Towards Human-like Perception with Connectionist Models

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
Anand Gopalakrishnan

under the supervision of
Prof. Jürgen Schmidhuber

August 2025

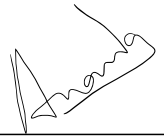
Dissertation Committee

Prof. Cesare Alippi	Università della Svizzera italiana, Switzerland
Prof. Piotr Didyk	Università della Svizzera italiana, Switzerland
Prof. Michael C. Mozer	University of Colorado, Boulder/Google DeepMind, USA
Prof. Thomas Serre	Brown University, USA

Dissertation accepted on 22 August 2025

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

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.



Anand Gopalakrishnan
Lugano, 22 August 2025

Abstract

The last decade has seen rapid progress in Artificial Intelligence (AI) capabilities across domains such as game-playing, language understanding, multimedia generation, scientific applications etc. with neural networks (NNs) being the centerpiece. Large NNs trained on internet-scale datasets (Foundation Models) have led to breakthrough task performance across vision, audio and language modalities. However, these Foundation Models continue to struggle on tasks that require a fine-grained understanding of inputs—objects and relations, suggesting a gap in their ability to comprehend the underlying structure in sensory inputs compared to humans. Such failures can be seen as manifestations of an inability to form and relate symbol-like representations of objects, i.e. resolve the *binding problem*. This dissertation focuses on this challenge of grouping sensory inputs into modular (object-centric) representations using NNs with no supervision. In the first half, we develop a new method to discovery object keypoints that are more robust to distractors and alleviate certain systematic biases of previous methods. We extend slot-based models typically designed to spatially group pixels to visual objects to temporally group state-action sequences as sub-routines. We develop new masking and decoding methods to enable each slot to model contiguous input elements. The second half focuses on synchrony-based models, an alternate class of object-centric models to slots, that use phase-components of complex-valued activations to store object bindings. We design new contrastive training procedures for synchrony-based models which improve phase synchronization and object storage capacity. We refine these ideas, by simplifying the inductive biases and training process of synchrony-based models using complex-valued weights and recurrent computation. Lastly, we show how complex-valued activations allows a natural decoupling of content and position-based matches and design a powerful relative position encoding scheme for Transformer models. Broadly, we believe that a key component of human-level intelligence is our ability to construct abstract *mental models* of the world by composing structured primitives. We hope this thesis contributes towards the grand challenge of endowing machines with the same capacity.

Acknowledgements

This dissertation wouldn't have taken its current form without the influence of many people in several ways over the years. I would like to start off by thanking Jürgen Schmidhuber for the opportunity to do my PhD under his supervision. Our discussions have been one of the driving forces that shaped my understanding of AI and helped me develop a broad view of the field. I'm grateful for the complete freedom he afforded me over so many years to chart my own research program. Second, I'm deeply indebted to Michael Mozer for his generosity and kindness, and for the impact of his mentorship on my research trajectory. I hope to emulate his approach to scientific research—one marked by endless curiosity, enthusiasm and attention to detail, all while retaining a playful spirit.

Further, I'm thankful to my other main research collaborators—Sjoerd van Steenkiste, Kazuki Irie and Aleksandar Stanić. Sjoerd for helping me get started in the PhD with my first research project and introducing me to the binding problem. Sjoerd's ability to scope out tractable research problems and provide critical but constructive feedback was invaluable. Kazuki for his encouragement, technical feedback and relentless work ethic during our collaborations. Aleks for our collaboration on complex-valued autoencoders, shared research interests, but mostly for our close friendship and camaraderie outside work.

I'm grateful to my lab members, Imanol Schlag, Róbert Csordás, Francesco Faccio, Louis Kirsch, Aditya Ramesh, Vincent Herrmann, Dylan Ashley, Mikhail Andronov and Eric Alcaide for all the wonderful times we spent together either discussing research, playing boardgames or hiking. I would also like to thank Taimoor Tariq, Luca Surace, Sophie Kergaßner, Nicolai Hermann, Isha Das, Beatrice Granaroli, Martina Boschi, Alex Bortolotti, Alessandro Manenti, Tommaso Marzi and Nicolau Oliver Burwitz for making life at the university

more fun and collegial.

I had a wonderful time at the Center for Brains, Minds and Machines summer school in 2023 where I learned a lot about interdisciplinary work in neuroscience, psychology, cognitive science and AI while making great friends in Noémi Éltető, Emily Cheng, Julian Coda-Forno and Ravi Srinivasan. I'm also grateful to Etai Littwin, Josh Susskind, Vimal Thilak, Matteo Pagliardini and many others at Apple MLR for a stimulating internship in the summer of 2024.

I'm grateful to Vincent Herrmann for his friendship, for all the espressos, Neapolitan pizzas and movies we shared together. I'm also especially grateful to Marilena Palomba for her friendship, support and all the engaging conversations during our long walks exploring Lugano. I'm deeply grateful for my decades-long friendship with Aditya Ramesh, the only constant in my life. From being roommates in dingy dormitories in Surathkal to flatmates in less dingy student housing in Lugano, life has come full circle. Our endless conversations on AI, football culture, politics and philosophy have been instrumental to my personal growth.

I would also like to thank the members of my dissertation committee—Cesare Alippi, Piotr Didyk, Michael Mozer and Thomas Serre for taking time to review my dissertation and provide valuable feedback. Finally, I am deeply grateful to my parents, Meera and Gopal, and sibling Aparna, for their love, patience and encouragement from afar during these long years.

Contents

Contents vi

1 Introduction 2
- 1.1 The Decade of Deep Learning . . . 3
- 1.2 Gap Between Human and Machine Perception . . . 5
- 1.3 The Binding Problem . . . 9
- 1.4 Dissertation Overview and Contributions . . . 12
- 1.5 List of Publications . . . 13

2 Keypoint Discovery with Local Spatial Predictability 15
- 2.1 Overview . . . 15
- 2.2 Method . . . 17
- 2.3 Results . . . 22
- 2.4 Related Work . . . 28
- 2.5 Discussion . . . 29
- 2.6 Conclusion . . . 30

3 Learning Temporal Abstractions with Slot-based Transformers 31
- 3.1 Overview . . . 31
- 3.2 Method . . . 34
- 3.3 Results . . . 39
- 3.4 Related Work . . . 46
- 3.5 Conclusion . . . 48

4 Contrastive Training of Complex-valued Autoencoders 51
- 4.1 Overview . . . 51
- 4.2 Background . . . 54
- 4.3 Method . . . 55

4.4 Results . . . 59
4.5 Related Work . . . 67
4.6 Conclusion . . . 69

5 Recurrent Complex-Weighted Autoencoders 70
5.1 Overview . . . 70
5.2 Method . . . 74
5.3 Results . . . 77
5.4 Related Work . . . 84
5.5 Conclusion . . . 86

6 Polar Coordinate Positional Embedding 89
6.1 Overview . . . 89
6.2 Background . . . 90
6.3 Method . . . 92
6.4 Results . . . 95
6.5 Related Work . . . 100
6.6 Conclusion . . . 102

7 Outlook 103

A Appendix on Keypoint Discovery using Local Spatial Predictability 109
A.1 Datasets . . . 109
A.2 Architecture and Training Details . . . 109
A.3 Additional Visualizations . . . 112

B Appendix on Temporal Abstractions with Slot-based Transformers 116
B.1 Datasets . . . 116
B.2 Evaluation Metrics . . . 117
B.3 Architecture and Training Details . . . 118
B.4 Baseline Models and Training Details . . . 124
B.5 Visualizations . . . 129
B.6 Additional Results . . . 132

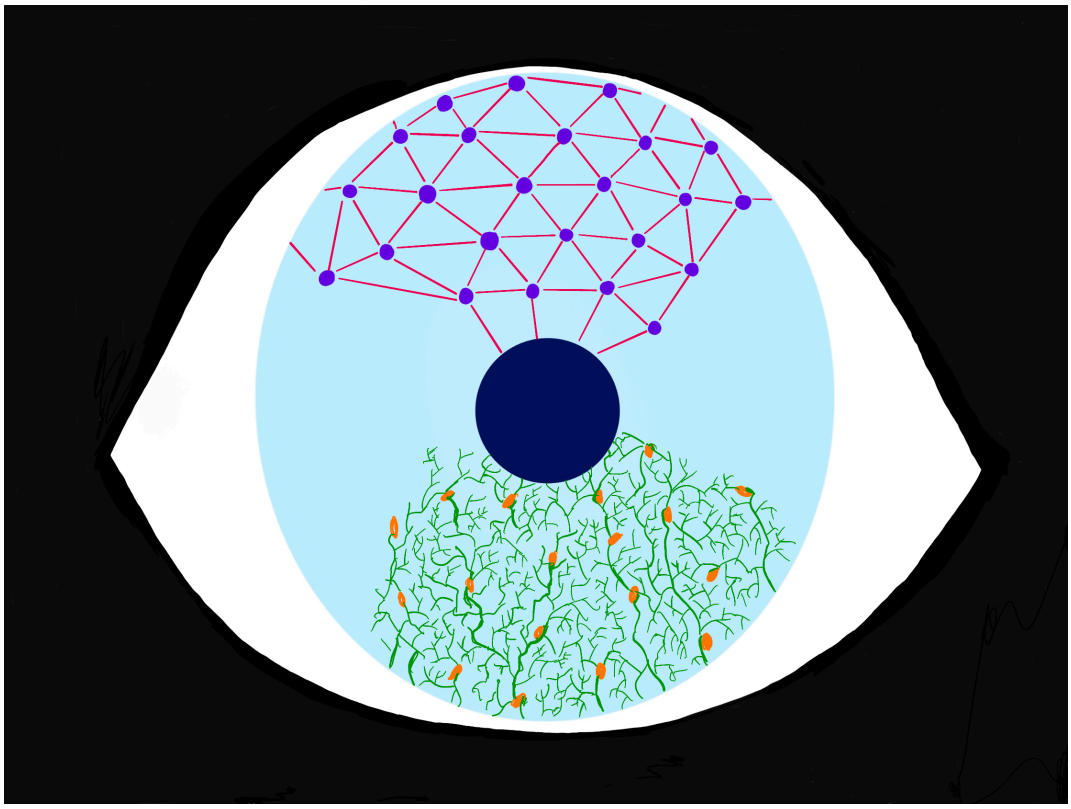
C Appendix on Contrastive Training of Complex-valued Autoencoders 133
C.1 Datasets . . . 133
C.2 Models . . . 133
C.3 Training Details . . . 135
C.4 Computational Efficiency . . . 136
C.5 Object Assignments from Phase Maps . . . 136

C.6 Additional Results 136
C.7 Additional Visualizations 142

D Appendix on Recurrent Complex-weighted Autoencoders 152
D.1 Datasets 152
D.2 Models 152
D.3 Training Details 153
D.4 Computational Efficiency 153
D.5 Phase Map Visualization 154
D.6 Evaluation Details 155
D.7 Additional Results 156
D.8 Additional Visualizations 157

E Appendix on Polar Coordinate Position Embedding 161
E.1 RoPE in polar form 161
E.2 Datasets 162
E.3 Models 164
E.4 Training Details 165
E.5 Additional Results 165
E.6 Additional Visualizations 166

Bibliography 167



Chapter 1

Introduction

The term Artificial Intelligence (AI) was coined by John McCarthy during a summer research conference at Dartmouth in 1956. The goal of this research program was *"to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it."* In essence, their goal was to build a machine capable of emulating human intelligence.

The remarkable power of human cognition extends well beyond our immediate sensory experience, allowing us to conceptualize abstract ideas and reason about phenomena we cannot directly observe. Through reason, imagination, and symbolic thought, we can construct *"mental models"* of unobservable phenomena—from subatomic particles to black holes, from historical civilizations to potential futures. This remarkable capacity for abstract thought enables us to develop scientific theories, create mathematical proofs, connect distant concepts through analogies [Hofstadter, 1996], all while physically confined to a narrow slice of reality.

Two schools of thought soon emerged within AI with contrasting proposals to build intelligent systems. The symbolic school sought to build-in knowledge of the world using symbols and rules for their manipulation whose semantics are apriori specified by designers. The most successful approaches from this design paradigm include expert systems, logic programs, automated theorem provers, heuristic search algorithms etc. Despite these successes this paradigm faced some severe limitations. These systems require experts to manually encode

knowledge and rules, making it labor-intensive and difficult to scale them to complex problem domains as the number of possible states to consider grows exponentially with problem complexity. Further, they struggle to apply their knowledge to novel situations which are not explicitly covered by rules and therefore generalize poorly. They face difficulty connecting abstract symbols to real-world meaning and perception (*symbol grounding problem* [Harnad, 1990]). They also struggle with uncertainty and ambiguity, typical of the real world, and find it difficult to reason in a probabilistic fashion.

In contrast, the connectionist school took a brain-inspired approach with an emphasis on *tabula-rasa learning* using neural networks (NNs) starting from low-level sensory data. While methods from the symbolic school enjoyed periods of dominance in the early days, the balance has swung decisively in the direction of the connectionists in the preceding decade with NNs becoming the centerpiece in modern AI systems [Schmidhuber, 2020].

1.1 The Decade of Deep Learning

The past decade has witnessed an astounding rate of progress in the capabilities of AI systems across wide-ranging domains such as game-playing [Silver et al., 2016, 2017; Vinyals et al., 2019; Brown and Sandholm, 2019], language understanding [Brown et al., 2020], conversational assistance [Ouyang et al., 2022], multimedia generation [Ramesh et al., 2022; Baldridge et al., 2024; Brooks et al., 2024; Poole et al., 2023; Agostinelli et al., 2023; Copet et al., 2024], for scientific discoveries eg. protein structure prediction [Jumper et al., 2021], plasma control in fusion reactors [Degrave et al., 2022], chip layout design [Mirhoseini et al., 2021] etc. This has resulted in increased fervor of the near-term possibility of achieving human-level intelligence or even surpassing it in the most general sense.

A central component underpinning the remarkable capabilities of the current generation of AI systems are NNs [Schmidhuber, 2015, 2022]. Learning in the context of NNs amounts to the process of finding a good set of connection strengths (weights) between neurons (units) to optimize an objective function which is a measure of competence on a task given a collection of input-output examples. This process of training NNs leads them to compress and represent knowledge about complex statistical regularities in these input-output examples of text, images, videos, audio or a combination thereof.

NNs perform massively parallel distributed information processing with the earliest successful demonstrations of learning with deep NNs made by Ivakhnenko and Lapa [1965]; Amari [1967]. The strengths of this paradigm of learning such distributed representations [Hinton, 1984] include— i) coding efficiency: relatively small number of neurons can represent a much larger number of concepts, as each neuron contributes to the representation of multiple concepts making it vastly more efficient than localist representations. ii) generalization: since similar concepts tend to have similar patterns of activity, leading to overlaps. This facilitates the network to generalize to new, unseen examples that share varying degrees of similarity to those it has already encountered. The network can naturally capture semantic relationships and similarities between concepts. iii) robustness: since information is distributed across many neurons, the network is less susceptible to noisy and/or partial inputs. If some information is lost or corrupted, the quality of representation is still reasonably well preserved. iv) relational: their ability to "fill in" missing information in partial inputs also makes them a good candidate for an associative memory. This ability to "fill in" missing information is achieved by learning to capture relations across several related concepts and exploit them to recall the missing pieces.

However, NNs often learn spurious or "shortcut" features [Arjovsky et al., 2019; Geirhos et al., 2020] relying on surface-level regularities in the training data to solve the task at hand. This leads them to generalize poorly on new data which share the same underlying structure with only superficial differences. This phenomena has been observed in the language domain where systems for natural language inference adopt brittle heuristics such as lexical overlap, constituent and subsequence that successfully solve the frequent (easy) examples in the training data but fail on the rare (hard) ones [McCoy et al., 2019]. More recently, Mirzadeh et al. [2025] show that foundation models trained on math problems which require several steps of show noticeable drops or variance in performance when numerical values in questions are modified or number of clauses increase. This highlights the brittleness of reasoning strategies learned and raises doubts on whether these models simply learn to mimic reasoning chains observed in the training data rather than learn the underlying algorithms.

While deep reinforcement learning (RL) agents have achieved superhuman performance on many games [Mnih et al., 2015; Schrittwieser et al., 2020] they continue to overfit to their training data and struggle to generalize on similar but unseen environments [Cobbe et al., 2018; Zhang et al., 2018a]. This suggests

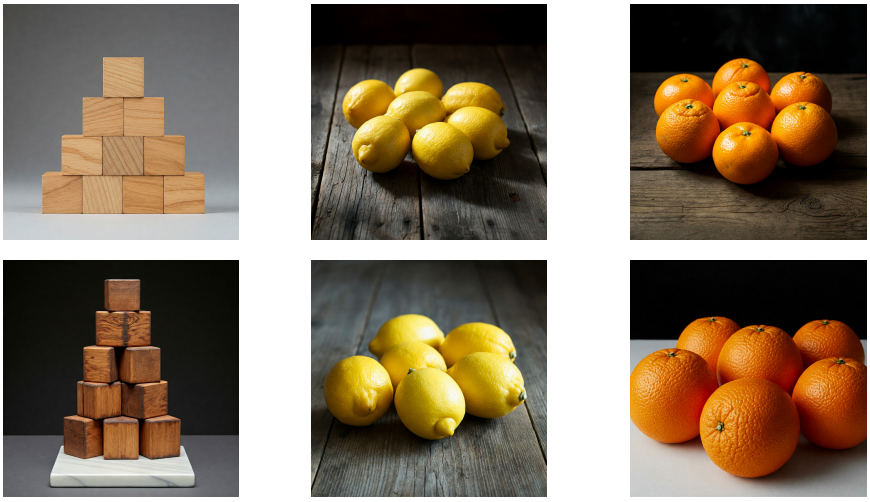


Figure 1.1: Illustration of failure mode in state-of-the-art text-to-image generative model Imagen 3 [Baldridge et al., 2024] to count objects. Prompt for panels on the left is "A pyramid of 7 wooden cubes". However, it generates samples of pyramids with 10 or 9 cubes instead. Prompt for panels in the center is "5 lemons on a wooden floor". However, generated samples contain 7 and 6 lemons instead. Prompt for panels on the right is "5 oranges on a wooden floor". However, generated samples contain 7 and 6 oranges instead.

a limited capacity of NNs to form modular representations and adapt to unseen scenarios by composing learned primitives. Therefore, while NNs have made remarkable strides their propensity to rely on such surface-level statistics leads them to fall short of human-level generalization and efficiency to build models of the world and reuse these models in novel ways [Lake et al., 2017].

1.2 Gap Between Human and Machine Perception

Current state-of-the-art foundation models use vastly more data harvested from the internet compared to human children [Frank, 2023]. Humans on the other hand develop an understanding of the world around them in terms of objects and relations from an early age [Spelke and Kinzler, 2007]. Infants as young as 6 months develop mechanisms for structured perception and intuitive physics. They divide perceptual arrays into units that move as connected 'wholes', maintain their form as they move, act upon each other only on contact and can infer their states even under occlusions [Baillargeon et al., 1985; Spelke,



Figure 1.2: Illustration of novel object-attribute binding failures in Imagen 3 [Baldridge et al., 2024]. Prompt for the top panel "A yellow orange and an orange lemon". However, the model consistently fails to apply the right colors to the respects fruits always reverting to their default colors. Prompt for the bottom panel is "A red blueberry and a blue strawberry". However, it continues to generate a blue blueberry and a red berry of a different kind not present in the prompt. The samples only a partially successful attempt to bind the right color to the fruit.

1990]. Such mechanisms to modularize knowledge of the external world into abstract entities and relations is believed to support subsequent cognitive development including object recognition [Landau et al., 1998], categorization [Rosch, 1978], language acquisition [Miller and Johnson-Laird, 1976], causal reasoning [Saxe and Carey, 2006], tool use [Van Leeuwen et al., 1994] etc. Current state-of-the-art systems for machine perception still fall short of the generality and efficiency of the human perceptual system to learn such powerful representations directly from sensory stimuli with minimal supervision.

In a recent study, Campbell et al. [2024] investigated state-of-the-art vision-language models (VLMs) on several tasks that involve reasoning about multi-object scenes such as counting, localization and visual analogy. They observed that while these models are good at counting, searching or describing scene components in the small number setting, their performance quickly

degrades with greater number of objects (distractors). They attribute this degradation in performance to representational interference an outcome of insufficiently addressing the *binding problem* via purely feedforward processing. They argue for the incorporation of appropriate mechanisms that enable serial processing to limit such representational interference in current VLMs and alleviate these failures. In a similar vein, Schulze Buschoff et al. [2025] also tested the vision capabilities of several state-of-the-art VLMs including GPT-4V [OpenAI, 2023] and Claude Opus [Anthropic, 2024] on the domains of intuitive physics and physical reasoning. They observed that models are inferior to humans in terms of their understanding of common sense physics and causation. For example, they struggle to count the number of blocks in a block tower or near above-chance level predictions about the tower's stability or which blocks would fall off whereas humans are much more competent at making such predictions. They also struggle to infer the underlying causation in collision events, eg. did ball A help or hinder ball B to pass through the gate?

Further, scalable methods for representation learning from images and natural language, eg. CLIP [Radford et al., 2021], are known to struggle on structured visual tasks that require counting or fine-grained relational reasoning [Thrush et al., 2022]. This suggests that the learned visual representations largely capture coarse scene-level properties but miss out on fine-grained local structure.

Similarly, while state-of-the-art generative text-to-image models like DALL·E [Ramesh et al., 2022] and Imagen [Baldridge et al., 2024] have made remarkable progress to synthesize images of high-fidelity and visual realism almost indistinguishable to the human eye. These models still struggle to faithfully represent relations between objects, logical operators or numerosity of objects as highlighted by several recent studies [Conwell and Ullman, 2022; Leivada et al., 2023; Conwell et al., 2024]. We can see in Figure 1.1 that Imagen 3 fails to faithfully generate images which contain a given number of objects. We also observe that Imagen 3 struggles to correctly bind attributes (color) to the respective objects for novel (rare) combinations as highlighted in Figure 1.2. Further, it also struggles to capture the correct relational attributes between objects as shown in Figure 1.3.

While CNNs and Vision Transformers (ViTs) achieve superhuman-level performance [He et al., 2015] on classification benchmarks like ImageNet [Russakovsky et al., 2015], they employ markedly different strategies compared to humans. These models seem to be heavily reliant on the object's texture



Figure 1.3: Illustration of failure mode with respect to modeling relations between objects in state-of-the-art text-to-image generative model Imagen 3 [Baldridge et al., 2024]. Prompt for the top panel is "A horse riding an astronaut" which is rare relation between known objects with respect to training data. As shown, the model always seems to revert to the "default" of "An astronaut riding a horse". Prompt for the bottom panel is "An orange under a cup". However, we can see that the orange is placed either inside or in front of but never under the cup.

[Baker et al., 2018; Geirhos et al., 2018; Hermann et al., 2020] to classify them, whereas people define and classify solid objects by shape [Landau et al., 1988]. Other common shortcuts include a predisposition to classify based on backgrounds rather than foreground objects [Beery et al., 2018; Xiao et al., 2021; Moayeri et al., 2022], or based on individual diagnostic pixels rather than higher-order image features [Malhotra et al., 2020]. Such classification strategies used by these models typically makes them brittle when confronted with adversarial examples, i.e. natural images that contain additive noise imperceptible to the human eye and do not change our perception of the underlying semantic information but dramatically alter it for artificial vision systems.

We believe that failures in current state-of-the-art vision models resulting from poor representational structure arise from a deeper underlying issue, an inability

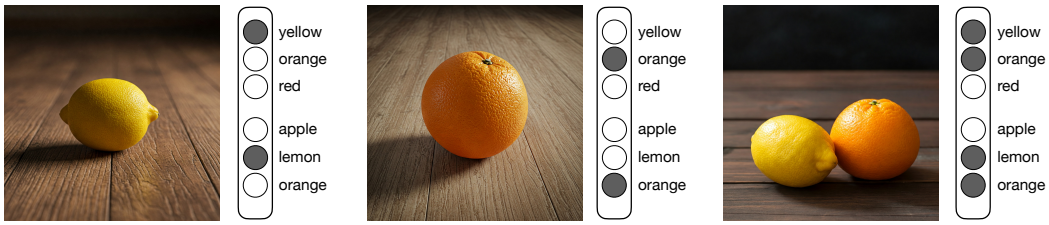


Figure 1.4: Illustration of *superposition catastrophe*: Distributed representation as a collection of disentangled features like color and type of fruit. (Left and center): leads to ambiguity when dealing with multiple objects. (Right): representation could denote either a yellow lemon and an orange or an orange lemon and a yellow orange. Template used from [Greff et al., 2020].

of NNs to form and manipulate symbol-like representations from raw sensory inputs without explicit supervision. This inability to decompose inputs into modular constituents and relations in its internal representations subsequently limits them to combine known constituents in new ways to generalize on unseen inputs. The promise of learning symbol-like representations with NNs is that it combines the expressiveness of distributed representations with the compositionality of symbols. However, it faces significant challenges including *"superposition catastrophe"* first highlighted by Milner [1974]; von der Malsburg [1981] (illustrated in Figure 1.4) which necessitate the use of specialized inductive biases to overcome. But before we delve into this, perhaps it is instructive to first understand how the human perceptual system learns to construct symbol-like representations starting from raw sensory inputs?

1.3 The Binding Problem

The question of how the human brains acquire and manipulate symbol-like entities has intrigued neuroscientists and psychologists alike who coined "the binding problem" to refer to the explanatory gap in our understanding of information processing in the brain. It refers to the difficulty to dynamically and flexibly combine information that is distributed throughout the network, which is required to effectively represent symbol-like entities. The binding problem occurs in several perceptual tasks including visual binding (color, shape, texture), auditory binding (cocktail-party problem [Cherry, 1953]), temporal binding (motion), cross-modal binding (sound and sight as a joint event), mind-body problem [von der Malsburg, 1995; Treisman, 1996; Roskies, 1999;

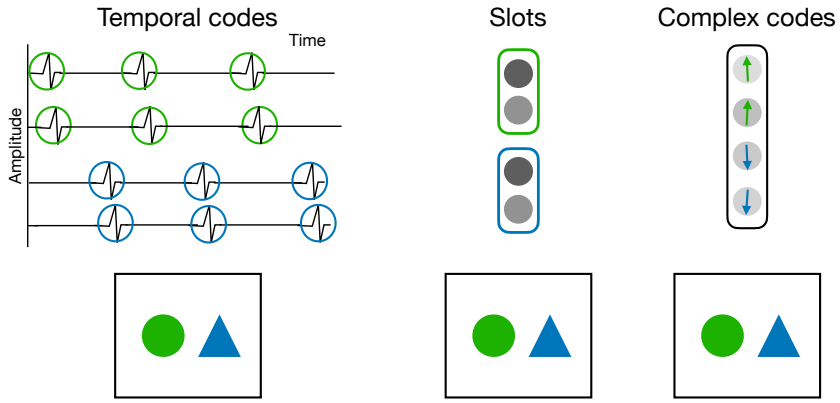


Figure 1.5: Illustration of different binding mechanisms and corresponding representational formats in biological and artificial NNs. Input image contains two objects, a green circle and a blue triangle. (left) spiking neurons using spike timing to indicate which features are bound together. (center) features are bound to separate slots using attention-like mechanisms. (right) neural synchrony implemented using complex-valued codes where phase components (indicated as directional vectors) play the role of spike timing in temporal codes.

Feldman, 2013].

Specifically, in the case of the visual binding which concerns with how the visual system integrates low-level features (color, shape, texture) to perceive whole objects. Grouping behavior was first studied and systematized by Gestalt psychologists [Wertheimer, 1923; Koffka, 1935; Köhler, 1967] who proposed a set of principles according to which perception operates, e.g., grouping by *proximity*, *similarity*, *closure*, *good continuation* and *common fate*. But, what are the computational mechanisms and internal representations used by our brains that give rise to such perceptual characteristics?

One prominent theory called the feature integration theory developed by Treisman and Gelade [1980] states that visual perception proceeds in two stages. In the first stage, called preattentive, basic features (colors, shape, movement) are automatically computed in parallel. In the second stage, called focused attention, combines the individual features into a *"master map"* to perceive the whole object. When attention is focused at a particular location on the "master map", the features in that position are attended to and are stored in *"object files"*. An alternative hypothesis, called the temporal correlation hypothesis [Milner,

1974; Grossberg, 1976; von der Malsburg, 1981; Singer and Gray, 1995], posits that the brain achieves visual binding using synchronized firing of neuronal spikes, as illustrated in Figure 1.5 for an image containing a green circle and a blue triangle. Experimental evidence that supports this hypothesis was first obtained by Gray and Singer [1989] by recordings from a cat visual cortex.

In recent years, perceptual grouping has been modeled in deep learning under the guise of *object-centric learning*. We refer readers to Greff et al. [2020] for an overview and taxonomy of methods in this framework. Such a two stage process for visual binding prescribed by feature integration theory has been implemented using artificial NNs by the so-called *slot-based models* where 'slots' play the role of *"object files"*. The general design template in these class of models first compute features in a parallel fashion from the visual input, i.e. feature maps from a given layer of a CNN. Slot-based models maintain the separation of information using separate pools of neurons ('slots') (as shown in Figure 1.5) but differ in the segregation mechanism to infer their contents. These segregation mechanisms are used to gather the appropriate subset of features into each 'slot' akin to the focused attention stage.

Alternate class of models, called *synchrony-based models*, for visual binding which operationalize the principle of synchronization as advocated by the temporal correlation hypothesis have received comparatively lesser attention within deep learning. We refer readers to McLelland and VanRullen [2016] for a taxonomy of models that employ neural synchrony to implement object-based attention mechanisms. Initial proposals to implement binding by synchrony using temporal codes include coupled-oscillator models by von der Malsburg and Schneider [1986]; Grossberg and Somers [1991]; von der Malsburg and Buhmann [1992] which segment toy stimulus patterns in 1D or 2D. The earliest demonstration of a synchrony-based model using complex codes was by Mozer et al. [1991] which focused on a supervised setting with teacher-specified target phases for hard-coded one-hot features of contour types. Later, Reichert and Serre [2014] initiated the revival of synchrony-based models by introducing a biologically motivated gating mechanism for adjusting layer-wise phases in an unsupervised setting.

The overarching goal of this dissertation is to emulate aspects of human perception such as saliency and perceptual grouping using connectionist models to bridge the gap in generalization, computational and data-efficiency between human perception and their artificial counterparts. To achieve this, we take

functional inspiration from human perception and cognition to design inductive biases, computational primitives or training objectives for NNs. Such an approach allows learning from raw sensory data to largely shape the kinds of symbols and relations inferred and combines the best design principles from the symbolic and connectionist traditions in AI.

1.4 Dissertation Overview and Contributions

In Chapter 2, we describe a novel approach to discover object keypoints that correspond to salient image regions based on the notion of *local spatial predictability* which draws functional inspiration from bottom-up models of salience in human perception. Our approach discovers object keypoints that are more robust to distractors and alleviates certain systematic biases prevalent in prior state-of-the-art models. We show how such object keypoints form a compressed spatial abstraction of images useful for sample-efficient learning of downstream tasks such as game-playing.

In Chapter 3, we develop a model for grouping behavioral sequences into modular chunks ('sub-routines') analogous to the process of grouping visual stimuli into objects. We draw an analogy between the spatial grouping of pixels into visual objects and temporal grouping To achieve such temporal grouping behavior with slot-based models, we develop new decoding and masking algorithms that enables each 'slot' to model a contiguous set of elements. Our novel slot-based Transformer model outperforms previous state-of-the-art models in grouping state-action sequences into modular 'sub-routines' and more gracefully handles sequences with variable number of 'sub-routines' as well.

The second half of the dissertation focuses on synchrony-based models for perceptual grouping. Synchrony-based models are NNs that compute with complex-valued activations and use the phase-components to store binding information. Chapter 4 presents a new architecture and contrastive training method for synchrony-based models Mozer et al. [1991]; Reichert and Serre [2014] leading to improvements in perceptual grouping of images and simultaneously represent a greater number of objects compared to previous state-of-the-art models. In Chapter 5, we design a new architecture starting from first-principles that greatly simplifies the inductive biases and training procedure for synchrony-based models and remedies certain systematic grouping errors of previous state-of-the-art models. Chapter 6 introduces a powerful generalization

to the popular RoPE scheme for relative positional encoding in Transformer models that decouples the matches from the 'what' and the 'where' information using complex-valued activations in polar form leading to improved sequence modeling capabilities across domains such as natural language, symbolic music and genomic data. Finally in Chapter 7, we outline some remaining research questions and promising future threads to explore.

1.5 List of Publications

Following is the list of publications presented in this dissertation that contribute towards the problem of learning symbol-like representations with NNs.

- **Anand Gopalakrishnan**, Sjoerd van Steenkiste and Jürgen Schmidhuber. "Unsupervised Object Keypoint Learning using Local Spatial Predictability.", *In Proceedings of International Conference on Learning Representations (ICLR), 2020*.

- **Anand Gopalakrishnan**, Kazuki Irie, Jürgen Schmidhuber and Sjoerd van Steenkiste. "Unsupervised Learning of Temporal Abstractions With Slot-Based Transformers.", *Neural Computation 35, pp. 593-626, 2022*.

- Aleksandar Stanić*, **Anand Gopalakrishnan***, Kazuki Irie and Jürgen Schmidhuber. "Contrastive Training of Complex-Valued Autoencoders for Object Discovery." *Advances in Neural Information Processing Systems 36 (NeurIPS), 2023*.

- **Anand Gopalakrishnan**, Aleksandar Stanić, Jürgen Schmidhuber, Michael Curtis Mozer. "Recurrent Complex-Weighted Autoencoders for Unsupervised Object Discovery" *Advances in Neural Information Processing Systems 37 (NeurIPS), 2024*.

- **Anand Gopalakrishnan**, Róbert Csordás, Jürgen Schmidhuber, Michael Curtis Mozer. "Polar Coordinate Position Embedding" *in submission for ICLR 2026; ArXiv preprint arXiv:2509.10534, 2025*.

During my PhD studies I have contributed towards research efforts on representation learning for music, scalable self-supervised methods, credit assignment methods for sequence models and harnessing a team of Foundation Models to solve vision and language-based tasks through debate. Publications

resulting from these research efforts lie outside the scope of this dissertation but have been listed below for completeness:

- Joonsu Gha, Vincent Herrmann, Benjamin F. Grewe, Jürgen Schmidhuber and **Anand Gopalakrishnan**. "Unsupervised Musical Object Discovery from Audio." *Machine Learning for Audio Workshop, NeurIPS 2023*.

- Kazuki Irie, **Anand Gopalakrishnan** and Jürgen Schmidhuber. "Exploring the Promise and Limits of Real-Time Recurrent Learning." *In Proceedings of International Conference on Learning Representations (ICLR), 2024*.

- Etai Littwin, Vimal Thilak and **Anand Gopalakrishnan**. "Enhancing JEPAs with Spatial Conditioning: Robust and Efficient Representation Learning." *Workshop on Self-supervised Learning: Theory and Practice, NeurIPS 2024*.

- Mingchen Zhuge, Haozhe Liu, Francesco Faccio, Dylan R. Ashley, Róbert Csordás, **Anand Gopalakrishnan**, Abdullah Hamdi, Hasan Hammoud, Vincent Herrmann, Kazuki Irie, Louis Kirsch, Bing-chuan Li, G. Li, Shuming Liu, Jinjie Mai, Piotr Pikekos, Aditya Ramesh, Imanol Schlag, Weimin Shi, Aleksandar Stanić, Wenyi Wang, Yu Han Wang, Mengmeng Xu, Deng-Ping Fan, Bernard Ghanem and Jürgen Schmidhuber. "Mindstorms in Natural Language-Based Societies of Mind." *In Computational Visual Media, vol. 11, no. 1, pp. 29-81, 2025*.

Chapter 2

Keypoint Discovery with Local Spatial Predictability[1]

2.1 Overview

An intelligent agent situated in the visual world critically depends on a suitable representation of its incoming sensory information. For example, a representation that captures only information about relevant aspects of the world makes it easier to learn downstream tasks efficiently [Barlow, 1989; Bengio et al., 2013]. Similarly, when explicitly distinguishing abstract concepts, such as objects, at a representational level, it is easier to generalize (systematically) to novel scenes that are composed of these same abstract building blocks [Lake et al., 2017; van Steenkiste et al., 2019; Greff et al., 2020].

In recent work, several methods have been proposed to learn unsupervised representations of images that aim to facilitate agents in this way [Veerapaneni et al., 2019; Janner et al., 2019]. Of particular interest are methods based on learned *object keypoints* that correspond to highly informative (salient) regions in the image as indicated by the presence of object parts [Zhang et al., 2018b; Jakab et al., 2018; Kulkarni et al., 2019; Minderer et al., 2019]. Many real world tasks primarily revolve around (physical) interactions between objects and agents. Therefore it is expected that a representation based on a set

[1] For the full paper please see "Unsupervised Object Keypoint Learning using Local Spatial Predictability" [Gopalakrishnan et al., 2021]

of task-agnostic object keypoints can be re-purposed to facilitate downstream learning (and generalization) on many different tasks [Lake et al., 2017].

One of the main challenges for learning representations based on object keypoints is to discover salient regions belonging to objects in an image without supervision. Recent methods take an information bottleneck approach, where a neural network is trained to allocate a fixed number of keypoints (and learn corresponding representations) in a way that helps making predictions about an image that has undergone some transformation [Jakab et al., 2018; Minderer et al., 2019; Kulkarni et al., 2019]. However, keypoints that are discovered in this way strongly depend on the specific transformation that is considered and therefore lack generality. For example, as we will confirm in our experiments, the recent *Transporter* [Kulkarni et al., 2019] learns to prioritize image regions that change over time, even when they are otherwise uninformative. Indeed, when relying on *extrinsic* object properties (i.e. that are not unique to objects) one becomes highly susceptible to distractors as we will demonstrate.

In this work, we propose *PermaKey* a novel representation learning approach based on object keypoints that does not overly bias keypoints in this way. The key idea underlying our approach is to view objects as local regions in the image that have high internal predictive structure (self-information). We argue that local predictability is an intrinsic property of an object and therefore more reliably captures *objectness* in images [Alexe et al., 2010]. This allows us to formulate a *local spatial prediction problem* to infer which of the image regions contain object parts. We perform this prediction task in the learned feature space of a convolutional neural network (CNN) to assess predictability based on a rich collection of learned low-level features. Using *PointNet* [Jakab et al., 2018] we can then convert these predictability maps to highly informative object keypoints.

We extensively evaluate our approach on a number of Atari environments and compare to *Transporter* [Kulkarni et al., 2019]. We demonstrate how our method is able to discover keypoints that focus on image regions that are unpredictable and which often correspond to salient object parts. By leveraging local predictability to learn about objects, our method profits from a simpler yet better generalizable definition of an object. Indeed, we demonstrate how it learns keypoints that do not solely focus on temporal motion (or any other extrinsic object property) and is more robust to uninformative (but predictable) distractors in the environment, such as moving background. On Atari

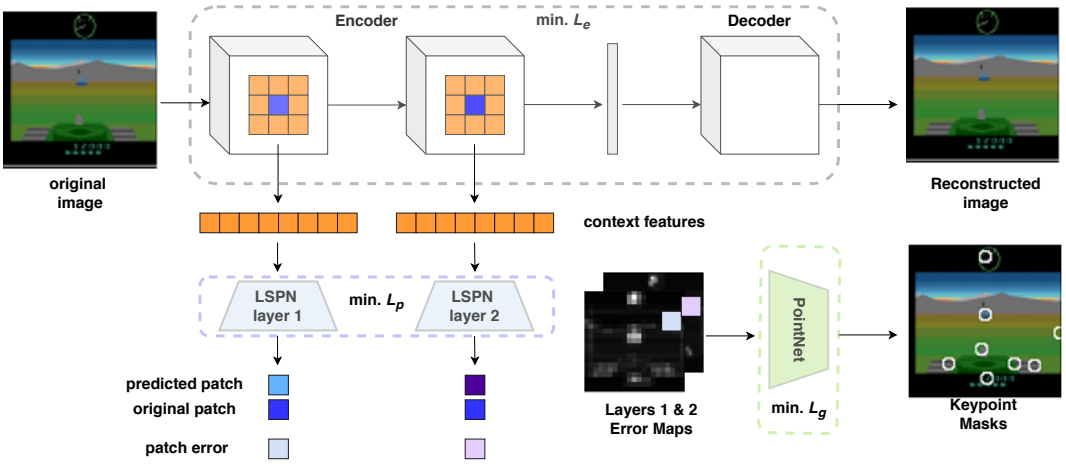


Figure 2.1: PermaKey consists of three modules (encapsulated by the dotted lines): learning a suitable spatial feature embedding (1), solving a local spatial prediction task (2), and converting error maps to keypoints (3). Objective functions used to learn each of the 3 modules shown within dotted blocks.

games, agents equipped with our keypoints outperform those using *Transporter* keypoints. Our method shows good performance even on challenging environments such as *Battlezone* involving shifting viewpoints where the *Transporters'* explicit motion bias fails to capture any task-relevant objects. As a final contribution, we investigate the use of graph neural networks [Battaglia et al., 2018] for processing keypoints, which potentially better accommodates their discrete nature when reasoning about their interactions, and provide an ablation study.

2.2 Method

At a high level, we define objects as abstract patterns in the visual input that can serve as *modular* building blocks (i.e. they are self-contained and reusable independent of context) for solving a particular task, in the sense that they can be separately intervened upon or reasoned with [Greff et al., 2020]. This lets us treat objects as *local regions* in input space that have high internal predictive structure based on statistical co-occurrence of features such as color, shape, etc. across a large number of samples. Hence, our focus is on their local predictability, which can be viewed as an "intrinsic" object property according to this definition. For example, Bruce and Tsotsos [2005] have previously shown that local regions with high self-information typically correspond to

salient objects. More generally, self-information approximated via a set of cues involving center-surround feature differences has been used to quantify *objectness* [Alexe et al., 2010].

In this work we introduce *Prediction ERror MAp based KEYpoints (PermaKey)*, which leverages this definition to learn about object keypoints and corresponding representations. The main component of *PermaKey* is a *local spatial prediction network* (LSPN), which is trained to solve a local spatial prediction problem in feature space (light-blue trapezoid in Figure 2.1). It involves predicting the value of a feature from its surrounding neighbours, which can only be solved accurately when they belong to the same object. Hence, the error map (predictability map) that is obtained by evaluating the LSPN at different locations carves the feature space up into regions that have high internal predictive structure (see rows 4 & 5 in Figure 2.3a). Retaining only such locally unpredictable features in the representation is reminiscent of redundancy reduction that occurs in early visual processing as posited by the efficient coding hypothesis [Barlow et al., 1961; Field, 1987]. In what follows, we delve into each of the 3 modules that constitute our *PermaKey* system: 1) the VAE to learn spatial features 2) the LSPN to solve the local spatial prediction task and 3) PointNet to group error maps into keypoints (see Figure 2.1 for an overview).

Spatial Feature Embedding To solve the local spatial prediction task we require an informative set of features at each image location. While prior approaches focus on low-level features of image patches (eg. RGB values Isola et al. [2014], or ICA features Bruce and Tsotsos [2005]), we propose to learn features using a Variational Auto-Encoder (VAE; Kingma and Welling [2014]). It consists of an *Encoder* that parametrizes the approximate posterior $q_\phi(z|x)$ and a *Decoder* that parametrizes the generative model $p_\theta(x_i|z)$, which are trained to model the observed data via the ELBO objective:

$$L_e(\theta, \phi) = \mathbb{E}_{q_\phi(z|x)} \log[p_\theta(x|z)] - \mathrm{D}_{\mathrm{KL}}[q_\phi(z|x)||p(z)]. \tag{2.1}$$

The encoder is based on several layers of convolutions that offer progressively more abstract image features by trading-off spatial resolution with depth. Hence, which layer(s) we choose as our feature embedding will have an effect on the outcome of the local spatial prediction problem. While more abstract high-level features are expected to better capture the internal predictive structure of an object, it will be more difficult to attribute the error of the prediction network to the exact image location. On the other hand, while more low-level features can be localized more accurately, they may lack the expressiveness to capture

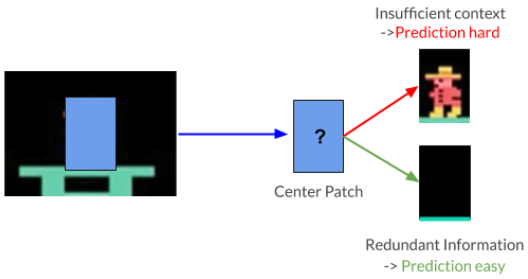


Figure 2.2: Two different prediction scenarios encountered when attempting to solve the LSP task. In the first case, too little context information is available to predict the center patch from its surroundings, likely yielding a prediction error (assuming the character is not *always* present). Alternatively, when sufficient context is available as in the second case, prediction becomes easy.

high-level properties of objects. Nonetheless, in practice we find that a spatial feature embedding based on earlier layers of the encoder works well (see also Section 2.3 for an ablation).

Local Spatial Prediction Task Using the learned spatial feature embedding we seek out salient regions of the input image that correspond to object parts. Our approach is based on the idea that objects correspond to local regions in feature space that have high internal predictive structure, which allows us to formulate the following local spatial prediction (LSP) task. For each location in the learned spatial feature embedding, we seek to predict the value of the features (across the feature maps) from its neighbouring feature values. When neighbouring areas correspond to the same object-(part), i.e. they regularly appear together, we expect that this prediction problem is easy (green arrow in Section 2.2). In contrast, this is much harder when insufficient context is available (red arrow in Section 2.2), or when features rarely co-occur. Similarly, when parts of an image remain fixed across a large number observations it is easily predictable regardless of how much context is available.

We implement the local spatial prediction task by training a neural network (LSPN in Figure 2.1) to predict a center patch of features from the 8 first-order neighbouring patches using the learned spatial feature embedding at layer l of the VAE encoder. The weights of this network are trained to solve this prediction problem across images and across feature locations using the

following objective:

$$L_p(\psi_l) = \frac{1}{NHW} \sum_x \sum_{i,j}^{H,W} (f_{\psi_l}^{\text{LSPN}}(A[ne(i,j)]) - A[i,j])^2, \tag{2.2}$$

where $A \in \mathbb{R}^{H \times W \times C}$ is the set of feature maps at layer l of the encoder $q_\phi(z|x)$, and $ne(i,j)$ are the coordinates of the first-order neighbours surrounding i,j and H,W are the height and width of the spatial feature maps respectively. The features of the neighbouring locations are concatenated before being fed to the network. We train a separate LSPN for each of the chosen feature embedding specified by the encoder layer(s) and record their error at each location to obtain *predictability maps* that are expected to indicate the presence of objects (rows 4 & 5 in Figure 2.3a).

Extracting Keypoints from Error Maps The predictability maps can be thought of as being a superposition of the local predictability errors due to each of the individual objects in the scene. Hence, what remains is to extract the k most salient regions (belonging to object parts) as our object keypoints from these predictability maps. We do so by training a *PointNet* [Jakab et al., 2018] to reconstruct the error maps through a bottleneck consisting of k fixed-width 2D Gaussian windows with learnable centers μ that are the keypoints (L_g loss in Figure 2.1) as shown below:

$$L_g(\omega) = \frac{1}{MHW} \sum_m^M \sum_{i,j}^{H,W} (f_\omega^{\text{PointNet}}(\mathcal{H}[m,i,j]) - \mathcal{H}[m,i,j])^2. \tag{2.3}$$

Here $\mathcal{H} \in \mathbb{R}^{M \times H \times W}$ is the concatenation of layer-wise LSP error maps for various encoder layer(s) l (M in total), and H,W are the height and width of the error maps (potentially up-sampled). Since the bottleneck layer has only limited capacity it is forced to consider locations that are most helpful in minimizing reconstruction error. This also requires the *PointNet* to consider statistical regularities across the predictability maps that can efficiently be described. In that sense this procedure can be thought of as a way of (spatially) clustering the local errors belonging to the same object by placing keypoints.

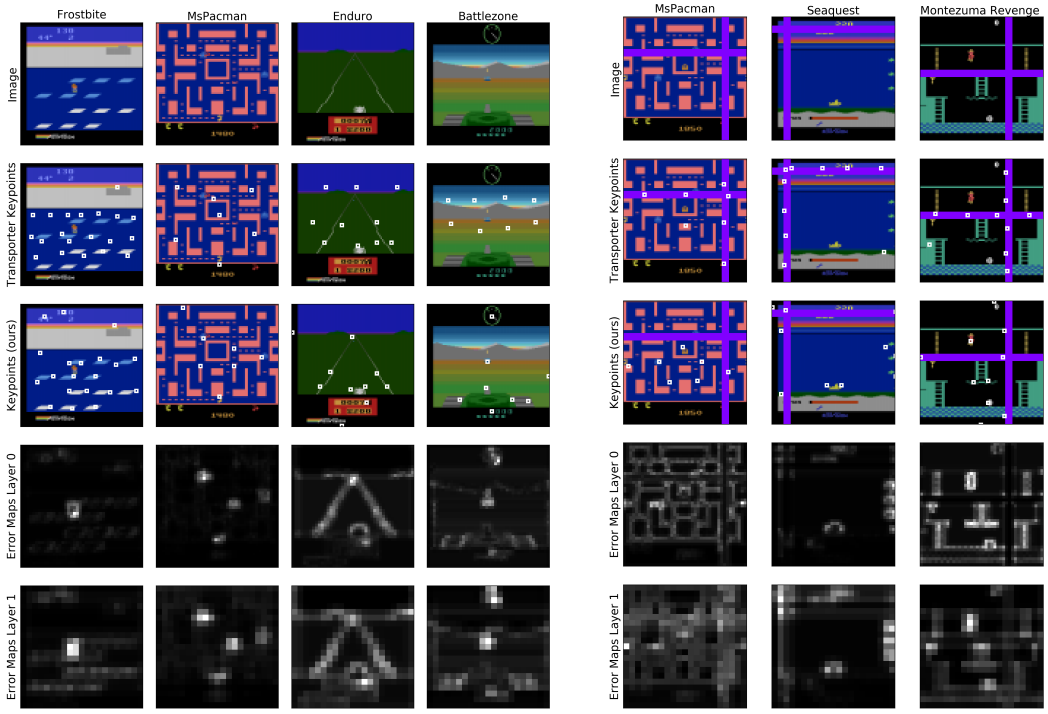
Using Keypoints for Downstream Tasks Given a set of keypoint locations and their associated features learned in an purely unsupervised framework we examine how to incorporate them into a state representation suitable to solve a

set of downstream RL tasks. Here we will consider Atari games as an example, which essentially consist of entities such as a player, enemies, reward objects like health potions, food etc. Player(s) obtain positive reward through collecting these reward objects or killing enemies and in turn risk getting attacked by these enemies. Successful game-play in such environments would therefore require an understanding of these interaction effects between player, enemies and reward objects in order to estimate the best next set of actions to take to maximize the game score.

Prior work like the *Transporter* [Kulkarni et al., 2019] uses a CNN to encode features of object keypoints for downstream RL tasks on the Atari domain. However, a CNN permits an implicit encoding of relational factors between keypoints only in a limited sense since it measures interactions between spatially close keypoints through the weight-sharing scheme of the convolutional kernel. An alternative is to use a graph neural network (GNN) [Scarselli et al., 2009; Battaglia et al., 2018] (see also Pollack [1990]; Küchler and Goller [1996] for earlier approaches) which have been applied to model relational inductive biases in the state representations of deep RL agents [Zambaldi et al., 2019]. In a similar spirit, we consider treating keypoints as an unordered set and using a GNN to explicitly encode relational factors between pairs of such entities. In this case, we spatially average the convolutional features within each keypoint mask to obtain a keypoint feature. These are appended with learned positional embedding and initialized as node values of the underlying graphs. Underlying graphs are fully-connected and edge factors initialized with zeros. We use the *Encode-Process-Decode* design strategy [Battaglia et al., 2018] for the graph keypoint encoder in our PKey+GNN model for Atari in Section 2.3. The *Encode* and *Decode* blocks independently process nodes and edges of underlying graphs. We use an *Interaction Network* [Battaglia et al., 2016] for the *Process* block with a single message-passing step to compute updated edges e' and nodes v' as shown below:

$$e'_{ij} = f^e([v_i, v_j, e_{ij}]), \qquad \overline{e}'_i = \rho^{e\to v}(E'_i), \qquad v'_i = f^v([v_i, \overline{e}'_i]). \tag{2.4}$$

Here e_{ij} is the edge connecting nodes v_i and v_j, f^e is the edge update function, $\rho^{e\to v}$ is the edge aggregation function, E'_i represents all incoming edges to node v_i and f^v is the node update function. After one round of message-passing by the *Interaction Network* the resultant nodes are decoded as outputs. These output node features of all nodes are then concatenated and passed through an MLP before being input to the agent. For further implementation details we refer to Appendix A.2.



(a) Atari *(b)* Noisy Atari

Figure 2.3: Image (row 1), *Transporter* keypoints overlaid on image (row 2), *PermaKey* (ours) keypoints overlaid on image (row 3), predictability maps from feature layers 0 and 1 (rows 4 & 5 respectively).

2.3 Results

We empirically evaluate the efficacy of our approach on a subset of Atari games [Bellemare et al., 2013] that ensure a good variability in terms of object sizes, colour, types, counts, and movement. As a baseline, we compare against the recently proposed *Transporter* [Kulkarni et al., 2019], which was previously evaluated on Atari. To evaluate the efficacy of object keypoints to facilitate sample-efficient learning on RL tasks we train agents on the low-data regime of 100,000 interactions following the same experimental protocol as in Kulkarni et al. [2019]; Kaiser et al. [2020]. We use a deep recurrent Q-learning variant [Hausknecht and Stone, 2015] using double-Q learning [Hasselt et al., 2016], target networks [Mnih et al., 2015] and a 3-step return as our RL algorithm for all agents with a window size of 8 and batch size of 16. To implement the Q-function we use a 128 unit LSTM network [Hochreiter and Schmidhuber, 1997a; Gers et al., 1999]. Full experimental details can be

found in Appendix A.2 and additional results in Appendix A.3[2].

Atari We train *PermaKey* on Atari and find that it is frequently able to recover keypoints corresponding to object parts. For example, in Figure 2.3a (row 3) it can be seen how our method discovers keypoints correponding to the road and traffic in *Enduro* (column 3) and player and enemy tank parts (eg. wheels, cannon) in *Battlezone* (column 4). On *Frostbite* we find that it learns about the borders of the ice platforms, the player, scoreboard and birds, while on *MsPacman* we observe that it tracks the various characters like the blue ghosts and Pac-Man. We emphasize that while the precise location of the keypoints often seems to focus on the *borders* of predictability, their associated window of attention (i.e. in *PointNet*) is able to capture most of the object part.

This is better seen in Figure 2.4 (and in Figure A.5 in Appendix A.3), where the corresponding windows of attention (available to an agent) are shown. From the error maps (rows 4 and 5 in Figure 2.3a) it can be seen how the LSPN implicitly learns to discover salient parts of the image, and how the choice of feature embedding shifts the focus from local edges to entire objects. In Figure 2.5 (and Figures A.3 and A.4) it is shown how *PermaKey* is stable across multiple different runs.

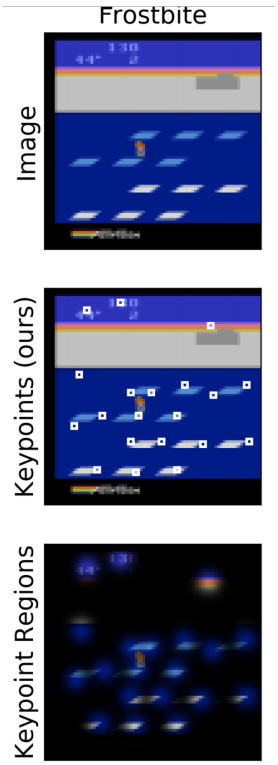


Figure 2.4: Keypoint regions.

Compared to *Transporter* (Figure 2.3a row 2) we find that our approach often produces qualitatively better keypoints. For example, on *Enduro* and *Battlezone* the *Transporter* fails to place keypoints on salient object parts belonging to the road and traffic or parts of the player and enemy tank (eg. its canon, wheels etc.). This is intuitive, since in these environments moving image regions do not always correspond to objects. Indeed, on *Frostbite* and *MsPacman*, where this is not the case, the keypoints produced by *Transporter* and our method are of comparable quality.

In Table 2.1 we provide quantitative results of training agents that use these keypoints on 4 Atari environments. We observe that agents using *PermaKey* keypoints

[2]Code repository for the paper: `https://github.com/agopal42/permakey`.

(PKey-CNN in Table 2.1) outperform corresponding variants using *Transporter* keypoints (original from Kulkarni et al. [2019] and our re-implementation) on 3 environments[3]. Notice how on *Battlezone* we were unable to train a policy that is better than random when using *Transporter* keypoints. This is intuitive, since in Figure 2.3a we observed how *Transporter* learns keypoints that are not concerned with salient objects on this environment. For comparison, we also include scores from Kaiser et al. [2020] in Table 2.1, where a similar experimental setup is used. It can be seen that SimPLE [Kaiser et al., 2020], Rainbow [Hessel et al., 2018] and PPO [Schulman et al., 2017] also achieve worse scores than our PKey agents.

When using a GNN for downstream processing (PKey + GNN) we are able to obtain additional improvements on the *Battlezone* and *Frostbite* environments. Meanwhile on *MsPacman* and *Seaquest* performance drops, indicating that the stronger spatial inductive bias (i.e. nearby keypoints are processed similarly) implemented by CNNs can be advantageous as well. Note that the relative improvements of (Transporter + GNN) compared to (Transporter + CNN), largely mirrors the relative changes shown by analogous *PermaKey* variants (CNN vs GNN keypoint encoder) across all environments. Further, it can be seen how *PermaKey* consistently outperforms *Transporter*, even when using GNN keypoint encoders for both. On *MsPacman* we observed a larger standard-deviation for *Transporter*, compared to the results reported by Kulkarni et al. [2019], although we did tune the hyper-parameters for our re-implementation separately. A possible reason could be due to us using a lower batch size of 16 (compared to 32) and a lower window size of 8 (compared to 16). We were unable to investigate this further due to constraints on the available amount of compute. On Frostbite, we observe a large variances for both (PKey + GNN) and (Transporter + GNN) agents as some policies complete the first level (resulting in very high scores of about 2000 points), while others learn sub-optimal strategies yielding episodic returns around 220.

Robustness to Distractors In order to demonstrate the limitations of the explicit motion bias in *Transporter* we synthesized a modified version of Atari ("noisy" Atari) that contains a coloured strip (either horizontal, vertical or both) at random locations in the environment (see row 1 in Figure 2.3b), thereby creating

[3] Unfortunately, Kulkarni et al. [2019] have not open-sourced their code (needed to reproduce RL results and apply *Transporter* on other environments) and their self-generated dataset containing 'ground-truth' keypoints for evaluation on Atari. We were able to obtain certain hyper-parameters through personal communication.



Figure 2.5: Observed *Enduro* frames (odd columns) and with overlaid keypoints from 5 random seeds having a unique color (even columns). The inferred keypoints are stable across different runs and consistently focus on salient object parts. Additional frames can be seen in Figure A.3 in Appendix A.3.

Game	Rainbow	PPO	SimPLE	Transporter (orig.)	Transporter (re-imp.)	Transporter -GNN	PKey-CNN (ours)	PKey-GNN (ours)
Battlezone	3363.5 (523.8)	5300.0 (3655.1)	5184.4 (1347.5)	N/A	N/A	N/A	10266.7 (3272.1)	**12566.7** (3297.9)
MsPacman	364.3 (20.4)	496.0 (379.8)	762.8 (331.5)	999.4 (145.4)	983.0 (806.3)	838.3 (537.3)	**1038.5** (417.1)	906.3 (382.2)
Seaquest	206.3 (17.1)	370.0 (103.3)	370.9 (128.2)	236.7 (22.2)	455.3 (160.6)	296.0 (82.5)	**520.0** (159.9)	375.3 (75.4)
Frostbite	140.1 (2.7)	174.0 (40.7)	254.7 (4.9)	388.3 (142.1)	263.7 (14.9)	465.0 (442.1)	310.7 (150.1)	**657.3** (556.8)

Table 2.1: Atari Results: mean scores (and std. dev.) obtained for each agent. For PKey and Transporter (re-imp.) we report the mean for 3 different policies (using the median evaluation run over 10 environment seeds). Results for Transporter (orig.) are taken from Kulkarni et al. [2019] and for SimPLE, PPO, and Rainbow from [Kaiser et al., 2020], where a similar experimental set-up is used. "N/A" indicates that a policy did not make any learning progress w.r.t a random policy.

the illusion of motion. Figure 2.3b (row 3) demonstrates how our method is more resistant against this distractor by focusing on predictability and is able to recover keypoints belonging to object parts as before (eg. player, skull, ladder positions in *Montezuma's Revenge*). Indeed, notice how the distractor does not show up in the error maps (rows 4 and 5 in Figure 2.3b) as it corresponds to a predictable pattern that was frequently encountered across the entire dataset[4]. In contrast, we find that *Transporter* (row 2) dedicates many of the available keypoints to the distractor as opposed to the static salient object parts.

[4]This observation also confirms that the LSPN does not collapse to a naive color thresholder.

Game	Transp. (re-imp.)	PKey + CNN
Seaquest	357.3 (99.10)	**562.0** (114.78)
MsPacman	470.0 (212.90)	**574.7** (290.07)

Table 2.2: "Noisy" Atari Results: mean (and std. dev.) for 3 different policies (using the median over environment seeds as before).

As a quantitative experiment, we consider the downstream performance of agents that receive either *Transporter* keypoints or *PermaKey* keypoints on "Noisy" versions of *Seaquest* and *MsPacman*. In Table 2.2 it can be seen how the performance of the *PermaKey* agent (PKey + CNN) remains largely unaffected on *Seaquest* (compared to Table 2.1), while the sensitivity of *Transporter* to such moving distractors causes a significant drop in performance. On "Noisy" *MsPacman*, both agents perform worse compared to before, but the *PermaKey* only to a lesser extent. We hypothesize that the general performance drop is caused by the distractor bars occluding the fruits (collecting fruits yields rewards) located along the horizontal and vertical columns at every timestep. In general, we observe that *PermaKey* consistently outperforms *Transporter* on these tasks.

We perform ablation studies that inspect the qualitative and quantitative effects of the number of keypoints and choice of feature layer for the local spatial prediction task on our overall system below.

Number of keypoints We gain additional insight in our approach by modulating the number of keypoints and observe its qualitative effects. On the Frostbite environment we use $15, 20, 25$ keypoints (shown in Figure A.1), while keeping all the other hyperparameters same as described in Appendix A.2. When using additional keypoints we observe that *PermaKey* prioritizes regions in the error map that contribute most to the reconstruction loss. On the other hand, when the number of keypoints is too large, redundant keypoints are placed at seemingly random locations e.g. the border of the image.

MsPacman	5 keypoints	7 keypoints	10 keypoints
Transp. (re-imp.)	923.0 (433.95)	983.0 (806.3)	907.0 (317.41)
PKey + CNN	**1004.3** (319.15)	**1038.5** (417.1)	**1003.3** (313.07)

Table 2.3: Keypoint ablation on the *MsPacman* env.

We also quantitatively measure the effects of varying the number of keypoints $\{5, 7, 10\}$ produced by *PermaKey* and *Transporter* on downstream performance in *MsPacman* (Table 2.3), using the same evaluation protocol as before. The agent using *PermaKey* outperforms the one using *Transporter* keypoints across the entire keypoint range. Since *Transporter* keypoints are biased towards image regions that change across frames, additional keypoints only seem to track the different characters that move around the maze. With *PermaKey* focusing on predictability, additional keypoints tend to capture other salient parts of the maze, although this did not yield improved downstream performance in this case.

Spatial resolution of the feature embedding We experiment with the choice of feature layer(s) used for the local spatial prediction task and observe its qualitative effects on the keypoints discovered. On the Space Invaders environment we use the following sets of feature layer(s) $= \{(0), (0, 1), (2, 3), (0, 1, 2, 3)\}$ and retain the same values for all the other hyperparameters as described in Appendix A.2. Results are shown in Figure A.2 in Appendix A.3. When only the lower layers are used, we find that the corresponding error maps tend to focus more on highly local image features. In contrast, when only higher layers are used, we find that the loss of spatial resolution leads to highly coarse keypoint localization. In practice, we found that using feature layers 0 and 1 offers the right balance between spatial locality and expressiveness of features.

The ice floats in *Frostbite*, which are captured as keypoints (Figure 2.3a), serve as a useful example that illustrates the importance of integrating multiple spatial resolutions (i.e. that trade off local and global information as also noted in Itti et al. [1998]; Jagersand [1995]). Ice floats seen at the default image resolution (layer-0 with 'conv' receptive field = 4) might seem very predictable in it's highly local context of other ice float parts. However, casting the same local prediction task of predicting the ice float features at layer-1 (image resolution 0.5x due to stride = 2 in conv-encoder) given only blue background context would lead to some error spikes. This is because at many locations in the image, it seems reasonable to expect blue background patches to surround other blue background patches. Only at certain locations white ice floats are interspersed with blue background patches thereby driving some amount of error (surprise) in local predictability (In Figure 2.3a column 1, compare rows 4&5). Hence, varying the spatial resolution of the feature embedding and integrating the corresponding error maps lets us accommodate objects of varying shapes, sizes

and geometry etc.

2.4 Related Work

Inferring object keypoints offers a flexible approach to segmenting a scene into a set of informative parts useful for downstream processing. Recent approaches [Jakab et al., 2018; Zhang et al., 2018b; Kulkarni et al., 2019; Minderer et al., 2019] apply a transformation (i.e. rotation, deformation, viewpoint shift or temporal-shift) on a source image to generate a target image. An autoencoder network with a keypoint-bottleneck layer is trained to reconstruct the target image given the source image and thereby learns to infer the most salient image regions that differ between source and target image pairs [Jakab et al., 2018]. However, since the initial transformation does not focus on intrinsic object properties, they are more susceptible to 'false-positives' and 'false-negatives'. In extreme scenarios, such approaches may even fail to infer any useful object keypoints at all (e.g. as shown in fig. 2.3b).

The more general framework of discovering salient image regions [Itti et al., 1998] is closely related to inferring object keypoints. Early work explored the connections between information theoretic measures of information gain or local entropy [Kadir and Brady, 2001; Jagersand, 1995] and its role in driving fixation(attention) patterns in human vision. For example, Bruce and Tsotsos [2005] use Shannon's self information metric as a measure of saliency, while Zhmoginov et al. [2019] leverage mutual information to guide saliency masks. Our prediction objective accounts for both: the LSP account for local saliency, while more global saliency is achieved having the prediction network solve this local prediction problem across an entire dataset of images. In this way, it can also reason about statistical regularities that only occur at this more global level.

An alternative to inferring object keypoints (and corresponding representations) is to learn entire object representations. Recent approaches can be broadly categorized into spatial mixture models [Greff et al., 2017, 2019; Burgess et al., 2019], sequential attention models [Eslami et al., 2016; Kosiorek et al., 2018] or hybrid models that are combinations of both [Lin et al., 2020]. While these methods have shown promising results, they have yet to scale to more complex visual settings. In contrast, object keypoints focus on more low-level information (i.e. object parts) that can be inferred more easily, even in complex visual settings [Minderer et al., 2019] Recent work on object detection [Duan et al.,

2019; Zhou et al., 2019] have explored the use of a center point (i.e. keypoint) and a bounding box as an object abstraction. However, they use ground-truth bounding boxes of objects to infer keypoint locations whereas our method is fully unsupervised. Alternatively Isola et al. [2015] propose a self-supervised framework for instance segmentation where local image patches are classified as belonging together based on their spatial or temporal distance.

Graph neural networks (GNNs) have been previously used for intuitive physics models [Battaglia et al., 2016; Chang et al., 2017; van Steenkiste et al., 2018; Janner et al., 2019; Veerapaneni et al., 2019; Kipf et al., 2020; Stanić et al., 2021]. Other work instantiate the policy [Wang et al., 2018] or value function [Bapst et al., 2019] as GNNs for RL tasks on continuous control or physical construction domains. However, the latter assume access to underlying ground-truth object(-part) information. In contrast, we learn our keypoint representations in a fully unsupervised manner from raw images. Further, we do not instantiate the policy or value function as a GNN, but rather use it to model relational factors between learned keypoint for downstream RL tasks in a similar spirit to Zambaldi et al. [2019].

2.5 Discussion

We have seen how *PermaKey*, by focusing on local *predictability*, is able to reliably discover salient object keypoints while being more robust to certain distractors compared to *Transporter* [Kulkarni et al., 2019]. However, it should be noted that *PermaKey* can only account for a "bottom-up" notion of saliency [Itti, 2007] based on the observed inputs, and does not consider task-dependent information. Therefore a potential limitation of this approach is that it might capture objects that are not relevant for solving the task at hand (i.e. corresponding to "unpredictable distractors"). On the other hand, a purely unsupervised approach to extracting keypoints may allow for greater re-use and generalizability. The same (overcomplete) set of keypoints can now be used to faciliate a number of tasks, for example by querying (and attending to) only the relevant keypoints using top-down attention [Mott et al., 2019; Mittal et al., 2020]. In this way it may readily facilitate other tasks, such as a "new object" acting as the key in *Montezuma's Revenge*, without having to re-train the bottom-up vision module.

Other kinds of "unpredictable distractors" could include certain types of image

noise having highly unpredictable local structure, such as salt-and-pepper noise. It is unclear how this affects the local spatial prediction problem, since the learned spatial embedding of the VAE may readily suppress such information (due to it being trained for reconstruction). In that case, since *PermaKey* acts on these learned spatial embeddings, it could leave the quality of error map(s) relatively unchanged. Further, we note that such distractors are typically the result of artificial noise and uncommon in the real world, unlike distractors due to motion, which are therefore more important to handle effectively.

In this work, we have trained *PermaKey* in a purely unsupervised fashion by pre-training it once on a dataset of collected episodes of game-play. On games where new objects are introduced only at much later levels it would be beneficial to continue refining the keypoint module using the unsupervised loss [Kulkarni et al., 2019]. Further, it may be interesting to consider whether there are benefits to learning both the keypoint module and policy network in an end-to-end fashion. While this is expected to reduce the re-use and generalizability of the discovered keypoints, it may make it easier to learn relevant keypoints due to directly considering task-specific information (e.g. via policy gradients).

2.6 Conclusion

We proposed *PermaKey*, a novel approach to learning object keypoints (and representations) that leverages *predictability* to identify salient regions that correspond to object parts. Through extensive experiments on Atari it was shown how our method is able to learn accurate object keypoints that are more robust to distractors, unlike the *Transporter* baseline approach. RL agents that employ *PermaKey* keypoints as input representations show sample-efficient learning on several Atari environments and outperform several other baselines. Further, it was shown how additional improvements can sometimes be obtained when processing keypoints using graph neural networks. Future work includes scaling *PermaKey* to more complex 3D visual worlds involving changing viewpoints, occlusions, egocentric vision etc. Similarly, designing quantitative evaluation metrics to directly measure the quality of keypoints extracted by different methods without the need for computationally expensive downstream tasks would allow for efficient performance analysis and model design as we begin to scale-up these ideas to more complex visual domains.

Chapter 3

Learning Temporal Abstractions with Slot-based Transformers[1]

3.1 Overview

An intelligent goal-seeking agent situated in the real world should make decisions along various timescales to act and plan efficiently. A natural approach that facilitates this behavior is via a divide-and-conquer strategy, where goals are decomposed along sub-goals, and solutions to such sub-goals (i.e. 'correct' sequences of actions) are stored as reusable 'primitive' *sub-routines* [Schmidhuber, 1991; Dayan and Hinton, 1992; Bakker et al., 2004; Schmidhuber, 1990]. Viewing complex goal-directed behavior as (novel) compositions of known sub-routines simplifies both decision-making and planning [Schmidhuber and Wahnsiedler, 1993; Tadepalli and Dietterich, 1997], while also benefiting out-of-distribution generalization [Peng et al., 2019].

The options framework [Sutton et al., 1999] introduced a design strategy for the hierarchical organization of behavior within the context of reinforcement learning. Crucial to its success is the quality of each 'option' (or sub-routine) in terms of the level of *modularity* and *reusability* it offers. Consider for example, the task of planning a travel itinerary to go from London to New York. Useful

[1] For the full paper please see "Unsupervised Learning of Temporal Abstractions with Slot-based Transformers" [Gopalakrishnan et al., 2023]

sub-routines in this context may include purchasing flight tickets, navigating to the airport, or boarding the flight. This factorization into sub-routines is desirable as these sub-routines capture mostly *self-contained* activities. Thus they are far more likely to apply to other travel plans between two different cities, which may again involve taking flights, etc.

Prior approaches propose to address this issue by learning about useful sub-routines directly from data [Andreas et al., 2017; Shiarlis et al., 2018; Kipf et al., 2019; Lu et al., 2021]. Of particular interest is the setting where the model is only given access to state-action trajectories from an (expert) policy and learns in an unsupervised manner (while the ground-truth number of sub-routines are assumed to be known). Two relevant works can be distinguished: CompILE [Kipf et al., 2019] and Ordered Memory Policy Network (OMPN) [Lu et al., 2021]. In CompILE, expert trajectories are modeled using a latent variable model based on a recurrent neural network (RNN), which infers a pre-determined number of boundary points by iteratively processing the entire trajectory multiple times to recover segments associated with reusable sub-routines. Alternatively, OMPN equips an RNN with a multi-layer hierarchical memory, where information processing at different levels in the memory can be interpreted as belonging to separate segments. While OMPN additionally includes a strong preference for capturing hierarchical relationships between different sub-routines, both methods are limited insofar that they process the trajectory in an entirely *sequential* manner. This prevents them from revising earlier decisions about boundary points in light of new information that becomes only accessible at a future stage. Moreover, iteratively processing the sequence multiple times (as in CompILE) or interfacing with a deep hierarchical memory (as in OMPN) incurs significant computational costs.

In this work we propose a novel architecture to learning sub-routines that addresses these shortcomings, which we call *Slot-based Transformer for Temporal Abstraction (SloTTAr)*. Central to our approach is the similarity between the *spatial grouping* of pixels into visual objects and the *temporal grouping* of state-action pairs into self-contained sub-routines. This motivates us to combine a parallel Transformer encoder with a Slot Attention module [Locatello et al., 2020] (developed for learning object representations c.f. Greff et al. [2020]), to group learned features at different temporal positions and recover a modular factorization into 'slots' (sub-routines) of the inputs. A parallel Transformer decoder reconstructs the action sequence from each slot and outputs an unnormalized distribution over endpoints. From these, the

segment belonging to each sub-routine and standard reconstruction objective can be derived.

In more realistic scenarios, coping with a variable number of sub-routines per trajectory adds an additional layer of complexity in order to achieve an ideal "fully unsupervised" sequence decomposition (i.e. without the knowledge about the number of sub-routines). In CompILE and OMPN, this issue is not addressed as the number of sub-routines in each trajectory is assumed to be known in advance, either at training and/or at test time, which limits their applicability. In contrast, in SloTTAr we cast the problem of estimating the number of sub-routines as a form of *adaptive computation* (where each available slot corresponds to an available computational step), requiring it to learn to use only some adaptive subset of all the slots available based on an estimated prior. This motivates us to use the adaptive computation loss introduced in PonderNet [Banino et al., 2021] as our learning objective. It allows for only an adaptive subset of the available slots in SloTTAr to be actively involved in the decomposition of any trajectory. While SloTTAr is still not "fully unsupervised", it only requires weak supervision for the problem of predicting the number of sub-routines in the form of their empirical distribution during training, and unlike CompILE and OMPN, it does not require any such information at test time.

We demonstrate the efficacy of our approach on Craft [Lu et al., 2021] and Minigrid [Chevalier-Boisvert et al., 2018] where we typically observe significant improvements over CompILE and OMPN in terms of recovering 'ground-truth' sub-routines (Section 3.3). In this way, we show how general principles of similarity-based grouping used to segment visual inputs into objects [Spelke, 1990; Köhler, 1929; Koffka, 1935; Greff et al., 2020] are also relevant for grouping other input modalities [Hommel, 1998]. Our qualitative and quantitative analysis reveals how SloTTAr leverages global access to the full trajectory to better capture individual sub-routines (Section 3.3) and leads to a substantial speed-up on existing benchmarks (Section 3.3). Finally, we demonstrate how our model is capable of modeling trajectories, each of which constituted of a variable number of sub-routines, without requiring access to ground-truth information about this quantity at test-time (Section 3.3).

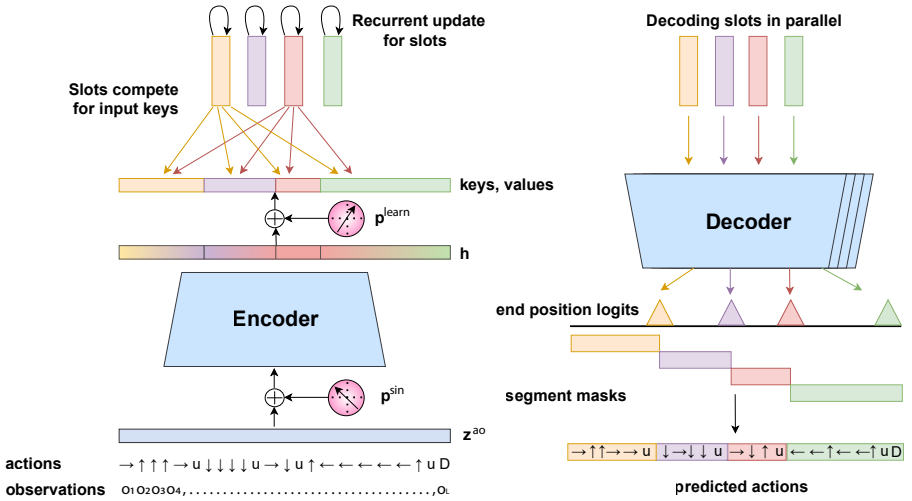


Figure 3.1: SloTTAr consists of 3 modules, used for: learning spatio-temporal features of action-observation sequences (*Encoder*), learning a similarity-based grouping of actions to their respective sub-routines through computing slot-based representations (*Slot Attention*; Locatello et al. [2020]), and for decoding slots to end positions and action segments of the sub-routine they capture (*Decoder*).

3.2 Method

Given an input sequence of actions $\boldsymbol{a} = [a_1, a_2, ..., a_L] : a_l \in \{1, .., A\}$ where A is the size of the action space and observations $\boldsymbol{o} = [\boldsymbol{o}_1, \boldsymbol{o}_2, ..., \boldsymbol{o}_L] : \boldsymbol{o}_l \in \mathbb{R}^{D_{\text{obs}}}$ of length L. Our goal is to infer the unique constituent latent *sub-routine* to which a pair of $(a_l, \boldsymbol{o}_l)$ belongs, and learn its associated representation (`slot_k` $\in \mathbb{R}^{D_{\text{slots}}}$). The input sequence of actions $\boldsymbol{a}$ is assumed to be a composition of at most K sub-routines. Similar to CompILE [Kipf et al., 2019], we consider an unsupervised reconstruction objective for training, yet here we strive for a fully parallel approach that employs more general-purpose modules for grouping inputs based on their internal predictive structure.

Our model, which we call *Slot-based Transformer for Temporal Abstraction (SloTTAr)*, consists of three modules, as shown in Figure 3.1. First, a Transformer encoder [Vaswani et al., 2017] learns suitable spatio-temporal features from the input sequences $\boldsymbol{a}$ and $\boldsymbol{o}$. This encoder benefits from *global* access to the whole input sequence to learn suitable context features at each location, unlike prior purely sequential RNN-based approaches [Kipf et al., 2019; Lu et al., 2021].

Subsequently, we adapt the Slot Attention module [Locatello et al., 2020] to iteratively group the features at each temporal location in parallel based on their internal predictive structure and obtain K slot representations `slot_k`. The slot representations are decoded in parallel by a decoder that outputs both predicted action logits $\hat{\boldsymbol{a}}^{(k)}$ and an unnormalized distribution over the end position of the sub-routine `end_logits_k`, modeled by the respective slot k.

Sub-routine segment masks are generated from the unnormalized distributions and combined with the predicted action logits (across slots) to obtain aggregated action sequence logits $\hat{\boldsymbol{a}}$. For training, we use the PonderNet loss [Banino et al., 2021] as our learning objective which comprises two terms. The first being a weighted sum of reconstruction errors of the action sequence, where each term corresponds to the error obtained when using k slots, and the weight is the probability that the trajectory contains $k \in \{1, \ldots, K\}$ sub-routines. The second is a regularization cost given by a KL-divergence term between the posterior and prior distributions of the number of sub-routines in a trajectory.

Encoder. We process action tokens $\boldsymbol{a}$ and observations $\boldsymbol{o}$ by `Embedding` and `Linear` layers respectively to acquire separately learned distributed representations for each. Next, we learn a joint representation of the action and observation representations at each position using a one-layer `MLP` with `ReLU` activation, after which a sinusoidal positional encoding $p_l^{\text{sin}} \in \mathbb{R}^{D_{\text{enc}}}$ is added [Vaswani et al., 2017]. Finally, we apply a number of standard Transformer encoder layers to learn suitable spatio-temporal features based on the content of the *entire* sequence, and add a learned positional encoding $p^{\text{learn}} \in \mathbb{R}^{D_{\text{enc}}}$ for follow-up processing:

$$\boldsymbol{z}_l^a,\ \boldsymbol{z}_l^o = \texttt{Embedding}(a_l),\ \texttt{Linear}(\boldsymbol{o}_l) \qquad \boldsymbol{z}_l^{ao} = \texttt{MLP}(\texttt{concat}(\boldsymbol{z}_l^a, \boldsymbol{z}_l^o)) \qquad \forall\ l \in \{1, \ldots, L\} \tag{3.1}$$

$$\boldsymbol{h} = \texttt{TransformerEnc}(\boldsymbol{z}^{ao} + \boldsymbol{p}^{\text{sin}}) + \boldsymbol{p}^{\text{learn}} \tag{3.2}$$

Slot Attention. We adapt Slot Attention [Locatello et al., 2020] to group these spatio-temporal features according to their constituent sub-routines and learn associated representations given by the slots. Slot Attention was previously only applied to the visual setting where pixels are grouped according to constituent visual objects to model images or videos. Here we hypothesize that sub-routines take on a similar role as modular and reusable primitives when modeling action sequences, suggesting that they similarly can be inferred using a grouping mechanism that focuses on their internal predictive structure (modularity) [Greff et al., 2020].

Algorithm 1 Modified Slot Attention update [Locatello et al., 2020].
Our modification uses separate shift and scale parameters per-slot (highlighted in blue) to initialize each slot representation `slot_k`.

Inputs: `inputs` $\in \mathbb{R}^{L \times D_{\text{in}}}$, `slots with slot_k` $= \boldsymbol{\mu}_k + \boldsymbol{\sigma}_k * \boldsymbol{z}$, $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, diag(\sigma)) \in \mathbb{R}^{D_{\text{slots}}}\ \forall k \in \{1, \ldots, K\}$

Params: shift and scale parameters : $\boldsymbol{\mu}_k$, $\boldsymbol{\sigma}_k$ per-slot; Linear projections for attention: $key, query, value$; `GRU`; `MLP`; `LayerNorm x2`

for $t = 0, 1, ..., T$ **do**
  `slots_prev = slots`
  `slots = LayerNorm(slots)`
  `attn = Softmax(`$\frac{1}{\sqrt{D_{\text{slots}}}}$ key`(inputs)`$\cdot$ $query$`(slots)`^{T}`, axis='slots')` ▷ norm. over K
  `updates = WeightedMean(weights=attn+`ϵ`, values=`$value$`(inputs))` ▷ norm `attn` values over L
  `slots = GRU(state=slots_prev, inputs=updates)` ▷ GRU update (per-slot)
  `slots += MLP(LayerNorm(slots))` ▷ residual update (per-slot)
end for
return `slots`

The iterative grouping mechanism in Slot Attention (reproduced in Algorithm 1) is implemented via key-value attention and a stateful slot update rule using a recurrent neural network (GRU; Cho et al. [2014], see also the earlier work by Gers et al. [2000]). The input features are projected to keys and values using linear layers $key(\cdot)$, $value(\cdot)$, queries are computed from slots using a linear layer $query(\cdot)$, and dot-product attention between keys and queries is used to distribute value vectors among the slots. Initial representations for all slots are sampled from a Gaussian $\mathcal{N}(\mathbf{0}, diag(\sigma))$ where σ is a hyperparameter. Here, unlike in the original Slot Attention formulation, we additionally use separate shift and scaling parameters ($\boldsymbol{\mu}_k$ and $\boldsymbol{\sigma}_k$) for each slot[2]. This allows a slot to differentiate itself from others and thus to focus on a specific part of the input sequence in the first iteration (e.g. by targeting the positional embedding). Importantly, slots *compete* to represent parts of the input sequence via a softmax function (`Softmax`) over slots, thereby encouraging a decomposition of the input into modular parts that can be processed separately and represented efficiently.

[2] A similar modification was explored as an ablation in [Locatello et al., 2020].

Algorithm 2 Mask Generation

`CumSum()` is a function that computes the cumulative sum of the input array it receives along the specified dimension (here the time-axis 'L'). Further, `mask_upto_k` is a variable that is a union over masks generated by all slots up to the kth while `mask_upto_km1` is an analogous quantity except that is up to the $k-1^{th}$ slot. For a visualization of the mask generation process, we refer the reader to Figure 3.2.

Inputs: `end_logits` $\in \mathbb{R}^{K \times L}$, `masks = []`, `end_cdf_k = 1`, `mask_upto_km1 = 0`

1: **for** $k = 1, \ldots, K$ **do**
2: `end_dist_k = Softmax(end_logits_k, axis='L')` ▷ norm. over L
3: `end_cdf_k = CumSum(end_dist_k, axis='L')` ▷ end position CDF
4: `mask_upto_k = 1 - end_cdf_k` ▷ union of all masks upto k
5: `mask_k = mask_upto_k * (1 - mask_upto_km1)` ▷ compute kth mask
6: `mask_upto_km1 = mask_upto_k` ▷ update
7: `masks = Append(masks, mask_k)` ▷ append kth mask
8: **end for**
9: **return** `masks`

Decoder. To decode the slot representations and obtain a correspondence of actions in $\mathbf{a}$ to their constituent sub-routine segments, we adapt the spatial broadcast decoder architecture [Watters et al., 2019] to the sequential setting using Transformers. Each slot representation is decoded in parallel to obtain the predicted action logits $\widehat{\mathbf{a}}^{(k)}$ as well as an unnormalized distribution over the end position of the sub-routine in the sequence (`end_logits_k`). The latter is used to generate the segment masks corresponding to each sub-routine as described in Algorithm 2. Algorithm 2 imposes that each slot takes responsibility for modeling a contiguous sub-sequence of the input. This is because mask generated by `slot_k+1` is active (value of 1) for all the time steps from the endpoint of that of `slot_k` to its own predicted end point. This is in contrast to the standard spatial mixture formulation adopted by prior work on discovering visual objects [Greff et al., 2017, 2019; Locatello et al., 2020]. This has a similar effect to the sequential decoding in CompILE [Kipf et al., 2019] and provides a useful inductive bias for treating sub-routines as contiguous chunks in time that can be ordered along the temporal axis, while keeping all other computations parallel. Figure 3.2 shows representative visualization of the outputs of computational steps 2-5 of Algorithm 2 respectively.

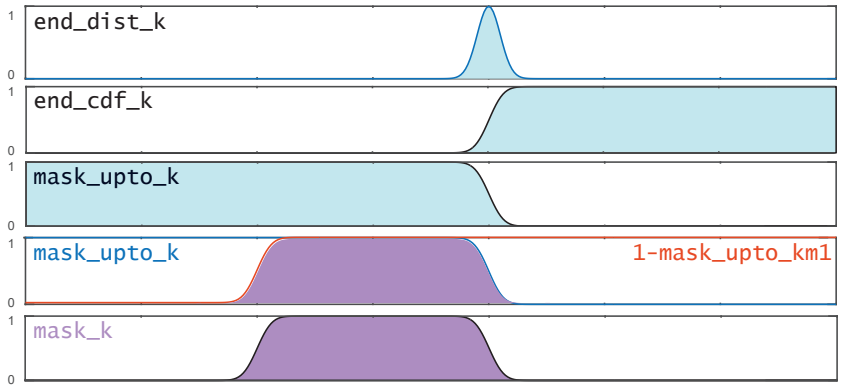


Figure 3.2: Panels 1-5 show a representative visualization of the outputs of computational steps 2-5 of Algorithm 2 respectively. In panel 4, curve in red shows the negation of the variable `mask_upto_km1` and curve in blue shows the variable `mask_upto_k`.

Objective Function. We adopt the PonderNet [Banino et al., 2021] adaptive computation loss formulation to model the variable number of sub-routines contained in different trajectories. We define a Bernoulli random variable Λ_k which represents a Markov process for halting with two states "continue" ($\Lambda_k = 0$) and "halt" ($\Lambda_k = 1$). The decision process always starts from the "continue" state and the transition probability is given by:

$$P(\Lambda_k = 1 | \Lambda_{k-1} = 0) = \lambda_k, \qquad \forall 1 \leq k \leq K. \tag{3.3}$$

where λ_k is the probability of halting at the kth `slot`, which is computed by applying a sigmoid activation to the last dimension of its respective slot representation `slot_k`. Then, the total probability that halting occurs in step $k \in \{1, \ldots, K\}$ (and the induced distribution p_{halt}) is given by:

$$p_k = \lambda_k \prod_{j=1}^{k-1} \big(1 - \lambda_j\big), \qquad p_{\text{halt}}(k) = \frac{p_k}{\sum_{k'=1}^{K} p_{k'}}, \tag{3.4}$$

where the normalization on the right ensures that p_{halt} is a valid probability distribution. At test time, we determine the number of 'active' slots by sampling for the kth `slot` from a Bernoulli distribution $\mathcal{B}(\lambda_k)$ and proceeding until we receive a halting signal. The entire system is trained end-to-end to minimize the following objective function:

$$\mathcal{L} = \sum_{k=1}^{K} p_{\text{halt}}(k) \mathcal{L}_{\text{CE}}\big(\widehat{\mathbf{a}}^{(\leq k)}; \mathbf{a}\big) + \beta D_{KL}\big[p_{\text{halt}} || p_e\big], \tag{3.5}$$

where $\mathcal{L}_{\text{CE}}$ refers to the standard cross-entropy loss between the reconstruction $\widehat{\mathbf{a}}^{(\leq k)}$ obtained by using up to k slots (i.e., $\widehat{\mathbf{a}}^{(\leq k)} = \sum_{k'=1}^{k} \texttt{mask_k}' * \widehat{\mathbf{a}}^{(k')}$) and the ground-truth action sequence $\mathbf{a}$, and β is a hyperparameter that weights the influence of the second term. The second term is a KL-divergence between the probability distribution induced by p_{halt} (Equation (3.4)) and a prior distribution p_e governing the expected number of sub-routines. The empirical prior distribution p_e is derived from the histogram of the number of sub-routines in each trajectory across the training dataset. Note that this requires only a weaker form of ground-truth information about the *distribution* of the number of sub-routines in the training data (compared to CompILE), while no such information is needed at test-time (compared to OMPN). Please refer to Appendix B.3 for additional details about the model architecture and the estimated empirical prior distribution.

We compare SloTTAr to CompILE [Kipf et al., 2019] and OMPN [Lu et al., 2021] on the partially and fully observable versions of the environments in Craft [Lu et al., 2021] and on partially observable environments in Minigrid [Chevalier-Boisvert et al., 2018]. Here our focus is on the unsupervised setting without the use of task sketches as an auxiliary supervision signal [Andreas et al., 2017; Shiarlis et al., 2018].

3.3 Results

Datasets We use 3 tasks in the Craft environment namely, `MakeAxe`, `MakeBed`, `MakeShears` consistent with prior work [Lu et al., 2021]. We also compare models on 4 tasks in Minigrid suite of environments namely `DoorKey-8x8`, `UnlockPickup-v0`, `BlockedUnlockPickup` and `KeyCorridor-S4R3`. Episodes in this Minigrid are procedurally generated, leading to far greater variability for the actions and observations that constitute a sub-routine compared to in Craft. Further, in Minigrid environments, two actions (`PICKUP` or `TOGGLE`) typically indicate the presence of sub-routine boundaries as opposed to in Craft where this is marked by only a single `USE` action. In Figure 3.3, we can see some representative trajectories from these environments with the coloured arrows indicating the various ground-truth sub-routines.

Evaluation Metrics To quantitatively measure the quality of the action sequence decomposition, we use the F1 score (with a tolerance of 1 consistent with Lu et al. [2021]) based on the accuracy of the boundary index predictions

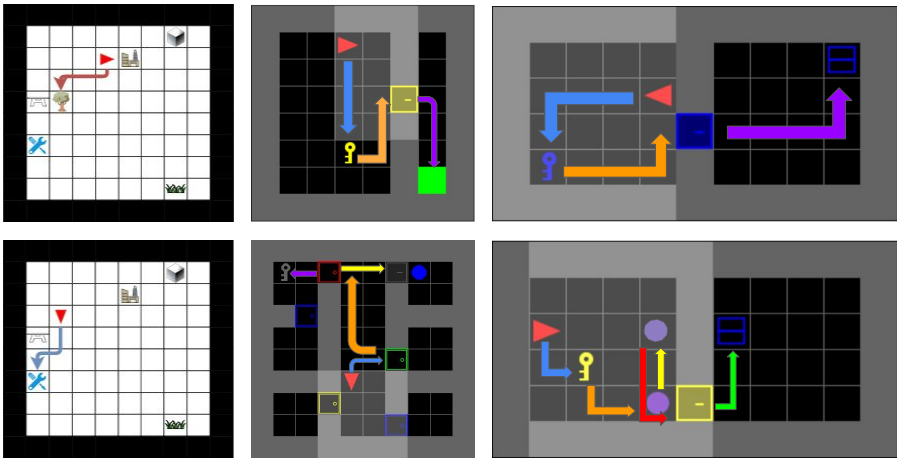


Figure 3.3: Environments in Craft [Lu et al., 2021] and Minigrid [Chevalier-Boisvert et al., 2018] used for sequence decomposition. Agent (red arrow) navigates in the grid world and takes actions such as move up, down, left, right, pick up keys, collect wood, open doors etc. to solve various tasks. Ground-truth sub-routine action sequences are shown as colored arrows.

with respect to ground-truth. Further, we also report the alignment accuracy between the predicted and ground-truth sub-routines consistent with prior work [Lu et al., 2021]. Please refer to Appendix B.1 for further details on how ground truth sub-routine boundaries are computed.

Hyperparameter Search We report results after conducting an extensive hyperparameter search (up to 300 configurations) for each method. We include training parameters like batch size and learning rate, capacity parameters like layer sizes, and model-specific parameters such as the number of Slot Attention iterations in case of SloTTAr.

General trends we encountered as part of this search for SloTTAr are that (1) using only a single Transformer layer (for the Transformer Encoder and Decoder modules) was necessary to limit the tendency of the self-attention layer to aggregate information across non-contiguous temporal blocks; (2) that the capacity of the slots must be sufficiently bottlenecked since otherwise the model tends to only use a single slot to model the entire sequence; and (3) that often a single iteration of Slot Attention was sufficient to learn the decomposition

	Craft (fully)		Craft (partial)	
	F1 score	Align. acc.	F1 score	Align. acc.
CompILE	83.10 (3.07)	85.77 (0.51)	70.74 (14.14)	67.71 (16.81)
OMPN	98.49 (0.16)	97.32 (0.33)	**95.84 (0.71)**	**93.68 (0.96)**
OMPN-2	97.04 (1.03)	94.58 (2.44)	91.20 (2.13)	85.04 (3.82)
SloTTAr	**99.84 (0.02)**	**99.41 (0.25)**	80.51 (4.47)	83.14 (3.87)

Table 3.1: F1 scores and alignment accuracies on Craft [Lu et al., 2021] (fully and partially observable).

for all the environments. In fact, performance usually degraded slightly with additional iterations, as is consistent with prior work [Locatello et al., 2020; Kipf et al., 2022].

Below we report results only for the best configurations of SloTTAr and the baseline models (CompILE, OMPN) with mean and standard deviation computed over 5 seeds. Details about the search parameters for each method and the best configurations can be found in Appendix B.3 [3].

Craft Table 3.1 shows the results of comparing SloTTAr to the baseline models on environments in Craft. All trajectories in these environments have 4 sub-routines executed. On the fully observable Craft tasks, it can be seen how SloTTAr outperforms both CompILE and OMPN. Similarly, on the partially observable settings in Craft, it can be seen how SloTTAr outperforms CompILE, although this time it performs worse compared to OMPN. We note that the best configuration for OMPN in Table 3.1 was obtained when using 3 levels of hierarchy depth (Table B.12), and speculate how the strong hierarchical inductive bias in OMPN gives it an edge in this setting as it closely reflects the hierarchical sub-routine structure in Craft. Indeed, when we only consider OMPN configurations having two levels of memory (OMPN-2) it can be observed how the difference in alignment accuracy to the best performing configuration greatly reduces.

Minigrid – DoorKey-8x8 and UnlockPickup All trajectories from these environments involve executing 3 sub-routines. Table 3.2 shows that on

[3] Code repository for the paper: `https://github.com/agopal42/slottar`.

	DoorKey-8x8 (partial)		UnlockPickup-v0 (partial)	
	F1 score	Align. acc.	F1 score	Align. acc.
CompILE	43.78 (2.92)	64.93 (2.96)	43.81 (1.70)	69.85 (2.01)
OMPN	45.03 (14.04)	57.20 (4.35)	55.27 (5.76)	62.08 (7.78)
OMPN-2	48.08 (10.48)	60.29 (8.47)	48.47 (14.39)	46.25 (6.84)
SloTTAr ($\beta=0$)	72.19 (9.87)	63.93 (16.14)	75.08 (8.78)	76.64 (7.14)
SloTTAr	**91.73 (3.75)**	**93.96 (0.96)**	**79.08 (8.01)**	**81.63 (7.41)**

Table 3.2: F1 scores and alignment accuracies on partially observable versions of Minigrid environments – `DoorKey-8x8` and `UnlockPickup-v0`.

these harder `DoorKey-8x8` and `UnlockPickup-v0` partially observable Minigrid environments, SloTTAr significantly outperforms both CompILE and OMPN in terms of both F1 and alignment accuracy. We also show results from an ablation where the KL term is removed from the loss by setting $\beta = 0$ (in Equation (3.5)) to emphasize the role of the empirical prior in ascertaining the correct number of "active" slots to be used.

We investigated whether the reduced performance for OMPN on the Minigrid environments is due to these datasets using multiple delimiting tokens. Table B.14 in Appendix B.6 reports the performance for synthetic versions of the `DoorKey-8x8` and `UnlockPickup` datasets with a single delimiting token (`TOGGLE`) instead of the default two i.e., `TOGGLE` and `PICKUP`. We see that the quality of decomposition by OMPN improves modestly but a significant gap to SloTTAr still remains.

Minigrid – BlockedUnlockPickup and KeyCorridor All trajectories in `BlockedUnlockPickup` involve executing either 4 or 5 sub-routines and in `KeyCorridor-S4R3` between 4 to 9 sub-routines. Table 3.3 shows the performance of SloTTAr against baselines on `BlockedUnlockPickup` and `KeyCorridor-S4R3` environments which have variable number of sub-routines. The baseline models (i.e., CompILE and OMPN) receive supervision on the number of sub-routines in every trajectory during training and/or evaluation. In contrast, SloTTAr performs the decomposition in a mostly unsupervised manner and only uses the empirical *distribution* of the number of sub-routines across the training dataset as a prior during training.

We can see that despite this advantage given to the baseline models, SloTTAr still significantly outperforms CompILE on both environments and remains competitive with OMPN. The decomposition quality of all these models on the `KeyCorridor-S4R3` environment is rather poor. This result partially serves to highlight the limits of current state-of-the-art approaches to decompose action trajectories and recover semantically meaningful sub-routine parts in a mostly unsupervised manner.

	BlockedUnlockPickup (partial)		KeyCorridor-S4R3 (partial)	
	F1 score	Align. acc.	F1 score	Align. acc.
CompILE (oracle)	19.99 (1.73)	18.15 (1.11)	0.00 (0.00)	40.37 (0.01)
OMPN (oracle)	**55.80 (11.31)**	45.83 (8.73)	**48.86 (18.34)**	**41.58 (2.64)**
SloTTAr	41.44 (3.54)	**63.48 (0.13)**	**48.81 (5.81)**	37.98 (6.04)

Table 3.3: F1 scores and alignment accuracies on Minigrid environments with variable number of sub-routines per-trajectory – `BlockedUnlockPickup` and `KeyCorridor-S4R3`. CompILE and OMPN receive supervision on the number of sub-routines in every trajectory during training and/or evaluation (denoted by "oracle").

Analysis

To better understand the performance of SloTTAr and verify that global access to input sequence is beneficial, we quantitatively measure the extent to which slots access information from past and future sequence tokens when modeling a particular sub-routine. We introduce two metrics we call "Backward Access" (BA) and "Forward Access" (FA), which are computed for a single trajectory as follows:

$$\begin{aligned}
\text{BA} &= \frac{1}{K}\sum_{k=1}^{K}\frac{\sum_{l=1}^{\alpha_{min}-1}\mathbb{1}(\texttt{slot_attn_k}_l > \texttt{t}_{on})}{\alpha_{min}-1} \quad \text{where} \quad \alpha_{min} = \min\{1 \le l \le L : \texttt{mask_k}_l > \texttt{t}_{on}\} \\
\text{FA} &= \frac{1}{K}\sum_{k=1}^{K}\frac{\sum_{l=\alpha_{max}+1}^{L}\mathbb{1}(\texttt{slot_attn_k}_l > \texttt{t}_{on})}{L-\alpha_{max}+1} \quad \text{where} \quad \alpha_{max} = \max\{1 \le l \le L : \texttt{mask_k}_l > \texttt{t}_{on}\}
\end{aligned} \tag{3.6}$$

where $\mathbb{1}()$ is an indicator function and $\texttt{t}_{on} \in (0, 1)$ denotes a threshold for activation (here $\texttt{t}_{on} = 0.8$). For the degenerate case where $\alpha_{min} = 1$ or it is an empty set, the BA is set to zero. Likewise, when $\alpha_{max} = L$ or is an empty set the FA is set to zero. Intuitively, these metrics measure the fraction of total timesteps

	Forward Access	Backward Access
Craft (fully)	16.94 (6.63)	9.93 (6.76)
Craft (partial)	21.17 (1.75)	9.44 (2.97)
DoorKey-8x8	26.85 (8.69)	8.10 (8.77)
UnlockPickup-v0	15.94 (3.76)	6.06 (3.71)

Table 3.4: Forward Access Backward Access metrics (mean and standard deviation over 5 seeds) for SloTTAr on 4 datasets – Craft (fully and partially observable), DoorKey-8x8 and UnlockPickup-v0.

accessed by each slot `slot_k` that lie in the past or future as indicated by the corresponding slot attention weights over the input sequence (`slot_attn_k`) exceeding the threshold $\texttt{t}_{on}$. To determine the boundary start (α_{min}) and end (α_{max}) indices of the segment modeled by `slot_k`, we binarize its associated decoder mask `mask_k` and threshold it in the same way.

Table 3.4 shows these metrics for SloTTAr on the test split of 4 environments namely — fully and partially observable variants of Craft, `DoorKey-8x8` and `UnlockPickup-v0`. It can be seen how the slots in our fully parallel model learn to effectively utilize information from past and future sequence tokens to solve the sequence decomposition task.

Qualitatively, in Figure 3.4 we show a representative visualization of the slot attention weights and alpha masks (`mask_k`) on a sample trajectory from the Craft fully observable environment (delimiters are the `USE` action 'u'). It can be seen that the 'active' slots (first four rows) have learned to gather information to the left (past) and to the right (future) of the sub-routine segment it models to acquire the necessary contextual information needed to perform this decomposition. Please refer to Appendix B.5 for additional visualizations of sample trajectories from other environments.

Computational Efficiency

A further advantage of using a fully parallel architecture as in SloTTAr is the potential gain in speed. While, the use of Transformers yields a computational complexity that is quadratic in sequence length, all these computations can be performed in parallel. In Table 3.5 we report the number of tokens processed per-second for each model during training (forward and backward pass) and at testing time (forward pass). Additionally, we report the total wall clock time

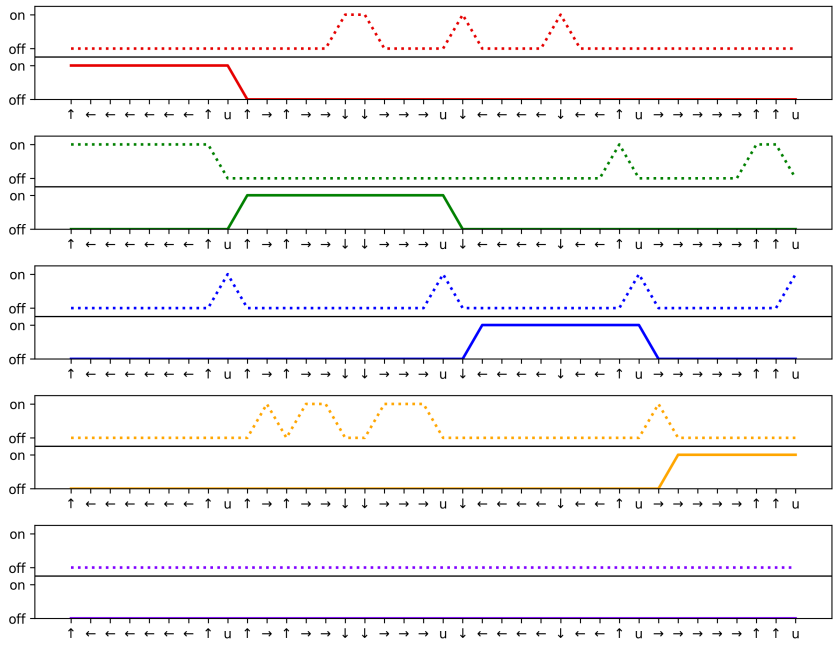


Figure 3.4: In each subplot, bottom panel shows thresholded alpha mask (solid lines) and top panel shows the thresholded slot attention weights (dotted lines) on the y-axis for a sample trajectory from fully observable Craft environment with actions on the x-axis. Different subplots correspond to different slots (colour coded). We see how each slot learns to gather contextual information (from both the left and right neighborhood) in order to model its respective sub-sequence.

for training each model. It can be seen how SloTTAr is about $3\mathrm{x}$ faster to train compared to CompILE and upto $7\mathrm{x}$ faster to train compared to the OMPN model.

Limitations

The focus of this work is on the fundamental problem of learning "meaningful" sub-routines with an emphasis on modularity and compositionality as the desirable properties used to learn these temporal abstractions. The applications of discovered sub-routines to actual online decision making (e.g., in hierarchical reinforcement learning) are left for future work. In fact, our current model design does not allow an out-of-the-box application for an RL setting. This is because currently our Decoder module requires global context (past and future timesteps) to output action logits. However, we can consider potential remedies and strategies to address this limitation. The most straight-forward approach is to train a separate auto-regressive model (e.g. an auto-regressive Transformer) on

	Train	Test	Wall clock
CompILE	14	36	0h 47m
OMPN	3	7	1h 46m
OMPN-2	4	10	1h 18m
SloTTAr	**46**	**114**	**0h 14m**

Table 3.5: First and second columns show the number of tokens (in thousand) processed per-second during training and testing on a single Nvidia GTX 1080Ti GPU card respectively. While, the third column shows the wall clock time for the model training to converge. These numbers are computed on the fully observable Craft task using a batch size of 64 and sequence length of 65 tokens. SloTTAr uses both hidden size and slot size of 128 units and 8 attention heads. The number of segments is 4. OMPN and OMPN-2 use 3 and 2 levels of memory hierarchy respectively. The hidden size of OMPN models is 128 and batch size of 128.

the discovered sub-routine state/action sequences by using the corresponding slot representation as a sequence-start symbol embedding (and by appending an end-of-sequence token at the end of the sub-routine).

3.4 Related Work

Imitation Learning of Temporal Abstractions. The Craft environment was originally introduced to evaluate the method of Andreas et al. [2017], however they rely on annotations of sub-routine sequences ('policy sketches') as additional supervision to the model. In a similar way, TACO [Shiarlis et al., 2018] treats this setting as a sequence alignment and classification problem using an LSTM [Hochreiter and Schmidhuber, 1997b] trained with a CTC loss [Graves et al., 2006]. In a recent work of Ajay et al. [2021], the primitives are learned directly from offline data for continuous control. However, these methods learn a continuous (low-dimensional) space of primitives whereas our method represents primitives as a discrete set. Our focus on learning useful sub-routines can also be viewed as an instance of learning temporal abstractions more broadly, such as event segments in video. Relevant approaches propose to use recurrent latent variable models for this task [Gregor et al., 2019; Kim et al., 2019], while making stronger assumptions regarding prior knowledge about boundary locations and the existence of hierarchical structure between

latent states and across time.

Option Discovery. In the context of the options framework [Sutton et al., 1999], several methods have been proposed for option and/or sub-goal discovery [Bacon et al., 2017; Machado et al., 2017; McGovern and Barto, 2001; Stolle and Precup, 2002; Şimşek and Barto, 2004; Şimşek et al., 2005; Simsek and Barreto, 2008]. However, these algorithms are not easily extendable to the case with nonlinear function approximation as needed in our case. Alternatively, other methods using nonlinear function approximation propose to maximize coverage (diversity) of learned skills by maximizing the mutual information between options and terminal states achieved by their execution [Gregor et al., 2016; Eysenbach et al., 2019]. It remains difficult to precisely evaluate the level of semantically independent modes of behavior that options discovered in this way afford. In contrast, here we explicitly optimize and evaluate the sub-routines with regards to their modularity.

Transformer-based Sequence Modeling Beyond Language. Many recent works have applied Transformers beyond the domain of natural language, such as to images [Dosovitskiy et al., 2021; Zhu et al., 2021; Singh et al., 2022a], and other high-dimensional data [Jaegle et al., 2021b,a]. Transformers were successfully applied to reason about visual primitives, such as object representations [Ding et al., 2021a], and have recently gained a lot of interest in reinforcement learning settings [Parisotto et al., 2020; Rae et al., 2020; Fan et al., 2020; Irie et al., 2021; Chen et al., 2021a; Zeng et al., 2022]. Especially in the offline setting [Janner et al., 2021; Chen et al., 2021b], which closely relates to the setting explored here, Transformers have shown promise in predicting actions from certain input commands in a supervised manner [Schmidhuber, 2019; Srivastava et al., 2019].

Adaptive Computation. The idea of allowing dynamic program halting behavior in recurrent neural networks was initially explored in Schmidhuber [2012]. Here, a special 'halting' unit, when activated, resulted in termination of the program and emission of program outputs. Similarly, the Adaptive Computation Time (ACT) algorithm [Graves, 2016] uses a sigmoidal 'halting unit' (and associated weights) for halting program execution, which is regularized to penalize the amount of computation used. The ACT algorithm for halting has later been adapted to the Transformer family of models as well [Dehghani et al., 2019]. PonderNet [Banino et al., 2021] builds on ACT but crucially differs in

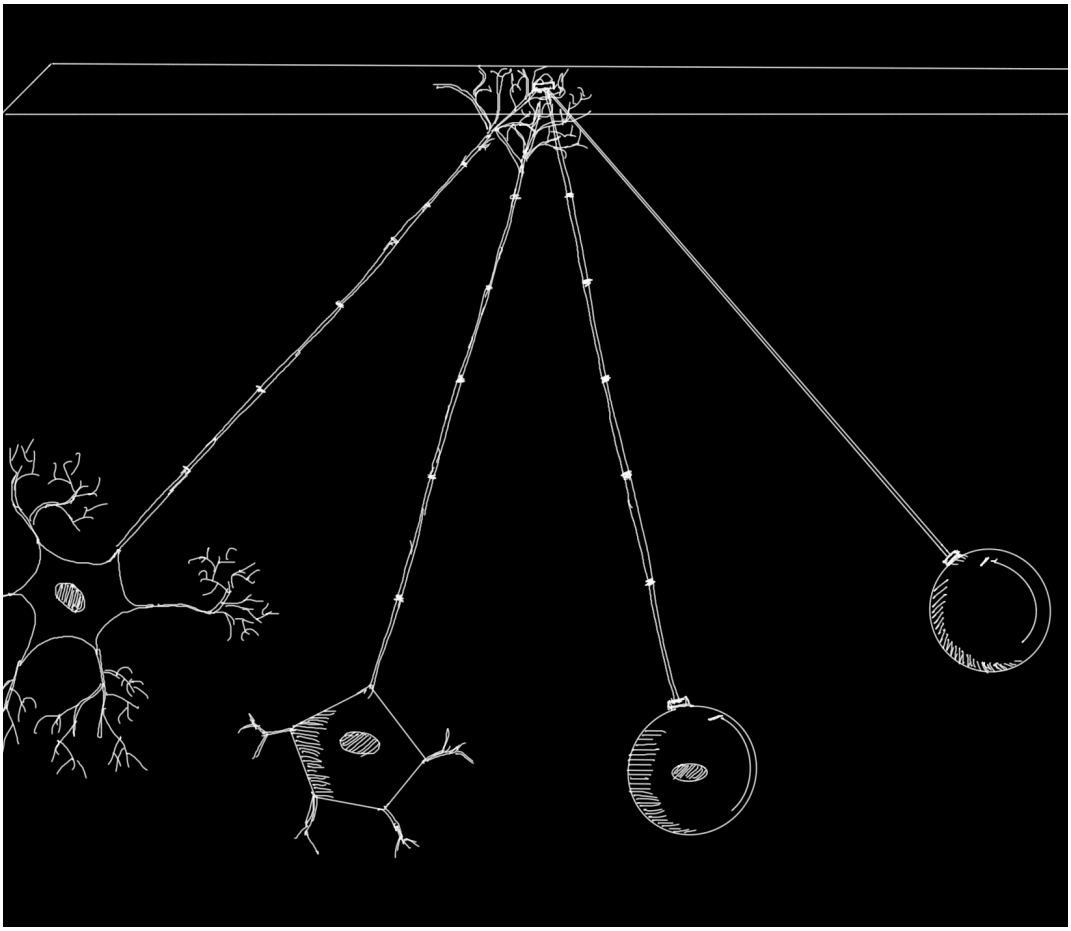
its use of a probabilistic halting strategy. It defines a probability distribution over all possible computation steps available and weights the output prediction loss at each step with their associated halting probability. Further, PonderNet uses a KL-divergence between the posterior distribution over the number of computation steps and a prior distribution as a regularizer.

Event Cognition. Influential work from cognitive psychology on object perception in humans [Kahneman et al., 1992] posit that the brain integrates distributed representations of perceptual (visual) stimuli into *object files*. Further studies have suggested that such a feature integration mechanism extends beyond just the visual stimuli and include associated behavioral responses (actions) as well [Hommel, 1998, 2004, 2007]. These works suggest that the definition of an *object files* could be extended to include a notion of *event files* which store aggregated information of action-perception stimuli. More specifically, our ability to chunk up a continuous stream of activity into a set of semantically meaningful *event entities* depends on both bottom-up sensory cues of color, sound, movement etc. as well as top-down cues like goals and beliefs [Radvansky and Zacks, 2014]. Further, these events tend to have hierarchical organization [Zacks et al., 2001; Hard et al., 2006] and play an important role in decision-making and planning [Radvansky and Zacks, 2014]. The Slot Attention module in SloTTAr analogously integrates information by attending to bottom-up features of both perceptual (observations) and behavioral (actions) stimuli to learn the appropriate slot representations for sequence decomposition.

3.5 Conclusion

We have proposed SloTTAr, a novel approach to learning action segments belonging to modular sub-routines (suitable as 'options' [Sutton et al., 1999]) along with the number of such sub-routines in a mostly unsupervised fashion. Our approach draws insight from prior literature on learning about visual objects [Locatello et al., 2020], Transformers [Vaswani et al., 2017] and adaptive computation [Banino et al., 2021] to improve over existing purely sequential approaches. We found that SloTTAr outperforms CompILE and OMPN in terms of recovering ground-truth sub-routine segments across a wide range environments in Craft [Lu et al., 2021] and Minigrid [Chevalier-Boisvert et al., 2018] that have a fixed number of sub-routines. On more sophisticated environments with trajectories containing varying numbers of sub-routines, it could also be observed how SloTTAr outperforms CompILE and remains

competitive with OMPN, despite these baseline models requiring ground-truth information about the number of sub-routines at the level of individual trajectories. At the same time, these results also indicated how current state-of-the-art approaches are still limited in their ability to decompose action sequences into semantically meaningful modular sub-routines. Our analysis revealed how SloTTAr leverages parallel access to the full sequence to perform sequence decomposition, which further leads to a substantial speed-up through leveraging parallel computation. In this way, we demonstrated how general principles of similarity-based grouping used to segment visual inputs are also relevant for grouping other input modalities, suggesting many additional avenues for future research.



Chapter 4

Contrastive Training of Complex-valued Autoencoders[1]

4.1 Overview

When shown arrays of simple visual elements, people have a natural tendency to perceive the arrays in disjoint groups based on their color, shape, spatial configuration, etc. Grouping behavior was first studied and systematized by Gestalt psychologists [Wertheimer, 1923; Koffka, 1935; Köhler, 1967] who proposed a set of principles according to which perception operates, e.g., grouping by *proximity*, *similarity*, *closure*, *good continuation* and *common fate*. Although these Gestalt principles—also called laws—were assumed to be innate, statistics of the environment may explain some forms of grouping [Kim et al., 2021], and people rapidly learn new grouping principles when they are exposed to novel environments [Zemel et al., 2002]. Grouping abilities are acquired early in an infant's developmental cycle [Spelke, 1990] and serve as a building block in the human ability to form concepts [Spelke and Kinzler, 2007], build abstractions and categorize [Kemp and Tenenbaum, 2008], and reason about the physical world [Battaglia et al., 2013].

Perceptual grouping has been modeled in deep learning under the guise of

[1] For the full paper please see "Contrastive Training of Complex-valued Autoencoders for Object Discovery" [Stanić et al., 2023]. Aleksandar Stanić and Anand Gopalakrishnan are equal first-authors of this paper. Contents of this chapter are partially shared with the PhD thesis of Aleksandar Stanić.

object-centric learning. (See Greff et al. [2020] for an overview.) Object-centric models discover modular and compositional representations that can facilitate stronger generalization and relational reasoning capabilities on downstream visual tasks such as question answering [Ding et al., 2021b], game playing [Bapst et al., 2019; Stanić et al., 2023; Yoon et al., 2023] and robotics [Mandikal and Grauman, 2021; Wu et al., 2021; Zadaianchuk et al., 2021]. Both human and AI object-centric learning aim to solve the *binding problem* [Treisman, 1996; Roskies, 1999]—determining how to integrate visual information in a flexible, dynamic manner to form unified wholes.

The current mainstream approach to implement such object binding mechanisms in artificial neural networks is to maintain a set of separate activation vectors (so-called *slots*) [Locatello et al., 2020]. Various segregation mechanisms [Greff et al., 2020] are then used to route requisite information from the inputs to infer each slot in a iterative fashion. Such 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 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-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., 2019] (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 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, the 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 like `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 4.2) 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 4.2). 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.

4.2 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 $\mathbf{x} \in \mathbb{C}^{h \times w \times 3}$:

$$\mathbf{x} = \mathbf{x}' \odot e^{i\boldsymbol{\phi}}, \quad \text{where } \odot \text{ denotes a Hadamard product} \tag{4.1}$$

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 $f_{\mathbf{w}} : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ with real-valued trainable parameters $\mathbf{w} \in \mathbb{R}^p$. $f_{\mathbf{w}}$ is typically a convolutional or linear layer. First, $f_{\mathbf{w}}$ is applied separately to the real and imaginary components of the input:

$$\boldsymbol{\psi} = f_{\mathbf{w}}\left(\text{Re}(\mathbf{x})\right) + f_{\mathbf{w}}\left(\text{Im}(\mathbf{x})\right) i \quad \in \mathbb{C}^{d_{\text{out}}} \tag{4.2}$$

Note that both $\text{Re}(\mathbf{x}), \text{Im}(\mathbf{x}) \in \mathbb{R}^{d_{\text{in}}}$. 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 $\boldsymbol{\psi} \in \mathbb{C}^{d_{\text{out}}}$:

$$\mathbf{m}_{\boldsymbol{\psi}} = |\boldsymbol{\psi}| + \mathbf{b}_{\mathbf{m}} \quad \in \mathbb{R}^{d_{\text{out}}} \quad ; \quad \boldsymbol{\phi}_{\boldsymbol{\psi}} = \arg(\boldsymbol{\psi}) + \mathbf{b}_{\boldsymbol{\phi}} \quad \in \mathbb{R}^{d_{\text{out}}} \tag{4.3}$$

Third, the CAE uses an additional gating function proposed by Reichert and Serre [2014] to further transform this "intermediate" magnitude $\mathbf{m}_\psi \in \mathbb{R}^{d_{\text{out}}}$. 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 $f_{\mathbf{w}}$ 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 $\mathbf{m}_\psi$ (called *synchrony term* [Reichert and Serre, 2014]) from Eq. 4.3 is computed to yield "gated magnitudes" $\mathbf{m}_\mathbf{z} \in \mathbb{R}^{d_{\text{out}}}$ as follows:

$$\boldsymbol{\chi} = f_{\mathbf{w}}\left(|\mathbf{x}|\right) + \mathbf{b}_\mathbf{m} \quad \in \mathbb{R}^{d_{\text{out}}} \quad ; \quad \mathbf{m}_\mathbf{z} = \frac{1}{2}\mathbf{m}_\psi + \frac{1}{2}\boldsymbol{\chi} \quad \in \mathbb{R}^{d_{\text{out}}} \tag{4.4}$$

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. 4.4) while leaving the phase values $\boldsymbol{\phi}_\psi$ (Eq. 4.3) untouched:

$$\mathbf{z} = \text{ReLU}(\text{BatchNorm}(\mathbf{m}_\mathbf{z})) \odot e^{i\boldsymbol{\phi}_\psi} \quad \in \mathbb{C}^{d_{\text{out}}} \tag{4.5}$$

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].

4.3 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

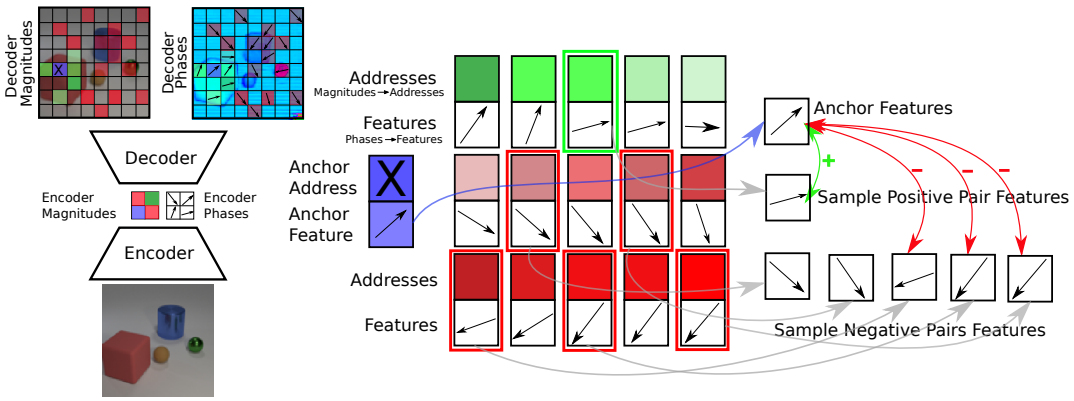


Figure 4.1: Sampling process of positive (green) and negative (red) pairs for one anchor (purple) in the CtCAE model. The sampling process here is visualized only for the decoder output. Note that we contrast the encoder output in an identical manner. Anchor address (the purple box marked with an X) corresponds to the patch of magnitude values and the feature corresponds to the phase values. See **Contrastive Training of CAEs** in Section 4.3 for more details.

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 4.1, 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 *separability*[2] (we also illustrate this in Section 4.4). 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 4.1). For setting

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

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 (4.7)), 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 $h' \times w' \times d_{\text{f}}$. 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 $\mathbf{A} \in \mathbb{R}^{N_A \times d_{\text{f}}}$; each anchor is thus denoted as $\mathbf{A}_k \in \mathbb{R}^{d_{\text{f}}}$ for all $k \in \{1, ..., N_A\}$. Now by using Algorithm 3, we extract 1 positive and $M - 1$ negative examples for each anchor. We denote these examples by a matrix $\mathbf{P}^k \in \mathbb{R}^{M \times d_{\text{f}}}$ for each anchor $k \in \{1, ..., N_A\}$ arranged such that $\mathbf{P}^k_1 \in \mathbb{R}^{d_{\text{f}}}$ is the positive example and all other rows $\mathbf{P}^k_j \in \mathbb{R}^{d_{\text{f}}}$ 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:

$$\mathcal{L}_{\text{ct}} = \frac{1}{N_A} \sum_{k=1}^{N_A} \log \left(\frac{\exp\left(d\left(\mathbf{A}_k; \mathbf{P}^k_1\right)/\tau\right)}{\sum_{j=1}^{M} \exp\left(d\left(\mathbf{A}_k; \mathbf{P}^k_j\right)/\tau\right)} \right) \tag{4.6}$$

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

Algorithm 3 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 $k_{\text{top}}, m_{\text{bottom}}, M$.

```
addresses ← Flatten(addresses, dims=(0,1))
features ← Flatten(features, dims=(0,1))
distances ← CosineDistance(addresses, anchor_address)
top_k_features ← features[argsort(distances, dim=0)[:k_top]]
bottom_m_features ← features[argsort(distances,
dim=0)[-m_bottom:]]
pos_pair_idx ~ Uniform [0, k_top]                ▷ sample 1 positive pair
neg_pair_idxs ~ Uniform [0, m_bottom]×(M − 1)    ▷ M − 1 samples
without replacement
positive_pair ← top_k_features[pos_pair_idx]
negative_pairs ← bottom_m_features[neg_pair_idxs]
return positive_pair, negative_pairs
```

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 4.5). 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:

$$\mathcal{L} = \mathcal{L}_{\text{mse}} + \beta \cdot \mathcal{L}_{\text{ct}} \quad ; \quad \mathcal{L}_{\text{mse}} = ||\mathbf{x}' - \widehat{\mathbf{x}}||_2^2 \quad ; \quad \widehat{\mathbf{x}} = |\mathbf{y}| \tag{4.7}$$

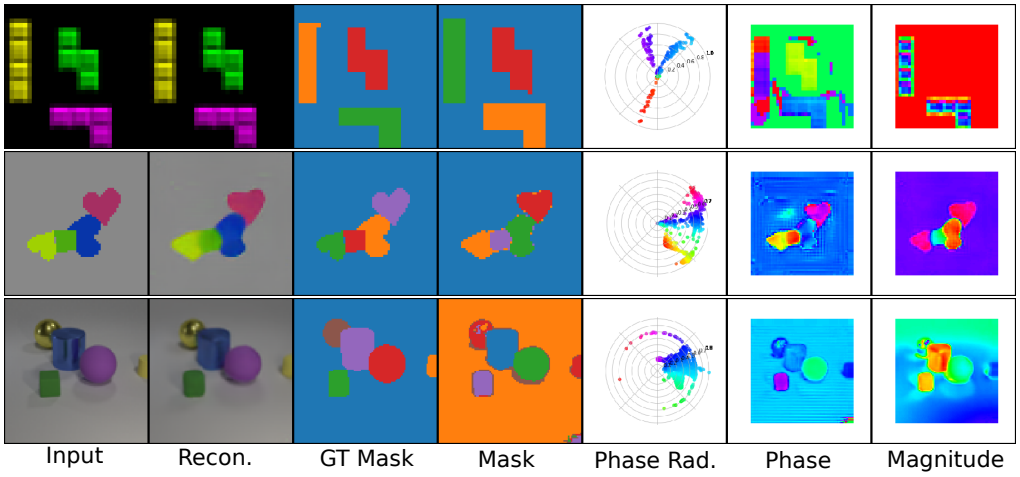


Figure 4.2: Unsupervised object discovery on `Tetrominoes`, `dSprites` and `CLEVR` with CtCAE. "Phase Rad." (col. 5) is the radial plot with the phase values from -π to π radians. "Phase" (col. 6), are phase values (in radians) averaged over the 3 output channels as a heatmap (colors correspond to those from "Phase Rad.") and "Magnitude" (col. 7) is the magnitude component of the outputs.

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.

4.4 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 4.2) 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 of all models are ordered in the same way on 32x32 resolution as the original resolution (see Table 4.1), 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 4.3. For more details about the encoder and decoder architecture of all models see Appendix C.2. 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 Ba, 2015b] with a constant learning rate of 4e-4, i.e., no warmup schedules or annealing (all hyperparameter details are given in Appendix C.3)[3].

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, 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.

Unsupervised Object Discovery. Table 4.1 shows the performance of our CAE++, CtCAE, and the baseline CAE Löwe et al. [2022] on `Tetrominoes`,

[3]Code repository for the paper: `https://github.com/agopal42/ctcae`.

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 4.1: 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].

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 4.1 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 existing state-of-the-art synchrony-based model, CAE.

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 4.2: 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.

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 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

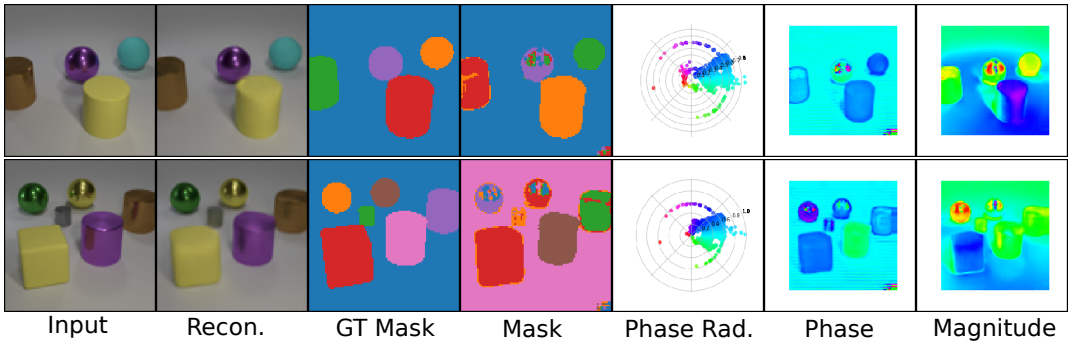


Figure 4.3: CtCAE on `CLEVR` is able to infer more than four objects, although it sometimes makes mistakes, such as specular effects or grouping together based on color (two yellow objects).

	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 ± 0.07	0.58 ± 0.15	0.39 ± 0.06	0.53 ± 0.15	0.43 ± 0.04	0.45 ± 0.13	0.45 ± 0.05	0.44 ± 0.11
CtCAE	**0.45** ± **0.11**	**0.76** ± **0.07**	**0.48** ± **0.06**	**0.72** ± **0.11**	**0.48** ± **0.04**	**0.65** ± **0.13**	**0.48** ± **0.04**	**0.61** ± **0.13**
CLEVR	3 objects		4 objects		5 objects		6 objects	
CAE++	0.32 ± 0.06	0.35 ± 0.29	0.33 ± 0.05	0.32 ± 0.26	0.31 ± 0.04	0.26 ± 0.20	0.32 ± 0.04	0.27 ± 0.18
CtCAE	**0.53** ± **0.07**	**0.71** ± **0.20**	**0.52** ± **0.06**	**0.70** ± **0.21**	**0.47** ± **0.07**	**0.67** ± **0.21**	**0.45** ± **0.06**	**0.68** ± **0.20**

Table 4.3: 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.

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 4.2 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 number 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 4.3 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.6), 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.

Ablation on Architectural Changes. Table 4.4 shows an ablation study on the proposed architectural modifications on `Tetrominoes` (for similar findings on other datasets, see Appendix C.6). 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.

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 4.4: Architectural ablations on `Tetrominoes`.

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 4.5 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

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 4.5: Contrastive loss ablation for CtCAE on `CLEVR`.

that this is because these two contrastive strategies are complementary: one uses low-level cues (dec) and the other high-level abstract features (enc).

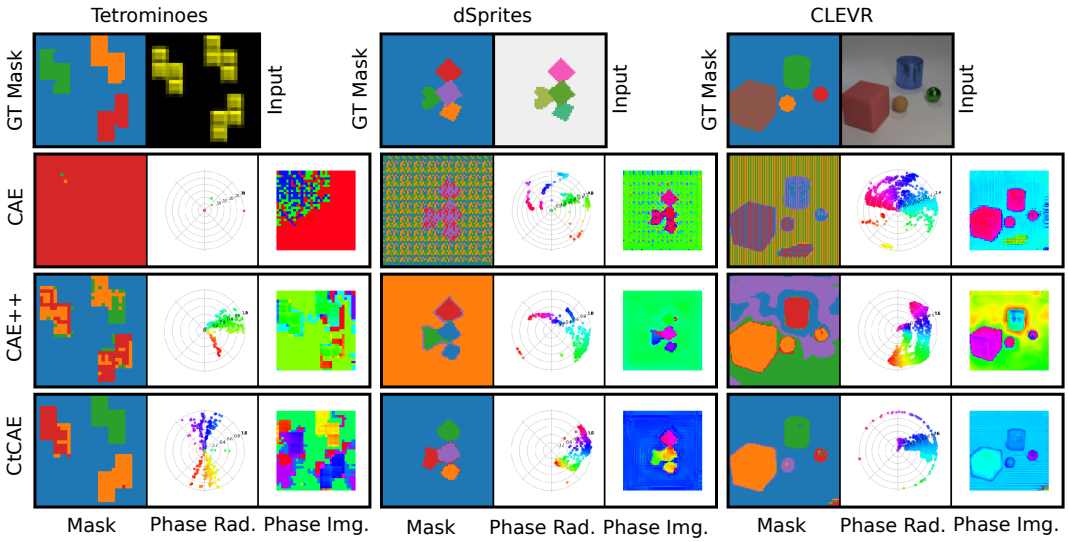


Figure 4.4: CAE, CAE++ and CtCAE comparison on `Tetrominoes` (columns 1-3), `dSprites` (columns 4-6) and `CLEVR` (columns 7-9). First row: ground truth masks and input images.

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 4.4, `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 *separability* 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 4.3 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 Appendix C.7). 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.7).

Discussion on the Evaluation Protocol. The reviewers raised some concerns about the validity of our evaluation protocol[4]. The main point of contention was that the thresholding applied before clustering the phase values may lead to a trivial separation of objects in RGB images based on their color. While it is true that in our case certain color information may contribute to the separation process, a trivial separation of objects based on solely their color cannot happen since CtCAE largely outperforms CAE++ (see Table 4.1) despite both having near perfect reconstruction losses and with the same evaluation protocol. This indicates that this potential issue only has a marginal effect in practice in our case and separation in phase space learned by the model is still crucial. In fact, "pure" RGB color tones (which allow such trivial separation) rarely occur in the datasets (`dSprites` and `CLEVR`) used here. The percentage of pixels with a magnitude less than 0.1 are 0.44%, 0.88% and 9.25% in `CLEVR`, `dSprites` and `Tetrominoes` respectively. While 9.25% in `Tetrominoes` is not marginal, we observe that this does not pose an issue in practice, as many examples in `Tetrominoes` where two or three blocks with the same color are separated by CAE++/CtCAE (e.g. three yellow blocks in Figure 4.4 or two magenta blocks in Appendix C.7).

To support our argument that using a threshold of 0.1 has no effect on the findings and conclusions, we conduct additional evaluations with a threshold of 0 and no threshold at all. From Table C.4 (see Appendix C.6) we can see that using a threshold of 0 hardly change the ARI scores at all or in some cases it even improves the scores slightly. Using no threshold results in marginally improved ARI scores on `CLEVR`, near-identical scores on `dSprites` but worse scores on `Tetrominoes` (CtCAE still widely outperforms CAE++). However, this is not due to a drawback in the evaluation or our model. Instead, it stems from an inherent difficulty with estimating the phase of a zero-magnitude complex number. The evaluation clusters pixels based on their phase values, and if a complex number has a magnitude of exactly zero (or very small, e.g., order of $1e^{-5}$), the phase estimation is ill-defined and will inherently be random as was also previously noted by Löwe et al. [2022]. In fact, filtering those pixels in the evaluation is crucial in their work [Löwe et al., 2022] as they largely used datasets with pure black background (exact zero pixel value). Evaluation of discrete object assignments from continuous phase outputs in a model-agnostic way remains an open research question.

[4] Discussion thread:
`https://openreview.net/forum?id=nF6X3u0FaA&noteId=5BeEERfCvI`

4.5 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.

4.6 Conclusion

We propose architectural improvements and a novel contrastive learning method to address limitations of current state-of-the-art synchrony-based model for perceptual grouping, CAE [Löwe et al., 2022]. Our improved model, CAE++, is the first synchrony-based model capable of dealing with color images (e.g., `Tetrominoes`). Our contrastive training further boosts CAE++ by improving its phase separation process. The resulting model, CtCAE, 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., 2022a], but this is to be expected, as research on modern synchrony-based models is still in its infancy. We hope our work will inspire the community to invest greater effort into such promising models.

Chapter 5

Recurrent Complex-weighted Autoencoders[1]

5.1 Overview

Previously, we saw that in the synchrony-based paradigm, bindings between features are expressed in terms of the relative phases of complex-valued neural activities. The earliest demonstration of a synchrony-based model in neural networks focused on a supervised setting with teacher-specified target phases for hard-coded one-hot features of contour types [Mozer et al., 1992], although consideration was given to an unsupervised extension obtained by phase clustering [Mozer, 1998]. Recent synchrony-based models [Reichert and Serre, 2014; Löwe et al., 2022; Stanić et al., 2023; Löwe et al., 2023] using complex-valued activations to learn suitable features have made progress in unsupervised grouping performance on synthetic and more naturalistic scenes.

However, current state-of-the-art unsupervised synchrony-based models such as CAE ([Löwe et al., 2022]), CtCAE ([Stanić et al., 2023]), and RF ([Löwe et al., 2023]) have a number of limitations. They employ real-valued weights to process complex-valued activations by weight sharing across the real and imaginary components. As a result, the models do not exploit the constructive or destructive interference of complex-valued activations that occur naturally via

[1] For the full paper please see "Recurrent Complex-Weighted Autoencoders for Unsupervised Object Discovery" [Gopalakrishnan et al., 2024]

multiplication and addition operations (cf., [Hinton, 2023]). And the models require use of additional inductive biases such as gating mechanisms (χ-binding [Reichert and Serre, 2014; Löwe et al., 2022, 2023; Stanić et al., 2023]) to implement binding. Further, it remains unclear how χ-binding mechanism works in more general scenarios where the complex-valued activations are not exactly out-of-phase or norms of weights and features do not satisfy certain conditions as noted by Löwe et al. [2024]. Cosine binding [Löwe et al., 2024] was proposed as a more computationally motivated and interpretable alternative that easily handles such scenarios. However, this model continues to use real-valued weights to process complex-valued activations. Therefore, it uses cosine distance between inputs and intermediate outputs to implement binding which has a large memory overhead.

These state-of-the-art synchrony-based models [Löwe et al., 2023; Stanić et al., 2023] are not as robust as one might hope. As we show later, these models tend to exploit color as a shortcut feature, a helpful heuristic when different objects have different colors but a harmful one when color is an unreliable shortcut feature. And on more naturalistic datasets, RF largely learns semantic-level groups (see, e.g., Figures 2 & 20 in [Löwe et al., 2023]) rather than instance-level groups which is the central focus of object-centric learning. These state-of-the-art models require either an additional contrastive loss (see, e.g., Figure 8 in [Stanić et al., 2023]) or pre-trained features from self-supervised vision transformers or 'depth masks' even on synthetic datasets (see, e.g., Figure 6 in [Löwe et al., 2023]) to partially resolve such grouping errors.

Many of the complexities introduced in recent models seem to be moving the field away from the core intuition that initially motivated a synchrony-based approach to binding [Mozer et al., 1992]. We step back and describe the original conception of synchrony-based grouping mechanisms from Mozer et al. [1992]. Consider the shapes composed of horizontal and vertical bars in Figure 5.1: the letters $\mathbb{T}$ and $\mathbb{H}$ and a pair of overlapping squares with occlusion. Perceptual grouping involves determining which of the bars

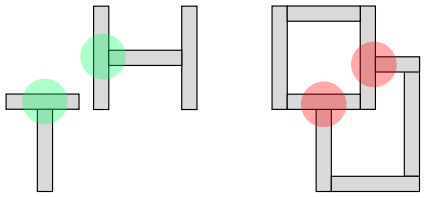


Figure 5.1: Configurations of local features are insufficient to determine whether they belong to the same object: highlighted horizontal and vertical junctions sometimes belong to the same object (green) and at others to different ones (red).

belong together. The pairs highlighted in green are part of the same object; the pairs highlighted in red are parts of different objects. A feedforward convolutional architecture is not well suited to this task because spatially local patches are ambiguous with regard to grouping. While a feedforward, fully connected architectures may work in principle, it is also problematic because statistical regularities in images are local and fairly low order (i.e., involve a subset of image features). However, if we drop the feedforward restriction, a convolutional model can iteratively converge on a solution: each iteration suggests soft constraints on how features should be grouped, and over iterations, constraints can propagate from one region in the image to adjacent regions. This methodology was adopted in classical AI vision models [Waltz, 1972] and in early synchrony-based models [Mozer et al., 1992]. In the latter, phases—representing object binding—are initialized to random values and each iteration, the phases are updated to be consistent with constraints from neighboring patches. After multiple iterations the model converges on an interpretation that assigns each image location to a particular phase (object).

Complex weights are a natural choice to instantiate this function in our synchrony formulation outlined above. Just as hidden units in a standard architecture detect configurations of features, hidden units in a complex-valued analogue detect configurations of features in a particular relative phase arrangement. A hidden unit that is tuned to the top of a $\mathbb{T}$ will expect the horizontal and vertical bar to have the same phase; a hidden unit that is tuned to occlusion (the red patches in Figure 5.1) will expect the horizontal and vertical bars to be out of phase. In this manner, the hidden units can encode alternative hypotheses concerning object groupings, and iterative processing will attempt to find global configurations that are locally consistent.

Drawing on these intuitions we design a model and training strategy for the fully unsupervised setting. Figure 5.2 shows our proposed model, **Syn**chronous **C**omple**x** Network (SynCx), which uses complex-valued weights (detectors) to process complex-valued activations (features) in every layer of a fully convolutional autoencoder. The matrix vector product between complex-valued weights and activations ensures that weights process inputs not only based on their features (magnitudes $\rightarrow$ features) but also their phase relationships (phases $\rightarrow$ object bindings). The addition and multiplication operators in complex algebra mechanistically implement the binding mechanism in our model removing the need for additional gating mechanisms (χ [Reichert and Serre, 2014] / cosine binding [Löwe et al., 2024])

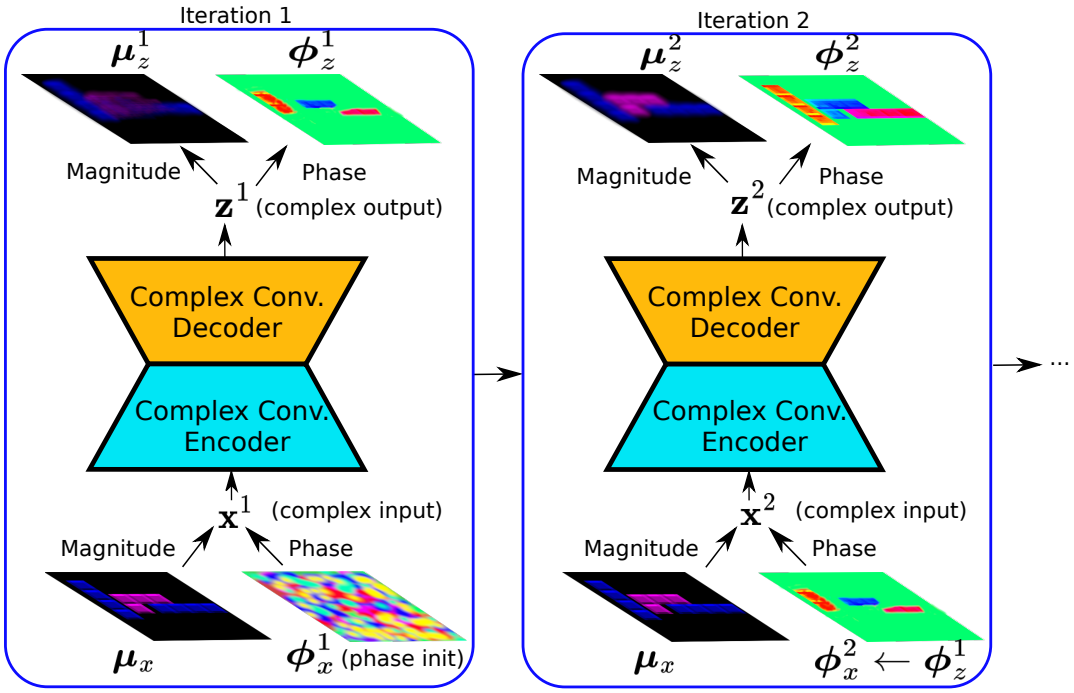


Figure 5.2: SynCx is a fully convolutional autoencoder that iteratively processes an input image. It starts with a randomly initialized phase ϕ_x^1 and the input image μ_x in the magnitude-component updates the phases in a stateful manner, i.e., output phase at iteration 1 fed as input at iteration 2 ($\phi_x^2 \leftarrow \phi_z^1$) and so on. The magnitude-component at the input is always clamped to the input image μ_x. SynCx is trained to reconstruct μ_x using the output magnitude-component μ_z^n at every step.

or contrastive training [Stanić et al., 2023]. The spatial locality of constraints are ensured via 2D convolutions used at every layer in the autoencoder. The model is initialized with a random phase map (prior on object bindings) and learns to (de)synchronize groups of features based on their (dis)similarity iteratively. The autoencoder weights are shared across iterations and predicted output phases (top-down feedback) from the previous iteration are used as the input phases at the next iteration while the pixel values are the input magnitudes at each step. Iteratively updating the phases allows spatially local constraints to be slowly propagated across the spatial axes and in principle should lead to the system settling to a fixed point. The autoencoder is trained to simply reconstruct the input image at each iteration.

In contrast to the synchrony formulation of Mozer et al. [1992], all recent state-of-the-art models use purely feedforward processing of inputs. Therefore, these models lack the ability to propagate information about local constraints to align phases which we show ($\mathbb{T}$ and $\mathbb{H}$ junction examples) plays a key role in achieving binding via synchrony. We show that binding in synchrony-based models can be mechanistically implemented simply by matrix-vector products between complex-valued weights and activations to iteratively process inputs. We do not need any additional mechanisms like χ-binding [Reichert and Serre, 2014; Löwe et al., 2022, 2023] or cosine binding [Löwe et al., 2024] or contrastive training [Stanić et al., 2023] as in prior work to achieve the same. Our conceptually simpler model (SynCx) designed from first principles outperforms (on `Tetrominoes`) or competitive with (on `dSprites` and `CLEVR`) state-of-the-art synchrony-based models for unsupervised object discovery. RF groups objects using largely color cues requiring supervision via additional features ('depth masks') to segregate objects with similar colors. Whereas SynCx more gracefully separates objects of the same color as it relies on features (color/shape/texture) and the local spatial context removing the need for 'depth masks'. We visualize the phase maps across iterations to qualitatively inspect the binding process. Lastly, we discuss some remaining practical limitations of current state-of-the-art synchrony-based models.

5.2 Method

We describe the intuitions behind our model, its architecture and training procedure below.

Basic Intuition. Our autoencoder model is trained to reconstruct the input image using a representational bottleneck in the hidden layer(s). This requires the model to learn an efficient coding scheme to compress the input image. Images are typically composed of objects which are modular (reusable) units of feature information (shape, color, texture etc.) For a network with complex-valued activations, one intuitive coding scheme is using the phase components to express relationships between lower-order features (edges/textures/shape) in order to compress them into appropriate higher-order features (objects/parts). As the image is processed by the encoder it progressively compresses the input by storing and processing modular 'parts' together as higher-order features while the decoder learns the inverse mapping to recover the input from this compressed format. It is then possible to characterize objects

as the collection of features with similar phases across the entire spatial map.

Model. Our model (SynCx) is a fully convolutional autoencoder architecture. It uses *complex-valued* weights to manipulate *complex-valued* activations at each layer. Let $h, w, h', w', c, d_{\text{in}}, d_{\text{out}}$ and p denote positive integers. Every layer of the network is a parametric function $f_{\mathbf{W}} : \mathbb{C}^{h \times w \times d_{\text{in}}} \rightarrow \mathbb{C}^{h' \times w' \times d_{\text{out}}}$ which maps complex-valued inputs $\mathbf{x}$ to complex-valued outputs $\mathbf{h}$ using complex-valued weights $\mathbf{W} \in \mathbb{C}^p$. First, we compute the complex-valued pre-activation response $\mathbf{y}$:

$$\mathbf{y} = f_{\mathbf{W}}(\mathbf{x}), \quad \text{where } f_{\mathbf{W}} \text{ denotes a 2D convolution} \tag{5.1}$$

Then, we apply the modReLU activation rule [Arjovsky et al., 2016] element-wise only on the magnitude component of $\mathbf{y}$ (i.e., $\boldsymbol{\mu}_y$) to obtain the complex-valued layer output $\mathbf{h}$ as:

$$\mathbf{h} = \text{modReLU}(\mathbf{y}) = \text{ReLU}(\boldsymbol{\mu}_y + b) \odot e^{i\boldsymbol{\phi}_y} \;\; \text{with} \;\; \mathbf{y} \equiv \boldsymbol{\mu}_y \odot e^{i\boldsymbol{\phi}_y} \tag{5.2}$$

where $\odot$ denotes a Hadamard product and $b \in \mathbb{R}^{d_{\text{out}}}$ is a learnable parameter. We use modReLU instead of ReLU because the nonlinearity is applied only to the magnitude component (strictly non-negative). Consequently, the standard ReLU activation rule would operate only in its linear region. The modReLU activation rule [Arjovsky et al., 2016] was introduced specifically for a setting such as ours where the activation function is applied only to the magnitude components of complex-valued activations. The encoder module progressively downsamples the resolution of the complex-valued spatial feature map using strided 2D convolutions. The decoder module upsamples the resolution of the spatial feature map using the upsampling function independently on its magnitude and phase components.

Training and Loss Function. Given an image $\boldsymbol{\mu}_x \in \mathbb{R}^{h \times w \times c}$ of height h, width w, and c channels (3 for color images) with non-negative pixel values. We construct a complex-valued input with magnitudes $\boldsymbol{\mu}_x$ and phases $\boldsymbol{\phi}_x$ which are randomly initialized at every spatial location and channel of the image (see Figure 5.2). For each vector-valued feature at a given location, its phase component encodes the network's current hypothesis about its object binding; random initialization reflects lack of knowledge initially. The autoencoder processes the input image to update its hypotheses about object bindings at every iteration $n \in \{1, ..., N\}$. The complex-valued input $\mathbf{x}^n \in \mathbb{C}^{h \times w \times c}$ to the autoencoder at the nth iteration is:

$$\mathbf{x}^n = \boldsymbol{\mu}_x \odot e^{i\boldsymbol{\phi}_x^n} \tag{5.3}$$

Notice in Equation (5.3), that the magnitude components are always clamped to the image ($\boldsymbol{\mu}_x$) while the phase components ($\boldsymbol{\phi}_x^n$) are fed back. At the nth iteration, $\mathbf{x}^n$ is processed by the encoder and decoder layers, denoted as $\text{Net}(\mathbf{x}^n)$, to compute a complex-valued output $\mathbf{z}^n \in \mathbb{C}^{h \times w \times c}$. The magnitude component of $\mathbf{z}^n$, i.e. $\boldsymbol{\mu}_z^n$, is the reconstruction of input image $\boldsymbol{\mu}_x$. The phase component of $\mathbf{z}^n$, i.e. $\boldsymbol{\phi}_z^n$, is used to initialize the complex-valued input $\mathbf{x}^{n+1}$ at the next iteration:

$$\mathbf{z}^n = \text{Net}(\mathbf{x}^n) \quad ; \quad \boldsymbol{\phi}_x^{n+1} \leftarrow \boldsymbol{\phi}_z^n \tag{5.4}$$

This way the phase maps act as the state of a recurrent function which processes a clamped input image at all steps as shown in Figure 5.2. After N such iterative updates to the phase maps by the autoencoder, it is trained to minimize the average pixel-wise reconstruction loss across all iterations:

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^{N} ||\boldsymbol{\mu}_x - \boldsymbol{\mu}_z^n||^2 \tag{5.5}$$

We use the weight initialization scheme outlined by Trabelsi et al. [2018] for the complex-valued weights $\mathbf{W}$ at every layer of the encoder and decoder modules. This initialization scheme derives the variances of complex-valued weights to satisfy the initialization criterion of He et al. [2015]. The magnitude components of the complex-valued weights at every layer are sampled from a Rayleigh distribution with $\sigma = 1/\text{fan}_{\text{in}}$ while the phase components are sampled from a uniform distribution between 0 to 2π.

Through ablation studies in the following section, we quantify the effects of key components such as random initialization of the phase map, presence of a bottleneck at latent layer(s) and the number of iterative updates have on the performance of our system.

Extracting Object Assignments. Here, we outline the processing steps used to extract discrete object assignments for every pixel given its continuous phase values. We start by taking the output complex-valued feature map $\mathbf{h}^N \in \mathbb{C}^{h \times w \times d_{\text{out}}}$ from the penultimate decoder layer at the last iteration N. We use latent-level complex-valued features instead of output-level so as to use high-order features (texture/shape/color etc.) than simply color cues (RGB space) to determine their object constituency. Then, we construct a complex-valued feature map with unit magnitudes (i.e., $\boldsymbol{\mu}_h^N \leftarrow \mathbf{1} \in \mathbb{R}^{h \times w \times d_{\text{out}}}$) and phases the same as that of $\mathbf{h}^N$ (i.e., $\boldsymbol{\phi}_h^N$). Background regions in this constructed feature map are masked out by setting their magnitudes to zero and then converted to

the Cartesian form element-wise. Finally, these features in Cartesian form are clustered using k-Means to compute the object assignments for every location in the spatial map. Using unit magnitudes for complex-valued activations ensures that object assignments are solely determined by their orientations. We note that it would be most natural to cluster the phases from the output layer at each location to assign it to an object. However, we observed that the object separation shown in the output phases is weaker than that from the high-dimensional latent layer. Therefore, for all results presented here we choose to cluster the higher dimensional latent representations (from the penultimate decoder layer) by location. For more details on the object assignment process refer to Appendix D.6.

5.3 Results

First, we describe details of the datasets, our model/baselines, training and evaluation procedures. We visualize the phase maps and qualitatively characterize the grouping behavior shown by our model. We quantitatively compare the grouping performance on the unsupervised object discovery task of our model (SynCx) against state-of-the-art synchrony-based baselines (CAE, CAE++, CtCAE, and RF). We also quantify the effects of key components of our model using ablation experiments. We conclude with some practical limitations shown by the binding mechanisms in these synchrony-based models.

Datasets. We evaluate models on three datasets from the multi-object suite [Kabra et al., 2019] namely `Tetrominoes`, `dSprites` and `CLEVR` used by prior work in object-centric learning [Greff et al., 2019; Locatello et al., 2020; Emami et al., 2021]. For `CLEVR`, we use a filtered version [Emami et al., 2021] which consists of images containing less than seven objects. In all experiments we use image resolutions identical to Emami et al. [2021], i.e., 35x35 for `Tetrominoes`, 64x64 for `dSprites` and 96x96 for `CLEVR` (center crop of 192x192 resized to 96x96). 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 the evaluation metrics. For further details about datasets and preprocessing, we refer to Appendix D.1.

Model & Training Details. For more details about the encoder and decoder architecture of our model we refer to Appendix D.2. We train our model for 40K steps on `Tetrominoes`, and 100K steps on `dSprites` and `CLEVR` with

Adam optimizer [Kingma and Ba, 2015b] with a constant learning rate of 5e-4, i.e., no warmup schedules or decay (all hyperparameter details are given in Appendix D.3). The phase initialization at each spatial location and channel of the input image are independent samples from a von-Mises distribution with a mean of 0 and concentration of 1 [2].

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]. The ARI scores are measured only for the foreground pixels common practice in the object-centric literature.

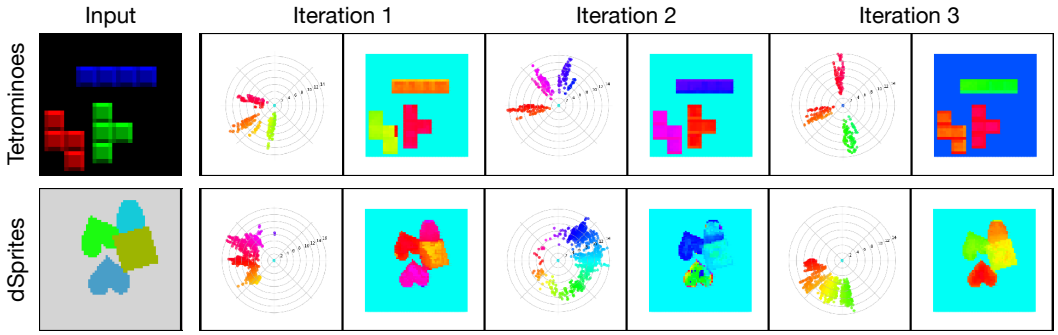


Figure 5.3: Evolution of phase maps in radial and heatmap form (colors matched) across iterations in SynCx for two inputs from `Tetrominoes` (row 1) and `dSprites` (row 2).

Phase Map Visualization. We seek to qualitatively inspect phase components of the output complex-valued feature map $\mathbf{h} \in \mathbb{C}^{h' \times w' \times d_{\text{out}}}$ from an intermediate decoder layer to see if our model has indeed learned phases specialized in an object-centric manner. We visualize the phase component of the high-dimensional feature map as a heatmap in two dimensions. We assign unit magnitudes and retain the phase components of $\mathbf{h}$ and elementwise convert these to the Cartesian form as we did previously to extract object assignments. We apply t-SNE [van der Maaten and Hinton, 2008] to perform dimensionality reduction on the complex-valued feature map of an input image to recover a scalar 'composite' phase value at every spatial location (see Appendix D.5 and Figure D.4 for more details). This 'composite' phase map is visualized as a two dimensional heatmap and radial plot with colors in the former corresponding to

[2] Code repository for the paper: `https://github.com/agopal42/syncx`.

orientations in the latter (see columns 2 & 3 in Figure 5.3). From Figure 5.3 shows how the phase maps evolve across the iterations in SynCx grouping process. In the sample image from `Tetrominoes` (row 1 in Figure 5.3), we can see how the phase 'bands' corresponding to each tetris block are progressively separated in their orientations. Such a visualization gives an interpretable artefact that allows a qualitative inspection of the phase synchronization process.

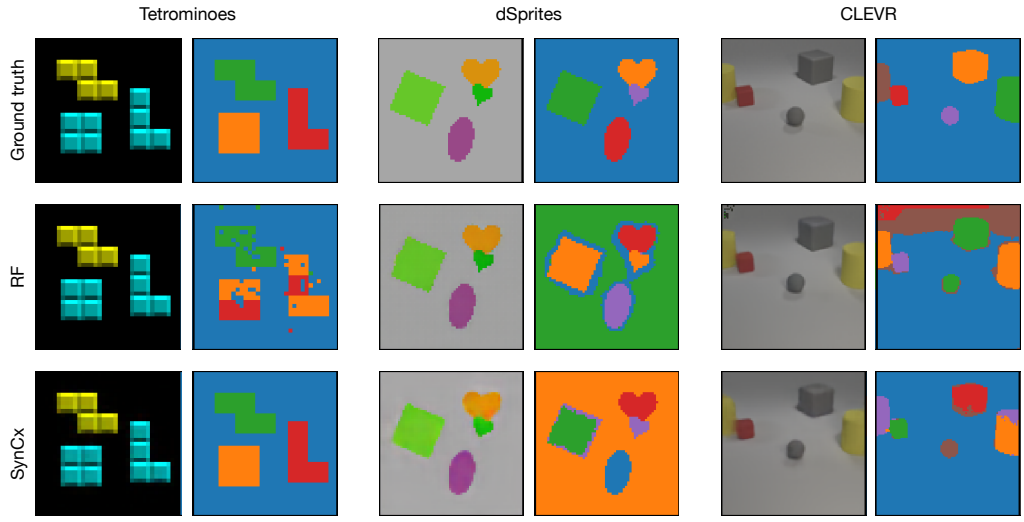


Figure 5.4: Comparison between RF and SynCx grouping on `Tetrominoes`, `dSprites` and `CLEVR`. RF tends to systematically group similarly colored objects together while SynCx is more adept at separating them such as blue tetris blocks (left), green square and heart (middle) and yellow cylinders (right).

Unsupervised Object Discovery. In Table 5.1 we compare the performance of recent state-of-the-art synchrony-based baselines (CAE, CAE++, CtCAE, and RF) against ours (SynCx) on the unsupervised object discovery task. On `Tetrominoes`, we observe that SynCx outperforms all baselines on grouping performance. SynCx more gracefully separates objects of the same color (see Figure 5.4) compared to RF which in addition to complex-valued activations requires χ-binding mechanism. SynCx also outperforms the CtCAE baseline which in addition to complex-valued activations requires χ-binding and contrastive training to separate similarly colored objects. On `dSprites`, SynCx significantly outperforms CAE, CAE++ and CtCAE baselines while being strongly competitive with RF. Similarly on `CLEVR`, SynCx strongly outperforms CAE, CAE++ and CtCAE baselines while being competitive with RF. Overall,

SynCx despite its simple binding mechanism (no gates) and training strategy (no contrastive training) outperforms or strongly competitive with all state-of-the-art synchrony-based models. However, there is still a significant gap in grouping performance between synchrony-based models with respect to the dominant slot-based models (*SlotAttention* [Locatello et al., 2020]) especially on `CLEVR`. We refer to Appendix D.7 for the model comparisons that also includes the MSE loss. We refer to Appendix D.8 for additional qualitative grouping examples from our model.

Model	Tetrominoes	dSprites	CLEVR
CAE	0.00 $\pm$ 0.00	0.05 $\pm$ 0.02	0.04 $\pm$ 0.03
CAE++	0.78 $\pm$ 0.07	0.51 $\pm$ 0.08	0.27 $\pm$ 0.13
CtCAE	0.84 $\pm$ 0.09	0.56 $\pm$ 0.11	0.54 $\pm$ 0.02
RF	0.42 $\pm$ 0.09	0.84 $\pm$ 0.03	0.65 $\pm$ 0.01
SynCx	0.89 $\pm$ 0.01	0.82 $\pm$ 0.01	0.59 $\pm$ 0.03
SlotAttention	0.99 $\pm$ 0.01	0.91 $\pm$ 0.01	0.99 $\pm$ 0.01

Table 5.1: ARI scores (mean $\pm$ standard deviation across 5 seeds) for CAE, CAE++, CtCAE, RF, SynCx and SlotAttention on `Tetrominoes`, `dSprites` and `CLEVR`. Results for CAE, CAE++ and CtCAE baselines taken from Stanić et al. [2023] and for SlotAttention taken from Locatello et al. [2020].

Below, we quantify the effects of key components of our model such as representational bottlenecks, number of iterations and phase initialization have on its grouping performance.

Effect of Bottlenecks. We measure the effect of having representational bottlenecks (i.e. spatial resolution of feature maps) in the hidden layers of our model. To do so, we design a variant of our model (denoted as SynCx w/o bottl. in Table 5.2) that preserves the spatial resolution of feature maps in the encoder (using stride of 1 in all layers) thereby removing all bottlenecks.

We ensure that both these variants (SynCx and SynCx w/o bottleneck) have the same number of parameters. Instead we only vary the spatial resolution of feature maps, so the bottleneck is w.r.t spatial resolution and not the number of model

Model	MSE $\downarrow$	ARI $\uparrow$
SynCx w/o bottl.	6.29e-5 $\pm$ 9.00e-5	0.10 $\pm$ 0.06
SynCx	2.07e-3 $\pm$ 1.09e-4	0.89 $\pm$ 0.01

Table 5.2: Ablation on use of hidden layer bottlenecks on `Tetrominoes`.

parameters. We observe a sharp drop in grouping performance for our ablated model variant without any bottlenecks despite it achieving a lower test loss (see Figure 5.5). This result supports our initial intuition that representational bottlenecks force an autoencoder with complex-valued activations to compress spatial regularities in features by using phase components to capture relationships between them. Since objects are modular units containing highly regular features within their boundaries, this leads to phases strongly specializing (synchronizing) towards them to facilitate better image compression.

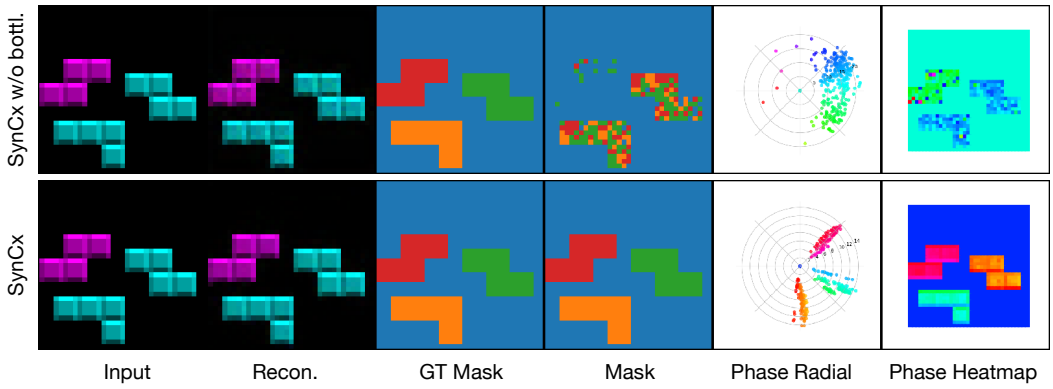


Figure 5.5: Reconstruction, object masks, radial phase plot and phase heatmaps (colors matched between columns 5 & 6) for SynCx without the bottleneck (row 1) and the full SynCx model (row 2).

Effect of Iterations. We measure the effect of multiple iterations to update phases on the grouping performance of our model. Table 5.3 shows the grouping performance of our model with increasing number of iterations used during training. We observe a general trend of increasing phase specialization towards objects and better reconstructions as we increase the number of iterations. This supports our qualitative characterization of the models' operation — starting with random object assignments it iteratively refines the hypotheses for the assignment for each pixel.

No. of Iters.	MSE ↓	ARI ↑
1	3.49e-3 ± 1.11e-4	0.57 ± 0.01
2	2.20e-3 ± 6.24e-5	0.81 ± 0.03
3	1.73e-3 ± 1.42e-4	0.82 ± 0.01
4	1.73e-3 ± 3.85e-4	0.79 ± 0.03

Table 5.3: Ablation on number of iterations used in training on `dSprites`.

We also measure the effect of increasing the number of iterations at test-time compared to that used during training. Table 5.4 shows the grouping performance of our model as we increase the number of iterations from 4 to 6 while during training the model used 3 iterations. We observe no drop in object separation performance of our model when we extrapolate the number of iterations used at test-time.

No. of Iters.	MSE ↓	ARI ↑
4	1.71e-3 ± 1.38e-4	0.81 ± 0.01
5	1.71e-3 ± 1.40e-4	0.82 ± 0.01
6	1.70e-3 ± 1.41e-4	0.82 ± 0.01

Table 5.4: Ablation on the number of iterations used at test-time on `dSprites`.

Effect of Phase Initialization. We measure the effect that the initialization of phases has on the grouping performance of our model. To achieve this, we compare (see Table 5.5) the grouping performance of three variants — i) initial phases sampled from a uniform distribution between -π and π ii) initial phases sampled from a von-Mises distribution with mean of 0 and concentration of 1 and iii) all values in the initial phase map are set to zero.

The rest of the model and training hyperparameters are kept the same between these two variants. The variant using von-Mises distribution to sample initial input phases achieves the best test loss and improved grouping scores among all variants. This suggests the variance of the noise distribution is an important hyperparameter to tune for phase synchronization. The von-Mises distribution with a mean of 0 and concentration of 1 (circular analogue of Normal distribution) and samples "noisy" phase values with an intermediate level of variance compared to the other two options which have maximum and zero variance. We use the von-Mises variant as the default phase initialization.

Phase Init.	MSE ↓	ARI ↑
Zero	2.46e-3 ± 5.08e-7	0.74 ± 0.04
Uniform	1.02e-2 ± 8.07e-3	0.75 ± 0.05
von-Mises	2.07e-3 ± 1.09e-4	0.89 ± 0.01

Table 5.5: Ablation on phase initialization types on `Tetrominoes`.

Parameter and Training Efficiency Table 5.6 shows parameters counts for various synchrony-based models. Our model (SynCx) is 6-23x more parameter efficient compared to other synchrony-based alternatives. Table 5.7 shows the wall-clock training times for convergence of SynCx

Model	Tetrominoes	dSprites
RF	7h 20m	4h 50m
SynCx	1h 35m	4h 10m

Table 5.7: Wall-clock training times for models on a P100 GPU.

Model	Tetrominoes	dSprites	CLEVR
CAE++	5,372,000	11,713,000	22,281,000
CtCAE	5,372,000	11,713,000	22,281,000
RF	6,630,000	11,861,000	22,592,000
SynCx	966,000	974,000	974,000

Table 5.6: Comparison of parameter counts (rounded up to the nearest thousand) of various synchrony-based models. Total number of parameters expressed in terms of number of real-valued floats.

against the strongest synchrony baseline (RF) on `Tetrominoes` and `dSprites`. Our model on these datasets consistently takes lesser wall-clock time for training in these settings. These results highlight the computational benefits of the simple ideas of complex-valued weights and recurrence that are the essence of our model design.

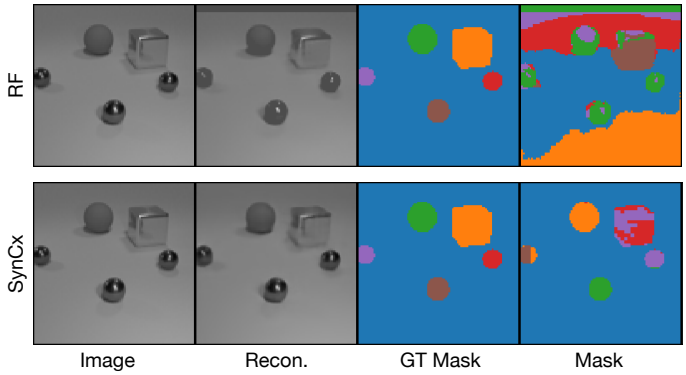


Figure 5.6: Reconstruction and object masks for RF and SynCx on grayscale `CLEVR`. While both models reconstruct well, RF struggles to group the objects without access to color unlike SynCx.

Limitations. We test how reliant on color cues are the two best performing synchrony-based models from Table 5.1, i.e., RF and SynCx, for unsupervised object discovery. We train and evaluate them on a grayscale version of the `CLEVR` dataset. To separate objects in this dataset, the models' would need to rely more on other features like shape, texture, etc. From Table 5.8, we see that the grouping performance of RF on grayscale `CLEVR` drops sharply compared to the original (see Table 5.1) indicating its heavy reliance on color cues. Our models' grouping performance also drops but to a much lesser degree compared to RF.

This indicates a weaker dependence on color cues in the binding mechanism of our model. In fact, on grayscale CLEVR SynCx significantly outperforms RF despite having slightly worse performance on the original colored version.

Model	MSE ↓	ARI ↑
RF	5.64e-4 ± 3.76e-4	0.22 ± 0.04
SynCx	2.49e-4 ± 3.76e-5	0.45 ± 0.01

Table 5.8: Grayscale CLEVR.

The binding mechanism (i.e., matrix-vector product) in our model updates phases based on agreement between feature detectors (complex-valued weights) and high-dimensional attributes (complex-valued activations) that model color, edges, texture, shape etc. while taking it account their phase relationships (local context). This facilitates it to bind to objects more robustly and not simply rely on color. Overall, we can see that despite its simple architecture and training procedure our model shows good grouping performance on a wider range of visual environments. However, binding mechanisms in state-of-the-art synchrony models are far from perfect. They are unable to reliably capture objects even on synthetic datasets like CLEVR containing simple 3D shapes with metallic or matte textures under camera lighting with a non-textured background.

5.4 Related Work

Slot-based Binding. Recent years have seen a growing number of models for unsupervised perceptual grouping (see Greff et al. [2020] for a summary). These models maintain the separation of information using a separate set of activations ('slots') but differ in the segregation mechanism to infer their contents. Representational symmetry in 'slots' is broken via temporal ordering [Eslami et al., 2016; Kosiorek et al., 2018], spatial ordering [Crawford and Pineau, 2019; Stanić et al., 2021; Lin et al., 2020], type-specificity [Hinton et al., 2018], iterative routing procedures [Greff et al., 2016, 2017, 2019; Locatello et al., 2020; Wu et al., 2023] (cf. earlier work [Schmidhuber, 1992]) or combinations thereof [Burgess et al., 2019; Engelcke et al., 2020; Singh et al., 2022b]. Extensions of slot-based grouping models for videos [Kipf et al., 2022; Elsayed et al., 2022; Singh et al., 2022c; Zadaianchuk et al., 2023], novel view synthesis [Li et al., 2020; Stelzner et al., 2021; Sajjadi et al., 2022] as well as other modalities than vision including speech [Reddy et al., 2023], music [Gha et al., 2023] and actions [Gopalakrishnan et al., 2023] have been explored. Conceptual limitations of slot-based models include separation only

maintained at one level, binding information stored in hard-wired architectural components unsuited for gradient-based adaptation, uniform capacity, inability to store object-level relational factors and high computational cost for training as noted by Stanić et al. [2023].

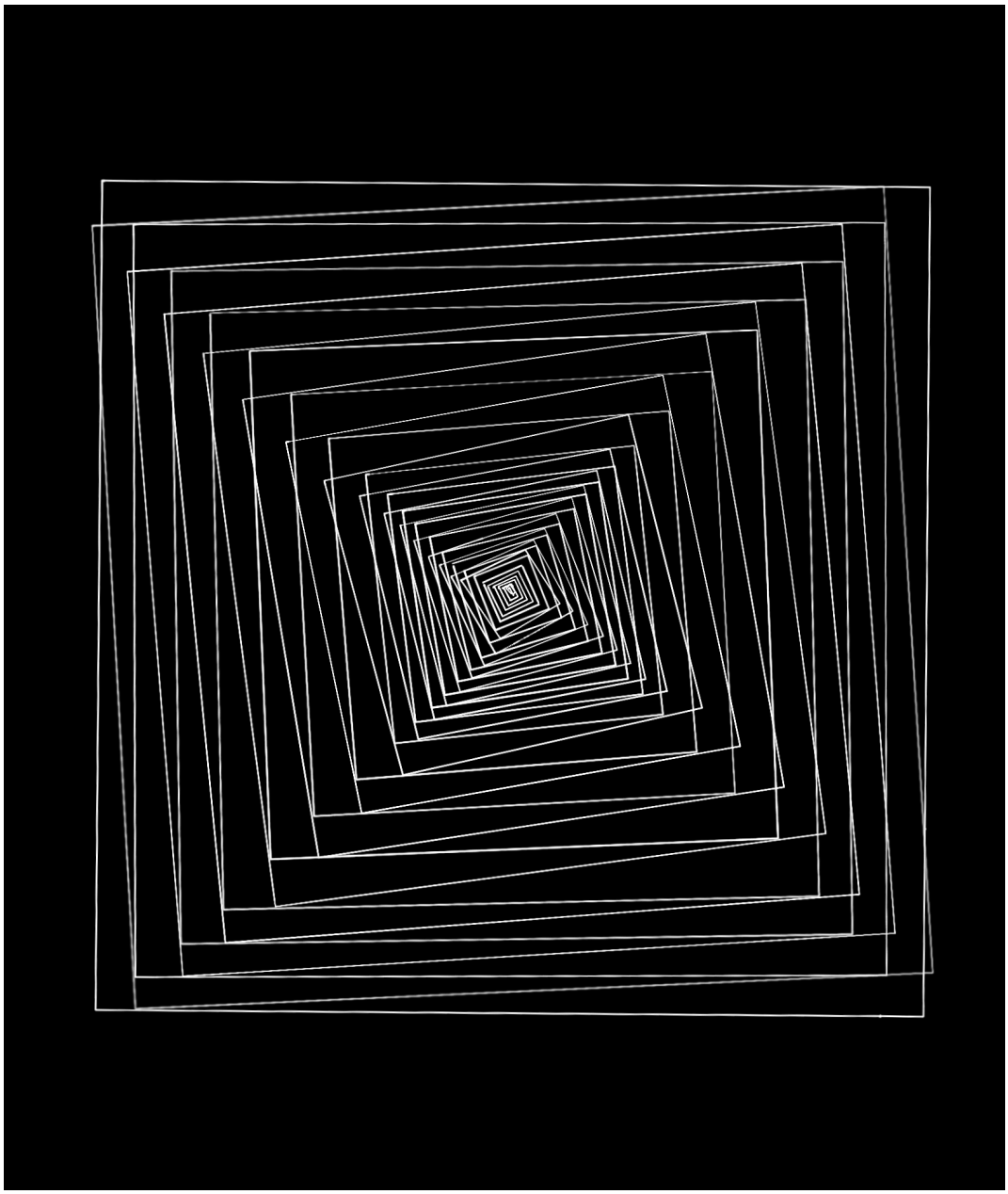
Synchrony-based Binding. Synchrony-based models could in principle address the above shortcomings of slot-based models but have so far received comparatively little attention. They resolve the binding problem by augmenting each activation with additional grouping features. They can be broadly divided into two classes — *temporal* and *complex*, based on the type of coding strategy used for the augmentation. Models using temporal codes rely on spiking neurons with rhythmic firing behavior [von der Malsburg and Schneider, 1986; von der Malsburg and Buhmann, 1992; Wang, 2005]. In such models, features of an object are expressed by neurons that fire in-sync. However, spiking neurons are non-differentiable and therefore incompatible with gradient-based learning, requiring specialized learning algorithms difficult to scale with current hardware accelerators for training. Models using complex-valued codes [Mozer et al., 1992; Mozer, 1998; Rao et al., 2008; Reichert and Serre, 2014; Löwe et al., 2022] are based on complex-valued activations suitable for scalable training using backpropagation. These models have been empirically benchmarked only on binarized images of simple geometric shapes or MNIST digits. More recent models [Löwe et al., 2023; Stanić et al., 2023] show improved unsupervised grouping performance on more visually challenging color images from the multi-object suite [Kabra et al., 2019]. However, these recent models heavily rely on supervision in the form of 'depth masks', gating mechanisms, contrastive training or a combination thereof. In contrast, our model is simply a fully convolutional autoencoder with complex-valued weights that iteratively updates phases to reconstruct an input image. Binding by synchrony has also been investigated using coupled oscillator models [Miyato et al., 2025; Muzellec et al., 2025a] where phase synchronization is achieved by treating the set of activations as a population of coupled oscillators which reach a steady (synchronized) state through repeated application of the Kuramoto update [Kuramoto, 1975]. Phase synchrony has also been explored in other task settings beyond grouping such as object tracking. For instance, Muzellec et al. [2025b] design recurrent networks that use complex-valued activations to precisely control attention to features separately from their location in the world.

Temporal Correlation Hypothesis. Synchrony-based models draw functional inspiration from the *temporal correlation hypothesis* [von der Malsburg, 1995; Singer and Gray, 1995] in neuroscience. It posits that the brain uses synchronized dynamics of neuronal firings to bind together distributed feature information computed in parallel at different areas into coherent percepts. Neural synchrony is also believed to convey information about relationships between features needed for dynamic and context-dependent binding by *'relational coding'* [Singer, 1999]. This is in contrast to *'labeled line coding'* where each unit has a fixed label attached to it indicating the static feature conjunction being active. Our use of complex-valued weights to process complex-valued activations is akin to the *'relational coding'* scheme.

5.5 Conclusion

We explored synchrony-based binding with an architecture, SynCx, that differs from current models in three respects: i) SynCx's weights are complex valued, allowing it to encode joint feature-phase configurations in the weights; ii) SynCx is recurrent and stateful, allowing it to perform iterative constraint propagation; and iii) SynCx has an internal representational bottleneck, which require it to use the phases of complex-valued activations to encode statistical regularities in images. These three properties are crucial to extending object-centric phase synchronization behavior to the fully unsupervised setting. Our conceptually elegant model outperforms all state-of-the-art baselines on `Tetrominoes` and overcomes a common failure mode of these baselines in which similarly colored objects are grouped together. Over-reliance on color cues for grouping by current models is further highlighted on a grayscale variant of the `CLEVR` dataset where we see that the RF model struggles to group objects using other features (shape and texture), in contrast to SynCx. Our model shows strong performance compared to more sophisticated synchrony-based baselines [Löwe et al., 2023; Stanić et al., 2023] on `dSprites` and `CLEVR` without the need for additional supervision such as 'depth masks', gating mechanisms or contrastive training. Starting with our model as a simple template, future directions to explore include the use of a temporal difference style loss to weight reconstruction targets and incorporate spatial priors into the binding mechanism. The process of extracting discrete object assignments from continuous phase maps by clustering could be improved by accounting for outliers, non-Gaussian distributed phase values with unequal sizes and variances. While recent synchrony-based models have been steadily closing the

gap in grouping performance to well-established slot-based approaches such as SlotAttention [Locatello et al., 2020] on simple synthetic datasets (`Tetrominoes` and `dSprites`) the gap remains significant on `CLEVR`. Further, they are yet to be scaled up to more challenging multi-object visual benchmarks (e.g. MOVi [Greff et al., 2022]) which offers an open challenge for future work.



Chapter 6

Polar Coordinate Positional Embedding

6.1 Overview

In the prehistory of deep learning, a key challenge was representing sequential or position-coded data. Tasks of interest included recognizing words from letter orderings [McClelland and Rumelhart, 1981], recognizing speech from power spectra [Waibel et al., 1989], compression of sequences with long time-lags between predictable events [Schmidhuber et al., 1993], classifying hand-drawn symbols from strokes [Yaeger, 1996], and time-series forecasting [Mozer, 1993].

Two common approaches were recurrent neural nets (RNNs) and slot-based encodings. RNNs implicitly encode sequence position in the ordering of inputs. By contrast, slot-based encodings partition an input vector into separate components for each sequence step. A significant advantage of RNNs is their approximate *translation equivariance*, meaning that shifting the absolute position of a sub-sequence in the input yields analogous shifts in the model state. Slot-based representations do not share this property: learning to respond to content in one slot does not generalize to the same content in other slots.

The transformer architecture [Vaswani et al., 2017] changed the game for slot-based representations. Through its self-attention mechanism, the basic transformer is not only translation equivariant but also translation and permutation *invariant* (subject to causal attention boundaries). The challenge

with transformers thus became how to obtain translation equivariance without losing sensitivity to the relative positions of input elements. The solutions that emerged involve enriching latent representations to encode not only their content—the *what*—but also the relationship between their sequence positions—the *where* [Vaswani et al., 2017; Dai et al., 2019; Su et al., 2024]. These solutions rightly assume that both content and position are essential to modeling complex sequences like language.

In this work, we argue that the Rotary Position Embedding (RoPE) [Su et al., 2024], a widely adopted solution, entangles the what and where in a way that can impair model performance, particularly when decision-making requires independent matches on these two factors. We propose an alternative technique, called Polar Coordinate Position Embedding (PoPE), that shares the advantages of RoPE over other positional encodings while allowing the Transformer to implement rules in which the key-query match can be characterized as a conjunction of a content-based (what) match and a position-based (where) match. Although PoPE is only a minor modification of RoPE, it introduces a powerful inductive bias that improves data efficiency, asymptotic accuracy, and yields superior context-length generalization.

6.2 Background

RoPE [Su et al., 2024] is the dominant approach to incorporate positional information in current open-source frontier language models, e.g., Llama 3 [Grattafiori et al., 2024], DeepSeekv3 [Liu et al., 2024], Gemma 3 [Team et al., 2025], and Qwen3 [Yang et al., 2025]. It produces an attention score for each query-key pair that is based on both how well they match and their relative positions in the input sequence.

To explain RoPE, consider a specific attention head in a Transformer layer that matches a query in position m, denoted $\boldsymbol{q}_m$, to a key in position n, denoted $\boldsymbol{k}_n$. The key and query are d-dimensional vectors that are partitioned into $d/2$ two-dimensional components. We denote component $c \in \{1, \ldots, d/2\}$ of query and key by $\boldsymbol{q}_{mc}$ and $\boldsymbol{k}_{nc}$, respectively. RoPE first rotates each component c in the 2D plane by an angle proportional to the key and query positions. If $\boldsymbol{R}(\phi)$ is a 2×2 matrix that performs rotation by angle ϕ:

$$\boldsymbol{R}(\phi) = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{6.1}$$

Then rotated query and key are $\boldsymbol{R}(m\theta_c)\,\boldsymbol{q}_{mc}$ and $\boldsymbol{R}(n\theta_c)\,\boldsymbol{k}_{nc}$, where θ_c is a component-specific frequency. The formation of query (or key) components and their rotation in 2D are depicted on the left side of Figure 6.1.

The corresponding components of key and query are matched via a dot product and summed to obtain an attention score:

$$a_{mn} = \sum_{c=1}^{d/2} \left[\boldsymbol{R}(m\theta_c)\,\boldsymbol{q}_{mc}\right]^{\mathsf{T}} \left[\boldsymbol{R}(n\theta_c)\,\boldsymbol{k}_{nc}\right] = \sum_{c=1}^{d/2} \boldsymbol{q}_{mc}^{\mathsf{T}}\,\boldsymbol{R}((n-m)\theta_c)\,\boldsymbol{k}_{nc} \tag{6.2}$$

The rotation to bring components into alignment depends only on the relative positions of the key and query, not the absolute positions.

Our perspective on RoPE begins by re-expressing the key and query components from Cartesian to polar coordinates:

$$\boldsymbol{k}_{nc} = \boldsymbol{R}\Big(\psi(\boldsymbol{k}_{nc})\Big) \begin{bmatrix} \mu(\boldsymbol{k}_{nc}) \\ 0 \end{bmatrix} \quad \text{and} \quad \boldsymbol{q}_{mc} = \boldsymbol{R}\Big(\psi(\boldsymbol{q}_{mc})\Big) \begin{bmatrix} \mu(\boldsymbol{q}_{mc}) \\ 0 \end{bmatrix} \tag{6.3}$$

where $\mu(\boldsymbol{x}) = \sqrt{x_1^2 + x_2^2}$ and $\psi(\boldsymbol{x}) = \tan^{-1}(x_2/x_1)$ extract the magnitude and phase respectively of an input 2D vector $\boldsymbol{x}$. With this notation, we can compose the rotations and write the attention score as:

$$\begin{aligned} a_{mn} &= \sum_{c=1}^{d/2} \begin{bmatrix} \mu(\boldsymbol{q}_{mc}) & 0 \end{bmatrix} \boldsymbol{R}\Big((n-m)\theta_c - \psi(\boldsymbol{q}_{mc}) + \psi(\boldsymbol{k}_{nc})\Big) \begin{bmatrix} \mu(\boldsymbol{k}_{nc}) \\ 0 \end{bmatrix} \\ &= \sum_{c=1}^{d/2} \mu(\boldsymbol{q}_{mc})\mu(\boldsymbol{k}_{nc}) \cos\Big((n-m)\theta_c + \psi(\boldsymbol{k}_{nc}) - \psi(\boldsymbol{q}_{mc})\Big) \end{aligned} \tag{6.4}$$

This algebra makes clear that each component—while comprising two elements of the embedding—represents a single magnitude and also introduces, via $\psi(\boldsymbol{q}_{mc})$ and $\psi(\boldsymbol{k}_{nc})$, an adjustment to the relative position (phase) that yields the maximal response. Thus, both the key and the query confound information about the presence or absence of features (the 'what') and relative positions (the 'where'). Our goal is to decouple these two distinct sorts of information, specifically by removing the interaction term $\psi(\boldsymbol{k}_{nc}) - \psi(\boldsymbol{q}_{mc})$.

Readers can refer to Appendix E.1 for an equivalent formulation of RoPE (as shown above) expressed purely in complex algebra.

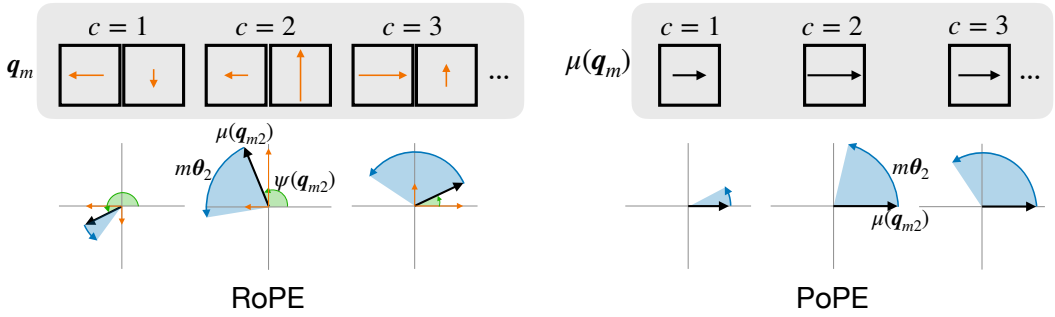


Figure 6.1: Illustration compares how RoPE and PoPE encode relative positions via rotations of queries. Left: Three complex-valued RoPE components having magnitudes $\mu(\boldsymbol{q}_{mc})$ (black arrows) and initial phases $\psi(\boldsymbol{q}_{mc})$ (green arcs) are constructed from six embedding features (orange arrows) of the query vector $\boldsymbol{q}_m$ (gray box) at sequence position m. These RoPE components are then rotated by angles $m\boldsymbol{\theta}_c$ (blue arcs). Right: Three magnitude components $\mu(\boldsymbol{q}_{mc})$ (black arrows) of complex-valued PoPE components are constructed from three embedding features of the query vector $\boldsymbol{q}_m$ (gray box) by applying Softplus activation. These magnitudes (complex numbers with zero phases) are then rotated by angles $m\boldsymbol{\theta}_c$ (blue arcs). PoPE uses twice the number of components than RoPE as it applies rotations in an element wise manner without creating chunks of two from the query vector.

6.3 Method

We reformulate RoPE using complex-valued vectors in the polar form. The magnitude of a complex-valued activation indicates the amount of presence of a feature while the phase indicates its sequence position. We describe the formulation of our method called PoPE below.

Just as the elements of the key and query in RoPE can be understood as complex numbers, we start with this premise for PoPE and define the attention score for a complex-valued key and query as:

$$\begin{aligned} \boldsymbol{a}_{mn} &= \Re\left[\boldsymbol{q}_m^H \, \boldsymbol{k}_n\right] \\ &= \Re\left[\sum_{c=1}^{d} \boldsymbol{q}_{mc}^H \, \boldsymbol{k}_{nc}\right] \\ &= \sum_{c=1}^{d} \mu(\boldsymbol{q}_{mc})\mu(\boldsymbol{k}_{nc}) \cos(\psi(\boldsymbol{k}_{nc}) - \psi(\boldsymbol{q}_{mc})) \end{aligned} \tag{6.5}$$

where H denotes the conjugate transpose which involves applying complex conjugation to each entry, $\boldsymbol{q}_m, \boldsymbol{k}_n \in \mathbb{C}^d$ are complex-valued queries and keys and as before $\mu(\boldsymbol{x}), \theta(\boldsymbol{x})$ compute the magnitude and phase of each element in a complex-valued vector $\boldsymbol{x}$. In contrast to RoPE, c is an index over individual elements of the keys and queries and not over pairs of elements. One way to define the key and query results in a formulation nearly identical to RoPE except it avoids the confound of the what and where signals is, if we form the complex-valued key and query with a real-valued linear transform of the token embedding, $\boldsymbol{z}$, as shown below:

$$\mu(\boldsymbol{k}_n) = \sigma(\mathbf{W}^k \boldsymbol{z}_n) \quad \text{and} \quad \mu(\boldsymbol{q}_m) = \sigma(\mathbf{W}^q \boldsymbol{z}_m) \tag{6.6}$$

where $\boldsymbol{z}_m, \boldsymbol{z}_n \in \mathbb{R}^d$ are the embeddings of tokens at positions m, n respectively and $\sigma(x) = \ln(1 + e^x)$ denotes the softplus activation function. The softplus activation ensures the non-negativity of outputs and maintains their semantics as valid magnitude components of complex-valued features. We use a fixed rotation (prespecified θ_c values) to determine the phase:

$$\psi(\boldsymbol{k}_{nc}) = n\boldsymbol{\theta}_c \quad \text{and} \quad \psi(\boldsymbol{q}_{mc}) = m\boldsymbol{\theta}_c \tag{6.7}$$

Thereby, we obtain a pure form of RoPE that avoids the what-where interaction and doubles the number of frequencies from $d/2$ to d. Further, each attention head might benefit from using a learnable bias term on the angle difference:

$$\begin{aligned} \boldsymbol{a}_{mn} &= \sum_{c=1}^{d} \mu(\boldsymbol{q}_{mc})\mu(\boldsymbol{k}_{nc}) \cos\Big(\psi(\boldsymbol{k}_{nc}) - \psi(\boldsymbol{q}_{mc}) + \boldsymbol{\theta}_b\Big) \\ &= \sum_{c=1}^{d} \mu(\boldsymbol{q}_{mc})\mu(\boldsymbol{k}_{nc}) \cos\Big((n-m)\boldsymbol{\theta}_c + \boldsymbol{\theta}_b\Big) \end{aligned} \tag{6.8}$$

where $\boldsymbol{\theta}_b \in \mathbb{R}^d$ is a learnable bias vector that tunes the optimal relative offset for each frequency c. Figure 6.1 visualizes the different ways by which RoPE and PoPE encode relative positions as rotations applied to queries and keys. There are several ways to initialize the bias vector $\boldsymbol{\theta}_b$. We find that two good options are to initialize it either as zeros or uniformly in the range $[-2\pi, 0]$. We scale $\boldsymbol{\theta}_b$ by the reciprocal of the rotational period, $\boldsymbol{\theta}_c$, along each component c. Further, we clamp $\boldsymbol{\theta}_b$ so that it to always lies in the interval $[-2\pi, 0]$, i.e. $\boldsymbol{\theta}_b = \min(\max(\boldsymbol{\theta}_b, -2\pi), 0)$ and found this improves stability. We find that the zero initialization is important for length generalization while the uniform one gives slightly better in-distribution performance.

Efficient Implementation. We implemented PoPE using Triton, starting from the example code for Flash Attention 2 [Dao, 2024] [1]. We modify the kernel to take complex-valued keys and queries in cartesian form, and compute the real part of their product inside the kernel, without ever materializing the resulting complex matrix of the key-query dot product. We can compute the Cartesian form of a complex-valued vector $\boldsymbol{q}$ from its polar form as:

$$\boldsymbol{q}_{\text{re}} = \mu(\boldsymbol{q}) * \cos(\psi(\boldsymbol{q})) \text{ and } \boldsymbol{q}_{\text{im}} = \mu(\boldsymbol{q}) * \sin(\psi(\boldsymbol{q})) \tag{6.9}$$

where $\boldsymbol{q}_{\text{re}}$ and $\boldsymbol{q}_{\text{im}}$ denote the real and imaginary components of a complex-valued $\boldsymbol{q} \in \mathbb{C}^d$. We can get the real and imaginary components of $\boldsymbol{k}$ in a similar way while treating the learnable phase-shifts $\boldsymbol{\theta}_b$ as an additional rotation by:

$$\boldsymbol{k}_{\text{re}} = \mu(\boldsymbol{k}) * \cos(\psi(\boldsymbol{k}) + \boldsymbol{\theta}_b) \quad \text{and} \quad \boldsymbol{k}_{\text{im}} = \mu(\boldsymbol{k}) * \sin(\psi(\boldsymbol{k}) + \boldsymbol{\theta}_b) \tag{6.10}$$

Then, note that dot product of complex-valued vectors in Cartesian form is computed as:

$$\begin{aligned} \boldsymbol{q}_m^H \boldsymbol{k}_n &= (\boldsymbol{q}_{\text{re}} - i\boldsymbol{q}_{\text{im}})(\boldsymbol{k}_{\text{re}} + i\boldsymbol{k}_{\text{im}}) \\ &= \boldsymbol{q}_{\text{re}}\boldsymbol{k}_{\text{re}} - i^2\boldsymbol{q}_{\text{im}}\boldsymbol{k}_{\text{im}} + i(\boldsymbol{q}_{\text{re}}\boldsymbol{k}_{\text{im}} - \boldsymbol{q}_{\text{im}}\boldsymbol{k}_{\text{re}}) \end{aligned} \tag{6.11}$$

The real-component of the dot product is the attention score $\boldsymbol{a}_{mn}$, i.e. $\Re(\boldsymbol{q}_m^H \boldsymbol{k}_n) = \boldsymbol{q}_{\text{re}}\boldsymbol{k}_{\text{re}} + \boldsymbol{q}_{\text{im}}\boldsymbol{k}_{\text{im}}$. Thus, our custom Flash Attention for PoPE requires only a single additional multiplication compared to the standard one. However, it requires twice the memory usage and bandwidth to store and load the complex-valued keys and values from the global memory. It is possible to avoid this memory overhead by loading the magnitudes of the keys and queries directly and performing the rotation inside the kernel. Now, the only overhead is the additional multiplication. We chose to go with the slower, but general variant, which takes as arguments complex-valued keys and queries in cartesian form and not perform the memory optimization, since this would prevent us from easy prototyping of different PoPE variants.

[1] https://triton-lang.org/main/getting-started/tutorials/06-fused-attention.html

6.4 Results

In all experiments, we compare our method, PoPE, to the popular RoPE [Su et al., 2024] scheme using two Transformers with identical model and training hyperparameters, with the only difference being their positional encoding schemes. For more details on how the datasets are generated, preprocessed, and tokenized, refer to Appendix E.2. In all experiments, we use a decoder-only Transformer architecture [Vaswani et al., 2017; Radford et al., 2018] with causal masking for autoregressive sequence modeling. The only change applied is the use of RMSNorm [Zhang and Sennrich, 2019] instead of LayerNorm [Ba et al., 2016] for normalization. For more details on the model configuration and training hyperparameters used for each experiment refer to Appendix E.3 and Appendix E.4 respectively.

Indirect Indexing. We introduce a task that requires identifying a target character within a variable-length source string. The target is defined to be at a specified relative offset (left or right) from a specified source character. For example, in the input `QEOHoUbKfeSrMVNlCzXu, z, -3, N`; the target `N` is three places to the left of source `z` in the string. Predicting the final (target) character of this sequence requires models to learn to independently manipulate the content and positional information of tokens and to apply pointer arithmetic operations. The dataset is constructed by procedurally generating examples of source strings, source symbols, and relative shifts. We compare RoPE [Su et al., 2024] against PoPE by training two Transformer models with cross-entropy loss applied only on the final (target) token and evaluated on the accuracy of final token. Table 6.1 shows the mean and standard deviation (3 seeds) in final token accuracy on the test set. While RoPE struggles to learn this task and reaches an average accuracy of just 11%, our method PoPE solves the task almost perfectly, reaching an average accuracy of nearly 95%. This result highlights the difficulty that RoPE has in teasing apart 'what' and 'where' information, i.e., identifying the position of certain content and determining the content at a certain relative position. In contrast, PoPE efficiently learns a generalizable solution for the task, presumably because 'what' and 'where' information can be disentangled in key-query matching.

Pos. Enc.	Indirect Idx.
RoPE	11.16 ± 2.45
PoPE	$\mathbf{94.82 \pm 2.91}$

Table 6.1: Accuracy on the test split for Indirect Indexing task.

Next, we test our method on sequence modeling in the domains of music and genomic data. Like human language, both domains have a hierarchical organization and are rule governed systems. In contrast to human language, structural rules can be quite strict and precise positional information is critical. Musical pieces contain hierarchical repeating structures (chords, phrases, motifs), where relative pitch and timing changes are more predictive than their absolute values [Huang et al., 2019]. Huang et al. [2019] also note that the characteristic grammar in piano gestures relies more on relative intervals. Likewise, genomic sequences contain local patterns that rely on relative position and ordering of elements.

Sequence modeling of symbolic music. We train Transformer models using cross-entropy loss on MIDI-based inputs with a maximum length of 2048 from two popular music datasets, Bach-Chorales (JSB) [Boulanger-Lewandowski et al., 2012] and MAESTRO [Hawthorne et al., 2019]. We closely follow the preprocessing from the seminal Music Transformer [Huang et al., 2019]. Table 6.2 reveals that PoPE achieves a decrease in negative log likelihood (NLL) compared to RoPE on both datasets.

Pos. Enc.	JSB	MAESTRO
RoPE	0.5081	1.501
PoPE	**0.4889**	**1.486**

Table 6.2: Best NLL on the test split for Transformer models with RoPE or PoPE positional encodings on music datasets (JSB and MAESTRO).

Sequence modeling of human genome. We train Transformer models on sequences from the Human Reference Genome dataset [Dalla-Torre et al., 2025] using the standard next-token prediction loss. We follow the preprocessing and tokenization procedure from the recent state-of-the-art Nucleotide Transformer [Dalla-Torre et al., 2025] to obtain sequences with a maximum length of 1000 tokens and vocabulary size of 4107. Our PoPE-based model achieves a significant drop in NLL compared to the baseline RoPE model (Table 6.3).

Pos. Enc.	Human Ref. Genome
RoPE	4.217
PoPE	**4.152**

Table 6.3: Best NLL on the test split for the Human Reference Genome dataset.

Although we find a benefit of incorporating PoPE into Transformers on diverse domains like music and genomic sequences, we chose these domains

specifically because they appear to require the separation of position and content as well as precise positional information. It is much less clear that these properties hold true for human language. In our next set of experiments, we pretrain Transformer models at varying model scales on OpenWebText.

Language modeling on OpenWebText. We test PoPE's efficacy on language modeling by training Transformer models of three sizes on the OpenWebText dataset [Gokaslan and Cohen, 2019]. At respective model sizes, both models use identical architecture and training parameters but differ only in the positional encoding. Across model sizes, PoPE consistently achieves lower perplexities than RoPE (Table 6.4). It is also interesting to note that the performance gap between RoPE and PoPE holds steady or possibly increases with model size. We run ablation experiments by training the 124M sized model with PoPE variants that do not use either the Softplus activation, $\sigma()$, or the learnable bias vector $\boldsymbol{\theta}_b$ and find these components each contribute to the performance gains (refer to Appendix E.5 for more details). Next, we move beyond perplexity to evaluate model efficacy on a standard suite of downstream tasks.

Pos. Enc.	124M	253M	774M
RoPE	21.55	18.88	15.85
PoPE	**21.33**	**18.55**	**15.45**

Table 6.4: Perplexity on the validation split of OpenWebText for Transformer models with RoPE or PoPE positional encodings.

Model size	Pos. Encoding	LAMBADA ↑	Blimp ↑	CBT ↑	HellaSwag ↑	PIQA ↑	ARC-E ↑	Avg. ↑
124M	RoPE	**28.74**	77.55	38.80	**29.55**	**60.28**	37.08	45.33
	PoPE	27.67	**77.75**	**44.43**	29.18	59.58	**38.52**	**46.19**
253M	RoPE	**31.38**	79.12	48.51	31.47	**62.24**	**39.87**	48.76
	PoPE	30.47	**80.01**	**49.39**	**31.82**	61.70	39.32	**48.78**
774M	RoPE	35.95	81.05	52.56	35.39	63.55	42.28	51.80
	PoPE	**36.89**	**82.03**	**53.67**	**35.86**	**63.93**	**42.37**	**52.46**

Table 6.5: Zero-shot performance on downstream tasks using Transformer models pretrained on OpenWebText with RoPE or PoPE positional encoding.

Zero-shot performance on downstream tasks. We evaluate the zero-shot performance of the Transformer models pretrained on OpenWebText (from above) on six downstream tasks namely: LAMBADA Paperno et al. [2016], BLiMP [Warstadt et al., 2020], Children's Book Test (CBT) [Hill et al., 2016],

HellaSwag [Zellers et al., 2019], PIQA [Bisk et al., 2020], and ARC-E [Clark et al., 2018]. Following Gao et al. [2024], we use the de-tokenized version from OpenAI for LAMBADA and evaluate the top-1 accuracy on the last word (which can be multiple tokens; we use greedy decoding). Table 6.5 reports accuracy for three models sizes and for each of the six downstream tasks. The last column of the Table presents the mean accuracy across the tasks. For all three model sizes, PoPE-based Transformers have higher mean accuracy than RoPE-based Transformers.

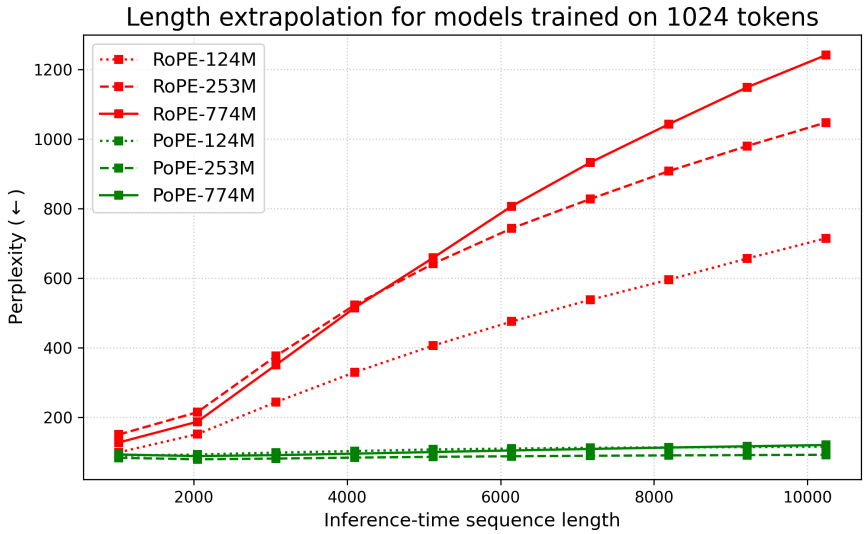


Figure 6.2: Length extrapolation at test-time on PG-19 dataset for different model sizes. We evaluate the models using RoPE (red) and PoPE (green) without any fine-tuning or position interpolation techniques at test-time on longer sequences (multiples of 1024 up to 10240).

Test-time length extrapolation. We measure the ability of PoPE to generalize to longer sequences at test time compared to those presented during training. We examine models pretrained on OpenWebText using a sequence length (context window) of 1024 tokens and assess zero-shot perplexity on much longer sequences (up to 10240 tokens) from the test split of the PG-19 dataset [Rae et al., 2020]. PoPE shows strong out-of-the-box length extrapolation capabilities without any fine-tuning or position interpolation techniques at test time on 10x longer sequences (Figure 6.2). In stark contrast, RoPE's performance significantly degrades on longer sequences at test time. It is also interesting to note that RoPE's extrapolation performance degrades with model size, PoPE's extrapolation performance remains largely stable. RoPE's failure is

due to its allowance of a what-where interaction: aspects of the key and query representation can dynamically shift the position tuning of a component. These shifts become problematic, especially for the lowest frequency components, become influential only when the context window is expanded.

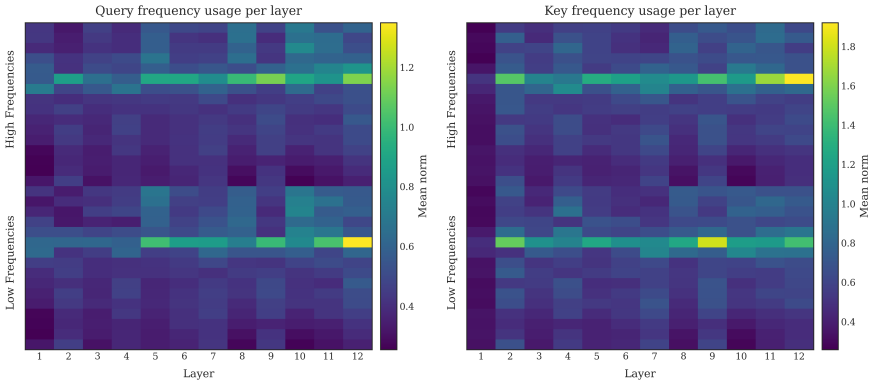


Figure 6.3: 2-norm plotted over 2D RoPE 'chunks' of queries (left) and keys (right) in each layer of the 124M Transformer over different RoPE frequencies. Mean over 10 different Shakespeare sonnets and 12 attention heads at each layer.

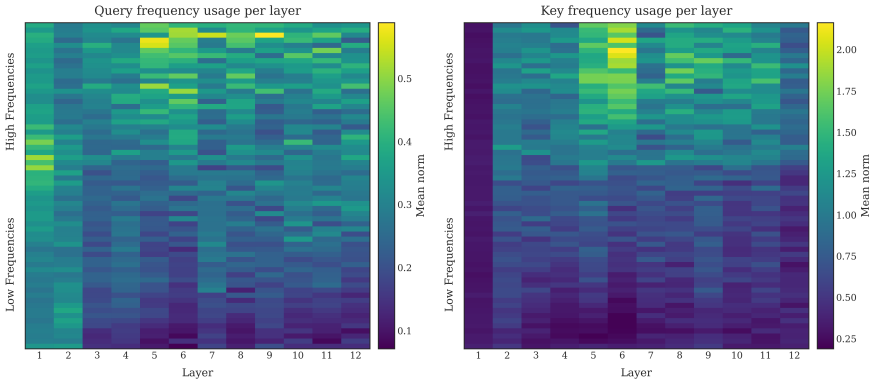


Figure 6.4: Magnitude of each complex-valued features of queries (left) and keys (right) in each layer of the 124M Transformer over different PoPE frequencies. Mean over 10 different Shakespeare sonnets and 12 attention heads at each layer.

Frequency usage analysis. Barbero et al. [2025] analyze RoPE's use of features at different period lengths by plotting the mean norm over the 2D RoPE components for all layers. They observe that the Gemma 7B model prefers to maintain low norms on the high-frequency channels to minimize interference,

because the contribution of these channels to the attention dot product behaves similar to random noise. Gemma has high norms only for features in a sparse set of low frequencies. We apply their analysis on our 124M Transformer models pretrained on OpenWebText (from Table 6.4). In our 124M Transformer, we similarly find that RoPE produces high norms for only a sparse subset of frequency channels across all layers (Figure 6.3). While there are qualitative differences in visualizations of Gemma 7B from Barbero et al. [2025] and our 124M RoPE baseline model, we note that the pretraining and model scales are vastly different, which might explain the different behaviors they've extracted from training data. In contrast, the PoPE transformer assigns high norms on the high frequency features across all layers except the first (Figure 6.4). PoPE also shows a more distributed usage of features across the full frequency range compared to RoPE. Notice the doubling in the number of frequencies (rows of the heatmap) with PoPE (Figure 6.4) versus RoPE (Figure 6.3).

6.5 Related Work

RoPE and its extensions. The most popular positional encoding used by current LLMs [Touvron et al., 2023; Grattafiori et al., 2024; Liu et al., 2024; Team et al., 2025; Yang et al., 2025] is RoPE [Su et al., 2024]. However, RoPE is known to be unable to generalize to longer sequences than it was trained on. Pretraining on long sequences is expensive; thus, the models are typically trained on relatively short context lengths, and then extended to longer context lengths during post-training. This can be done, for example, by changing the rotation frequency for each position so that the total amount of rotations for each component stays the same at the maximum context length as during pretraining [Chen et al., 2023]. A more performant alternative is YaRN [Peng et al., 2024] that scales higher frequencies less than the lower ones, retaining the ability of the model to recall small relative distances precisely. Ding et al. [2024] shows that this strategy can be further improved by skipping the scaling for early token positions and searching for more optimal scaling actors using a genetic algorithm. Furthermore, they perform iterative scaling followed by tuning phases, which allows them to expand the context window to 2M tokens. Sun et al. [2023] takes a different approach: the authors add a decay factor similar to ALiBi [Press et al., 2022] and block-wise masking to limit the span of the attention during inference. Although this prevents a sudden performance drop in pure RoPE, it does not allow us to recall information from long distances. Wang et al. [2024] proposes rounding all RoPE wavelengths to integers, such that there is

no increasing shift in the rotation angle after each time the given channel wraps around. This further improves the performance of YaRN, but is also effective without length extension techniques.

Alternative positional embeddings. The original Transformer [Vaswani et al., 2017] uses sinusoidal embeddings to encode absolute positions; which are added to the token embeddings only at the input layer. The sinusoidal positional embeddings also generally underperform w.r.t more modern methods for relative positional embeddings [Su et al., 2024] which inject this information at every layer. It has been shown that autoregressive Transformers do not require explicit encoding of positional information to operate [Irie et al., 2019; Irie, 2025]. Such Transformers tend to have better length extrapolation properties than both absolute and relative positional embeddings [Kazemnejad et al., 2023], although this comes at the expense of their in-distribution performance. This is also somewhat reminiscent of fast-weight programmers (FWPs) [Schmidhuber, 1992] which are now known as unnormalized linear Transformers that also used no explicit positional encodings. Shaw et al. [2018] introduced a relative positional embedding which uses a separate set of keys that are selected based on the distance of the key and query. Music Transformer [Huang et al., 2019] proposed a more efficient implementation of this relative positional embedding. Dai et al. [2019] propose a different variant, where instead of learning a separate key for each offset, they use a learned projection of sinusoidal "offset encodings", which are inspired by the absolute positional encodings. This approach generalizes better to longer lengths in practice. Additionally, the authors train sequentially within documents and allow attention to the previous batch, enhancing the long-range capabilities of the models further. Wang et al. [2020] generalized this idea of "offset encodings" by using complex-valued embeddings of inputs to encode information about the content, global position and their order relationships within the sequence. T5 [Raffel et al., 2020] takes a different approach: their method groups token pairs in log-sized buckets based on their distance, and it adds a learned per-bucket bias to the attention logit, allowing it to decrease the importance of far-away context. ALiBi Press et al. [2022] takes inspiration from this, adding learned scores to the attention matrix that decay with relative distance. Geometric [Csordás et al., 2022] or stick-breaking attention [Tan et al., 2025] take a radically different approach: it replaces the softmax activation function with a stick-breaking process that gives priority to good matches that are nearby without explicitly encoding positions or offsets and claim to have superior length generalization.

6.6 Conclusion

We propose a new relative positional encoding scheme called PoPE which decouples the matches based on content and position while computing the attention scores. In contrast, RoPE confounds the 'what' and 'where' information in the keys and queries which leads to interference and impedes learning as we highlight on a diagnostic task we introduce, called Indirect Indexing, how Transformers using RoPE struggle to solve whereas those that use PoPE learn this task easily. Further, on autoregressive sequence modeling across music, genomic and natural language we show how Transformer models using PoPE as the positional encoding scheme outperform baselines using RoPE w.r.t training loss (perplexity) and downstream task performance. On language modeling, these gains continue to persist (and even mildly increase) when scaling model parameters from 124 to 774 million. Importantly, PoPE shows strong length extrapolation capabilities without the need for any fine-tuning or position interpolation in contrast to RoPE whose performance is known to degrade significantly on longer sequences at test-time in a zero-shot manner without any fine-tuning or interpolation methods.

Chapter 7

Outlook

While large NNs trained on internet-scale data (Foundation Models) have made remarkable progress in their visual capabilities, they continue to struggle with tasks that require a fine-grained understanding of the visual scene such as counting, inferring relations between objects, inferring causes underlying collisions, handling novel object-attribute combinations, applying logical operators to primitives etc. We view these failures as stemming from an inability to flexibly and dynamically combine information in their distributed representations in a context-dependent manner, i.e. address the *binding problem*. We contrast these failures of current state-of-the-art systems for machine perception with the efficiency and generality of human perception. Infants learn to partition their sensory stimulus into a set of meaningful discrete entities (objects) and relations, infer underlying causes of observations, simulate plausible futures etc. with greater generality and efficiency than their artificial counterparts. This capacity to learn increasingly complex abstractions helps us reason about observable phenomena in terms of their underlying causes and serves as a powerful tool for general-purpose problem solving. This dissertation seeks to emulate certain aspects of human perception such as perceptual grouping and visual saliency using connectionist models.

Concretely, we have contributed several methodological advances that improve object-centric learning using slot-based and synchrony-based models. Chapter 2 shows an approach to discover object-parts (keypoints) that correspond to salient image regions based on the principle of *local spatial predictability* and alleviates certain systematic errors in prior methods for keypoint discovery. Chapter 3

extends perceptual grouping with slot-based models from the spatial to the temporal domain (state-action sequences) by drawing an analogy between the spatial grouping of pixels as visual objects and temporal grouping actions as sub-routines. We developed new decoding and masking algorithms that enables each slot to model a contiguous set of input elements. Our method outperforms previous state-of-the-art models for discovering modular "chunks" in sequences in terms of segmentation accuracy and training time. Chapter 4 presents improvements to the architecture and training procedure leading to improved object storage capacity and grouping performance on in state-of-the-art synchrony-based models. Chapter 5 adopts a first-principles approach to design a new architecture with complex-valued weights and recurrence that greatly simplifies the inductive biases and training of synchrony-based models for object-centric phase synchronization. This new method outperforms prior synchrony-based models for object discovery while being faster to train and using far fewer model parameters. Chapter 6 introduces a powerful generalization of the popular RoPE used by large-scale Transformer models for general-purpose sequence modeling. Interpreting the RoPE computations in polar form complex algebra, informs the design of our scheme, PoPE, which decouples the matches based on the 'what' and the 'where' in contrast to RoPE which entangles them when computing attention scores leading to interference in learning. Our new method PoPE achieves gains in sequence modeling across natural language, music and genomic data while also offering strong length extrapolation capabilities out-of-the-box compared to RoPE.

While this dissertation has taken strides towards improving object-centric learning in NNs using either slot-based or synchrony-based models, the empirical successes shown have been limited to synthetic data of far lesser complexity than sensory inputs from the real world. The empirical results in this dissertation provide proof-of-concept and we do not expect these models to work off-the-shelf on highly complex real-world datasets. Modeling real-world perceptual data requires handling complex relationships between the constituent elements at varying levels of granularity—the sub-parts, parts and objects. This aspect gets at the fundamental tension that exists between the opposing needs for structure and expressiveness in the representational format used by NNs.

Slot-based models partition the encoder's output representation into a set of discrete, independent and equal-sized modules (*slots*) but this in-turn poses difficulties for them to learn meaningful partitions from naturalistic images.

This is because objects in naturalistic images are often of varying complexity and are related to each another in complex ways that go beyond a notion of independence. Empirical successes of either slot-based or synchrony-based models for learning object-centric representations of naturalistic data has remained limited with a few notable exceptions [Singh et al., 2022c; Seitzer et al., 2023; Zadaianchuk et al., 2023; van Steenkiste et al., 2024] with moderate levels of success and largely remains an open research problem. However, some of these recent works use Foundation models as components, e.g. DINO [Caron et al., 2021] to extract as patch-level features, which are then post-processed to recover object-centric representations. Instead, it would be far more desirable if we could bias Vision Transformers (ViTs) [Dosovitskiy et al., 2021] to learn object-centric representations during the pretraining process using either some set of minimal tweaks such as modifications to the attention mechanism or pretraining objective, adding a bottleneck on the number of tokens used in latent layers to encourage greater compression of inputs etc. Ultimately, we wish to add the minimal set of biases to the architecture or training process that allows NNs to discover, represent and process information in terms of symbol-like entities by exploiting the principles of modularity and compositionality.

Slot-based and synchrony-based models lie at the opposing ends of the spectrum of representational modularity. At one end, slots can be treated as "hard" modules that allow for minimal interference between one other. While at the other end, synchrony-based models which use complex-valued activations whose phases store binding information allow for varying degrees of interference via the constructive or destructive interference and therefore act like a "soft" module. Slots divide the encoder's output representation into equal-sized independent partitions which can be quite limiting as a representational format as objects in the real world are often of varying complexities and share complex relations with one another at different granularity. Complex-valued activations used by synchrony-based models can in-principle address these conceptual limitations of slots.

Synchrony-based models use complex-valued activations at all layers, which allows the phase components of the complex-valued activations to maintain this separation of information (modularity) at several levels of granularity in the layers of the convolutional encoder. Further, angular distance in phase space in-principle can serve as a measure of relational similarity between components. They also allow for maintaining partitions of different sizes by clustering the phase space into without the need for extensive retraining. But,

synchrony-based models have only recently experienced a revival due to the contributions made in this dissertation and the efforts of Löwe et al. [2022, 2023] and research on them is still in its infancy. Significant work remains to resolve the limitations that prevent synchrony-based models from fulfilling their full potential. Slot-based models have so far received much greater attention from the research community. The space of ideas for the architecture and training process of slot-based models has been explored at greater depth, leading to more mature design and better performance on standard multi-object datasets.

We highlight three promising directions to explore going forward for advancing object-centric learning and complex-valued NNs:

Role of relations and tasks. Objects in the real world are often related to one another in complex ways. Therefore, any effort to group them in meaningful parts must take into account these relations that connect them. A significant limiting factor with the type of modularity enforced by slots in current models, is that the grouping process assumes some notion of 'independence' among the slots to infer its contents; which does not reflect the underlying structure often found in the real world. It can be argued that objects are meaningful only in relation to other objects, they carry little information in total isolation. Such a relational approach to concept learning assumes that the meaning of individual modules can be deduced from the nature of their interactions with other modules.

Drawing on these intuitions, one way to facilitate object-centric models to learn meaningful parts from real-world sensory data is to ensure that the grouping mechanism simultaneously accounts for both contents of individual objects as well as their relational structure. This raises the question, what is a suitable medium to express relations between objects? Graphs and natural language offer two powerful mediums to express relational quantities in real-world data. For example, images on the internet contain captions in natural language or metadata that can be used to generate relational triples between objects or parse trees computed from natural language captions of scenes that capture object-level relational information. Then, we need to design ways to transform such relational data into a suitable format to condition the grouping process. Orthogonally, tasks provide feedback in the real world to measure the usefulness of several candidate groupings of the sensory stream in a goal-directed agent. For example, imagine a robot that assists with household chores in our home. It will encounter various objects in its visual stream like doors, walls, appliances,

utensils, furniture, people, pets etc. The granularity and types of useful objects for such a house robot is dependent on the current task it needs to solve and the available set of actions it can apply on 'parts' of its environment to achieve the same. Therefore, it is useful to learn modular factors alongside tasks that require agency as opposed to just passive observation.

Unifying slots and synchrony. While this dissertation has developed improvements to slots and synchrony-based models for binding, it has not explored ways to unify them in a hybrid system that exploits their complementary benefits. Slots are better at minimizing interference from each other, for example in *SlotAttention*, the softmax activation applied across slots ensures that most of the information from a given input token is routed to one slot and minimizes interference across them. Therefore slots are a kind of "hard" module and exist at one end of the spectrum. Further, all slots have equal capacity and tend to learn chunks of similar complexity. Also, it has been very challenging to train slot-based models with multiple slot levels to represent hierarchical structure. Complex-valued activations at the other end of the spectrum allow for a continuous degree of interference depending on the phase distribution for different inputs. Since we discretize the phase maps at test-time by clustering we can recover different levels of granularity in groupings from the same phase map including modeling hierarchical structure.

Using "augmented" neurons in NNs. The idea of using complex-valued neurons as the basic processing unit in NNs makes the broader case for augmented information to shape information processing at every layer of a NN. But, what can these augmented signals look like? Looking at biological brains, information is encoded and processed not only using spike amplitudes but also from the firing dynamics, so can firing dynamics be similarly useful for our artificial NNs as well? This dissertation has shown two different ways (binding and relative positional encoding) in which complex-valued activations offer an elegant solution that yields computational advantages for representation learning. First, we show that in synchrony-based models the magnitude-component stores feature information while the phase-component stores binding information and can be trained to achieve object-centric synchronization of phases. Second, in Transformer models we show how by interpreting the mechanisms by which RoPE encodes relative positional information, as complex-valued activations in polar form offers a "natural" separation of the content and position-based matches in the attention layer. The

decoupling of these two pieces of information is the key insight behind our novel relative positional encoding scheme called PoPE.

Going back to our example of biological brains with spiking neurons and the role of firing dynamics, the design of information processing by artificial neurons entirely abstracts away such information. It could be worthwhile to revisit these fundamental design choices and ask if the basic processing units can be similarly "augmented". However, much work remains to be done that enables us to stably train large-scale models made of complex-valued neurons including design of new normalization layers, weight initializations, activation functions and other such engineering innovations that paved the way for their real-valued counterparts to be scaled up.

Progress along these directions could pave the way for NNs that perceive the world around them in a more human-like form.

Appendix A

Appendix on Keypoint Discovery using Local Spatial Predictability

A.1 Datasets

To obtain Atari game frames, we use rollouts of various pre-trained agents in the Atari Model Zoo [Such et al., 2019]. We split the aggregated set of game frames for each of the chosen environments into separate train, validation and test sets of 85,000, 5000 and 5000 samples respectively. For "noisy" Atari, we start with the regular Atari dataset and superimpose colored bars (either horizontal, vertical or both) centered at random x-y co-ordinates. This is done on-the-fly during training/evaluation to generate the required Noisy Atari samples.

A.2 Architecture and Training Details

PermaKey We train our method using the Adam optimizer [Kingma and Ba, 2015a] with a initial learning rate of 0.0002 and decay rate of 0.85 every 10000 steps. We use a batch size of 32 and in all cases train for 100 epochs, using use early stopping of 10 epochs on the validation set to prevent overfitting. All modules (i.e. the VAE, the prediction network, and *PointNet*) are trained in parallel using their respective losses. We do not backpropagate gradients between modules to improve stability. The same hyperparameters are used across all environments except for the number of keypoints.

Variational Autoencoder We use a convolutional neural network with 4 `Conv-BatchNorm-ReLU` layers for the encoder network of the VAE. The encoder network uses kernel sizes $[4, 3, 3, 3]$, filters $[32, 64, 64, 128]$ and strides $[1, 2, 2, 1]$. The architecture of the decoder is the transpose of that of the encoder with $2\times$ bi-linear upsampling used to undo the striding. Preliminary experiments were used to determine the kernel size of the convolutional layers in the encoder. There it was observed that a kernel size of 4 on the first layer for 84×84 images achieves the best results and that larger kernel sizes and strides (≥ 2) reduce the spatial resolution of the feature maps and lead to coarser keypoint localization.

Prediction Network We use a separate 3-layer MLP with hidden layer sizes $[8 \times P \times P \times C, 512, 256, P \times P \times C]$ where P denotes height and width of activation patch and C the number of channels of the activation map of encoder CNN. We use linear output activations for the prediction network and use a separate network for each of the selected layers of the VAE encoder that perform a separate local spatial prediction task. We use encoder layers $[0, 1]$ and $P = 2$ for the local spatial prediction task.

PointNet For the *PointNet* network we use the same encoder architecture as for the VAE, but add a final 1×1 regressor to K feature-maps corresponding to K keypoints [Jakab et al., 2018] with a standard deviation of 0.1 for the Gaussian masks. We resize predictability maps to 84×84 and concatenate them channel-wise before feeding it to the PointNet.

Transporter Re-implementation We re-implemented the *Transporter* model [Kulkarni et al., 2019] for our baseline method. We used an encoder network of 4 *Conv-BatchNorm-ReLU* layers. The feature extractor network Φ uses kernel sizes $[3, 3, 3, 3]$, filters $[16, 16, 32, 32]$ and strides $[1, 1, 2, 1]$. The *PointNet* Ψ uses the same architecture but includes a final 1×1 regressor to K feature-maps corresponding to K keypoints. 2D co-ordinates are computed from these K maps as described in Jakab et al. [2018]. The *RefineNet* uses the transpose of Φ with $2\times$ bi-linear upsampling to undo striding. We used the Adam optimizer with a learning rate of 0.0002 with a decay rate of 0.9 every 30000 steps and batch size of 64. We trained for a 100 epochs with an early-stopping parameter of 10 to prevent over-fitting.

The same hyperparameters are used across all environments except for the number of keypoints.

RL Agent and Training The convolutional keypoint encoder in the *Transporter* and PKey+CNN models consists of 4 `Conv-BatchNorm-ReLU` layers with kernel sizes $[3, 3, 3, 3]$, strides $[1, 1, 2, 1]$ and filters $[128, 128, 128, 128]$. The final convolution layer activation is flattened and passed through a *Dense* layer with 128 units and ReLU activation. The convolutional keypoint encoder receives the keypoint features obtained in the following manner: let $\Phi(x_t)$ a $[H, W, C]$ tensor be the convolutional features and $\mathcal{H}(x_t)$ a $[H, W, K]$ tensor be the keypoint masks for an input image x_t. Keypoint features are obtained by multiplying keypoint masks with convolutional features and superposing masked features of all the K keypoints.

The MLP architecture used as the building block throughout the graph neural network consists of a 2-layers, each having 64 ReLU units. We use a similar MLP (but having linear output units) as the positional encoder network to produce learned positional embeddings given keypoint centers. Output node values from the *Decode* block of the GNN are concatenated and fed into a 2-layer MLP, with 128 ReLU units each before being input to the agent.

For training the RL agents, we use Polyak weight averaging scheme at every step (constant = 0.005) to update the target Q-network. We use Adam optimizer [Kingma and Ba, 2015a] with a learning rate of 0.0002 and clip gradients to maximum norm of 10. For ϵ-greedy exploration we linearly anneal ϵ from 1 to 0.1 over the entire 100,000 steps (or 400,000 frames).

During training we checkpoint policies by running 10 episodes on a separate validation environment initialized with a seed different from the one used for training. We checkpoint policies based on their mean scores on this validation environment. For final evaluation (reporting test scores), we take the best checkpointed models from 3 different runs and run 10 evaluation episodes for each of the 3 policies on 10 previously unseen test environment seeds. We report the mean (and std. dev.) for the 3 different policies using the median evaluation run over the 10 test seeds. We emphasize that different environment seeds are used for training, validation (checkpointing), and testing. Further we do not add any noise in our policy used for evaluation episodes.

A.3 Additional Visualizations

Number of Keypoints On the *Frostbite* environment we modulate the number of keypoints produced by the *PointNet* while keeping all other hyperparameters same as described above in Appendix A.2 (visualization shown in Figure A.1).

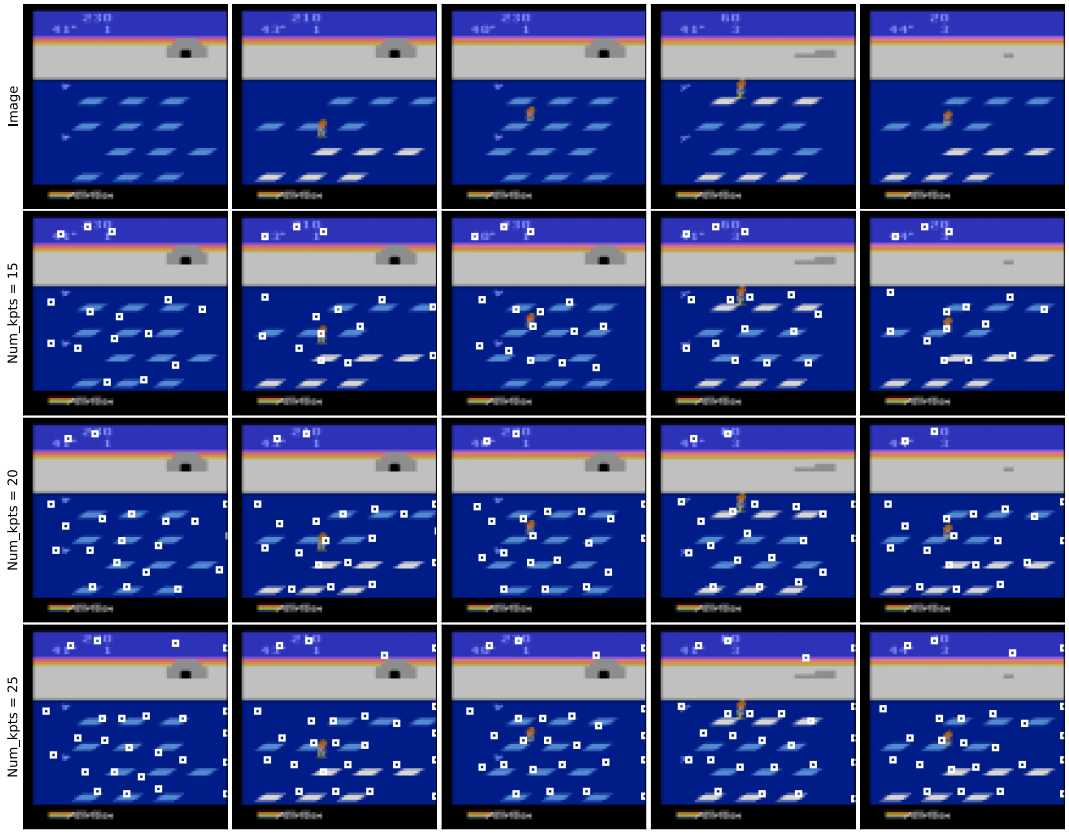


Figure A.1: Image (row 1), keypoints (15, 20 and 25) overlaid on image (rows 2, 3 and 4). With 15 keypoints (row 2) we can see that our model prioritizes the keypoints to capture only the most salient object parts in the scene i.e. ice floats, player, scoreboard etc. As we increase the number of keypoints available it places the excess keypoints on the right border, after having captured all the highly salient object parts i.e. ice floats, player, birds etc.

Effect of Layer Choice On the *Space Invaders'* environment we modulate the feature layer(s) chosen $= \{(0), (0,1), (2,3), (0,1,2,3)\}$ for the local spatial prediction task while keeping all other hyperparameters same as described in Appendix A.2.

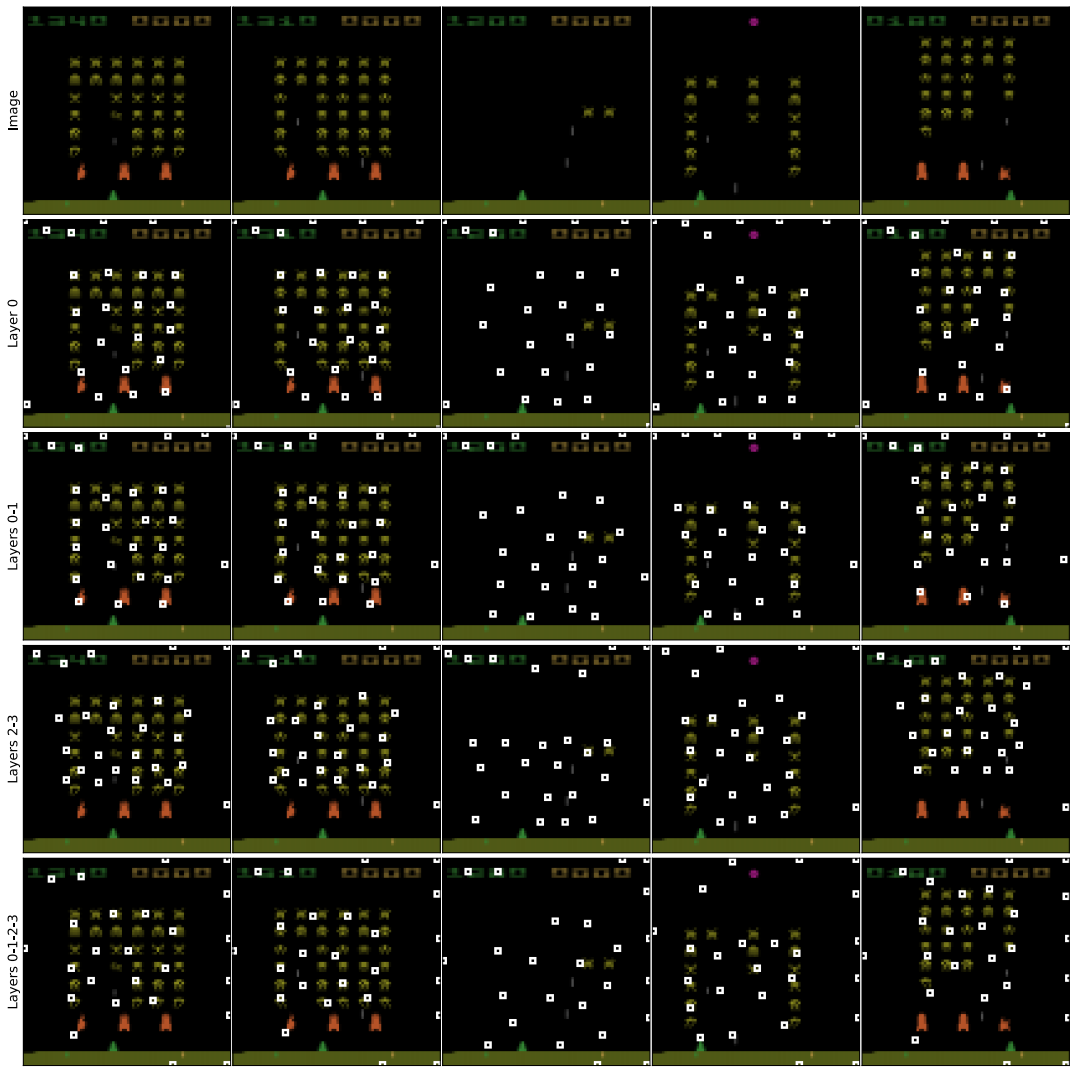


Figure A.2: Image (row 1), keypoints using feature layer(s) $= \{(0), (0,1), (2,3), (0,1,2,3)\}$ (rows 2, 3, 4 and 5 respectively) for local spatial predictability task. We can see that using feature layers 0 and 1 achieves the best keypoint quality. Including higher feature layers (rows 4 and 5) results in more uneven grouping of rows of enemy aliens to keypoints (columns 1, 2 and 6 row 5) and failure to track orange shields (rows 4 and 5).

Seeds We evaluate the stability of our keypoint discovery method over 5 random seeds on *Enduro* and *Battlezone* environments with all hyperparameter settings same as described in Appendix A.2 and the qualitative results are shown in Figure A.3 and Figure A.4.

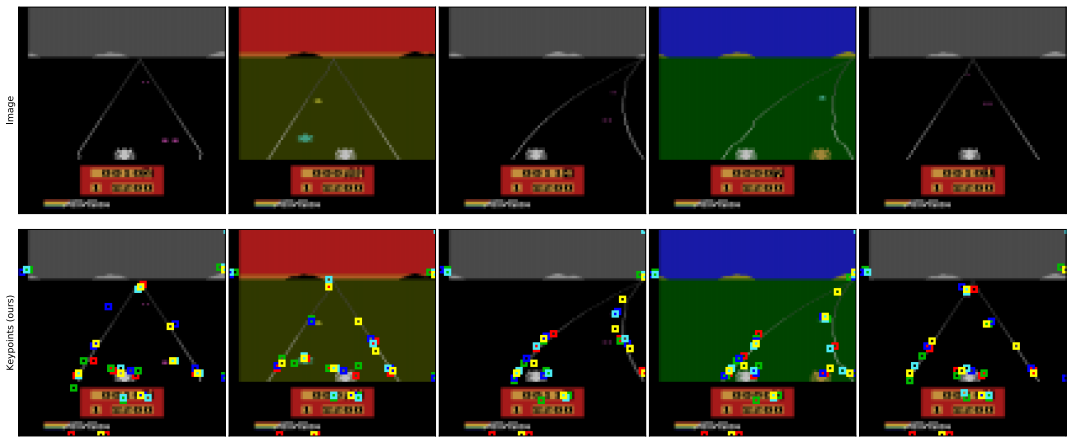


Figure A.3: image (row 1), keypoints colour coded from 5 random seeds overlaid simultaneously on image (row 2). As we can see the learned keypoints remain stable across different runs and capture the salient object parts i.e. player, edges of the road, oncoming traffic, scoreboard etc. each time.

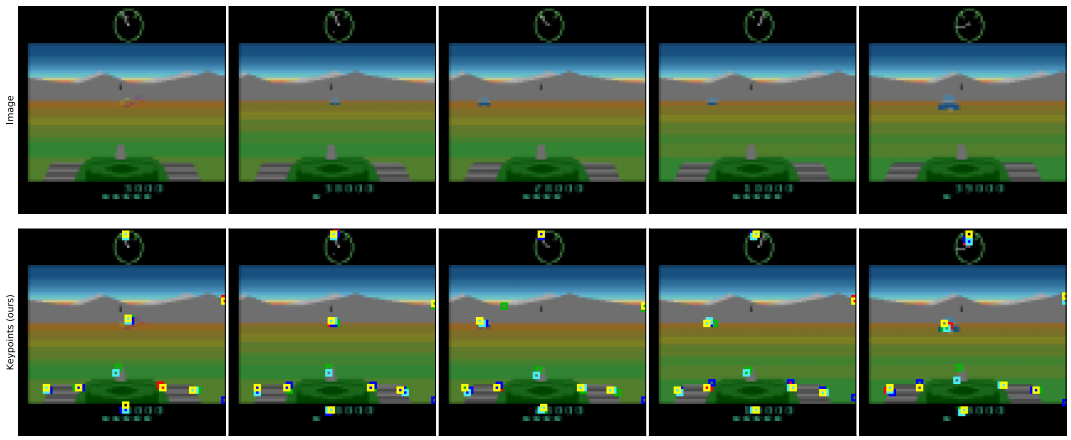


Figure A.4: image (row 1), keypoints colour coded from 5 random seeds overlaid simultaneously on image (row 2). As we can see the learned keypoints remain stable across different runs and capture the salient object parts i.e. player tank parts, enemy tank, orientation dial at the top etc. each time.

Keypoint Masked Regions Figure A.5 shows the keypoint mean (center) as well as Gaussian mask around it. We can see clearly from Figure A.5 that although the keypoint centers might be slightly focused towards the borders of *predictability* their corresponding attention windows ensure good coverage of salient object parts in the scene.

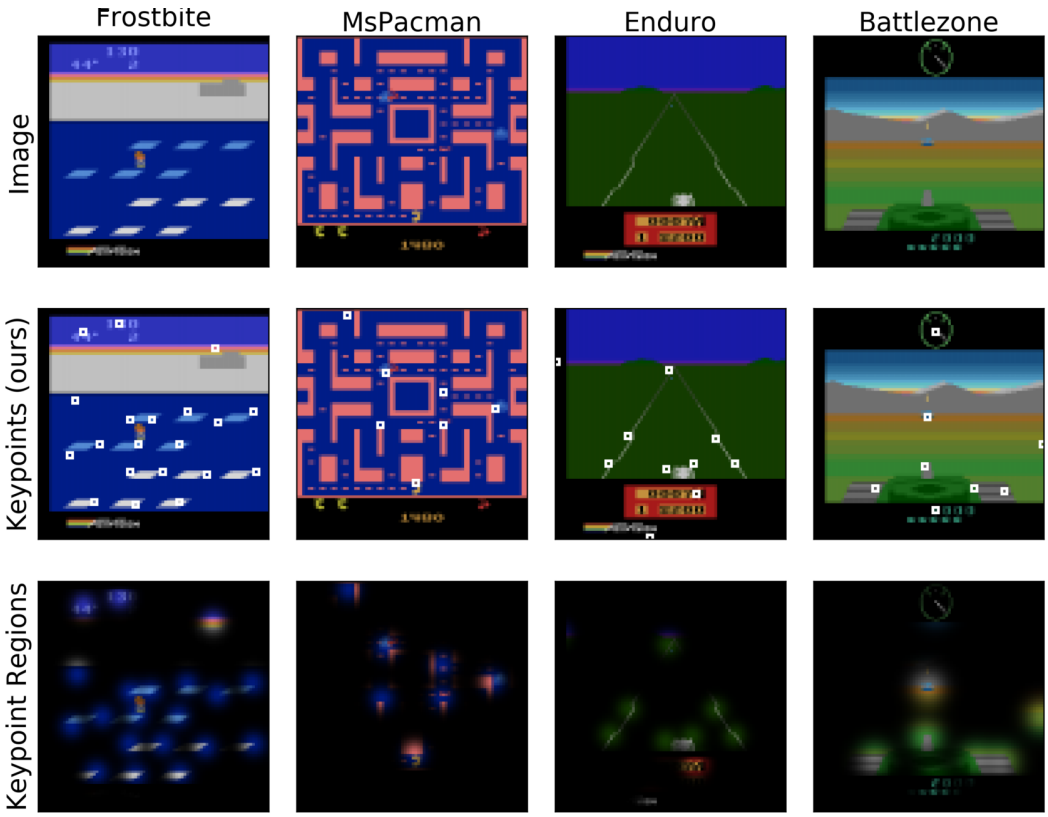


Figure A.5: image (row 1), keypoint centers (row 2) and keypoint masks (row 3).

Appendix B

Appendix on Temporal Abstractions with Slot-based Transformers

B.1 Datasets

Craft The craft environment was initially introduced and re-implemented as a gym environment [Andreas et al., 2017; Lu et al., 2021]. Demonstration data is collected from the rollouts of the heuristic bots [Lu et al., 2021]. We use rollouts from each of the 3 tasks `MakeAxe`, `MakeBed` and `MakeShears` for both the fully observable and partial observable versions of these environments Lu et al. [2021] to report the results in Table 3.1. We use $10\,000$ trajectories for training and another separate 1000 trajectories each for creating validation and test splits. These are used for hyperparameter tuning and reporting evaluation metrics respectively.

Minigrid We use the `DoorKey-8x8`, `UnlockPickup-v0`, `BlockedUnlockPickup` and `KeyCorridor-S4R3` environments from Minigrid [Chevalier-Boisvert et al., 2018] as additional datasets. These environments use procedural generation for each episode ensuring more variability in the action sequences that make up a sub-routine in comparison to Craft. We collect demonstration data by training an A2C agent [Mnih et al., 2016] on this environment until the policy receives an episodic return of ≥ 0.9, which is close to optimal. We use $10\,000$ rollouts collected from this "expert" A2C agent [Mnih et al., 2016] as the training set and 1000 rollouts each for the

validation and test splits used to tune hyperparameters and report evaluation metrics respectively.

Ground-truth segment generation For the Craft environments, the `USE` action serves as a delimiter that marks the end of sub-routines. In the Minigrid environments, the `PICKUP` and `TOGGLE` actions serve as delimiters. For the synthetic versions of the `DoorKey-8x8` and `UnlockPickup` datasets with a single delimiter (used in section 3.3), we simply replace all occurrences of the delimiting action `PICKUP` with `TOGGLE`. We extract the ground-truth boundaries of sub-routines using the indices of these action tokens in the sequences as markers. Ground-truth sub-routine indices for every sequence are in ascending order from left-to-right where delimiting points are specified using the heuristics described above. Table B.1 shows the ground-truth sub-routine types for each of the environments.

MakeAxe	get wood, make at workbench, get iron, make at toolshed
MakeBed	get wood, make at toolshed, get grass, make at workbench
MakeShears	get wood, make at workbench, get iron, make at workbench
DoorKey-8x8	pickup key, open door, go to green goal
UnlockPickup-v0	pickup key, open door, pickup box
BlockedUnlockPickup	pickup key, remove boulder, open door, pickup box
KeyCorridor	pickup key, open door, pickup ball

Table B.1: Examples of ground-truth sub-routines for Craft [Lu et al., 2021] (fully & partially observable) , `DoorKey-8x8`, `UnlockPickup`, `BlockedUnlockPickup` and `KeyCorridor` environments from Minigrid suite [Chevalier-Boisvert et al., 2018].

B.2 Evaluation Metrics

Evaluation metrics used to quantitatively measure the segmentation performance of models are the alignment accuracy of sub-routine prediction and F1 score of boundary index prediction and is consistent with prior work [Lu et al., 2021; Kipf et al., 2019].

Alignment Accuracy The expression to compute alignment accuracy is shown below:

$$\text{Align Acc.} = \frac{1}{L * N} \sum_{n}^{N} \sum_{l}^{L} \mathbb{1}(\hat{s}_l^n = s_l^n)$$

where N is the number of sequences, L is the sequence length (possibly different for each sequence), $\hat{s}_l$ is the predicted sub-routine and s_l is the ground-truth sub-routine action a_l belongs to in the nth sequence.

F1 Score The F1 score on predicted boundary indices is computed as shown below:

$$\text{F1 score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

where precision is computed as,

$$\text{precision} = \frac{\text{\# matches of boundary predictions with ground-truth}}{\text{total \# of boundary predictions}}$$

and recall is computed as,

$$\text{recall} = \frac{\text{\# matches of boundary predictions with ground-truth}}{\text{total \# ground-truth boundaries}}$$

Please refer to the Appendix D.1 in prior work [Kipf et al., 2019] for further details.

B.3 Architecture and Training Details

Input Processing The `Embedding` layer used (Equation (3.1)) maps action tokens a_l to D_{enc} dimensions (denoted by hidden size in Table B.4 and Table B.5). The `Linear` layer used (Equation (3.1)) maps observations $\boldsymbol{o}_l$ to D_{enc} dimensions. These D_{enc} dimensional action and observation features are then concatenated and processed by a 1-layer MLP with D_{enc} units and `ReLU` activation to learn a joint action-observation embedding space.

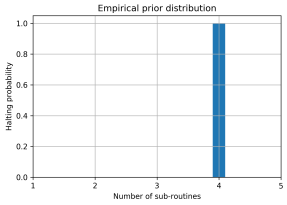
Encoder The linear layers used in self-attention have D_{enc} units in the `Encoder` block. The position-wise feedforward network used in the `Encoder` uses 4 times the number of units as hidden size. We add the standard sinusoidal positional encoding $\boldsymbol{p}_{\text{sin}}$ [Vaswani et al., 2017] to the joint action-observation features $\boldsymbol{z}_{\boldsymbol{ao}}$. Further, we use a linear layer to generate the learned positional encoding $\boldsymbol{p}_{\text{learn}}$ that is added to the outputs $\boldsymbol{h}$ from the `Encoder` module. The final configuration(s) for each of these hyperparameters are shown in Table B.4 and Table B.5 for fixed and variable slot environments respectively. The hyperparameter sweep configurations are shown in Table B.2 and Table B.3 for

fixed and variable slot environments respectively. The overall `Encoder` block uses the same architectural template as the Transformer encoder [Vaswani et al., 2017].

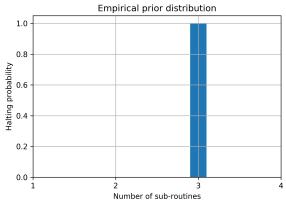
Slot Attention The slot attention module uses linear layers with D_{slots} units to generate keys and values. The recurrent update function is implemented using a GRU [Cho et al., 2014] (see also earlier work [Gers et al., 2000]) with D_{slots} units (denoted by slot size in Table B.4 and Table B.5). The MLP network used for the residual update is implemented using a 2-layer network with D_{slots} units and ReLU and no activation for the hidden layer and output layers respectively. Further, the initial query vector for `slot_k` is sampled from a Gaussian distribution with mean 0 and standard-deviation σ.

Decoder The decoder architecture adapts the spatial broadcast decoder [Watters et al., 2019] in a suitable manner to decode all slots into their corresponding action segments. We broadcast the slot representations along the sequence length L and pass the observations $\boldsymbol{o_l}$ as inputs at each timestep to the `Decoder`. The Decoder module has the exact same architecture as the Transformer encoder with the exception that an additional output linear layer of $A + 1$ (where A is the size of action space) dimensions is used to decode the slot representations into predicted action logits and the end position logits. Segment masks are computed given end position logits as shown in Algorithm 2. Then we composite segment masks and predicted action logits to generate the full predicted action sequence.

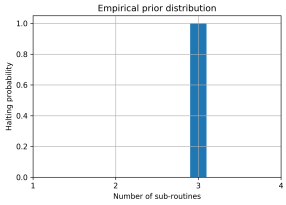
Hyperparameter Tuning We perform a random search of 300 hyperparameter configurations from all possible configurations in the hyperparameter sweeps shown in Table B.2 and Table B.3 on 3 seeds for fixed and variable slot environments respectively. We train our model for a maximum of 100 epochs on all environments with early stopping using the evaluation metrics (average over F1 and Align. acc.) on the validation set. Then, we picked the best performing configuration(s) for all of the environments for our model based on the evaluation metrics (average of F1 and Align. acc.) on the validation set and ran them again using 5 seeds to report final scores (shown in Table 3.1, Table 3.2, Table 3.3). The final configurations used for our model are shown in Table B.4 and Table B.5 for fixed and variable number of sub-routine environments respectively.



(a) Craft.



(b) DoorKey-8x8.



(c) UnlockPickup.

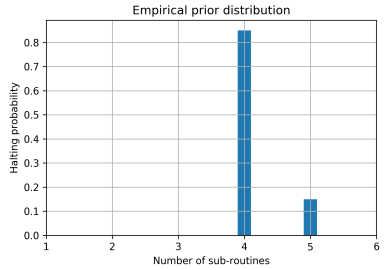
Figure B.1: Empirical piece-wise priors for the Craft (fully and partially observable), Doorkey-8x8 and UnlockPickup environments in Minigrid all of which contains fixed number of sub-routines across trajectories.

Empirical Prior Distribution For the empirical prior distribution used in the PonderNet loss Equation (3.5), we use a categorical distribution over the maximum number of slots K. We compute the categorical probabilities as ratio of counts of the number of trajectories which have $k \in \{1, ..., K\}$ sub-routines and the total number of trajectories in the training dataset. Please refer to Figure B.1 and Figure B.2, which show these categorical priors for various datasets.

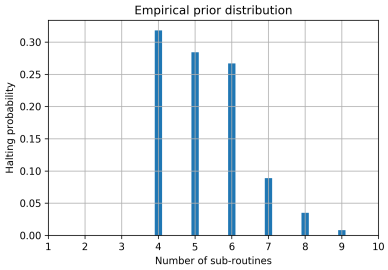
Since we have a discrete distribution for our prior and posteriors, we directly compute the KL-divergence between the halting probability (p_{halt}) and prior distribution (p_e) as shown below:

$$D_{KL}[p_{\text{halt}}||p_{\text{e}}] = \sum_{k'=1}^{K} p_{\text{halt}}(k') \log \left(\frac{p_{\text{halt}}(k')}{p_{\text{e}}(k') + \epsilon} \right) \tag{B.1}$$

where ϵ is a small positive number (here set to $1e-4$) for numerical stability. The maximum number of slots K used for each of the environments can be seen in Figure B.1 and Figure B.2.



(a) BlockedUnlockPickup.



(b) KeyCorridor-S4r3.

Figure B.2: Empirical discrete prior distributions for the BlockedUnlockPickup and KeyCorridor-S4R3 environments in Minigrid both of which contains variable number of sub-routines across trajectories.

SloTTAr	Craft (fully)	Craft (partial)	DoorKey-8x8	UnlockPickup
batch size	[32, 64, 128]	[32, 64, 128]	[32, 64, 128]	[32, 64, 128]
beta	[0.1, 0.5, 1.0]	[0.1, 0.5, 1.0]	[0.1, 0.5, 1.0]	[0.1, 0.5, 1.0]
hidden size	[64, 96, 128]	[64, 96, 128]	[64, 96, 128]	[64, 96, 128]
number of heads	[4, 8, 16]	[4, 8, 16]	[4, 8, 16]	[4, 8, 16]
slot size	[64, 96, 128]	[64, 96, 128]	[64, 96, 128]	[64, 96, 128]
slot std_dev	[0.5, 1.0]	[0.5, 1.0]	[0.5, 1.0]	[0.5, 1.0]
number of iterations	[1, 2]	[1, 2]	[1, 2]	[1, 2]
learning rate	[2.5e-4, 5e-4 1e-3]	[2.5e-4, 5e-4 1e-3]	[2.5e-4, 5e-4 1e-3]	[2.5e-4, 5e-4 1e-3]

Table B.2: Hyperparameter sweep configurations for SloTTAr on for Craft [Lu et al., 2021] (fully & partially observable), `DoorKey-8x8` and `UnlockPickup` environments from Minigrid [Chevalier-Boisvert et al., 2018]. All of these environments have a fixed number of sub-routines across all trajectories.

SloTTAr	BlockedUnlockPickup	KeyCorridor-S4R3
batch size	[32, 64, 128]	[32, 64, 128]
beta	[0.05, 0.1, 0.2]	[0.1, 0.2, 0.5]
hidden size	[64, 96, 128]	[64, 96, 128]
number of heads	[4, 8, 16]	[4, 8, 16]
slot size	[64, 96, 128]	[64, 96, 128]
slot std_dev	[0.5, 1.0]	[0.5, 1.0]
number of iterations	[1, 2]	[1, 2]
learning rate	[2.5e-4, 5e-4 1e-3]	[2.5e-4, 5e-4 1e-3]

Table B.3: Hyperparameter sweep configurations for SloTTAr for `BlockedUnlockPickup` and `KeyCorridor-S4R3` Minigrid environments [Chevalier-Boisvert et al., 2018]. These environments have a variable number of sub-routines across different trajectories.

SloTTAr	Craft (fully)	Craft (partial)	(DoorKey-8x8)	UnlockPickup
batch size	64	64	32	32
beta	0.1	0.5	0.1	1.0
hidden size	128	128	128	128
number of heads	8	16	8	8
number of layers	1	1	1	1
slot size	128	64	128	128
slot std_dev	1.0	1.0	1.0	1.0
number of iterations	1	1	1	1
number of slots	5	5	4	4
number of segments	4	4	3	3
optimizer	Adam	Adam	Adam	Adam
learning rate	0.0005	0.0005	0.0005	0.0005

Table B.4: Final hyperparameter configurations for SloTTAr for Craft [Lu et al., 2021] (fully & partially observable), `Doorkey-8x8`, `UnlockPickup` environments from Minigrid [Chevalier-Boisvert et al., 2018].

SloTTAr	BlockedUnlockPickup	KeyCorridor-S4R3
batch size	64	32
beta	0.05	0.5
hidden size	96	96
number of heads	8	8
number of layers	1	1
slot size	96	96
slot std_dev	1.0	1.0
number of iterations	1	2
number of slots	5	10
number of segments	4-5	4-9
optimizer	Adam	Adam
learning rate	0.0005	0.00025

Table B.5: Final hyperparameter configurations for SloTTAr for `BlockedUnlockPickup` and `KeyCorridor-S4R3` Minigrid environments [Chevalier-Boisvert et al., 2018]. These environments have a variable number of sub-routines across different trajectories.

B.4 Baseline Models and Training Details

We re-implemented CompILE [1] and OMPN [2] by adapting the original authors' implementations available on GitHub. We perform a random search of up to 300 hyperparameter configurations from all possible configurations in the hyperparameter sweeps shown in Table B.6, Table B.7, Table B.10 and Table B.11 for all environments and both baseline models (CompILE and OMPN) with 3 seeds. Then, we picked the best performing configuration(s) based on their evaluation metrics on the validation set from this sweep. We run these best performing configuration(s) on 5 seeds to obtain all results shown in Table 3.1, Table 3.2 and Table 3.3. We checkpoint models during the course of training based on their evaluation metric scores on the validation set. We use the best checkpoints to report the final scores shown in Table 3.1, Table 3.2 and Table 3.3 for all the baseline models.

The final configurations used for the baseline models (CompILE and OMPN) on the 3 environments are shown in Table B.8, Table B.9, Table B.12 and Table B.13 for fixed and variable slot environments.

CompILE	Craft (fully)	Craft (partial)	DoorKey-8x8	UnlockPickup
batch size	[64, 128, 256]	[64, 128, 256]	[32, 64, 128]	[32, 64, 128]
beta_b = beta_z	[0.01, 0.05, 0.1]	[0.01, 0.05, 0.1]	[0.01, 0.05, 0.1]	[0.01, 0.05, 0.1]
hidden size	[64, 128]	[64, 128]	[64, 128]	[64, 128]
latent dist.	['gaussian', 'concrete']	['gaussian', 'concrete']	['gaussian', 'concrete']	['gaussian', 'concrete']
latent size	[64, 128]	[64, 128]	[64, 128]	[64, 128]
learning rate	[5e-5, 2.5e-4 1e-3]	[5e-5, 2.5e-4 1e-3]	[1e-4, 2e-4 5e-4]	[1e-4, 2e-4 5e-4]
prior rate	[2, 3, 4]	[2, 3, 4]	[2, 3, 4]	[2, 3, 4]

Table B.6: Hyperparameter sweep configurations for CompILE for Craft [Lu et al., 2021] (fully & partially observable), `Doorkey-8x8` and `UnlockPickup` environments from Minigrid [Chevalier-Boisvert et al., 2018]. For the full description of all these hyperparameters, we refer the readers to Kipf et al. [2019].

[1] https://github.com/tkipf/compile
[2] https://github.com/Ordered-Memory-RL/ompn_craft

CompILE	BlockedUnlockPickup	KeyCorridor-S4R3
batch size	[32, 64, 128]	[32, 64, 128]
beta_b = beta_z	[0.05, 0.1, 0.2]	[0.01, 0.05, 0.1]
hidden size	[64, 128]	[64, 128]
latent dist.	['gaussian', 'concrete']	['gaussian', 'concrete']
latent size	[64, 128]	[64, 128]
learning rate	[5e-5, 2.5e-4 1e-3]	[5e-5, 2.5e-4 1e-3]
prior rate	[2, 3, 4]	[2, 3, 4]

Table B.7: Hyperparameter sweep configurations for CompILE for `BlockedUnlockPickup` and `KeyCorridor-S4R3` environments from Minigrid [Chevalier-Boisvert et al., 2018].

CompILE	Craft (fully)	Craft (partial)	DoorKey-8x8	UnlockPickup
batch size	128	128	32	64
beta_b	0.01	0.01	0.01	0.05
beta_z	0.01	0.01	0.01	0.05
prior rate	3	3	4	2
hidden size	128	128	128	64
latent dist.	'gaussian'	'gaussian'	'gaussian'	'gaussian'
latent size	128	128	128	64
number of segments	4	4	3	3
optimizer	Adam	Adam	Adam	Adam
learning rate	0.00025	0.00025	0.0005	0.0002

Table B.8: Final hyperparameter configurations for CompILE for Craft [Lu et al., 2021] (fully & partially observable), `DoorKey-8x8` and `UnlockPickup` environments from Minigrid Chevalier-Boisvert et al. [2018]. All of these environments have a fixed number of sub-routines across all trajectories.

CompILE	BlockedUnlockPickup	KeyCorridorS4R3
batch size	128	128
beta_b	0.1	0.1
beta_z	0.1	0.1
prior rate	3	3
hidden size	128	128
latent dist.	'gaussian'	'gaussian'
latent size	128	128
number of segments	4-5	4-9
optimizer	Adam	Adam
learning rate	0.00025	0.00025

Table B.9: Final hyperparameter configurations for CompILE for `BlockedUnlockPickup` and `KeyCorridor-S4R3` environments from Minigrid Chevalier-Boisvert et al. [2018]. All of these environments have a fixed number of sub-routines across all trajectories.

OMPN	Craft (fully)	Craft (partial)	DoorKey-8x8	UnlockPickup
batch size	[64, 128, 256]	[64, 128, 256]	[32, 64, 128]	[32, 64, 128]
hidden size	[64, 128]	[64, 128]	[64, 128]	[64, 128]
learning rate	[1e-4, 2.5e-4, 5e-4]	[1e-4, 2.5e-4, 5e-4]	[1e-4, 2e-4, 5e-4]	[1e-4, 2e-4, 5e-4]
number of slots	[2, 3]	[2, 3]	[2, 3]	[2, 3]
max. gradient norm	[0.5, 1.0, 2.0]	[0.2, 0.5, 1.0]	[0.2, 0.5, 1.0]	[0.5, 1.0, 2.0]

Table B.10: Hyperparameter sweep configurations for OMPN for Craft [Lu et al., 2021] (fully & partially observable) , `Doorkey-8x8` and `UnlockPickup` environments from Minigrid [Chevalier-Boisvert et al., 2018]. All of these environments have a fixed number of sub-routines across all trajectories.

OMPN	BlockedUnlockPickup	KeyCorridor-S4R3
batch size	[32, 64, 128]	[32, 64, 128]
hidden size	[64, 128]	[64, 128]
learning rate	[1e-4, 2e-4, 5e-4]	[1e-4, 2e-4, 5e-4]
number of slots	[2, 3, 4]	[2, 3, 4]
max. gradient norm	[0.2, 0.5, 1.0]	[0.2, 1.0, 2.0]

Table B.11: Hyperparameter sweep configurations for OMPN for `BlockedUnlockPickup` and `Doorkey-8x8` environments from Minigrid [Chevalier-Boisvert et al., 2018]. All of these environments have a variable number of sub-routines across all trajectories.

OMPN	Craft (fully)	Craft (partial)	DoorKey-8x8	UnlockPickup
batch size	128	128	64	64
hidden size	128	128	128	64
number of slots	3	3	3	3
number of segments	4	4	3	3
max. gradient norm	1.0	1.0	1.0	2.0
optimizer	Adam	Adam	Adam	Adam
learning rate	0.00025	0.00025	0.0001	0.0002

Table B.12: Final hyperparameter configurations for OMPN for Craft [Lu et al., 2021] (fully & partially observable), `Doorkey-8x8` and `UnlockPickup` environments from Minigrid [Chevalier-Boisvert et al., 2018]. All of these environments have a fixed number of sub-routines across all trajectories.

OMPN	BlockedUnlockPickup	KeyCorridor-S4R3
batch size	32	32
hidden size	128	128
number of slots	3	3
number of segments	4-5	4-9
max. gradient norm	0.2	2.0
optimizer	Adam	Adam
learning rate	0.0001	0.0001

Table B.13: Final hyperparameter configurations for OMPN `BlockedUnlockPickup` and `KeyCorridor-S4R3` environments from Minigrid [Chevalier-Boisvert et al., 2018]. These environments have a variable number of sub-routines across all trajectories.

B.5 Visualizations

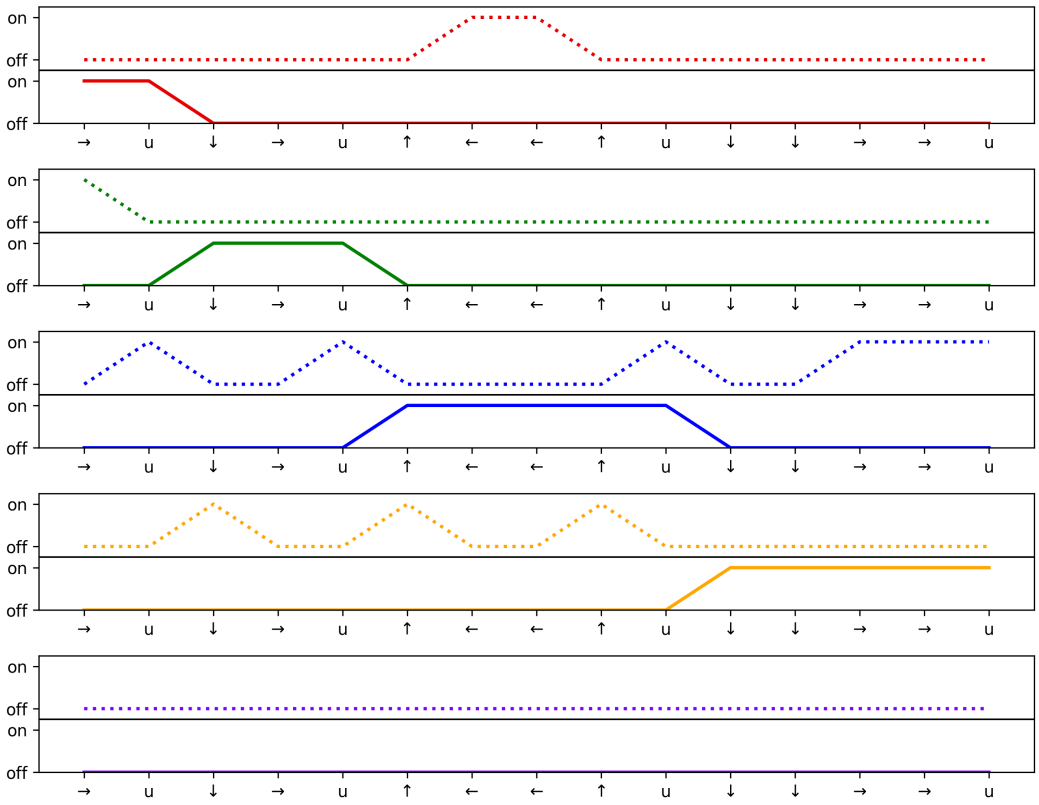


Figure B.3: Within each subplot, the bottom panel shows thresholded alpha mask (in solid lines) and the top panel shows the thresholded slot attention weights (in dotted lines) on the y-axis for a sample trajectory from partially observable Craft environment against actions on the x-axis. Different subplots correspond to different slots (colour coded).

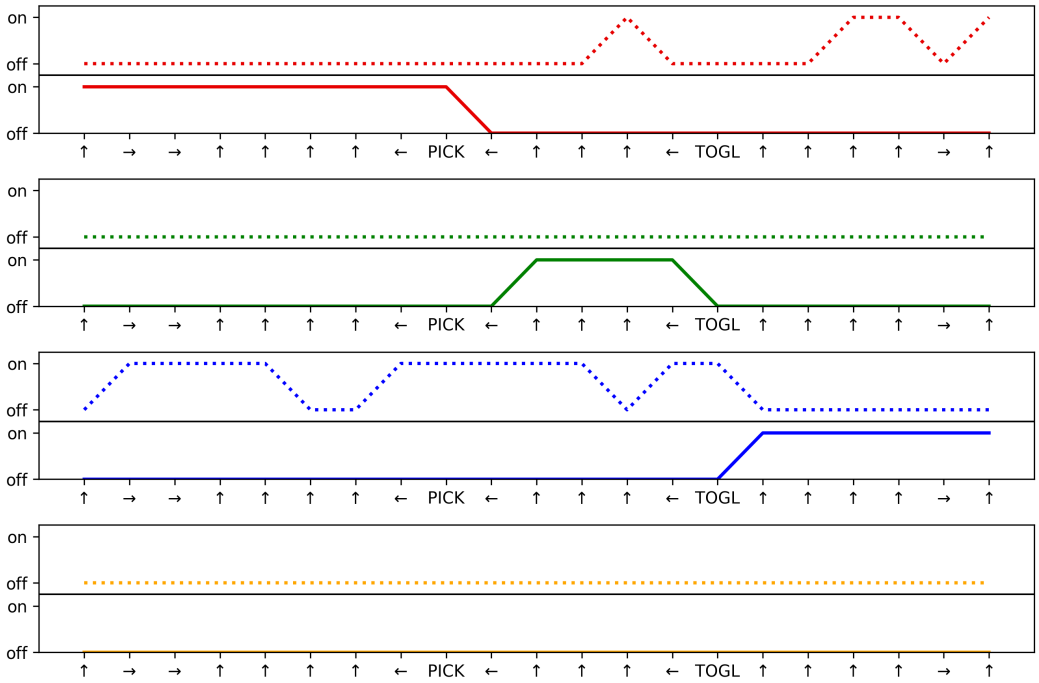


Figure B.4: Within each subplot, the bottom panel shows thresholded alpha mask (in solid lines) and the top panel shows the thresholded slot attention weights (in dotted lines) on the y-axis for a sample trajectory from partially observable `DoorKey-8x8` Minigrid environment against actions on the x-axis. Different subplots correspond to different slots (colour coded).

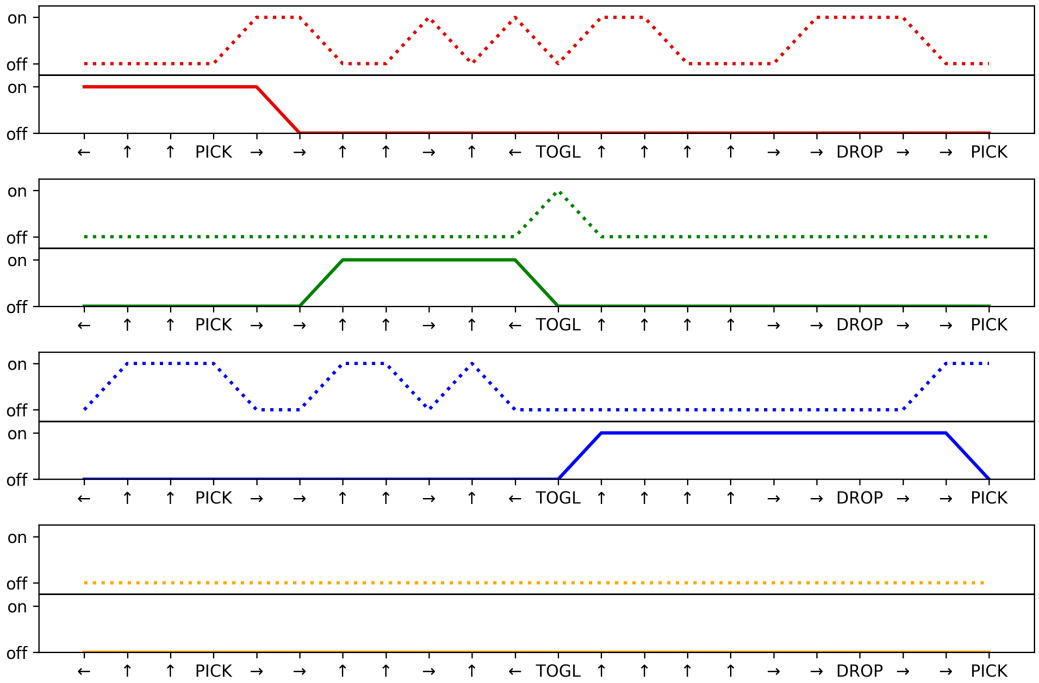


Figure B.5: Within each subplot, the bottom panel shows thresholded alpha mask (in solid lines) and the top panel shows the thresholded slot attention weights (in dotted lines) on the y-axis for a sample trajectory from partially observable `UnlockPickup` Minigrid environment against actions on the x-axis. Different subplots correspond to different slots (colour coded).

B.6 Additional Results

single-delimiter	DoorKey-8x8 (partial)		UnlockPickup-v0 (partial)	
	F1 score	Align. acc.	F1 score	Align. acc.
OMPN	46.26 (13.99)	56.11 (6.23)	62.35 (18.39)	70.83 (10.72)
OMPN-2	53.70 (12.01)	65.55 (8.48)	53.58 (19.57)	59.84 (15.41)

Table B.14: F1 scores and alignment accuracies on the synthetic single delimiter versions of Minigrid environments – `DoorKey-8x8` and `UnlockPickup-v0`. F1 scores are computed with tolerance = 1 consistent with Lu et al. [2021].

Appendix C

Appendix on Contrastive Training of Complex-valued Autoencoders

C.1 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].

C.2 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.

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.

C.3 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.

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.

C.4 Computational Efficiency

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

C.5 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].

C.6 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.

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.

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.

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.

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

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.

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	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 4.4 to all 3 multi-object datasets.

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 4.5 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.7 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.

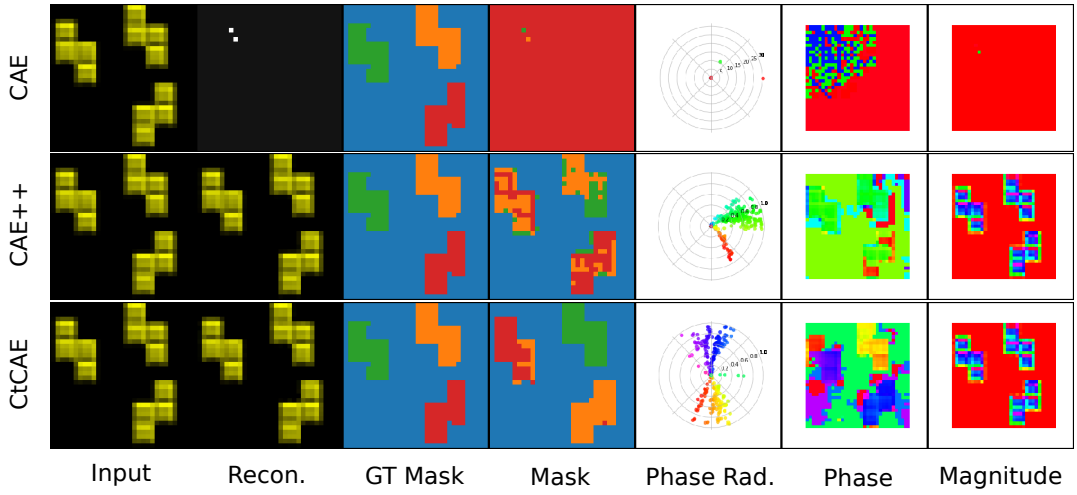


Figure C.1: Visualizations for CAE (row 1), CAE++ (row 2) and CtCAE (row 3) on the `Tetrominoes` dataset example from Figure 4.4. We see that CtCAE achieves near perfect grouping (column 4) and better *separability* in phase space (column 5) compared to our improved CAE++ variant which shows significant 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.

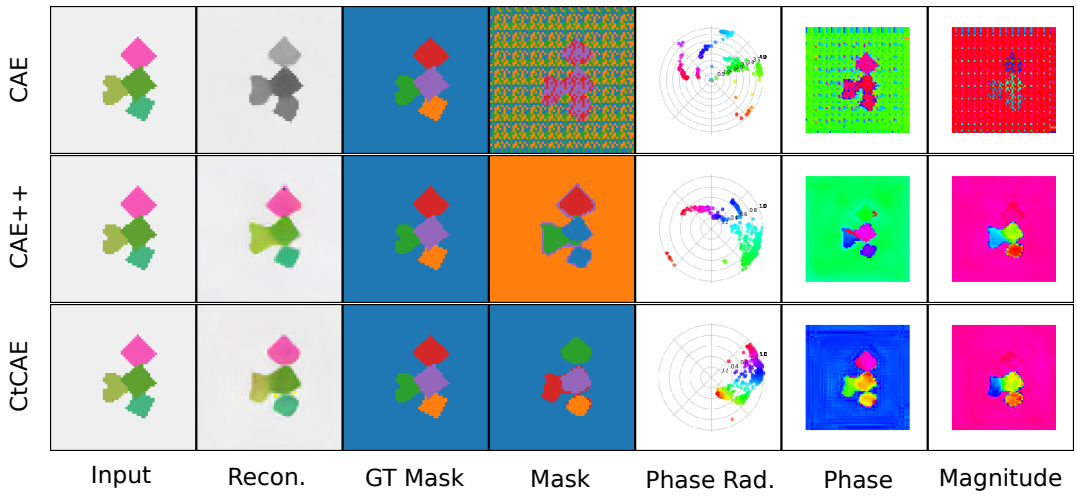


Figure C.2: Visualizations for CAE (row 1), CAE++ (row 2) and CtCAE (row 3) on the `dSprites` dataset example from Figure 4.4. We see that CtCAE successfully separates the 4 scene objects into 4 clusters while CAE++ mistakenly groups 2 scene objects into 1 cluster, i.e., the blue cluster (column 4). Again, the baseline CAE fails to reconstruct (no color) or group the image.

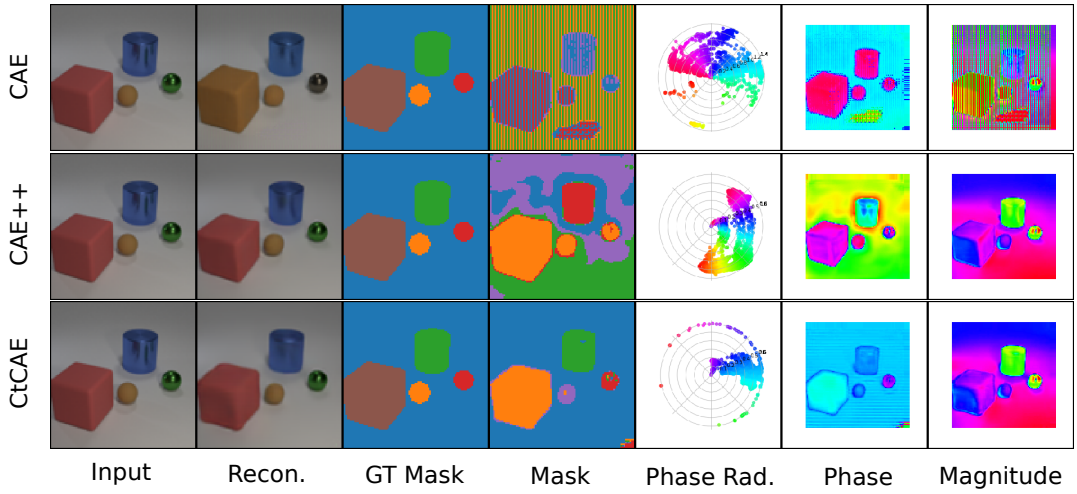


Figure C.3: Visualizations for CAE (row 1), CAE++ (row 2) and CtCAE (row 3) on the CLEVR dataset example from Figure 4.4. We see that the CtCAE in this example is able to separate the 4 scene objects into 4 distinct clusters while our improved CAE++ fails (column 4). This good separation in phase space 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.

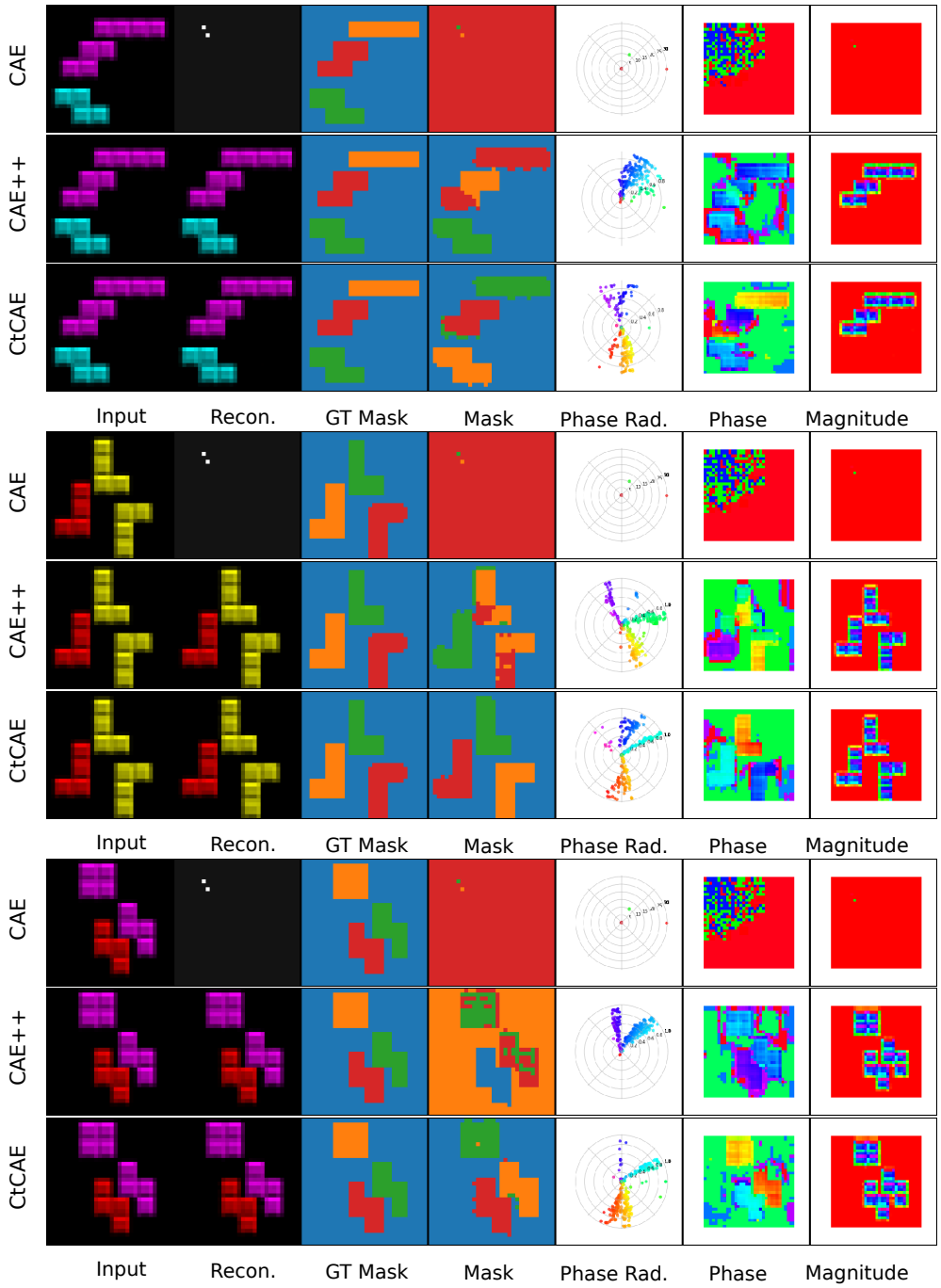


Figure C.4: Visualization on grouping performance for CAE, CAE++ and CtCAE on images containing same colored objects on `Tetrominoes`. CAE struggles to even reconstruct the images, whereas CAE++ often groups objects with the same colour together. CtCAE on the other hand has no problem separating objects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

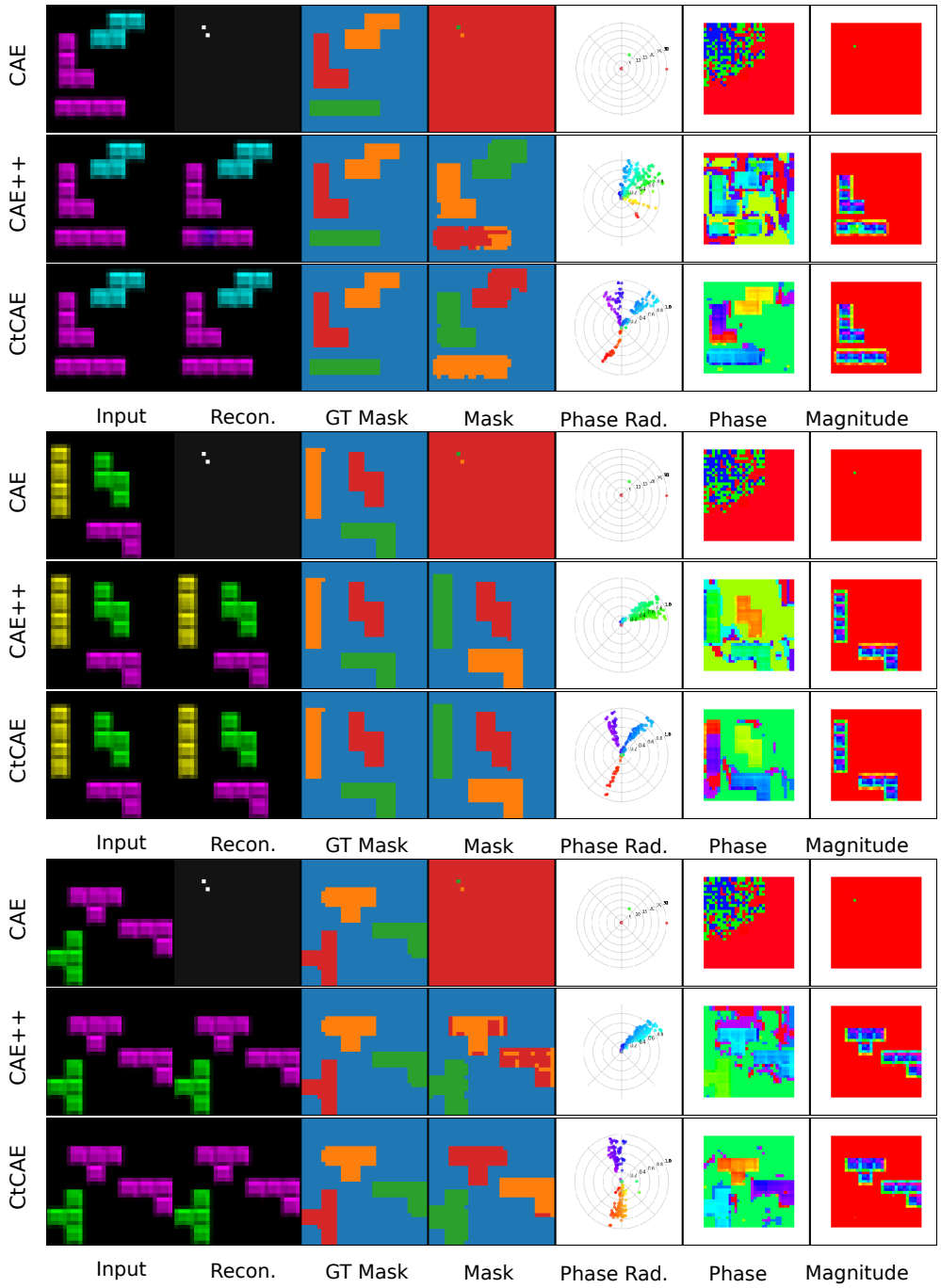


Figure C.5: Visualization on grouping performance for CAE, CAE++ and CtCAE on `Tetrominoes`. CtCAE shows a larger spread in phase values for different clusters (column 5) compared to CAE. We hypothesize that this facilitates CtCAE to perform better grouping in the scenario of multiple object instances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

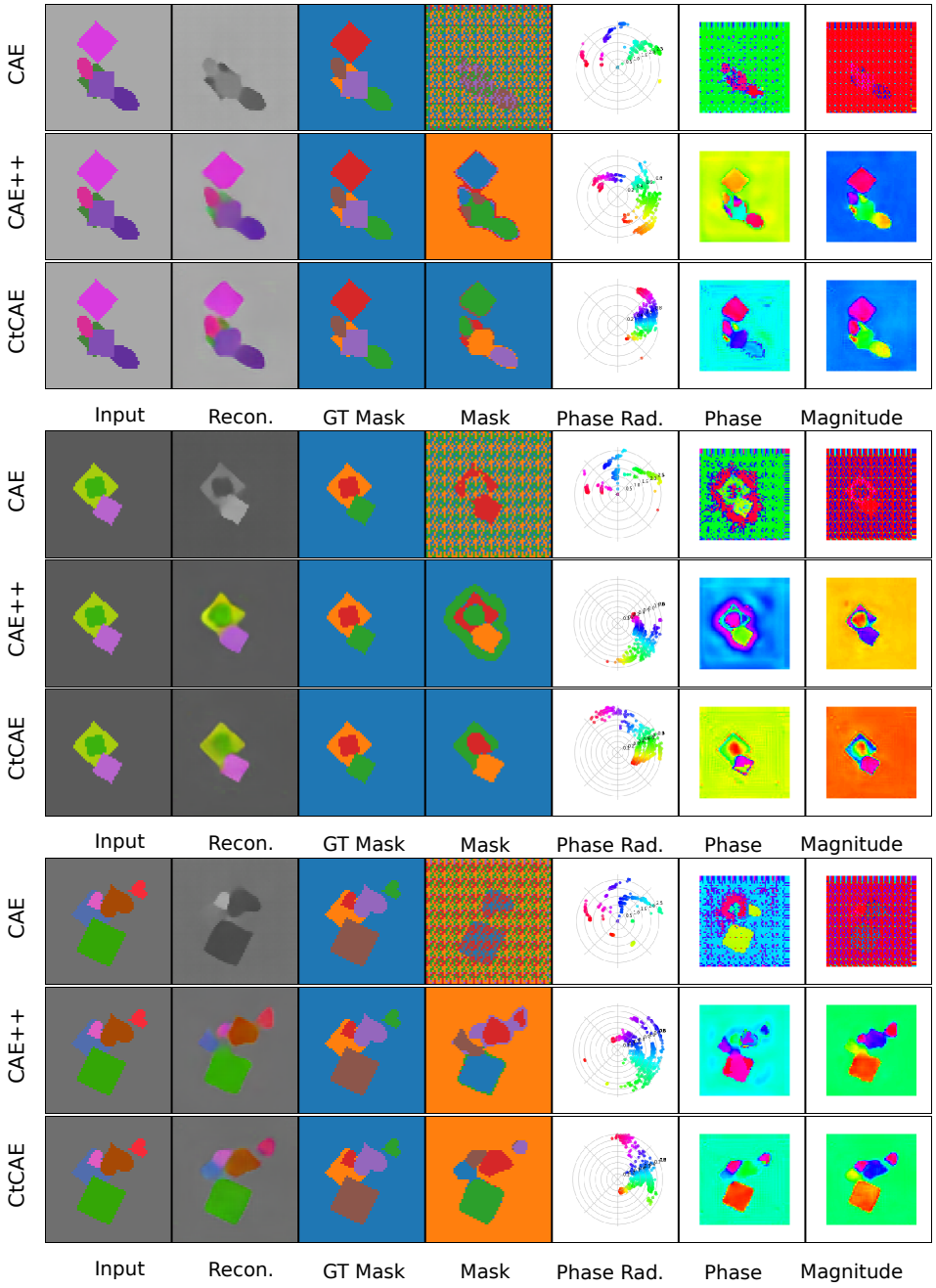


Figure C.6: Visualization on grouping performance for CAE, CAE++ and CtCAE on images with similar colored objects on `dSprites`. CAE has poor grouping capability, whereas CAE++ often groups objects with the same colour together. Eg., purple ellipse and square in the top panel or red and brown hearts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

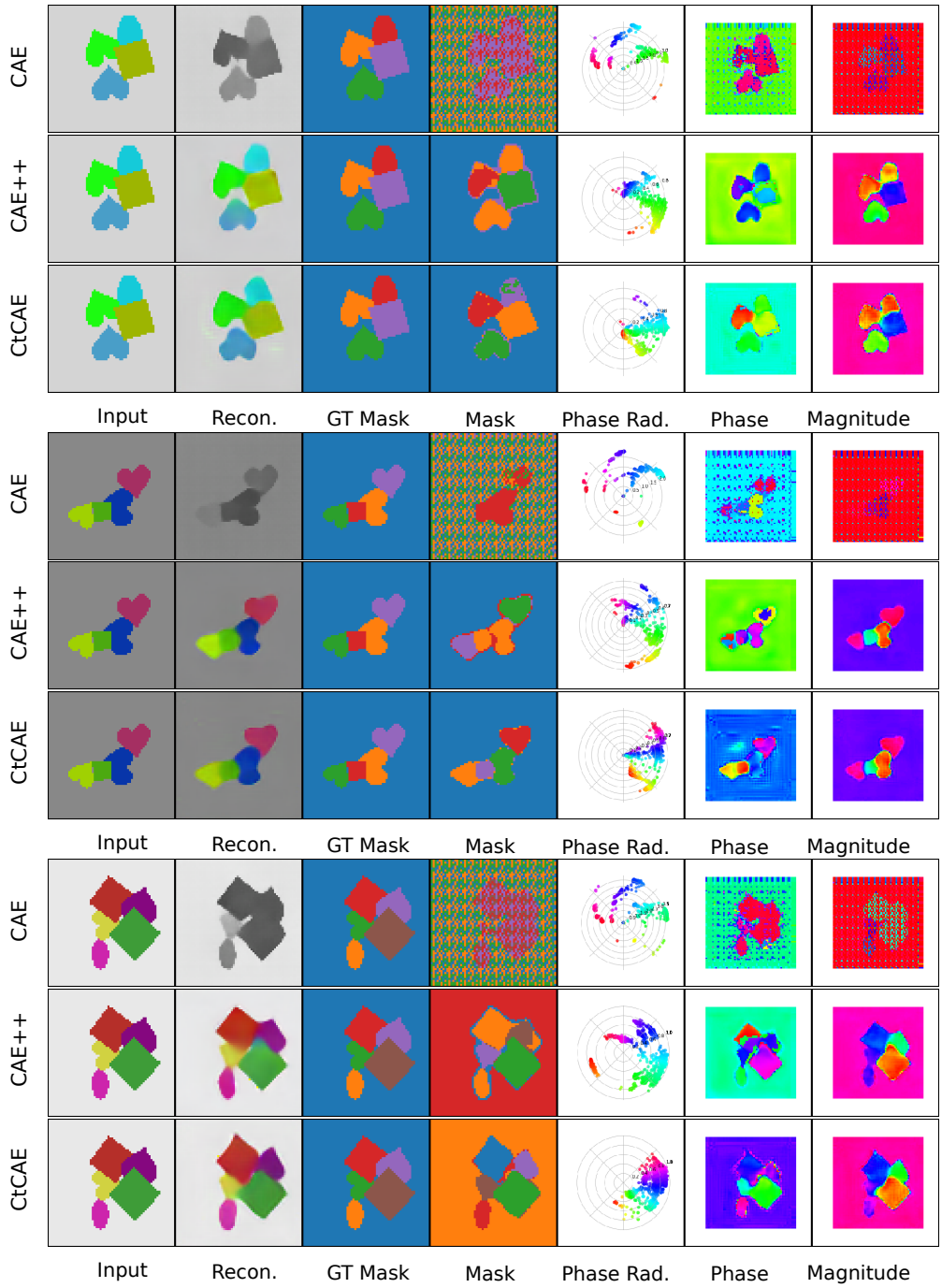


Figure C.7: Performance for CAE, CAE++ and CtCAE on images containing many objects on `dSprites`. CAE shows poor grouping performance, as noted previously. CtCAE has no issues grouping up to 5 objects in the same image, whereas CAE++ struggles, often grouping multiple objects (typically of same or similar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

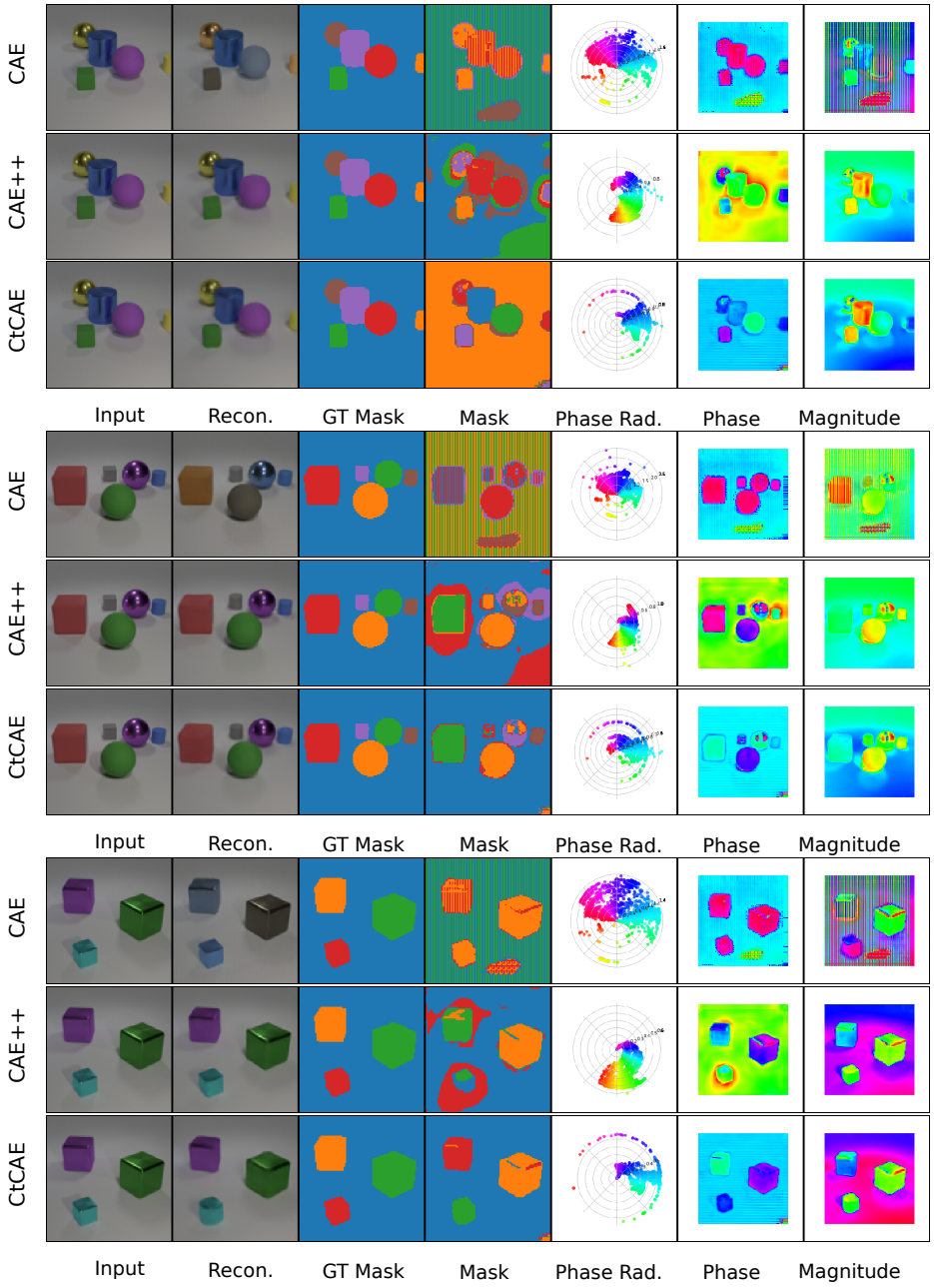


Figure C.8: Visualization of groupings for CAE, CAE++ and CtCAE on CLEVR dataset. CAE has a very large spread in phase values (column 5), but almost random object discovery performance, often grouping all objects together. But if we see the actual phase values assigned to objects (column 6), we observe that CAE tends to assign very similar phase values (similar colors) to regions belonging to different objects whereas the corresponding maps show better separation in CAE++ and much superior in CtCAE.

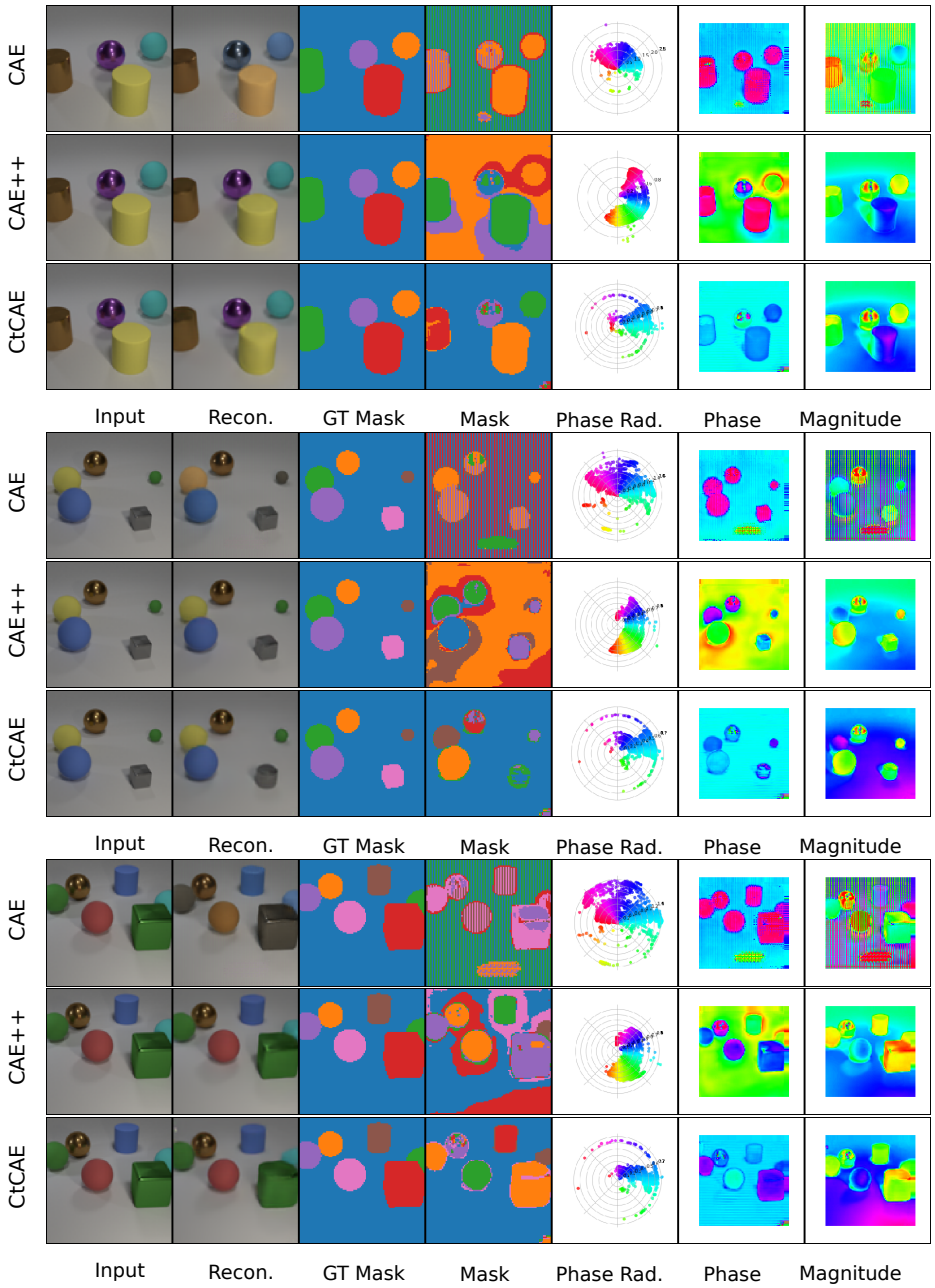


Figure C.9: Visualization on groupings for CAE, CAE++ and CtCAE on samples with 4, 5 and 6 objects on CLEVR dataset. CAE consistently groups objects poorly. CAE++ improves grouping substantially with respect to CAE, but still struggles in certain cases. Eg., in the top and middle panels it wrongly groups together 2 or more different objects for scenes with 4, 5 or 6 objects. CtCAE improves on these grouping failures of CAE++. CtCAE correctly groups all objects in the scene with 4 and 5 objects, even though it makes noticeable grouping errors with 6 objects.

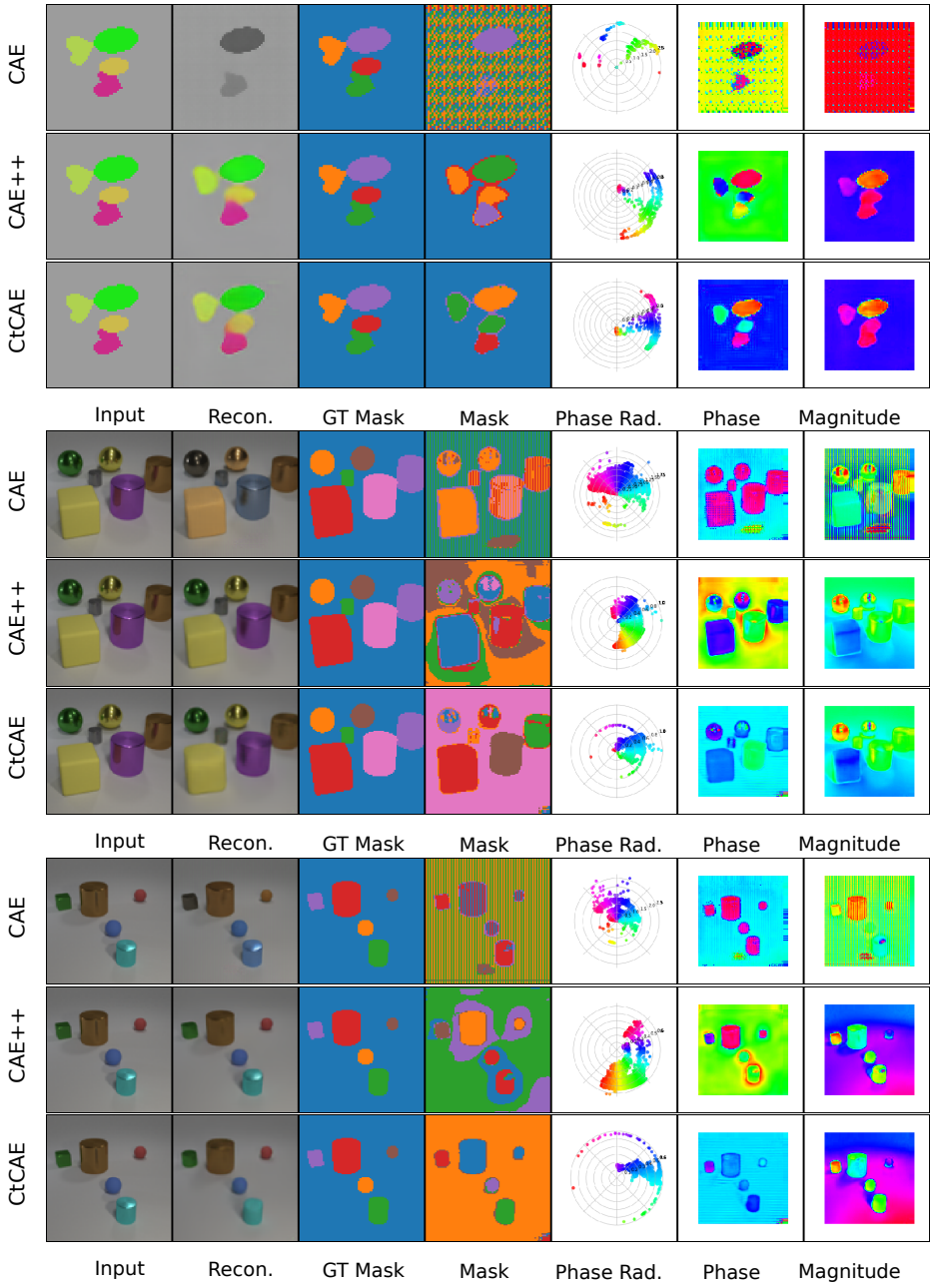


Figure C.10: One of the limitations (failure modes) we observe is that in some images containing many objects, with some of them having the same (or similar) colors, our CtCAE model groups two such objects into the same cluster. Even in this example, CtCAE still clearly outperforms CAE++, which in turn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

Appendix D

Appendix on Recurrent Complex-weighted Autoencoders

D.1 Datasets

We evaluate all models on a subset of three datasets from the multi-object suite [Kabra et al., 2019] namely: `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]. To generate the grayscale variant of the `CLEVR` dataset we apply the color to grayscale conversion function from the `Pillow` library as part of the data preprocessing pipeline.

D.2 Models

Table D.1 shows the architecture specifications such as number of layers, kernel sizes, stride lengths, number of channels etc., for the convolution layers used by the encoder and decoder modules in SynCx. We reproduce unsupervised object discovery results of the RF baseline by adapting the source code released

by the authors [1]. Similarly, we reproduce unsupervised object discovery for the CAE, CAE++ and CtCAE by adapting the source code released by the authors [2].

Encoder
3×3 conv, 64 channels, stride 2, modReLU
3×3 conv, 128 channels, stride 2, modReLU
3×3 conv, 128 channels, stride 2, modReLU
Decoder
Nearest neighbor upsample x2
3×3 conv, 128 channels, stride 1, modReLU
Nearest neighbor upsample x2
3×3 conv, 64 channels, stride 1, modReLU
Nearest neighbor upsample x2
3×3 conv, 64 channels, stride 1, modReLU
For 64×64 and 96×96 inputs, 1 additional decoder layer:
1×1 conv, 64 channels, stride 1, modReLU
Output Layer
1×1 conv, 3 channels, stride 1

Table D.1: Encoder and Decoder architecture specifications for SynCx.

D.3 Training Details

Table D.2 shows the hyperparameter configurations used to report unsupervised object discovery results (Table 5.1) for SynCx. To reproduce the unsupervised object discovery results for RF in Table 5.1 we use the hyperparameter configurations reported by the authors [Löwe et al., 2023] listed in Appendix D.6.

D.4 Computational Efficiency

We report the training and inference time (wall-clock) for our models across 3 image resolutions for the 3 datasets used in this work from the multi-object

[1] https://github.com/loeweX/RotatingFeatures
[2] https://github.com/agopal42/ctcae

Hyperparameter	Tetrominoes	dSprites	CLEVR
Training Steps	40,000	100,000	100,000
Batch size	64	16	32
Learning rate	5e-4	5e-4	5e-4
Gradient Norm Clipping	1.0	1.0	1.0
Number of iterations	3	3	4
Phase initialization	von-Mises	von-Mises	von-Mises

Table D.2: Training hyperparameters for SynCx.

suite. Inference on the test set containing 320 images of 35x35 resolution takes 7.87 seconds using 3 iterations, for 64x64 images it takes 38.96 seconds using 3 iterations and for the 96x96 images it takes 112.75 seconds using 4 iterations on a single NVIDIA GTX1080Ti GPU. Training time(s) on the other hand differs depending on the image resolution, number of iterations and model size. To train our model for 40k steps on 35x35 resolution images from Tetrominoes took 1.7 hours on a NVIDIA Tesla P100 GPU. To train our model for 100k steps on 64x64 resolution images from dSprites took 4.25 hours on a NVIDIA Tesla P100 GPU. To train our model for 100k steps on 96x96 resolution images from CLEVR took 17.87 hours on a NVIDIA Tesla V100-SXM2 GPU. To reproduce all the results/tables (mean and std-dev across 5 seeds) reported in this work we estimate the compute requirement to be 390 GPU hours in total for training models (RF and SynCx). Further, we estimate that the total compute used in this project is roughly 10-15 times more than the above figure, used for experiments in the prototyping phase. Note, that the figure above does not account for the phase map visualization and evaluation (clustering) procedures to extract object masks using the trained models primarily executed on CPUs.

D.5 Phase Map Visualization

We seek to visualize the complex-valued feature map $\mathbf{h} \in \mathbb{C}^{h' \times w' \times d_{\text{out}}}$ at some intermediate layer of the decoder module as an image. We apply dimensionality reduction along the channels of $\mathbf{h}$ so as to represent it as an one dimensional feature map. We start by constructing a complex-valued feature map in polar form with phase components $(\boldsymbol{\phi}_h)$ and magnitude components being the identity, i.e., $\boldsymbol{\mu}_h \leftarrow \mathbf{1} \in \mathbb{R}^{h' \times w' \times d_{\text{out}}}$. Then, we convert the feature map obtained in the previous step from polar to Cartesian form. Next, we apply

t-SNE to project the Cartesian domain feature map in two dimensions at each spatial location. We use the t-SNE implementation in the `scikit-learn` library [Pedregosa et al., 2011] (`sklearn.manifold.TSNE`) with `n_iter` set to 500, `metric` set to Euclidean and `perplexity` set to 20. We also experimented with UMAP [McInnes et al., 2018a] for the dimensionality reduction computation but found it to be inferior to t-SNE qualitatively (see Figure D.4) and processing time per image (UMAP takes $\approx$ 24 seconds whereas t-SNE takes $\approx$ 8 seconds). We use the UMAP implementation in the `umap-learn` library [McInnes et al., 2018b] (`umap.UMAP`) with default arguments. Therefore, we use t-SNE for dimensionality reduction in the phase visualization process. We recover 'composite' phase $\overline{\boldsymbol{\phi}}_h \in \mathbb{R}^{h' \times w'}$ by converting the two dimensional feature back to the polar form. The 'composite' magnitude $\overline{\boldsymbol{\mu}}_h \in \mathbb{R}^{h' \times w'}$ at each spatial location of the map is the Euclidean norm (along channels) of the vector-valued features (i.e., magnitude components of $\mathbf{h}$) at that location. The 'composite' phase at every spatial location is visualized as the heatmap value and the 'composite' magnitude is its distance from the center in the radial plot. The colors of points are matched across the heatmap and the radial plot such that the color of a pixel in the former correspond to its orientation in the latter. Lastly, the magnitude of background regions are masked out as was the case while extracting object assignments from phase maps.

D.6 Evaluation Details

To evaluate our model for the unsupervised object discovery task we require discrete object assignments at every location in the image. We cluster the continuous phase components of the complex-valued feature maps from an intermediate layer (second to last by default) of the decoder to compute these object assignments for every pixel. We use the k-means implementation in the scikit-learn library [Pedregosa et al., 2011] (`sklearn.clustering.KMeans`) with `n_clusters` set using the ground-truth value for each dataset and `n_init` set to 5.

D.7 Additional Results

Dataset	Model	MSE ↓	ARI ↑
Tetrominoes	CAE	4.57e-2 ± 1.08e-3	0.00 ± 0.00
	CAE++	5.07e-5 ± 2.80e-5	0.78 ± 0.07
	CtCAE	9.73e-5 ± 4.64e-5	0.84 ± 0.09
	RF	5.27e-6 ± 2.60e-6	0.42 ± 0.09
	SynCx	2.07e-3 ± 1.09e-4	0.89 ± 0.01
dSprites	CAE	8.16e-3 ± 2.54e-5	0.05 ± 0.02
	CAE++	1.60e-3 ± 1.33e-3	0.51 ± 0.08
	CtCAE	1.56e-3 ± 1.58e-4	0.56 ± 0.11
	RF	5.96e-4 ± 1.62e-4	0.84 ± 0.03
	SynCx	1.73e-3 ± 1.42e-4	0.82 ± 0.01
CLEVR	CAE	1.50e-3 ± 4.53e-4	0.04 ± 0.03
	CAE++	2.41e-4 ± 3.45e-5	0.27 ± 0.13
	CtCAE	3.39e-4 ± 3.65e-5	0.54 ± 0.02
	RF	2.69e-4 ± 9.37e-5	0.65 ± 0.01
	SynCx	3.30e-4 ± 5.30e-5	0.59 ± 0.03

Table D.3: MSE and ARI scores (mean ± standard deviation across 5 seeds) for CAE, CAE++, CtCAE, RF and SynCx models for `Tetrominoes`, `dSprites` and `CLEVR`. Results for CAE, CAE++ and CtCAE baselines are taken from Stanić et al. [2023].

D.8 Additional Visualizations

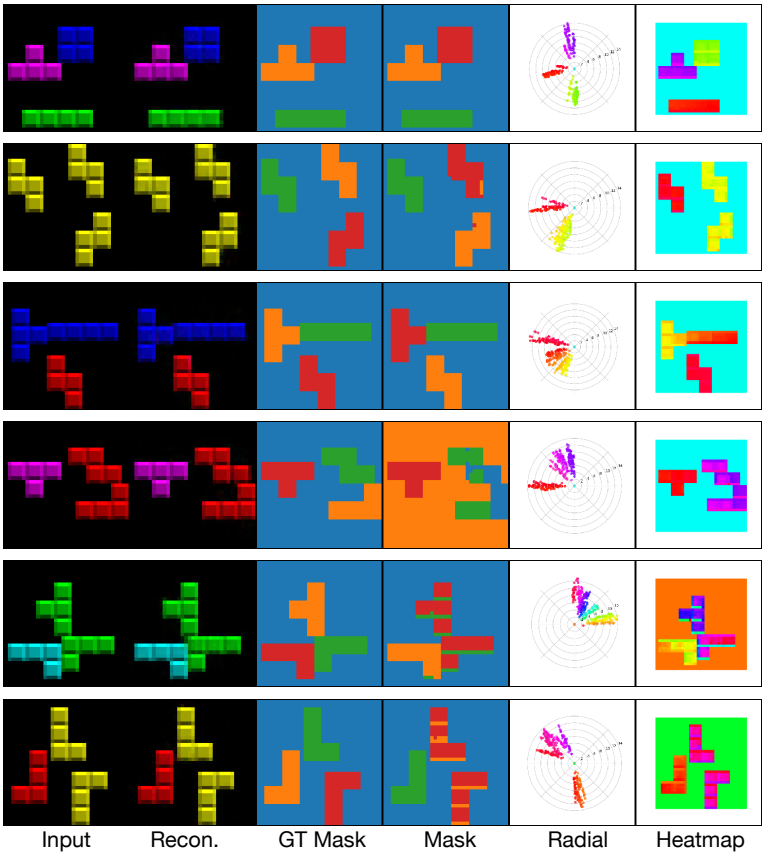


Figure D.1: Collection of images grouped by SynCx from the `Tetrominoes` dataset. Rows 1-3 show cases where SynCx perfectly groups the 3 tetris blocks even with multiple objects of the same color. Row 4 shows a failure mode where it groups the two red blocks into one straight parts and two $\mathbb{L}$-shaped parts. Rows 5 and 6 show another failure mode where it struggles to decompose the two similarly colored tetris blocks. Rather SynCx learns specialized phases for the edges and interior portions of each of the similarly colored tetris blocks. This could be due to poor random initialization of phase maps.

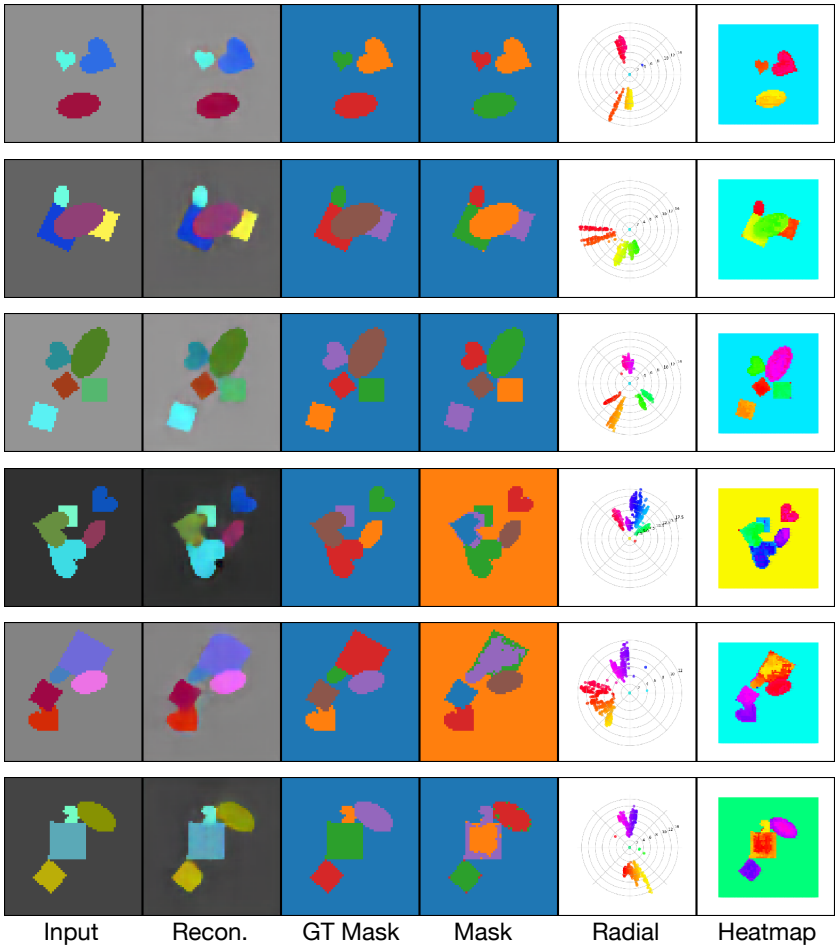


Figure D.2: Collection of images grouped by SynCx from the `dSprites` dataset. Rows 1-3 show cases where SynCx perfectly groups images containing 3-5 objects. We can also see that the pairs of phase map and radial plot visualizations show discernible specialization in phase values towards an object. Row 4 shows a grouping failure where the square and heart in light blue are incorrectly grouped together. However, on closer inspection of the corresponding phase plots we can see that the phases do show specialization (see row 4 column 6) towards both the square (light blue) and heart (dark blue). This grouping failure is a limitation in the clustering process (KMeans) to extract object assignments and assumes clusters have equal angular variance which need not hold true in practice. Similarly, grouping failures in rows 5 and 6, where blue oval and purple square (row 5) or light blue heart and cobalt square (row 6) are incorrectly grouped together despite showing enough phase specialization in each case.

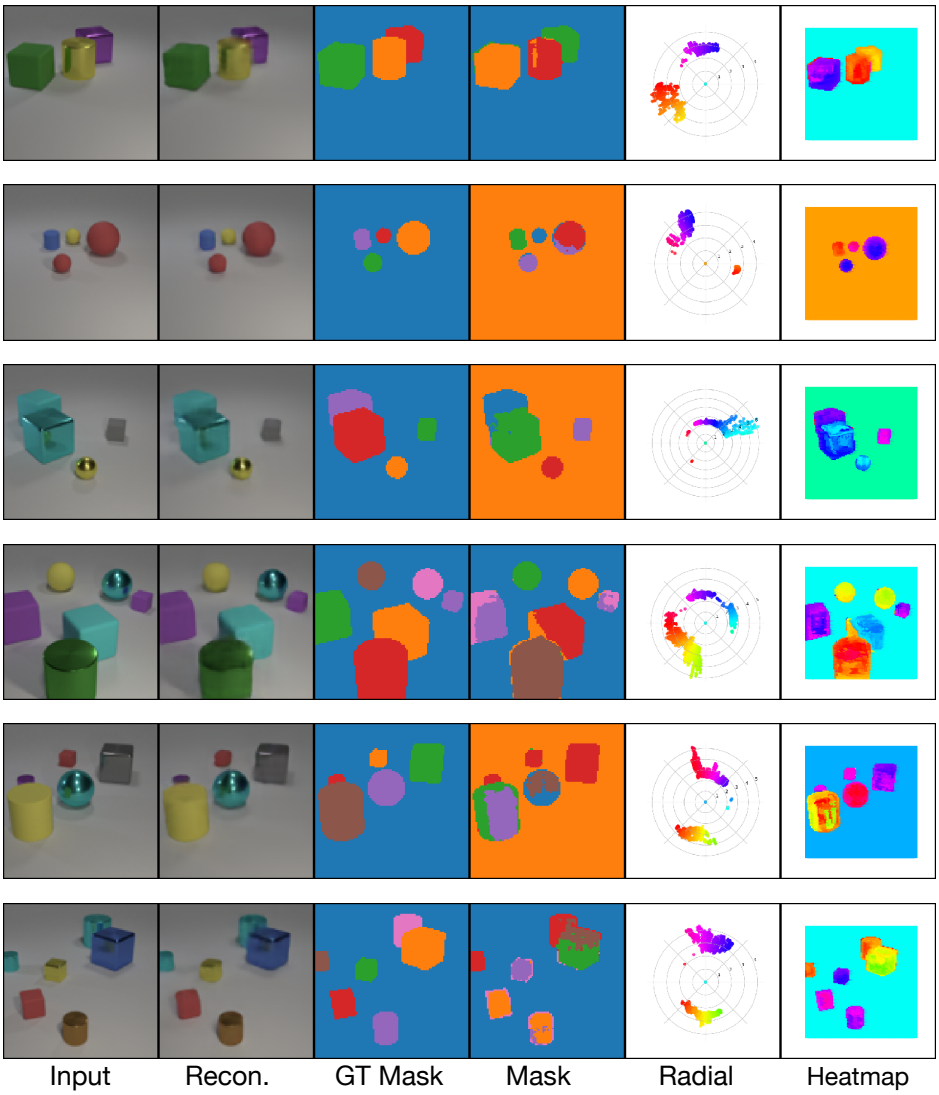


Figure D.3: Collection of images grouped by SynCx from the `CLEVR` dataset. Rows 1-4 show cases where SynCx groups scenes containing three to six objects reasonably well. Row 5 shows a failure mode where the gray, red and purple objects are incorrectly grouped together despite showing noticeable (see column 6). This can be attributed to the limitations in the clustering process similar to the ones highlighted previously with regards to assumption of equal angular variance of clusters. Similar effects can be observed with the grouping errors in row 6.

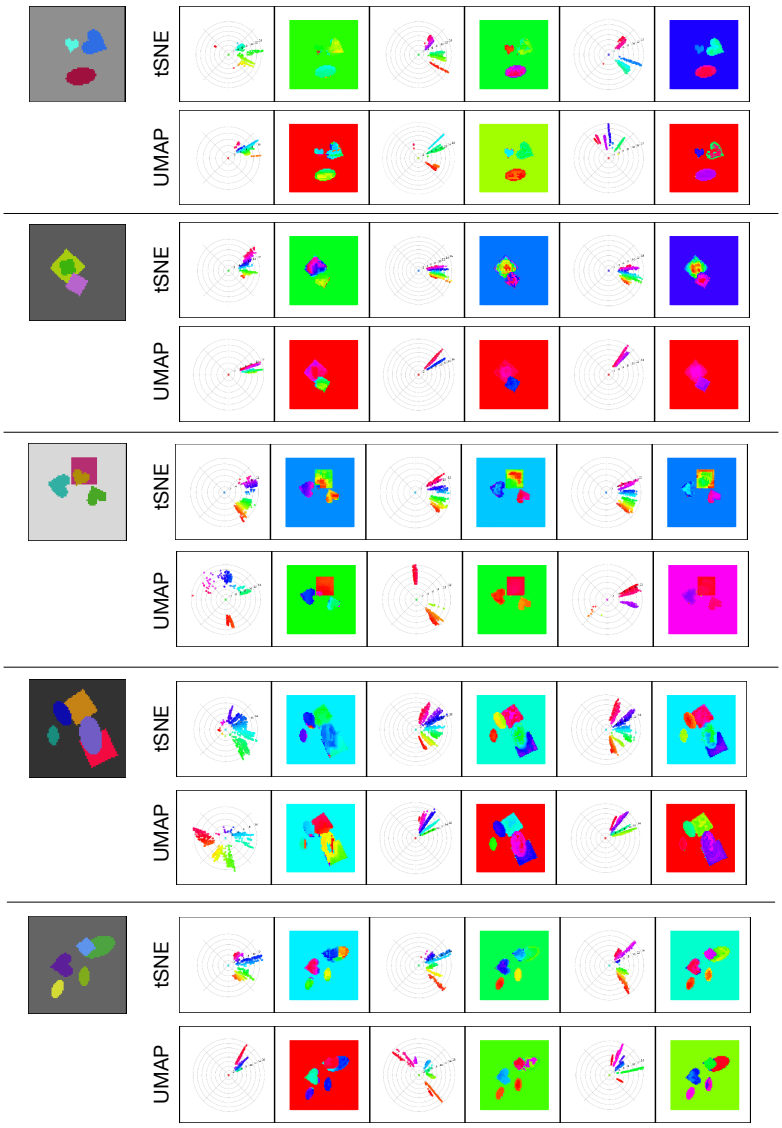


Figure D.4: Collection of samples comparing the use of t-SNE versus UMAP for the dimensionality reduction computation in the visualization process. Panels 1 and 5 show cases where the clusters of projected phase maps of both t-SNE and UMAP correlate with the number of objects in the image. Panels 2, 3 and 4 show cases where the clusters of phases projected using t-SNE correlate with the number of objects in the image whereas the clusters of phases projected using UMAP do not. Overall, we find that t-SNE produces qualitatively better projections than UMAP, i.e., phase clusters correlate better to the number of objects in the image.

Appendix E

Appendix on Polar Coordinate Position Embedding

E.1 RoPE in polar form

Although RoPE is typically described in terms of rotation matrices acting on 2D vectors of real-valued keys and queries, it can also be described using complex algebra. We begin by defining a surrogate attention score $\bar{\boldsymbol{a}}_{mn} \in \mathbb{C}$ for a query in position m and a key in position n:

$$\bar{\boldsymbol{a}}_{mn} = (\boldsymbol{R}^m \boldsymbol{q}_m)^H (\boldsymbol{R}^n \boldsymbol{k}_n) = \boldsymbol{q}_m^H \boldsymbol{R}^{n-m} \boldsymbol{k}_n = \sum_{c=1}^{d/2} \boldsymbol{q}_{mc}^H \, \rho((n-m)\boldsymbol{\theta}_c) \, \boldsymbol{k}_{nc} \tag{E.1}$$

where H is the Hermitian transpose, $\boldsymbol{q}_m, \boldsymbol{k}_n \in \mathbb{C}^{d/2}$ are the complex-valued queries and keys in Cartesian form obtained by grouping pairs of elements of the output of query and key linear projection, $\boldsymbol{R} \in \mathbb{C}^{\frac{d}{2} \times \frac{d}{2}}$ specifies a sparse (banded) matrix for a given position ($\boldsymbol{R}^m$, $\boldsymbol{R}^n$) or relative position ($\boldsymbol{R}^{n-m}$), $\boldsymbol{\theta}_c \in \mathbb{R}^{d/2}$ is a vector of reciprocal of the wavelengths of rotation for component, $\boldsymbol{\theta}_c = (\theta^{-2(c-1)/d})$, where θ is the base wavelength and $\rho(\theta) = e^{i\theta} \in \mathbb{C}$ is a rotation in the complex plane (phase-shift) by angle $\theta \in \mathbb{R}$. Then, to obtain the attention score we take the real component of $\bar{\boldsymbol{a}}_{mn}$, i.e. $\boldsymbol{a}_{mn} = \Re(\bar{\boldsymbol{a}}_{mn})$. We use indices c in $\boldsymbol{k}_{nc}$ and $\boldsymbol{q}_{mc}$ to denote the cth component of the key and query vectors, respectively.

We can express the keys and queries in polar coordinates as:

$$\boldsymbol{q}_{mc} = \mu(\boldsymbol{q}_{mc}) e^{i\psi(\boldsymbol{q}_{mc})} \quad \text{and} \quad \boldsymbol{k}_{nc} = \mu(\boldsymbol{k}_{nc}) e^{i\psi(\boldsymbol{k}_{nc})} \tag{E.2}$$

where $\mu(\boldsymbol{x})$ and $\psi(\boldsymbol{x})$ compute the magnitude and phase of each complex-valued element in the vector $\boldsymbol{x}$.
By substituting Equation (E.2) into Equation (E.1), we get:

$$\begin{aligned}\bar{\boldsymbol{a}}_{mn} &= \sum_{c=1}^{d/2} \left(\mu(\boldsymbol{q}_{mc})e^{i\psi(\boldsymbol{q}_{mc})}\right)^{\mathsf{H}} e^{i(n-m)\boldsymbol{\theta}_c}\mu(\boldsymbol{k}_{nc})e^{i\psi(\boldsymbol{k}_{nc})} \\ &= \sum_{c=1}^{d/2} \mu(\boldsymbol{q}_{mc})e^{-i\psi(\boldsymbol{q}_{mc})}e^{i(n-m)\boldsymbol{\theta}_c}\mu(\boldsymbol{k}_{nc})e^{i\psi(\boldsymbol{k}_{nc})} \\ &= \sum_{c=1}^{d/2} \mu(\boldsymbol{q}_{mc})\mu(\boldsymbol{k}_{nc})e^{i(\psi(\boldsymbol{k}_{nc})-\psi(\boldsymbol{q}_{mc})+(n-m)\boldsymbol{\theta}_c)}\end{aligned} \tag{E.3}$$

As mentioned before the attention score $\boldsymbol{a}_{mn}$ is the real component of $\bar{\boldsymbol{a}}_{mn}$:

$$\boldsymbol{a}_{mn} = \sum_{c=1}^{d/2} \mu(\boldsymbol{q}_{mc})\mu(\boldsymbol{k}_{nc}) \cos\Big((n-m)\boldsymbol{\theta}_c + \psi(\boldsymbol{k}_{nc}) - \psi(\boldsymbol{q}_{mc})\Big) \tag{E.4}$$

E.2 Datasets

Indirect Indexing. This diagnostic task (Indirect Idx.) requires the model to locate a target character in a variable-length source string that is at a certain relative distance (left or right) from a source character. It tests for the structured manipulation (pointer arithmetic operations) of the content and positional information of tokens by a sequence processing architecture. The dataset for this task is constructed by procedurally generating examples of source strings, source character and relative shifts. We generate source strings of length between 20 and 40 characters from the set of uppercase [A-Z] and lowercase [a-z] letters by uniform sampling without replacement. We uniformly sample shifts in the range [-15, +15] where - and + indicate left and right shifts respectively. We generate a train/validation/test splits of size 1M/10k/10k respectively. We use character-level tokenization and supervise the model only on the final answer (target character) token. The format of each examples is: <source string>, <source character>, <shift>, <target character> and ',' as a delimiter and the model is given the entire sequence except the target character. Below are a few samples from the dataset for reference.

```
TzbkWoKDyscBepYvfwxEVQtgPa, c, -8, b
NZTUIGWkXFrhCJDzscat, N, +4, I
RBEvOPgtaGDnjhbJCLScruZpMNsyWfQxXFAzUT, x, +2, F
```

OpenWebText. This dataset is an open source effort to reproduce OpenAI's WebText dataset [Gokaslan and Cohen, 2019] which was used to train GPT-2 [Radford et al., 2019]. The training and validation splits roughly contain 9B and 4M tokens respectively and maximum sequence length of 1024 for pretraining. We use the GPT-2 tokenizer with a vocabulary size of 50257.

Bach-Chorales. This dataset (JSB) consists of 4-part scored choral music, which are represented as a matrix with rows corresponding to voices and columns to time discretized to 16th notes. The entries in this 2D matrix are integers that denote the pitch being played. We serialize this matrix in raster-scan fashion by first going down the rows and then moving right through the columns as in prior work [Huang et al., 2019]. We use the variant of the dataset with 16th note temporal "quantizations" where silence is represented by a pitch of -1 rather than NaN, available in JSON file format [1]. We use a maximum sequence length of 2048 for training with 229/76/77 sequences present in the train/validation/test sets. We use a vocabulary size of 90 which includes the MIDI notes, silence and padding tokens.

MAESTRO. The dataset contains about 200 hours of paired audio and MIDI recordings from ten years of International Piano-e-Competition. The MIDI data includes key strike velocities and sustain/sostenuto/una corda pedal positions. We use version v3.0.0 of this dataset in MIDI format [2] for our experiments. We apply data augmentation by using pitch transpositions sampled uniformly from $\{-3, -2, -1, 0, +1, +2, +3\}$ similar to prior work [Huang et al., 2019], divide it into sequences with a maximum length of 2048 and use a 90/5/5 percent split for train/validation/test sets. We use the REMI tokenizer [Huang and Yang, 2020] with EOS, BOS, MASK and PAD tokens leading to a total vocabulary size of 328.

Human Reference Genome. The human reference genome (HRG) dataset was constructed by considering all autosomal and sex chromosomes sequences from reference assembly GRCh38/hg38 [3] and reached a total of 3.2 billion nucleotides. We follow the preprocessing and tokenization procedures from the recent state-of-the-art model for genomic sequence modeling, the Nucleotide Transformer [Dalla-Torre et al., 2025].

[1] https://github.com/czhuang/JSB-Chorales-dataset
[2] https://magenta.withgoogle.com/datasets/maestro
[3] https://huggingface.co/datasets/InstaDeepAI/human_reference_genome

PG-19. This dataset includes a set of books extracted from the Project Gutenberg books library, that were published before 1919. It is a popular dataset first introduced by [Rae et al., 2020] to benchmark long-range language models. In this work, we use it to evaluate the test-time length extrapolation capabilities of Transformer models that use different relative positional encoding schemes. We use the test split of the dataset containing 100 books or roughly 7M tokens. [4]

E.3 Models

Table E.1 contains the Transformer model configuration used on each dataset.

Hyperparameter	Indirect Idx.	OpenWebText	JSB	MAESTRO	HRG
Embedding size	512	768/1024/1280	256	384	1024
Num. heads	8	12/16/20	8	8	16
Num. layers	8	12/16/36	6	6	16
Norm. type	RMSNorm	RMSNorm	RMSNorm	RMSNorm	RMSNorm
Base wavelength (θ)	10,000	10,000	10,000	10,000	10,000
Init. range for $\boldsymbol{\theta}_b$	2π	0/0/0	2π	2π	2π
Dropout	0.0	0.0/0.0/0.0	0.2	0.1	0.1

Table E.1: Transformer model configurations for the different datasets. For the OpenWebText language modeling dataset we train 3 model sizes 124M/253M/774M and the hyperparams for each of these model sizes as a triple in column 2.

[4] https://huggingface.co/datasets/emozilla/pg19-test

E.4 Training Details

Table E.2 contains the Transformer model hyperparameters for all the datasets.

Hyperparameter	Indirect Idx.	OpenWebText	JSB	MAESTRO	HRG
Batch size	64	64	4	16	64
Sequence length	40	1024	2048	2048	1000
Learning rate	2e-4	6e-4	6e-4	6e-4	2.5e-4
Min. learning rate	2e-5	6e-5	6e-5	6e-5	2.5e-5
Weight decay	0.01	0.01	0.01	0.01	0.01
Grad. clipping	1.0	1.0	1.0	1.0	1.0
β_2 for AdamW	0.99	0.95	0.99	0.99	0.999
Max. iters	100,000	100,000	3000	60,000	100,000
Decay iters	100,000	100,000	3000	60,000	100,000
Warmup iters	4000	1000	10	500	4000

Table E.2: Hyperparameter configurations for training on different datasets. For OpenWebText dataset we train Transformer models at sizes 124M/253M/774M using identical hyperparameters.

E.5 Additional Results

To test the efficacy of key design choices, we run ablations by training 124M Transformer models on OpenWebText with identical configurations, comparing our full PoPE method against variants that remove either the Softplus activation $\sigma()$ (which ensures non-negativity of $\boldsymbol{\mu}_k$ and $\boldsymbol{\mu}_q$) or the learnable bias vector $\boldsymbol{\theta}_b$ applied to the rotation angles. From Table E.3 we see that by removing each of these components the performance of our PoPE variants worsens with increases in perplexity scores. The full PoPE model outperforms these ablated variants and indicates the efficacy of these design choices.

Positional Encoding	124M
PoPE without $\sigma()$	21.57
PoPE without θ_b	21.42
Full PoPE	**21.33**

Table E.3: Validation set perplexity scores for the 124M Transformer model with ablated versions of PoPE pretrained on OpenWebText.

E.6 Additional Visualizations

We apply the frequency usage analysis for the 253M-sized Transformer model pretrained on OpenWebText (from Table 6.4). From Figure E.1 and Figure E.2, we see a similar pattern emerge as before, i.e. RoPE tends to use a sparse set of frequencies indicated by the high norms and does not use the highest frequencies. Whereas PoPE uses a broader set of frequencies most notably the highest ones.

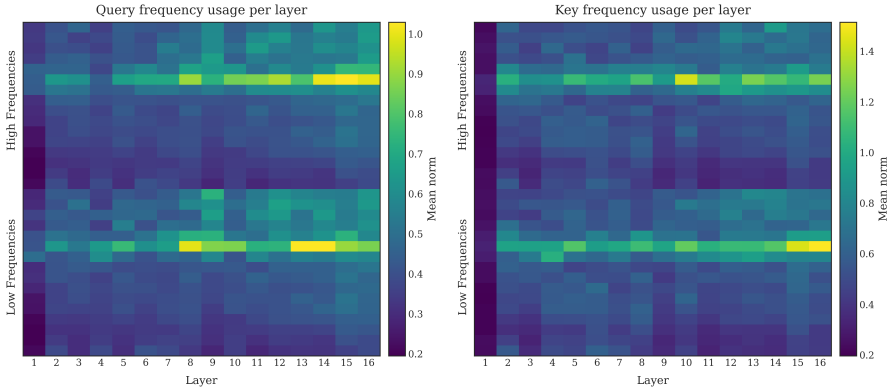


Figure E.1: 2-norm plotted over 2D RoPE components of queries (left) and keys (right) in each layer of the 253M Transformer over different RoPE frequencies. Mean over 10 Shakespeare sonnets and 16 attention heads at each layer.

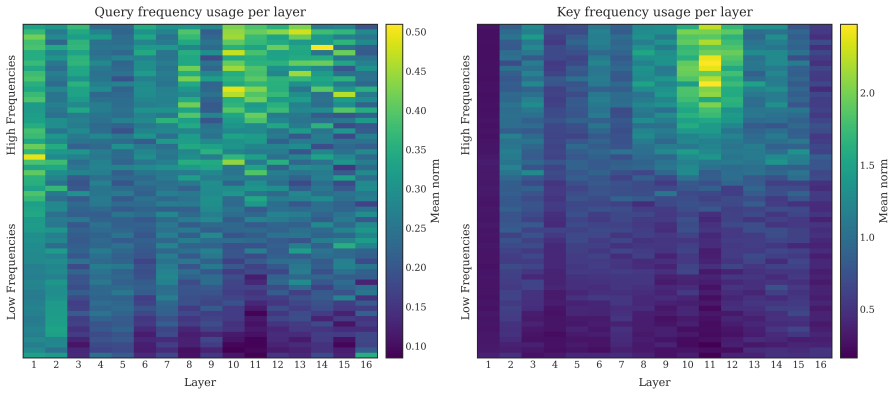


Figure E.2: Magnitudes of complex-valued features of queries (left) and keys (right) in each layer of 253M Transformer over different PoPE frequencies. Mean over 10 different Shakespeare sonnets and 16 attention heads at each layer.

Bibliography

Andrea Agostinelli, Timo I Denk, Zalán Borsos, Jesse Engel, Mauro Verzetti, Antoine Caillon, Qingqing Huang, Aren Jansen, Adam Roberts, Marco Tagliasacchi, et al. Musiclm: Generating music from text. *arXiv preprint arXiv:2301.11325*, 2023.

Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. OPAL: Offline primitive discovery for accelerating offline reinforcement learning. In *Int. Conf. on Learning Representations (ICLR)*, 2021.

B. Alexe, Thomas Deselaers, and V. Ferrari. What is an object? In *Proc. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

Shun-ichi Amari. A theory of adaptive pattern classifiers. *IEEE Trans. Electron. Comput.*, 16(3):299–307, 1967.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2017.

Anthropic. The claude 3 model family: Opus, sonnet, haiku. Technical report, Anthropic, 2024.

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2016.

Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.

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

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proc. AAAI Conf. on Artificial Intelligence*, 2017.

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

Nicholas Baker, Hongjing Lu, Gennady Erlikhman, and Philip J. Kellman. Deep convolutional networks do not classify based on global object shape. *PLoS Computational Biology*, 14, 2018.

Bram Bakker, Jürgen Schmidhuber, et al. Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization. In *Proc. of the 8-th Conf. on Intelligent Autonomous Systems*, pages 438–445. Citeseer, 2004.

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

Jason Baldridge, Jakob Bauer, Mukul Bhutani, Nicole Brichtova, Andrew Bunner, Kelvin Chan, Yichang Chen, Sander Dieleman, Yuqing Du, Zach Eaton-Rosen, et al. Imagen 3. *arXiv preprint arXiv:2408.07009*, 2024.

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder. In *ICML Workshop on Automated Reasoning*, 2021.

V. Bapst, Alvaro Sanchez-Gonzalez, C. Doersch, Kimberly L. Stachenfeld, Pushmeet Kohli, P. Battaglia, and Jessica B. Hamrick. Structured agents for physical construction. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2019.

Federico Barbero, Alex Vitvitskyi, Christos Perivolaropoulos, Razvan Pascanu, and Petar Veličković. Round and round we go! What makes rotary positional encodings useful? In *Int. Conf. on Learning Representations (ICLR)*, 2025.

Horace B Barlow. Unsupervised learning. *Neural computation*, 1(3):295–311, 1989.

Horace B Barlow et al. Possible principles underlying the transformation of sensory messages. *Sensory communication*, 1(01):217–233, 1961.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2016.

Peter W. Battaglia, Jessica B. Hamrick, and Joshua B. Tenenbaum. Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, 110(45):18327–18332, 2013.

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.

Sara Beery, Grant Van Horn, and Pietro Perona. Recognition in terra incognita. In *Proc. The European Conference on Computer Vision (ECCV)*, 2018.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47(1):253–279, 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.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proc. AAAI Conf. on Artificial Intelligence*, 2020.

Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2012.

Tim Brooks, Bill Peebles, Connor Holmes, Will DePue, Yufei Guo, Li Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, Clarence Ng, Ricky Wang, and Aditya Ramesh. Video generation models as world simulators. Technical report, OpenAI, 2024.

Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890, 2019. doi: 10.1126/science.aay2400.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott

Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Neil D. B. Bruce and John K. Tsotsos. Saliency based on information maximization. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2005.

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.

Declan Iain Campbell, Sunayana Rane, Tyler Giallanza, C. Nicolò De Sabbata, Kia Ghods, Amogh Joshi, Alexander Ku, Steven M Frankland, Thomas L. Griffiths, Jonathan D. Cohen, and Taylor Whittington Webb. Understanding the limits of vision language models through the lens of the binding problem. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proc. The IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

Michael Chang, Tomer Ullman, Antonio Torralba, and Joshua B. Tenenbaum. A compositional object-based approach to learning physical dynamics. In *Int. Conf. on Learning Representations (ICLR)*, 2017.

Chang Chen, Yi-Fu Wu, Jaesik Yoon, and Sungjin Ahn. Transdreamer: Reinforcement learning with transformer world models. In *NeurIPS Workshop on Deep Reinforcement Learning*, 2021a.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2021b.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*, 2023.

E. Colin Cherry. Some experiments on the recognition of speech, with one and with two ears. *Journal of the Acoustical Society of America*, 25:975–979, 1953.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

Kyunghyun Cho, Çağlar Gülçehre, Bart van Merriënboer, Dzmitry Bahdanau, Fethi Bougares Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proc. Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv*, abs/1803.05457, 2018.

Karl Cobbe, Oleg Klimov, Christopher Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2018.

Colin Conwell and Tomer Ullman. Testing relational understanding in text-guided image generation. *arXiv preprint arXiv:2208.00005*, 2022.

Colin Conwell, Rupert Tawiah-Quashie, and Tomer Ullman. Relations, negations, and numbers: Looking for logic in generative text-to-image models. *arXiv preprint arXiv:2411.17066*, 2024.

Jade Copet, Felix Kreuk, Itai Gat, Tal Remez, David Kant, Gabriel Synnaeve, Yossi Adi, and Alexandre Défossez. Simple and controllable music generation. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Eric Crawford and Joelle Pineau. Spatially invariant unsupervised object detection with convolutional neural networks. In *Proc. AAAI Conf. on Artificial Intelligence*, 2019.

Özgür Şimşek and Andrew G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2004.

Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. The neural data router: Adaptive control flow in transformers improves systematic generalization. In *Int. Conf. on Learning Representations (ICLR)*, Virtual only, April 2022.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc Viet Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proc. Association for Computational Linguistics (ACL)*, 2019.

Hugo Dalla-Torre, Liam Gonzalez, Javier Mendoza-Revilla, Nicolas Lopez Carranza, Adam Henryk Grzywaczewski, Francesco Oteri, Christian Dallago, Evan Trop, Bernardo P. de Almeida, Hassan Sirelkhatim, Guillaume Richard, Marcin Skwark, Karim Beguir, Marie Lopez, and Thomas Pierrot. Nucleotide transformer: building and evaluating robust foundation models for human genomics. *Nature Methods*, 22(2):287–297, 2025.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *Int. Conf. on Learning Representations (ICLR)*, 2024.

Peter Dayan and Geoffrey E. Hinton. Feudal reinforcement learning. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, 1992.

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 *Int. Conf. on Learning Representations (ICLR)*, New Orleans, LA, USA, May 2019.

David Ding, Felix Hill, Adam Santoro, Malcolm Reynolds, and Matt Botvinick. Attention over learned object embeddings enables complex visual reasoning. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2021a.

David Ding, Felix Hill, Adam Santoro, Malcolm Reynolds, and Matthew Botvinick. Attention over learned object embeddings enables complex visual reasoning. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2021b.

Yiran Ding, Li Lyna Zhang, Chengruidong Zhang, Yuanyuan Xu, Ning Shang, Jiahang Xu, Fan Yang, and Mao Yang. Longrope: Extending LLM context window beyond 2 million tokens. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2024.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *Int. Conf. on Learning Representations (ICLR)*, 2021.

Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6569–6578, 2019.

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 *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Patrick Emami, Pan He, Sanjay Ranka, and Anand Rangarajan. Efficient iterative amortized inference for learning symmetric and disentangled multi-object representations. In *Proc. Int. Conf. on Machine Learning (ICML)*, 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 *Int. Conf. on Learning Representations (ICLR)*, 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 *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2016.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *Int. Conf. on Learning Representations (ICLR)*, 2019.

Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. Addressing some limitations of Transformers with feedback memory. *Preprint arXiv:2002.09402*, 2020.

Jerome Feldman. The neural binding problem (s). *Cognitive neurodynamics*, *7*: 1–11, 2013.

David J Field. Relations between the statistics of natural images and the response properties of cortical cells. *Journal of the Optical Society of America A*, 4(12):2379–2394, 1987.

Michael C. Frank. Bridging the data gap between children and large language models. *Trends in Cognitive Sciences*, *27*(11):990–992, 2023. ISSN 1364-6613.

Pascal Fries, Danko Nikolić, and Wolf Singer. The gamma cycle. *Trends in neurosciences*, 30(7):309–316, 2007.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL `https://zenodo.org/records/12608602`.

Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. In *Int. Conf. on Learning Representations (ICLR)*, 2018.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, *2*(11): 665–673, 2020.

F.A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: continual prediction with lstm. In *Proc. Int. Conf. on Artificial Neural Networks (ICANN)*, volume 2, 1999.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

Joonsu Gha, Vincent Herrmann, Benjamin Grewe, Jürgen Schmidhuber, and Anand Gopalakrishnan. Unsupervised musical object discovery from audio. In *Workshop on Machine Learning for Audio, NeurIPS*, 2023.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2017.

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. `http://Skylion007.github.io/OpenWebTextCorpus`, 2019.

Anand Gopalakrishnan, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Unsupervised object keypoint learning using local spatial predictability. In *Int. Conf. on Learning Representations (ICLR)*, 2021.

Anand Gopalakrishnan, Kazuki Irie, Jürgen Schmidhuber, and Sjoerd van Steenkiste. Unsupervised Learning of Temporal Abstractions With Slot-Based Transformers. *Neural Computation*, 35(4):593–626, 2023.

Anand Gopalakrishnan, Aleksandar Stanić, Jürgen Schmidhuber, and Michael Curtis Mozer. Recurrent complex-weighted autoencoders for unsupervised object discovery. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Alex Graves. Adaptive computation time for recurrent neural networks. *Preprint arXiv:1603.08983*, 2016.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2006.

Charles M Gray and Wolf Singer. Stimulus-specific neuronal oscillations in orientation columns of cat visual cortex. *Proceedings of the National Academy of Sciences*, 86(5):1698–1702, 1989.

Klaus Greff, Antti Rasmus, Mathias Berglund, Tele Hao, Harri Valpola, and Jürgen Schmidhuber. Tagger: Deep unsupervised perceptual grouping. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2016.

Klaus Greff, Sjoerd Van Steenkiste, and Jürgen Schmidhuber. Neural expectation maximization. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, 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 *Proc. Int. Conf. on Machine Learning (ICML)*, 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 *Proc. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

Karol Gregor, George Papamakarios, Frederic Besse, Lars Buesing, and Theophane Weber. Temporal difference variational auto-encoder. In *Int. Conf. on Learning Representations (ICLR)*, 2019.

Stephen Grossberg. Adaptive pattern classification and universal recoding: Ii. feedback, expectation, olfaction, illusions. *Biological cybernetics*, 23(4): 187–202, 1976.

Stephen Grossberg and David Somers. Synchronized oscillations during cooperative feature linking in a cortical model of visual perception. *Neural Networks*, 4(4):453–466, 1991.

Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proc. Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*, 2010.

Bridgette M Hard, Barbara Tversky, and David S Lang. Making sense of abstract events: Building event schemas. *Memory & cognition*, 34(6):1221–1235, 2006.

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

H. V. Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proc. AAAI Conf. on Artificial Intelligence*, 2016.

Matthew J. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps. In *Proc. AAAI Conf. on Artificial Intelligence*, 2015.

Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *Int. Conf. on Learning Representations (ICLR)*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proc. The IEEE/CVF International Conference on Computer Vision (ICCV)*, 2015.

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.

Katherine Hermann, Ting Chen, and Simon Kornblith. The origins and prevalence of texture bias in convolutional neural networks. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proc. AAAI Conf. on Artificial Intelligence*, 2018.

Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. The goldilocks principle: Reading children's books with explicit memory representations. In *Int. Conf. on Learning Representations (ICLR)*, 2016.

Geoffrey Hinton. How to Represent Part-Whole Hierarchies in a Neural Network. *Neural Computation*, 35(3):413–452, 2023.

Geoffrey E Hinton. Distributed representations. Technical report, Carnegie Mellon University, 1984.

Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with EM routing. In *Int. Conf. on Learning Representations (ICLR)*, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997a.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997b.

Douglas R. Hofstadter. *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*. Basic Books, Inc., 1996.

Bernhard Hommel. Event files: Evidence for automatic integration of stimulus-response episodes. *Visual Cognition*, 5(1-2):183–216, 1998.

Bernhard Hommel. Event files: feature binding in and across perception and action. *Trends in Cognitive Sciences*, 8(11):494–500, 2004.

Bernhard Hommel. Feature integration across perception and action: Event files affect response choice. *Psychological research*, 71:42–63, 02 2007.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, and Douglas Eck. Music transformer: Generating music with long-term structure. In *Int. Conf. on Learning Representations (ICLR)*, 2019.

Yu-Siang Huang and Yi-Hsuan Yang. Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions. In *Proc. ACM Int. Conf. on Multimedia (MM)*, 2020.

Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2:193–218, 1985.

Kazuki Irie. Why are positional encodings nonessential for deep autoregressive transformers? revisiting a petroglyph. *Preprint arXiv:2501.00659*, 2025.

Kazuki Irie, Albert Zeyer, Ralf Schlüter, and Hermann Ney. Language modeling with deep Transformers. In *Proc. Interspeech*, 2019.

Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. Going beyond linear transformers with recurrent fast weight programmers. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Phillip Isola, Daniel Zoran, Dilip Krishnan, and Edward H Adelson. Crisp boundary detection using pointwise mutual information. In *European Conference on Computer Vision*, pages 799–814. Springer, 2014.

Phillip Isola, Daniel Zoran, Dilip Krishnan, and Edward H Adelson. Learning visual groups from co-occurrences in space and time. *arXiv preprint arXiv:1511.06811*, 2015.

L. Itti. Visual salience. *Scholarpedia*, 2(9):3327, 2007. doi: 10.4249/scholarpedia.3327. revision #72776.

Laurent Itti, Christof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 20(11):1254–1259, 1998.

Alekseĭ Grigor evich Ivakhnenko and Valentin Grigorévich Lapa. *Cybernetic Predicting Devices*. CCM Information Corporation, 1965.

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. *Preprint arXiv:2107.14795*, 2021a.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and João Carreira. Perceiver: General perception with iterative attention. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2021b.

Martin Jagersand. Saliency maps and attention selection in scale and spatial coordinates: An information theoretic approach. In *Proc. The IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, 1995.

Tomas Jakab, Ankush Gupta, Hakan Bilen, and Andrea Vedaldi. Unsupervised learning of object landmarks through conditional image generation. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Michael Janner, Sergey Levine, William T. Freeman, Joshua B. Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about physical interactions with object-centric models. In *Int. Conf. on Learning Representations (ICLR)*, 2019.

Michael Janner, Qiyang Li, and Sergey Levine. Reinforcement learning as one big sequence modeling problem. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

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.

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.

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.

Timor Kadir and Michael Brady. Saliency, scale and image description. *Int. J. Comput. Vision*, 45(2):83–105, 2001.

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.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model based reinforcement learning for atari. In *Int. Conf. on Learning Representations (ICLR)*, 2020.

Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. The impact of positional encoding on length generalization in transformers. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Charles Kemp and Joshua B. Tenenbaum. The discovery of structural form. *Proceedings of the National Academy of Sciences*, 105(31):10687–10692, 2008.

B. Kim, E. Reif, M Wattenberg, S. Bengio, and M. C. Mozer. Neural networks trained on natural scenes exhibit gestalt closure. *Computational Brain & Behavior*, 4:251–263, 2021.

Taesup Kim, Sungjin Ahn, and Yoshua Bengio. Variational temporal abstraction. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *Int. Conf. on Learning Representations (ICLR)*, 2015a.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Int. Conf. on Learning Representations (ICLR)*, 2015b.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *Int. Conf. on Learning Representations (ICLR)*, 2014.

Thomas Kipf, Yujia Li, Hanjun Dai, Vinícius Flores Zambaldi, Alvaro Sanchez-Gonzalez, Edward Grefenstette, Pushmeet Kohli, and Peter W. Battaglia. Compile: Compositional imitation learning and execution. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2019.

Thomas Kipf, Elise van der Pol, and Max Welling. Contrastive learning of structured world models. In *Int. Conf. on Learning Representations (ICLR)*, 2020.

Thomas Kipf, Gamaleldin Fathy Elsayed, Aravindh Mahendran, Austin Stone, Sara Sabour, Georg Heigold, Rico Jonschkowski, Alexey Dosovitskiy, and Klaus Greff. Conditional object-centric learning from video. In *Int. Conf. on Learning Representations (ICLR)*, 2022.

K Koffka. Principles of gestalt psychology. *Philosophy and Scientific Method*, 32(8), 1935.

W Köhler. *Gestalt psychology*. Liveright, 1929.

Wolfgang Köhler. Gestalt psychology. *Psychologische Forschung*, 31(1), 1967.

Adam Kosiorek, Hyunjik Kim, Yee Whye Teh, and Ingmar Posner. Sequential attend, infer, repeat: Generative modelling of moving objects. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

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 *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Yoshiki Kuramoto. Self-entrainment of a population of coupled non-linear oscillators. In Huzihiro Araki, editor, *Mathematical Problems in Theoretical Physics*, volume 39, pages 420–422. Springer, 1975.

Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and brain sciences*, 40, 2017.

Barbara Landau, Linda B. Smith, and Susan S. Jones. The importance of shape in early lexical learning. *Cognitive Development*, 3(3):299–321, 1988.

Barbara Landau, Linda Smith, and Susan Jones. Object perception and object naming in early development. *Trends in cognitive sciences*, 2(1):19–24, 1998.

Evelina Leivada, Elliot Murphy, and Gary Marcus. Dall·e 2 fails to reliably capture common syntactic processes. *Social Sciences and Humanities Open*, 8(1), 2023. ISSN 2590-2911.

Nanbo Li, Cian Eastwood, and Robert Fisher. Learning object-centric representations of multi-object scenes from multiple views. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

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 *Int. Conf. on Learning Representations (ICLR)*, 2020.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

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.

Sindy Löwe, Phillip Lippe, Francesco Locatello, and Max Welling. Rotating features for object discovery. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Sindy Löwe, Francesco Locatello, and Max Welling. Binding dynamics in rotating features. *ArXiv*, abs/2402.05627, 2024.

Yuchen Lu, Yikang Shen, Siyuan Zhou, Aaron Courville, Joshua B. Tenenbaum, and Chuang Gan. Learning task decomposition with ordered memory policy network. In *Int. Conf. on Learning Representations (ICLR)*, 2021.

Marlos C. Machado, Marc G. Bellemare, and Michael Bowling. A Laplacian framework for option discovery in reinforcement learning. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2017.

Gaurav Malhotra, Benjamin D. Evans, and Jeffrey S. Bowers. Hiding a plane with a pixel: examining shape-bias in cnns and the benefit of building in biological constraints. *Vision Research*, 174:57–68, 2020.

Priyanka Mandikal and Kristen Grauman. Learning dexterous grasping with object-centric visual affordances. In *IEEE international conference on robotics and automation (ICRA)*, pages 6169–6176. IEEE, 2021.

James L McClelland and David E Rumelhart. An interactive activation model of context effects in letter perception: I. an account of basic findings. *Psychological Review*, 88(5):375–407, 1981.

Tom McCoy, Ellie Pavlick, and Tal Linzen. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proc. Association for Computational Linguistics (ACL)*, 2019.

Amy McGovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2001.

Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018a.

Leland McInnes, John Healy, Nathaniel Saul, and Lukas Grossberger. Umap: Uniform manifold approximation and projection. *The Journal of Open Source Software*, 3(29):861, 2018b.

Douglas McLelland and Rufin VanRullen. Theta-gamma coding meets communication-through-coherence: neuronal oscillatory multiplexing theories reconciled. *PLoS computational biology*, 12(10), 2016.

George A Miller and Philip N Johnson-Laird. *Language and perception*. Harvard University Press, 1976.

Peter M Milner. A model for visual shape recognition. *Psychological review*, 81(6):521, 1974.

Matthias Minderer, Chen Sun, Ruben Villegas, Forrester Cole, Kevin P Murphy, and Honglak Lee. Unsupervised learning of object structure and dynamics from videos. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.

Seyed Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *Int. Conf. on Learning Representations (ICLR)*, 2025.

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 *Proc. Int. Conf. on Machine Learning (ICML)*, 2020.

Takeru Miyato, Sindy Löwe, Andreas Geiger, and Max Welling. Artificial kuramoto oscillatory neurons. In *Int. Conf. on Learning Representations (ICLR)*, 2025.

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.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2016.

Mazda Moayeri, Phillip Pope, Yogesh Balaji, and Soheil Feizi. A comprehensive study of image classification model sensitivity to foregrounds, backgrounds, and visual attributes. In *Proc. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Alexander Mott, Daniel Zoran, Mike Chrzanowski, Daan Wierstra, and Danilo Jimenez Rezende. Towards interpretable reinforcement learning using attention augmented agents. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Michael C. Mozer. Neural network architectures for temporal pattern processing. In Andreas S. Weigand and Neil A. Gershenfeld, editors, *Time Series Prediction: Forecasting the Future and Understanding the Past*, volume XVII of *Sante Fe Institute Studies in the Sciences of Complexity*, pages 243–264, Redwood City, CA, 1993. Addison-Wesley Publishing.

Michael C Mozer. A principle for unsupervised hierarchical decomposition of visual scenes. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, 1998.

Michael C Mozer, Richard Zemel, and Marlene Behrmann. Learning to segment images using dynamic feature binding. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, volume 4, 1991.

Michael C. Mozer, Richard S. Zemel, Marlene Behrmann, and Christopher K. I. Williams. Learning to Segment Images Using Dynamic Feature Binding. *Neural Computation*, 4(5):650–665, 1992.

Sabine Muzellec, Andrea Alamia, Thomas Serre, and Rufin VanRullen. Enhancing deep neural networks through complex-valued representations and kuramoto synchronization dynamics. *Transactions on Machine Learning Research*, 2025a.

Sabine Muzellec, Drew Linsley, Alekh Karkada Ashok, Ennio Mingolla, Girik Malik, Rufin VanRullen, and Thomas Serre. Tracking objects that change in appearance with phase synchrony. In *Int. Conf. on Learning Representations (ICLR)*, 2025b.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

OpenAI. Gpt-4v system card. Technical report, OpenAI, 2023.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proc. Association for Computational Linguistics (ACL)*, 2016.

Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, Matthew M. Botvinick, Nicolas Heess, and Raia Hadsell. Stabilizing Transformers for reinforcement learning. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2020.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient context window extension of large language models. In *Int. Conf. on Learning Representations (ICLR)*, 2024.

Xue Bin Peng, Michael Chang, Grace Zhang, Pieter Abbeel, and Sergey Levine. MCP: Learning composable hierarchical control with multiplicative compositional policies. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Jordan B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1):77 – 105, 1990.

Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. In *Int. Conf. on Learning Representations (ICLR)*, 2023.

Ofir Press, Noah A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *Int. Conf. on Learning Representations (ICLR)*, 2022.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. Technical report, OpenAI, 2018.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

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 *Proc. Int. Conf. on Machine Learning (ICML)*, 2021.

Gabriel A Radvansky and Jeffrey M Zacks. *Event cognition*. Oxford University Press, 2014.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *Int. Conf. on Learning Representations (ICLR)*, 2020.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of

transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research (JMLR)*, 21:140:1–140:67, 2020.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2):3, 2022.

William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

A. Ravishankar Rao, Guillermo A. Cecchi, Charles C. Peck, and James R. Kozloski. Unsupervised segmentation with dynamical units. *IEEE Transactions on Neural Networks*, 19(1):168–182, 2008.

Pradyumna Reddy, Scott Wisdom, Klaus Greff, John R. Hershey, and Thomas Kipf. Audioslots: A slot-centric generative model for audio separation. In *IEEE International Conference on Acoustics, Speech, and Signal Processing Workshops (ICASSPW)*, 2023.

David P Reichert and Thomas Serre. Neuronal synchrony in complex-valued deep networks. In *Int. Conf. on Learning Representations (ICLR)*, 2014.

Eleanor Rosch. Principles of categorization. In *Cognition and categorization*, pages 27–48. Routledge, 1978.

Adina L Roskies. The binding problem. *Neuron*, 24(1):7–9, 1999.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Mehdi S. M. Sajjadi, Daniel Duckworth, Aravindh Mahendran, Sjoerd van Steenkiste, Filip Pavetić, Mario Lučić, Leonidas J. Guibas, Klaus Greff, and Thomas Kipf. Object Scene Representation Transformer. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Rebecca Saxe and Susan Carey. The perception of causality in infancy. *Acta Psychologica*, 123(1):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, 2009.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2021.

J. Schmidhuber. Learning to generate subgoals for action sequences. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume ii, pages 453 vol.2–, 1991. doi: 10.1109/IJCNN.1991.155375.

J Schmidhuber, MC Mozer, and D Prelinger. Continuous history compression. In *Proc. of Intl. Workshop on Neural Networks*, pages 87–95, 1993.

Juergen Schmidhuber. Annotated history of modern ai and deep learning. *arXiv preprint arXiv:2212.11279*, 2022.

Jürgen Schmidhuber. Towards compositional learning in dynamic networks. *Technical Report FKI-129-90*, 1990.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to recurrent nets. *Neural Computation*, 4(1):131–139, 1992.

Jürgen Schmidhuber. Self-delimiting neural networks. *Preprint arXiv:1210.0118*, 2012.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

Jürgen Schmidhuber. Reinforcement learning upside down: Don't predict rewards–just map them to actions. *Preprint arXiv:1912.02875*, 2019.

Jürgen Schmidhuber. The 2010s: our decade of deep learning/outlook on the 2020s. `https://people.idsia.ch/~juergen/2010s-our-decade-of-deep-learning.html`, 2020.

Jürgen Schmidhuber and Reiner Wahnsiedler. Planning simple trajectories using neural subgoal generators. In *Proc. Int. Conf. on From Animals to Animats 2: Simulation of Adaptive Behavior*, page 196–202, 1993.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, K. Simonyan, L. Sifre, Simon Schmitt, A. Guez, Edward Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588 7839:604–609, 2020.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Luca M. Schulze Buschoff, Elif Akata, Matthias Bethge, and Eric Schulz. Visual cognition in multimodal large language models. *Nature Machine Intelligence*, 7(1):96–106, 2025.

Maximilian Seitzer, Max Horn, Andrii Zadaianchuk, Dominik Zietlow, Tianjun Xiao, Carl-Johann Simon-Gabriel, Tong He, Zheng Zhang, Bernhard Schölkopf, Thomas Brox, and Francesco Locatello. Bridging the gap to real-world object-centric learning. In *Int. Conf. on Learning Representations (ICLR)*, 2023.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proc. North American Chapter of the Association for Computational Linguistics on Human Language Technologies (NAACL-HLT)*, 2018.

Kyriacos Shiarlis, Markus Wulfmeier, Sasha Salter, Shimon Whiteson, and Ingmar Posner. TACO: Learning task decomposition via temporal alignment for control. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2018.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, L. Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017.

Ozgur Simsek and Andre S Barreto. Skill characterization based on betweenness. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2008.

Özgür Şimşek, Alicia P Wolfe, and Andrew G Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2005.

Wolf Singer. Neuronal synchrony: A versatile code for the definition of relations? *Neuron*, 24(1):49–65, 1999.

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)*, 2022a.

Gautam Singh, Yeongbin Kim, and Sungjin Ahn. Neural systematic binder. In *Int. Conf. on Learning Representations (ICLR)*, 2022b.

Gautam Singh, Yi-Fu Wu, and Sungjin Ahn. Simple unsupervised object-centric learning for complex and naturalistic videos. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2022c.

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.

Rupesh Kumar Srivastava, Pranav Shyam, Filipe Mutz, Wojciech Jaśkowski, and Jürgen Schmidhuber. Training agents using upside-down reinforcement learning. *Preprint arXiv:1912.02877*, 2019.

Stanić, Aleksandar Gopalakrishnan, Anand, Kazuki Irie, and Jürgen Schmidhuber. Contrastive training of complex-valued autoencoders for object discovery. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Aleksandar Stanić and Jürgen Schmidhuber. R-sqair: Relational sequential attend, infer, repeat. In *Workshop on Perception as Generative Reasoning: Structure, Causality, Probability, NeurIPS*, 2019.

Aleksandar Stanić, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Hierarchical relational inference. In *Proc. AAAI Conf. on Artificial Intelligence*, 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.

Karl Stelzner, Kristian Kersting, and Adam R Kosiorek. Decomposing 3d scenes into objects via unsupervised volume segmentation. *arXiv:2104.01148*, 2021.

Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *International Symposium on abstraction, reformulation, and approximation*, pages 212–223. Springer, 2002.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568, 2024.

Felipe Petroski Such, Vashisht Madhavan, Rosanne Liu, Rui Wang, Pablo Samuel Castro, Yulun Li, Ludwig Schubert, Marc G. Bellemare, Jeff Clune, and Joel Lehman. An atari model zoo for analyzing, visualizing, and comparing deep reinforcement learning agents. In *Proc. Int. Joint Conf. on Aritificial Intelligence (IJCAI)*, 2019.

Yutao Sun, Li Dong, Barun Patra, Shuming Ma, Shaohan Huang, Alon Benhaim, Vishrav Chaudhary, Xia Song, and Furu Wei. A length-extrapolatable transformer. In *Proc. Association for Computational Linguistics (ACL)*, 2023.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Prasad Tadepalli and Thomas G Dietterich. Hierarchical explanation-based reinforcement learning. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 358–366, Nashville, TN, USA, July 1997.

Shawn Tan, Songlin Yang, Aaron C. Courville, Rameswar Panda, and Yikang Shen. Scaling stick-breaking attention: An efficient implementation and in-depth study. In *Int. Conf. on Learning Representations (ICLR)*, Singapore, April 2025.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Tristan Thrush, Ryan Jiang, Max Bartolo, Amanpreet Singh, Adina Williams, Douwe Kiela, and Candace Ross. Winoground: Probing vision and language

models for visio-linguistic compositionality. In *Proc. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models. *Preprint arXiv:2302.13971*, 2023.

Chiheb Trabelsi, Olexa Bilaniuk, Ying Zhang, Dmitriy Serdyuk, Sandeep Subramanian, Joao Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, and Christopher J Pal. Deep complex networks. In *Int. Conf. on Learning Representations (ICLR)*, 2018.

Anne Treisman. The binding problem. *Current opinion in neurobiology*, 6(2): 171–178, 1996.

Anne M. Treisman and Garry Gelade. A feature-integration theory of attention. *Cognitive Psychology*, 12(1):97–136, 1980.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

Lieselotte Van Leeuwen, Ad Smitsman, and Cees van Leeuwen. Affordances, perceptual complexity, and the development of tool use. *Journal of Experimental Psychology: Human Perception and Performance*, 20(1), 1994.

Sjoerd van Steenkiste, Michael Chang, Klaus Greff, and Jürgen Schmidhuber. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. In *Int. Conf. on Learning Representations (ICLR)*, 2018.

Sjoerd van Steenkiste, Klaus Greff, and Jürgen Schmidhuber. A perspective on objects and systematic generalization in model-based rl. *ICML Workshop on Generative Modeling and Model-Based Reasoning for Robotics and AI*, 2019.

Sjoerd van Steenkiste, Daniel Zoran, Yi Yang, Yulia Rubanova, Rishabh Kabra, Carl Doersch, Dilara Gokay, Etienne Pot, Klaus Greff, Drew Hudson, et al. Moving off-the-grid: Scene-grounded video representations. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2017.

Rishi Veerapaneni, John D Co-Reyes, Michael Chang, Michael Janner, Chelsea Finn, Jiajun Wu, Joshua B Tenenbaum, and Sergey Levine. Entity abstraction in visual model-based reinforcement learning. In *Conference on Robot Learning*, 2019.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Christoph von der Malsburg. The correlation theory of brain function. Technical report, Dept. of Neurobiology, Max-Planck-Institute for Biophysical Chemistry, Göttingen, Germany, 1981.

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.

A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K.J. Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(3):328–339, 1989.

David Waltz. Generating semantic descriptions from drawings of scenes with shadows. Technical report, Massachusetts Institute of Technology, MIT MAC-TR 271, 1972.

Benyou Wang, Donghao Zhao, Christina Lioma, Qiuchi Li, Peng Zhang, and Jakob Grue Simonsen. Encoding word order in complex embeddings. In *Int. Conf. on Learning Representations (ICLR)*, 2020.

DeLiang Wang. The time dimension for scene analysis. *IEEE Transactions on Neural Networks*, 16(6):1401–1426, 2005.

Suyuchen Wang, Ivan Kobyzev, Peng Lu, Mehdi Rezagholizadeh, and Bang Liu. Resonance rope: Improving context length generalization of large language models. In *Proc. Association for Computational Linguistics (ACL)*, 2024.

Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. In *Int. Conf. on Learning Representations (ICLR)*, 2018.

Alex Warstadt, Alicia Parrish, Haokun Liu, Anhad Mohananey, Wei Peng, Sheng-Fu Wang, and Samuel R. Bowman. BLiMP: The benchmark of linguistic minimal pairs for English. *Transactions of the Association for Computational Linguistics (TACL)*, 8:377–392, 2020.

Nicholas Watters, Loic Matthey, Christopher P Burgess, and Alexander Lerchner. Spatial broadcast decoder: A simple architecture for learning disentangled representations in VAEs. In *Learning from Limited Labeled Data (LLD) Workshop, ICLR*, 2019.

Max Wertheimer. Untersuchungen zur lehre von der gestalt. ii. *Psychologische Forschung*, 4(1):301–350, 1923.

Yi-Fu Wu, Klaus Greff, Gamaleldin Fathy Elsayed, Michael Curtis Mozer, Thomas Kipf, and Sjoerd van Steenkiste. Inverted-attention transformers can learn object representations: Insights from slot attention. In *UniReps: the First Workshop on Unifying Representations in Neural Models, NeurIPS*, 2023.

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.

Kai Yuanqing Xiao, Logan Engstrom, Andrew Ilyas, and Aleksander Madry. Noise or signal: The role of image backgrounds in object recognition. In *Int. Conf. on Learning Representations (ICLR)*, 2021.

L. Yaeger. Neural networks provide robust character recognition for newton pdas. *IEEE Expert*, 11(4):10–11, 1996.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Jaesik Yoon, Yi-Fu Wu, Heechul Bae, and Sungjin Ahn. An investigation into pre-training object-centric representations for reinforcement learning. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2023.

J. Zacks, B. Tversky, and G. Iyer. Perceiving, remembering, and communicating structure in events. *Journal of experimental psychology. General*, 130 1: 29–58, 2001.

Andrii Zadaianchuk, Maximilian Seitzer, and Georg Martius. Self-supervised visual reinforcement learning with object-centric representations. In *Int. Conf. on Learning Representations (ICLR)*, 2021.

Andrii Zadaianchuk, Maximilian Seitzer, and Georg Martius. Object-centric learning for real-world videos by predicting temporal feature similarities. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 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 *Int. Conf. on Learning Representations (ICLR)*, 2019.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proc. Association for Computational Linguistics (ACL)*, 2019.

Richard S Zemel, Marlene Behrmann, Michael C Mozer, and Daphne Bavelier. Experience-dependent perceptual grouping and object-based attention. *Journal of Experimental Psychology: Human Perception and Performance*, 28(1):202, 2002.

Catherine Zeng, Jordan Docter, Alexander Amini, Igor Gilitschenski, Ramin Hasani, and Daniela Rus. Dreaming with transformers. In *AAAI Workshop on Reinforcement Learning in Games*, 2022.

Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018a.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Yuting Zhang, Yijie Guo, Yixin Jin, Yijun Luo, Zhiyuan He, and Honglak Lee. Unsupervised discovery of object landmarks as structural representations. In *Proc. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018b.

Andrey Zhmoginov, Ian Fischer, and Mark Sandler. Information-bottleneck approach to salient region discovery. *ICML Workshop on Self-Supervised Learning*, 2019.

Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. In *arXiv preprint arXiv:1904.07850*, 2019.

Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable DETR: Deformable transformers for end-to-end object detection. In *Int. Conf. on Learning Representations (ICLR)*, 2021.