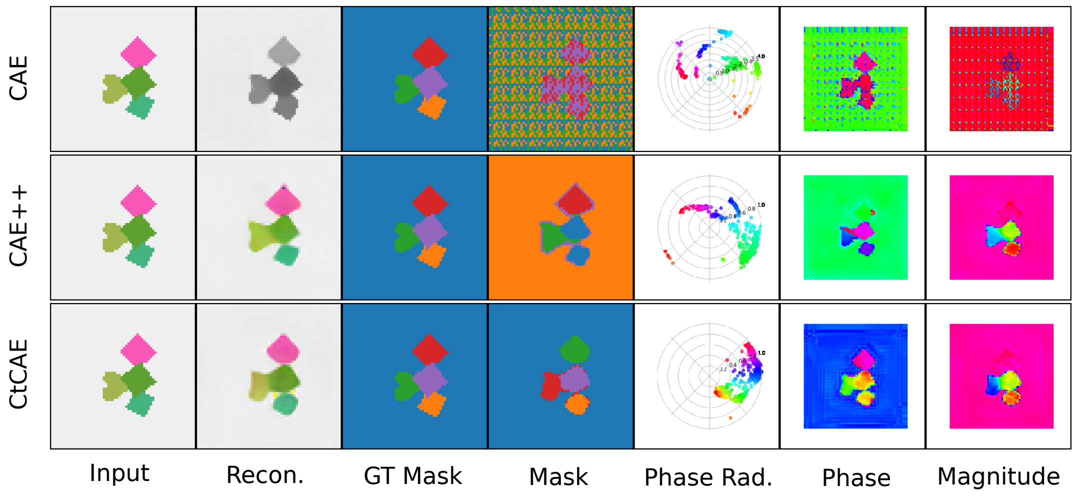


Figure C.2: Visualizations for CAE (row 1), CAE++ (row 2) and CtCAE (row 3) on the `dSprites` dataset example from Figure 5.3. We see that CtCAE successfully separates the 4 scene objects into 4 clusters while CAE++ mistakenly groups 2 scene objects into 1 cluster, i.e., the blue cluster (column 4). Again, the baseline CAE fails to reconstruct (no color) or group the image.

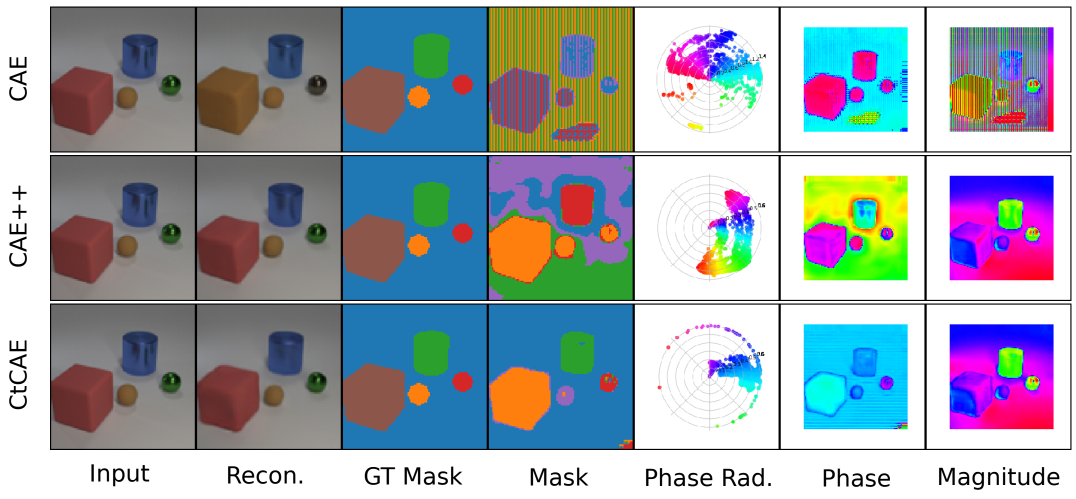


Figure C.3: Visualizations for CAE (row 1), CAE++ (row 2) and CtCAE (row 3) on the CLEVR dataset example from Figure 5.3. We see that the CtCAE in this example is able to separate the 4 scene objects into 4 distinct clusters while our improved CAE++ fails (column 4). This good separation in phase space is reflected in the continuous phase maps (column 6) where the pixel colors in the phase maps are representative of their values/differences when comparing CtCAE and CAE++. Finally, we see that although the CAE is able to reasonably reconstruct the image it shows very poor grouping of the scene objects.

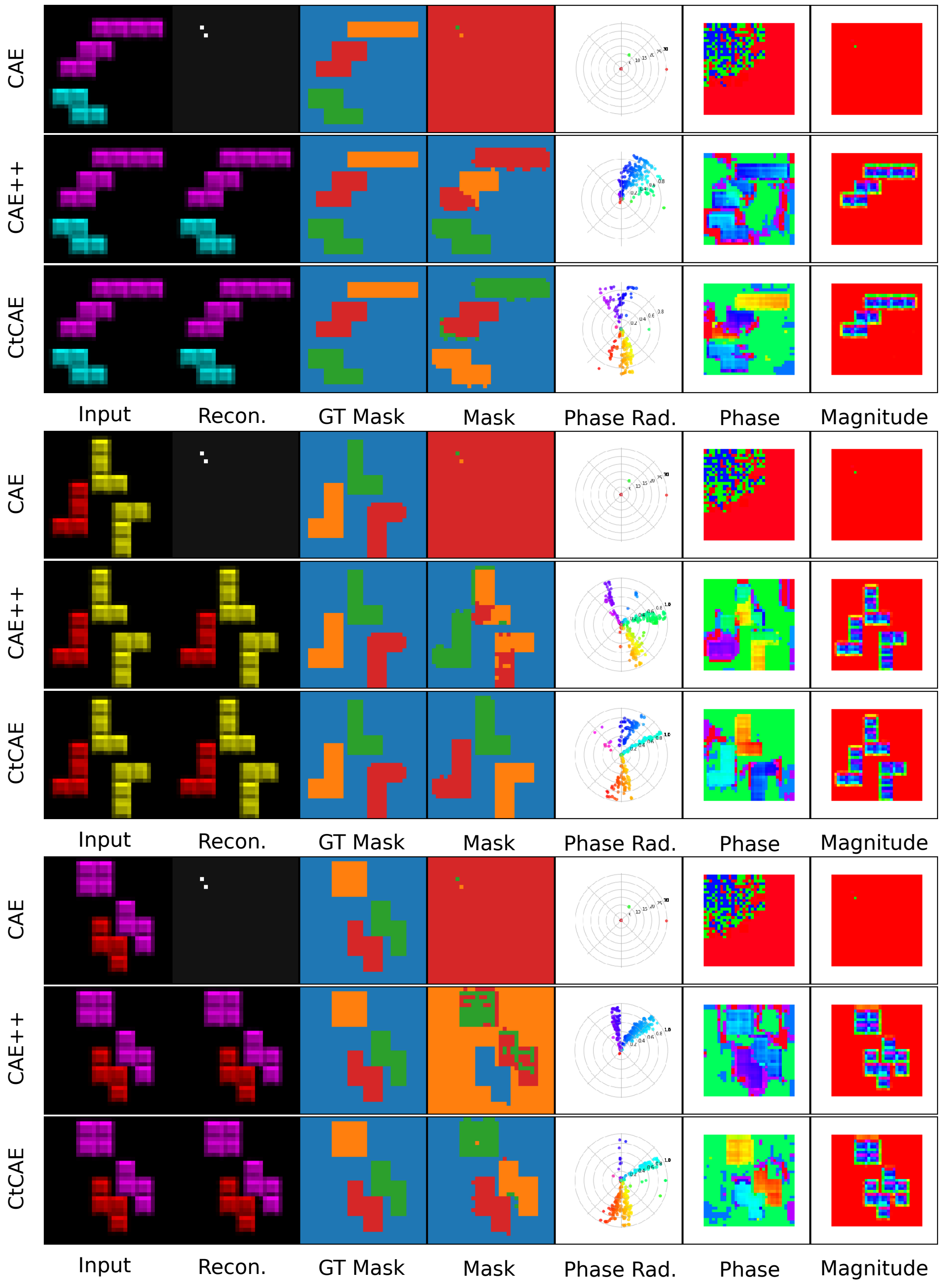


Figure C.4: Visualization on grouping performance for CAE, CAE + + and CtCAE on images containing same colored objects on `Tetrominoes`. CAE struggles to even reconstruct the images, whereas CAE + + often groups objects with the same colour together. CtCAE on the other hand has no problem separating objects of the same colour on `Tetrominoes`.

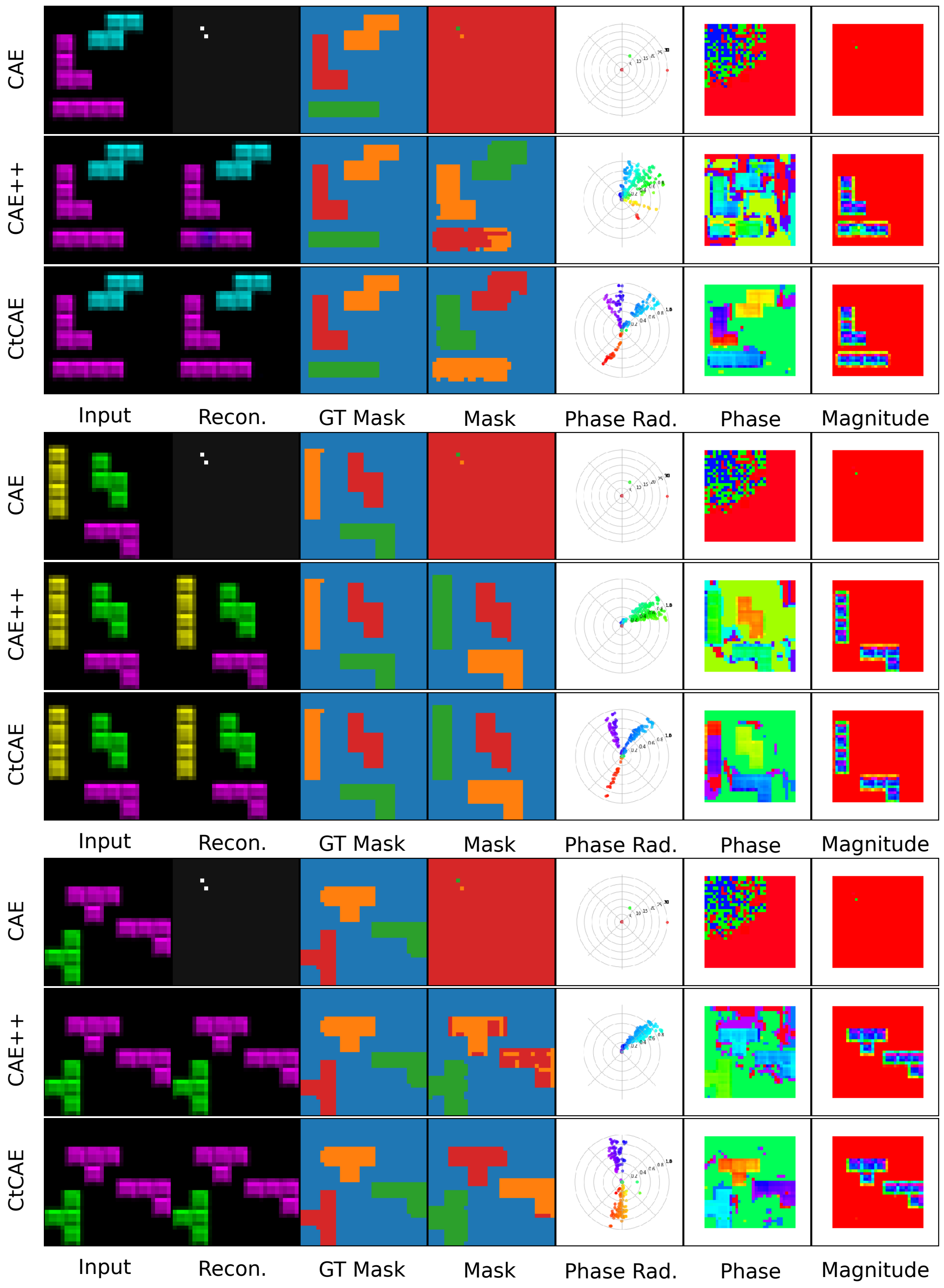


Figure C.5: Visualization on grouping performance for CAE, CAE++ and CtCAE on `Tetrominoes`. CtCAE shows a larger spread in phase values for different clusters (column 5) compared to CAE. We hypothesize that this facilitates CtCAE to perform better grouping in the scenario of multiple object instances with the same color.

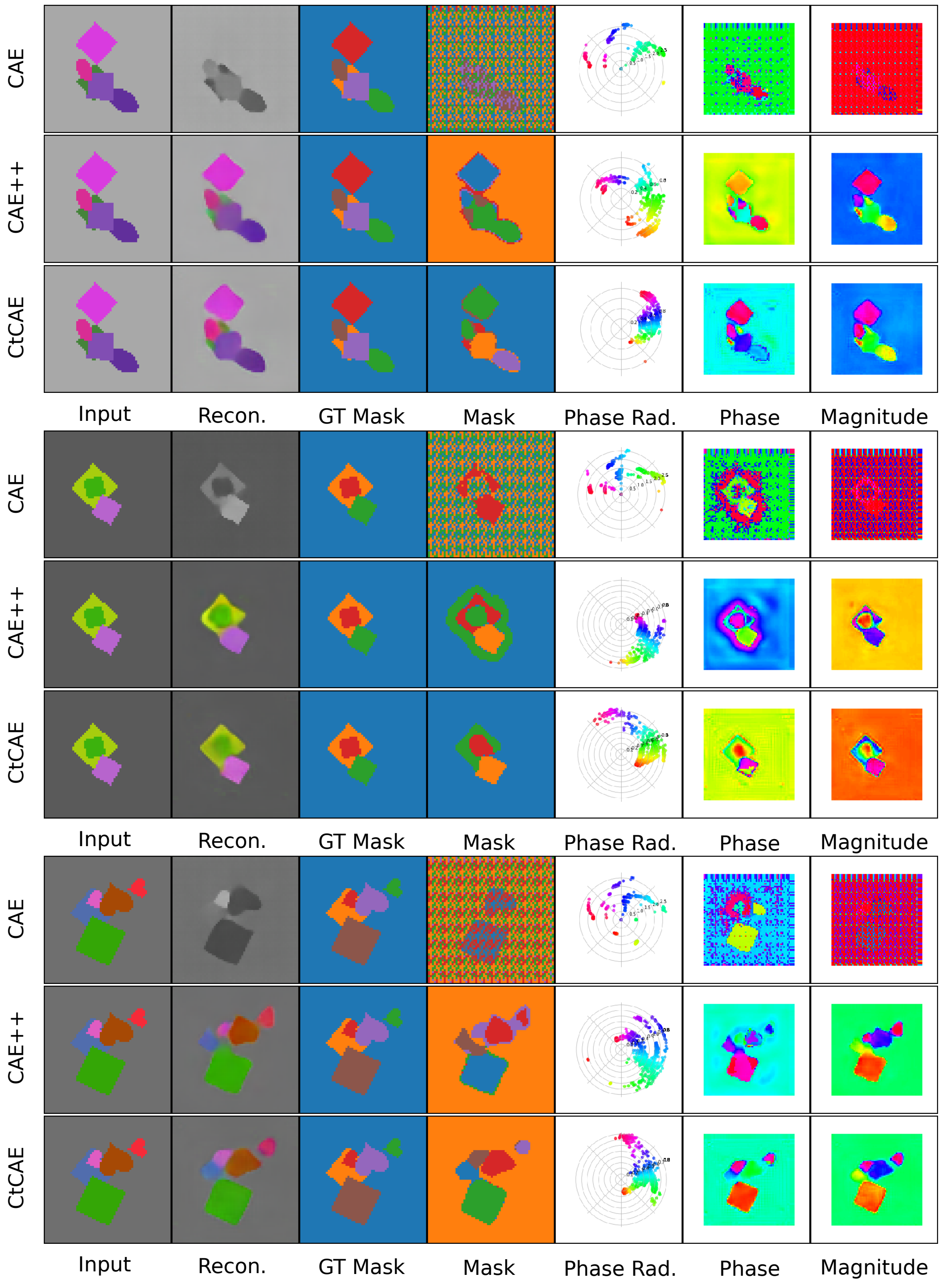


Figure C.6: Visualization on grouping performance for CAE, CAE + + and CtCAE on images containing similar colored objects on `dSprites`. CAE has very poor grouping capability, whereas CAE + + often groups objects with the same colour together For example, purple ellipse and square in the first example (top panel) or red and brown hearts grouped together in the last example (bottom panel) are grouped together. CtCAE on the other hand has significantly better performance, grouping correctly objects with the same (or similar) colors, except in the first example (top panel) where it groups the pink square and ellipse together.

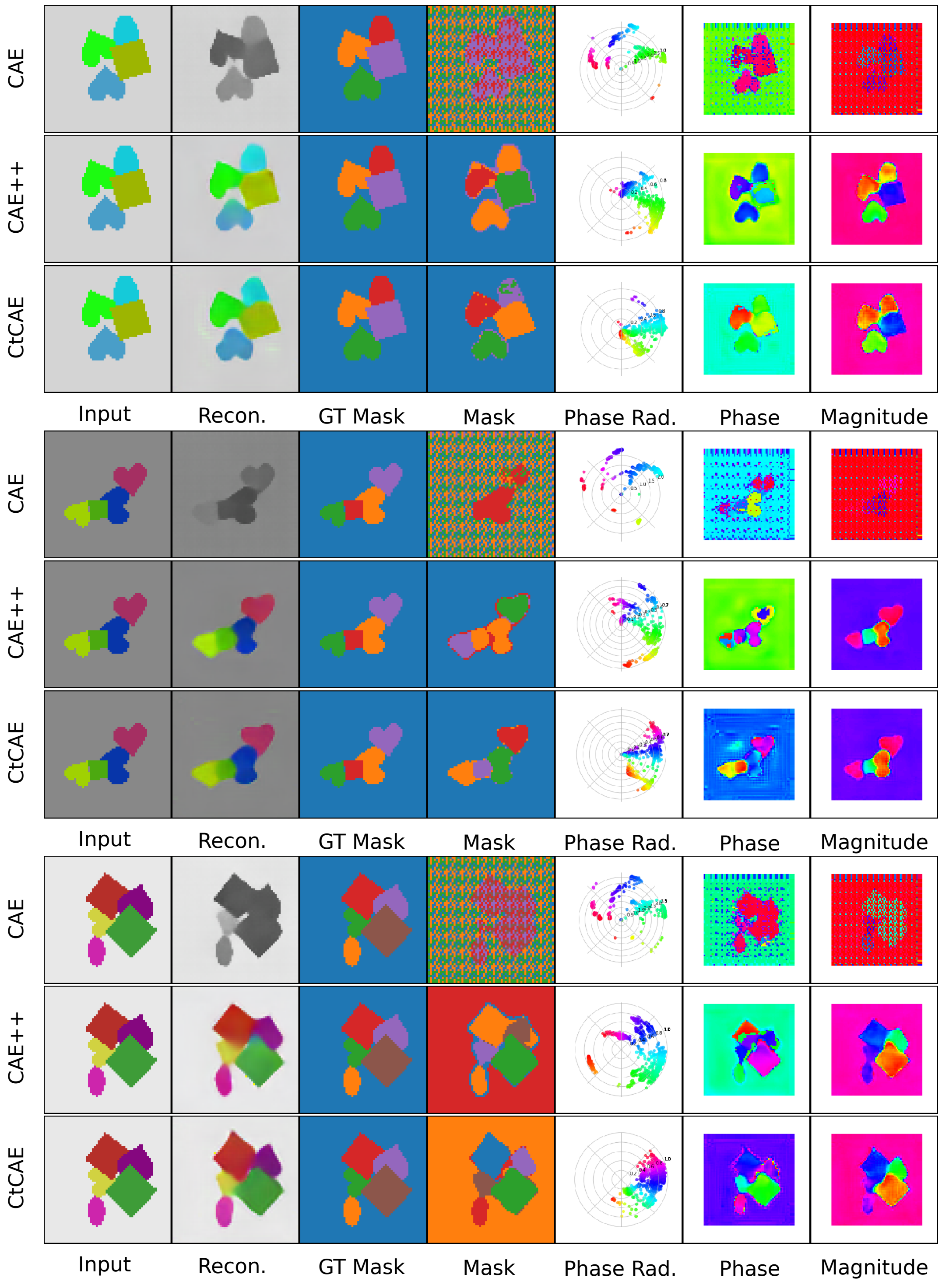


Figure C.7: Performance for CAE, CAE++ and CtCAE on images containing many objects on `dSprites`. CAE shows poor grouping performance, as noted previously. CtCAE has no issues grouping up to 5 objects in the same image, whereas CAE++ struggles, often grouping multiple objects (typically of same or similar colors) together into the same cluster.

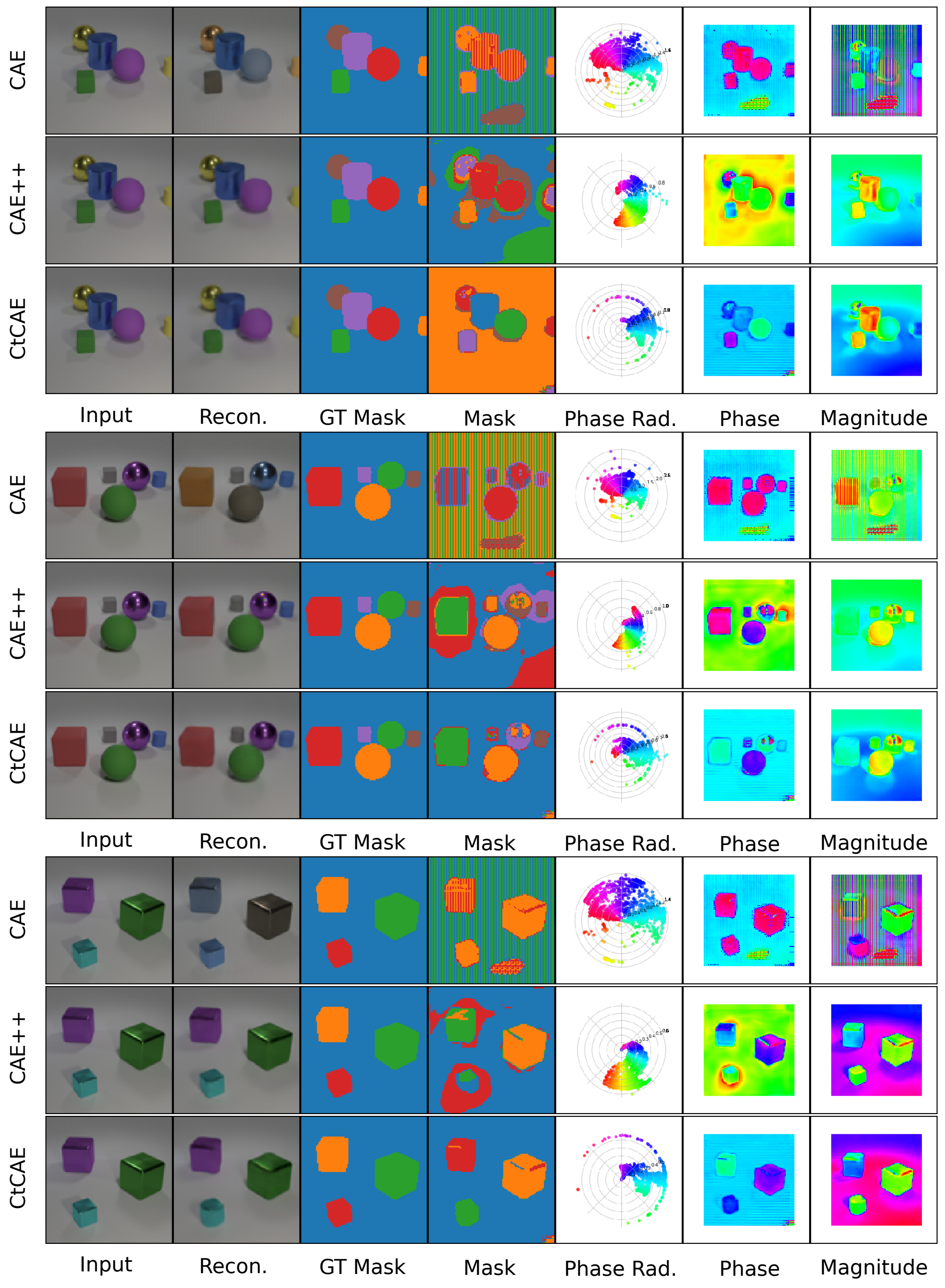


Figure C.8: Visualization on grouping performance for CAE, CAE++ and CtCAE on `CLEVR` dataset. We are particularly interested in the phase space spread (column 5) here. CAE has a very large spread in phase values (column 5), but almost random object discovery performance, often grouping all objects together. But if we see the actual phase values assigned to objects (column 6), we observe that CAE tends to assign very similar phase values (e.g. all object phases' are in shades of magenta seen in column 6, rows 1, 4, and 7) to regions belonging to different objects whereas the corresponding maps show better separation in CAE++ and much superior in CtCAE. This explains better grouping performance (column 4) shown by CAE++ and CtCAE over the vanilla CAE.

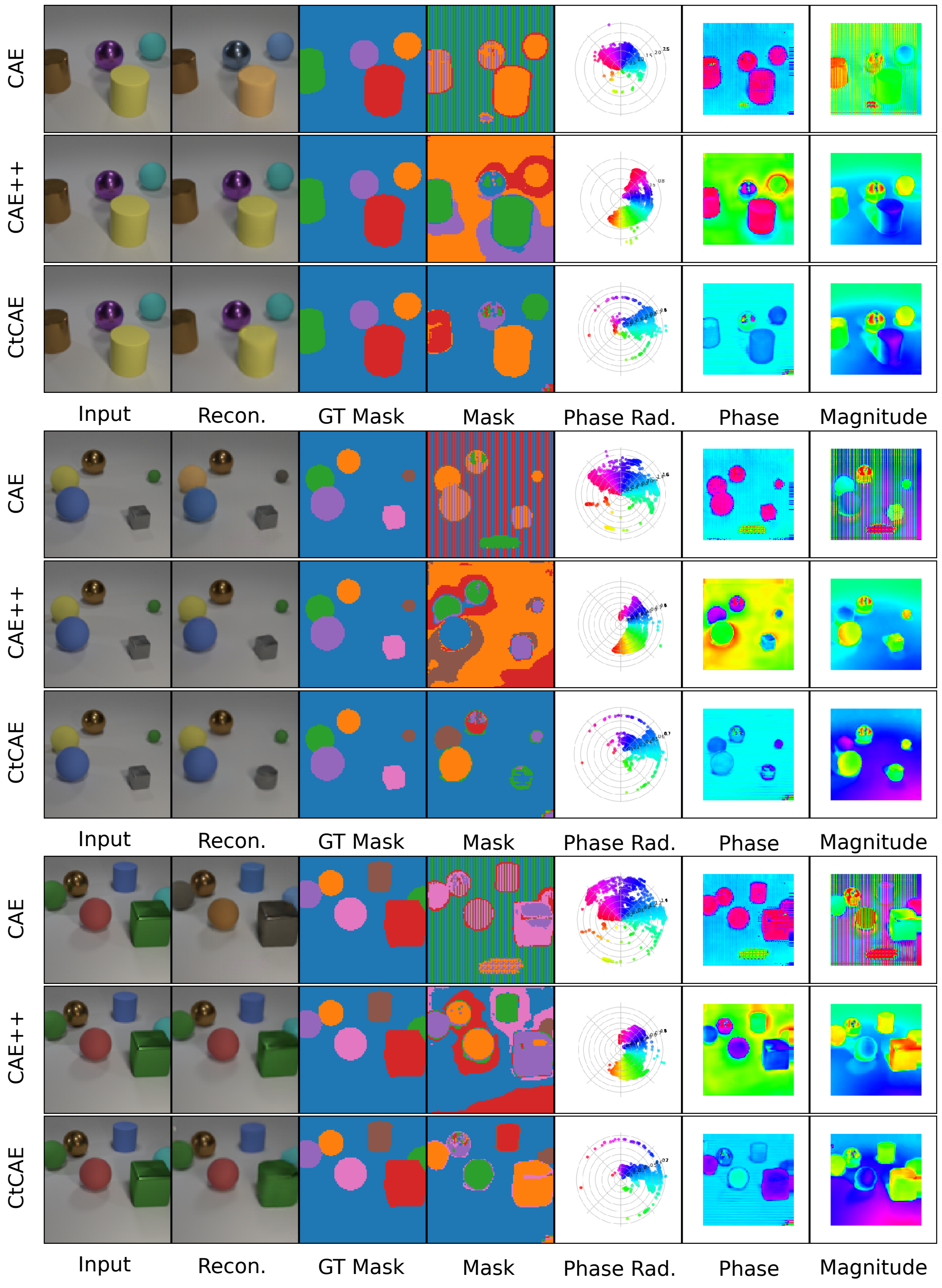


Figure C.9: Visualization on grouping performance for CAE, CAE + + and CtCAE on samples with 4, 5 and 6 objects from the CLEVR dataset. CAE consistently shows poor object discovery, often grouping all objects together. CAE + + improves grouping substantially over the CAE baseline, but also struggles in certain scenarios. For example, in the scene with 4 objects it groups two different objects together, whereas for scenes with 5 and 6 objects on two samples here we see that it wrongly groups two objects together. CtCAE improves on these grouping failures of CAE + +. CtCAE correctly discovers all objects in the scene with 4 and 5 objects, even though it makes an error grouping two objects together in the hardest example scene containing 6 objects.

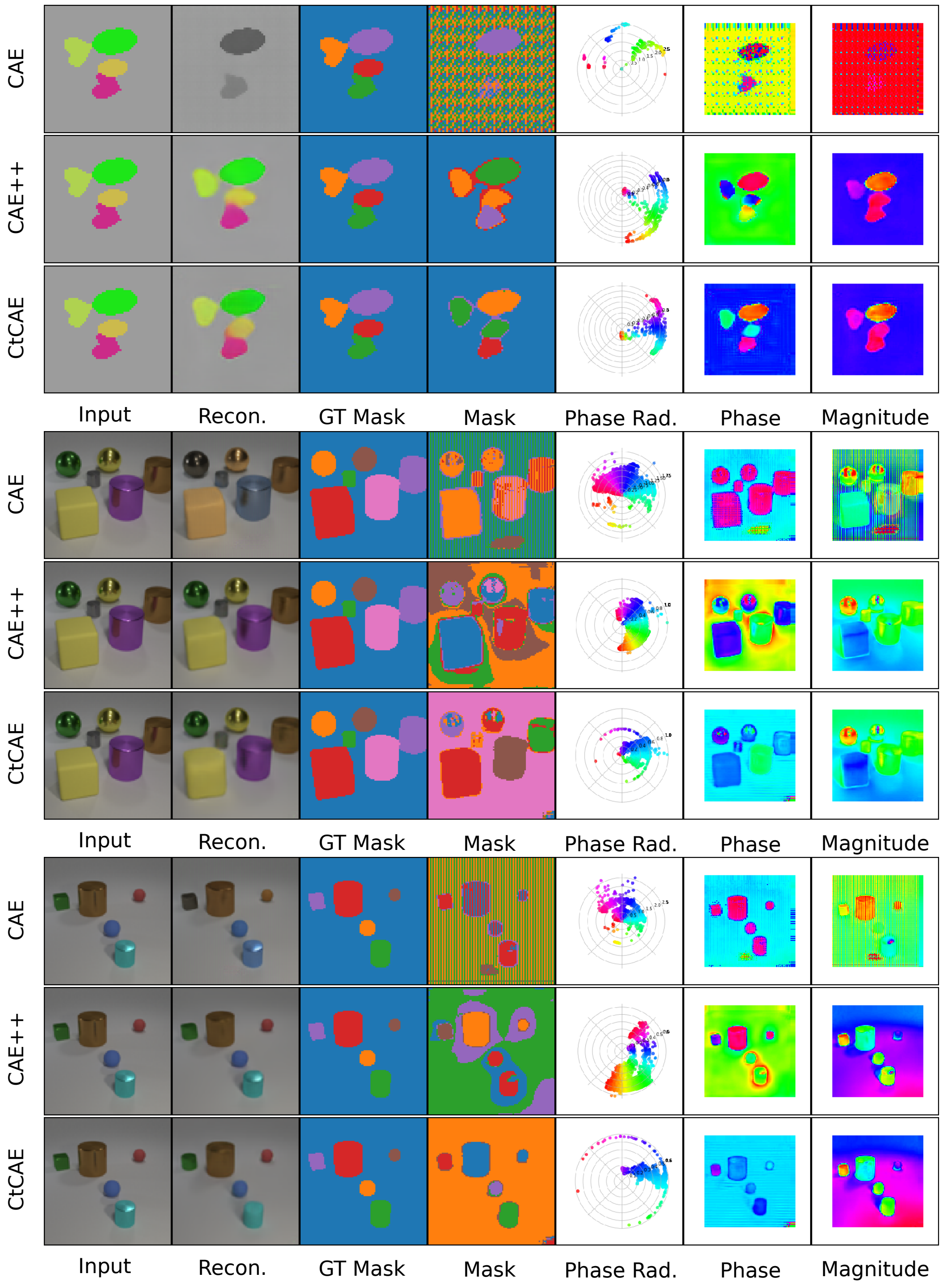


Figure C.10: One of the limitations (failure modes) we observe is that in some images containing many objects, with some of them having the same (or similar) colors, our CtCAE model groups two such objects into the same cluster. Even in this example, CtCAE still clearly outperforms CAE++, which in turn outperforms the CAE baseline.

Appendix D

Additional Details for Compositional Visual Reasoning with LLMs as Programmers

D.1 Ablations

Here we present hyperparameter ablations, namely over the code-generating LLM (`code-bison`) temperature and the open-vocabulary object detector (OWLv2) threshold.

Model	LLM temp.	RefCOCO (IoU)	RefCOCO+ (IoU)	NExT-QA (acc.)
ViperGPT API	0.0	41.4 $\pm$ 0.3	39.8 $\pm$ 0.0	36.0 $\pm$ 0.1
	0.4	46.9 $\pm$ 0.8	41.7 $\pm$ 0.4	53.1 $\pm$ 0.1
	0.8	46.9 $\pm$ 0.3	42.0 $\pm$ 0.7	53.2 $\pm$ 0.1
	1.0	46.2 $\pm$ 0.7	40.9 $\pm$ 0.6	50.3 $\pm$ 0.5
Abstract API	0.0	47.0 $\pm$ 0.1	41.9 $\pm$ 0.3	59.1 $\pm$ 0.0
	0.4	48.2 $\pm$ 0.1	**44.7** $\pm$ 0.2	**61.0** $\pm$ 0.4
	0.8	48.2 $\pm$ 0.0	43.0 $\pm$ 0.2	60.6 $\pm$ 0.6
	1.0	**48.8** $\pm$ 0.1	42.8 $\pm$ 0.3	59.9 $\pm$ 0.7

Table D.1: Code-generating LLM (`code-bison`) temperature hyperparameter ablations (with ACEs). The scores are reported as mean $\pm$ standard deviation across three random seeds.

In Table D.1, we report the scores for different `code-bison` LLM temperatures: 0, 0.4, 0.8 and 1.0. We found the deterministic case to underperform compared to the cases with a temperature higher than zero. This indicates that the solutions of which the models are most "confident" of are not necessarily always correct. On the other hand, when the temperature is too high, the model starts to hallucinate functions that do not exist in the API and the performance degrades. Early in our work, we settled on the `code-bison` LLM temperature of 0.4 and did not tune it further.

Model	OWLv2 thrs.	RefCOCO (IoU)	RefCOCO+ (IoU)
ViperGPT API	0.05	42.6 ± 0.3	41.9 ± 0.3
	0.10	46.9 ± 0.8	41.7 ± 0.4
	0.15	45.0 ± 0.5	38.2 ± 0.2
	0.20	33.3 ± 0.6	29.8 ± 0.2
Abstract API	0.05	42.8 ± 0.3	42.7 ± 0.7
	0.10	**48.2** ± 0.1	**44.7** ± 0.2
	0.15	47.1 ± 0.2	38.6 ± 0.1
	0.20	33.7 ± 0.4	30.1 ± 0.4

Table D.2: Open-vocabulary object detector (OWLv2) threshold hyperparameter ablations. The scores are reported as mean ± standard deviation across three random seeds.

Table D.2 shows the effect of using different thresholds for OWLv2 open vocabulary detector. This threshold controls the level of 'sensitivity' of the open vocabulary detector. If the threshold value is set too high, we will have fewer false positives, but also more false negatives. We perform this study on RefCOCO and RefCOCO+. On both datasets, the threshold of 0.1 achieves the best results, so by default we use this threshold in all our experiments.

In Table D.3 and Table D.4 we show results for RefCOCO and RefCOCO+ when using randomly sampled samples for the generation of ACEs. For comparison purposes, in the tables we also provide scores when not using any ACEs and with the default setting when generating ACEs with 16 manually selected few-shot examples. From the tables, we can see that randomly selected 100 or 50 samples perform similarly as when using 16 manual samples. With 16 random samples we observe a small drop in performance, though we still observe an improvement compared to the setting without any ACEs. In summary, this shows that the manual labor for generating ACEs can be removed altogether if

API	w/o ACE	16 manual	100 random	50 random	16 random
ViperGPT	41.7 ± 0.6	46.9 ± 0.8	47.9 ± 0.2	45.8 ± 0.4	41.5 ± 0.2
Abstract	44.4 ± 0.9	48.2 ± 0.1	49.1 ± 0.2	49.0 ± 0.2	46.9 ± 0.2

Table D.3: Results on RefCOCO with randomly sampled few-shot samples for generating ACEs. Shown are IoU scores for zero-shot (w/o ACE) setting, with the default setting that we used throughout the paper (16 manually sampled few-shot samples from the dataset (16 manual)) and with 100, 50 and 16 randomly-sampled samples (100, 50 and 16 random) from the dataset. The IoU scores are reported as mean ± standard deviation across three random seeds.

API	w/o ACE	16 manual	100 random	50 random	16 random
ViperGPT	36.7 ± 0.4	41.7 ± 0.4	41.3 ± 0.1	40.3 ± 0.3	36.3 ± 0.1
Abstract	38.2 ± 0.0	42.9 ± 0.2	42.8 ± 0.6	41.8 ± 0.0	40.2 ± 0.4

Table D.4: Results on RefCOCO+ with randomly sampled few-shot samples for generating ACEs. Shown are IoU scores for zero-shot (w/o ACE) setting, with the default setting that we used throughout the paper (16 manually sampled few-shot samples from the dataset (16 manual)) and with 100, 50 and 16 randomly-sampled samples (100, 50 and 16 random) from the dataset. The IoU scores are reported as mean ± standard deviation across three random seeds.

we already have some labeled examples.

D.2 Pretrained models

Here we specify the pretrained models we used, and compare them with the ones used in ViperGPT:

- Open-vocabulary object detector:
  - Ours: OWLv2 [Minderer et al., 2023].
  - ViperGPT: GLIP Li et al. [2022] from the official GitHub repository[1].
- Depth estimation model:
  - Ours: MiDaS [Ranftl et al., 2020] v2 "DPT_Small" from PyTorch hub[2].
  - ViperGPT: MiDaS [Ranftl et al., 2020] v2 "DPT_Large" version from the PyTorch hub[3].
- Vision-language captioning model:
  - Ours: PaLI-3 [Chen et al., 2023b].
  - ViperGPT: BLIP-2 [Li et al., 2023] from the official repository[4].
- CLIP-style image-text embedding model:
  - Ours: SigLiT [Zhai et al., 2023].
  - ViperGPT: X-VLM [Zeng et al., 2021] version finetuned for retrieval on MSCOCO from the official repository[5].
- Code-generating LLM:
  - Ours: `code-bison` accessible via the Google Cloud Vertex AI API [Google, 2023].
  - ViperGPT: Codex (`code-davinci-002`) via the official OpenAI Python API[6].

[1] https://github.com/microsoft/GLIP
[2] https://pytorch.org/hub/intelisl_midas_v2/
[3] https://pytorch.org/hub/intelisl_midas_v2/
[4] https://github.com/salesforce/LAVIS/tree/main/projects/blip2
[5] https://github.com/zengyan-97/X-VLM
[6] https://openai.com/blog/openai-api

- Answer selector (based on context information) LLM for multiple choice questions in NExT-QA:
  - Ours: `code-bison` accessible via the Google Cloud Vertex AI API [Google, 2023].
  - ViperGPT: GPT-3 via the official OpenAI Python API[7].

D.3 Self-debugging prompt

```
prompt += f"""

Previously, for the query:
# {query}
you've generated the following code:
{code}

The execution of the above code failed and returned the following
    error message:
{error}.

Given this, correct the above function, such that it executes
    correctly and solves the same query.

"""
```

[7]https://openai.com/blog/openai-api

D.4 Prompt listings

D.4.1 RefCOCO and GQA - ViperGPT API

```python
import math

class ImagePatch:
    """A Python class containing a crop of an image centered around a particular object, as well as relevant information.
    Attributes
    ----------
    cropped_image : array_like
        An array-like of the cropped image taken from the original image.
    left, lower, right, upper : int
        An int describing the position of the (left/lower/right/upper) border of the crop's bounding box in the original image.

    Methods
    -------
    find(object_name: str)->List[ImagePatch]
        Returns a list of new ImagePatch objects containing crops of the image centered around any objects found in the
        image matching the object_name.
    exists(object_name: str)->bool
        Returns True if the object specified by object_name is found in the image, and False otherwise.
    verify_property(property: str)->bool
        Returns True if the property is met, and False otherwise.
    compute_depth()->float
        Returns the median depth of the image crop.
    crop(left: int, lower: int, right: int, upper: int)->ImagePatch
        Returns a new ImagePatch object containing a crop of the image at the given coordinates.
    """

    def __init__(self, image, left: int = None, lower: int = None, right: int = None, upper: int = None):
        """Initializes an ImagePatch object by cropping the image at the given coordinates and stores the coordinates as
        attributes. If no coordinates are provided, the image is left unmodified, and the coordinates are set to the
        dimensions of the image.
        Parameters
        -------
        image : array_like
            An array-like of the original image.
        left, lower, right, upper : int
            An int describing the position of the (left/lower/right/upper) border of the crop's bounding box in the original image.
        """
        if left is None and right is None and upper is None and lower is None:
            self.cropped_image = image
            self.left = 0
            self.lower = 0
            self.right = image.shape[2]  # width
            self.upper = image.shape[1]  # height
        else:
            self.cropped_image = image[:, lower:upper, left:right]
            self.left = left
            self.upper = upper
            self.right = right
            self.lower = lower

        self.width = self.cropped_image.shape[2]
        self.height = self.cropped_image.shape[1]

        self.horizontal_center = (self.left + self.right) / 2
        self.vertical_center = (self.lower + self.upper) / 2

    def find(self, object_name: str) -> List[ImagePatch]:
        """Returns a list of ImagePatch objects matching object_name contained in the crop if any are found.
        Otherwise, returns an empty list.
        Parameters
        ----------
        object_name : str
            the name of the object to be found

        Returns
        -------
        List[ImagePatch]
            a list of ImagePatch objects matching object_name contained in the crop

        Examples
        --------
        >>> # return the foo
        >>> def execute_command(image) -> List[ImagePatch]:
```

```
        >>>     image_patch = ImagePatch(image)
        >>>     foo_patches = image_patch.find("foo")
        >>>     return foo_patches
        """
        return find_in_image(self.cropped_image, object_name)

    def exists(self, object_name: str) -> bool:
        """Returns True if the object specified by object_name is found in the image, and False otherwise.
        Parameters
        -------
        object_name : str
            A string describing the name of the object to be found in the image.

        Examples
        -------
        >>> # Are there both foos and garply bars in the photo?
        >>> def execute_command(image)->str:
        >>>     image_patch = ImagePatch(image)
        >>>     is_foo = image_patch.exists("foo")
        >>>     is_garply_bar = image_patch.exists("garply bar")
        >>>     return is_foo and is_garply_bar
        """
        return len(self.find(object_name)) > 0

    def verify_property(self, object_name: str, visual_property: str) -> bool:
        """Returns True if the object possesses the visual property, and False otherwise.
        Differs from 'exists' in that it presupposes the existence of the object specified by object_name, instead checking whether the object possesses the property.
        Parameters
        -------
        object_name : str
            A string describing the name of the object to be found in the image.
        visual_property : str
            A string describing the simple visual property (e.g., color, shape, material) to be checked.

        Examples
        -------
        >>> # Do the letters have blue color?
        >>> def execute_command(image) -> str:
```

```
        >>>     image_patch = ImagePatch(image)
        >>>     letters_patches = image_patch.find("letters")
        >>>     # Question assumes only one letter patch
        >>>     return letters_patches[0].verify_property("letters", "blue")
        """
        return verify_property(self.cropped_image, object_name, property)

    def compute_depth(self):
        """Returns the median depth of the image crop
        Parameters
        ----------
        Returns
        -------
        float
            the median depth of the image crop

        Examples
        --------
        >>> # the bar furthest away
        >>> def execute_command(image)->ImagePatch:
        >>>     image_patch = ImagePatch(image)
        >>>     bar_patches = image_patch.find("bar")
        >>>     bar_patches.sort(key=lambda bar: bar.compute_depth())
        >>>     return bar_patches[-1]
        """
        depth_map = compute_depth(self.cropped_image)
        return depth_map.median()

    def crop(self, left: int, lower: int, right: int, upper: int) -> ImagePatch:
        """Returns a new ImagePatch cropped from the current ImagePatch.
        Parameters
        -------
        left, lower, right, upper : int
            The (left/lower/right/upper)most pixel of the cropped image.
        -------
        """
        return ImagePatch(self.cropped_image, left, lower, right, upper)

    def overlaps_with(self, left, lower, right, upper):
        """Returns True if a crop with the given coordinates overlaps with this one,
        else False.
        Parameters
        ----------
        left, lower, right, upper : int
            the (left/lower/right/upper) border of the crop to be checked

        Returns
        -------
        bool
            True if a crop with the given coordinates overlaps with this one, else False

        Examples
        --------
        >>> # black foo on top of the qux
        >>> def execute_command(image) -> ImagePatch:
        >>>     image_patch = ImagePatch(image)
        >>>     qux_patches = image_patch.find("qux")
        >>>     qux_patch = qux_patches[0]
        >>>     foo_patches = image_patch.find("black foo")
        >>>     for foo in foo_patches:
        >>>         if foo.vertical_center > qux_patch.vertical_center
        >>>             return foo
        """
        return self.left <= right and self.right >= left and self.lower <= upper and self.upper >= lower


def best_image_match(list_patches: List[ImagePatch], content: List[str], return_index=False) -> Union[ImagePatch, int]:
    """Returns the patch most likely to contain the content.
    Parameters
    ----------
    list_patches : List[ImagePatch]
    content : List[str]
        the object of interest
    return_index : bool
        if True, returns the index of the patch most likely to contain the object

    Returns
    -------
    int
        Patch most likely to contain the object
    """
    return best_image_match(list_patches, content, return_index)


def distance(patch_a: ImagePatch, patch_b: ImagePatch) -> float:
    """
    Returns the distance between the edges of two ImagePatches. If
     the patches overlap, it returns a negative distance
    corresponding to the negative intersection over union.

    Parameters
    ----------
    patch_a : ImagePatch
    patch_b : ImagePatch

    Examples
    --------
    # Return the qux that is closest to the foo
    >>> def execute_command(image):
    >>>     image_patch = ImagePatch(image)
    >>>     qux_patches = image_patch.find('qux')
    >>>     foo_patches = image_patch.find('foo')
    >>>     foo_patch = foo_patches[0]
    >>>     qux_patches.sort(key=lambda x: distance(x, foo_patch))
    >>>     return qux_patches[0]
    """
    return distance(patch_a, patch_b)

INSERT_IN_CONTEXT_EXAMPLES_HERE

Write a function using Python and the ImagePatch class (above)
    that could be executed to provide an answer to the query.

Consider the following guidelines:
- Use base Python (comparison, sorting) for basic logical
    operations, left/right/up/down, math, etc.
- Make sure to always return an ImagePatch object.
- Make sure that for all possible control flows, the program
    always returns an ImagePatch object.

INSERT_PREVIOUS_CODE_AND_ERROR_HERE

# INSERT_QUERY_HERE
```

D.4.2 RefCOCO and GQA - Abstract API

```python
import math

class ImagePatch:
    """A Python class containing a crop of an image centered around a particular object, as well as relevant information.
    Attributes
    ----------
    cropped_image : array_like
        An array-like of the cropped image taken from the original image.
    left, lower, right, upper : int
        An int describing the position of the (left/lower/right/upper) border of the crop's bounding box in the original image.

    Methods
    -------
    find(object_name: str)->List[ImagePatch]
        Returns a list of new ImagePatch objects containing crops of the image centered around any objects found in the
        image matching the object_name.
    exists(object_name: str)->bool
        Returns True if the object specified by object_name is found in the image, and False otherwise.
    verify_property(property: str)->bool
        Returns True if the property is met, and False otherwise.
    compute_depth()->float
        Returns the median depth of the image crop.
    crop(left: int, lower: int, right: int, upper: int)->ImagePatch
        Returns a new ImagePatch object containing a crop of the image at the given coordinates.
    """

    def __init__(self, image, left: int = None, lower: int = None, right: int = None, upper: int = None):
        """Initializes an ImagePatch object by cropping the image at the given coordinates and stores the coordinates as
        attributes. If no coordinates are provided, the image is left unmodified, and the coordinates are set to the
        dimensions of the image.
        Parameters
        -------
        image : array_like
            An array-like of the original image.
        left, lower, right, upper : int
```

```
            An int describing the position of the (left/lower/right/upper) border of the crop's bounding box in the original image.
        """
        if left is None and right is None and upper is None and lower is None:
            self.cropped_image = image
            self.left = 0
            self.lower = 0
            self.right = image.shape[2]  # width
            self.upper = image.shape[1]  # height
        else:
            self.cropped_image = image[:, lower:upper, left:right]
            self.left = left
            self.upper = upper
            self.right = right
            self.lower = lower

        self.width = self.cropped_image.shape[2]
        self.height = self.cropped_image.shape[1]

        self.horizontal_center = (self.left + self.right) / 2
        self.vertical_center = (self.lower + self.upper) / 2

    def find(self, object_name: str) -> List[ImagePatch]:
        """Returns a list of ImagePatch objects matching object_name contained in the crop if any are found.
        Otherwise, returns an empty list.
        Parameters
        ----------
        object_name : str
            the name of the object to be found

        Returns
        -------
        List[ImagePatch]
            a list of ImagePatch objects matching object_name contained in the crop

        Examples
        --------
        >>> # return the foo
        >>> def execute_command(image) -> List[ImagePatch]:
        >>>     image_patch = ImagePatch(image)
        >>>     foo_patches = image_patch.find("foo")
        >>>     return foo_patches
        """
        return find_in_image(self.cropped_image, object_name)

    def exists(self, object_name: str) -> bool:
        """Returns True if the object specified by object_name is found in the image, and False otherwise.
        Parameters
        -------
        object_name : str
            A string describing the name of the object to be found in the image.

        Examples
        -------
        >>> # Are there both foos and garply bars in the photo?
        >>> def execute_command(image)->str:
        >>>     image_patch = ImagePatch(image)
        >>>     is_foo = image_patch.exists("foo")
        >>>     is_garply_bar = image_patch.exists("garply bar")
        >>>     return is_foo and is_garply_bar
        """
        return len(self.find(object_name)) > 0

    def verify_property(self, object_name: str, visual_property: str) -> bool:
        """Returns True if the object possesses the visual property, and False otherwise.
        Differs from 'exists' in that it presupposes the existence of the object specified by object_name, instead checking whether the object possesses the property.
        Parameters
        -------
        object_name : str
            A string describing the name of the object to be found in the image.
        visual_property : str
            A string describing the simple visual property (e.g., color, shape, material) to be checked.

        Examples
        -------
        >>> # Do the letters have blue color?
        >>> def execute_command(image) -> str:
        >>>     image_patch = ImagePatch(image)
        >>>     letters_patches = image_patch.find("letters")
        >>>     # Question assumes only one letter patch
        >>>     return letters_patches[0].verify_property("letters", "blue")
        """
        return verify_property(self.cropped_image, object_name, property)

    def compute_depth(self):
        """Returns the median depth of the image crop
        Parameters
        ----------
        Returns
        -------
        float
            the median depth of the image crop

        Examples
        --------
        >>> # the bar furthest away
        >>> def execute_command(image)->ImagePatch:
        >>>     image_patch = ImagePatch(image)
        >>>     bar_patches = image_patch.find("bar")
        >>>     bar_patches.sort(key=lambda bar: bar.compute_depth())
        >>>     return bar_patches[-1]
        """
        depth_map = compute_depth(self.cropped_image)
        return depth_map.median()

    def crop(self, left: int, lower: int, right: int, upper: int) -> ImagePatch:
        """Returns a new ImagePatch cropped from the current ImagePatch.
        Parameters
        -------
        left, lower, right, upper : int
            The (left/lower/right/upper)most pixel of the cropped image.
        -------
        """
        return ImagePatch(self.cropped_image, left, lower, right, upper)

    def overlaps_with(self, left, lower, right, upper):
        """Returns True if a crop with the given coordinates overlaps with this one,
        else False.
        Parameters
        ----------
        left, lower, right, upper : int
            the (left/lower/right/upper) border of the crop to be checked

        Returns
        -------
        bool
            True if a crop with the given coordinates overlaps with this one, else False

        Examples
        --------
        >>> # black foo on top of the qux
        >>> def execute_command(image) -> ImagePatch:
        >>>     image_patch = ImagePatch(image)
        >>>     qux_patches = image_patch.find("qux")
        >>>     qux_patch = qux_patches[0]
        >>>     foo_patches = image_patch.find("black foo")
        >>>     for foo in foo_patches:
        >>>         if foo.vertical_center > qux_patch.vertical_center
        >>>             return foo
        """
        return self.left <= right and self.right >= left and self.lower <= upper and self.upper >= lower


def best_image_match(list_patches: List[ImagePatch], content: List[str], return_index=False) -> Union[ImagePatch, int]:
    """Returns the patch most likely to contain the content.
    Parameters
    ----------
    list_patches : List[ImagePatch]
    content : List[str]
        the object of interest
    return_index : bool
        if True, returns the index of the patch most likely to contain the object

    Returns
    -------
    int
        Patch most likely to contain the object
    """
```

```python
    return best_image_match(list_patches, content, return_index)


def distance(patch_a: ImagePatch, patch_b: ImagePatch) -> float:
    """
    Returns the distance between the edges of two ImagePatches. If
     the patches overlap, it returns a negative distance
    corresponding to the negative intersection over union.

    Parameters
    ----------
    patch_a : ImagePatch
    patch_b : ImagePatch

    Examples
    --------
    # Return the qux that is closest to the foo
    >>> def execute_command(image):
    >>>     image_patch = ImagePatch(image)
    >>>     qux_patches = image_patch.find('qux')
    >>>     foo_patches = image_patch.find('foo')
    >>>     foo_patch = foo_patches[0]
    >>>     qux_patches.sort(key=lambda x: distance(x, foo_patch))
    >>>     return qux_patches[0]
    """
    return distance(patch_a, patch_b)

def get_patch_left_of(patch: ImagePatch) -> ImagePatch:
    left_patch = get_patch_left_of(patch)
    return left_patch

def get_patch_right_of(patch: ImagePatch) -> ImagePatch:
    right_patch = get_patch_right_of(patch)
    return right_patch

def get_patch_above_of(patch: ImagePatch) -> ImagePatch:
    above_patch = get_patch_above_of(patch)
    return above_patch

def get_patch_below_of(patch: ImagePatch) -> ImagePatch:
    below_patch = get_patch_below_of(patch)
    return below_patch

def get_patch_around_of(patch: ImagePatch) -> ImagePatch:
    around_patch = get_patch_around_of(patch)
    return around_patch
```

```

def sort_patches_left_to_right(list_patches: List[ImagePatch]) ->
    List[ImagePatch]:
    """
    Sorts patches according to their horizontal centers.

    Parameters
    ----------
    list_patches : List[ImagePatch]

    Examples
    --------
    # Right foo
    >>> def execute_command(image):
    >>>     image_patch = ImagePatch(image)
    >>>     foo_patches = image_patch.find('foo')
    >>>     foo_patches = sort_patches_left_to_right(foo_patches)
    >>>     right_foo_patch = foo_patches[-1]
    >>>     return right_foo_patch
    """
    return sort_patches_left_to_right(list_patches)


def sort_patches_bottom_to_top(list_patches: List[ImagePatch]) ->
    List[ImagePatch]:
    """
    Sorts patches according to their vertical centers.

    Parameters
    ----------
    list_patches : List[ImagePatch]

    Examples
    --------
    # Second bar from the top
    >>> def execute_command(image):
    >>>     image_patch = ImagePatch(image)
    >>>     bar_patches = image_patch.find('bar')
    >>>     bar_patches = sort_patches_bottom_to_top(bar_patches)
    >>>     second_topmost_bar_patch = bar_patches[-2]
    >>>     return second_topmost_bar_patch
    """
    return sort_patches_bottom_to_top(list_patches)


def sort_patches_front_to_back(list_patches: List[ImagePatch]) ->
```

```
    List[ImagePatch]:
    """
    Sorts patches according to how far from camera they are.

    Parameters
    ----------
    list_patches : List[ImagePatch]

    Examples
    --------
    # Person in the back
    >>> def execute_command(image):
    >>>     image_patch = ImagePatch(image)
    >>>     person_patches = image_patch.find('person')
    >>>     person_patches = sort_patches_front_to_back(
    person_patches)
    >>>     person_in_the_back = person_patches[-1]
    >>>     return person_in_the_back
    """
    return sort_patches_front_to_back(list_patches)


def get_middle_patch(list_patches: List[ImagePatch]) -> ImagePatch
    :
    """
    Returns the middle patch.

    Parameters
    ----------
    list_patches : List[ImagePatch]

    Examples
    --------
    # Middle ham
    >>> def execute_command(image):
    >>>     image_patch = ImagePatch(image)
    >>>     ham_patches = image_patch.find('ham')
    >>>     middle_ham_patch = get_middle_patch(ham_patches)
    >>>     return middle_ham_patch
    """
    return get_middle_patch(list_patches)


def get_patch_closest_to_anchor_object(list_patches: List[
    ImagePatch], anchor_object: ImagePatch) -> ImagePatch:
    """
    Returns the object from list_patches that is the closest to
    the anchor_object.

    Parameters
    ----------
    list_patches : List[ImagePatch]
    anchor_object : ImagePatch

    Examples
    --------
    # Foo next to bar
    >>> def execute_command(image):
    >>>     image_patch = ImagePatch(image)
    >>>     foo_patches = image_patch.find('foo')
    >>>     bar_patches = image_patch.find('bar')
    >>>     bar_patch = bar_patches[0]
    >>>     foo_next_to_bar_patch =
    get_patch_closest_to_anchor_object(foo_patches, bar_patch)
    >>>     return foo_next_to_bar_patch
    """
    return get_patch_closest_to_anchor_object(list_patches,
    anchor_object)


INSERT_IN_CONTEXT_EXAMPLES_HERE

Write a function using Python and the ImagePatch class (above)
    that could be executed to provide an answer to the query.

Consider the following guidelines:
- Use base Python (comparison, sorting) for basic logical
    operations, left/right/up/down, math, etc.
- Make sure to always return an ImagePatch object.
- Make sure that for all possible control flows, the program
    always returns an ImagePatch object.
- ImagePatch class uses left and right to denote horizontal edges.
- ImagePatch class uses bottom and top to denote vertical edges.

INSERT_PREVIOUS_CODE_AND_ERROR_HERE

# INSERT_QUERY_HERE
```

D.4.3 NExT-QA - ViperGPT API

```python
import math

class ImagePatch:
    """A Python class containing a crop of an image centered around a particular object, as well as relevant information.
    Attributes
    ----------
    cropped_image : array_like
        An array-like of the cropped image taken from the original image.
    left, lower, right, upper : int
        An int describing the position of the (left/lower/right/upper) border of the crop's bounding box in the original image.

    Methods
    -------
    find(object_name: str)->List[ImagePatch]
        Returns a list of new ImagePatch objects containing crops of the image centered around any objects found in the
        image matching the object_name.
    exists(object_name: str)->bool
        Returns True if the object specified by object_name is found in the image, and False otherwise.
    verify_property(property: str)->bool
        Returns True if the property is met, and False otherwise.
    best_text_match(option_list: List[str], prefix: str)->str
        Returns the string that best matches the image.
    simple_query(question: str=None)->str
        Returns the answer to a basic question asked about the image. If no question is provided, returns the answer to "What is this?".
    compute_depth()->float
        Returns the median depth of the image crop.
    crop(left: int, lower: int, right: int, upper: int)->ImagePatch
        Returns a new ImagePatch object containing a crop of the image at the given coordinates.
    """

    def __init__(self, image, left: int = None, lower: int = None, right: int = None, upper: int = None):
        """Initializes an ImagePatch object by cropping the image at the given coordinates and stores the coordinates as
        attributes. If no coordinates are provided, the image is left unmodified, and the coordinates are set to the
        dimensions of the image.
        Parameters
        -------
        image : array_like
            An array-like of the original image.
        left, lower, right, upper : int
            An int describing the position of the (left/lower/right/upper) border of the crop's bounding box in the original image.
        """
        if left is None and right is None and upper is None and lower is None:
            self.cropped_image = image
            self.left = 0
            self.lower = 0
            self.right = image.shape[2]  # width
            self.upper = image.shape[1]  # height
        else:
            self.cropped_image = image[:, lower:upper, left:right]
            self.left = left
            self.upper = upper
            self.right = right
            self.lower = lower

        self.width = self.cropped_image.shape[2]
        self.height = self.cropped_image.shape[1]

        self.horizontal_center = (self.left + self.right) / 2
        self.vertical_center = (self.lower + self.upper) / 2

    def find(self, object_name: str) -> List[ImagePatch]:
        """Returns a list of ImagePatch objects matching object_name contained in the crop if any are found.
        Otherwise, returns an empty list.
        Parameters
        ----------
        object_name : str
            the name of the object to be found

        Returns
        -------
        List[ImagePatch]
            a list of ImagePatch objects matching object_name contained in the crop

        Examples
        --------
        >>> # return the foo
        >>> def execute_command(image) -> List[ImagePatch]:
        >>>     image_patch = ImagePatch(image)
        >>>     foo_patches = image_patch.find("foo")
        >>>     return foo_patches
        """
        return find_in_image(self.cropped_image, object_name)

    def exists(self, object_name: str) -> bool:
        """Returns True if the object specified by object_name is found in the image, and False otherwise.
        Parameters
        -------
        object_name : str
            A string describing the name of the object to be found in the image.

        Examples
        -------
        >>> # Are there both foos and garply bars in the photo?
        >>> def execute_command(image)->str:
        >>>     image_patch = ImagePatch(image)
        >>>     is_foo = image_patch.exists("foo")
        >>>     is_garply_bar = image_patch.exists("garply bar")
        >>>     return bool_to_yesno(is_foo and is_garply_bar)
        """
        return len(self.find(object_name)) > 0

    def verify_property(self, object_name: str, visual_property: str) -> bool:
        """Returns True if the object possesses the visual property, and False otherwise.
        Differs from 'exists' in that it presupposes the existence of the object specified by object_name, instead checking whether the object possesses the property.
        Parameters
        -------
        object_name : str
            A string describing the name of the object to be found in the image.
        visual_property : str
            A string describing the simple visual property (e.g., color, shape, material) to be checked.

        Examples
        -------
        >>> # Do the letters have blue color?
        >>> def execute_command(image) -> str:
        >>>     image_patch = ImagePatch(image)
        >>>     letters_patches = image_patch.find("letters")
        >>>     # Question assumes only one letter patch
        >>>     return bool_to_yesno(letters_patches[0].verify_property("letters", "blue"))
        """
        return verify_property(self.cropped_image, object_name, property)

    def best_text_match(self, option_list: List[str]) -> str:
        """Returns the string that best matches the image.
        Parameters
        -------
        option_list : str
            A list with the names of the different options
        prefix : str
            A string with the prefixes to append to the options

        Examples
        -------
        >>> # Is the foo gold or white?
        >>> def execute_command(image)->str:
        >>>     image_patch = ImagePatch(image)
        >>>     foo_patches = image_patch.find("foo")
        >>>     # Question assumes one foo patch
        >>>     return foo_patches[0].best_text_match(["gold", "white"])
        """
        return best_text_match(self.cropped_image, option_list)

    def simple_query(self, question: str = None) -> str:
        """Returns the answer to a basic question asked about the image. If no question is provided, returns the answer
        to "What is this?". The questions are about basic perception, and are not meant to be used for complex reasoning
        or external knowledge.
        Parameters
        -------
        question : str
            A string describing the question to be asked.

        Examples
        -------

        >>> # Which kind of baz is not fredding?
        >>> def execute_command(image) -> str:
        >>>     image_patch = ImagePatch(image)
        >>>     baz_patches = image_patch.find("baz")
        >>>     for baz_patch in baz_patches:
        >>>         if not baz_patch.verify_property("baz", "
    fredding"):
        >>>             return baz_patch.simple_query("What is
    this baz?")

        >>> # What color is the foo?
        >>> def execute_command(image) -> str:
        >>>     image_patch = ImagePatch(image)
        >>>     foo_patches = image_patch.find("foo")
        >>>     foo_patch = foo_patches[0]
        >>>     return foo_patch.simple_query("What is the color
    ?")

        >>> # Is the second bar from the left quuxy?
        >>> def execute_command(image) -> str:
        >>>     image_patch = ImagePatch(image)
        >>>     bar_patches = image_patch.find("bar")
        >>>     bar_patches.sort(key=lambda x: x.horizontal_center
    )
        >>>     bar_patch = bar_patches[1]
        >>>     return bar_patch.simple_query("Is the bar quuxy?")
        """
        return simple_query(self.cropped_image, question)

    def compute_depth(self):
        """Returns the median depth of the image crop
        Parameters
        ----------
        Returns
        -------
        float
            the median depth of the image crop

        Examples
        --------
        >>> # the bar furthest away
        >>> def execute_command(image)->ImagePatch:
        >>>     image_patch = ImagePatch(image)
        >>>     bar_patches = image_patch.find("bar")
        >>>     bar_patches.sort(key=lambda bar: bar.compute_depth
())
        >>>     return bar_patches[-1]
        """
        depth_map = compute_depth(self.cropped_image)
        return depth_map.median()

    def crop(self, left: int, lower: int, right: int, upper: int)
-> ImagePatch:
        """Returns a new ImagePatch cropped from the current
ImagePatch.
        Parameters
        -------
        left, lower, right, upper : int
            The (left/lower/right/upper)most pixel of the cropped
image.
        -------
        """
        return ImagePatch(self.cropped_image, left, lower, right,
upper)

    def overlaps_with(self, left, lower, right, upper):
        """Returns True if a crop with the given coordinates
overlaps with this one,
        else False.
        Parameters
        ----------
        left, lower, right, upper : int
            the (left/lower/right/upper) border of the crop to be
checked

        Returns
        -------
        bool
            True if a crop with the given coordinates overlaps
with this one, else False

        Examples
        --------
        >>> # black foo on top of the qux
        >>> def execute_command(image) -> ImagePatch:
        >>>     image_patch = ImagePatch(image)
        >>>     qux_patches = image_patch.find("qux")
        >>>     qux_patch = qux_patches[0]
        >>>     foo_patches = image_patch.find("black foo")
        >>>     for foo in foo_patches:
        >>>         if foo.vertical_center > qux_patch.
```

```
vertical_center
        >>>             return foo
        """
        return self.left <= right and self.right >= left and self.lower <= upper and self.upper >= lower


class VideoSegment:
    """A Python class containing a set of frames represented as ImagePatch objects, as well as relevant information.
    Attributes
    ----------
    video : torch.Tensor
    A tensor of the original video.
    start : int
    An int describing the starting frame in this video segment with respect to the original video.
    end : int
    An int describing the ending frame in this video segment with respect to the original video.
    num_frames->int
    An int containing the number of frames in the video segment.

    Methods
    -------
    frame_iterator->Iterator[ImagePatch]
    trim(start, end)->VideoSegment
    Returns a new VideoSegment containing a trimmed version of the original video at the [start, end] segment.
    select_answer(info, question, options)->str
    Returns the answer to the question given the options and additional information.
    """

    def __init__(self, video: torch.Tensor, start: int = None, end: int = None, parent_start=0, queues=None):
        """Initializes a VideoSegment object by trimming the video at the given [start, end] times and stores the
        start and end times as attributes. If no times are provided, the video is left unmodified, and the times are
        set to the beginning and end of the video.

        Parameters
        -------
        video : torch.Tensor
        A tensor of the original video.
```

```python
        start : int
        An int describing the starting frame in this video segment with respect to the original video.
        end : int
        An int describing the ending frame in this video segment with respect to the original video.
        """

        if start is None and end is None:
            self.trimmed_video = video
            self.start = 0
            self.end = video.shape[0] # duration
        else:
            self.trimmed_video = video[start:end]
        if start is None:
            start = 0
        if end is None:
            end = video.shape[0]
        self.start = start + parent_start
        self.end = end + parent_start

        self.num_frames = self.trimmed_video.shape[0]

    def frame_iterator(self) -> Iterator[ImagePatch]:
        """Returns an iterator over the frames in the video segment."""
        for i in range(self.num_frames):
            yield ImagePatch(self.trimmed_video[i], self.start + i)

    def trim(self, start: Union[int, None] = None, end: Union[int, None] = None) -> VideoSegment:
        """Returns a new VideoSegment containing a trimmed version of the original video at the [start, end]
        segment.

        Parameters
        ----------
        start : Union[int, None]
        An int describing the starting frame in this video segment with respect to the original video.
        end : Union[int, None]
        An int describing the ending frame in this video segment with respect to the original video.

        Examples
        --------
        >>> # Return the second half of the video
        >>> def execute_command(video):
        >>> video_segment = VideoSegment(video)
        >>> video_second_half = video_segment.trim(video_segment.num_frames // 2, video_segment.num_frames)
        >>> return video_second_half
        """
        if start is not None:
            start = max(start, 0)
        if end is not None:
            end = min(end, self.num_frames)

        return VideoSegment(self.trimmed_video, start, end, self.start)

    def select_answer(self, info: dict, question: str, options: List[str]) -> str:
        return select_answer(self.trimmed_video, info, question, options)

    def __repr__(self):
        return "VideoSegment({}, {})".format(self.start, self.end)

    def simple_query(self, question) -> str:
        """Ask a simple question about the video.

        Examples
        --------
        # why does X happen?
        # possible_answers: ['answer1', 'answer2', 'answer3', 'answer4', 'answer5']
        def execute_command(video, question, possible_answers)->[str, dict]:
            # Create a video segment object
            video_segment = VideoSegment(video)
            # The question is simple, so just ask
            info = video_segment.simple_query("why does X happen?")
            # Choose the answer among given possible answers
            answer = select_answer(info, question, possible_answers)
            return answer
        """
        answer = simple_query(question)
        return answer
```

```python


def best_image_match(list_patches: List[ImagePatch], content: List
    [str], return_index=False) -> Union[ImagePatch, int]:
    """Returns the patch most likely to contain the content.
    Parameters
    ----------
    list_patches : List[ImagePatch]
    content : List[str]
        the object of interest
    return_index : bool
        if True, returns the index of the patch most likely to
    contain the object

    Returns
    -------
    int
        Patch most likely to contain the object
    """
    return best_image_match(list_patches, content, return_index)


def distance(patch_a: ImagePatch, patch_b: ImagePatch) -> float:
    """
    Returns the distance between the edges of two ImagePatches. If
     the patches overlap, it returns a negative distance
    corresponding to the negative intersection over union.

    Parameters
    ----------
    patch_a : ImagePatch
    patch_b : ImagePatch

    Examples
    --------
    # Return the qux that is closest to the foo
    >>> def execute_command(image):
    >>>     image_patch = ImagePatch(image)
    >>>     qux_patches = image_patch.find('qux')
    >>>     foo_patches = image_patch.find('foo')
    >>>     foo_patch = foo_patches[0]
    >>>     qux_patches.sort(key=lambda x: distance(x, foo_patch))
    >>>     return qux_patches[0]
    """
    return distance(patch_a, patch_b)

```

```

def bool_to_yesno(bool_answer: bool) -> str:
    return "yes" if bool_answer else "no"


def select_answer(info: str, question: question, possible_answers: str) -> str:
    """Given an info, question and possible answers, select the correct answer.

    Examples
    --------
    # what does man do at the end of the video
    # possible_answers: ['answer1', 'answer2', 'answer3', 'answer4', 'answer5']
    def execute_command(video, question, possible_answers)->[str, dict]:
        # Create a video segment object
        video_segment = VideoSegment(video)
        # Caption last frame of the video (end of video)
        last_frame = ImagePatch(video_segment, -1)
        last_caption = last_frame.simple_query("What is this?")
        men = last_frame.find("man")
        if len(men) == 0:
            men = [last_frame]
        man = men[0]
        man_action = man.simple_query("What is the man doing?")
        # Answer the question. Remember to create the info dictionary
        info = {
            "Caption of last frame": last_caption,
            "Man looks like he is doing": man_action
        }
        answer = video_segment.select_answer(info, question, possible_answers)
        return answer, info
    """


INSERT_IN_CONTEXT_EXAMPLES_HERE


Write a function using Python and the VideoSegment class (above) that could be executed to provide an answer to the query.

Consider the following guidelines:
```

```
- Use base Python (comparison, sorting) for basic logical
    operations, left/right/up/down, math, etc.

INSERT_PREVIOUS_CODE_AND_ERROR_HERE

# INSERT_QUERY_HERE
```

D.4.4 NExT-QA - Abstract API

```python
import math


class VideoSegment:
    """A Python class containing a video, as well as relevant information.
    Attributes
    ----------
    video : np.ndarray
    A tensor of the original video.
    start : int
    An int describing the starting frame in this video segment with respect to the original video.
    end : int
    An int describing the ending frame in this video segment with respect to the original video.
    num_frames->int
    An int containing the number of frames in the video segment.

    Methods
    -------
    trim(start, end)->VideoSegment
    Returns a new VideoSegment containing a trimmed version of the original video at the [start, end] segment.
    select_answer(info, question, possible_answers)->str
    Returns the answer to the question given the possible answers and additional information.
    """

    def __init__(self, video: np.ndarray, start: int = None, end: int = None, parent_start=0, queues=None):
        """Initializes a VideoSegment object by trimming the video at the given [start, end] times and stores the
        start and end times as attributes. If no times are provided, the video is left unmodified, and the times are
        set to the beginning and end of the video.

        Parameters
        -------
        video : np.ndarray
        A tensor of the original video.
        start : int
        An int describing the starting frame in this video segment with respect to the original video.
        end : int
        An int describing the ending frame in this video segment with respect to the original video.
        """

        if start is None and end is None:
            self.trimmed_video = video
            self.start = 0
            self.end = video.shape[0] # duration
        else:
            self.trimmed_video = video[start:end]
        if start is None:
            start = 0
        if end is None:
            end = video.shape[0]
        self.start = start + parent_start
        self.end = end + parent_start

        self.num_frames = self.trimmed_video.shape[0]

    def trim(self, start: Union[int, None] = None, end: Union[int, None] = None) -> VideoSegment:
        """Returns a new VideoSegment containing a trimmed version of the original video at the [start, end]
        segment.

        Parameters
        ----------
        start : Union[int, None]
        An int describing the starting frame in this video segment with respect to the original video.
        end : Union[int, None]
        An int describing the ending frame in this video segment with respect to the original video.

        Examples
        --------
        >>> # Return the second half of the video
        >>> def execute_command(video):
        >>>     video_segment = VideoSegment(video)
        >>>     video_second_half = video_segment.trim(video_segment.num_frames // 2, video_segment.num_frames)
        >>>     return video_second_half
        """
        if start is not None:
            start = max(start, 0)
        if end is not None:
            end = min(end, self.num_frames)

        return VideoSegment(self.trimmed_video, start, end, self.start)

    def get_video_segment_of_event(self, event) -> VideoSegment:
        return get_video_segment_of_event(event)

    def get_video_segment_before_event(self, event) -> VideoSegment:
        return get_video_segment_before_event(event)

    def get_video_segment_after_event(self, event) -> VideoSegment:
        return get_video_segment_after_event(event)

    def caption_video(self, question) -> str:
        return caption_video(question)

    def simple_query(self, question) -> str:
        """Ask a simple question about the video.

        Examples
        --------
        # why does X happen?
        # possible_answers: ['answer1', 'answer2', 'answer3', 'answer4', 'answer5']
        def execute_command(video, question, possible_answers)->[str, dict]:
            # Create a video segment object
            video_segment = VideoSegment(video)
            # The question is simple, so just ask
            info = video_segment.simple_query("why does X happen?")
            # Choose the answer among given possible answers
            answer = select_answer(info, question, possible_answers)
            return answer
        """
        answer = simple_query(question)
        return answer


def select_answer(info: str, question: question, possible_answers: str) -> str:
    """Given an info, question and possible answers, select the
    correct answer.

    Examples
    --------
    # what does person A do after event X?
    # possible_answers: ['answer1', 'answer2', 'answer3', 'answer4
    ', 'answer5']
    def execute_command(video, question, possible_answers)->[str,
    dict]:
        # Create a video segment object
        video_segment = VideoSegment(video)
        # Get video segment after event X
        video_segment_after = video_segment.
    get_video_segment_after_event("event X")
        # Ask what the person A is doing
        info = video_segment_after.caption_video("What is person A
     doing?")
        # Choose the answer among given possible answers
        answer = select_answer(info, question, possible_answers)
        return answer
    """


INSERT_IN_CONTEXT_EXAMPLES_HERE

Write a function using Python and the VideoSegment class (above)
    that could be executed to provide an answer to the query.

Consider the following guidelines:
- Use base Python (comparison, sorting) for basic logical
    operations, left/right/up/down, math, etc.
- The input to your program is a video, question and possible
    answers.
- Always start your function by creating a `video_segment =
    VideoSegment(video)` object.

INSERT_PREVIOUS_CODE_AND_ERROR_HERE

# INSERT_QUERY_HERE
```

Bibliography

David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *arXiv preprint arXiv:2204.14198*, 2022.

S-I Amari. Learning patterns and pattern sequences by self-organizing nets of threshold elements. *IEEE Transactions on computers*, 100(11):1197–1206, 1972.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 39–48, 2016.

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

Fred Attneave. Multistability in perception. *Scientific American*, 225(6):62–71, 1971.

James Atwood and Don Towsley. Diffusion-convolutional neural networks. *Advances in neural information processing systems*, 29, 2016.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H. Campbell, and Sergey Levine. Stochastic Variational Video Prediction. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rk49Mg-CW`.

Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron Courville. Systematic generalization: what is required and can it be learned? *arXiv preprint arXiv:1811.12889*, 2018.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Renee Baillargeon, Elizabeth S Spelke, and Stanley Wasserman. Object permanence in five-month-old infants. *Cognition*, 20(3):191–208, 1985.

Bowen Baker, Ilge Akkaya, Peter Zhokhov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video Pretraining (vpt): Learning to Act by Watching Unlabeled Online Videos, 2022. URL `https://arxiv.org/abs/2206.11795`.

Federico Baldassarre and Hossein Azizpour. Towards self-supervised learning of global and object-centric representations. *arXiv preprint arXiv:2203.05997*, 2022.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michal Podstawski, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. *arXiv preprint arXiv:2308.09687*, 2023.

James Betker, Gabriel Goh, Li Jing, Tim Brooks, Jianfeng Wang, Linjie Li, Long Ouyang, Juntang Zhuang, Joyce Lee, Yufei Guo, Wesam Manassra, Prafulla Dhariwal, Casey Chu, Yunxin Jiao, and Aditya Ramesh. Improving Image Generation with Better Captions. 2023. URL `https://cdn.openai.com/papers/dall-e-3.pdf`.

Irving Biederman. Recognition-by-components: a theory of human image understanding. *Psychological review*, 94(2):115, 1987.

Ondrej Biza, Robert Platt, Jan-Willem van de Meent, Lawson LS Wong, and Thomas Kipf. Binding actions to objects in world models. *arXiv preprint arXiv:2204.13022*, 2022.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Shyamal Buch, Cristóbal Eyzaguirre, Adrien Gaidon, Jiajun Wu, Li Fei Fei, and Juan Carlos Niebles. Revisiting the" video" in video-language understanding. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2917–2927, 2022.

Emanuele Bugliarello, Laurent Sartran, Aishwarya Agrawal, Lisa Anne Hendricks, and Aida Nematzadeh. Measuring Progress in Fine-grained Vision-and-language Understanding. *arXiv preprint arXiv:2305.07558*, 2023.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

Christopher P Burgess, Loic Matthey, Nicholas Watters, Rishabh Kabra, Irina Higgins, Matt Botvinick, and Alexander Lerchner. Monet: Unsupervised scene decomposition and representation. *arXiv preprint arXiv:1901.11390*, 2019.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021.

Wilka Carvalho, Andrew Lampinen, Kyriacos Nikiforou, Felix Hill, and Murray Shanahan. Feature-attending Recurrent Modules for Generalization in Reinforcement Learning. *arXiv preprint arXiv:2112.08369*, 2021.

Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G Bellemare. Dopamine: A research framework for deep reinforcement learning. *arXiv preprint arXiv:1812.06110*, 2018.

Stephanie CY Chan, Andrew K Lampinen, Pierre H Richemond, and Felix Hill. Zipfian environments for Reinforcement Learning. *arXiv preprint arXiv:2203.08222*, 2022.

Michael Chang, Tomer Ullman, Antonio Torralba, and Joshua Tenenbaum. A Compositional Object-based Approach to Learning Physical Dynamics. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=Bkab5dqxe`.

Angelica Chen, Jérémy Scheurer, Tomasz Korbak, Jon Ander Campos, Jun Shern Chan, Samuel R Bowman, Kyunghyun Cho, and Ethan Perez. Improving code generation by training with natural language feedback. *arXiv preprint arXiv:2303.16749*, 2023.

Jiuhai Chen, Lichang Chen, Chen Zhu, and Tianyi Zhou. How Many Demonstrations Do You Need for In-context Learning?, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022a.

Xi Chen, Xiao Wang, Soravit Changpinyo, AJ Piergiovanni, Piotr Padlewski, Daniel Salz, Sebastian Goodman, Adam Grycner, Basil Mustafa, Lucas Beyer, et al. Pali: A jointly-scaled multilingual language-image model. *arXiv preprint arXiv:2209.06794*, 2022b.

Xi Chen, Josip Djolonga, Piotr Padlewski, Basil Mustafa, Soravit Changpinyo, Jialin Wu, Carlos Riquelme Ruiz, Sebastian Goodman, Xiao Wang, Yi Tay, et al. Pali-x: On Scaling up a Multilingual Vision and Language Model. *arXiv preprint arXiv:2305.18565*, 2023a.

Xi Chen, Xiao Wang, Lucas Beyer, Alexander Kolesnikov, Jialin Wu, Paul Voigtlaender, Basil Mustafa, Sebastian Goodman, Ibrahim Alabdulmohsin, Piotr Padlewski, et al. Pali-3 Vision Language Models: Smaller, Faster, Stronger. *arXiv preprint arXiv:2310.09199*, 2023b.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023c.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using Rnn Encoder–decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

N. Chomsky. *Aspects of the Theory of Syntax*. The MIT Press. MIT Press, 1969. ISBN 9780262260503. URL `https://books.google.ch/books?id=u0ksbFqagU8C`.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Dan Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3642–3649. IEEE, 2012.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Eric Crawford and Joelle Pineau. Spatially invariant unsupervised object detection with convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3412–3420, 2019.

Antonia Creswell, Rishabh Kabra, Chris Burgess, and Murray Shanahan. Unsupervised object-based transition models for 3d partially observable environments. *Advances in Neural Information Processing Systems*, 34, 2021.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2018.

Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning*, pages 7480–7512. PMLR, 2023.

Fei Deng, Zhuo Zhi, and Sungjin Ahn. Generative Hierarchical Models for Parts, Objects, and Scenes. *arXiv preprint arXiv:1910.09119*, 2019.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Emily Denton and Rob Fergus. Stochastic Video Generation with a Learned Prior. In *International Conference on Machine Learning*, pages 1174–1183, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

David Ding, Felix Hill, Adam Santoro, Malcolm Reynolds, and Matthew Botvinick. Attention over Learned Object Embeddings Enables Complex Visual Reasoning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

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. *arXiv preprint arXiv:2010.11929*, 2020.

Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

Gamaleldin F. Elsayed, Aravindh Mahendran, Sjoerd van Steenkiste, Klaus Greff, Michael C. Mozer, and Thomas Kipf. SAVi++: Towards end-to-end object-centric learning from real-world videos. In *Advances in Neural Information Processing Systems*, 2022.

Patrick Emami, Pan He, Sanjay Ranka, and Anand Rangarajan. Efficient iterative amortized inference for learning symmetric and disentangled multi-object representations. In *International Conference on Machine Learning*, pages 2970–2981. PMLR, 2021.

Andreas K Engel, Peter König, Andreas K Kreiter, Thomas B Schillen, and Wolf Singer. Temporal coding in the visual cortex: new vistas on integration in the nervous system. *Trends in neurosciences*, 15(6):218–226, 1992.

Martin Engelcke, Adam R. Kosiorek, Oiwi Parker Jones, and Ingmar Posner. Genesis: Generative Scene Inference and Sampling with Object-centric Latent Representations. In *International Conference on Learning Representations*, 2020.

SM Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Geoffrey E Hinton, et al. Attend, infer, repeat: Fast scene understanding with generative models. In *Advances in Neural Information Processing Systems*, pages 3225–3233, 2016.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential Self-improvement Via Prompt Evolution. *arXiv preprint arXiv:2309.16797*, 2023.

Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pages 64–72, 2016.

Gottlob Frege. über Sinn Und Bedeutung. *Zeitschrift für Philosophie Und Philosophische Kritik*, 100(1):25–50, 1892a.

Gottlob Frege. Letter to Jourdain. *Meaning and Reference*, pages 43–45, 1892b.

Gottlob Frege. Gedankenfüge. *Beiträge zur Philosophie des deutschen Idealismus III*, pages 36–51, 1923.

Pascal Fries, Danko Nikolić, and Wolf Singer. The gamma cycle. *Trends in neurosciences*, 30(7):309–316, 2007.

Zhe Gan, Linjie Li, Chunyuan Li, Lijuan Wang, Zicheng Liu, Jianfeng Gao, et al. Vision-language pre-training: Basics, recent advances, and future trends. *Foundations and Trends® in Computer Graphics and Vision*, 14(3–4): 163–352, 2022.

Deep Ganguli, Amanda Askell, Nicholas Schiefer, Thomas Liao, Kamilė Lukošiūtė, Anna Chen, Anna Goldie, Azalia Mirhoseini, Catherine Olsson, Danny Hernandez, et al. The capacity for moral self-correction in large language models. *arXiv preprint arXiv:2302.07459*, 2023.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR, 2023.

Tianyu Gao, Adam Fisch, and Danqi Chen. Making pre-trained language models better few-shot learners. *arXiv preprint arXiv:2012.15723*, 2020.

Marta Garnelo and Murray Shanahan. Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. *Current Opinion in Behavioral Sciences*, 29:17 – 23, 2019. ISSN 2352-1546. doi: https://doi.org/10.1016/j.cobeha.2018.12.010. URL `http://www.sciencedirect.com/science/article/pii/S2352154618301943`. SI: 29: Artificial Intelligence (2019).

G. M. Ghose and J. H. R. Maunsell. Spatial Summation Can Explain the Attentional Modulation of Neuronal Responses to Multiple Stimuli in Area V4. *Journal of Neuroscience*, 28(19):5115–5126, May 2008. ISSN 0270-6474, 1529-2401. doi: 10.1523/JNEUROSCI.0138-08.2008.

Geoffrey M. Ghose and John H. R. Maunsell. Attentional modulation in visual cortex depends on task timing. *Nature*, 419(6907):616–620, Oct 2002. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature01057.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

Christoph Goller and Andreas Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 1, pages 347–352. IEEE, 1996.

Google. Google Cloud Vertex Ai Api [code-bison], Available at: `https://cloud.google.com/vertex-ai/docs/generative-ai/model-reference/code-generation`. 2023.

Anand Gopalakrishnan, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Unsupervised Object Keypoint Learning using Local Spatial Predictability. In *International Conference on Learning Representations*, 2021.

Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*, 2019.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471, 2016.

Klaus Greff, Rupesh Kumar Srivastava, and Jürgen Schmidhuber. Binding via reconstruction clustering. *arXiv preprint arXiv:1511.06418*, 2015.

Klaus Greff, Antti Rasmus, Mathias Berglund, Tele Hao, Harri Valpola, and Jürgen Schmidhuber. Tagger: Deep unsupervised perceptual grouping. In *Advances in Neural Information Processing Systems*, pages 4484–4492, 2016.

Klaus Greff, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Neural expectation maximization. In *Advances in Neural Information Processing Systems*, pages 6691–6701, 2017.

Klaus Greff, Raphaël Lopez Kaufman, Rishabh Kabra, Nick Watters, Christopher Burgess, Daniel Zoran, Loic Matthey, Matthew Botvinick, and Alexander Lerchner. Multi-object Representation Learning with Iterative Variational Inference. In *International Conference on Machine Learning*, pages 2424–2433, 2019.

Klaus Greff, Sjoerd van Steenkiste, and Jürgen Schmidhuber. On the Binding Problem in Artificial Neural Networks. *arXiv preprint arXiv:2012.05208*, 2020.

Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J Fleet, Dan Gnanapragasam, Florian Golemo, Charles Herrmann, Thomas Kipf, Abhijit Kundu, Dmitry Lagun, Issam Laradji, Hsueh Ti (Derek) Liu, Henning Meyer, Yishu Miao, Derek Nowrouzezahrai, Cengiz Oztireli, Etienne Pot, Noha Radwan, Daniel Rebain, Sara Sabour, Mehdi S. M. Sajjadi, Matan Sela, Vincent Sitzmann, Austin Stone, Deqing Sun, Suhani Vora, Ziyu Wang, Tianhao Wu, Kwang Moo Yi, Fangcheng Zhong, and Andrea Tagliasacchi. Kubric: a scalable dataset generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Jake Grigsby and Yanjun Qi. Measuring visual generalization in continuous control from pixels. *arXiv preprint arXiv:2010.06740*, 2020.

Stephen Grossberg. Some networks that can learn, remember, and reproduce any number of complicated space-time patterns. *Indiana University Journal of Mathematics and Mechanics*, 19:53-91(1), 1969.

Stephen Grossberg. Adaptive Resonance Theory: How a brain learns to consciously attend, learn, and recognize a changing world. *Neural networks*, 37:1–47, 2013.

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962, 2023.

William H Guss, Cayden Codel, Katja Hofmann, Brandon Houghton, Noboru Kuno, Stephanie Milani, Sharada Mohanty, Diego Perez Liebana, Ruslan Salakhutdinov, Nicholay Topin, et al. Neurips 2019 competition: the Minerl competition on sample efficient reinforcement learning using human priors. *arXiv preprint arXiv:1904.10079*, 2019.

William H Guss, Mario Ynocente Castro, Sam Devlin, Brandon Houghton, Noboru Sean Kuno, Crissman Loomis, Stephanie Milani, Sharada Mohanty, Keisuke Nakata, Ruslan Salakhutdinov, et al. The minerl 2020 competition on sample efficient reinforcement learning using human priors. *arXiv preprint arXiv:2101.11071*, 2021.

Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 297–304, 2010.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, pages 2450–2462, 2018.

Danijar Hafner. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*, 2021.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Jessica B Hamrick, Kelsey R Allen, Victor Bapst, Tina Zhu, Kevin R McKee, Joshua B Tenenbaum, and Peter W Battaglia. Relational inductive bias for physical construction in humans and machines. *arXiv preprint arXiv:1806.01203*, 2018.

Nicklas Hansen and Xiaolong Wang. Generalization in reinforcement learning by soft data augmentation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13611–13617. IEEE, 2021.

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

Olivier J Hénaff, Skanda Koppula, Evan Shelhamer, Daniel Zoran, Andrew Jaegle, Andrew Zisserman, João Carreira, and Relja Arandjelović. Object discovery and representation networks. In *European Conference on Computer Vision*, pages 123–143, 2022.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Lisa Anne Hendricks and Aida Nematzadeh. Probing image-language transformers for verb understanding. *arXiv preprint arXiv:2106.09141*, 2021.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

Geoffrey Hinton. How to represent part-whole hierarchies in a neural network. *Neural Computation*, 35(3):413–452, 2023.

Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with EM routing. In *International Conference on Learning Representations*, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

Yedid Hoshen. Vain: Attentional multi-agent predictive modeling. In *Advances in Neural Information Processing Systems*, pages 2701–2711, 2017.

Cheng-Yu Hsieh, Jieyu Zhang, Zixian Ma, Aniruddha Kembhavi, and Ranjay Krishna. Sugarcrepe: Fixing Hackable Benchmarks for Vision-language Compositionality. *arXiv preprint arXiv:2306.14610*, 2023.

Jun-Ting Hsieh, Bingbin Liu, De-An Huang, Li F Fei Fei, and Juan Carlos Niebles. Learning to decompose and disentangle representations for video prediction. In *Advances in Neural Information Processing Systems*, pages 517–526, 2018.

Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 804–813, 2017.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*, 2023.

David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2:193–218, 1985.

Drew A Hudson and Christopher D Manning. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6700–6709, 2019.

John E Hummel and Keith J Holyoak. Distributing structure over time. *Behavioral and Brain Sciences*, 16(3):464–464, 1993.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Francis R. Bach and David M. Blei, editors, *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015. URL `http://dblp.uni-trier.de/db/conf/icml/icml2015.html#IoffeS15`.

Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3. 6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE transactions on pattern analysis and machine intelligence*, 36(7):1325–1339, 2013.

Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. Going beyond linear transformers with recurrent fast weight programmers. *Advances in Neural Information Processing Systems*, 34, 2021.

Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. A Modern Self-referential Weight Matrix That Learns to Modify Itself. *arXiv preprint arXiv:2202.05780*, 2022.

Michael Iuzzolino, Yoram Singer, and Michael C Mozer. Convolutional bipartite attractor networks. *arXiv preprint arXiv:1906.03504*, 2019.

Max Jaderberg, Karen Simonyan, Andrew Zisserman, and koray kavukcuoglu. Spatial Transformer Networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2017–2025. Curran Associates, Inc., 2015. URL `http://papers.nips.cc/paper/5854-spatial-transformer-networks.pdf`.

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. *arXiv preprint arXiv:2107.14795*, 2021a.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning*, pages 4651–4664. PMLR, 2021b.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=rkE3y85ee`.

John G.R Jefferys, Roger D Traub, and Miles A Whittington. Neuronal networks for induced '40 Hz' rhythms. *Trends in Neurosciences*, 19(5): 202–208, 1996.

Jindong Jiang*, Sepehr Janghorbani*, Gerard De Melo, and Sungjin Ahn. Scalor: Generative World Models with Scalable Object Representations. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=SJxrKgStDH`.

Woojeong Jin, Yu Cheng, Yelong Shen, Weizhu Chen, and Xiang Ren. A good prompt is worth millions of parameters: Low-resource prompt-based learning for vision-language models. *arXiv preprint arXiv:2110.08484*, 2021.

Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Judy Hoffman, Li Fei Fei, C Lawrence Zitnick, and Ross Girshick. Inferring and executing programs for visual reasoning. In *Proceedings of the IEEE international conference on computer vision*, pages 2989–2998, 2017.

Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The Malmo Platform for Artificial Intelligence Experimentation. In *IJCAI*, pages 4246–4247. Citeseer, 2016.

Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, et al. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2018.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with Alphafold. *Nature*, 596(7873):583–589, 2021.

Niels Justesen and Sebastian Risi. Automated curriculum learning by rewarding temporally rare events. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2018.

Niels Justesen, Philip Bontrager, Julian Togelius, and Sebastian Risi. Deep learning for video game playing. *IEEE Transactions on Games*, 12(1):1–20, 2019.

Rishabh Kabra, Chris Burgess, Loic Matthey, Raphael Lopez Kaufman, Klaus Greff, Malcolm Reynolds, and Alexander Lerchner. Multi-object Datasets. https://github.com/deepmind/multi-object-datasets/, 2019.

Rishabh Kabra, Daniel Zoran, Goker Erdogan, Loic Matthey, Antonia Creswell, Matt Botvinick, Alexander Lerchner, and Chris Burgess. Simone: View-invariant, temporally-abstracted object representations via unsupervised video decomposition. *Advances in Neural Information Processing Systems*, 34:20146–20159, 2021.

Daniel Kahneman. *Thinking, fast and slow*. 2017.

Daniel Kahneman, Anne Treisman, and Brian J Gibbs. The reviewing of object files: Object-specific integration of information. *Cognitive psychology*, 24 (2):175–219, 1992.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and Franccois Fleuret. Transformers are Rnns: Fast Autoregressive Transformers with Linear Attention. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2020.

Jerrold J Katz and Jerry A Fodor. The structure of a semantic theory. *language*, 39(2):170–210, 1963.

Philip J Kellman and Elizabeth S Spelke. Perception of partly occluded objects in infancy. *Cognitive psychology*, 15(4):483–524, 1983.

Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE conference on computational intelligence and games (CIG)*, pages 1–8. IEEE, 2016.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference for Learning Representations*, 2015.

Diederik P Kingma and Max Welling. Auto-encoding Variational Bayes. In *International Conference on Learning Representations*, 2013.

Thomas Kipf, Elise van der Pol, and Max Welling. Contrastive Learning of Structured World Models. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=H1gax6VtDB`.

Thomas Kipf, Gamaleldin F Elsayed, Aravindh Mahendran, Austin Stone, Sara Sabour, Georg Heigold, Rico Jonschkowski, Alexey Dosovitskiy, and Klaus Greff. Conditional Object-centric Learning from Video. *arXiv preprint arXiv:2111.12594*, 2021.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Thomas N. Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard S. Zemel. Neural Relational Inference for Interacting Systems. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 2693–2702, 2018. URL `http://proceedings.mlr.press/v80/kipf18a.html`.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794*, 2021.

Kurt Koffka. Principles of Gestalt psychology. *Philosophy and Scientific Method*, 32(8), 1935.

Wolfgang Köhler. Gestalt psychology. *Psychologische Forschung*, 31(1), 1967.

Teuvo Kohonen. *Self-organization and associative memory*, volume 8. Springer Science & Business Media, 2012.

Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16*, pages 491–507. Springer, 2020.

Mojtaba Komeili, Kurt Shuster, and Jason Weston. Internet-augmented dialogue generation. *arXiv preprint arXiv:2107.07566*, 2021.

Adam Kosiorek, Hyunjik Kim, Yee Whye Teh, and Ingmar Posner. Sequential attend, infer, repeat: Generative modelling of moving objects. *Advances in Neural Information Processing Systems*, 31, 2018.

Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1061–1068, 2013.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Andreas Küchler. *Adaptive processing of structural data: from sequences to trees and beyond*. PhD thesis, Universität Ulm, 2000.

Andreas Küchler and Christoph Goller. Inductive learning in symbolic domains using structure-driven recurrent neural networks. In *Annual Conference on Artificial Intelligence*, pages 183–197. Springer, 1996.

Tejas D Kulkarni, Ankush Gupta, Catalin Ionescu, Sebastian Borgeaud, Malcolm Reynolds, Andrew Zisserman, and Volodymyr Mnih. Unsupervised learning of object keypoints for perception and control. In *Advances in Neural Information Processing Systems*, pages 10723–10733, 2019.

Manoj Kumar, Mohammad Babaeizadeh, Dumitru Erhan, Chelsea Finn, Sergey Levine, Laurent Dinh, and Durk Kingma. Videoflow: A flow-based generative model for video. In *International Conference on Learning Representations*, 2020.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Alex X. Lee, Richard Zhang, Frederik Ebert, Pieter Abbeel, Chelsea Finn, and Sergey Levine. Stochastic Adversarial Video Prediction. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HyEl3o05Fm`.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pages 3744–3753. PMLR, 2019.

Adam Lerer, Sam Gross, and Rob Fergus. Learning Physical Intuition of Block Towers by Example. In *International Conference on Machine Learning*, pages 430–438, 2016.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. *arXiv preprint arXiv:2301.12597*, 2023.

Liunian Harold Li, Pengchuan Zhang, Haotian Zhang, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Lijuan Wang, Lu Yuan, Lei Zhang, Jenq-Neng Hwang, et al. Grounded language-image pre-training. In *Proceedings of the IEEE/CVF*

Conference on Computer Vision and Pattern Recognition, pages 10965–10975, 2022.

Wenbin Li, Seyedmajid Azimi, Aleš Leonardis, and Mario Fritz. To fall or not to fall: A visual approach to physical stability prediction. *arXiv preprint arXiv:1604.00066*, 2016.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Zhixuan Lin, Yi-Fu Wu, Skand Vishwanath Peri, Weihao Sun, Gautam Singh, Fei Deng, Jindong Jiang, and Sungjin Ahn. SPACE: Unsupervised Object-oriented Scene Representation via Spatial Attention and Decomposition. In *International Conference on Learning Representations*, 2020.

Michael John Lingelbach, Damian Mrowca, Nick Haber, Li Fei Fei, and Daniel L K Yamins. Towards Curiosity-driven Learning of Physical Dynamics. *ICLR 2020 Bridging AI and Cognitive Science Workshop*, page 6, 2020.

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 *Advances in Neural Information Processing Systems*, pages 9605–9616, 2018.

Zhijian Liu, Jiajun Wu, Zhenjia Xu, Chen Sun, Kevin Murphy, William T. Freeman, and Joshua B. Tenenbaum. Modeling Parts, Structure, and System Dynamics via Predictive Learning. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJe10iC5K7`.

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. *Advances in Neural Information Processing Systems*, 33:11525–11538, 2020.

William Lotter, Gabriel Kreiman, and David Cox. Deep predictive coding networks for video prediction and unsupervised learning. In *International Conference on Learning Representations*, 2017.

Sindy Löwe, Klaus Greff, Rico Jonschkowski, Alexey Dosovitskiy, and Thomas Kipf. Learning object-centric video models by contrasting sets. *arXiv preprint arXiv:2011.10287*, 2020.

Sindy Löwe, Phillip Lippe, Maja Rudolph, and Max Welling. Complex-valued Autoencoders for Object Discovery. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856.

Sindy Löwe, Phillip Lippe, Francesco Locatello, and Max Welling. Rotating Features for Object Discovery. *arXiv preprint arXiv:2306.00600*, 2023.

Jiasen Lu, Christopher Clark, Rowan Zellers, Roozbeh Mottaghi, and Aniruddha Kembhavi. Unified-io: A unified model for vision, language, and multi-modal tasks. *arXiv preprint arXiv:2206.08916*, 2022.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv preprint arXiv:2104.08786*, 2021.

Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

Aman Madaan and Amir Yazdanbakhsh. Text and patterns: For effective chain of thought, it takes two to tango. *arXiv preprint arXiv:2209.07686*, 2022.

Aman Madaan, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. Language models of code are few-shot commonsense learners. *arXiv preprint arXiv:2210.07128*, 2022.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017.

Priyanka Mandikal and Kristen Grauman. Learning dexterous grasping with object-centric visual affordances. In *2021 IEEE international conference on robotics and automation (ICRA)*, pages 6169–6176. IEEE, 2021.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J Colwell, and Adrian Weller. Rethinking Attention with Performers. In *Int. Conf. on Learning Representations (ICLR)*, 2021.

Đorđe Miladinović, Aleksandar Stanić, Stefan Bauer, Jürgen Schmidhuber, and Joachim M Buhmann. Spatial dependency networks: Neural layers for improved generative image modeling. *arXiv preprint arXiv:2103.08877*, 2021.

Stephanie Milani, Nicholay Topin, Brandon Houghton, William H Guss, Sharada P Mohanty, Keisuke Nakata, Oriol Vinyals, and Noboru Sean Kuno. Retrospective analysis of the 2019 Minerl competition on sample efficient reinforcement learning. In *NeurIPS 2019 Competition and Demonstration Track*, pages 203–214. PMLR, 2020.

Peter M Milner. A model for visual shape recognition. *Psychological review*, 81(6):521, 1974.

Matthias Minderer, Alexey Gritsenko, and Neil Houlsby. Scaling Open-vocabulary Object Detection, 2023.

Sarthak Mittal, Alex Lamb, Anirudh Goyal, Vikram Voleti, Murray Shanahan, Guillaume Lajoie, Michael Mozer, and Yoshua Bengio. Learning to combine top-down and bottom-up signals in recurrent neural networks with attention over modules. In *International Conference on Machine Learning*, pages 6972–6986. PMLR, 2020.

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

Milad Moradi and Matthias Samwald. Evaluating the robustness of neural language models to input perturbations. *arXiv preprint arXiv:2108.12237*, 2021.

Alexander Mott, Daniel Zoran, Mike Chrzanowski, Daan Wierstra, and Danilo Jimenez Rezende. Towards interpretable reinforcement learning using attention augmented agents. *Advances in Neural Information Processing Systems*, 32, 2019.

Michael C Mozer. A Principle for Unsupervised Hierarchical Decomposition of Visual Scenes. In *Advances in Neural Information Processing Systems*, volume 11, 1998.

Michael C Mozer, Richard Zemel, and Marlene Behrmann. Learning to segment images using dynamic feature binding. *Advances in Neural Information Processing Systems*, 4, 1991.

Michael C Mozer, Denis Kazakov, and Robert V Lindsey. State-denoised recurrent neural networks. *arXiv preprint arXiv:1805.08394*, 2018.

Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li F Fei Fei, Josh Tenenbaum, and Daniel L Yamins. Flexible neural representation for physics prediction. In *Advances in neural information processing systems*, pages 8799–8810, 2018.

C Rodrigues Neto and José Fernando Fontanari. Multivalley structure of attractor neural networks. *Journal of Physics A: Mathematical and General*, 30(22):7945, 1997.

Binh X Nguyen, Tuong Do, Huy Tran, Erman Tjiputra, Quang D Tran, and Anh Nguyen. Coarse-to-fine reasoning for visual question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4558–4566, 2022.

Theo X Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar Lezama. Demystifying Gpt Self-repair for Code Generation. *arXiv preprint arXiv:2306.09896*, 2023.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

OpenAI. Introducing Chatgpt. 2022. URL `https://openai.com/blog/chatgpt`.

OpenAI. Dall·e 3 is now available in Chatgpt Plus and Enterprise. 2023a. URL `https://openai.com/blog/dall-e-3-is-now-available-in-chatgpt-plus-and-enterprise`.

OpenAI. Openai Chatgpt Api [gpt-3.5-turbo], Available at: `https://platform.openai.com/docs/model-index-for-researchers`. 2023b.

OpenAI. Gpt-4 Technical Report, 2023c.

OpenAI. Gpt-4v(ision) System Card. 2023d. URL https://cdn.openai.com/papers/GPTV_System_Card.pdf.

Ian Osband, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepesvári, Satinder Singh, Benjamin Van Roy, Richard Sutton, David Silver, and Hado van Hasselt. Behaviour Suite for Reinforcement Learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygf-kSYwH.

Roni Paiss, Ariel Ephrat, Omer Tov, Shiran Zada, Inbar Mosseri, Michal Irani, and Tali Dekel. Teaching clip to count to ten. *arXiv preprint arXiv:2302.12066*, 2023.

Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models. *arXiv preprint arXiv:2205.12255*, 2022.

Viorica Pătrăucean, Lucas Smaira, Ankush Gupta, Adrià Recasens Continente, Larisa Markeeva, Dylan Banarse, Skanda Koppula, Joseph Heyward, Mateusz Malinowski, Yi Yang, et al. Perception Test: A Diagnostic Benchmark for Multimodal Video Models. *arXiv preprint arXiv:2305.13786*, 2023.

Jordan B Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1-2):77–105, 1990.

Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. Grips: Gradient-free, edit-based instruction search for prompting large language models. *arXiv preprint arXiv:2203.07281*, 2022.

Isabeau Prémont-Schwarz, Alexander Ilin, Tele Hao, Antti Rasmus, Rinu Boney, and Harri Valpola. Recurrent ladder networks. In *Advances in Neural Information Processing Systems*, pages 6009–6019, 2017.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with" gradient descent" and beam search. *arXiv preprint arXiv:2305.03495*, 2023.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268): 1–8, 2021. URL `http://jmlr.org/papers/v22/20-1364.html`.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE transactions on pattern analysis and machine intelligence*, 44(3):1623–1637, 2020.

David Raposo, Adam Santoro, David Barrett, Razvan Pascanu, Timothy Lillicrap, and Peter Battaglia. Discovering objects and their relations from entangled scene representations. *arXiv preprint arXiv:1702.05068*, 2017.

Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *Advances in neural information processing systems*, pages 3546–3554, 2015.

John C Raven and JH Court. *Raven's progressive matrices*. Western Psychological Services Los Angeles, CA, 1938.

David P Reichert and Thomas Serre. Neuronal synchrony in complex-valued deep networks. In *International Conference on Learning Representations*, 2014.

Laria Reynolds and Kyle McDonell. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–7, 2021.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *International Conference on Machine Learning*, pages 1278–1286, 2014.

Maximilian Riesenhuber and Tomaso Poggio. Are Cortical Models Really Bound by the "binding Problem"? *Neuron*, 24(1):87–93, Sep 1999a. ISSN 08966273. doi: 10.1016/S0896-6273(00)80824-7.

Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019–1025, Nov 1999b. ISSN 1097-6256, 1546-1726. doi: 10.1038/14819.

Adina L Roskies. The binding problem. *Neuron*, 24(1):7–9, 1999.

Alvaro Sanchez Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin A. Riedmiller, Raia Hadsell, and Peter Battaglia. Graph Networks as Learnable Physics Engines for Inference and Control. In *ICML*, pages 4467–4476, 2018. URL `http://proceedings.mlr.press/v80/sanchez-gonzalez18a.html`.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W Battaglia. Learning to simulate complex physics with graph networks. *arXiv preprint arXiv:2002.09405*, 2020.

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*, pages 4967–4976, 2017.

Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy Lillicrap. Relational recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 7299–7310, 2018.

Rebecca Saxe and Susan Carey. The perception of causality in infancy. *Acta psychologica*, 123(1-2):144–165, 2006.

F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20 (1):61–80, Jan 2009. ISSN 1045-9227. doi: 10.1109/TNN.2008.2005605.

Timo Schick, Jane Dwivedi Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International Conference on Machine Learning*, pages 9355–9366. PMLR, 2021.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15*, pages 593–607. Springer, 2018.

J. Schmidhuber. Towards Compositional Learning with Dynamic Neural Networks. Technical Report FKI-129-90, Institut für Informatik, Technische Universität München, 1990a.

J. Schmidhuber. An On-line Algorithm for Dynamic Reinforcement Learning and Planning in Reactive Environments. In *Proc. IEEE/INNS International Joint Conference on Neural Networks, San Diego*, volume 2, pages 253–258, 1990b.

J. Schmidhuber. Curious Model-building Control Systems. In *Proceedings of the International Joint Conference on Neural Networks, Singapore*, volume 2, pages 1458–1463. IEEE press, 1991a.

J. Schmidhuber. Learning Factorial Codes By Predictability Minimization. Technical Report CU-CS-565-91, Dept. of Comp. Sci., University of Colorado at Boulder, Dec 1991b.

J. Schmidhuber. On decreasing the ratio between learning complexity and number of time-varying variables in fully recurrent nets. In *Proceedings of the International Conference on Artificial Neural Networks, Amsterdam*, pages 460–463. Springer, 1993a.

J. Schmidhuber and R. Huber. Learning to Generate Artificial Fovea Trajectories for target Detection. *International Journal of Neural Systems*, 2(1 & 2):135–141, 1991.

Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

Jürgen Schmidhuber. Making the world differentiable: On using fully recurrent self-supervised neural networks for dynamic reinforcement learning and planning in non-stationary environments. *Institut für Informatik, Technische Universität München. Technical Report FKI-126*, 90, 1990.

Jürgen Schmidhuber. Learning to Control Fast-weight Memories: An Alternative to recurrent Nets. Technical Report FKI-147-91, Institut für Informatik, Technische Universität München, March 1991c.

Jürgen Schmidhuber. Steps towards "self-referential" Learning. Technical Report CU-CS-627-92, Dept. of Comp. Sci., University of Colorado at Boulder, November 1992.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Jürgen Schmidhuber. An introspective network that can learn to run its own weight change algorithm. In *Proc. IEE Int. Conf. on Artificial Neural Networks*, pages 191–195, Brighton, UK, May 1993b.

Jürgen Schmidhuber. A self-referential weight matrix. In *Proc. Int. Conf. on Artificial Neural Networks (ICANN)*, pages 446–451, Amsterdam, Netherlands, September 1993c.

Jürgen Schmidhuber. A neural network that embeds its own meta-levels. In *Proc. IEEE Int. Conf. on Neural Networks (ICNN)*, San Francisco, CA, USA, March 1993d.

Jürgen Schmidhuber. Reducing the ratio between learning complexity and number of time varying variables in fully recurrent nets. In *International Conference on Artificial Neural Networks (ICANN)*, pages 460–463, Amsterdam, Netherlands, September 1993.

Jürgen Schmidhuber. Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem. *Frontiers in psychology*, 4:313, 2013.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

Jürgen Schmidhuber, Martin Eldracher, and Bernhard Foltin. Semilinear predictability minimization produces well-known feature detectors. *Neural Computation*, 8(4):773–786, 1996.

Christian Schuldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: a local Svm approach. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 3, pages 32–36. IEEE, 2004.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Maximilian Seitzer, Max Horn, Andrii Zadaianchuk, Dominik Zietlow, Tianjun Xiao, Carl-Johann Simon Gabriel, Tong He, Zheng Zhang, Bernhard Schölkopf, Thomas Brox, et al. Bridging the gap to real-world object-centric learning. *arXiv preprint arXiv:2209.14860*, 2022.

Mohit Sharma and Oliver Kroemer. Generalizing Object-centric Task-axes Controllers using Keypoints. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7548–7554, 2021.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*, 2023.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. *arXiv preprint arXiv:2010.15980*, 2020.

Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*, 2023.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419): 1140–1144, 2018.

Wolf Singer. Neuronal synchrony: a versatile code for the definition of relations? *Neuron*, 24(1):49–65, 1999.

Wolf Singer. Distributed processing and temporal codes in neuronal networks. *Cognitive neurodynamics*, 3:189–196, 2009.

Wolf Singer and Charles M Gray. Visual feature integration and the temporal correlation hypothesis. *Annual review of neuroscience*, 18(1):555–586, 1995.

Gautam Singh, Fei Deng, and Sungjin Ahn. Illiterate DALLE Learns to Compose. In *Int. Conf. on Learning Representations (ICLR)*, Virtual only, April 2022.

Gautam Singh, Yi-Fu Wu, and Sungjin Ahn. Simple unsupervised object-centric learning for complex and naturalistic videos. *Advances in Neural Information Processing Systems*, 35:18181–18196, 2022.

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

Elizabeth S Spelke and Katherine D Kinzler. Core knowledge. *Developmental science*, 10(1):89–96, 2007.

Elizabeth S Spelke, Roberta Kestenbaum, Daniel J Simons, and Debra Wein. Spatiotemporal continuity, smoothness of motion and object identity in infancy. *British Journal of Developmental Psychology*, 13(2):113–142, 1995.

Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3): 714–735, 1997.

Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pages 843–852, 2015.

Aleksandar Stanić and Jürgen Schmidhuber. R-sqair: Relational Sequential Attend, Infer, Repeat. *NeurIPS PGR workshop*, 2019.

Aleksandar Stanić, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Hierarchical relational inference. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

Aleksandar Stanić, Dylan Ashley, Oleg Serikov, Louis Kirsch, Francesco Faccio, Jürgen Schmidhuber, Thomas Hofmann, and Imanol Schlag. The Languini Kitchen: Enabling Language Modelling Research at Different Scales of Compute. *arXiv preprint arXiv:2309.11197*, 2023a.

Aleksandar Stanić, Anand Gopalakrishnan, Kazuki Irie, and Jürgen Schmidhuber. Contrastive Training of Complex-valued Autoencoders for Object Discovery. *Neural Information Processing Systems (NeurIPS)*, 2023b.

Aleksandar Stanić, Yujin Tang, David Ha, and Jürgen Schmidhuber. Learning to Generalize with Object-centric Agents in the Open World Survival Game Crafter. *IEEE Transactions on Games*, pages 1–20, 2023. doi: 10.1109/TG.2023.3276849.

Aleksandar Stanić, Sergi Caelles, and Michael Tschannen. Towards Truly Zero-shot Compositional Visual Reasoning with LLMs as Programmers. *arXiv preprint arXiv:2401.01974*, 2024.

Kaya Stechly, Matthew Marquez, and Subbarao Kambhampati. Gpt-4 Doesn't Know It's Wrong: An Analysis of Iterative Prompting for Reasoning Problems, 2023.

Karl Stelzner, Robert Peharz, and Kristian Kersting. Faster Attend-infer-repeat with Tractable Probabilistic Models. In *International Conference on Machine Learning*, pages 5966–5975, 2019.

Austin Stone, Oscar Ramirez, Kurt Konolige, and Rico Jonschkowski. The Distracting Control Suite–a Challenging Benchmark for Reinforcement Learning from Pixels. *arXiv preprint arXiv:2101.02722*, 2021.

Sanjay Subramanian, Ben Bogin, Nitish Gupta, Tomer Wolfson, Sameer Singh, Jonathan Berant, and Matt Gardner. Obtaining faithful interpretations from compositional neural networks. *arXiv preprint arXiv:2005.00724*, 2020.

Sanjay Subramanian, Medhini Narasimhan, Kushal Khangaonkar, Kevin Yang, Arsha Nagrani, Cordelia Schmid, Andy Zeng, Trevor Darrell, and Dan Klein. Modular Visual Question Answering via Code Generation. *arXiv preprint arXiv:2306.05392*, 2023.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448, 2015.

Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. *arXiv preprint arXiv:2303.08128*, 2023.

Gabriel Synnaeve, Nantas Nardelli, Alex Auvolat, Soumith Chintala, Timothée Lacroix, Zeming Lin, Florian Richoux, and Nicolas Usunier. Torchcraft: a library for machine learning research on real-time strategy games. *arXiv preprint arXiv:1611.00625*, 2016.

Yujin Tang and David Ha. The sensory neuron as a transformer: Permutation-invariant neural networks for reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Yujin Tang, Duong Nguyen, and David Ha. Neuroevolution of self-interpretable agents. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 414–424, 2020.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

Anne Treisman. The binding problem. *Current opinion in neurobiology*, 6(2): 171–178, 1996.

Anne Treisman. Solutions to the binding problem: progress through controversy and convergence. *Neuron*, 24(1):105–125, 1999.

Shubham Tulsiani, Hao Su, Leonidas J Guibas, Alexei A Efros, and Jitendra Malik. Learning shape abstractions by assembling volumetric primitives. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2635–2643, 2017.

Shimon Ullman. Visual routines. In *Readings in computer vision*, pages 298–328. Elsevier, 1987.

Karthik Valmeekam, Matthew Marquez, and Subbarao Kambhampati. Can Large Language Models Really Improve by Self-critiquing Their Own Plans?, 2023.

Sjoerd van Steenkiste, Michael Chang, Klaus Greff, and Jürgen Schmidhuber. Relational Neural Expectation Maximization: Unsupervised Discovery of Objects and their Interactions. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=ryH20GbRW`.

Sjoerd van Steenkiste, Klaus Greff, and Jürgen Schmidhuber. A Perspective on Objects and Systematic Generalization in Model-based Rl. In *ICML workshop on Generative Modeling and Model-Based Reasoning for Robotics and AI*, 2019.

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*, pages 5998–6008, 2017.

Rishi Veerapaneni, John D Co Reyes, Michael Chang, Michael Janner, Chelsea Finn, Jiajun Wu, Joshua Tenenbaum, and Sergey Levine. Entity abstraction in visual model-based reinforcement learning. In *Conference on Robot Learning*, pages 1439–1456. PMLR, 2020.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Pablo Villalobos, Jaime Sevilla, Lennart Heim, Tamay Besiroglu, Marius Hobbhahn, and Anson Ho. Will we run out of data? An analysis of the limits of scaling datasets in Machine Learning. *arXiv preprint arXiv:2211.04325*, 2022.

Ruben Villegas, Mohammad Babaeizadeh, Pieter-Jan Kindermans, Hernan Moraldo, Han Zhang, Mohammad Taghi Saffar, Santiago Castro, Julius Kunze, and Dumitru Erhan. Phenaki: Variable length video generation from open domain textual description. *arXiv preprint arXiv:2210.02399*, 2022.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John

Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

Christoph Von Der Malsburg. Am I thinking assemblies? In *Brain Theory: Proceedings of the First Trieste Meeting on Brain Theory, October 1–4, 1984*, pages 161–176. Springer, 1986.

Christoph Von Der Malsburg. The correlation theory of brain function. In *Models of neural networks*, pages 95–119. Springer, 1994.

Christoph von der Malsburg. Binding in models of perception and brain function. *Current opinion in neurobiology*, 5(4):520–526, 1995.

Christoph von der Malsburg and Joachim Buhmann. Sensory segmentation with coupled neural oscillators. *Biological cybernetics*, 67(3):233–242, 1992.

Christoph von der Malsburg and Werner Schneider. A neural cocktail-party processor. *Biological cybernetics*, 54(1):29–40, 1986.

W. von Humboldt, W.F. von Humboldt, M. Losonsky, P. Heath, X. Yao, H.W. von, K. Ameriks, and D.M. Clarke. *Humboldt: 'On Language': On the Diversity of Human Language Construction and Its Influence on the Mental Development of the Human Species*. Cambridge Texts in the History of Philosophy. Cambridge University Press, 1999. ISBN 9780521667722. URL `https://books.google.ch/books?id=_UODbGlD4WUC`.

Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. In *Advances in neural information processing systems*, pages 613–621, 2016.

Jianfeng Wang, Zhengyuan Yang, Xiaowei Hu, Linjie Li, Kevin Lin, Zhe Gan, Zicheng Liu, Ce Liu, and Lijuan Wang. Git: A generative image-to-text transformer for vision and language. *arXiv preprint arXiv:2205.14100*, 2022a.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*, 2023a.

Peng Wang, An Yang, Rui Men, Junyang Lin, Shuai Bai, Zhikang Li, Jianxin Ma, Chang Zhou, Jingren Zhou, and Hongxia Yang. Ofa: Unifying architectures, tasks, and modalities through a simple sequence-to-sequence learning framework. In *International Conference on Machine Learning*, pages 23318–23340. PMLR, 2022b.

Su Wang, Chitwan Saharia, Ceslee Montgomery, Jordi Pont Tuset, Shai Noy, Stefano Pellegrini, Yasumasa Onoe, Sarah Laszlo, David J Fleet, Radu Soricut, et al. Imagen editor and editbench: Advancing and evaluating text-guided image inpainting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18359–18369, 2023b.

Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7794–7803, 2018.

Xingyao Wang, Sha Li, and Heng Ji. Code4struct: Code generation for few-shot event structure prediction. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3640–3663, 2023c.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022c.

Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. In *Advances in neural information processing systems*, pages 4539–4547, 2017.

Nicholas Watters, Loic Matthey, Matko Bosnjak, Christopher P Burgess, and Alexander Lerchner. Cobra: Data-efficient Model-based Rl through Unsupervised Object Discovery and Curiosity-driven Exploration. *ICML workshop on Generative Modeling and Model-Based Reasoning for Robotics and AI*, 2019a.

Nicholas Watters, Loic Matthey, Christopher P Burgess, and Alexander Lerchner. Spatial broadcast decoder: A simple architecture for learning disentangled representations in vaes. *arXiv preprint arXiv:1901.07017*, 2019b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

Jerry Wei, Jason Wei, Yi Tay, Dustin Tran, Albert Webson, Yifeng Lu, Xinyun Chen, Hanxiao Liu, Da Huang, Denny Zhou, et al. Larger language models do in-context learning differently. *arXiv preprint arXiv:2303.03846*, 2023.

Yuxin Wen, Neel Jain, John Kirchenbauer, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Hard prompts made easy: Gradient-based discrete optimization for prompt tuning and discovery. *arXiv preprint arXiv:2302.03668*, 2023.

Markus Werning, Edouard Machery, and Gerhard Schurz. The Compositionality of Meaning and Content, Volume 1: Foundational issues. 2005.

Max Wertheimer. Untersuchungen zur Lehre von der Gestalt. Ii. *Psychologische Forschung*, 4(1):301–350, 1923.

Daan Wierstra, Alexander Foerster, Jan Peters, and Juergen Schmidhuber. Solving deep memory Pomdps with recurrent policy gradients. In *International conference on artificial neural networks*, pages 697–706. Springer, 2007.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Yizhe Wu, Oiwi Parker Jones, Martin Engelcke, and Ingmar Posner. Apex: Unsupervised, Object-centric Scene Segmentation and Tracking for Robot Manipulation. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3375–3382, 2021.

Ziyi Wu, Nikita Dvornik, Klaus Greff, Thomas Kipf, and Animesh Garg. Slotformer: Unsupervised Visual Dynamics Simulation with Object-centric Models. In *The Eleventh International Conference on Learning Representations*, 2023.

Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. Torcs, the open racing car simulator. *Software available at http://torcs. sourceforge. net*, 4(6):2, 2000.

Junbin Xiao, Xindi Shang, Angela Yao, and Tat-Seng Chua. Next-qa: Next phase of question-answering to explaining temporal actions. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9777–9786, 2021.

Fei Xu and Susan Carey. Infants' metaphysics: The case of numerical identity. *Cognitive psychology*, 30(2):111–153, 1996.

Hanwei Xu, Yujun Chen, Yulun Du, Nan Shao, Yanggang Wang, Haiyu Li, and Zhilin Yang. Gps: Genetic Prompt Search for Efficient Few-shot Learning. *arXiv preprint arXiv:2210.17041*, 2022a.

Jiarui Xu, Shalini De Mello, Sifei Liu, Wonmin Byeon, Thomas Breuel, Jan Kautz, and Xiaolong Wang. Groupvit: Semantic segmentation emerges from text supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18134–18144, 2022b.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2023a.

Lingfeng Yang, Yueze Wang, Xiang Li, Xinlong Wang, and Jian Yang. Fine-grained Visual Prompting. *arXiv preprint arXiv:2306.04356*, 2023b.

Zhengyuan Yang, Zhe Gan, Jianfeng Wang, Xiaowei Hu, Yumao Lu, Zicheng Liu, and Lijuan Wang. An empirical study of gpt-3 for few-shot knowledge-based vqa. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 3081–3089, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

Yuan Yao, Ao Zhang, Zhengyan Zhang, Zhiyuan Liu, Tat-Seng Chua, and Maosong Sun. Cpt: Colorful prompt tuning for pre-trained vision-language models. *arXiv preprint arXiv:2109.11797*, 2021.

Qinghao Ye, Guohai Xu, Ming Yan, Haiyang Xu, Qi Qian, Ji Zhang, and Fei Huang. Hitea: Hierarchical temporal-aware video-language pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15405–15416, 2023.

Qi Yi, Rui Zhang, Shaohui Peng, Jiaming Guo, Xing Hu, Zidong Du, Xishan Zhang, Qi Guo, and Yunji Chen. Object-category Aware Reinforcement Learning. *arXiv preprint arXiv:2210.07802*, 2022.

Jaesik Yoon, Yi-Fu Wu, Heechul Bae, and Sungjin Ahn. An Investigation into Pre-training Object-centric Representations for Reinforcement Learning. *arXiv preprint arXiv:2302.04419*, 2023.

Jiahui Yu, Zirui Wang, Vijay Vasudevan, Legg Yeung, Mojtaba Seyedhosseini, and Yonghui Wu. Coca: Contrastive captioners are image-text foundation models. *arXiv preprint arXiv:2205.01917*, 2022.

Licheng Yu, Patrick Poirson, Shan Yang, Alexander C Berg, and Tamara L Berg. Modeling context in referring expressions. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14*, pages 69–85. Springer, 2016.

Jinyang Yuan, Bin Li, and Xiangyang Xue. Generative Modeling of Infinite Occluded Objects for Compositional Scene Representation. In *International Conference on Machine Learning*, pages 7222–7231, 2019.

Mert Yuksekgonul, Federico Bianchi, Pratyusha Kalluri, Dan Jurafsky, and James Zou. When and Why Vision-language Models Behave like Bags-of-words, and What to Do About It? In *The Eleventh International Conference on Learning Representations*, 2022.

Andrii Zadaianchuk, Maximilian Seitzer, and Georg Martius. Self-supervised visual reinforcement learning with object-centric representations. *arXiv preprint arXiv:2011.14381*, 2020.

Andrii Zadaianchuk, Maximilian Seitzer, and Georg Martius. Object-centric Learning for Real-World Videos by Predicting Temporal Feature Similarities. *arXiv preprint arXiv:2306.04829*, 2023.

Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew Botvinick, Oriol Vinyals, and Peter Battaglia. Deep reinforcement learning with relational inductive biases. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HkxaFoC9KQ`.

Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, et al. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*, 2022.

Yan Zeng, Xinsong Zhang, and Hang Li. Multi-grained vision language pre-training: Aligning texts with visual concepts. *arXiv preprint arXiv:2111.08276*, 2021.

Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, et al. The visual task adaptation benchmark. 2019.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. *arXiv preprint arXiv:2303.15343*, 2023.

Amy Zhang, Yuxin Wu, and Joelle Pineau. Natural environment benchmarks for reinforcement learning. *arXiv preprint arXiv:1811.06032*, 2018.

Tiancheng Zhao, Tianqi Zhang, Mingwei Zhu, Haozhan Shen, Kyusong Lee, Xiaopeng Lu, and Jianwei Yin. Vl-checklist: Evaluating pre-trained vision-language models with objects, attributes and relations. *arXiv preprint arXiv:2207.00221*, 2022.

Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate before use: Improving few-shot performance of language models. In *International Conference on Machine Learning*, pages 12697–12706. PMLR, 2021.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022a.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022b.

Mingchen Zhuge, Haozhe Liu, Francesco Faccio, Dylan R Ashley, Róbert Csordás, Anand Gopalakrishnan, Abdullah Hamdi, Hasan Abed Al Kader Hammoud, Vincent Herrmann, Kazuki Irie, et al. Mindstorms in Natural Language-based Societies of Mind. *arXiv preprint arXiv:2305.17066*, 2023.

Daniel Zoran, Rishabh Kabra, Alexander Lerchner, and Danilo J Rezende. Parts: Unsupervised segmentation with slots, attention and independence maximization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10439–10447, 2021.