Understanding and Identifying Challenges in Design of Safety-Critical AI Systems

by

Ekdeep Singh Lubana

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical and Computer Engineering)
in the University of Michigan
2024

Doctoral Committee:

Associate Professor Robert P. Dick, Chair
Assistant Professor Dallas Card
Assistant Professor David Krueger, University of Cambridge
Assistant Professor Andrew Owens
Dr. Hidenori Tanaka, Harvard University & NTT Research

Ekdeep Singh Lubana
eslubana@umich.edu
ORCID iD: 0009-0004-9849-7859

© Ekdeep Singh Lubana 2024

DEDICATION

To the martyrs and founders of my country, whose sacrifices and vision have given me the freedom to stand tall and express myself without fear. To my father, whose tireless dedication ensured his children can have a better life. To my mother, whose unwavering support enabled me to forge my own path.

ACKNOWLEDGEMENTS

It takes a village to raise a child, and indeed a village stood behind me as I was growing as a researcher over the course of my PhD. The list of people I am grateful to is much longer than my episodic memory will allow me to recall, and I am confident I will forget to acknowledge a few crucial people below—please forgive me and know I am forever indebted to you! To begin, I can trace some of my most crucial acknowledgments to three cold emails.

The first email was to my advisor at Michigan, i.e., Prof. Robert Dick. I reached out to Robert 7 years ago, while I was still a sophomore at IIT Roorkee and a teenager with lofty goals. He took me under his wing, taught me from the ground up how research is done, and gave me the flexibility—rather, encouragement—to explore essentially any topic I was interested in. This encouragement during my undergrad research was precisely the reason I joined his group as a PhD student. As I grew and established my research agenda, Robert walked alongside me and was willing to explore any new topic or tangent I would veer off on—tangents that neither he nor I had much background in. I am grateful for all the discussions we had, all the textbooks he read with me, and the extremely open-ended conversations he was willing to engage with me in. Beyond flexibility in research directions, he gave me the flexibility in how I did research itself. He encouraged me to take any opportunity that came my way and made sure I had the necessary support to make the most out of it. Robert is also one of the most principled people I know, and, as I started to get interested in research on design of safe AI systems, he was my most ardent supporter.

A similar story came to play as I started working with Dr. Hidenori Tanaka, my advisor at Harvard. This was the second cold email—sent to discuss a research direction we were both working on. Hidenori took me under his wing and, as I started to move away from my systems background to more theory and understanding-driven deep learning research, he invited me to spend the latter half of my PhD working with him at Harvard. Hidenori's enthusiasm is beyond infectious: his excitement about rather arbitrary ideas I came up with gave me the confidence that no idea is ever too wild; if I have an intuition that I deem worthy of exploration, he taught me that I should give it a shot. Further, Hidenori's own unconventional background as a physicist working on neuroscience and machine learning introduced me to entirely new perspectives on research that I had never explored before.

He introduced me to infinite new topics I was unlikely to stumble upon myself. I started to appreciate and pursue more hypothesis-driven research due to him, and a bulk of that work is now summarized in this dissertation. His willingness to engage in extremely long discussions debugging results (at extremely odd hours, I might add), putting in all-nighters before a conference deadline, and willingness to go above and beyond to ensure I have all resources necessary to achieve my goals barely scratches the surface of what I am indebted to him for. I am also grateful to the broader team at Physics and Informatics Lab, NTT Research—especially Yoshihisa Yamamoto, Kazuhiro Gomi, and Hideaki Ozawa—who funded me, and the leadership at Center for Brain Science, Harvard—especially, Venkatesh Murthy, Kenneth Blum, and Talia Konkle—who helped organize my stay at Harvard to work with Hidenori.

During the last two years, I have also deemed Prof. David Krueger as an unofficial advisor to me. The third crucial email I sent was to him: we connected at ICML 2022 and found that we had independently come up with (almost literally) the same research directions. We decided to collaborate and, since then, have been collaborating. Though I had been wanting to work on the topic for a while, discussions with him and members of his group convinced me that science of deep learning can play a crucial role in design of safe AI systems. The work I did with David constitutes the last part of this dissertation; though, I note that our conversations have found their way to the other problems I explored, including ones presented in the first two parts of this dissertation. Beyond helping me identify important research directions, I am also indebted to David for introducing me to people who are now some of my closest collaborators and arguably my research support system.

Related to cold emails, I would also like to thank my committee members Prof. Dallas Card and Prof. Andrew Owens for responding to my cold email about joining my dissertation committee, and providing extremely useful and detailed feedback that helped improve the writeup.

When it comes to on-ground work, I have been exceedingly fortunate to have some of the most smart, enthusiastic, and hard-working peers. I am especially grateful to Robert Kirk, Eric Bigelow, Samyak Jain, Maya Okawa, Core Francisco Park, Kyogo Kawaguchi, Usman Anwar, Gautam Reddy, Andrew Lee, Pulkit Gopalani, Kento Nishi, Itamar Pres, Abhinav Menon, Leni Shor, Laura Ruis, Felix Sosa, Puja Trivedi, Ziyin Liu, Rahul Ramesh, Mikail Khona, Chi Ian Tang, Lorenzo Noci, and Jayaraman Thiagarajan. Our collaborations were some of the most exciting work I participated in, and I keenly look forward to the work we will do together in future. More importantly, I thank you all for indulging me, and deeming my ideas worthy of your time. I especially want to thank Rob, for my conversations with him defined a significant portion of my research agenda—some of my most "out there" ideas first undergo what I think of as "Rob-filtering" to ensure I am working on projects that are

meaningful. I also thank Eric, who has been one of my closest friends and collaborators during my PhD, and the amount of time we spent talking research and whiteboarding is perhaps incalculable. Eric served as my guide into the broader landscape of cognitive science, and introduced me to several tools that I had never encountered before—tools that now serve as my go-to protocols for analyzing neural networks. I also thank Samyak, Maya, Core, Andrew, Kento, Itamar, Leni, Pulkit, Rahul, and Mikail for helping me bring to life several research directions I wanted to explore. I am especially grateful to Samyak and Maya for the huge amounts of contributions they made to parts of this dissertation; at a time when my symptoms from Carpal tunnel had reached their worst and it became difficult for me to perform experiments myself, their assistance proved crucial to seeing several projects reach the finish line. Gautam has been yet another unofficial advisor to me, especially during the last leg of my PhD, and I am looking forward to the outputs our collaborations will soon produce. I am also thankful to my collaborators' advisors, who allowed us to operate seamlessly and were willing to let me steer their students towards directions and research methodologies I was keen on exploring at that moment. I especially thank Tomer Ullman, Sam Gershman, Tim Rocktaschel, Edward Grefenstette, Puneet Dokania, Amartya Sanyal, Philip Torr, and Danai Koutra. I would also like to thank the following people for conversations that were consequential to the work I did during my PhD: Anna Golubeva, Vasudev Shyam, Nikhil Vyas, Naomi Saphra, Ben Edelman, Lauro Langosco, Usman Anwar, Dima Krasheninnikov, Bruno Mlodozeniec, Nikunj Saunshi, Suraj Srinivas, Yamini Bansal, Sadhika Malladi, Samuel Marks, Xander Davies, Cristian Trout, Max Nadeau, Gabe Wu, Uzay Girit, and the broader AISST team, Alan Chan, Stephen Casper, Sadhika Malladi, Jacob Zavatone-Veth, Demba Ba, Martin Wattenberg, Kiah Hardcastle, Jonathan Frankle, the ML foundations group at Harvard (especially Sham Kakade and Boaz Barak), Yash Sharma, Luigi Gresele, Peter Hase, Alex Lew, and Tan Zhi Xuan. I especially highlight Anna, Vasu, Nikhil, Naomi, Ben, Nikunj, and Usman in this list, since their feedback was at the forefront of improving the work presented in this dissertation.

To enable the work that I did, there were several tools I had to pick up. Fortunately, I had some of the most amazing teachers. I especially thank Profs. Jeff Fessler, Sandeep Pradhan, David Neuhoff, Andrew Owens, Liz Spelke, and Sylvia Lu. I was also fortunate to do exciting internships that allowed me to learn more empirical tools and played a non-trivial role in helping me define my research agenda. I am especially grateful to my advisors at these internships, including Akhil Mathur, Fahim Kawsar, Pim de Haan, Johann Brehmer, and Taco Cohen. I also thank my co-interns, who created a beautiful environment to work in, and to other research scientists at these places who engaged in detailed technical conversations that influenced my ideas on AI.

A PhD is a long endeavor and one needs a strong support system to navigate the involved journey. To this end, I first want to thank the administrative staff who ensured all the ways I was bending the system worked out—this includes especially Kristen Thornton, the most compassionate grad coordinator there ever has been. I also thank several friends I had during the last five years: Ariba Javed, Anna Golubeva, Vasudev Shyam, Rohit Rothe, Nilesh Kulkarni, Mohammad El Banani, Karan Desai, Richard Higgins, Yakshita Malhotra, Rupesh, Hansal Shah, Chris Rockwell, Fatih Dinc, Dandan Shan, Jiong Zhu, Santiago Castro, Caleb Belth, Andrew Larson, Fenil Doshi, Srijani Saha, Nick Blauch, Binxu Wang, and JJ Andrade. I especially thank Ariba for always being there, and reminding me it was siblings day; Anna and Vasu for their effort into bringing me out of my shell as a confident researcher, and consistently checking in that I am not burning myself out; Nilesh and Caleb for spending long nights doing information theory homework together; Santi for connecting me with Andrew Lee, who has become one of my closest collaborators; Mo for being extremely accommodating and always becoming my summer sublet, regardless of which state I was in; KD for the random encounters at Songbird, leading to exciting conversations on the latest computer vision research; and Richard for motivating me to think about causal representation learning, which served as a turning point in my research. I especially thank the basketball crew, who kept me sane—including Rohit Rothe, Yakshita, Johan, Akshay, Revanth, Hari, Maryknoll, Srihari, and (at times) Nilesh Kulkarni.

To get to this stage, there has been a large crew cheering and motivating me for a long time now. I want to thank the multitude of friends I made at IIT Roorkee, especially Vinayak Aggarwal, Shashwat Kumar, Nitin Sethi, Sumit Kumar Yadav, Vinam Arora, Anish Chhabra, Ashutosh Parija, Vipul Gupta, Sai Sri Vathsa, Rohith ASRK, Gourav Dhar, Himanshu Dua, Satyam Yadav, the basketball and WatchOut teams, and the awesome mentors I had, including Raghav Chawla, Jay Desai, Aman Mehta, Kartik Patel, Sidakpal Singh, Abhishek Gaur, Ariba Javed, Nishtha Sharma, Nipun Sawhney, and Ankit Bagaria. I especially thank Vinayak, Shashwat, and Nitin for, perhaps unintentionally, motivating me to pursue a path of my own during undergrad. I also thank my teachers, especially Saumik Bhattacharya, Vinod Pankajakshan, and Pyari Mohan Pradhan.

Going back a bit further, I note that my intentions to pursue a research career were set in second grade, when my father first told me what a scientist is. Hailing from a non-technical background, however, my friends and teachers from childhood have played a huge role in getting me here. To this end, I would like to thank my friends who indulged me and my ideas, and motivated me to keep pursuing them. This includes Himanshu Matharu, Harleen Singh, Avtaar Gill, Bhanu Goyal, Ashish Bansal, Bhavya Bahl, Ratinderjit Singh Bajwa, Savitoj Singh, Rohan Garg, Jaskirat Singh, Deepinder Singh Virk, and Surdeep Singh. I

made some of my best friends and study-mates while preparing for the IIT examination and the Indian National Mathematics olympiad. In particular, I thank Lovish Madaan, Kashish Bansal, Anish Chhabra, Oshin Bhatia, Prabhav Kumar Gaur, Robin Bansal, Roopanjit Singh, Varun Singla, Sarthak Khurana, Lakshay Gaba, Mohit Jindal, Varun Goyal, Arshdeep Singh, Himanshu Gupta, Tanav Jain, Shubham Goel, and many more. I also thank my teachers, including Renu Garg, Rishi Singla, Puneet Chhabra, Paroksh, Harpreet Singh, Sahil, Anand, and Vandana Ralhan, who played a huge role in me purusing the trajectory that I did. In particular, I highlight Rishi Singla, who was my most consistent and ardent supporter—the sheer thrill of attending his classes and absorbing all his excitement helped me manage the competition and stress of JEE preparation. I also highlight Puneet Chhabra, who went out of his way and became my mentor to help me prepare for Maths olympiad—my joy for science in general was inherited to a large extent from him. Finally, my academic turning point was brought about by Renu Garg, to whom I owe arguably more than any other person.

There are four more people I must thank. First, my father. He is perhaps the most hard-working person I know of. Hailing from an extremely humble background, his stories still baffle me, for I cannot comprehend how an individual with so few resources can overcome essentially any obstacle ever thrown his way. He is a true hero to me and I aspire to be half as kind and amazing a person as he is. As I noted above, my father told me what a scientist is when I was six years old, and for whatever reason that moment stuck with me and led me to where I am now. Second, my mother, who has been my source of hope, courage, and love throughout my life. While my father steered me towards a path of stability, my mother motivated me to do the exact opposite: do what I feel happy about, and not worry about the returns. Was it not for her support, I would not have achieved anything. Third, I must thank my sister, for she being elder to me inherited my parents' aspirations, allowing me to pursue my path. Fourth, I thank my partner, Puja Trivedi. It is difficult to overstate the source of joy and happiness she has been over the course of my PhD. She has supported me through all the (mis)steps I have taken, ensured that I stay on top of my game, and been the beacon I keep following. I am extremely grateful for the stability she brought to my tumultuous mind—one that would not stay calm and wanted to continuously be in pursuit of some new aspiration.

Finally, as noted in the dedication of this dissertation, I owe everything to my people—ones who came before me and ensured I could proudly claim to be a free individual with independent thoughts. From the more well known—like Sardar Bhagat Singh, Rabindranath Tagore, Jawaharlal Nehru, Mahatama Gandhi, Lala Lajpat Rai, Mohammad Ali Jinnah—to arguably less known ones (at this point)—including Rajguru, Sukhdev, Chandrashekhar

Azad, and Kartar Singh Sarabha—I owe you all my motivation to pursue the research direction I am invested in now: developing a future grounded in equality and equity.

Financial support for my work at University of Michigan was provided by NSF Grants CNS-2008151 and CNS-2211509 and at Harvard by CBS-NTT program on Physics of Intelligence.

TABLE OF CONTENTS

DEDICATION ii

ACKNOWLEDGEMENTS iii

LIST OF FIGURES xii

LIST OF TABLES xlix

LIST OF APPENDICES li

ABSTRACT liii

CHAPTER

1 Introduction **1**
- 1.1 Motivation 1
- 1.2 Contributions 3
- 1.3 Approach: Model Experimental Systems 5
- 1.4 Timeline 6

Part 1: Unpredictable Capabilities Render Exhaustive Characterization Difficult **7**

2 The Multiplicative Emergence Hypothesis **10**
- 2.1 Introduction 10
- 2.2 Related Work 13
- 2.3 Concept Graph: A Minimalistic Framework for Compositionality 14
  - 2.3.1 Experimental and Evaluation Setup 17
- 2.4 Multiplicative Emergence of Compositional Abilities 18
- 2.5 Challenges for Compositional Generalization 22
- 2.6 Additional Experimental Results with Natural Data 25
- 2.7 Discussion 25
- 2.8 Follow-Up Work 26

3 Percolation as a Model of Emergence **28**
- 3.1 Introduction 28

3.2 Related work . . . 31
3.3 A Phenomenological Definition of Emergence . . . 32
3.4 Formal Languages as a Model Experimental System to Study Emergence . . . 33
3.5 Learning Tasks and Experimental Setup . . . 35
3.6 Results: Emergent Capabilities in Formal Language Learning . . . 37
3.6.1 Phases of Language and Capabilities Acquisition . . . 37
3.6.2 Effect of Number of Descriptive Properties on Language Acquisition . . . 41
3.7 A Percolation Model of Emergence . . . 41
3.7.1 Percolation Transition on Descriptive Constraints . . . 43
3.8 Discussion . . . 46

Part 2: Latent Capabilities and Underspecification as Hurdles to Evaluations 47

4 Competence Emerges Before Performance 50
4.1 Introduction . . . 50
4.2 Related Work . . . 52
4.3 Concept Space: A Framework for Analyzing Concept Learning Dynamics . . . 53
4.3.1 Experimental and Evaluation Setup . . . 55
4.4 Learning Dynamics in Concept Space . . . 56
4.4.1 Concept Signal Determines Learning Speed . . . 56
4.4.2 Concept Signal Governs Generalization Dynamics . . . 56
4.4.3 Towards a Landscape Theory of Learning Dynamics . . . 58
4.4.4 Sudden Transitions in Concept Learning Dynamics . . . 59
4.5 Effects of Underspecification on Learning Dynamics . . . 60
4.6 Additional Results on Natural Data . . . 63
4.7 Discussion . . . 64

5 In-Context Learning Dynamics via Random Binary Sequences 66
5.1 Introduction . . . 66
5.2 In-Context Learning as Hypothesis Selection . . . 68
5.3 Algorithmic and Subjective Randomness . . . 70
5.4 Experiments . . . 71
5.5 Results . . . 76
5.5.1 Subjectively Random Sequence Generation . . . 76
5.5.2 Sub-Sequence Memorization and Complexity . . . 77
5.5.3 Distinguishing Formal Languages from Randomness . . . 78
5.6 Discussion . . . 79

Part 3: Limitations of Fine-tuning for Removing Model Vulnerabilities 79

6 Mechanistic Mode Connectivity 82

6.1 Introduction . . . 82
6.2 Preliminaries: Mode Connectivity . . . 84
6.3 Defining Mechanistic Similarity . . . 85
6.4 Setup for a Mechanistic Evaluation . . . 88
6.5 Mechanistic Analysis of Mode Connectivity . . . 89
6.6 Altering a Model's Mechanisms Efficiently . . . 93
6.6.1 Connectivity-Based Fine-Tuning (CBFT) . . . 93
6.7 Related Work . . . 95
6.8 Discussion . . . 97
6.9 Future Work . . . 97

7 Mechanistic Analysis of Fine-Tuning on Procedurally Defined Tasks 99
7.1 Introduction . . . 99
7.2 Related Work . . . 101
7.3 Defining our notion of capabilities . . . 102
7.4 Building Capable Models: Tracr and PCFGs . . . 104
7.5 Results and Discussion . . . 106
7.5.1 Behavioral assessment of fine-tuning . . . 107
7.5.2 Pruning / Probing fine-tuned models indicates learning of wrapped capabilities . . . 108
7.5.3 `reFT` enables "revival" of pretraining capabilities . . . 109
7.5.4 Attention map visualizations further corroborate the wrappers hypothesis . . . 110
7.5.5 Validating our Hypotheses on TinyStories . . . 111
7.6 Discussion . . . 112
7.7 Follow-Up Work . . . 113

8 Conclusion, Limitations, and Future Work 115

APPENDICES . . . 119

BIBLIOGRAPHY . . . 340

LIST OF FIGURES

FIGURE

1.1 **Towards design of safety critical systems.** Risk regulation frameworks for design of safety-critical systems emphasize the need for exhaustive characterization to identify and fix *expected* vulnerabilities in a safety-critical system before deployment. To the extent a vulnerability is an inherent behavior of the system, it can be added to guidelines of safe use. However, user interactions or use in novel scenarios can reveal further vulnerabilities during deployment of a system, requiring rigorous auditing and red-teaming to identify such novel vulnerabilities. Intervention protocols should be designed to address and mitigate such vulnerabilities. 2

1.2 **A Model Experimental Systems Approach to Understanding Complex Systems.** Consider a complex system like the human brain, which is often the target of study in natural sciences. Directly analyzing the brain can often be infeasible, so researchers focus their experiments on *model systems*, i.e., a substitute for the target system that is expected to solve the task of interest in a similar way as the target system. Analysis of the model system leads to derivation of a hypothesis for how the target system might be solving the task, hence allowing design of precise experiments (e.g., behavioral tests when analyzing the brain) that are meant to accept or refute just that hypothesis. 6

2.1 **(Lack of) Compositionality in text-conditioned diffusion models.** Images generated using Stable Diffusion v2.1 [1]. We see diffusion models conditioned on text descriptions of the concepts in an image often allow generation of novel concepts that are absent from the training data, indicating an ability to compose learned concepts and generalize out-of-distribution (lizard and goldfish panels). However, the model sometimes unpredictably fails to compose concepts it has learned, i.e., compositional capabilities depend on the specific concepts being composed. For ex., generating a panda in the above figure is difficult for the model, likely because it has not been exposed to different colors of pandas. The model instead alters the background or lighting to change the color. 11

2.2 **Compositionality in a minimalistic conditioned generation task.** (a) We train diffusion models on pairs of images and tuples, where the tuples denote the values of the *concepts* represented in the image (e.g., values of color and shape in the figure). (b) Samples between which only a single tuple element differs simplify the learning of a *capability* to recognize and alter the concept distinguishing the images. (c) To test the existence of such capabilities of a trained model, we ask the model to generate images corresponding to novel tuples that are out-of-distribution, hence requiring compositional generalization. 12

2.3 **Concept graphs.** We organize our study using a simple but expressive framework called *concept graphs*. The basis of a concept graph is a set of primitives called *concept variables* (e.g., `shape` , `color` , etc.). A subset of these variables is instantiated with specific values to yield a *concept class*, e.g., $\{\texttt{shape} = 0, \texttt{size} = 0, \texttt{color} = 1\}$ implies a small, blue circle. This is akin to defining a broad set of objects that share some common properties, e.g., all lizards in Fig. 2.1 belong to the same species, despite their diverse colors. A specific *object* in this class is instantiated by fixing the remaining variables, e.g., small, blue circles at different locations. Each concept class corresponds to a graph node, where nodes are connected if their concept classes differ by a *concept distance* of 1. 14

2.4 **An in-the-wild Concept Graph.** Analyzing CLIP embeddings via a 2D PCA projection demonstrates models naturally organize representations for sufficiently independent concepts—in this case, `object`, `color`, and `background color`—akin to a concept graph (see Fig. 2.3). 16

2.5 **Capabilities and compositionality in a concept graph.** Consider a lattice representation of a concept graph corresponding to four concept variables. Blue nodes denote classes represented in the training data; a model can either memorize data from these classes or learn capabilities to transform samples from one class to another. If it learns capabilities, it can compose them with data-generating process of samples from a concept class in the training data and produce samples that are entirely out-of-distribution, denoted as pink and green nodes. 17

2.6 **Concept distance from the training set governs the order in which compositional capabilities emerge.** (a) Concept graph (cube) depicting training data points (blue nodes) and concept distances for test data points, where pink nodes represent distance = 1, and darkpink nodes represent distance = 2. Each trajectory represents the learning dynamics of generated images given a tuple prompt. Each trajectory represents the learning dynamics of generated images based on each tuple prompt. During every training epoch, 50 images are generated and binary classification is performed to predict each concept, including color, shape, and size. (b) Compositional generalization happens in sequence, starting with concept distance = 1 and progressing to concept distance = 2. The x-axis represents the number of epochs, and the y-axis represents the progress of compositional generalization. (c) Images generated as a function of time clearly show a sudden emergence of the capability to change the color of small, blue, triangles. 19

2.7 **Multiplicative influence of individual capabilities elicits an "emergence" of compositionality.** (a) Accuracy of producing samples from the concept class {111, (triangle, blue, small)}, which has a concept distance of 2 from the training data. A multiplicative measure (solid line) has a score of 1 when all concept variables of shape, color, and size are correctly predicted. Conversely, an additive measure (dashed line) independently relates each concept variable prediction accuracy to the compositional accuracy, deceptively suggesting smooth progress. (b) Learning dynamics of predicting each of the three concept variables: shape (blue), color (orange), and size (green). 20

2.8 **Toy Model of Compositional Emergence.** (a) Probability of learning a compositional capability at a concept distance n w.r.t. atomic abilities as a function of time t. (b) Critical optimization time steps t^* required for the above probability to reach a threshold P^* as a function of concept distance n. 21

2.9 **How does the frequency of data samples impact the learning of capabilities?** We systematically control the frequency of a specific concept class in the training dataset and observe how it affects the model's learning of capabilities. (a) Capability to alter colors is quickly learned after introducing approx. 10 samples with a concept class of large blue circle, 001. (b) In contrast, a critical threshold (marked with dotted vertical lines) exists for learning a capability to alter the shape: as we gradually introduce samples, with a concept class of large red triangle, 100. 23

2.10 **Fine-tuning is not a remedy for misgeneralization in an adversarial task.** (a) The model struggles to generalize when the training setup is configured adversarially. As shown in the plot, the model incorrectly produces triangles (instead of circles) for both 011 and 010. (b) The model faces difficulty learning new concepts through fine-tuning. We add node 101 (triangle, large, blue) to the dataset and attempt to fine-tune the model. However, even with a large learning rate equal to the one used for training, the model fails to learn the capability to alter object size. 24

2.11 **Investigating our claims on a naturalistic data setting (CelebA).** (a) Concept graph (cube) depicting training data points (blue nodes) and concept distances for test data points for CelebA dataset. (b) Compositional generalization happens in sequence, starting with concept distance 1 and progressing to concept distance 2. The quick learning of node `011` and the 50% initial accuracy of node `101` is a consequence of the biased dataset (see Fig. 2.7 for a similar effect in our synthetic setup where there is a color bias). (c) Gender prediction accuracy based on generated samples at each step during training. We observe that the ability to generate the underrepresented attribute (e.g., gender) is slightly delayed as concept distance increases. 26

3.1 **Emergence as phases of learning.** Emergence is a well-characterized phenomenon in natural sciences [2, 3, 4] and deeply entangled with the notion of phase changes, i.e., when change in some control variable (e.g., temperature) yields systematic changes in a system's underlying structure (e.g., formation of hexagonal configurations in a crystal) and simultaneously affects several of its properties. We argue for a similar characterization of emergence in machine learning: identifying systematic changes in a model's behavior that influence its downstream abilities and lead to sudden performance improvements. For example, learning a language's syntax will affect all downstream capabilities where coherent, grammatically correct generations are necessary. 29

3.2 **Grammar and type constraints to define our formal language.** (a) We use a PCFG to define our language's grammar (shown rules are examples; see App. B.4.3 for precise details). The grammar's terminals are parts-of-speech from English and yield symbolic sentences that can be populated by tokens from the language's vocabulary. (b) Akin to natural language, wherein properties of an entity constrain sentences seen in a dataset corresponding to that entity, we define constraints (called type constraints) on our language that restrict which tokens can be seen together in a sentence. These constraints map entities to descriptive or relative properties, hence restricting which descriptive adjectives and verbs are valid for an entity. (c) Once a symbolic sentence is sampled from the grammar, we populate it with tokens from the language while respecting the type constraints. Training on string from this language in fact shows that the model deems sentences that do not respect type constraints to be extremely unlikely (see App. B.4.2). 34

3.3 **Task definitions.** Our model is trained and evaluated on three types of tasks. (i) Free generation: the model generates sentences with correct grammar. (ii) Unscrambling: the model is provided with a set of words and must reorder them to form valid sentences. (iii) Conditional generation: model is given a set of entities or properties and must generate valid sentences using them. Note that examples in the figure are merely indicative. See App. B.1.1 for details. 36

3.4 **Learning of structures drives emergent capabilities.** For a detailed discussion, see main text. **(a) Grammaticality and Type Check evaluations** as a function of iterations or data (iterations×batch-size). We see phases in the learning dynamics corresponding to *emergent acquisition of structures underlying our language*: grammar (black), relative type constraints (pink), and descriptive type constraints (green), shaded gray, pink, and green respectively. **(b, c) Performance on Unscrambling and Conditional Generation.** After a slight delay from phase boundaries, we see sudden improvements in the performance of individual tasks. **(d) Learning curves.** Loss also shows sudden changes at phase boundaries corresponding to the acquisition of structures. **(e) Performance on descriptive/relative sentences.** Decomposing by sentence type, we find a sublinear growth in descriptive type checks drives performance boost on descriptive sentences for the unscrambling task. 38

3.5 **Effect of scaling number of descriptive properties.** Scaling the number of descriptive properties in our language (see legend), we find relative type checks and unscrambling performance for relative sentences are essentially unaffected by the number of properties. Meanwhile, both descriptive type checks and unscrambling performance for descriptive sentences show a change in performance and delay in transition points. Despite these effects, we find the geometry of performance curves is extremely consistent; for descriptive type checks, this geometry indicates a memorization to generalization picture. 40

3.6 **Casting the ability to compose unseen concepts (e.g., entities and properties) as percolation on a bipartite graph.** (a) When only a fraction of the concept classes are included in the dataset and the edge density is low, nodes (e.g., entities and properties) form many disconnected clusters, indicated by different colors (left). As more concept classes are added (dashed edges) to the bipartite graph, the small clusters begin to merge (middle). With a sufficient number of edges, a macroscopic number of nodes become connected, forming a single cluster (right). (b) Our formalism establishes this transition as a second-order phase transition, where the size of the largest cluster increases non-linearly as the fraction of connected node pairs is scaled. Shown curves are from simulations on bipartite graphs with the number of nodes equal to the number of entities and properties in our formal language experiments (see Sec. 3.5 or App. E.2). 44

3.7 **Scaling of point of emergence matches with our theory.** We replot the result from Fig. 3.5 by rescaling the x-axis with a power of the number of descriptive properties; See App. B.5.1 for further discussion on the intuition underlying this visualization. For Average probability of generating a valid descriptive property given some entity in context under free generation and descriptive type check accuracy, we see a 0.5 exponent scaling yields a collapse of the transition point—this matches the toy model of percolation posited in Sec. 3.7.1. We also see a very clear scaling of the point where the ability to unscramble descriptive sentences starts to emerge, but with an exponent of 1.5. 45

4.1 **Concept Learning Geometry underlies emergence.** (a) Top: A multimodal model learns to generate concepts in the order of "astronaut", "horse", and finally "riding" as it scales up (adapted from Yu et al.[5]). Middle: "blue square apple" is generated in the order of "apple", "square", and "blue" (adapted from Li et al.[6]). Bottom: Despite its simplicity, our model trained on synthetic data shows *concept learning dynamics* where it first learns "shape" and then "color". (b) Concept space is an abstract coordinate space where individual axes correspond to different concepts and a given point corresponds to a "concept class", i.e., a predefined collection of concepts (e.g., `large blue circles` on bottom left corner). Traversal along axes of the concept space yield change in a specific property of the sample (e.g., going from `large blue circle` to `large red circle` along `object color` axis). Trajectories show a model's dynamics in concept space for learning to generate classes shown in-distribution (blue nodes) versus out of distribution (pink / red nodes). As we show, dynamics in concept space are highly interpretable, enabling precise comments on which concepts the model learns first, why, and what order it follows. (c) Measuring how accurately a model generates samples from a given concept class, showing an order of concept learning: first background color is learned, then size, and then object color. 51

4.2 **Concept spaces with different concept values see different concept signal.** Consider a concept space comprised of concepts `size` and `color`. (Left) The `color` separation between the classes is stronger than the `size` separation, resulting in a stronger concept signal in the `color` dimension. (right) The `size` separation between the classes is stronger, thus resulting in a stronger signal for `size`. 55

4.3 **Concept signal determines learning speed.** The speed of concept learning as an inverse of the time in gradient steps when the separation in color (left) and size (right) between different classes increases. Concept learning is faster when pixel differences among concept class and hence concepts are larger. 56

4.4 **Concept signal governs generalization dynamics.** (a) Learning dynamics in the concept space for in-distribution concept class `00` (bottom left). (b) Learning dynamics for out-of-distribution (OOD) concept class `11` (top right). We plot the accuracy for `color` on the x-axis and `size` on the y-axis. The [0,1) normalized `color` concept signal level is color coded. Two trajectories for `01` and `10` are shown to illustrate *concept memorization*. 57

4.5 **A Phenomenological Model of Learning Dynamics in Concept Space.** Using Eq. 4.1, we simulate the learning trajectory for concept class `00` in panel (a) and the OOD class `11` in panel (b). Initially, target values are set at $(0, 1)$ or $(1, 0)$ based on the concept signal strengths for color or size, respectively. As the model progressively learns each concept, the target values gradually shift towards $(1, 1)$. This simple toy model accurately reproduces the observed curves in Fig. 4.3(c), which arise from *concept memorization*. . 58

4.6 **Emergence of hidden capabilities.** We plot accuracy as a function of gradient steps for five different runs, using three different protocols for prompting the model to generate outputs for OOD concept classes. (a) The baseline, naive prompting protocol; (b) linear latent intervention, applied in the activation space; and (c) overprompting, akin to an intervention on the input space. 60
4.8 **Underspecification and Concept Learning.** (a) The state-of-the-art generative models [7] erroneously produces a `red strawberry` (top right corner) for the prompt "`yellow strawberry`". (b) Without underspecification in the training data, a model F accurately learns the concepts of shape and color, successfully generalizes to the unseen node `blue triangle` (leftmost). As masks are applied to the word `red` for the prompt `red triangle`, concept signal for `triangle` increasingly starts to correlate with the concept `red`. This causes the output images to change from blue to purple as the level of masking increases (panels left to right). Eventually, the color dimension for `triangle` collapses, biasing the model towards generating solely `red triangles` (rightmost). 61
4.7 **Underspecification delays out-of-distribution (OOD) generalization.** The number of gradient steps required to reach accuracy above 0.8, as the percentage of masked prompts increases. A higher proportion of masked prompts slows down the speed of concept learning. 61
4.9 **Underspecification hinders out-of-distribution (OOD) generalization.** (a) The learning dynamics with varying levels of prompt masking, from 0% to 100%, and the generated images. At 0% masking (top right image), the model correctly produces an image of `blue triangle` from the prompt "`blue triangle`." As the masking increases (from right to left), the images gradually shift towards the incorrect color, `red`. (b) The simulation of the learning dynamics under underspecification in concept space based on Eq. 4.2. Our toy model replicates a trained network's learning dynamics. . . 62
4.10 **Underspecification and Hidden Capabilities.** We use the over-prompting protocol from Sec. 4.4.4 to assess if the capabilities to enable OOD generalization are learned before naive input prompting. (a) Accuracy for OOD generalization across five different seed runs under naive prompting; (b) Accuracy under over-prompting, with a fixed masking percentage of 50%. . 63
4.11 **Validating our findings on CelebA.** (a) Analyzing the learning dynamics for concepts `Gender` and `With Hat`, we find the training run gets stuck in concept memorization: converging near (likely) the closest class of `Females`, `With Hat`. (b) Performing latent interventions however clearly demonstrates that the model is indeed *capable* of generalizing out-of-distribution, but is not *performant* when evaluated via naive prompting. 64

5.1 **Overview of our modeling framework.** Given a pre-trained Large Language Model, we systematically vary input context prompts, in this case $x = (001)^n$ with $n = \{2, 4\}$ (`001001`, `001001001001`). LLM outputs y (Bottom) significantly as a function of x (Top), based on some unknown latent concept space embedded in the LLM. We model a subset of the algorithms embedded in the LLM's latent concept space $\mathcal{H}$ that are invoked by ICL (Middle), to predict the LLM outputs y. With very little context (Left), GPT-3.5+ generates subjectively random sequences, whereas with enough context matching a simple formal language (x = `001001001001` = $(\texttt{001})^4$), it begins deterministically repeating patterns in x. 67

5.2 **Sharp changes in the predictive distribution $p(y|x)$ suggest hypothesis selection** (Left) Here we show an illustrative example of a simple Bayesian model with two hypotheses in $\mathcal{H}$: a random Bernoulli hypothesis, and a deterministic Non-Random hypothesis for the concept $(\texttt{01})^n$. When hypothesis posteriors $p(h|x)$ cross as a function of additional data, i.e., larger $|x|$, the predictive distribution $p(y|x)$ for hypothesis selection suddenly transitions from one pattern of behavior to another (Right). With hypothesis averaging, instead there is a steady, gradual change in $p(y|x)$ (see App. D.4 for further explanation). We find analogous S-shaped curves in ICL with LLM predictive distributions $p(y|x)$ in GPT-3.5+, suggesting hypothesis selection rather than hypothesis averaging (Fig. 5.4, 5.5). These phase changes are not predicted by theories of ICL as linear regression [8] or few-shot learning [9, 10], where loss steadily decreases with more context examples in x. 69

5.3 **Determining whether a sequence is random can be viewed as search for a simple program that could generate that sequence.** `HTTTTTTT` can be described with a short deterministic program `simpleSequence`, and has a higher $p(h)$ according to a simplicity prior, compared to `TTHTHHTH` and `complexSequence`. Both sequences can be generated by `randomSequence`, but with lower likelihood $p(x|h)$. 71

5.4 **Prompt templates and GPT-3.5 Randomness Judgments** (Left) Prompt templates used for the Randomness Generation (top) and Judgment (bottom) tasks. {`sequence`} is substituted with a list of context flips x, e.g. `Heads`, `Tails`, `Tails`, `Heads`. {`p`} is substituted with the probability of Tails, and {`1 - p`} with the probability of Heads. (Right) In Randomness Judgment tasks, we observe sharp transitions in the predictive distribution $p(y = \texttt{random}|x)$ for GPT-3.5, from high-confidence in x being generated by a random process, to high confidence in a non-random algorithm, as additional data x is provided from the same concept. See App. D.7 for additional concepts. 72

5.5 **With enough context, GPT-3.5 learns simple formal language concepts, transitioning from generating subjectively random sequences to only generating values matching the concept** (Left) GPT-3.5 next-token predictive distribution $p(y|x)$ visualized as a binary tree, where red arrows correspond to *Heads* (0), blue arrows to *Tails* (1), and nodes matching the target concept $C = (011)^n$ are dark green. $p(y|x)$ changes with varying $|x|$, here $|x| = 39$ and next-token predictions strongly follow the concept C. Also see Fig D.7, D.8 in Appendix. (Right) In-context learning dynamics for simple formal languages $x = (\texttt{HTH})^n$ (`010`) and $x = (\texttt{HTT})^n$ (`011`) as a function of context length $n = |x|$. Prediction accuracy computed as the total probability mass of $p(y|x)$ assigned to valid continuations of the formal language x, as a function of prediction depth $d = |y|$ and context length $|x|$. Curves shown are for depth $d = 6$, where only 3 out of 64 possible length-6 binary sequences y match concept C. Note: bottom model is `gpt-3.5-turbo-instruct-0914`. Also see App. D.6. 74

5.6 **GPT-3.5 generates subjectively random binary sequences that deviate from a Bernoulli process.** (Left) Running averages for flip sequences generated by each model, where 0 denotes '`Heads`' and 1 denotes '`Tails`'. Compared to a Bernoulli process (top), sequences generating using GPT (middle) and those of our Window Average model (bottom) stay closer to the mean, repeating the same patterns more often. (Right) Empirical conditional probabilities for a third-order Markov Chain fit to sequences y generated by GPT-3.5 `text-davinci-003`, a Bernoulli process centered at the mean of GPT sequence $\overline{y}$, and our Window Average model ($w = 5$). In the simulated Bernoulli process, edges are fairly uniform; edges for GPT-3.5 and the Window Average model are non-uniform, but similar to one another. 75

5.7 **GPT-3.5 shows a Gambler's fallacy bias of avoiding long runs.** (Top) Distribution of mean values of flip sequences ($\mu = \frac{1}{T}\sum_t y_t$) generated by GPT-3.5 (`text-davinci-003`) with the specified $p(\texttt{Tails})$, compared with a Bernoulli process and our Window Average model with the same mean as the GPT-3.5 flips. Flips generated by GPT approximately follow the expected mean $p(\texttt{Tails})$, but have lower variance than a Bernoulli distribution. (Bottom) Length of the longest run for each sequence, where a run is a sub-sequence of the same value repeating. In this case, we see a clear bias in GPT-3.5 to avoid long runs, with a similar pattern across all values of $p(\texttt{Tails})$ despite the x-axis changing in scale. 76

5.8 **GPT repeats certain sub-sequences more often than chance.** We find that GPT-3.5 repeats certain sub-sequences more often than is predicted by our Window Average model. While the Window Average model (green) generates fewer unique sub-sequences than a Bernoulli process (red), this does not account for the bias in GPT-3.5 (`text-davinci-003`, in blue) to repeat many of the same sub-sequences. This disparity increases with longer sub-sequences (see Fig. D.18 in Appendix). 78

6.1 **Mechanistic Lens on Mode connectivity.** Consider two sets of parameters that minimize loss using background $\theta_{\text{Background}}$ and object shape θ_{Shape} as the input attributes for prediction, respectively. Are such *mechanistically dissimilar* minimizers connected via paths of low loss in the landscape? Does the dissimilarity of these mechanisms affect the simplicity of their connectivity paths? Can we exploit this connectivity to switch between minimizers that use our desired mechanisms? 82

6.2 **Mechanistic Similarity:** We define mechanistic similarity of two models based on how they respond to unit interventions on the data-generating process, i.e., interventions on specific dimensions of the latent vector z; e.g., $\mathcal{A}_1$ (shape) and $\mathcal{A}_2$ (background) in the figure. Here, yellow circles represent the prediction of a given model (column) on a counterfactual image (row). Models whose predictions are invariant to the same set of interventions (denoted $\theta_1 \sim \theta_2$) are termed mechanistically similar. 86

6.3 **Data-Generating Process (left).** We augment the natural latents $\{z_n\}$ of a data-generating process with a set of synthetic latents $\{z_s\}$. The attributes induced in the input by these synthetic latents are called *cues*. Conditioning (grey, dotted line) the value of a synthetic latent on the target label (y), we can induce correlation between its corresponding cue and the desired model output. If the cue is made easily separable, a DNN will preferentially learn mechanisms to use the cue for making its predictions [11] (see also training curves in App. E.2). **Synthetic Datasets (right).** Following the protocol above, we embed synthetic cues in three existing datasets: (1) CIFAR-10 with 3×3 box cues whose locations depend on the target label; (2) CIFAR-100 with 3×3 box cues colored according to the first digit of the object label, and located according to the second digit; and (3) Dominoes [11], where CIFAR-10 images are concatenated with Fashion-MNIST images of the same class. We analyze counterfactual datasets that involve removing the cue (*w/o Cue*), keeping it (*w/ cue*), randomizing it (*Rand. cue*), or randomizing the natural image (denoted *Rand. image*). These counterfactuals help us ascertain the extent to which a model relies on natural vs. spurious attributes. 88

6.4 **Non-Linear Mode Connectivity of Mechanistically Dissimilar Models.** We train ResNet-18 models on our synthetic CIFAR-10 datasets with and without box-cues (denoted θ_{C} and θ_{NC}, respectively). We evaluate quadratic and linear connectivity paths; quadratic paths identified using both data with and w/o cues are analyzed. Line colors denote proportion of the training data with synthetic cues. Plot titles denote evaluation data (see Fig. 6.3), including data where either the cue is present (w/ Cue), absent (w/o Cue), randomized (Rand. Cue), or the underlying image is randomized (Rand. Image). As shown, θ_{NC} yields the same performance upon randomization of the cue, while the performance of θ_{C} decreases substantially; i.e., the two minimizers induce mechanistically dissimilar models. We see: (i) quadratic paths can be easily identified to mode connect mechanistically dissimilar models; (ii) linear paths cannot be identified, even after permutations; and (iii) mechanistic connectivity is unfounded. See App. E.7 for similar results on other settings and loss curves. 90

6.5 **Analyzing Pre-trained vs. Fine-Tuned Models: Lack of Linear Connectivity implies Mechanistic Dissimilarity.** We train VGG-13 and ResNet-18 models on our synthetic CIFAR-10 dataset with box-cues and perform naïve fine-tuning on data without cues for 100 epochs using different initial learning rates (LR) and a step-decay schedule. Corresponding models are denoted θ_{C} and θ_{FT}; line colors denote proportion of dataset with synthetic cues; titles denote evaluation datasets, similar to Fig. 6.4. We plot test accuracy as a function of location on the linear paths (after permutation). Using a large learning rate or enforcing perfect correlation between the cue and label induces loss barriers along the linear path, i.e., linear mode connectivity does not hold. Simultaneously, the models respond differently to counterfactuals, i.e, they are mechanistically dissimilar and not connected. For a small/medium learning rate, we notice θ_{FT} remains linear mode connectivity θ_{C} on data with cues. Simultaneously, we see the corresponding models responding similarly on counterfactuals and are mechanistically similar. See App. E.8 for similar results on other datasets, models, and loss curves. . . . 91

6.6 **Clues for altering a model's mechanisms.** Given a discriminative attribute C, the loss landscape along the linear path connecting a model invariant to the attribute (θ_{NC}) versus a model that relies on the attribute (θ_C) generally shows (i) a loss barrier along the path and (ii) invariance at the endpoint corresponding to θ_{NC}, i.e., $\mathcal{L}(\theta_{NC}, D_{NC}) = \mathcal{L}(\theta_{NC}, D_C)$. 93

7.1 **How does fine-tuning alter a model's capabilities?** (a) Pretraining on huge, web-crawled datasets leads to LLMs learning several capabilities that can justifiably process an input. The figure shows this using an illustrative query, "write a story a 5-year old would understand." Via careful prompting, the desired answer can be retrieved, indicating both desired and undesired capabilities exist in an LLM. (b) Upon fine-tuning, e.g., to avoid use of undesirable capabilities, we hypothesize that three explanations are possible: (i) a minimal transformation of the original capability is learned, e.g., a negation of the original capability; (ii) the undesirable capability is deleted altogether; or (iii) the use of another relevant capability is amplified. 100
7.2 **Capability Relevance.** Consider the task of completing a passage while maintaining its narrative. Herein, the ability to recognize the sentiment of a text will be deemed *strongly relevant* and the ability to recognize negative words *weakly relevant*. Such words are often spuriously correlated with a negative sentiment. 103
7.3 **Experimental setup.** We primarily analyze two setups: (i) Tracr "compiled" models with predefined capabilities and (ii) models trained to learn capabilities defined via a PCFG, following prior work [12]. During fine-tuning, we train the model on a dataset $\mathcal{D}_{\texttt{FT}}$ that promotes learning of a capability $\mathcal{C}'$. We randomly embed spurious attributes in the fine-tuning dataset that correlate with features extracted by a pretraining capability $\mathcal{C}$ to operationalize capability relevance. 104
7.4 **Analysis protocols.** We analyze how fine-tuning affects a pretrained model's capabilities by (i) Reverse Fine-tuning, (ii) network pruning, (iii) attention visualization, and (iv) probing classifiers. We use (ii)—(iv) to show fine-tuning often yields wrapped capabilities. For further evidence, we use (i) and (ii) and find we can "revive" the original capabilities, i.e., the model starts performing well on the pretraining task again. See App. F.4 for precise details. 106
7.5 **Fine-tuning accuracy w.r.t. number of training iterations.** We vary the probability of sampling the token $\texttt{O}_{\texttt{FT}}$ in the pretraining data and the spurious correlation in the fine-tuning datasets. When the prior is sufficiently high (a, b), we find the model learns to perform well on the downstream task. Meanwhile, if the prior is low (c), the model learns the downstream task only if a high enough learning rate is used and the spurious correlation is imperfect. This indicates the ability to extract information relevant for the downstream task is likely to be exploited during fine-tuning. 107
7.6 **Impact of sampling prior on the pretraining task's accuracy as fine-tuning is performed.** We plot accuracy on the pretraining task w.r.t. fine-tuning iterations. When the sampling prior of the $\texttt{O}_{\texttt{FT}}$ is low during pretraining, the pretraining task accuracy quickly plummets, especially if the spurious correlation is high; having a high sampling prior mitigates this behavior. This *indicates* pretraining capabilities are affected the most when they are weakly relevant. 108

7.7 **Pruning a few neurons is sufficient to retrieve pretraining task accuracy.** We plot accuracy w.r.t. number of neurons pruned to improve performance on the pretraining task. We see when a small learning rate is used for fine-tuning, the pretraining task's performance improves after just 5–15 neurons are pruned (top), while the fine-tuning task's performance reduces correspondingly (bottom). We argue these neurons serve as a wrapper to minimally alter the weakly relevant pretraining capability and exploit the spurious correlation present in the fine-tuning data. 109

7.8 **Probing the presence of pretraining (top) and fine-tuning (bottom) capabilities.** We plot probe accuracy versus the index of the block in the Transformer model. $C_{\texttt{Te}}$ is set to 0. The pretrained model (left) acts as a baseline for the trend of performance through the model's blocks. In most scenarios, we find we can infer the count of $\texttt{O}_{\texttt{PT}}$ with a similar trend as the pretrained model (left). A drop in performance is observed only when learning rate η_M is used with a weakly relevant capability (low sampling prior). This indicates pretraining capabilities continues to persist upon fine-tuning. . . . 110

7.9 **Reverse Fine-Tuning:** We set $C_{\texttt{Te}}$ to be 0 to test if the model performs well regardless of a spurious correlation. Models are fine-tuned for 10K iterations. We observe that when a strongly relevant capability is present (a, b), the model very quickly (0.1–1K iterations) starts to perform well on the task via `reFT` , even if behavior relevant to the capability ceased during pretraining (e.g., when $C_{\texttt{Tr}}$ is 1). Meanwhile, when the model possesses a weakly relevant capability (c), this "revival" is *comparatively* slower (3K iterations). In contrast, the $\texttt{Scr.} + \texttt{FT}$ baseline only reaches perfect accuracy at 4.5K iterations and when using a larger learning rate, i.e., η_M. 111

7.10 **Visualizing attention maps of fine-tuned Tracr models.** Leftmost panel shows the Tracr compiled model's attention map on the counter task. Upon fine-tuning under different spurious correlations, we see the model continues to pay attention to the pretraining target $\texttt{O}_{\texttt{PT}} = a$. Only when a large enough learning rate and zero spurious correlation is used, there is a change in the attention pattern. 111

7.11 **Validation on TinyStories.** Models are trained to produce stories with several features (e.g., `foreshadowing`) and fine-tuned via different protocols to not produce stories with a specific, predefined feature (specifically, `twists`) (see App. F.6 for details). We call this latter feature "forbidden". **Left:** We probe the existence of this feature at intermediate Transformer blocks. Probe accuracy on models pretrained with or without `twist` data (*Present*/*Not in Pretraining*, respectively) act as upper and lower bounds on the expected accuracy, and are plotted for ease of comparison. Regardless of the fine-tuning protocol (Filtering, Filtering + Randomization, Filtering + Mix & Match), for the lower LR, no protocol removes a meaningful amount of information and a similar but less strong trend holds for the higher LR. **Right:** We plot the loss during reverse fine-tuning (`reFT`) to again produce stories with the forbidden feature. Fine-tuned models' loss goes down very quickly (30–300 iterations) compared to baselines (which never reach the same loss; also see Tab. 7.1). Both these results indicate the capability of identifying the forbidden feature, a necessary capability for story modelling, continues to persist after fine-tuning. 112

A.1 **Examples of data samples with pairs of images and corresponding concept classes.** 120

A.2 **The architecture of the conditional diffusion model.** The architecture of the conditional diffusion model involves an iterative process comprising noise addition and denoising steps. The model leverages conditioning information, specifically concept classes, to guide the transformation of the input image towards a desired state. In our implementation, we utilize a U-Net to parameterizethe denoising process. The U-Net architecture consists of three upsampling convolutional layers and three downsampling convolutional layers, which are connected through skip connections. Each layer within the U-Net includes a pooling layer, a global attention mechanism, and a GELU activation function. 121

A.3 **Average pixel values of the generated images and prediction outcomes of the linear probing classifier for shape.** 122

A.4 **Average pixel values of the generated images and prediction outcomes of the linear probing classifier for color.** 123

A.5 **Average pixel values of the generated images and prediction outcomes of the linear probing classifier for size.** 123

A.6 **Compositional generalization.** (a) Consider a lattice representation of a concept graph corresponding to four concept variables. Blue nodes denote classes represented in the training data; a model can either memorize data from these classes or learn capabilities to transform samples from one class to another. If it learns capabilities, it can compose them with data-generating process of samples from a concept class in the training data and produce samples that are entirely out-of-distribution, denoted as pink and green nodes. (b) We find compositional generalization happens in a sequence, starting with concept distance of 1 and then progressing to concept distance of 2 and 3. . 124

A.7 **Analyzing the bottlenecks of compositional generalization.** We examine the prediction accuracy of concept variables (shape, color, and size) and the overall classification accuracy. In each epoch of training, 50 samples were generated and classified using linear probe. We observe that various concept bottlenecks hinder the learning of different compositional objects. For example, compositional generalization to blue-colored objects (011, 101, 111) is bottlenecked by the color concept, while generalization to small red triangles (111) is bottlenecked by the shape concept. 125
A.8 **Concept distance from the training set govern the order in which compositional capabilities emerge.** (a) Concept graph (cube) depicting training data points (blue nodes) and concept distances for test data points. (b) Compositional generalization happens in sequence, starting with concept distance = 1 and progressing to concept distance = 2. The x-axis represents the number of epochs, and the y-axis represents the progress of compositional generalization. (c) Images generated as a function of time clearly show a sudden emergence of capability to change color for small, red triangles. . . . 126
A.9 **Delayed emergence of abilities to generate underrepresented colors for distant classes.** Prediction accuracy of color based on generated samples at each epoch during training. We observe that the ability to generate underrepresented colors emerges significantly later as the concept distance of a concept class increases, highlighting the need for extended training beyond the point of achieving in-distribution generalization. This prolonged training enables effective composition of the underrepresented concept and leads to improved generalization. 126
A.10 **Multiplicity underlies the sudden emergence of compositional capabilities.** (a) Accuracy of producing samples from the concept class {111, (triangle, blue, small)}, which has a concept distance of 2 from the training data. (b) Learning dynamics of accuracies for predicting each of the three concept variables: shape (blue), color (orange), and size (green). 127
A.11 **The additional experimental results with different configurations by using varied sets of nodes in the training data.** 127

B.1 **Adjacency matrix with time.** As the model undergoes training, entities are seen in the context of more properties. Recording whether a pair of object and property have been seen together, we can get the adjacency matrix for a bipartite graph corresponding to entities and properties that constitute the data-generating process, as shown in the plots. The red-dotted lines indicate class boundaries. This graph can be deemed as the empirical type constraints graph, and, as training occurs, it will get closer to the ground truth graph. Note that several entities and properties will never be seen together, however, other entites from the class may be paired with a property, allowing the model a signal to infer that the entities likely belong to the same class. 132
B.2 **Exemplars where an entity's properties are described.** Each box represents a different task; specifically, free generation, unscrambling, and conditional generation. 133

B.3 **Exemplars where a subject and object are bound via a verb.** Each box represents a different task; specifically, free generation, unscrambling, and conditional generation. 134
B.4 **Learning curves with varying number of properties (base setting).** As number of properties are varied, we see the overall loss is substantially more continuous, but individual tasks can see sudden learning. For example, in free generation, we see a sudden loss drop. This point is precisely when the model learns to produce grammatically correct sentences. Very slightly after this point, both tasks of unscrambling and conditional generation see improvement as well. There is a second change of slope for these tasks between 10^3 to 7×10^3 iterations, depending on the number of properties. These points match match the moment where the model starts to improve in its Type Check performance. 141
B.5 **Learning curves for setting with** 18000 **properties (base setting).** We report this plot to zoom into a specific configuration. As can be seen, there is some minimal variance across runs, but mostly points of transition are in a similar range. 142
B.6 **Learning curves with varying number of properties (1800 entities).** As number of properties are varied, we see the overall loss is substantially more continuous, but individual tasks can see sudden learning. For example, in free generation, we see a sudden loss drop. This point is precisely when the model learns to produce grammatically correct sentences. Very slightly after this point, both tasks of unscrambling and conditional generation see improvement as well. There is a second change of slope for these tasks between 10^3 to 7×10^3 iterations, depending on the number of properties. These points match match the moment where the model starts to improve in its Type Check performance. 142
B.7 **Learning curves for setting with** 18000 **properties (1800 entities).** We report this plot to zoom into a specific configuration. As can be seen, there is some minimal variance across runs, but mostly points of transition are in a similar range. 143
B.8 **Learning curves with varying number of properties (2 classes).** As number of properties are varied, we see the overall loss is substantially more continuous, but individual tasks can see sudden learning. For example, in free generation, we see a sudden loss drop. This point is precisely when the model learns to produce grammatically correct sentences. Very slightly after this point, both tasks of unscrambling and conditional generation see improvement as well. There is a second change of slope for these tasks between 10^3 to 7×10^3 iterations, depending on the number of properties. These points match match the moment where the model starts to improve in its Type Check performance. 143
B.9 **Learning curves for setting with** 18000 **properties (2 classes).** We report this plot to zoom into a specific configuration. As can be seen, there is some minimal variance across runs, but mostly points of transition are in a similar range. 144

B.10 **Grammaticality and Type Checks with varying number of properties (base setting).** As number of properties are varied, we see the grammar is learned around broadly the same time, i.e., grammar learning is invariant to number of properties (as also seen in learning curves). For type check, we see as soon as grammaticality reaches its maximum, relative constraints quickly improve, leading to a boost in accuracy of all constraints evaluation (rightmost panel). Descriptive constraints see a transition at this point as well, but then after a period of saturation (akin to a saddle point), start to improve at an approximately linear rate (on log-log scale) until saturating again. These results match the base setting shown in main body. 145
B.11 **Grammaticality and Type Checks with 18000 properties (base setting).** We report this plot to zoom into a specific configuration, in particular the one used in main body. As can be seen, there is some minimal variance across runs, but mostly points of transition are in a similar range. 145
B.12 **Grammaticality and Type Checks with varying number of properties (1800 entities).** To demonstrate robustness of results, we increase number of entities in this experiment. We again find that as number of properties are varied, the grammar is learned around broadly the same time, i.e., grammar learning is invariant to number of properties (as also seen in learning curves). For type check, we see as soon as grammaticality reaches its maximum, relative constraints quickly improve, leading to a boost in accuracy of all constraints evaluation (rightmost panel). Descriptive constraints see a transition at this point as well, but then after a period of saturation (akin to a saddle point), start to improve at an approximately linear rate (on log-log scale) until saturating again. These results match the base setting shown in main body of the dissertation. 146
B.13 **Grammaticality and Type Checks with 18000 properties (1800 entities).** We report this plot to zoom into a specific configuration. As can be seen, there is some minimal variance across runs, but mostly points of transition are in a similar range. 146
B.14 **Grammaticality and Type Checks with varying number of properties (2 classes).** To demonstrate robustness of results, we increase fewer classes in this experiment. We again find that as number of properties are varied, the grammar is learned around broadly the same time, i.e., grammar learning is invariant to number of properties (as also seen in learning curves). For type check, we see as soon as grammaticality reaches its maximum, relative constraints quickly improve, leading to a boost in accuracy of all constraints evaluation (rightmost panel). Descriptive constraints see a transition at this point as well, but then after a period of saturation (akin to a saddle point), start to improve at an approximately linear rate (on log-log scale) until saturating again. The results are less prominent for this setting, but zooming in (see figure below) shows the claims do follows the base setting shown in main body of the dissertation. 147

B.15 **Grammaticality and Type Checks with 18000 properties (2 classes).** We report this plot to zoom into a specific configuration. As can be seen, there is some minimal variance across runs, but mostly points of transition are in a similar range. 147
B.16 **NLL of sentences with different perturbations and varying number of properties (base setting).** See text in App. B.4.3 for a detailed discussion. Broadly, we see NLL of Seen evaluation is very slightly better than Uniform. 150
B.17 **NLL of sentences with different perturbations for setting with** 18000 **properties (base setting).** We report this plot to zoom into a specific configuration. As can be seen, there is some minimal variance across runs, but mostly points of transition are in a similar range. 150
B.18 **NLL of sentences with different perturbations and varying number of properties (1800 entities).** See text in App. B.4.3 for a detailed discussion. Broadly, these plots show results with fewer classes show similar behavior as the base setting. 151
B.19 **NLL of sentences with different perturbations for setting with** 18000 **properties (1800 entities).** We report this plot to zoom into a specific configuration. As can be seen, there is some minimal variance across runs, but mostly points of transition are in a similar range. 151
B.20 **NLL of sentences with different perturbations and varying number of properties (2 classes).** See text in App. B.4.3 for a detailed discussion. Broadly, these plots show results with fewer classes show similar behavior as the base setting. 152
B.21 **NLL of sentences with different perturbations for setting with** 18000 **properties (2 classes).** We report this plot to zoom into a specific configuration. As can be seen, there is some minimal variance across runs, but mostly points of transition are in a similar range. 152
B.22 **Grammar characteristics of model generation as number of properties are varied (base setting).** See App. B.4.4 for a detailed discussion. Broadly, model generated sentences are in a similar range as our language's. 154
B.23 **Grammar characteristics of model generation as number of properties are varied (1800 entities).** See App. B.4.4 for a detailed discussion. Broadly, model generated sentences are in a similar range as our language's. 154
B.24 **Grammar characteristics of model generation as number of properties are varied (2 classes).** See App. B.4.4 for a detailed discussion. Broadly, model generated sentences are in a similar range as our language's. 155

B.25 **Results on unscrambling task as number of properties are varied (Base setting).** See App. B.4.4 for a detailed discussion on metrics used. Broadly, we see the results presented in the main body of the dissertation are consistent across varying number of properties: the model first witnesses improvement in exact match because relative sentences start to improve, i.e., the grammar is learned, performance then saturates, and then it finally starts improving again once descriptive sentences' accuracy starts to improve. We emphasize metrics assigning partial credit also show sudden changes, i.e., our results are not sensitive to metrics used. 157
B.26 **Results on the unscrambling task for setting with** 18000 **properties (base setting).** We report this plot to zoom into a specific configuration. As can be seen, there is some minimal variance across runs, but mostly points of transition are in a similar range. 158
B.27 **Results on unscrambling task as number of properties are varied (1800 entities).** See App. B.4.4 for a detailed discussion on metrics used. Broadly, we see the results presented in the main body of the dissertation are consistent across varying number of properties and a larger number of entities compared to the base setting: the model first witnesses improvement in exact match because relative sentences start to improve, i.e., the grammar is learned, performance then saturates, and then it finally starts improving again once descriptive sentences' accuracy starts to improve. We emphasize metrics assigning partial credit also show sudden changes, i.e., our results are not sensitive to metrics used. 159
B.28 **Results on the unscrambling task for setting with** 18000 **properties (1800 entities).** We report this plot to zoom into a specific configuration. As can be seen, there is some minimal variance across runs, but mostly points of transition are in a similar range. 160
B.29 **Results on unscrambling task as number of properties are varied (Base setting).** See App. B.4.4 for a detailed discussion on metrics used. Broadly, we see the results presented in the main body of the dissertation are consistent across varying number of properties and fewer classes than the base setting: the model first witnesses improvement in exact match because relative sentences start to improve, i.e., the grammar is learned, performance then saturates, and then it finally starts improving again once descriptive sentences' accuracy starts to improve. We emphasize metrics assigning partial credit also show sudden changes, i.e., our results are not sensitive to metrics used. 161
B.30 **Results on the unscrambling task for setting with** 18000 **properties (2 classes).** We report this plot to zoom into a specific configuration. As can be seen, there is some minimal variance across runs, but mostly points of transition are in a similar range. 162

B.31 **Evolution of attention maps for descriptive sentences.** Since visualizing the evolution across time is difficult, we pick a few salient points according to region identified as different phase according to learning curves and visualize them. We find that indeed the first time the model sees a sparse attention structure is around 1000 iterations of training, i.e., when it becomes accurate at producing grammatically correct sentences. From 5–30000 iterations, the model learns to focus on subject when producing next token, but the link verb and descriptive property do not pay substantial attention to subject. However, there is a point after 30000 iterations post which the attention on subject substantially increases; this is the range where other evaluation show a change in performance as well. 163
B.32 **Evolution of attention maps for relative sentences.** Since visualizing the evolution across time is difficult, we pick a few salient points according to region identified as different phase according to learning curves and visualize them. We find that indeed the first time the model sees a sparse attention structure is around 1000 iterations of training, i.e., when it becomes accurate at producing grammatically correct sentences. Once the model learns relative type constraints (around 5000 iterations generally), we see the attention patterns essentially stabilizes and does not change much; albeit, it does get sharper (more attention to the tokens that were already being attended). Interestingly, we see object tokens paying large attention to verbs, i.e., objects are being selected by analyzing which verbs came before them. We see some non-trivial dependence between verb and the adverb preceding it, which itself solely attends to the subject token (i.e., it copies that token's representation); this could suffice to ensure the model gets verbs right for a subject. 164
B.33 **Demonstration (27600 properties).** We report this plot to zoom into a specific configuration. As can be seen, there is some minimal variance across runs, but mostly points of transition are in a similar range. 165
B.34 **Results on metrics specific to unscrambling (Base setting).** We see a collapse of all metrics under a 1.5 scaling exponent. 170
B.35 **Results on metrics designed to approximate cluster membership (Base setting).** We see an approximate collapse of inflection points in the normalized rank and percent top-K metrics under a 0.5 scaling exponent. Average probability is expected to follow accuracy curves, yielding a 1.5 scaling exponent. Interestingly, NLL has a scaling exponent of 2.0. 170
B.36 **Average probability curves for different x-axis rescalings (Base setting).** An exponent of 1.5 induces the best collapse of experimental curves. Inset plots zoom in near transition point. 171
B.37 **Results on unscrambling task as number of properties are varied (1800 entities).** We see a collapse of all metrics under a 2.0 scaling exponent, indicating an effect of number of entities possibly on the transition point. Arguably, this is expected since unscrambling is affected by both number of entities and properties involved in the language. 171

B.38 **Results on unscrambling task as number of properties are varied (1800 entities).** We see an approximate collapse of inflection points in the normalized rank and percent top-K metrics with a 0.5 scaling exponent. Average probability is expected to follow accuracy curves, yielding a 2.0 scaling exponent. NLL again shows a scaling exponent of 2.0, similar to base setting. 172
B.39 **Average probability curves for different x-axis rescalings (1800 entities).** An exponent between 1.5–2.0 induces the best collapse of experimental curves, similar to base setting. Inset plots zoom in near transition point. . . 172
B.40 **Results on unscrambling task as number of properties are varied (2 classes).** We see a collapse of all metrics under a 1.5 scaling exponent. . . . 173
B.41 **Results on unscrambling task as number of properties are varied (2 classes).** We see an approximate collapse of inflection points in the normalized rank and percent top-K metrics with a 0.5 scaling exponent. Average probability is expected to follow accuracy curves, yielding a 1.5 scaling exponent. NLL again shows a scaling exponent of 2.0, similar to other settings. 173
B.42 **Average probability curves for different x-axis rescalings (2 classes).** An exponent of 1.5 induces a perfect collapse of experimental curves, similar to base setting. Inset plots zoom in near transition point. 174
B.43 **Results on metrics specific to unscrambling (Base setting).** We see a collapse of descriptive and all constraints metrics under a 0.5 scaling exponent; relative constraints are clearly invariant to number of properties. 174
B.44 **Results on metrics designed to approximate cluster membership (Base setting).** We see all metrics show an approximate collapse under a 0.5 scaling exponent. Collapse is in the inflection points for the normalized rank and percent top-K metrics. 175
B.45 **Average probability curves for different x-axis rescalings (Base setting).** An exponent between 0.5–1.0 can be expected to induce the best collapse for the metric of average probability; results in Fig. B.46 show the exponent is closer to 0.5. 175
B.46 **Average probability curves for different x-axis rescalings (Base setting).** An exponent of 0.5 better aligns the transition points. 176
B.47 **Results on metrics specific to unscrambling (1800 entities).** We see a collapse of descriptive and all constraints metrics under a 0.5 scaling exponent; relative constraints are clearly invariant to number of properties. 176
B.48 **Results on metrics designed to approximate cluster membership (1800 entities).** We see all metrics show an approximate collapse under a 0.5 scaling exponent. Collapse is in the inflection points for the normalized rank and percent top-K metrics. 177
B.49 **Average probability curves for different x-axis rescalings (1800 entities).** An exponent between 0.5–1.0 can be expected to induce the best collapse for the metric of average probability; results in Fig. B.50 show the exponent is closer to 0.5. 177
B.50 **Average probability curves for different x-axis rescalings (1800 entities).** An exponent of 0.5 better aligns the transition points. 178

B.51 **Results on metrics specific to unscrambling (2 classes).** We see a collapse of descriptive and all constraints metrics under a 0.5 scaling exponent; relative constraints are clearly invariant to number of properties. 178
B.52 **Results on metrics designed to approximate cluster membership (2 classes).** We see all metrics show an approximate collapse under a 0.5 scaling exponent. Collapse is in the inflection points for the normalized rank and percent top-K metrics. 179
B.53 **Average probability curves for different x-axis rescalings (2 classes).** An exponent between 0.5–1.0 can be expected to induce the best collapse for the metric of average probability; results in Fig. B.54 show the exponent is closer to 0.5. 179
B.54 **Average probability curves for different x-axis rescalings (2 classes).** An exponent of 0.5 better aligns the transition points. 180
B.55 **Bilinear Spline Fits.** Working with the hypothesis that the model's accuracy on unscrambling descriptive sentences undergoes first a saturation regime at low performance, before suddenly changing slope and rapidly improving in performance, we fit bilinear splines to maximally explain these results. The breakpoint identified using these fits is used to define the transition point scaling curve in Fig. B.56. One can easily see in these curves that the breakpoint is moving rightwards as the number of properties are increased. 180
B.56 **Transition point scaling.** Working with the hypothesis that the model's accuracy on unscrambling descriptive sentences undergoes first a saturation regime at low performance, before suddenly changing slope and rapidly improving in performance, we fit bilinear splines to maximally explain these results. The breakpoint identified using these fits is used to define the transition point scaling curve. Examples of these fits are shown in Fig. B.55. 181

C.1 **Different distributions in concept values result in different concept signal.** (Left) The `color` separation between the classes is stronger than the `size` separations resulting in a stronger concept signal in the `color` dimension. (right) The `size` separation between the classes is stronger, thus resulting in a stronger concept signal in `size` 183
C.2 **Loss vs. Accuracy vs. Concept Space.** Loss and accuracy do not always tell the full story of what, and even more importantly *when*, the model is learning a capability. The *star* symbol in panel (c) shows point of sudden turn and concept learning, i.e., the moment when well defined prompting protocols can elicit the desired output from the model, indicating the model has learned the capability to alter a concept. 186
C.3 **Concept Space Dynamics for all classes (`00`, `01`, `10`, `11`).** The experiment is identical as in Fig. 4.4. The [0,1) normalized `color` concept signal is color coded in every trajectory. Two training data trajectories are shown in gray in the last panel to show *concept memorization*. 186

C.4 **Concept Space Dynamics in CelebA.** We train on the classes `00`=(Male, NOT Smiling), `01`=(Male, Smiling), `10`=(Female, NOT Smiling) (plotted in blue) and test on the class `10`=(Female, Smiling) (plotted in pink). We find similar observations as in Fig. 4.4.2, *concept memorization* and the resulting bias. 188
C.5 **Concept Space Dynamics in CelebA.** We train on the classes `00`=(Male, NO Hat), `01`=(Male, Hat), `10`=(Female, NO Hat) (plotted in gray) and test on the class `10`=(Female, Hat) (plotted in pink). Due to the precise data distribution being more skewed in this experiment, we find the model never generalizes out-of-distribution, but does show behavior akin to concept memorization. 189
C.6 **Concept Learning Geometry underlies emergence** Success and Failure modes of out-of-distribution (OOD) generalization. (a) A success case where color carries the strongest concept signal. We see aspects of *concept memorization* from the class `011`. (b) A failure case where the background color didn't generalize. In this case, the model does not produce the right background color while the concept space suggest that this *capability* is present. 190
C.7 **Learning Dynamics in Concept space for Classifier-free Guidance** We show the learning dynamics in the concept space for all the classes (`00`, `01`, `10`, `11`). The generalization task (rightmost) shows a sharp transition from *concept memorization* to out-of-distribution (OOD) generalization independently of the Classifier-Free Guidance (CFG) scale. 190
C.8 **Embedding patching.** We patch the embedding module (an MLP) used for transforming the conditioning information into an embedding that the model processes from the last checkpoint to intermediate checkpoints. Panel (a) shows the baseline accuracy for out-of-distribution (OOD) generalization across five different seed runs, while panel (b) shows accuracy achieved when the patched embedding module is used. 191

D.1 **Consistent tokenization with 'Heads, Tails, . . . ' compared to '010 . . . '** With comma-separated lists of *'Heads'* and *'Tails'* flips, tokens consist of *'Heads'*, *'T'*, *'ails'*, *','*, as well as occasional erroneous tokens such as *'Head'*. With comma-less *'0'* and *'1'* tokens, the GPT tokenizer chunks many binary sub-sequences into individual tokens. 195
D.2 **Cognitive Interpretability in the context of prior work** Cognitive Interpretability studies in-context learning dynamics in LLMs, positing theories of the latent concepts enabling behavioral capabilities. It is a middle ground between behavioral benchmarks, which treat models as black boxes and evaluate hit-or-miss accuracy, and mechanistic interpretability, which studies toy models and training loss dynamics, analogous to how cognitive science is a middle ground between behaviorist psychology and neurobiology. 196

D.3 (Left) Normalized hypothesis posteriors from Fig. 5.2 Left. These are normalized so that probability of all hypotheses sums to one $\sum_h p(h|x) = 1$, i.e., $p(h = \text{Bernoulli}|x) + p(h = \text{Non-Random}|x) = 1$. (Right) The soft model selection curve shown in Fig. 5.2 Right, with varying temperature parameter τ interpolating $p_\tau(y|x)$ between Model Averaging ($\tau = 1$, yellow line) and Model Selection ($\tau \to \infty$, dark blue line). 198
D.4 **GPT-3.5 demonstrates a task bias, returning to answer MMLU questions after many distractor flips** In these experiments, we take MMLU questions and append sequences of 'distractor flips', repeating samples from $(\texttt{01})^n$, before querying the LLM for next-token probability. Individual lines are shown for flip tokens (*'Heads'*, *'Tails'*, *'T'*, *'ails'*) compared with MMLU answer tokens (*'A'*, *'B'*, *'C'*, *'D'*). Even with many distractor tokens, GPT-3.5 converges back to answering the MMLU question, demonstrating that our formal concept learning results for the Generation task are not merely a local text bias where the LLM forgets the original prompt. . . 201
D.5 **Predictive distributions $p(y|x)$ by each LLM for Concept $(\texttt{010})^n$, at each prediction depth d.** Colors correspond to different prediction depths, also refer to Figure 5.5. Note: `text-ada-001` results are not shown since results did not follow the required format (*'Heads, Tails, ...'*) adequately to be analyzed. 202
D.6 **Predictive distributions $p(y|x)$ by each LLM for Concept $(\texttt{011})^n$, at each prediction depth d.** Colors correspond to different prediction depths, also refer to Figure 5.5. 203
D.7 **Predictive distribution $p(y|x)$ trees with $d = 6$ for concept $C = (011)^n$ with $|x| \in \{6, 12, 18\}$.** Since $|x|$ is increasing by the same depth as the tree $\Delta_{|x|} = d = 6$, the transition from generating subjectively random numbers to deterministically repeating 011 is visibly apparent. Also see Figure 5.5. . . . 204
D.8 **Predictive distribution $p(y|x)$ trees with $d = 4$ for concept $C = (011)^n$ with $|x| \in \{1, 9, 18, 24, 39\}$.** Models shown are `text-davinci-002`, `text-davinci-003`, and `gpt-3.5-turbo-instruct`. Also see Figure 5.5. . . 205
D.9 **Randomness judgments across GPT models for 9 concepts** Also see Fig. 5.4. 206
D.10 Randomness Judgment $p(y = \texttt{random}|x)$ dynamics for additional concepts tested, for `text-davinci-003` (Left) and `gpt-3.5-turbo-instruct` (Right). 207
D.11 **Randomness Judgment with longer formal concepts** Similar to Fig. 5.4, we test Randomness Judgment with GPT-3.5 using longer repeating concepts. From left to right, concept lengths are: 8, 10, 12. Legends list colors corresponding to individual concepts. 208
D.12 **Random baseline for Randomness Judgment task** As a baseline to compare with our concept learning x sequences, we used a set of 20 sampled sequences $x \sim Bernoulli(.5)$. We varied $|x|$ along these individual sequences x, which enables us to consider ICL dynamics for individual sequences, represented here as lines. 209
D.13 50 sequences sampled by each GPT model, for each $p(Tails)$. Color is assigned according to specified $p(Tails)$. Red dotted lines are drawn for each $p(Tails)$. 211

D.14 50 sequences sampled by `text-davinci-003`, for each $p(Tails)$, compared with samples from Bernoulli and Window Average models fit to y_{LLM} for each $p(Tails)$. Color is assigned according to specified $p(Tails)$. 212
D.15 **Probability p(`Tails`) bias across LLMs** `text-davinci-003` and GPT-4 models are least biased relative to the specified p(`Tails`) (x-axis). In the left figure, error bars represent the maximum and minimum sequence means $\overline{y}$ for each p(`Tails`). 213
D.16 **Gambler's Fallacy histograms for ChatGPT (Top) and GPT-4 (Bottom)**. Also see Fig. 5.7. 214
D.17 **Comparing metrics of Gambler's Fallacy across probabilities and LLMs** (Left) The mean longest run for each sequence y, at each specified probability p(`Tails`), where a run is a consecutive sub-sequence of the same flip repeating multiple times in a row. (Right) The mean alternation rate for each LLM, where alternation rate is the fraction of consecutive flips that are not equal $p(y_t \neq y_{t-1})$. 215
D.18 **Distribution of unique sub-sequences for `text-davinci-003` for varying sub-sequence lengths** Also see Fig. 5.8. 216
D.19 **Distribution of unique sub-sequences for `text-davinci-003`, with additional models, varying sub-sequence lengths** MC-2, MC-5, and MC-10 are Markov Chain models fit to GPT-3.5 flips, with orders $k = \{2, 5, 10\}$ . . 217
D.20 **GPT-4 repeats the same sub-sequences more often than other GPT models** (Left) Number of unique length-10 and length-20 sub-sequences as a function of specified probability p(`Tails`), across all sequences y (note: $|y| = 50$) generated by each GPT model. (Right) The same distributions, with the y-axis normalized by dividing by the same metric (appended Gzip size, or mean Levenshtein distance) for a Bernoulli distribution centered at $\overline{y}$, to control for sequence compression being correlated with probability, e.g. with $\overline{y} = 0.99$, the same sub-sequences of only *'Tails'* flips will appear many times. 218
D.21 **GPT-generated sequences have lower complexity than Bernoulli sequences** 218

E.1 **Examples and counterfactuals:** CIFAR-10 with Box cue. 224
E.2 **Examples and counterfactuals:** CIFAR-100 with Box/Color cue. 224
E.3 **Examples and counterfactuals:** Dominoes, i.e., CIFAR-10 with their corresponding ID image from Fashion-MNIST as the cue. 225
E.4 **Learning curves for VGG-13 models.** 226
E.5 **Learning curves for ResNet-18 models.** 227
E.6 **K-Complex Outputs.** We illustrate the output range of randomized function $T_K(z)$ (Eq. E.4) for $K = 0$ (top) and $K = 4$ (bottom). Note the input $z \in \{0, 1\}$ (shown grey, red respectively) deterministically controls the output values. Thus, if $y = z$, the output $T_K(z)$ is perfectly predictive of the target label. However, inverting this function requires inference of K piece-wise linear splines to model the optimal decision boundaries (shown blue, dotted lines). Increasing the value of K makes the task consequently harder. 236

E.7 **Lack of Linear Connectivity implies Mechanistic Dissimilarity.** Plot titles denote Wasserstein-1 distance between the two models whose linear mode connectivity is being assessed. We see that models which have learned mechanisms to identify attributes of different complexity have a large Wasserstein-1 distance between their activation patterns; consequently, they cannot be linear mode connected, even after permutation of neurons. Meanwhile, models reliant on the same mechanisms have a small Wasserstein distance and can indeed be linear mode connected. For example, $\theta_{K=0}$ and $\theta_{K=0,4}$ learn the same mechanisms due to simplicity bias and can be linearly connected (see Tab. E.3), but they do not exhibit linear connectivity with $\theta_{K=4}$; meanwhile, $\theta_{K=4}$ and $\theta'_{K=4}$ can be linearly connected. Note however the latter case of more complex, i.e., $K = 4$ attribute required permutations to match the neurons for linear connectivity, while the former case of linearly separable attribute did not. This behavior emerges due to the fact that all neurons learn to be *always active* for the $K = 0$ predictive attribute–see [13, 11] for proof. 241

E.8 **VGG-13 on CIFAR-10 with Box Cue**. We plot test/train accuracy/loss curves along different connectivity paths and see thorough corroboration of our claims in the main text: Mechanistically dissimilar minimizers can be connected via nonlinear paths on a given dataset, but behave different on counterfactuals, indicating lack of mechanistic connectivity. 243

E.9 **ResNet-18 on CIFAR-10 with Box Cue**. We plot test/train accuracy/loss curves along different connectivity paths and see thorough corroboration of our claims in the main text: Mechanistically dissimilar minimizers can be connected via nonlinear paths on a given dataset, but behave different on counterfactuals, indicating lack of mechanistic connectivity. 244

E.10 **VGG-13 on CIFAR-100 with Box/Color Cue**. We plot test/train accuracy/loss curves along different connectivity paths and see thorough corroboration of our claims in the main text: Mechanistically dissimilar minimizers can be connected via nonlinear paths on a given dataset, but behave different on counterfactuals, indicating lack of mechanistic connectivity. 245

E.11 **ResNet-18 on CIFAR-100 with Box/Color Cue**. We plot test/train accuracy/loss curves along different connectivity paths and see thorough corroboration of our claims in the main text: Mechanistically dissimilar minimizers can be connected via nonlinear paths on a given dataset, but behave different on counterfactuals, indicating lack of mechanistic connectivity. 246

E.12 **VGG-13 on Dominoes**. We plot test/train accuracy/loss curves along different connectivity paths and see thorough corroboration of our claims in the main text: Mechanistically dissimilar minimizers can be connected via nonlinear paths on a given dataset, but behave different on counterfactuals, indicating lack of mechanistic connectivity. 247

E.13 **ResNet-18 on Dominoes**. We plot test/train accuracy/loss curves along different connectivity paths and see thorough corroboration of our claims in the main text: Mechanistically dissimilar minimizers can be connected via nonlinear paths on a given dataset, but behave different on counterfactuals, indicating lack of mechanistic connectivity. 248
E.14 **Fine-tuning of models trained on CIFAR-10 with Box Cue**. We plot test accuracy curves along the linear path between θ_{C} and θ_{FT} and see thorough corroboration of our claims in the main text: Linearly connected minimizers exhibit mechanistic similarity, behaving identically on counterfactual datasets, indicating mechanistic connectivity. 249
E.15 **Fine-tuning of models trained on CIFAR-100 with Box/Color Cue**. We plot test accuracy along the linear path between θ_{C} and θ_{FT} and see thorough corroboration of our claims in the main text: Linearly connected minimizers exhibit mechanistic similarity, behaving identically on counterfactual datasets, indicating mechanistic connectivity. 250
E.16 **Fine-tuning of models trained on Dominoes**. We plot test accuracy along the linear path between θ_{C} and θ_{FT} and see thorough corroboration of our claims in the main text: Linearly connected minimizers exhibit mechanistic similarity, behaving identically on counterfactual datasets, indicating mechanistic connectivity. 250
E.17 **Fine-tuning of models trained on CIFAR-10 with Box Cue**. We plot train accuracy curves along the linear path between θ_{C} and θ_{FT} and see thorough corroboration of our claims in the main text: Linearly connected minimizers exhibit mechanistic similarity, behaving identically on counterfactual datasets, indicating mechanistic connectivity. 251
E.18 **Fine-tuning of models trained on CIFAR-100 with Box/Color Cue**. We plot train accuracy curves along the linear path between θ_{C} and θ_{FT} and see thorough corroboration of our claims in the main text: Linearly connected minimizers exhibit mechanistic similarity, behaving identically on counterfactual datasets, indicating mechanistic connectivity. 251
E.19 **Fine-tuning of models trained on Dominoes**. We plot test accuracy curves along the linear path between θ_{C} and θ_{FT} and see thorough corroboration of our claims in the main text: Linearly connected minimizers exhibit mechanistic similarity, behaving identically on counterfactual datasets, indicating mechanistic connectivity. 252
E.20 **Fine-tuning of models trained on CIFAR-10 with Box Cue**. We plot test loss curves along the linear path between θ_{C} and θ_{FT} and see thorough corroboration of our claims in the main text: Linearly connected minimizers exhibit mechanistic similarity, behaving identically on counterfactual datasets, indicating mechanistic connectivity. 252
E.21 **Fine-tuning of models trained on CIFAR-100 with Box / Color Cue**. We plot test loss curves along the linear path between θ_{C} and θ_{FT} and see thorough corroboration of our claims in the main text: Linearly connected minimizers exhibit mechanistic similarity, behaving identically on counterfactual datasets, indicating mechanistic connectivity. 253

E.22 **Fine-tuning of models trained on Dominoes**. We plot test loss curves along the linear path between θ_C and θ_{FT} and see thorough corroboration of our claims in the main text: Linearly connected minimizers exhibit mechanistic similarity, behaving identically on counterfactual datasets, indicating mechanistic connectivity. 253
E.23 **Fine-tuning of models trained on CIFAR-10 with Box Cue**. We plot train loss curves along the linear path between θ_C and θ_{FT} and see thorough corroboration of our claims in the main text: Linearly connected minimizers exhibit mechanistic similarity, behaving identically on counterfactual datasets, indicating mechanistic connectivity. 254
E.24 **Fine-tuning of models trained on CIFAR-100 with Box / Color Cue**. We plot train loss curves along the linear path between θ_C and θ_{FT} and see thorough corroboration of our claims in the main text: Linearly connected minimizers exhibit mechanistic similarity, behaving identically on counterfactual datasets, indicating mechanistic connectivity. 254
E.25 **Fine-tuning of models trained on Dominoes**. We plot train loss curves along the linear path between θ_C and θ_{FT} and see thorough corroboration of our claims in the main text: Linearly connected minimizers exhibit mechanistic similarity, behaving identically on counterfactual datasets, indicating mechanistic connectivity. 255

F.1 **Exemplar for Counter Task:** A sample used for fine-tuning Tracr compiled models on counting 'b'. 258
F.2 **Exemplar for Max-Element Task:** A sample used for fine-tuning Tracr compiled models on the Max identifier task. 259
F.3 **PCFG setup:** Grammar rules considered to generate the PCFG dataset. The highlighted token represents the parent token. These rules have been adapted from prior work [12]. 259
F.4 **PCFG Exemplar.** A representative sample from the PCFG dataset [12] . . 261
F.5 **Distribution of the class labels for Counting (first row) and Index of occurrence tasks (second row).** (a) shows the distribution for the operand token `a` and (b) shows the same for the operand token `b`. The data is similarly distributed across different classes and the distribution shift for the two operands and the different values of $C_{\texttt{Tr}}$ is small. 262
F.6 **TinyStories Exemplar.** An example from the TinyStories Instruct dataset [14]. 262
F.7 **Effect of different sampling probabilities of pre-training target token $\texttt{O}_{\texttt{PT}}$ on fine-tuning task's performance.** We observe similar gains for different values of sampling probabilities of $\texttt{O}_{\texttt{PT}}$ during fine-tuning. 266
F.8 **Effect of different sampling probabilities of pre-training target token $\texttt{O}_{\texttt{PT}}$ on pre-training task's performance.** We observe a higher loss in performance if low sampling probability is used for sampling the pre-training target token $\texttt{O}_{\texttt{PT}}$ during fine-tuning. 266

F.9 **Effect of different sampling probabilities of pre-training target token $\texttt{O}_{\texttt{PT}}$ on fine-tuning task's performance.** Pre-training is done using high sampling prior for fine-tuning task family token. We observe similar gains for different values of sampling probabilities of $\texttt{O}_{\texttt{PT}}$ during fine-tunning. 267
F.10 **Effect of different sampling probabilities of pre-training target token $\texttt{O}_{\texttt{PT}}$ on pre-tuning task's performance.** Pre-training is done using high sampling prior for fine-tuning task family token. We observe similar gains for different values of sampling probabilities of $\texttt{O}_{\texttt{PT}}$ during fine-tunning. 267
F.11 **Probing analysis** corresponding to Fig-F.7 and F.8 267
F.12 **Probing analysis** corresponding to Fig-F.9 and F.10 268
F.13 **Jailbreaking analysis using PCFG.** We report performance on the pre-training task (counting $\texttt{O}_{\texttt{PT}}$) as a function of fine-tuning iterations, where the fine-tuning task (counting $\texttt{O}_{\texttt{FT}}$) is performed using the task family token $\texttt{T}_{\texttt{NJ}}$. We find that the model is able to learn the fine-tuning task and seemingly performs poorly on the pretraining task when task family token $\texttt{T}_{\texttt{NJ}}$ is used in the input. However, in presence of a sufficiently relevant capability (high pretraining prior for $\texttt{O}_{\texttt{FT}}$), using task family tokens $\texttt{T}_{\texttt{J}_1}$ or $\texttt{T}_{\texttt{J}_2}$ in the input shows the model can still perform the pretraining task perfectly—i.e., we can jailbreak the model. 269
F.14 **Probing analysis for the setup used to understand jail-breaking.** Similar results on using the fine-tuning token or the jailbreaking token for training the probe indicate that the pre-training capabilities are not removed on fine-tuning. 270
F.15 **Reverse Fine-Tuning on PCFGs:** We set $C_{\texttt{Te}}$ to be 0 to test if the model performs well regardless of a spurious correlation. We observe that when a strongly relevant capability is present (a, b), the model very quickly (0.1–1K iterations) starts to perform well on the task via `reFT` , even if behavior relevant to the capability ceased during pretraining (e.g., when $C_{\texttt{Tr}}$ is 1). Meanwhile, when the model possessesses a weakly relevant capability (c), this "revival" is *slightly* slower (3K iterations). In contrast, the $\texttt{Scr.} + \texttt{FT}$ baseline only reaches perfect accuracy at 4.5K iterations and when using a larger learning rate η_M. 271
F.16 **Reverse Fine-Tuning on Tracr:** We set $C_{\texttt{Te}}$ to be 0 to test if the model performs well regardless of a spurious correlation. We observe that the fine-tuned model upon `reFT` very quickly starts starts to perform well on the pretraining task. Moreover, the protocol works even if an extremely small learing rate is used. In contrast, the $\texttt{Scr.} + \texttt{FT}$ baseline only reaches a large learning rate η_M is used, and does so less sample efficiently. We note that the results for η_M learning rate look worse than the η_S learning rate around 10^3 iterations because η_M is too big of a learning rate, forcing the model to essentially go through a "retraining" phase. 272
F.17 **Reverse fine-tuning a model fine-tuned to remove its pretraining capability.** See text in Sec. F.5.4 for details. 274

F.18 **Reverse fine-tuning performance on using `randFT` fine-tuning protocol to forget the pre-training capability.** We follow the setup of Fig. 7.9 and plot results for several sampling priors of the target token for fine-tuning, i.e., $\texttt{O}_{\texttt{FT}}$, but we use `randFT` for fine-tuning the models before `reFT` . The baseline results `Scr.` + `FT` are copied from the Fig. F.17, i.e., baseline is not trained in an "adversarial" way, but the `randFT` results are. While this makes the baseline unfairly stronger, we find `reFT` the `randFT` models are still more sample efficient. 274

F.19 **Probing analysis of `randFT` fine-tuning protocol.** We plot probe accuracy versus the index of the block in the Transformer model. $C_{\texttt{Te}}$ is set to 0. The pretrained model (left) acts as a baseline for the trend of performance through the model's blocks. In most scenarios, we find we can infer the count of $\texttt{O}_{\texttt{PT}}$ with a similar trend as the pretrained model (left). A drop in performance is observed only when learning rate η_M is used with a weakly relevant capability (low sampling prior). This indicates pretraining capabilities continues to persist upon fine-tuning. 275

F.20 **Larger learning rates lead to more pronounced loss of modelling capability.** The plots show loss on data with the Twist feature present while fine-tuning to delete the capability to model text with the *Twist* feature, for different learning rates and fine-tuning protocols. 279

F.21 **Larger learning rates lead to more pronounced loss of generative capability.** The plots show the generation score for the Twist feature present while fine-tuning to delete the capability to model text with the *Twist* feature, for different learning rates and fine-tuning protocols. 279

F.22 `reFT` **easily recovers deleted capabilities**. We plot loss on data with the `Twist` for `reFT` of various models fine-tuned to delete the capability, as well as a control model which was pre-trained without data with `Twists`. The fine-tuned models learn the capability more sample-efficiently, and additionally converge to a lower loss than the control model. 280

F.23 `reFT` **easily recovers deleted generative capabilities**. We plot the generation scores for the `Twist` feature for `reFT` of various models fine-tuned to delete the capability, as well as a control model which was pre-trained without data with `Twists`. The fine-tuned models learn the capability much more sample-efficiently, and additinoally converge to a lower loss, than the control model. 280

F.24 **Probing the presence of capabilities in TinyStories Models.** We plot probe accuracy of classifying whether a story contains a `Twist` or not wrt. the layer of the Transformer model (similarly to Fig. 7.8). Accuracy on models pre-trained with or without `Twist` data (*Present/Not in Pretraining* respectively) act as upper and lower bounds on the expected accuracy of the probes, and are plotted on both LR figures for ease of comparison, although they do not use a fine-tuning learning rate. We find that regardless of fine-tuning protocol (Filtering, Filtering + Randomisation, Filtering + Mix & Match), for the lower LR no fine-tuning protocol removes a meaningful amount of information from the activations, and a similar but less strong trend holds for the higher LR, implying that the pre-trained model retains its capability of story identification (a necessary capability for story modelling) throughout fine-tuning. Identical to Fig. 7.11 281
F.25 **Probing the presence of capabilities in TinyStories Models.** We plot probe accuracy of classifying whether a story contains the Foreshadowing feature or not wrt. the layer of the Transformer model. All other details the same as Fig. F.24 282
F.26 **Probing the presence of capabilities in TinyStories Models.** We plot probe accuracy of classifying whether a story contains the Moral Value feature or not wrt. the layer of the Transformer model. All other details the same as Fig. F.24 282
F.27 **Probing the presence of capabilities in TinyStories Models.** We plot probe accuracy of classifying whether a story contains the Bad Ending feature or not wrt. the layer of the Transformer model. All other details the same as Fig. F.24 283
F.28 **Counter Task: Detailed visualization of the capability revival analysis on Tracr compiled for the Counter task. Observation:** On using η_{VS}, we are able to revive the compiled capability of the Tracr model fine-tuned with η_M, whereas using η_L for fine-tuning hampers the underlying capability of the model and therefore there is no revival seen. 284
F.29 **Counter Task: Detailed visualization of the self attention layer in Block-1 of the Tracr fine-tuned model. Observation:** (a) shows the map for the compiled Tracr model, where we observe that `a`'s are being attended by the second token in the input sequence. (b) On using η_S for fine-tuning in the absence of spurious correlations ($C_{\texttt{Te}} = 0$), `a`'s are still being attended in the attention map. (c) On using η_M, the model learns the fine-tuning task as `b`'s are also being attended now. On fine-tuning in the presence of the spurious correlation ($C_{\texttt{Te}} = 1$), the model doesn't learn to attend `b`'s, but rather learns the spurious correlation. 284

F.30 **Counter Task: Validation of Tracr observations on Counter task using three different input samples.** The rows represents different input samples. **Observation:** Using η_S for fine-tuning the Tracr model compiled on Counter task is unable to learn to attend `b`'s (a). But the model learns the spurious correlation. On the other hand, on using η_M the model is able to learn the fine-tuning task by attending to `b`'s. But in the presence of the spurious correlation $C_{\texttt{Tr}} = 1$ the model still doesn't learn to attend `b`'s. . . . 285
F.31 **New capabilities are not learned on using η_S for fine-tuning.** (a) The counting `a`'s capability implemented by the originally compiled Tracr model is to attend to `a`'s. On using η_S (10^{-3}) for fine-tuning, the compiled model is not able to learn the capability of counting `b`'s (b). Increasing the learning rate makes the model learn to attend `b`'s (c), but the pretraining capability of attending to `a`'s still exists. 287
F.32 **Capability Revival Analysis:** Using η_{VS} (a) is able to recover the old capability on reverse fine-tuning the model fine-tuned with η_M. But η_S is not able to recover the original capability, when the compiled model is fine-tuned with η_l. This is because using a large value of learning rate during=fine-tuning hampers the pre-training capabilities. 287
F.33 **Learning of the fine-tuning capability is affected by type of compiled capability present in the model.** (a) The Tracr program implements the sorting function in Block-0 and read function in Block-2. Using η_M and η_S can destroy the sorting capability present in Block-1 (b). But using η_{VS}, preserves the sorting capability (c). Thus on using η_{VS}, the model learns to read a different stream of output, while preserving the sorting capability. . . 287
F.34 **Counter Task: Visualization of the attention maps of the first and second blocks of the Tracr fine-tuned models.** (a) shows the analysis when the spurious correlation is not present in the fine-tuning datatset, whereas in case of (b) the spurious correlation is present in the fine-tuning dataset. The first row shows the maps for the Tracr compiled model and other rows shows the analysis for different learning rates. **Observation:** (a) Using η_S or η_{VS} the model is not able to learn to attend `b`'s and thus the fine-tuning task performance is poor. Whereas using η_M the model is able to learn to attend to `b`'s, however the capability to count `a`'s is likely still present since the model still attends to `a`'s. Further increasing the learning rate leads to distortion of the compiled capabilities, and thus model learns the fine-tuning task by learning a different capability. (b) In the presence of spurious correlation, even for large learning rate the compiled capability is still present, since the model attends to `a`'s. 289
F.35 **Counter Task: Visualization of the activated output of the first MLP layer in first and second blocks.** The visualization is shown only for channel numbers 80-100. **Observation:** Using η_M preserves the Tracr compiled capability, while also learning the fine-tuning task. This shows that the model changes behaviourally but mechanistically the compiled capabilities are still present in it. 290

F.36 **Counter Task: Visualization of the activated output of the first MLP layer in first and second blocks.** This is the complete visualization of the activation map presented in Fig. F.35. 293
F.37 **Counter Task: Effect of the presence of the spuriously correlated data-points in different fractions ($C_{\texttt{Tr}}$) in the fine-tuning dataset.** The Tracr compiled model with the capability to count `a`'s is fine-tuned to count `b`'s on different values of $C_{\texttt{Tr}}$. η_M is used for fine-tuning. **Observation:** On increasing the value of $C_{\texttt{Tr}}$, the model gives lower attention to `b`'s and in case of $C_{\texttt{Tr}} = 1$, almost no attention is given by the model to `b`'s. 293
F.38 **Counter Task: Validation of Tracr observations on Counter task on reverse fine-tuning on three different input samples.** The rows represents different input samples. **Observation:** Capability revival is possible on using η_M for fine-tuning but not on using η_L. 294
F.39 **Max Identifier Task: Validation of Tracr observations on max identifier task on three different input samples.** The rows represents different input samples. **Observation:** Using η_{VS} preserves the original Tracr capabilities and therefore performs well on the fine-tuning task, whereas using η_M distorts the compiled capabilities resulting in poor performance on the fine-tuning task. 295
F.40 **Max Identifier Task: Validation of Tracr observations on max identifier task with the spurious correlation defined as the difference between the indices of fifth and seventh maximum elements being three.** The rows represents different input samples. **Observation:** Using η_{VS} preserves the original Tracr capabilities and therefore performs well on the fine-tuning task, whereas using η_M distorts the compiled capabilities resulting in poor performance on the fine-tuning task. 296
F.41 **Max Identifier Task: Visualization of the attention maps of the zeroth and second blocks of the Tracr fine-tuned models on the max identifier task.** (a) shows the analysis when the spurious correlation is not present in the fine-tuning datatset, whereas in case of (b) the spurious correlation is present in the fine-tuning dataset. The first row shows the maps for the Tracr compiled model and other rows shows the analysis for different learning rates. **Observation:** Using η_L, η_M or η_S for fine-tuning distorts the capability of the programmed Tracr model in the Block-0 and as a result the Block-2 attention map is not able to attend to the desired output token. Whereas using η_{VS} is able to preserve the capability and as a result the fine-tuned model is able to attend to the correct token in the attention map in Block-2. 297
F.42 **Max Identifier Task: Visualization of the activated output of the first MLP layer in first and second blocks for the max identifier task.** The visualization is shown only for channel numbers 50-70. **Observation:** Using η_{VS} for fine-tuning, which enables the model to learn the fine-tuning task, preserves the Tracr compiled model's compiled capability of sorting tokens in Block-1. Whereas other learning rates are not able to preserve this capability. 298

F.43 **Max Identifier Task: Visualization of the activated output of the first MLP layer in first and second blocks.** This is the complete visualization of the activation map presented in Fig. F.42. 299
F.44 **Counter Task: Pruning evaluation on Tracr model fine-tuned to count b's. Observation:** Higher value of $C_{\texttt{Tr}}$ leads to the learning of the wrapper on top of the Tracr compiled capability. This wrapper is learned on using η_M and η_S and can be localized in a few weights of the model. 300
F.45 **Counter Task: Pruning evaluation on Tracr model fine-tuned to count b's. Observation:** Observations are consistent with Fig-7.7. 300
F.46 **Counter Task, weight pruning: Pruning weights of the Tracr model fine-tuned to count b's using different learning rates. Observation:** In the presence of spurious correlation the model learns a wrapper when learning rates η_M and η_S are used. This wrapper can be localized in a few weights of the model. 301
F.47 **Counter Task, neuron pruning: Pruning neurons of the Tracr model fine-tuned to count b's using different learning rates. Observation:** In the presence of spurious correlation the model learns a wrapper in case of η_M and η_S. This wrapper can be localized in a few weights of the model. . . 301
F.48 **Counter Task: Effect of pruning a single weight on the distribution of predicted class labels. Observation:** Even after pruning, the model predicts different classes indicating that the gain in accuracy on pruning is indeed because the model has removed the wrapper. 302
F.49 **Counter Task,** $n_{iters} = 200K$**,** $C_{\texttt{Te}} = 0$**:** Effect of learning rate (LR) on fine-tuning pre-trained models with weakly and strongly relevant capabilities and using different values of $C_{\texttt{Tr}}$ for fine-tuning. **Observation:** In the presence of strongly relevant capability, training with η_S yields good performance on the fine-tuning dataset. The convergence time to learn the fine-tuning task increases with an increase in $C_{\texttt{Tr}}$. 304
F.50 **Index of Occurrence Task,** $n_{iters} = 200K$**,** $C_{\texttt{Te}} = 0$**.** The settings are consistent with Fig. F.49 309
F.51 **Counter Task,** $n_{iters} = 200K$**:** Effect of the presence of strongly or weakly relevant pretuning capability on fine-tuning performance on using η_M and η_S. Test sets with and without the spurious correlations are used for evaluation. **Observation:** The convergence time for learning the fine-tuning task in the absence of strongly relevant capability is higher as compared to when the strongly relevant is present in the model. The time further increases if spurious correlations are present in the fine-tuning set. However, in the presence of spurious correlations, the convergence time to learn the spurious correlation is small and is possible even on using the learning rate η_S. Using η_S is unable to yield learning of the fine-tuning task if a weakly relevant capability is present in the model. 310
F.52 **Index of Occurrence Task,** $n_{iters} = 200K$**:** The settings are consistent with Fig. F.51. 311

F.53 **Counter Task,** $n_{iters} = 50K$**:** Effect of the presence of strongly or weakly relevant pretuning capability on fine-tuning performance on using η_M and η_S. Test sets with and without the spurious correlations are used for evaluation. The observations are consistent with Fig. F.51. 312
F.54 **Index of Occurrence Task,** $n_{iters} = 50K$**:** The settings are consistent with Fig. F.53. 313
F.55 **Counter Task,** $n_{iters} = 10K$**:** Effect of the presence of strongly or weakly relevant pretuning capability on fine-tuning performance on using η_M and η_S. Test sets with and without the spurious correlations are used for evaluation. The observations are consistent with Fig. F.51. 314
F.56 **Index of Occurrence Task,** $n_{iters} = 10K$**:** The settings are consistent with Fig. F.55. 315
F.57 **Counter Task,** η_M**,** $C_{\texttt{Te}} = 0$ **:** Learning of the wrapper in presence of different fraction of spuriously correlated data, values of $\mathcal{P}_{\texttt{T}}(\texttt{a})$ during pre-training, and training iterations. **Observation:** Using a higher fraction of spuriously correlated data in the fine-tuning set (higher value of $C_{\texttt{Tr}}$) leads to faster degradation in the pre-training accuracy. Further this degradation is even faster in presence of weakly relevant capability. 315
F.58 **Index of Occurrence Task, Medium LR,** $C_{\texttt{Te}} = 0$**:** The settings are consistent with Fig. F.57. 316
F.59 **Counter Task,** η_M**,** $C_{\texttt{Te}} = 1$ **:** Learning of the wrapper in presence of different fraction of spuriously correlated data, values of $\mathcal{P}_{\texttt{T}}(\texttt{a})$ during pre-training, and training iterations. **Observation:** Using a higher fraction of spuriously correlated data in the fine-tuning set (higher value of $C_{\texttt{Tr}}$) leads to faster degradation in the pre-training accuracy. Further this degradation is even faster in presence of weakly relevant capability. 316
F.60 **Index of Occurrence Task, Medium LR,** $C_{\texttt{Te}} = 1$ **:** The settings are consistent with Fig. F.59. 317
F.61 **Counter Task,** η_S**,** $C_{\texttt{Te}} = 0$ **:** Learning of the wrapper in presence of different fraction of spuriously correlated data, values of $\mathcal{P}_{\texttt{T}}(\texttt{a})$ during pre-training, and training iterations. **Observation:** The observations are consistent with Fig. F.57 317
F.62 **Index of Occurrence Task,** η_S**,** $C_{\texttt{Te}} = 0$ **:** The settings are consistent with Fig. F.61 318
F.63 **Counter Task,** η_S**,** $C_{\texttt{Te}} = 1$ **:** Learning of the wrapper in presence of different fraction of spuriously correlated data, values of $\mathcal{P}_{\texttt{T}}(\texttt{a})$ during pre-training, and training iterations. **Observation:** The settings are consistent with Fig. F.59 318
F.64 **Index of Occurrence Task,** η_S**,** $C_{\texttt{Te}} = 1$ **:** The settings are consistent with Fig. F.63 319

F.65 **Counter Task, $n_{iters} = 200K$: Reverse Fine-tuning** on weakly and strongly relevant capability fine-tuned models. Medium and small learning rates are used for reverse fine-tuning in the presence of different degrees of spuriously correlated data-points present in the train-set. The fine-tuned model was fine-tuned using Large LR. **Observation:** When the model possesses weakly relevant capability, the convergence time is lower for models fine-tuned on dataset with spurious correlations. If the model possesses strongly relevant capability, this difference is less. The "revival" of pre-training capability is observed for all values of $C_{\mathtt{Tr}}$. Even though fine-tuning was done using a large learning rate of 10^{-4}, capability revival is possible even on using a small learning rate. 320
F.66 **Counter Task, $n_{iters} = 50K$: Reverse Fine-tuning** on weakly and strongly relevant capability fine-tuned models. η_M and η_S are used for capability reverse fine-tuning in the presence of different fraction of spuriously correlated data-points present in the train-set. The fine-tuned model was fine-tuned using Large LR. **Observation:** The observations are consistent with Fig. F.65 321
F.67 **Counter Task: Reverse fine-tuning analysis for different pre-training iterations. Observation:** Capability Revival is seen for models pre-trained with different number of iterations. 322
F.68 **Counter task, $\mathcal{P}_{\mathtt{T}}(\mathtt{a}) = 0.999$, Pruning Analysis:** Revival of pre-training capability analysis for different learning rates, pre-training iterations and different values of $C_{\mathtt{Tr}}$. **Observation:** (a) On using η_L the model learns the wrapper only when fine-tuning on $C_{\mathtt{Te}} = 1$. This wrapper is learned only on fine-tuning set with the spurious correlations . (b) On using η_M the model learns the wrapper on fine-tuning with smaller values of $C_{\mathtt{Te}}$ as well. However, still this wrapper is learned only on fine-tuning set with the spurious correlations. (c) On using η_S the model learns the wrapper for all values of $C_{\mathtt{Te}}$ and for all the data samples. 323
F.69 **Index of Occurrence task, $\mathcal{P}_{\mathtt{T}}(\mathtt{a}) = 0.999$, Pruning Analysis:** The settings are consistent with Fig. F.68 324
F.70 **Index of Occurrence task, $\mathcal{P}_{\mathtt{T}}(\mathtt{a}) = 0.999$, Pruning Analysis:** The settings are consistent with Fig. 7.7 324
F.71 **Counter task, $n_{iters} = 200K$, Pruning Analysis:** Revival of pre-training capability analysis for different learning rates, weakly and strongly relevant capability fine-tuned models, and different values of $C_{\mathtt{Tr}}$. **Observation:** Learning of the wrapper is possible for weakly as well as strongly relevant capability pre-trained models. 325
F.72 **Index of Occurrence task, $n_{iters} = 200K$, Pruning Analysis:** The settings are consistent with Fig. F.71 326
F.73 **Counter task, $n_{iters} = 200K$ Pruning Analysis:** Revival of pre-training capability analysis for different learning rates, weakly and strongly relevant capability fine-tuned models and different values of $C_{\mathtt{Tr}}$. Here larger number of neurons are pruned as compared to Fig. F.71. 327
F.74 **Index of Occurrence task, $n_{iters} = 200K$, Pruning Analysis:** Here larger number of neurons are pruned. The settings are consistent with Fig. F.73 . 328

F.75 **Counter task,** η_L, $C_{\texttt{Te}} = 0$, **Pruning analysis:** Effect of strongly and weakly relevant capabilities for different number of pre-training iterations and different values of $C_{\texttt{Tr}}$ **Observation:** Fine-tuning a model with strongly relevant capability leads to learning of an "inhibitor" on its pre-training capability, i.e., a wrapper that disallows use of the pretraining capability. Revival of the pre-training capability is partly possible on pruning, if the model has strongly relevant capability and it was fine-tuned on dataset without spurious correlations.The inhibitor is mainly learned for the $200K$ iteration pre-trained model. 329
F.76 **Index of Occurrence task,** η_L, $C_{\texttt{Te}} = 0$, **Pruning analysis:** The settings are consistent with Fig. F.75 330
F.77 **Counter task,** $n_{iters} = 200K$, $C_{\texttt{Te}} = 0$, **Pruning analysis:** Effect of the strongly and weakly relevant capabilities for different number of pre-training iterations and different values of fraction of spurious correlations present in the fine-tuning dataset. **Observation:** Learning of the inhibitor is observed on using η_L. 331
F.78 **Index of Occurrence task,** $n_{iters} = 200K$, $C_{\texttt{Te}} = 0$ **Pruning analysis:** the settings are consistent with Fig. F.77 332
F.79 **Counter task,** $n_{iters} = 200K$, $C_{\texttt{Te}} = 0$, **Probing analysis:** The effect of different values of learning rate, weakly and strongly relevant capabilities is shown. **Observation:** Using η_L hampers the pre-training capability to count `a` especially when the probed model has a weakly relevant capability. The performance on the pre-training task of counting `a`'s continues to be high, especially with η_S. The accuracy of counting `b`'s shows that fine-tuning capability is learned on using η_L. On using η_S, models with weakly relevant capabilities are not able to learn the fine-tuning capability well. 333
F.80 **Counter task,** $n_{iters} = 50K$, $C_{\texttt{Te}} = 0$, **Probing analysis:** The observations are consistent with Fig. F.79 334
F.81 **Index of Occurrence task,** $n_{iters} = 50K$, $C_{\texttt{Te}} = 0$, **Probing analysis:** The settings are consistent with Fig. F.80 335
F.82 **Index of Occurrence task,** $n_{iters} = 200K$, $C_{\texttt{Te}} = 0$, **Probing analysis:** The settings are consistent with Fig. 7.8 335
F.83 **Counter task,** $n_{iters} = 200K$, $C_{\texttt{Te}} = 0$, **Probing analysis. Observation:** With an increase in $C_{\texttt{Tr}}$, the accuracy on counting `a`'s also increases for both weakly as well as strongly relevant capability models. 336
F.84 **Index of Occurrence task,** $n_{iters} = 200K$, $C_{\texttt{Te}} = 0$, **Probing analysis:** The settings are consistent with Fig. F.83 337
F.85 **Counter task,** $n_{iters} = 50K$, $C_{\texttt{Te}} = 0$, **Probing analysis. Observation:** With an increase in $C_{\texttt{Tr}}$, the accuracy on counting `b`'s also decreases for both weakly as well as strongly relevant capability models. 338
F.86 **Index of Occurrence task,** $n_{iters} = 50K$, $C_{\texttt{Te}} = 0$, **Probing analysis:** The settings are consistent with Fig. F.85 339

LIST OF TABLES

TABLE

6.1 **Evaluating CBFT.** We train ResNet-18 models on our synthetic CIFAR-10, CIFAR-100, and Dominoes dataset with different proportions of samples with cue features and fine-tune them using 2500 "clean" samples from a dataset without any cues. Test accuracies (%) on counterfactual test datasets with No Cue (NC), with Cue (C), Randomized Cue (RC), and Randomized Image (RI) are reported (mean of three seeds). We compare our method, Connectivity-Based Fine-Tuning (CBFT), with several baselines: Fine-tuning with a medium/small learning rate ($FT_{M/S}$), LLR [15], and LPFT [16]. $\sim$ denotes invariance is desirable, i.e., accuracy should be similar to that on NC; $\uparrow/\downarrow$ indicate higher/lower accuracy is desirable; best results are in bold. We generally see that all baselines yield large degradations in the absence of cues, and even achieve very high accuracy when the underlying image is randomized. Meanwhile, CBFT is able to break reliance on cues, inducing representations that are completely invariant to their presence. 95

6.2 **Summarizing our Findings.** ✓, ✗ respectively indicate whether there always exist paths along which mechanistically (dis)similar models identified using gradient-based optimization can exhibit the type of connectivity specified in the column title. * denotes there are exceptional, but primarily theoretical, cases where the connectivity definition can hold (see App. E.6). 96

7.1 **TinyStories `reFT` Analysis**. We report the percent of generations with a `twist` during reverse fine-tuning for the twist capability. `F`, `R`, and `MM` stand for our three fine-tuning protocols: Filtering, Randomisation and Mix & Match (see App. F.6 for details). Regardless of learning rate and protocol, models relearn to generate stories with `twist` more sample-efficiently than the control model pre-trained on data w/o twists and fine-tuned to generate them (Not in `PT`). 113

E.1 **Ablating CBFT.** We train ResNet-18 models on our synthetic CIFAR-10, CIFAR-100, and Dominoes dataset with different proportions of samples with cue features and fine-tune them using 2500 "clean" samples from a dataset without any cues. Test accuracies (%) on counterfactual test datasets with No Cue (NC), with Cue (C), Randomized Cue (RC), and Randomized Image (RI) are reported. We compare Connectivity-Based Fine-Tuning (CBFT) with two of its ablations (see App. E.5): (i) $-\mathcal{L}_{\text{barrier}}$, for which the barrier inducing loss is removed from the training process and (ii) $-\mathcal{L}_{\text{Inv.}}$, for which the invariance loss is removed. $\sim$ denotes invariance is desirable, i.e., accuracy should be similar to that on NC; $\uparrow/\downarrow$ indicate higher/lower accuracy is desirable; best results are in bold. For discussion, please see App. E.5. 231

E.2 **Training from scratch on minimal clean data.** We train ResNet-18 models on the 2500 "clean" samples used in Tab. 6.1 from the original CIFAR-10, CIFAR-100, and Dominoes datasets. Test accuracies (%) on counterfactual test datasets with No Cue (NC), with Cue (C), Randomized Cue (RC), and Randomized Image (RI) are reported. $\sim$ denotes invariance is desirable, i.e., accuracy should be similar to that on NC; $\uparrow/\downarrow$ indicate higher/lower accuracy is desirable. 231

E.3 **Illustrating Simplicity Bias.** We train models on a dataset with predictive attributes of complexities 0 and 4. Column titles indicate which attributes were allowed to remain predictive during training, i.e., were not randomized via interventions: e.g., $K_1 = 0$ implies only the linearly separable attribute is predictive in the training data. Rows report difference in loss on a test dataset $\mathcal{D}_{K_1}$ which contains attributes of complexity K_1 and another test dataset $\mathcal{D}_{K_2}$ which contains attributes of complexity K_2. Results are computed up to 4 digits of precision and averaged over 3 seeds. We see models trained on data with both predictive attributes behave similarly to models trained on $K = 0$ attribute only; that is, they are invariant to the more complex attribute for which $K = 4$. 239

F.1 Notations used in this work. 257

F.2 **Results on counting and max element task Tracr setups.** The evaluation is done on test sets with and without the spurious correlation. The Tracr compiled models are fine-tuned for different learning rates and different value of $C_{\texttt{Tr}}$. 291

F.3 **Results on counting task Tracr for reverse fine-tuning with different learning rates.** Fine-tuning was done using η_M. The evaluation is done on test sets with and without the spurious correlation. The Tracr compiled models are fine-tuned for different learning rates and different value of $C_{\texttt{Tr}}$. . 292

F.4 **Results on the PCFG counting task**: 200K iterations of pre-training. . . 305

F.5 **Results on the PCFG counting task**: 50K iterations of pre-training. . . 306

F.6 **Results on the PCFG index of occurrence task**: 200K iterations of pre-training. 307

F.7 **Results on the PCFG index of occurrence task**: 50K iterations of pre-training. 308

LIST OF APPENDICES

A The Multiplicative Emergence Hypothesis **119**
Experimental details . . . 119
Additional experimental results . . . 123

B A Percolation Model of Emergence **128**
Data-Generating Process: Defining our formal language . . . 128
Percolation threshold in the bipartite graph setup . . . 134
Experimental details . . . 138
Further Results: Robustness Across Settings and Evaluations . . . 139
Scaling of Point of Emergence / Transition Point . . . 163
Alternative Analysis of Scaling of Transition Point . . . 176

C Competence Emerges Before Performance **182**
Experimental Details . . . 182
Details on Alternative Protocols for Eliciting Model Capabilities . . . 184
Additional Results . . . 185

D In-Context Learning Dynamics via Random Binary Sequences **192**
Related Work . . . 192
Additional Experimental Details . . . 193
Cognitive Interpretability . . . 196
Bayesian Model Selection Example . . . 198
Formal concept learning is not merely local bias . . . 200
Formal Language Learning with Varying Prediction Depth . . . 202
Randomness Judgments . . . 206
Random Sequence Generation by GPT Model . . . 210
Gambler's Fallacy Metrics by GPT Model . . . 214
Memorization, Compression, and Complexity . . . 216
Background on Algorithmic and Subjective Randomness . . . 220

E Mechanistic Mode Connectivity **222**
Detailed Related Work . . . 222
Training Details and Datasets . . . 223
Quadratic Connectivity Paths and Matching Permutations . . . 228

A Note on Difference Between Permutation and other Architectural Symmetries in the context of mode connectivity 229
Ablation Experiments on CBFT 230
Deferred Proofs 232
Further Results: Non-Linear Connectivity of Mechanistically Dissimilar Minimizers 242
Further Results: Lack of Linear Connectivity implies Mechanistic Dissimilarity . . 243

F Mechanistic Analysis of Fine-Tuning on Procedurally Defined Tasks 256
Organization of Appendix 256
Additional details on Datasets 258
Details on Training and Evaluation 261
Mechanistic Interpretability Tools Setup 264
Additional results 266
Details and Results on Tinystories Experiments 276
Additional Tracr Results 284
Additional PCFG results 303

ABSTRACT

Increasingly powerful and general-purpose AI systems have found their way into our daily lives. As these systems spread to high-stakes applications, ensuring their safe deployment has become crucial: we must ensure these systems augment and benefit our society, instead of becoming an active source of harm. To this end, recent regulatory work has sought to define standards for identifying and mitigating vulnerabilities in AI driven applications. These works have grounded themselves in frameworks of *risk regulation*, motivated by the fact that establishing causality in AI-caused harms can be difficult. The goal of this dissertation is to challenge this design choice. Over three broad parts, the presented work argues that peculiarities of modern neural networks, such as their increasingly open-ended nature, lead to loopholes in off-the-shelf use of risk regulation as a framework for ensuring development of safe AI. Contributions of each part are summarized as follows.

(i) Risk regulation assumes one can preemptively list harms expected from a system, allowing design of protocols to monitor them. In the first part, we present several empirical and formal models that demonstrate the unpredictable nature of neural network capabilities, showing that they can suddenly emerge and enable a network to perform tasks it was not intended for. Specifically, we show emergent learning either occurs when general structures underlying the data-generating process are learned by the model, hence accelerating learning of narrower tasks, or when a task is compositional in nature and capabilities relevant to performing the composition are learned by the model. Such unpredictability renders preemptive expectations of risks infeasible.

(ii) Risk assessment requires well-defined evaluations. In the second part, we analyze two challenges to this goal. First, we show that when evaluating compositional capabilities, models exhibit an intriguing phenomenon wherein much before standard benchmarking helps claim the model possesses a capability, there exist latent interventions that can force the model to generate the desired output. Such capabilities can therefore evade risk assessment evaluations. Second, we analyze how minor input modifications can significantly alter a model's behavior, hence complicating the establishment of safe use standards. We formalize this problem as input underspecification and analyze the

mechanisms used by a model for inferring which solution, among the spectrum of valid ones, should be used to respond to an input. We show evidence suggesting large-scale models engage in a Bayesian selection protocol, i.e., minimal input changes that alter posterior probability of a solution can completely change these models' output.

(iii) Fine-tuning protocols are the current de-facto strategy for mitigating vulnerabilities identified in a neural network. In the final part, we identify limitations in these protocols, revealing they learn minimalistic "wrappers" over base capabilities of a model and hence do not adequately suppress undesirable behaviors outside of the fine-tuning data distribution.

Overall, the contributions of this dissertation suggest regulation of AI systems requires exploration of novel and more nuanced paradigms that go beyond mere risk regulation. This can involve intermixing several viable frameworks, e.g., liability based models and tort law, to define backstops when risk regulation fails to adequately foresee harms possible from a system.

CHAPTER 1

Introduction

Intelligence is not a privilege, it's a gift.
And you use it for the good of mankind.
— *Otto Octavius, Spider-man 2*

1.1 Motivation

Foundation models, such as large language models (LLMs) [17, 18, 19, 20, 21] and large vision models (LVMs) [22, 23, 24, 25, 26], have changed the landscape of machine learning and artificial intelligence (AI) [27]. These increasingly open-ended and ever-more capable systems have quickly become mainstream and have, arguably, brought forth a paradigm shift in the field [28]. Unlike the last AI era of pretraining and fine-tuning on narrow, specific tasks [29, 30], foundation models are extremely generic and can be used as plug-and-play backbones to serve one's desired applications with essentially no changes to the model itself [31]. Consequently, applications and challenges that seemed intractable are now within reach. For example, foundation models have been used to power the reasoning and decision making capabilities of autonomous robotics systems to unprecedented success [32, 33, 34, 35]; LLMs have been shown to learn *extremely* low-resource languages entirely in-context [17] and challenging forecasting problems have seen human-level performance using them [36]; and programming tools have been augmented with foundation model backed systems to increase user productivity [37, 38, 39].

While the achievements of foundation models are impressive and show a fertile ground for further possibilities, we must remember the flip side of any emerging technology is the (possibly novel) risks it brings forth [40, 41, 42, 43, 44, 45, 46]. Already, several risks of systems based on foundation models have been demonstrated [47, 48, 49]. For example, these systems can show egregious biases against historically marginalized communities in decision making environments like resume screening or welfare distribution [48]; they can displace

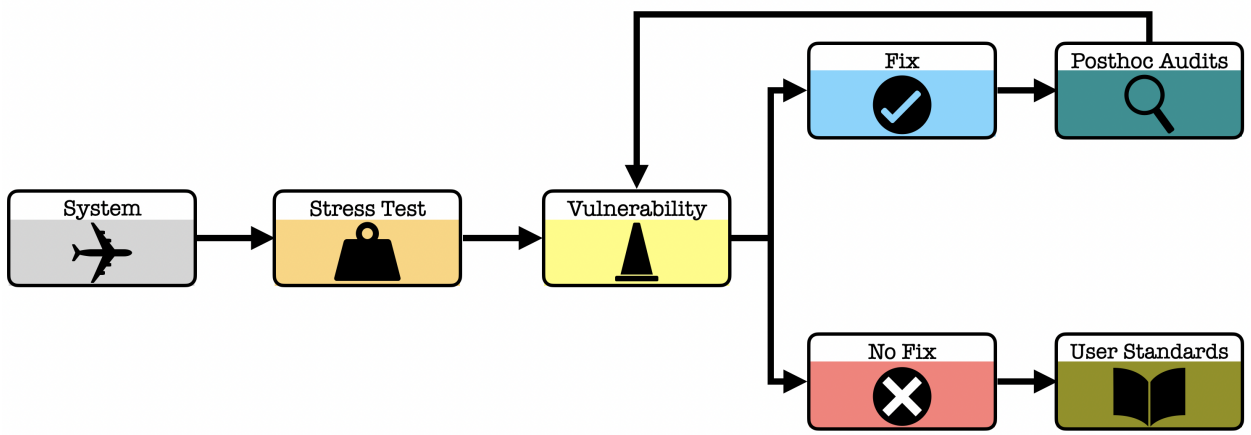


Figure 1.1: **Towards design of safety critical systems.** Risk regulation frameworks for design of safety-critical systems emphasize the need for exhaustive characterization to identify and fix *expected* vulnerabilities in a safety-critical system before deployment. To the extent a vulnerability is an inherent behavior of the system, it can be added to guidelines of safe use. However, user interactions or use in novel scenarios can reveal further vulnerabilities during deployment of a system, requiring rigorous auditing and red-teaming to identify such novel vulnerabilities. Intervention protocols should be designed to address and mitigate such vulnerabilities.

professionals and hamper their means for earning a livelihood by, e.g., generating extremely realistic and at times outright copied content [50, 51, 52, 53]; and they can be used to enable misinformation campaigns [54, 55, 56]. Slightly more speculatively, one can imagine the use of code assistance AI introducing vulnerabilities in a codebase, especially if the codebase is complex and difficult for the human supervisor to properly comprehend [57]. Similarly, the use of AI for planning and placement in chip design [58] may introduce novel hardware vulnerabilities that the system may not have been tested against. Such failures, if elicited in high-stakes situations, can lead to undesirable outcomes [59]. In the longer term, the use of foundation models in the context of military or biotechnological applications can further increase harms caused by them [60, 61].

The examples presented above may not seem "novel" from a sociological standpoint, but there is a dearth of legal and technical frameworks to address them in the context of AI. For example, current US law is unclear on how an individual affected by automated decision making can seek relief or contest such a decision [62]. Meanwhile, even in relatively simple problems, debiasing a model to reduce its tendencies to make biased decisions is an extremely difficult (socio)technical problem [49, 47]. To address this situation, the machine learning community has been increasingly seeking inspiration from the field of risk regulation to adopt their established practices for design of safety-critical systems [63, 46, 64, 65, 66]. Broadly,

these fields emphasize a design framework that can be summarized as follows (see Fig. 1.1)*.

(i) Exhaustively characterize the system by defining a list of potential vulnerabilities and stress-test it before deployment.

(ii) Design intervention protocols to address vulnerabilities identified in the system.

(iii) Establish explicit and well-defined use standards that disallow or discourage usage in scenarios of unfixed or possible vulnerabilities.

(iv) Red-team and perform rigorous posthoc audits to identify novel vulnerabilities.

The work presented in this dissertation is motivated by the framework above. Specifically, while seeking inspiration from systems engineering and risk regulation is reasonable, we argue there are novel challenges exhibited by systems grounded in modern, increasingly open-ended and capable foundation models that render this framework and the processes used to operationalize it less effective when compared to other scenarios. For example, when testing a system for potential vulnerabilities, an implicit assumption is that the system will behave "continuously", i.e., for inputs close enough to the ones used for testing the system, the behavior will be similar. However, as several recent works have shown, a minuscule change in the prompt of an LLM can yield a broad spectrum of capabilities [68, 69, 70, 71]. Stress testing such a system is therefore difficult, as a user might use a slightly novel prompt compared to the ones used for testing, *unintentionally* eliciting undesirable behaviors from the model (e.g., biased decision making) [42]. Similarly, most posthoc intervention protocols designed to fix such vulnerabilities rely on fine-tuning the model on some downstream dataset. However, several works have demonstrated that vulnerabilities can still be elicited via use of adversarial prompts [72]. What exactly does fine-tuning do to the model then? Is a fine-tuning based approach inherently limited in its ability to fix a model's vulnerabilities?

1.2 Contributions

This dissertation is divided into three parts that target different aspects of the risk regulation framework for the design of safety-critical systems. Specifically, we present either (i) improved understanding of known challenges or (ii) novel challenges that emerge in the appropriation

*Note that for brevity, the framework description is kept relatively narrow here. While the discussed steps are consistently present in risk regulation of safety-critical systems, there are several more crucial aspects like incentive structures in socio-technical problems (such as AI) that must be accounted for; see Levenson [67] for a more thorough discussion of safety-critical systems' engineering and Kaminski [63] for discussion of risk regulation frameworks for AI.

of this framework for systems grounded in foundation models. The content of these three parts is discussed in brief below. Note that unless mentioned otherwise, we will use the word "capabilities" as a blanket term to refer to a model's ability to perform a predefined task in a specific manner; wherever possible, though, we do formalize the notion of a capability to make the intended meaning precise. Specifically, in the context of systems grounded in foundation models, our intended meaning by the term vulnerability is the model possessing features that can activate an undesirable behavior. For example, the ability to exploit sensitive attributes, such as an individual's gender, is a vulnerability if it enables biased decision making in scenarios where such attributes are irrelevant.

- **Part 1: Unpredictable Capabilities Render Exhaustive Characterization Difficult.** Emergent or discontinuous learning with respect to some relevant scaling axis (e.g., model size or training budget) make it difficult to preemptively predict a model's capabilities, and hence its vulnerabilities. This renders exhaustive characterization of the model difficult, since a vulnerability may not be shown by the model until a certain training configuration is reached, and hence we may not develop a relevant evaluation protocol to test for it. The goal of this part is to develop a better understanding of emergent abilities: *are there precisely definable characteristics that capabilities that exhibit emergent learning possess?* To address this question, we first propose the multiplicative emergence hypothesis (Chapter 2), which implicates a capability's compositional nature as a source of its emergent learning curves—until all abilities involved in the composition are learned, the model is unable to exhibit the compositional capability. This suggests evaluations and risk-assessments should try to define a decomposition of a compositional capability: the model getting better at individual components of the decomposition may serve as a signature for it getting better at the compositional one. We further build on these results in Chapter 3 and show the point of emergence for compositional capabilities can be analogized to a percolation phase transition from graph theory. While a simplification, the chapter indicates theories of emergence from other fields may be of relevance to enable prediction of emergent capabilities.

- **Part 2: Latent Capabilities and Underspecification as Hurdles to Evaluations.** Well-defined evaluations are crucial to perform risk-assessments, helping us identify vulnerabilities in a system and establish its use standards. In this part, we show two hurdles in design of evaluations for increasingly open-ended foundation models. First, we analyze the learning dynamics of a model in a compositional generalization task and demonstrate that much before the model exhibits a capability (when prompted

via an input to do so), there already exist latent interventions that can elicit the capability (Chapter 4). Accordingly, measuring mere input-output behavior—as done in standard benchmarking—becomes insufficient to characterize what capabilities a model possesses. We then focus on challenges in defining use standards for models based on evaluations. The capabilities of open-ended models like LLMs are known to show high variance, preventing establishment of an explicit standard for safe use. We analyze this phenomenon through the lens of underspecification and Bayesian hypothesis selection. We characterize how a model resolves underspecification to produce a specific output over other possible ones (Chapter 5), yielding a mechanistic explanation for why behavior of highly open-ended models can drastically change under minimal changes to an input.

- **Part 3: Limitations of Fine-tuning for Removing Model Vulnerabilities.** Once an undesirable behavior or vulnerability is identified, we must fix it. The primary tool used for enabling this fix in contemporary literature is fine-tuning on a downstream dataset of desired behavior. While fine-tuning does enable mitigation of undesirable behaviors on the specific inputs used in this process, we find that when going out-of-distribution with respect to the fine-tuning data distribution (formalized in the language of causality in this dissertation), the model still exhibits its undesirable behaviors. To make this point, we (i) demonstrate fundamental limitations of fine-tuning through the lens of mode-connectivity [73, 74] in neural network loss landscapes (Chapter 6), and (ii) characterize the precise changes fine-tuning makes to a model via mechanistic tools and a procedural language learning task (Chapter 7).

1.3 Approach: Model Experimental Systems

While the presented work is motivated by practical, large-scale models and challenges seen in their safety, performing an exhaustive analysis to understand the causes behind such models' vulnerabilities can be unreliable, difficult, and expensive with academic resources. For example, when trying to understand challenges in stress-testing of an LLM due to input prompt sensitivity, a behavior seen from the model on a set of inputs can be confounded by the fact that those inputs may have been seen during training with a large enough frequency to have been memorized [75, 76, 77]. To address this challenge, we seek inspiration from natural sciences and borrow the approach of *model experimental systems* (see Fig. 1.2).

A model experimental system is an abstraction of the target system that captures the behavior one intends to investigate. The system is intentionally designed

to be simpler, more controllable, and easily steerable to enable development of mechanistic hypotheses [78] applicable to the more complex target system's understanding. For instance, in neuroscience research, model organisms like mice or fruit flies are used to study neural mechanisms and develop hypotheses that can be tested on the larger-scale human brain [79, 80, 81]. Similarly, for the work discussed in this dissertation, we will discuss several model experimental systems that are designed to enable analysis of the target behavior (e.g., emergent learning of a capability). Via an exhaustive characterization of the model experimental system, we can then develop concrete hypotheses that describe why a larger scale model may exhibit a certain behavior. This substantially reduces the design space of experiments to conduct on larger scale systems: experiments must now be designed to specifically verify or reject the hypotheses uncovered via results found on the model experimental system.

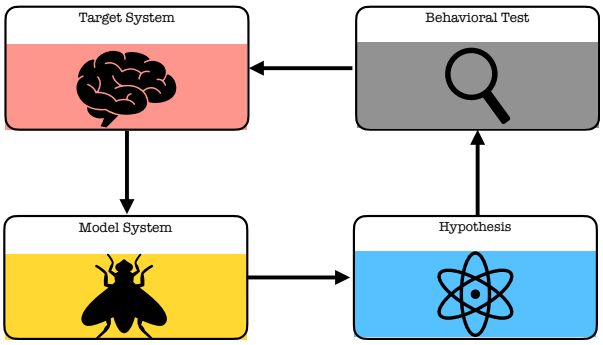


Figure 1.2: **A Model Experimental Systems Approach to Understanding Complex Systems.** Consider a complex system like the human brain, which is often the target of study in natural sciences. Directly analyzing the brain can often be infeasible, so researchers focus their experiments on *model systems*, i.e., a substitute for the target system that is expected to solve the task of interest in a similar way as the target system. Analysis of the model system leads to derivation of a hypothesis for how the target system might be solving the task, hence allowing design of precise experiments (e.g., behavioral tests when analyzing the brain) that are meant to accept or refute just that hypothesis.

It is important, however, to emphasize the trade-offs and limitations of a model-experimental systems approach. Just as neural mechanisms identified in model animals cannot be directly applied to human medical applications, our observations should not be considered definitive conclusions that can be directly transferred to modern large generative models. Instead, the presented work aims to establish conceptual frameworks, identify data-centric control variables, and formulate mechanistic hypotheses, paving the way for further theoretical and empirical explorations of larger models.

1.4 Timeline

An increased interest in the topic of design of safe AI systems in December 2022 made me switch directions from general understanding of neural networks to understanding and

formalization of concepts that were of relevance to design of safe AI systems. Consequently, the work presented in this dissertation was carried out over the last two years and, hence, discusses a subset of the overall contributions made during the course of this PhD [82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]. I hence note that the presented work describes my first steps into the field of design of safe AI systems, and there is unquestionably a long way left to go. However, the presented work and my explorations of this topic at broad have left me hopeful that if all stakeholders find common ground, we can in fact ensure that AI yields a net positive return for our society than otherwise. To enable this, I am deeply interested in several directions—both technical and otherwise—and am excited to start working on them in the very near future.

Part 1: Unpredictable Capabilities Render Exhaustive Characterization Difficult

Exhaustively characterizing a system for potential vulnerabilities requires one list a set of undesirable capabilities that the model can be expected to exhibit. A critical tool for defining such lists is the capabilities exhibited by other members of the model family; e.g., using smaller versions of the GPT models [100] to define vulnerabilities that larger variants should be tested for [9]. At times, however, progress on learning a capability may show "emergent" behaviors: as model size, data, or training budget is scaled, the model exhibits rapid learning of the capability beyond some critical threshold, such that the capability is present at scales larger than the critical one and (essentially) nonexistent at scales below it [101, 102]. Such emergent learning reduces the possibility of exhaustive characterization, as until a certain scale of training configuration is reached, the model may not exhibit the vulnerability, demonstrating it only after deployment.

To address the problem above, we must first identify what type of tasks show emergent learning—are there broad, yet precisely definable characteristics that guarantee a task with said characteristics will show emergent learning? The goal of work presented in the first part of this dissertation is to make progress on this question. Findings of each chapter involved in this part are summarized below.

Contributions of Part 1

- **The multiplicative emergence hypothesis**. A mechanistic explanation for emergent learning curves, the hypothesis states that when a target capability involves composing several other capabilities, until the model learns each individual capability involved in the composition sufficiently well, the multiplicative effect of performance on each capability will render the performance on the target

capability to be low. Only when all underlying capabilities have been learned, the model will show good performance on the target capability, hence producing an emergent learning curve.

- **A percolation model of emergence**. Drawing an analogy with studies of emergence in natural sciences, in this chapter, we argue for a phenomenological description of the term "emergence" in the context of neural networks: emergence should involve sudden learning of multiple capabilities and yield precise structural changes in the model. Such a definition grounds emergence in structural changes and motivates the design of "order parameters", i.e., evaluation metrics that gauge to what extent said structure has been acquired. Building on this analogy, we design a model of emergence grounded in theory of graph percolation and show that on a task of formal language learning, this model predicts a second-order phase transition occurs in the model's learning of capabilities to perform specific, narrower tasks on sentences from the language. Broadly, the goal of this chapter is to promote the argument that theories of phase transition can be of relevance to preemptively determining at what scale a certain capability may be learned by the model.

Author Roles

This part is based on a paper published at NeurIPS 2023 [86], co-authored with Maya Okawa, Robert Dick, and Hidenori Tanaka; and another that is yet to be submitted to a venue [87], co-authored with Kyogo Kawaguchi, Robert Dick, and Hidenori Tanaka. In the first work, I defined the experimental setup, performed preliminary experiments, wrote the paper, and did all relevant related work reading. Experiments were led by Maya to their final stage and the broader setup was formalized in frequent interactions with Hidenori. In the second work, Kyogo first made the analogy of compositional generalization to percolation and hence kick-started the project. I used the analogy to define an experimental setup grounded in formal languages that was amenable to standard neural network training. I then further extended the analogy to accommodate a notion of structure and order parameters broadly, conducted all experiments, and wrote the paper.

CHAPTER 2

The Multiplicative Emergence Hypothesis

2.1 Introduction

The scaling of data, models, and computation has unleashed powerful capabilities in generative models [103, 5, 104, 105, 106, 107, 108, 109], enabling controllable synthesis of realistic images [24, 110, 111, 112, 26, 25, 113, 114, 115, 116, 5], 3D scenes [117, 118, 119, 120, 121], videos [122, 123, 124, 125, 126, 127, 128, 129, 130], accurate image-editing [131, 132, 133, 134, 135, 136], and semantically coherent text generation [100, 137, 18, 138, 139, 140, 141, 142, 143, 30, 144, 145, 37, 38, 146]. With increased interest in incorporating these models in day-to-day applications [147, 148, 149, 150], e.g., to improve robotic systems via better planning and grounding [151, 152, 153, 154, 35, 155, 156, 157, 158], analyzing and improving their reliability has become crucial.

With the motivation above, we study compositional generalization abilities of conditioned diffusion models, i.e., diffusion models that are conditioned on auxiliary inputs to control their generated images (e.g., text-conditioned diffusion models [25, 131]). Given the inherent compositionality of the real world, it is difficult to create a training dataset that exposes a model to all combinations of different concepts underlying the data-generating process. We therefore argue that compositional generalization is central to model reliability in out-of-distribution scenarios, i.e., when the model has to compose a novel set of concepts to generate outputs not seen in the training data set [159, 160]. As shown by several prior works investigating the compositional generalization capabilities of off-the-shelf text-conditioned diffusion models [161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171], modern diffusion models often compose complicated concepts, producing entirely non-existent objects, but also unpredictably fail to compose apparently similarly complicated concepts (see Fig. 2.1). It remains unclear why models produce novel samples by composing some concepts but not others. Indeed, what concepts does a model learn to compose? What differentiates these concepts from the ones the model is unable to compose?

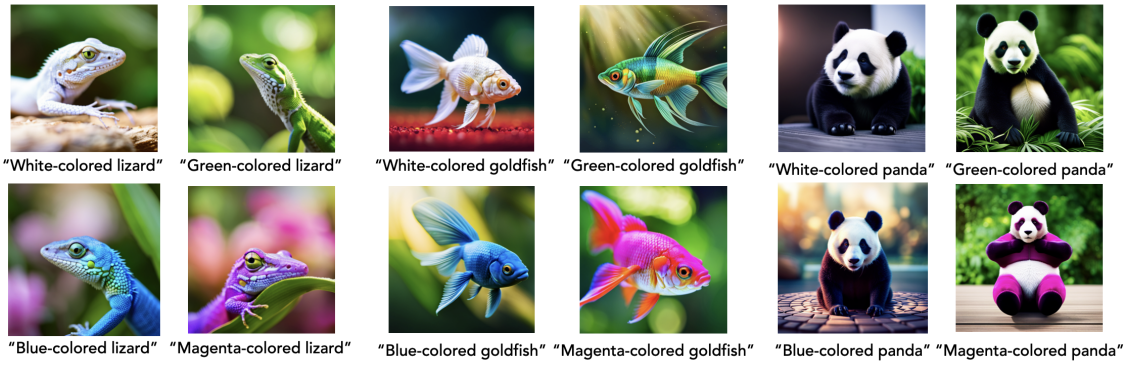


Figure 2.1: **(Lack of) Compositionality in text-conditioned diffusion models.** Images generated using Stable Diffusion v2.1 [1]. We see diffusion models conditioned on text descriptions of the concepts in an image often allow generation of novel concepts that are absent from the training data, indicating an ability to compose learned concepts and generalize out-of-distribution (lizard and goldfish panels). However, the model sometimes unpredictably fails to compose concepts it has learned, i.e., compositional capabilities depend on the specific concepts being composed. For ex., generating a panda in the above figure is difficult for the model, likely because it has not been exposed to different colors of pandas. The model instead alters the background or lighting to change the color.

"Model Experimental Systems" Approach with Interpretable Synthetic Data. To address the above questions, we seek to take a systematic approach to understanding complex systems, such as generative models, by dividing the process into hypothesis generation and testing via a *model-experiment system*: a synthetic experimental setup that pursues simplicity and controllability while preserving the essence of the phenomenon of interest, i.e., compositional generalization. Specifically, our data-generating process tries to mimic the structure of the training data used in text-conditioned diffusion models by developing pairs of images representing geometric objects and tuples that denote which concepts are involved in the formation of a given image (see Fig. 2.2). We train diffusion models on synthetic datasets sampled from this data-generating process, conditioning the model on tuples denoting which concepts an object in the image should possess, while systematically controlling the frequencies of concepts in the dataset. Thereafter, we study the model's ability to generate samples corresponding to novel combinations of concepts by conditioning the denoising process on a correspondingly novel tuple, thus assessing the model's ability to compositionally generalize. This approach allows us to systematically investigate the key properties of a dataset enabling compositional generalization in an interpretable and controlled manner in conditioned diffusion models. Our main contributions follow.

- ***Concept graphs:* A simple, synthetic framework for studying conditioned diffusion models.** We develop a minimalistic abstraction for training conditioned

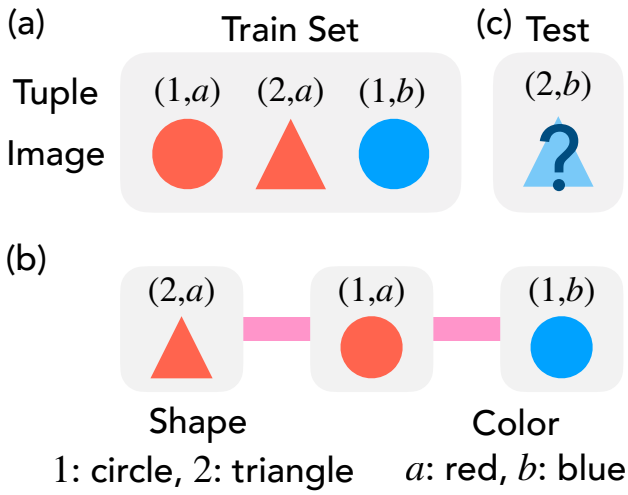


Figure 2.2: **Compositionality in a minimalistic conditioned generation task.** (a) We train diffusion models on pairs of images and tuples, where the tuples denote the values of the *concepts* represented in the image (e.g., values of color and shape in the figure). (b) Samples between which only a single tuple element differs simplify the learning of a *capability* to recognize and alter the concept distinguishing the images. (c) To test the existence of such capabilities of a trained model, we ask the model to generate images corresponding to novel tuples that are out-of-distribution, hence requiring compositional generalization.

diffusion models by introducing a framework called "concept graphs". The framework forms the foundation of our systematic study of how diffusion models learn to generate samples from a set of concepts and compose these concepts to generate out-of-distribution samples. To our knowledge, our results are the first demonstration of perfect compositional generalization on a (synthetic) vision problem.

- ***Multiplicative emergence* of compositional abilities.** We use our model system's interpretability to monitor learning dynamics of a diffusion model's capabilities and the ability to compose them to produce out-of-distribution data. As we show, a diffusion model first memorizes the training dataset and then sequentially generalizes to concepts that are at a greater "distance" from the training distribution. Since progress in learning each capability multiplicatively affects the model's performance in compositional generalization, we find a sudden ability to compose and produce samples out-of-distribution "emerges". We thus hypothesize compositionality may partially explain emergent phenomena seen in modern generative models [101, 172, 27, 173, 174].

- **Investigating challenges for compositional generalization.** We systematically examine the challenges arising from decreased frequency of specific concepts necessary for learning a corresponding capability. We further evaluate the effectiveness of using

fine-tuning to address misgeneralizations in adversarial settings, and find that it is *generally insufficient* to enable the learning of new capabilities.

2.2 Related Work

Diffusion models. Diffusion models are the state-of-the-art method for generating realistic visual data [24, 110, 111, 112, 26, 25, 113, 114, 115, 116, 5]. They are often trained using image-text pairs, where the text is processed using a text-specific model to produce semantically rich embeddings that allow controlled generation of novel images and editing of existing images [131, 132, 133, 134, 135, 136]. Such conditioned diffusion models are easy to test for compositional generalization, as one can directly give a text description that requires composition of concepts the model is likely to have learned (e.g., avocado and chair) to produce images that were not encountered in the model's training data (e.g., avocado chair; see [24, 25]). Such results demonstrate the model's ability to compose and generalize out-of-distribution. However, we emphasize that text-conditioned models may fail to generalize compositionally if the text model, depending on the type of model used, is unable to sufficiently disentangle relevant concepts in the text-embedding space or follow specified instructions. To avoid this pitfall, we use ordered tuples that denote precisely which concepts are involved in an image's composition.

Compositional generalization. Compositionality is an inherent property of the real world [175], wherein some primitive such as color can be composed with another primitive such as shape to develop or reason about entirely novel concepts that were not previously witnessed [159]. Notably, compositionality is hypothesized to play an integral role in human cognition, enabling humans to operate in novel scenarios [176, 177, 178, 179, 180]. Inspired by this argument, several works have focused on developing [181, 166, 167, 182, 183, 184, 185, 186] and benchmarking [187, 188, 189, 190, 191, 192, 193, 163, 183, 194, 165, 195] machine learning models to improve and analyze compositional generalization abilities. We especially highlight the works of Lewis et al. [189], who use a similar model experimental system approach as ours to evaluate factors influencing compositionality in contrastive language-image pretraining (CLIP) [22]. Our focus on generative models, instead of representation learning, distinguishes our results however. Relatedly, we note the notion of compositionality has been discussed in several recent works in the fields of disentangled and causal representation learning [196, 194, 197, 198, 199, 200]. Our framework for systematically understanding compositionality in generative models was heavily influenced by the synthetic dataset design considerations promoted in these papers. We also emphasize that a thorough formalization of compositionality in modern machine learning is generally lacking, but highlight the notable

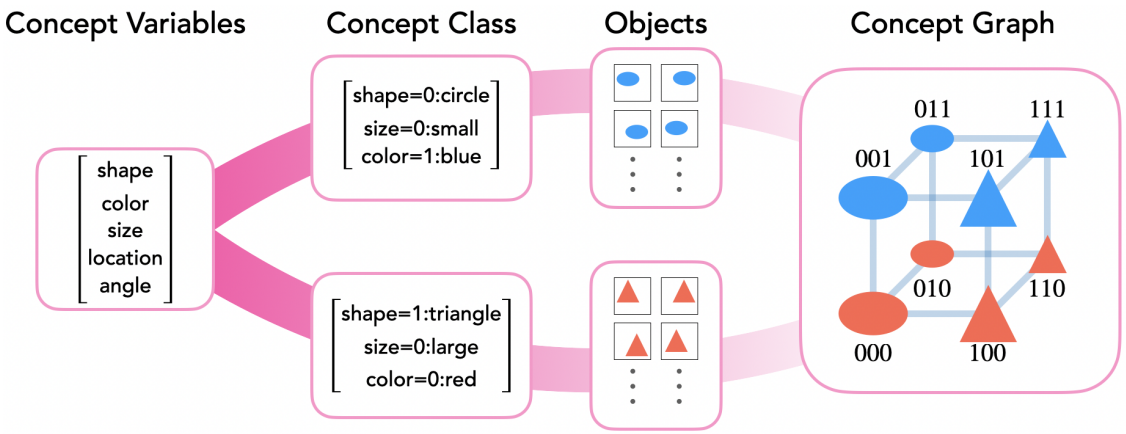


Figure 2.3: **Concept graphs.** We organize our study using a simple but expressive framework called *concept graphs*. The basis of a concept graph is a set of primitives called *concept variables* (e.g., `shape` , `color` , etc.). A subset of these variables is instantiated with specific values to yield a *concept class*, e.g., $\{\texttt{shape} = 0, \texttt{size} = 0, \texttt{color} = 1\}$ implies a small, blue circle. This is akin to defining a broad set of objects that share some common properties, e.g., all lizards in Fig. 2.1 belong to the same species, despite their diverse colors. A specific *object* in this class is instantiated by fixing the remaining variables, e.g., small, blue circles at different locations. Each concept class corresponds to a graph node, where nodes are connected if their concept classes differ by a *concept distance* of 1.

exceptions by Hupkes et al. [201] and Wiedemer et al. [199]. To avoid ambiguity, we provide a precise formalization that instantiates our intended meaning of compositionality.

2.3 Concept Graph: A Minimalistic Framework for Compositionality

In this section, we present the *concept graph* framework, as illustrated in Fig. 2.3, which enables us to visually depict compositional structure of our synthetic data and forms the basis for generating hypotheses and designing experiments in our work. We begin by defining the essential building blocks of our framework: concept variables and concept values. In the following, we call a specific output of our data-generating process an "object" (see Fig. 2.3).

Definition 1. ***(Concept Variables.)*** *A concept variable v_i uniquely characterizes a specific property of an object. $V = \{v_1, v_2, ..., v_n\}$ denotes a set of n concept variables, such that, given an object x, $v_i(x)$ returns the value of the i^{th} concept variable in that object.*

For instance, for geometric objects shown in Fig. 2.3, concept variables may include `shape` , `color` , `size` , `location` , and `angle`. These variables take on values from a pre-specified range, called concept values, as described below.

Definition 2. ***(Concept Values.)*** *For each concept variable $v_i \in V$, $C_i = \{c_{i1}, c_{i2}, ..., c_{ik_i}\}$ denotes the set of k_i possible values that v_i can take. Each element of C_i is called a concept value.*

For example, in Fig. 2.3, the concept values for the `shape` variable are in the set $\{\text{circle}, \text{triangle}, \text{square}\}$, while for the `color` values are in the set $\{\text{red}, \text{blue}, \text{green}\}$. Given n concept variables, with each variable v_i having k_i concept values, there can be $\prod_{i=1}^{n} k_i$ distinct combinations that yield a valid object. We use a unique combination of a subset of these concept variables, $v_1, \ldots, v_p$ $(p < n)$, to define the notion of a "concept class".

Definition 3. ***(Concept Class.)*** *A concept class C is an ordered tuple $(v_1 = c_1, v_2 = c_2, ..., v_p = c_p)$, where each $c_i \in C_i$ is a concept value corresponding to the concept variable v_i. If an object x belongs to concept class C, then $v_i(x) = c_i \, \forall i \in 1, \ldots, p$.*

Note that the remaining $n - p$ concept variables are free in the above definition. When these free variables are specified, we get a specific object from our data-generating process. That is, a concept class represents a family of objects by specifying the values of a pre-defined subset of concept variables. For example, in Fig. 2.1, different colored lizards instantiate images (objects) from the species (concept class) of lizards; here, color of the lizard serves as a free variable. Similarly, in the geometric objects scenario of Fig. 2.3, a "small red circle" would be a concept class for which the `shape` , `color` , and `size` variables have been assigned specific values. Objects are images designated by associating precise values with the remaining concept variables of `location` and `angle` . Next, we introduce the notion of concept distance, which serves as a proxy to succinctly describe the dissimilarity between two concept classes.

Definition 4. ***(Concept Distance.)*** *Given two concept classes $C^{(1)} = (c_1^{(1)}, c_2^{(1)}, ..., c_p^{(1)})$ and $C^{(2)} = (c_1^{(2)}, c_2^{(2)}, ..., c_p^{(2)})$, the concept distance $d(C^{(1)}, C^{(2)})$ is defined as the number of elements that differ between the two concept classes: $d(C^{(1)}, C^{(2)}) = \sum_{i=1}^{p} I(c_i^{(1)}, c_i^{(2)})$, where $I(c_{1i}, c_{2i}) = 1$ if $c_{1i} \neq c_{2i}$ and $I(c_{1i}, c_{2i}) = 0$ otherwise.*

The concept distance quantifies the dissimilarity between two concept classes by counting the number of differing concept values. It is important to note that *this distance serves only as a null model* because each axis represents distinct concept variables, wherein each of the variables can assume various possible concept values. We are now ready to define the notion of a concept graph (see Fig. 2.3), which provides a visual representation of the relationships among different concept classes.

Definition 5. ***(Concept Graph.)*** *A concept graph $G = (N, E)$ consists of nodes N and edges E, where each node $n \in N$ corresponds to a concept class, and an edge $e \in E$ connects*

two nodes n_1 and n_2 representing concept classes $C^{(1)}$ and $C^{(2)}$, respectively, if the concept distance between the two concept classes is 1, i.e., $d(C^{(1)}, C^{(2)}) = 1$.

A concept graph organizes different concept classes as nodes in the graph, while edges denote pairs of concept classes that differ by a single concept value. While a simplified abstraction meant to enable a precise discussion of compositional generalization, we note that one can in fact find concept graphs *in-the-wild*. For example, in Fig. 2.4 we show how a pretrained CLIP model's [22] representations for prompts comprised of concepts `object`, `color`, and `background color` natural organize into a concept graph.

Now, ideally when we train a conditioned diffusion model on a subset of nodes from the concept graph, we should see the model learns *capabilities* that allow it to produce objects from other concept classes. We formalize this notion of capabilities and, correspondingly the definition of compositionality, relevant to our discussion as follows.

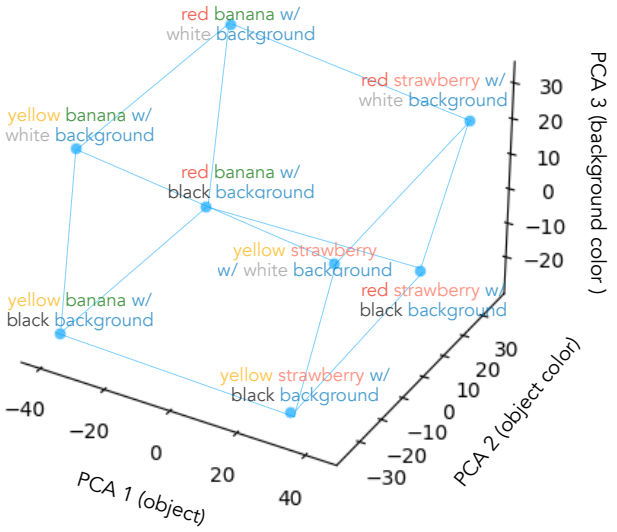


Figure 2.4: **An in-the-wild Concept Graph.** Analyzing CLIP embeddings via a 2D PCA projection demonstrates models naturally organize representations for sufficiently independent concepts—in this case, `object`, `color`, and `background color`—akin to a concept graph (see Fig. 2.3).

Definition 6. ***(Capability and Compositionality.)*** *Consider a diffusion model trained to generate samples from concept classes $\hat{C} = \{C_1, C_2, \ldots, C_T\}$. We define a* ***capability*** *as the ability to alter the value of a concept variable v_i to a desired value c_i. We say the model* ***compositionally generalizes*** *if it can generate samples from a class $\widetilde{C}$ such that $d(\widetilde{C}, C_i) \geq 1 \forall C_i \in \hat{C}$.*

The ideas above are best explained via Fig. 2.5. Specifically, assume we train a diffusion model on data from a subset of concept classes, i.e., a subset of nodes in the concept graph. To fit the training data, the model might just memorize the training data or it *may* learn the relevant capabilities to alter specific concept variables; e.g., given samples from classes `0000`, `0010`, and `0001` in Fig. 2.5 (a), the model may just memorize the data or learn how to alter the third and fourth concept variables. Models that just memorize the training data will lack the capability to generate samples from out-of-distribution classes (e.g., `1001`). In contrast, the

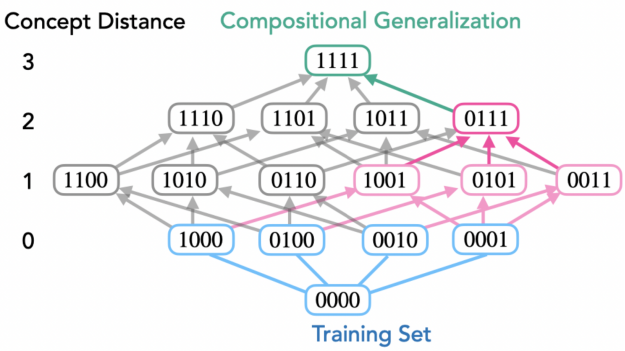


Figure 2.5: **Capabilities and compositionality in a concept graph.** Consider a lattice representation of a concept graph corresponding to four concept variables. Blue nodes denote classes represented in the training data; a model can either memorize data from these classes or learn capabilities to transform samples from one class to another. If it learns capabilities, it can compose them with data-generating process of samples from a concept class in the training data and produce samples that are entirely out-of-distribution, denoted as pink and green nodes.

ability to compose concepts would enable the model to produce out-of-distribution samples starting from classes seen in the training data.

In summary, our concept graph framework provides a systematic approach to representing and understanding a minimalistic compositional structure, allowing for an analysis and comparison of different learning algorithms' abilities to generalize across various concept classes.

2.3.1 Experimental and Evaluation Setup

With the concept graph framework in place, we now have the tools to systematically investigate the impact of different data-centric properties on learning and composing capabilities in conditioned diffusion models. Before proceeding further, we briefly define our experimental setup and evaluation protocol. This allows us to interleave our results with our hypotheses, explaining them in context.

Experimental Setup. The detailed setup is presented in Appendix A.1. In brief, we train conditioned diffusion models that follow the U-Net pipeline proposed by Dhariwal and Nichol [24]. Our dataset involves concept classes defined using three concept variables, each ranging between two values; specifically, `shape` = {circle, triangle}, `color` = {red, blue}, and `size` = {large, small}. Tuples representing these values are used to condition the diffusion model's training and serve as an abstraction of text conditioning. For example, the tuple 000 implies a large, red circle is present in the image. To sample images from this process,

we simply map the size and color axes to the range $[0, 1]$ and uniformly sample to develop a training dataset of 5000 samples. The precise samples depend on which concept classes are allowed in the data-generating process. Note that for continuous concepts, sampling happens in isolation, not combination—i.e., the training data does not include simultaneous size and color variations. Correspondingly, the, compositional generalization can be assessed by varying these concepts in combination, instead of isolation.

In the setup above, note that the minimal required set of concept classes for learning capabilities to alter concepts is four pairs of tuples and images, each drawn from one of the following four concept classes: $\{000, (\text{circle}, \text{red}, \text{large})\}$, $\{100, (\text{triangle}, \text{red}, \text{large})\}$, $\{010, (\text{circle}, \text{blue}, \text{large})\}$, $\{001, (\text{circle}, \text{blue}, \text{small})\}$. Consider the following two tuples: $\{000, (\text{circle}, \text{red}, \text{large})\}$ and $\{100, (\text{triangle}, \text{red}, \text{large})\}$. In this case, we see that the shape concept is encoded in the first elements. Here, 0 represents a circle and 1 represents a triangle. The remaining concepts are similarly encoded in later elements.

Evaluation Metric. Evaluating whether a generated image corresponds to the desired concept class might require a human in the loop. To eliminate this requirement, we follow methods from literature on disentanglement and train classifier probes to predict whether a generated image possesses some property of interest [196, 202, 203, 204, 205, 206, 207]. Specifically, we use the diffusion model training data to train three linear probes, which respectively infer the following three concept variables: `shape` , `color` , and `size` . We define a model's accuracy for generating images of a given concept class as the product of the probabilities outputted by the three probes that each concept variable matches the value for the concept class. Note that a random classifier for a concept variable will have an accuracy of 0.5. We indicate this random baseline with dotted, gray lines in our plots when useful.

2.4 Multiplicative Emergence of Compositional Abilities

We first investigate the order in which a model learn capabilities to produce samples from a concept class and how the learning of such capabilities affects the ability to compositionally generalize.

Learning dynamics respect the structure of the concept graph (Fig. 2.6). We find the ability to compositionally generalize, i.e., produce samples from out-of-distribution concept classes, emerges at a rate which is inversely related to a class's concept distance with respect to classes seen in training. Specifically, in Fig. 2.6 (a), we show the learning dynamics

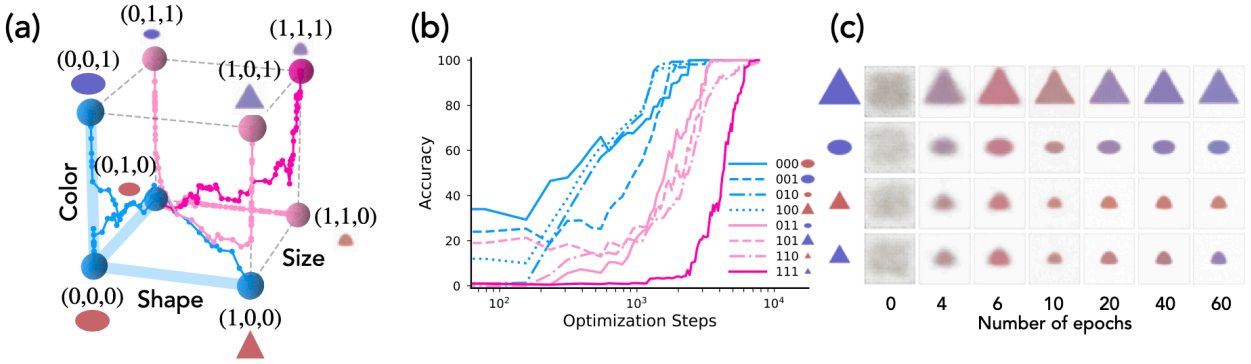


Figure 2.6: **Concept distance from the training set governs the order in which compositional capabilities emerge.** (a) Concept graph (cube) depicting training data points (blue nodes) and concept distances for test data points, where pink nodes represent distance = 1, and darkpink nodes represent distance = 2. Each trajectory represents the learning dynamics of generated images given a tuple prompt. Each trajectory represents the learning dynamics of generated images based on each tuple prompt. During every training epoch, 50 images are generated and binary classification is performed to predict each concept, including color, shape, and size. (b) Compositional generalization happens in sequence, starting with concept distance = 1 and progressing to concept distance = 2. The x-axis represents the number of epochs, and the y-axis represents the progress of compositional generalization. (c) Images generated as a function of time clearly show a sudden emergence of the capability to change the color of small, blue, triangles.

of the model. Here lightblue nodes denote concept classes within the training dataset, while pink and darkpink nodes respectively denote classes at concept distances of 1 and 2 from classes in the training dataset. As the model learns to fit its training data (lightblue nodes), it learns capabilities that can be composed to produce samples from out-of-distribution concept classes (pink / darkpink nodes). Fig. 2.6 (b) further shows that the learning dynamics of compositional generalization are influenced by the concept distance from the training set: the model first learns the concept classes in the training dataset (lightblue lines) and then generalizes to concept classes with a concept distance of 1 from the training dataset (pink lines). Thereafter, the model suddenly acquires the capability to compositionally generalize to a concept class with a concept distance of 2 from the training dataset (darkpink line). Fig. 2.6 (c) shows the images generated by the model over time. We observe that rough shapes and sizes are learned relatively early in training, by the 4th epoch, while the color is highly biased to be red, the majority color in the training dataset, up to the 10th epoch. Then, around the 20th epoch, the model learns to generate the underrepresented color (blue) for concept classes with a concept distance of 1. Finally, around the 40th epoch, the model learns to generate the underrepresented color (blue) for the class at distance 2, showing a sudden emergence of capability to generate samples from that class. The observations above

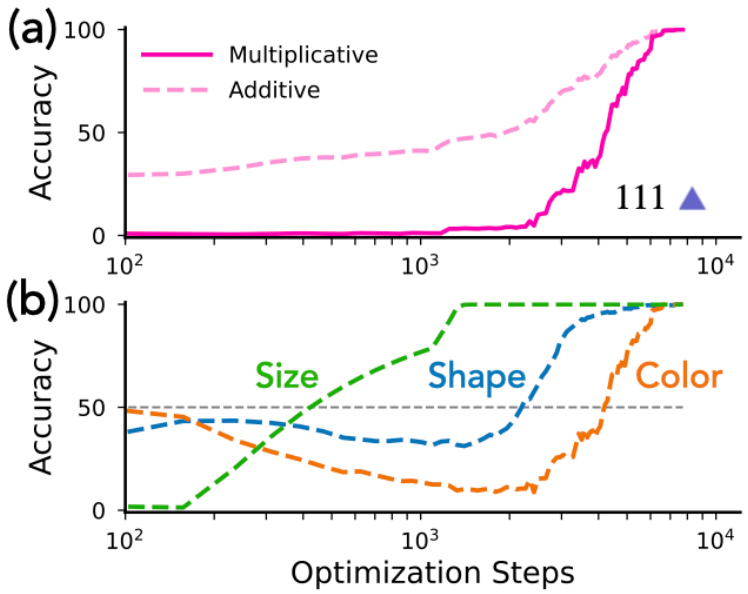


Figure 2.7: **Multiplicative influence of individual capabilities elicits an "emergence" of compositionality.** (a) Accuracy of producing samples from the concept class {111, (triangle, blue, small)}, which has a concept distance of 2 from the training data. A multiplicative measure (solid line) has a score of 1 when all concept variables of shape, color, and size are correctly predicted. Conversely, an additive measure (dashed line) independently relates each concept variable prediction accuracy to the compositional accuracy, deceptively suggesting smooth progress. (b) Learning dynamics of predicting each of the three concept variables: shape (blue), color (orange), and size (green).

generalize to higher-dimensional concept graphs as well (see Fig. A.6).

Multiplicative influence of capabilities drives the sudden emergence of compositional generalization (Fig. 2.7). The interpretability of our experimental setup illustrates that the *multiplicative* reliance on underlying capabilities is the critical structure behind the sudden emergence of compositional generalization in our setup. For example, in Fig. 2.7 (a), we show the dynamics of an additive vs. multiplicative accuracy measure for generating concept class {111, (triangle, small, blue)}, which has a concept distance of 2 from the training data. We observe a sudden improvement of the multiplicative measure. To better understand the above, in Fig. 2.7 (b), we plot the accuracy of probes used for predicting each concept variable (shape, size, color). From the plot, we notice that the model fails to acquire strong color transformation capabilities until the final stage of training, effectively bottlenecking compositional generalization of 111 class. This empirical observation leads us to the following hypothesis on *emergence of compositional abilities in generative models*.

Multiplicative Emergence Hypothesis

Hypothesis. ***(Compositional Abilities Emerge Multiplicatively.)*** *The nonlinear increase in performance observed in neural networks' training as model size and computational power scale up is partially driven by compositional generalization. Models must learn all required concepts, but compositional generalization is hindered by the multiplicative, not additive, impact of learning progress on each concept. This results in a rather sudden emergence of capabilities to produce or reason about* ***compositional*** *tasks.*

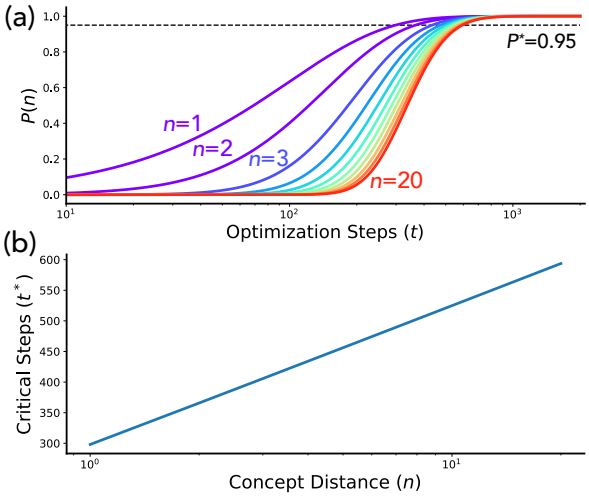


Figure 2.8: **Toy Model of Compositional Emergence.** (a) Probability of learning a compositional capability at a concept distance n w.r.t. atomic abilities as a function of time t. (b) Critical optimization time steps t^* required for the above probability to reach a threshold P^* as a function of concept distance n.

As highlighted in the hypothesis, the claim is centered on compositional generalization. For the purview of this chapter, this corresponds to the ability of the model to generate samples from out-of-distribution concept classes; in fact, in-distribution classes show relatively smooth learning progress in our results (see Fig. 2.6). We claim that while a model can "memorize" a concept based on the training distribution, e.g., what values it possesses, to compositionally generalize and produce out-of-distribution classes, it will need to learn a general, "atomic" capability that enables manipulation of said concept. In this sense, one can analogize compositional generalization to an AND logical condition: the model must learn and concurrently apply all involved "atomic" capabilities it possesses to produce out-of-distribution samples, leading to a non-linear increase in a model's performance once it learns to do so (see also concurrent work making similar claims [172, 208]). We analyze this further via a simple toy model below.

We consider a toy compositional problem which demonstrates how complex capabilities can rapidly develop ("emerge") from a set of atomic abilities. Assume there are n atomic abilities, each with a probability p of being learned in a given time step, i.e., the dynamics of learning an ability can be modeled as a Bernoulli coin flip: once the coin turns from 0 to 1, the model learns the said ability. The probability that the ability will be learned in t steps is

thus given by $1-(1-p)^t$. An implicit assumption in this model is that the learning dynamics of atomic abilities are independent (hence the name atomic) and that once an atomic ability is learned, it will not be forgotten. Now, our goal is to characterize the dynamics of learning a capability that involves composing these atomic abilities. The multiplicative emergence hypothesis argues that learning such a capability by time t requires that the model have learned the n atomic abilities by that time. We then get the probability that the compositional capability has been learned by time t is $P(n) = (1-(1-p)^t)^n$. These dynamics are plotted in Fig. 2.8 (a) and look strikingly similar to the rapid learning (emergence) we found in our diffusion model setup (see Fig. 2.6 b)): for increasing degree of compositionality, we see much sharper learning dynamics (i.e., emergence) that is controlled by the concept distance w.r.t. atomic abilities. Further, assuming a given threshold probability P^* at which one claims a capability has been learned, for a degree of composition n, we can solve for the critical time t^* at which a compositional capability is learned as follows: $t^* = \left\lceil \frac{\log(1-(P^*)^{1/n})}{\log(1-p)} \right\rceil$. Plotting this in Fig. 2.8 (b), we see the progress on increasingly more compositional tasks is logarithmic with respect to the number of atomic concepts being composed (i.e., concept distance). Importantly, this implies if we allow the model to train infinitely and it learns several atomic abilities, it will have a combinatorial explosion of capabilities due to the inherent compositionality of the data generating process—we further investigate this in a separate work [98].

The setup above thus captures the core aspect of our hypothesis: Compositional tasks require the concurrent acquisition of all involved atomic capabilities, akin to an "AND" condition in logical terms. This requirement leads to a non-linear increase in learning capability. Just as all coins must show heads at least once for the overall probability to be significant (an "AND" condition), in large generative models, the mastery of individual components (or atomic capabilities) must occur collectively for emergent capabilities to manifest. This phenomenon underlines the non-linear and threshold-dependent nature of learning in these models, providing valuable insights into their training and development.

2.5 Challenges for Compositional Generalization

We have shown that, given a well-structured synthetic training dataset, a conditioned diffusion model can learn to compose its capabilities to generate novel inputs not encountered in the training set. Next, we analyze an adversarial setups under which the model fails to learn relevant capabilities to compose concepts and if simple interventions can help mitigate these failures.

Critical frequency for learning capabilities (Fig. 2.9). We first probe the effect

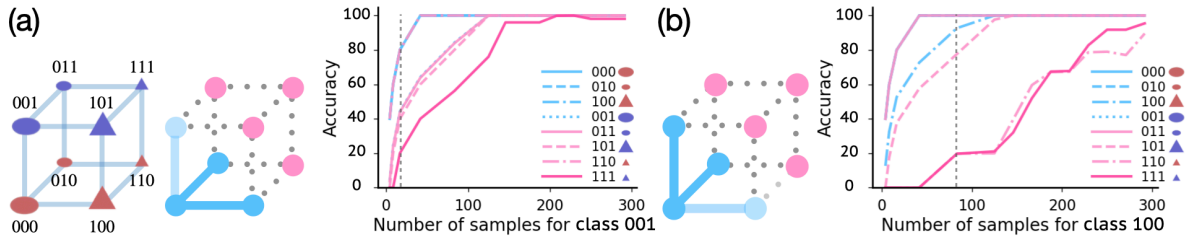


Figure 2.9: **How does the frequency of data samples impact the learning of capabilities?** We systematically control the frequency of a specific concept class in the training dataset and observe how it affects the model's learning of capabilities. (a) Capability to alter colors is quickly learned after introducing approx. 10 samples with a concept class of large blue circle, 001. (b) In contrast, a critical threshold (marked with dotted vertical lines) exists for learning a capability to alter the shape: as we gradually introduce samples, with a concept class of large red triangle, 100.

of changing the frequencies of samples from different concept classes in the training data and examine how this affects the model's ability to learn and compose capabilities involving that concept class. Results are shown in Fig. 2.9 and demonstrate how the frequency of color and size concepts in the training data impacts the generalization capabilities of the diffusion model. Specifically, we change the number of samples in the training data from 0 to 300 for concept class 001 (Fig. 2.9 (a)) and class 100 (Fig. 2.9 (b)). As can be seen, low frequencies of certain concept values degrade the accuracy of the model in both settings. Notably, training for out-of-distribution concept classes (pink lines) requires more samples than that for in-distribution ones (lightblue lines). This suggests that as the sample size grows, memorization occurs first, and generalization is achieved superlinearly as a function of data frequency. More importantly, we observe a critical number of samples are required before we can see the onset of capabilities to alter a concept. Specifically, in Fig. 2.9 (a), we can see that the model rapidly learns the color concept after being provided with 10 samples for a concept of large blue circle, 001. In contrast, in Fig. 2.9 (b), the model learns the shape concept only after reaching a certain threshold in the number of samples with a concept of large red triangle, 100.

We believe the results above are especially interesting because an often used strategy to prevent a generative model from learning harmful capabilities, such as the ability to generate images involving sensitive concepts, involves cleaning the dataset to remove images corresponding to such concepts, hopefully hindering the model's ability to generate samples containing the concept [209, 9, 137]. Such filtering is generally expensive and, arguably, statistically impossible to achieve perfectly in a large dataset, i.e., a few samples corresponding to the sensitive concept are likely to remain in the data. Our findings suggest that one

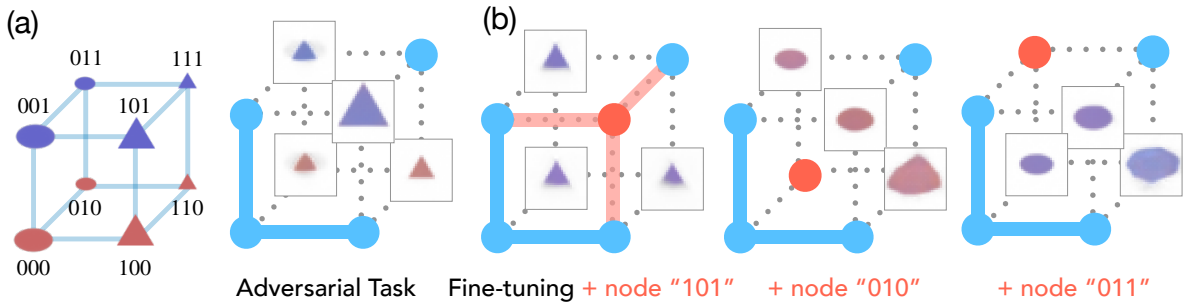


Figure 2.10: **Fine-tuning is not a remedy for misgeneralization in an adversarial task.** (a) The model struggles to generalize when the training setup is configured adversarially. As shown in the plot, the model incorrectly produces triangles (instead of circles) for both 011 and 010. (b) The model faces difficulty learning new concepts through fine-tuning. We add node 101 (triangle, large, blue) to the dataset and attempt to fine-tune the model. However, even with a large learning rate equal to the one used for training, the model fails to learn the capability to alter object size.

might better consider the relationship between the probability of learning a concept and its frequency of occurrence in the data to determine the degree of filtering required: if their presence in the training data is below a critical threshold, that can be sufficient to deter the model from learning a capability to generate samples related to that concept, obviating the need for a perfect filtering of the dataset.

Fine-tuning and misgeneralization in an adversarial task (Fig. 2.10). We next evaluate a setting where concept classes present in the training data are not neighbors, i.e., their concept distances are greater than 1, but nonetheless represent all concept variables to allow a model to learn relevant capabilities. Specifically, as shown in Fig. 2.10 (a), we use only the following four pairs of tuples and images for training: $\{000, (\text{circle}, \text{red}, \text{large})\}$, $\{100, (\text{triangle}, \text{red}, \text{large})\}$, $\{001, (\text{circle}, \text{blue}, \text{small})\}$, $\{111, (\text{triangle}, \text{blue}, \text{small})\}$. Arguably, capabilities corresponding to shape change (000 to 100) and color change (000 to 001) can still be learned as the setup for these is similar to our prior experiments. However, our results show this yields an interesting failure mode; specifically, in Fig. 2.10 (a), we find the model fails to dissociate a second element value of 1 with the shape of a small triangle, and this bias causes the model to generate small triangles for both 011 and 010, when it should have produced small circles.

We further analyze if the failure above can be fixed via mere downstream fine-tuning (see Fig. 2.10 (b)). Specifically, we fine-tune the trained model on a dataset that includes the concept class of 101 (left), 010 (middle), and 011 (right). We find the model still generates small triangles for 011 and 010 even after fine-tuning. When the concept classes 010 (middle) and 011 (right) are added, the newly introduced concepts "overwrite" all existing concepts,

causing the previously learned concepts (e.g., the color and shape concept for 101) to be forgotten. These results present a further challenge in addressing learned bias.

2.6 Additional Experimental Results with Natural Data

We also investigated our hypotheses in a more realistic setting by conducting experiments on the CelebA dataset. We selected three attributes as the concepts: Gender (categorized as Female and "Male"), Smiling ("Smiling" and Not Smiling), and Hair Color ("Black Hair" and "Blonde Hair"). The concept graph corresponding to these attributes in the CelebA dataset is illustrated in Fig. 2.11 (a). For each individual concept, we trained a 3-layer CNN and adopted the product of accuracies from each concept as the evaluation metric. Although not all nodes achieved an accuracy of 100% due to limited training time, we find that our observations (partially) extend: (i) learning dynamics follow the structure of the concept graph and (ii) delayed generalization of underrepresented attribute (gender), hold true, even at larger, more realistic scales. Fig. 2.11 (b) illustrates the learning dynamics using the real CelebA dataset. We again observe the sequential pattern in generalization. The accuracy begins to rise at concept distance 1 (denoted by pink lines), followed by concept distance 2 (red line); the suddenness of learning to compositionally generalize is relatively much less sharp, but still observed. Interestingly, the node labeled `011`—even though it is at concept distance 1—precedes the memorization stage (represented by blue lines). This exception can be attributed to the gender bias present in the training data. In Fig. 2.11 (c), we plotted the accuracy for the individual concept of gender (i.e., Female and Male). The Female concept classes' training curve (red lines) reaches convergence faster compared to the Male concept class (blue lines). This disparity stems from the Female class being more predominant in the dataset, thereby making the Male class underrepresented. This observation offers useful insights for practitioners: to mitigate biased outcomes against underrepresented groups, we need to further train the diffusion model beyond its initial convergence on the training dataset.

2.7 Discussion

In this work, we introduced a concept graph framework and used controlled interventions with a synthetic dataset to formulate and characterize four novel failure modes in the compositional generalization of diffusion models. We first hypothesized and verified that compositional generalization to a concept class far from the training data, based on concept distance, can fail, as it requires more optimization steps due to the sequential nature of generalization

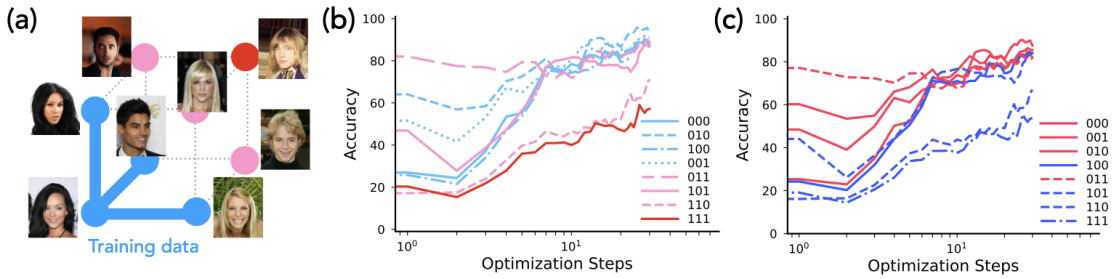


Figure 2.11: **Investigating our claims on a naturalistic data setting (CelebA).** (a) Concept graph (cube) depicting training data points (blue nodes) and concept distances for test data points for CelebA dataset. (b) Compositional generalization happens in sequence, starting with concept distance 1 and progressing to concept distance 2. The quick learning of node `011` and the 50% initial accuracy of node `101` is a consequence of the biased dataset (see Fig. 2.7 for a similar effect in our synthetic setup where there is a color bias). (c) Gender prediction accuracy based on generated samples at each step during training. We observe that the ability to generate the underrepresented attribute (e.g., gender) is slightly delayed as concept distance increases.

(Fig. 2.6). We show compositionality emerges in a sequence that respects the geometric structure of the concept graph, eliciting a *multiplicative emergence* effect that manifests as sudden increases in the model's performance to produce out-of-distribution data well after it has learned to produce samples from its training distribution. This behavior is reminiscent of the recently observed phenomenon of grokking in language modeling-like objectives [173, 174, 210]. Furthermore, when a particular concept value is underrepresented in the training dataset (e.g., an underrepresented color or gender), avoiding misgeneralization of the concept value, and thus achieving compositional generalization, requires far more additional optimization steps, even after achieving perfect performance on the training set. We also discovered a critical threshold for the inclusion of certain concepts in the training data, which is useful for deliberately making the model fail at learning harmful concepts (Fig. 2.9). Finally, we analyze misgeneralizations in an adversarial task and find that fine-tuning does not provide a solution (Fig. 2.10). Overall, our synthetic data approach allowed us to apply controlled interventions that shed light on four different mechanisms behind the failure of diffusion models in compositional generalization.

2.8 Follow-Up Work

The work presented in this chapter is currently the basis of several ongoing and finished projects. For example, in a recently finished project [98], we explored a language-like setup to assess the impact of compositionality in language modeling settings. The argument guiding

that work is however slightly different and focuses on how if a model learns to compose, its capabilities will be necessarily underestimated via mere benchmarking on predefined datasets. We provably show the existence of a Transformer model that can learn to compose out of distribution and the empirical match of our proposed construction with trained models.

CHAPTER 3

Percolation as a Model of Emergence

3.1 Introduction

Modern neural networks, e.g., large language models (LLMs) [17, 18, 19, 21], exhibit a broad spectrum of capabilities, allowing them to serve as the "foundation" for downstream, application-specific systems [211, 212, 213, 31]. As these models scale, either via addition of more data, parameters, or compute, an intriguing behavior is at times observed: until a certain critical scale is reached, there are capabilities that the model does not exhibit; however, beyond this point, such capabilities suddenly "emerge" [101, 214, 9, 5, 215, 216, 103, 217, 218, 219, 220]. More specifically, the performance of the model on a task or benchmark meant to evaluate said capabilities witnesses substantial growth in performance, even though the *overall* training loss undergoes minimal, if any, improvements [172, 221]. Empirical evidence in fact suggests that, at times, several capabilities can emerge simultaneously [101, 222].

Beyond developing a better scientific understanding of neural networks, understanding emergent capabilities is crucial to enable risk-centric regulation frameworks for AI, which assume a system's capabilities can be preemptively conjectured [223, 224, 225, 93, 226, 42]. To this end, recent work has made attempts at identifying factors that decide whether a capability will emerge. For example, Okawa et al. [86] and Arora et al. [172] implicate the underlying compositional structure of a capability as the cause for its sudden learning. Hoffmann et al. [227] argue capabilities that involve interactions between specialized components within a model are likely to yield sudden performance improvements once the correct interaction mechanism is learned; e.g., the interaction between the *previous token* and *copy* attention heads to enable in-context learning [220, 228, 229]. Meanwhile, Schaeffer et al. [208] argue emergent abilities are an artifact of poorly defined, discontinuous evaluation metrics, claiming that models undergo continuous, persistent improvements during training. Recent work has however demonstrated that even continuous metrics can witness sudden improvements, with such changes co-occurring with the model's learning of a new capability [230, 221, 231].

This undermines the claim that emergent capabilities are merely an artifact of evaluation protocols.

Taken together, the orthogonal explanations and disparate results above have resulted in emergence becoming an unclear phenomenon in machine learning. At its core, however, we claim that the concept has never been defined in prior work. This has arguably led to distinct mechanisms causing sudden changes in model performance to all be labeled as "emergence". What is the *phenomenology* that this term is meant to capture in the context of neural networks? Is it merely a sudden increase in performance with scale, or *broader than that*? Given *a* reasonable definition, can we show, even if in a simplified system, that emergent capabilities are commonplace and can we use the system's simplicity to better *understand what drives their sudden learning*?

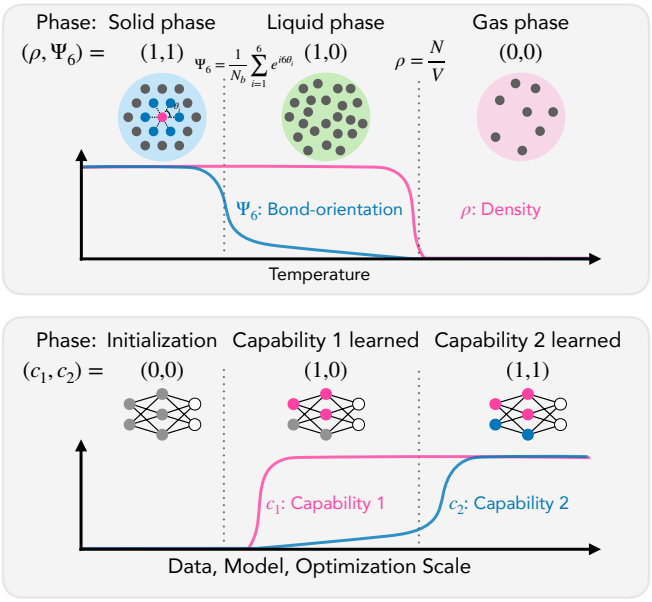


Figure 3.1: **Emergence as phases of learning.** Emergence is a well-characterized phenomenon in natural sciences [2, 3, 4] and deeply entangled with the notion of phase changes, i.e., when change in some control variable (e.g., temperature) yields systematic changes in a system's underlying structure (e.g., formation of hexagonal configurations in a crystal) and simultaneously affects several of its properties. We argue for a similar characterization of emergence in machine learning: identifying systematic changes in a model's behavior that influence its downstream abilities and lead to sudden performance improvements. For example, learning a language's syntax will affect all downstream capabilities where coherent, grammatically correct generations are necessary.

Taken together, the orthogonal explanations and disparate results above have resulted in emergence becoming an extremely unclear phenomenon in machine learning. At its core, however, we claim that the concept has never been defined in prior work. This has arguably led to distinct mechanisms causing sudden changes in model performance to all be labeled as "emergence". What is the *phenomenology* that this term is meant to capture in the context of neural networks? Is it merely a sudden increase in performance with scale, or *broader than that*? Given *a* reasonable definition, can we show, even if in a simplified system, that emergent capabilities are commonplace and can we use the system's simplicity to better *understand what drives their sudden learning*?

This work. To address the questions above, we propose a phenomenological definition for

emergence and try to understand what drives it in a toy task of learning formal languages [232, 12, 233, 234, 235, 236, 237, 89, 238]. Specifically, we argue three characteristics should be observed to claim a capability is emergent (see Def. 7): beyond (i) sudden performance improvement for a specific task, we claim emergence is more likely to represent a meaningful concept if (ii) performance on several tasks improves simultaneously and (iii) there are precise structures underlying the data-generating process learned by the model at the point of emergence. The intuition, borrowed from the study of emergence in other fields (see Fig. 3.1), is that if multiple tasks witness improvement in performance, there is likely some shared structure to them and the model acquires this structure at the point of emergence. For example, when in-context learning emerges in LLMs, precise context-sensitive structures emerge and interact with general-purpose capabilities, leading to in-parallel improvement in several downstream tasks' performance [222, 101, 239, 240, 241]; thus, in-context learning can be deemed an emergent capability under the scope of our definition. In this sense, understanding emergence can be formalized as a study of identifying structures the model acquires at the point of sudden learning of *a set of* capabilities, and understanding why that structure is relevant to said capabilities. Adopting this viewpoint, we make the following findings in our experiments.

- **Formal Languages as an Experimental System for Studying Emergence.** We define a probabilistic context-sensitive grammar (PCSG) with type constraints that allow an entity or a subject in a sentence (e.g., `man`) to be seen in the context of only a predefined set of properties (e.g., `walk`). We train models to perform minimalistic reasoning tasks over samples of this language and find their data scaling curves simultaneously show sudden learning across several metrics.
- **Learning of general data structures underlies simultaneous jumps in specific metrics.** We find points of sudden change for metrics evaluating individual tasks correlate with the model learning two relevant structures that underlie the language: grammatical rules and type constraints. Despite the simplicity of our setup, we claim learning of such general structures is what leads to sudden growth in the performance of narrower tasks where these structures are important.
- **A percolation model predicts the scaling of when capabilities emerge.** We propose a formal model grounded in the theory of graph percolation [242] that captures our experimental observations, and show that if we can describe the structure the model is learning at the point of emergence, a predictive theory for sudden learning can (at times) be constructed—analogous to theories of phase transitions in physics; see Fig. 3.1.

3.2 Related work

Explaining emergence. Focusing on the sudden learning characteristic of emergent capabilities, a few recent works have tried to explain the factors driving this phenomenon. For example, compositionality has been implicated for having a "multiplicative" effect on a model's performance, where the argument is that a model cannot perform well on a compositional task until the abilities needed to perform individual tasks involved in that composition are acquired [86, 172, 243, 214, 101, 244, 245]; when they are acquired, performance suddenly grows. A few papers have also shown that learning of specific capabilities (i.e., ones not compositional in nature) can be sudden [230, 246, 218, 219, 247]. In contrast, Schaeffer et al. [208] argue emergent scaling curves are a consequence of poorly defined, discontinuous evaluation metrics, and the seemingly sudden learning goes away once partial, continuous credit is given to the model. We emphasize that if the structure of a task is ignored, it is certainly easy to define arbitrary continuous metrics for a task; however, such metrics are unlikely to help measure progress toward learning a task. For example, consider the addition of two numbers, say 10 and 11, and the metric called *token edit distance* [208] that assesses the average distance between digits in the model's output, denoted $\texttt{xy}$, from the ground truth of 21; that is, $^{(|\texttt{x}-2|+|\texttt{y}-1|)}/_{2}$. For both $\texttt{xy} = 22$ and $\texttt{xy} = 11$, this metric equals 1; however, clearly 22 is a better approximation for the ground truth of 21. That is, once we account for the structure of the task, i.e., the fact that error in the most significant digit should be penalized more than error in the least significant one, we see limitations in token edit distance as a metric for assessing a model's ability to add numbers. We argue claims relating emergence to sensitivity of metrics can be confounded by use of metrics that do not respect the structure of the task.

Grokking vs. Emergence. We focus on the effect of data scaling on a model's capabilities; often called 'learning curve' or 'data scaling' analysis [248, 249, 250, 251, 252, 253, 254]. On surface, this might look similar to the seemingly related phenomenon of grokking [173, 255, 256, 257, 210, 258, 259], wherein a model's performance on a task rapidly improves long after it has fit the training data. However, we emphasize that we focus on an *online learning* setting in our experiments, i.e., a given sample is unlikely to be seen multiple times during training. Emergence is generally studied in such online learning scenarios. Since there is no distinction between train versus test data in such a setting, we argue mechanistic explanations of grokking identified in past work that involve a perfect training data memorization phase [260, 261] are unlikely to help explain our results of emergence under data scaling in an online learning scenario.

3.3 A Phenomenological Definition of Emergence

To analyze emergence, we first establish what we mean by the term for the purpose of this work. Specifically, we define emergence in a phenomenological manner, i.e., by assembling the characteristic properties associated with scaling curves claimed to depict emergent learning. We emphasize our definition is merely *a* definition for emergence, and does not necessarily represent all possible perspectives [262]. For example, often model capabilities that arise despite any explicit supervision are called emergent in self-supervised learning [263, 264, 95]. As our goal is to analyze the effects of scaling, regardless of supervision protocol used, we do not try to capture this property.

Definition 7. ***(Emergence of a capability.)*** *We say a capability $\mathcal{C}$ is emergent with scaling along a relevant axis (e.g., amount of data, compute, parameters) if:*

- ***P1:*** *nonlinear improvement occurs in the performance of a task where $\mathcal{C}$ is required;*
- ***P2:*** *multiple tasks simultaneously show nonlinear performance improvement; and*
- ***P3:*** *the model acquires a structure underlying the data generating process that is instrumental to learning the capability $\mathcal{C}$, and nonlinear progress in $\mathcal{C}$'s learning directly correlates with the learning of said structure.*

The definition above assigns a broader meaning to emergence than mere sudden performance improvement on a narrow task: it argues there should be precise structures learned by the model that have downstream effects on *several* capabilities, hence leading to sudden improvements in performance on several tasks. Note that we intentionally leave the notion of 'structure' informal in the definition. The salient property of a structure is that if a model learns it, downstream tasks should become easier to perform. For example, a fine-grained notion of a structure can be previous token and copy attention heads that lead to in-context learning [229, 265, 228]; a more coarse-grained structure can include the model learning the syntactical rules of a language that help it with generation of coherent language and hence with any task where coherence is important [230]. *In this sense, what is emergent is a structure, and what is observed is a change in the model's capabilities.* Hypothesizing what this structure is by identifying shared characteristics of a set of tasks that simultaneously show sudden learning, one can likely develop an evaluation meant to precisely gauge learning of the corresponding structure and hence infer at what point an independent training run will show sudden improvements on these tasks.

We note the intuition for Def. 7 comes from prior work in the fields of complex systems and physics [2, 3, 4], from where the term has sought its inspiration in recent machine learning literature [215, 101]. Therein, emergence describes the scenario where rapid changes occur in a

system's properties as some control parameter is varied. A range where the system's properties change relatively smoothly is called a phase, and a change of phase with a change in the control variable is called a phase transition. A crucial step in studying emergence in physics is identifying an *order parameter*—a measure that captures the formation of some specific structure in the system such that the development of this structure is what alters the system's properties and drives a phase transition. For example, in Fig. 3.1a, a system of particles transitions through phases (solid, liquid, gas) as the temperature is changed; the formation of a crystalline structure with the decrease in temperature can be identified by analyzing the bond-orientation order parameter, while the liquid-to-gas transition can be described by a jump in particle density. We argue that we must similarly define order parameters for studying emergence in neural networks as well, i.e., we must develop evaluation measures that are focused towards detecting the learning of *specific structures* that are *generally of use* to several downstream capabilities.

3.4 Formal Languages as a Model Experimental System to Study Emergence

Having defined our perspective on emergence, we now define a toy experimental system that allows us to precisely study the concept in a controlled setting. We note that our focus will be on emergence under data scaling in an online learning scenario (i.e., a sample is unlikely to be seen multiple times). To this end, we follow recent work on understanding language modeling and use formal languages to define our experimental setup [12, 89, 266, 195, 234, 236]. As discussed in detail next, the formal language we use in this work is (minimally) context-sensitive, with underlying syntactical rules defined using a context-*free* grammar and context-sensitivity enabled through posthoc *type constraints*. The grammar and type constraints serve as two structures that underlie our language, and, as we show in Sec. 3.7.1, their learning bottlenecks learning of other, narrower capabilities.

Definition 8. ***(Grammar.)*** *Consider a set of symbols* $\texttt{T}$ *(also called terminals). A symbolic string* σ *is designed by chaining symbols from* $t \in \texttt{T}$*; e.g.,* $\sigma = t_1 t_2 \ldots \texttt{EOS}$*, where* $\texttt{EOS}$ *is a special symbol that marks the end of string* σ*. The set* $\texttt{T}^*$ *denotes all possible strings defined using symbols from* $\texttt{T} \cup \texttt{EOS}$*. Meanwhile, a set of symbols* $\texttt{NT}$ *(also called non-terminals) define production rules* $\texttt{Q}$ *of the grammar, represented as* $A \to \alpha\beta$*, where* $A \in \texttt{NT} \cup \texttt{S}$ *is called the LHS and* $\alpha, \beta \in \texttt{T} \cup \texttt{NT}$ *is called the RHS of the rule;* $\texttt{S}$ *denotes a special 'start' symbol. A probabilistic context-free grammar* $\texttt{G}$ *is defined as a randomized process that first maps* $\texttt{S}$ *to a set of non-terminals according to rules* $\texttt{Q}$*, and then repeats the process over these intermediate*

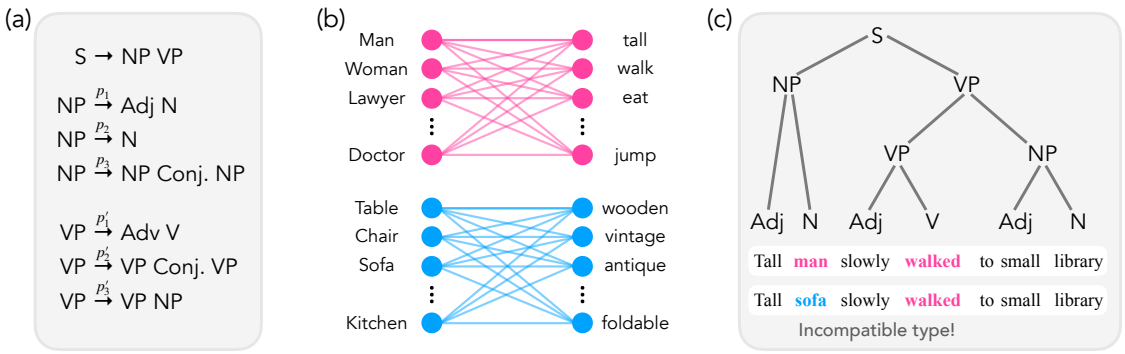


Figure 3.2: **Grammar and type constraints to define our formal language.** (a) We use a PCFG to define our language's grammar (shown rules are examples; see App. B.4.3 for precise details). The grammar's terminals are parts-of-speech from English and yield symbolic sentences that can be populated by tokens from the language's vocabulary. (b) Akin to natural language, wherein properties of an entity constrain sentences seen in a dataset corresponding to that entity, we define constraints (called type constraints) on our language that restrict which tokens can be seen together in a sentence. These constraints map entities to descriptive or relative properties, hence restricting which descriptive adjectives and verbs are valid for an entity. (c) Once a symbolic sentence is sampled from the grammar, we populate it with tokens from the language while respecting the type constraints. Training on string from this language in fact shows that the model deems sentences that do not respect type constraints to be extremely unlikely (see App. B.4.2).

*non-terminals by randomly sampling rules for which previously sampled non-terminals are the LHS. The process continues until only terminal symbols constitute the RHS. This yields a string $\sigma \in \Sigma$, where $\Sigma \subset \mathtt{T}^*$ denotes the set of all strings that can be sampled from $\mathtt{G}$. We say a randomly sampled string $\sigma \in \mathtt{T}^*$ is **grammatical** if $\sigma \in \Sigma$.*

For a more detailed treatment, see App. B.1.1. The terminal symbols used in our work include standard parts-of-speech from English, specifically: subjects, objects, verbs, adjectives, adverbs, conjunctions, determiners, and prepositions. Multiple short phrases can be combined together via conjunctions and verbs to form longer sentences (e.g., a verb can connect a subject phrase and an object phrase). We often use the term 'entities' to jointly refer to subjects and objects. We also distinguish between attributive and predicative adjectives, using the term adjectives to solely refer to adjectives that are attributive in nature and the term 'descriptors' to refer to adjectives that are predicative. Overall, the grammar $\mathtt{G}$ yields symbolic strings that are solely comprised of parts of speech (see Fig. 3.2 (a)); e.g., our grammar might yield a string like `adjective subject adverb verb preposition adjective object`. We next map these symbolic strings to our language.

Let $\mathcal{V}$ denote the vocabulary of our language $\mathcal{L}$. Each token $v \in \mathcal{V}$ has a symbolic role

$t := \texttt{role}(v)$ associated with it. The set of possible roles is denoted $\texttt{T}$, which is equal to the set of terminal nodes of grammar $\texttt{G}$. Thus, one can define a *context-free* language by simply sampling a string σ from $\texttt{G}$, and then replacing its terminals with tokens from the vocabulary that match the roles specified by symbols in σ. For example, the example string above can be resolved as `Tall man slowly walked to short building`. However, natural language is rich with constraints defined by the physical properties of an entity, which thereby restrict which tokens are seen in the context of which other tokens, hence yielding *context-sensitivity*. For example, one does not expect to see a sentence `Tall telephone slowly walked to short building`, since the entity `telephone` is neither expected to be `tall` nor the ability to `walk`. We develop an abstraction for such constraints by representing them as a bipartite graph (see Fig. 3.2 (b,c)).

Definition 9. ***(Type Constraints Graph.)*** *Let a* ***property*** $\texttt{k}$ *be a binary variable; the set of all properties is denoted* $\texttt{K}$*. Properties can be either* ***descriptive*** *(used to define descriptors; e.g.,* ***tall****) or* ***relative*** *(used to define verbs; e.g.,* ***walk****). A* ***concept class*** $\texttt{C}$ *is defined via the set* $\texttt{K}_\texttt{C} \subset \texttt{K}$ *that denotes which properties are valid for that class. When the properties in* $\texttt{K}_\texttt{C}$ *take values, we get an entity* e *from the class, denoted as* $e \in \texttt{C}$*. Entities have unique identifiers associated with them to help define subjects and objects in a sentence. The set of all possible entities is denoted* $\texttt{E}$*. The* ***type constraints graph*** $\mathcal{G} := (\texttt{E}, \texttt{K}, \texttt{I})$ *is a bipartite graph over entities and properties in the language whose edges* $\texttt{I}$ *denote whether an entity* $e \in \texttt{E}$ *possesses property* $\texttt{k} \in \texttt{K}$*.*

As an example, consider the class of `Humans`, which includes entities connected to properties like `tall`, `right-handed`, etc.; an entity from `humans` will be assigned a subset of these properties. When sampling sentences from our formal language, the type constraints will restrict which tokens can be seen together, i.e., which descriptors and verbs go with an entity, hence yielding context-sensitivity and making $\mathcal{L}$ a probabilistic context-sensitive language. Given two randomly sampled entities from the same class, they can be expected to share a subset of properties, giving a signal to the model trained on $\mathcal{L}$ that these entities are related (i.e., they belong to the same class).

3.5 Learning Tasks and Experimental Setup

Having described our language $\mathcal{L}$, now we briefly discuss our experimental setup (see App. B.3 for details). We train a GPT architecture model [267] with the standard autoregressive language modeling objective. Data is sampled "online", i.e., we sample a fresh batch of strings every iteration from $\mathcal{L}$. Unless mentioned otherwise, $\mathcal{L}$ is constituted of $|\texttt{E}| = 900$ entities

and $|\mathtt{K}| = 18000$ properties, equally and disjointly distributed over $|C| = 10$ classes, and with edges connecting entities to $p = 0.15$ fraction valid properties of a class in a uniformly random manner; results ablating these settings are in App. B.4. Before being fed into the model for training or evaluation, strings sampled from the language are restructured into a format that enables the specification of particular tasks (see Fig. 3.3). Specifically, we train the model to learn the following tasks with 80/10/10% splits.

- **Free generation:** Produce a valid string, i.e., one that respects the grammar and type constraints.
- **Unscrambling:** A string is sampled from $\mathcal{L}$ and randomly permuted; the model is expected to unscramble it. This task is known to show sudden learning in LLMs [101].
- **Conditional Generation:** A set of tokens corresponding to entities or properties are shown to the model, which is expected to generate a string combining these tokens in a valid manner.

Evaluation Protocols. Given an input x, which may correspond to any of the three tasks above, denote the model output as $f(x)$. Let $\delta(.)$ be an indicator variable that evaluates to 1 if its input is logically true. We track several metrics throughout training, as described below. We often decompose evaluations according to strings of two types: (i) *descriptive*, i.e., ones that describe that an entity possesses a descriptive property, and (ii) *relative*, i.e., ones that demonstrate a subject, object, and verb can be combined to create a valid sentence. All results are averaged over 3 seeds.

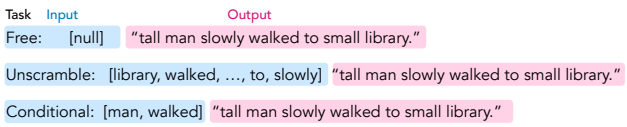


Figure 3.3: **Task definitions.** Our model is trained and evaluated on three types of tasks. (i) Free generation: the model generates sentences with correct grammar. (ii) Unscrambling: the model is provided with a set of words and must reorder them to form valid sentences. (iii) Conditional generation: model is given a set of entities or properties and must generate valid sentences using them. Note that examples in the figure are merely indicative. See App. B.1.1 for details.

- **Grammaticality/Type Check.** Used for evaluating free generation. Grammaticality involves checking whether model output follows the underlying grammar $\mathtt{G}$, i.e., $\delta(f(x) \in \Sigma)$. Type checks involve first extracting subjects, objects, and properties from the sentence and then evaluating whether this set of tokens is allowed in the context of each other. We decompose type checks as *descriptive* (do entities and descriptors match), *relative* (do subject, object, and verb match), and *all* (product of all constraints, including adjectives and adverbs).
- **Exact Match / Per Token Accuracy.** Used for evaluating unscrambling. Assume

the ground-truth unscrambled sentence y has l tokens. We compare whether the model output exactly matches the ground-truth $\left(\Pi_{i=1}^{l}\delta(y_i = f(x)_i)\right)$ or the per-token match ratio $\left(1/l \sum_{i=1}^{l} \delta(y_i = f(x)_i)\right)$.

- **Conditions Satisfied.** Used for evaluating conditional generation. If the model is expected to produce a sentence with m conditioning tokens $\{v_{c_1}, \ldots, v_{c_m}\}$, we analyze how many of those tokens are present in $f(x)$, i.e., we evaluate $1/m \sum_{i=1}^{m} \delta(v_{c_i} \in f(x))$.
- **Average Probability of Valid Tokens.** Used for evaluating descriptive type constraints in free generation and unscrambling. Specifically, we sample a sentence from $\mathcal{L}$ that remarks on an entity possessing a property (i.e., a descriptive sentence), and then evaluate the probability of the property being the next token when this sentence is inputted to the model. For example, let $x = \texttt{The fire was large}$. We evaluate $\texttt{Pr}\left(\delta(f(x)_1 = \texttt{large})|x_{-1}\right)$, where x_{-1} denotes the sentence up to the last token and $f(x)_1$ denotes the first token predicted by the model.

Other Evaluations. We perform several other evaluations, e.g., computing log-likelihoods of valid versus adversarially perturbed sentences that do not follow grammar or type constraints to check how well the model follows our language; comparing the distribution of lengths and parse tree depth for model's generations with the language's; analyzing grammaticality and type check accuracy for unscrambling and conditional generation; rank of property predictions; and the evolution of attention maps across time. These results are reported in App. B.4.

3.6 Results: Emergent Capabilities in Formal Language Learning

We now evaluate (i) whether our setup demonstrates emergence (see Def. 7), and (ii) whether we can extract insights into the mechanisms of what leads to emergence. In the following, we often use the terms "phase" and "phase change"; see discussion around Def. 7 for context on these terms.

3.6.1 Phases of Language and Capabilities Acquisition

We plot the model's performance as a function of training iterations. Since we are in an online learning, constant stepsize setting, this analysis corresponds to studying the effects of data scaling. Results are reported in Fig. 3.4 and show there are three phases to the learning dynamics.

Phase 1: Grammar acquisition. We find the model first learns to produce grammatically correct sentences, as measured by the grammaticality measure defined in Sec. 3.5. This process is relatively rapid, as we see the model starts generating grammatically accurate sentences in a short period of approximately 100 iterations; attention heads also rapidly evolve and reflect the parse structure of a sentence (see App. B.4.6) In this regime, however, the narrower tasks of unscrambling and conditional generation exhibit poor performance. However, precisely when grammaticality improves, we find that per-token accuracy starts to improve. *This indicates that the model learning a broad structure underlying the data (i.e., grammar) has an impact on the learning of other capabilities.*

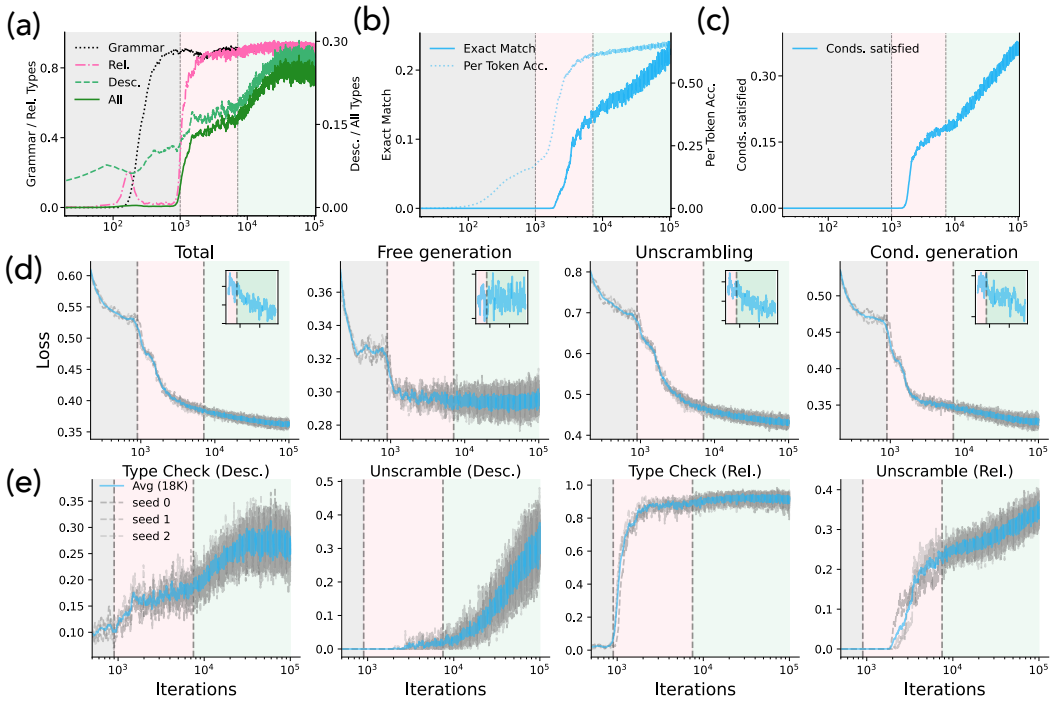


Figure 3.4: **Learning of structures drives emergent capabilities.** For a detailed discussion, see main text. **(a) Grammaticality and Type Check evaluations** as a function of iterations or data (iterations×batch-size). We see phases in the learning dynamics corresponding to *emergent acquisition of structures underlying our language*: grammar (black), relative type constraints (pink), and descriptive type constraints (green), shaded gray, pink, and green respectively. **(b, c) Performance on Unscrambling and Conditional Generation.** After a slight delay from phase boundaries, we see sudden improvements in the performance of individual tasks. **(d) Learning curves.** Loss also shows sudden changes at phase boundaries corresponding to the acquisition of structures. **(e) Performance on descriptive/relative sentences.** Decomposing by sentence type, we find a sublinear growth in descriptive type checks drives performance boost on descriptive sentences for the unscrambling task.

Phase 2: Acquisition of relative type constraints. At around 1000 iterations, we find there is a sudden increase in the model's performance on relative types from essentially zero to perfect accuracy; precisely at this point, we find the loss for all tasks, especially free generation, shows a sudden drop. Interestingly, we find this sudden improvement occurs precisely at the point where the model reaches its maximum performance on grammaticality for the first time. That is, *as soon as the first structure underlying the data is learned, the model rapidly learns the next relevant structure of relative type constraints.* Improvement occurs in descriptive constraints as well (and hence the overall Type Check performance), but hovers around slightly above 0.1. This is expected, since with $|C| = 10$ classes, if a model produces grammatically correct sentences, it will achieve a random performance of $1/|C| = 0.1$ on descriptive type checks. This also implies that the model is primarily relying on its syntactical knowledge and does not respect descriptive type constraints much.

During this phase, we see that shortly after the phase change, there is a sudden increase in performance for both unscrambling and conditional generation, across all metrics. These tasks' losses also show another loss drop occurs at this point; though the drop seems smoother in the total loss, likely due to averaging effects [247]. As shown in Fig. 3.4e, we find that this performance improvement is driven by sentences that require primarily correctness of grammar and relative type constraints, i.e., knowledge of which descriptors are associated with an entity is not necessary to perform well on these sentences. This also explains the loss drop seen in Fig. 3.4: *once grammar and relative type constraints are learned, the model learns to use them to solve inputs that do not require knowledge of descriptive properties, leading to a sudden improvement in both loss and accuracy.*

Phase 3: Learning of descriptive type constraints. During Phase 2, we find that the model's performance on descriptive type checks witnesses minimal improvement. However, as training proceeds, the model enters a third phase at whose boundary we see a sudden change of slope from a saturation region to approximately proportional growth in the performance of descriptive type checks with log-amount of data/iterations (i.e., sublinear growth). With a slight delay, we see a similar effect kicks in for the unscrambling and condition generation tasks as well, which start to show approximately linear improvement with log-amount of data/iterations. Zooming in at this point (see inset plots in Fig. 3.4), we see there is in fact a small, but nevertheless noticeable, loss drop in the unscrambling and condition generation tasks. We emphasize that since the model has seen merely an order of 10^4 iterations up to this point, if we assume the model can perfectly learn in only a few observations that some entity and a property can be seen together in a sentence, then our experimental setting can on average see only up to 15% performance (which matches the observed performance) and 20% at best (see App. B.3.0.1; the argument is that only a subset of pairs is shown during training,

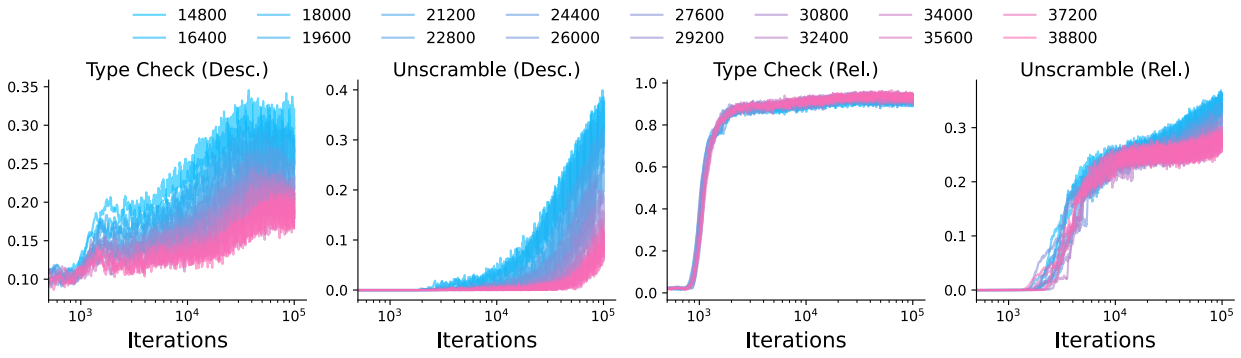


Figure 3.5: **Effect of scaling number of descriptive properties.** Scaling the number of descriptive properties in our language (see legend), we find relative type checks and unscrambling performance for relative sentences are essentially unaffected by the number of properties. Meanwhile, both descriptive type checks and unscrambling performance for descriptive sentences show a change in performance and delay in transition points. Despite these effects, we find the geometry of performance curves is extremely consistent; for descriptive type checks, this geometry indicates a memorization to generalization picture.

restricting maximum performance). However, as the model enters and progresses through the third phase, it shows a much larger rate of improvement and reaches ∼30–35% performance, indicating it is generalizing beyond the pairs of entities and properties it has seen together during training. If the model were simply relying on memorized knowledge, the performance values we observed would not be feasible. *We thus claim that the model is implicitly inferring, based on the structure of the type constraints graph, which properties and entities constitute a valid context.* This suggests a memorization effect is at play during Phase 2, and the end of this phase corresponds to a transition from a memorizing to a generalizing solution.

Takeaways. While the results above are for a specific configuration, we find the claims consistently generalize to an extremely broad array of experimental settings (see App. B.4). Specifically, we consistently see that the model first learns the two structures underlying our language (grammar and type constraints), and then witnesses performance improvements on narrower tasks in the data. Drawing on the physics analogy before, we emphasize that in this sense grammaticality and type constraints serve as our order parameters (here order is the grammar and types), and are sufficient to approximately predict when sudden performance improvements will be seen for a task. We also note that our results are not sensitive to specific choice of metrics, as shown by loss curves and several other metrics discussed in App. B.4.

3.6.2 Effect of Number of Descriptive Properties on Language Acquisition

Given the above picture of a model's learning dynamics, we next ask how the phase boundaries change with an increase in the number of properties $|\mathtt{K}|$ (see Fig. 3.5). We intentionally scale only the number of descriptive properties and hypothesize the learning of both grammar and relative type constraints to not be affected by this change. We relegate grammar learning to appendix (see App. B.4.2), which, as expected, is not affected by $|\mathtt{K}|$ since it is an entirely independent structure from type constraints. However, we see that even relative type constraints' learning is not affected by the increase in descriptive properties, indicating the model deems them (justifiably) to be independent structures. Focusing on descriptive type constraints then, we see performance curves for descriptive type checks and unscrambling performance on descriptive sentences are indeed affected, achieving higher values for fewer properties (i.e., the easier task). We further make two more interesting observations. (i) *The point of transition from memorization to generalization is delayed as we increase the number of properties.* This is most prominently seen in the delay in the transition point where the ability to unscramble descriptive sentences emerges. (ii) *We find the geometry of these performance curves are extremely similar to the geometry we observed for our base setting studied in Sec. 3.6.1*, indicating despite the increase in difficulty of the task, the same learning dynamics are at play. We devote the next section to formulate a hypothesis justifying these observations.

3.7 A Percolation Model of Emergence

We next propose a framework for modeling the emergence of capabilities that require a model to compose unseen entities and descriptive properties, e.g., learning descriptive type constraints, which, beyond allowing a model to produce accurate free generations, will aid with narrower tasks like conditional generation and unscrambling. We argue the relevant structure to analyze for this purpose is the *concept class*: if a model understands what entities and properties belong to the same concept class, regardless of whether they have been seen together in a sentence, it will deem their co-occurrence to be valid. We thus develop an abstraction for concept classes as bipartite graphs and cast their learning as a problem of percolation on a bipartite graph.

Recall that a concept class is defined as a set of entities that are expected to have shared properties (see Def. 9). The question is whether upon sub-sampling pairs of entities and properties from a concept class, can the model learn that, in fact, all pairs of entities and

properties are valid and compose the concept class. For instance, in the case of a concept class such as `human`, the set of entities can include humans with different genders (e.g., `man`) as well as human-associated entities such as a `lawyer` (see Fig. 3.2). The corresponding properties for the `human` concept class will be, for example, `walk, jump, tall`. A `man`, being `human`, is expected to have all these properties, although strings specifying these properties for a `lawyer` may be rare or even absent in the training data. We are interested in the case where the data, such as strings, includes examples of these pairs of entities and properties. We can represent this by a matrix whose rows and columns represent the entities and the properties, and the matrix values indicate the quantity or density of data available for each composition, such as an *entity-descriptor* pairing. We call this matrix the **concept density matrix**.

Definition 10. ***(Concept Density Matrix.)*** *Let D be an $|\mathtt{E}| \times |\mathtt{K}|$ matrix with real-valued entries between 0 and 1, inclusive. Each entry D_{ek} represents the density for the entity and property pair (e, k) (e.g., the amount of data that represents the specific composition), where $e \in \{1, ..., |\mathtt{E}|\}$ and $k \in \{1, ..., |\mathtt{K}|\}$ are the indices of the entities and properties, respectively.*

For example, consider the case where there are three values of entities and properties ($|\mathtt{E}| = |\mathtt{K}| = 3$), with entities (rows) being {`Man`, `Lawyer`, `Telephone`}, and properties (columns) being {`Walk`, `Stoic`, `Ring`}. The corresponding D can be:

$$D = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{3.1}$$

A common composition such as `Man walking` will lead to a value of 1 at the intersection of `Man` and `Walk`, i.e., $D_{00} = 1$, where D_{ij} denotes element at row-i and column-j. Conversely, a highly unlikely composition like `Lawyer ringing` will be absent in the dataset, and will be represented by a zero at the respective matrix position, i.e., $D_{12} = 0$. We can also assume for example that `Man ringing` or a `Telephone walking` are rare, which yields $D_{13} = D_{31} = 0$.

We next introduce the **concept propagation matrix** to model the inference of novel entity-feature combinations from the incomplete data represented in D.

Definition 11. ***Concept Propagation Matrix.*** *An n-th order concept propagation matrix ($n \geq 0$) is defined as $T^{(n)} = (DD^T)^n D = C^n D$, where $C := DD^T$.*

The concept propagation matrix can be intuitively understood using a bipartite graph, as shown in Figure 3.6a. A bipartite graph in this case is a sub-graph of the **type constraints graph** (see Def. 9 and Fig. 3.2), where one set of nodes represents entities while the other

represents properties, and edges indicate the presence of entity-feature pairings in the training data. The strength of connectivity of the graph directly corresponds to the values in the concept composition propagation matrix, $T^{(n)}$. Specifically, if two concepts are connected by a path of minimal length $2k+1$ (i.e., the shortest path between them alternates between the two sets k times), the corresponding entry in $T^{(n)}$ becomes non-zero only for $n \geq k$. That is, the number of propagation steps n required for the entity and property pair to be associated is determined by the minimal number of hops needed to connect the two nodes in the graph. Conversely, if two nodes belong to disconnected regions of the graph, their composition remains fundamentally unlearnable, regardless of the order of propagation, indicating that composition is not valid. This is reflected by the corresponding entry in $T^{(n)}$ remaining zero for all n. In the bipartite graph, this amounts to having two distinct clusters that are connected within themselves but not across each other. For example, in the case where the concepts represented by the first and third rows belong to disconnected regions of the graph, and consequently, their composition (e.g., `Lawyer ringing`) cannot be achieved even after an infinite number of hops between nodes. We call such a situation *learning of a concept class*: the system understands that `Man` and `Laywer` are both humans, whereas `Telephone` is not. Our experiments show that the model deems sentences composing entities and properties from incorrect classes to be much less likely than the correct ones (see Fig. B.4.3), i.e., the model indeed learns the structure of concept classes.

3.7.1 Percolation Transition on Descriptive Constraints

Using the bipartite graph framework, the generalization, or the learning of the concept class, can be defined as the situation where a large cluster of entity-property connected pairs arises despite the sparse concept density matrix. A critical aspect to examine is the proportion of the inference matrix values where $T_{ek}^{(\infty)}$ is non-zero, out of the total possible pairs $|\mathtt{E}| \times |\mathtt{K}|$. This particular scenario aligns with the bond percolation problem on a bipartite graph. In bond percolation, we investigate how the largest connected cluster's size varies with the probability p of each edge (bond) being present. In a typical setting, there exists a critical threshold value, $p = p_c$, called the percolation threshold. Below this threshold ($p < p_c$), the graph typically exhibits a disconnected phase characterized by the absence of extensively connected clusters, with most nodes either isolated or part of smaller clusters. Above this threshold ($p > p_c$), the graph transitions to a connected phase, significantly increasing the likelihood of a vast connected component spanning a large portion of the graph. This shift from a predominantly disconnected state to one with a macroscopic cluster is a defining characteristic of the percolation process, and this transition sharpens as the number of

components in the system increases. See Fig. 3.6 for a schematic.

In a simple percolation scenario, where connecting edges are selected randomly on the graph with probability p, the percolation threshold is obtained as $p_c \simeq \sqrt{1/|\mathtt{E}||\mathtt{K}|}$ for large $|\mathtt{E}|$ and $|\mathtt{K}|$ [3], which means that when around $\sqrt{|\mathtt{E}||\mathtt{K}|}$ edges are connected (out of the total $|\mathtt{E}||\mathtt{K}|$) there is a qualitative change in the growth of the cluster size. For $p > p_c$, the number of nodes included in the connected cluster will become macroscopic, meaning that the probability that a randomly selected pair of an entity and a property is connected will be finite. We present in Appendix B.2 the derivation of the percolation threshold for bipartite graphs that are uncorrelated, and how the cluster size (i.e., number of nodes in the largest connected graph) scales as $\sim (p - p_c)^\beta$ with $\beta = 1$ for usual cases.

We posit that the percolation threshold corresponds to the point at which our model generalizes from the sparse learning of pairs to a complete representation of the concept classes. When the number of edges surpasses the threshold, the model can infer novel compositions, even for entity-feature pairs that were not explicitly present in the training data. The model should also start to be able to discriminate between distinct concept classes beyond this threshold; in the community detection problem, for example in the stochastic block model [268, 269], the detection threshold for the partitions have the same scaling as p_c [270]. Since increasing the iterations through online learning should amount to increasing p (i.e., seeing more combinations in data), the iteration point at which the transition occurs in the model performance should be proportional to $\sqrt{|\mathtt{E}||\mathtt{K}|}$.

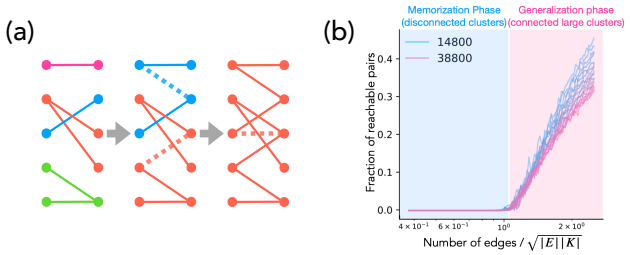


Figure 3.6: **Casting the ability to compose unseen concepts (e.g., entities and properties) as percolation on a bipartite graph.** (a) When only a fraction of the concept classes are included in the dataset and the edge density is low, nodes (e.g., entities and properties) form many disconnected clusters, indicated by different colors (left). As more concept classes are added (dashed edges) to the bipartite graph, the small clusters begin to merge (middle). With a sufficient number of edges, a macroscopic number of nodes become connected, forming a single cluster (right). (b) Our formalism establishes this transition as a second-order phase transition, where the size of the largest cluster increases non-linearly as the fraction of connected node pairs is scaled. Shown curves are from simulations on bipartite graphs with the number of nodes equal to the number of entities and properties in our formal language experiments (see Sec. 3.5 or App. E.2).

To check whether such theoretically predicted scaling can be seen in our experiments with formal language learning, we plot again the various performances of the model as a function of iterations divided by the square root of the number of properties (i.e., keeping

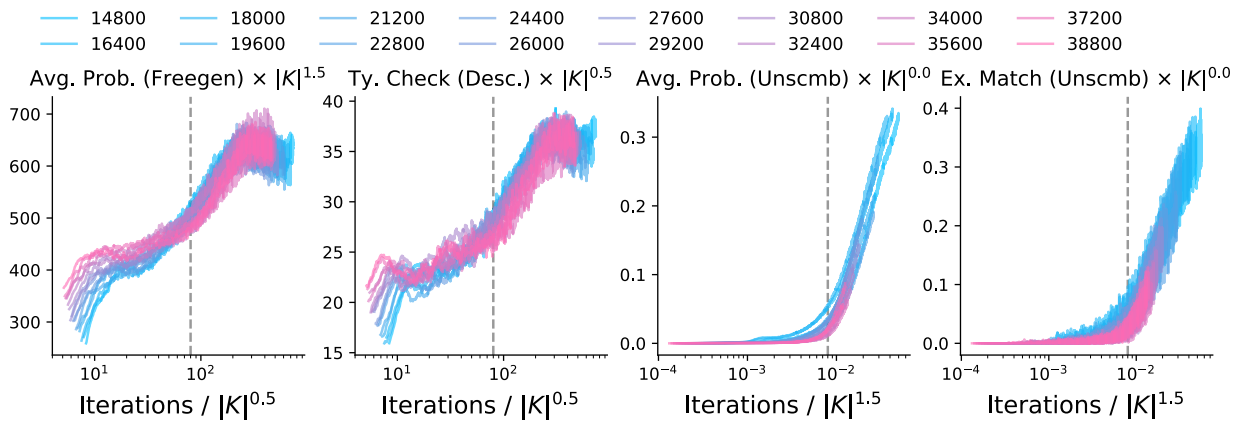


Figure 3.7: **Scaling of point of emergence matches with our theory.** We replot the result from Fig. 3.5 by rescaling the x-axis with a power of the number of descriptive properties; See App. B.5.1 for further discussion on the intuition underlying this visualization. For Average probability of generating a valid descriptive property given some entity in context under free generation and descriptive type check accuracy, we see a 0.5 exponent scaling yields a collapse of the transition point—this matches the toy model of percolation posited in Sec. 3.7.1. We also see a very clear scaling of the point where the ability to unscramble descriptive sentences starts to emerge, but with an exponent of 1.5.

number of entities constant). Results are shown in Fig. 3.7. *We find that indeed there is a growth trend in the descriptive type check metric and the average probability scores of free generation of descriptive sentences that occur at iterations proportional to* $\sqrt{|\mathtt{K}|}$. This is in contrast to the first large growth that is observed in these evaluations, which seems to be occurring at iteration numbers that do not depend on $|\mathtt{K}|$ (see Fig. 3.5 and Fig. B.45), likely because this step corresponds to the point where the model learns about syntax, which requires a constant amount of data irrespective of the number of properties (see Fig. B.10).

We also analyzed the scaling of the transition point for the narrower task of unscrambling upon change in the number of properties. Here, we found a different scaling of $|\mathtt{K}|^{3/2}$. We so far do not have an explanation for this scaling, but we believe it is likely due to interactions with learning of other mechanisms, as it seems to be sensitive to other characteristics of the language; e.g., when we increased $|\mathtt{E}|$ from 900 to 1800, the exponent shifted from 3/2 to close to 2 (see Fig. B.37), but remained 0.5 for descriptive type checks (see Fig. B.47). Given the task involves the composition of entities and properties too, we expect a transition and scaling effect to occur for unscrambling as well; however, learning the precise circuit to perform unscrambling over the learned grammar and type constraints will yield a delay (as was seen in experiments) that likely has some interaction with the complexity of the language. We leave explaining this scaling for future work.

3.8 Discussion

In this work, we take inspiration from other fields (e.g., physics and complex systems) and propose a phenomenological definition for emergence of capabilities in neural networks. Specifically, the definition argues that at the point of emergence, the model acquires general structures which are instrumental to the learning of specific, narrower capabilities; acquisition of such structures then leads to sudden performance improvement on several tasks (often with some delay). While relatively informal, this definition brings the notion of emergence in the context of neural networks closer to its meaning in physics, wherein the formation of systematic structures is known to drive *phase changes* that involve sudden changes in the system's properties. Characterizing these phase changes requires hypothesizing what the structure is, and defining an "order parameter" that can help gauge its change. Drawing on this definition and perspective, we then propose an experimental setup that involves learning of a formal language with two precisely defined structures—grammar and type constraints (what properties are valid in the context of what entities)—and a set of narrowly defined tasks. Defining order parameters for these structures (grammaticality and type checks), we find there indeed are phases in the model's learning dynamics, and the model suddenly acquires capabilities corresponding to the narrower scope tasks (unscrambling and conditional generation) close to these phase boundaries. Interestingly, the learning curves show a rather distinct geometry that remains consistent as we alter the number of properties in our language.

To explain these results, we propose a model that analogizes learning of type constraints in the formal language learning task to the problem of graph percolation. Drawing on the theory of percolation on bipartite graphs, which shows phase transitions in the formation of connected components on a graph, we argue this problem is similar to learning of concepts classes or type constraints in our setting, and hence should show scaling of the point of emergence where the model starts to follow descriptive type constraints that is of the order of $\sqrt{|\mathtt{E}||\mathtt{K}|}$. Our results show a strong qualitative match with this hypothesis. We also find an extremely clean scaling for other tasks' transition point, e.g., for unscrambling's results on descriptive sentences; explaining these results is left for future work.

While our goal in this work was primarily demonstrative, i.e., to develop a bridge with other fields studying emergence, we believe several exciting avenues now open up. For example, given that the whole point of the theory of phase transitions is that we can predict the point of emergence, can we draw on this rich literature to propose models for explaining and predicting emergent capabilities in neural networks? Can we go beyond the toy task of formal language learning studied in this work and analyze a more naturalistic setting? For example, can

we identify structures underlying emergent capabilities in open-source models, e.g., Pythia checkpoints [271], and demonstrate that our proposed perspective enables prediction of when capabilities emerge in LLMs? To begin, we can perhaps focus on capabilities that require similar knowledge acquisition and its compositional generalization on a downstream task.

Part 2: Latent Capabilities and Underspecification as Hurdles to Evaluations

Identifying vulnerabilities of a system requires designing well-defined evaluations. However, modern AI systems show peculiar behaviors that render this task challenging. For example, several recent works have shown how a minuscule change in the prompt of an LLM can elicit a broad spectrum of capabilities [68, 69, 70, 71]. This disallows the establishment of an explicit standard for safe usage for a model: a user may, possibly unintentionally, elicit model vulnerabilities by using a slightly different input to the model compared to the ones it was tested for. For example, as reported by Ganguli et al. [42], OpenAI produced a specialized GPT-3 model that was designed to play a video game (in collaboration with the designers of the game). When players were given access to the model, the expectation was that only capabilities relevant to the game will be exhibited by the model. However, by using prompts different from the ones expected by OpenAI, players circumvented the game-specific behavior and gained access to essentially all capabilities of the model. While this was an innocuous scenario, a broader question must be asked: why can minimal changes to an input elicit a wide range of model capabilities?

In this part of the dissertation, we try to better understand challenges in defining evaluations of generative models. We first identify and discuss a novel challenge: under a controlled setting, we show there often exist latent capabilities that, even if a model does not exhibit upon naive input prompting, can be elicited via alternative protocols (such as latent interventions). Thus, performance on a benchmark, which generally gauges input–output behavior becomes insufficient to assess model capabilities. We then analyze a known challenge: the input sensitivity of language models. Using the lens of underspecification and Bayesian hypothesis selection, we characterize how a model resolves underspecification to produce a specific output over other possible ones. This analysis allows us to propose a precise mechanistic explanation for why minimal change to an input can elicit drastically different behaviors from the model.

Contributions of Part 2

- **Emergence of competence before performance**. Assuming one can guess what capabilities a system might exhibit, defining evaluations that test for such capabilities is crucial for risk assessment. However, we find that for compositional capabilities, much before the model exhibits such capabilities (when prompted via an input to do so), there already exist latent interventions that can elicit them. These results add to our understanding of emergent capabilities and argue that beyond the "unknown unknowns", i.e., capabilities we cannot predict, emergence leads to the challenge of "unknown knowns": even for capabilities we expect the model to eventually learn, poorly defined evaluations can lead to such capabilities evading risk assessments. We thus advocate for evaluation protocols targeted towards latent interventions for stress-testing a model's capabilities.
- **In-Context Learning Dynamics in LLMs: Signatures of Bayesian Hypothesis Selection**. We design a task where there is no ground truth answer; instead, there are multiple plausible ways to respond to the input (similar to natural language). This is an example of input underspecification, i.e., given the input, it is unclear what the user intends and the model has to decide among multiple viable outputs to the input. We demonstrate in practical scale models that, on this underspecified task, models show signatures of Bayesian hypothesis selection. This explains the tendency of a model to quickly switch behaviors at the introduction of a few new tokens.

Author Roles

This part is based on a paper under submission at NeurIPS 2024 [90], co-authored with Core Francisco Park, Maya Okawa, Andrew Lee, Robert Dick, and Hidenori Tanaka; and another that was published at ICLR 2024 [87], co-authored with Eric Bigelow, Robert Dick, Hidenori Tanaka, and Tomer Ullman. In the first work, which was a follow-up to my prior work [86], I defined the project direction and narrative, established the experimental setup, co-led the section on latent capabilities and underspecification with Andrew and Maya, and wrote the paper. The section on learning dynamics was led by Core and Maya. In the second work, Eric and I defined the experimental setup by reading through literature on subjective randomness. Final experiments in this work were primarily carried by Eric; he and I co-wrote the paper.

CHAPTER 4

Competence Emerges Before Performance

4.1 Introduction

Modern generative models, such as text-conditioned diffusion models, show unprecedented capabilities [131, 272, 273, 122, 116, 117, 274, 275]. These abilities have led to use of such models in applications as valuable as training control policies for robotics [276, 277, 278] and models for weather forecasting [279], to as drastic as campaigning in democratic elections [280, 281]. Similar claims can be made for generative models of other modalities, e.g., large language models (LLMs) [20, 282, 283, 19], speech and audio models [284, 281], or even systems designed for enabling scientific applications such as drug discovery [285, 286]. Arguably, acquiring such general capabilities requires for models to internalize the data-generating process and disentangle the concepts (aka latent variables or factors of variation) underlying it [287, 207]. Flexibly manipulating these concepts can then enable generation of novel samples that are entirely out-of-distribution (OOD) with respect to the ones used for training [160, 288, 198, 175, 289].

As shown in prior work, modern generative models do exhibit signs of disentangling concepts underlying the data generating process and learning of corresponding capabilities to manipulate said concepts [113, 290, 291, 292, 293, 294, 295, 296, 297]. At the same time, contemporary work has argued that models' capabilities can be unreliable, arbitrarily failing for a given input and succeeding on another [298, 163, 164, 162, 165, 168, 169]. Thus, critical questions remain on how generative models acquire their capabilities (see Fig. 4.1): what determines whether the model will disentangle a concept and learn the capability to manipulate it; are all concepts and corresponding capabilities learned at the same time; and is there a structure to the order of concept learning such that, given insufficient time, the model learns capabilities to manipulate only a subset of concepts but not others?

This work. To address the questions above, we propose to analyze a model's *learning dynamics at the granularity of concepts*. Since performing such an analysis on realistic,

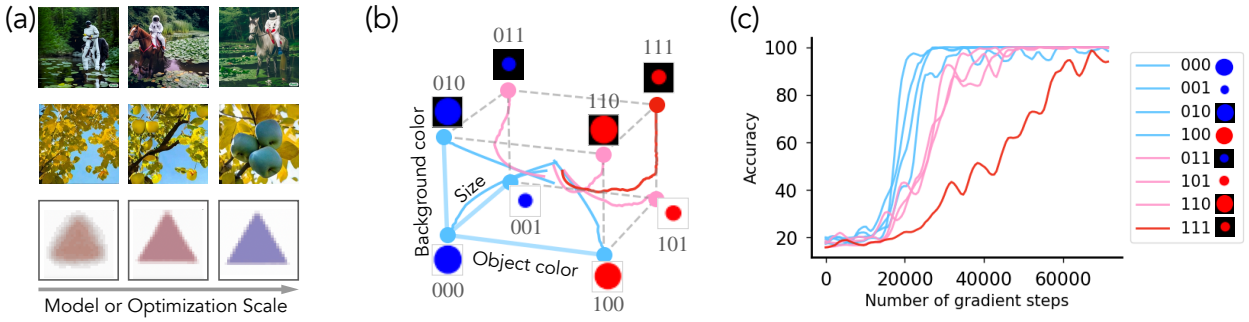


Figure 4.1: **Concept Learning Geometry underlies emergence.** (a) Top: A multimodal model learns to generate concepts in the order of "astronaut", "horse", and finally "riding" as it scales up (adapted from Yu et al.[5]). Middle: "blue square apple" is generated in the order of "apple", "square", and "blue" (adapted from Li et al.[6]). Bottom: Despite its simplicity, our model trained on synthetic data shows *concept learning dynamics* where it first learns "shape" and then "color". (b) Concept space is an abstract coordinate space where individual axes correspond to different concepts and a given point corresponds to a "concept class", i.e., a predefined collection of concepts (e.g., `large blue circles` on bottom left corner). Traversal along axes of the concept space yield change in a specific property of the sample (e.g., going from `large blue circle` to `large red circle` along `object color` axis). Trajectories show a model's dynamics in concept space for learning to generate classes shown in-distribution (blue nodes) versus out of distribution (pink / red nodes). As we show, dynamics in concept space are highly interpretable, enabling precise comments on which concepts the model learns first, why, and what order it follows. (c) Measuring how accurately a model generates samples from a given concept class, showing an order of concept learning: first background color is learned, then size, and then object color.

off-the-shelf datasets can be challenging, we develop synthetic toy datasets of 2D objects with different concepts (e.g., shape, size, color) that give us thorough control over the data-generating process and allow for an exhaustive characterization of the model's learning dynamics (see Fig. 4.1). Our contributions are as follows.

- **Introducing Concept Space.** We propose to evaluate a model's learning dynamics in the *concept space*—an abstract coordinate system whose axes correspond to specific concepts underlying the data-generating process. We instantiate a notion of underspecification in concept space and establish its effects on a model's (in)ability to disentangle concepts.
- **Concept Signal Dictates Speed of Learning.** We find the speed at which a model disentangles a concept and learns the capability to manipulate it is dictated by the sensitivity of the data-generating process to changes in values of said concept—a quantity we call *concept signal*. We show concept signals shape the geometry of a learning trajectory and hence control the overall order of concept learning.
- **Sudden Transitions in Concept Learning.** We use analytical curves to explain the

phenomenology of learning dynamics in concept space, showing the dynamics can be divided into two broad phases: **(P1)** learning of a *hidden capability*, whereby even if the model does not produce the desired output for a given input, there exist systematic latent interventions that lead the model to generate the desired output, and **(P2)** learning to generate the desired output from the input space.

Overall, while our results focus on a toy synthetic task and text-to-image diffusion models, we hypothesize a broader claim on *hidden emergence of latent capabilities* holds true: generative models possess latent capabilities that are learned suddenly and consistently during training, but these capabilities are not immediately apparent since prompting the model via the input space may not elicit them, hence hiding how "competent" the model actually is [299, 300]. Empirically, signs of "hidden capabilities" have already been shown in generative models at scale [301, 10, 302, 303, 304, 5], and our results help provide a formal framework to partially ground such results.

4.2 Related Work

Concept learning. The term concepts as used in this work is broadly equivalent to the notion of factors of variation from prior work on disentangled representation learning [207, 202, 206, 305, 306, 290, 307, 197, 308], and is motivated by use of the term in a similar sense in cognitive science [309, 310, 311, 312, 313, 314]. The focus of literature on disentanglement has been to prove identifiability results, e.g., when will a generative model trained on a dataset learn to invert the data-generating process, hence identifying the latent variables, i.e., concepts, that underlie it. Given the success of modern generative models, we argue, despite impossibility results from prior work, these models are in fact learning to disentangle some, if not all, concepts. Precisely what guides which concepts are disentangled however requires studying the learning dynamics—the target of our work.

Interpretability. A growing line of papers demonstrate highly interpretable representations of intuitive concepts exist in generative models, especially LLMs [315, 316, 317, 318, 319, 320, 321]. Similar work in this vein in image diffusion models has found semantic representations in various components of the model [303, 322, 323, 324, 297]: e.g., existence of linear representations for concepts such as 3D depth or object versus background distinctions [325, 326] . These papers further bolster our argument, modern generative models are truly inverting the data-generating process and identifying the concepts underlying it.

Competence vs. performance. In cognitive science, a system's competence on a task is often contrasted with its performance [300, 299, 327, 328, 329, 330]: competence is the system's *possession of a capability* (e.g., to converse in a language) and performance

is system's *use of that capability in concrete situations* [327]. For example, a bilingual person may generally converse in their primary language $\mathcal{L}$, despite possessing knowledge of another language $\mathcal{L}'$, unless it is crucial to use the latter—clearly, they are competent in both languages, but gauging their performance on $\mathcal{L}'$ requires appropriately "prompting" them to use it. One can analogize this distinction with remarks on a neural network possessing a capability versus us being able to elicit it on predefined benchmarks and measure their performance [331, 332, 333, 65]. For example, on the BigBench benchmark [214], LLMs were shown to have perform poorly on several tasks, but follow up work [334] showed mere chain-of-thought prompting [301, 10] leads to huge boosts on all tasks. This indicates the evaluated models were in fact "competent", but inappropriate prompting led to undermining how "performant" they are.

4.3 Concept Space: A Framework for Analyzing Concept Learning Dynamics

We first formally introduce our framework of *concept spaces*. Motivated by a rich body of literature on disentangled representation learning [207, 202, 206, 305, 306, 290, 307, 197, 308], this framework allows us to systematically analyze how different concepts underlying the data generating process and corresponding capabilities to manipulate them are learned during training. We highlight that since our primary empirical focus in this paper will be an abstraction of text-to-image generative models, parts of the framework will be specifically instantiated with text-to-image generation tasks in mind. To this end, we note our model class of interest is a generative model F that is trained using conditioning information h to produce images y. For example, F may be instantiated using a diffusion model that uses embeddings h of textual descriptions of a scene to produce images y. We next define a *concept space*.

Definition 12. ***(Concept Space.)*** *Consider an invertible data-generating process $\mathcal{G} : \mathcal{Z} \to \mathcal{X}$ that samples vectors $z \sim P(\mathcal{Z})$ from a vector space $\mathcal{Z} \subset \mathbb{R}^d$ and maps them to the observation space $\mathcal{X} \in \mathbb{R}^n$. We assume the sampling prior is factorizable, i.e., $P(z \in \mathcal{Z}) = \Pi_{i=1}^{d} P(z_i)$, and individual dimensions of $\mathcal{Z}$ correspond to semantically meaningful concepts. Then, a concept space $\mathcal{S}$ is defined as the multidimensional space composed of all possible concept vectors z, i.e., $\mathcal{S} = \{z \mid z \sim P(\mathcal{Z})\}$*

As an example, consider a concept space defined using three concepts, say, $z_1 =$ `shape`, $z_2 =$ `size`, and $z_3 =$ `color` that maps to a dataset of images with objects of different combinations of shapes, sizes, and colors (see Fig. 4.2). The assumption that concepts are

independently distributed implies one can intervene on a given concept without affecting the other ones. For example, given a sample from the dataset above, the concept `color` in an image can be altered by changing the value of the relevant latent variable and mapping it to the image space via the data-generating process. Now, using a mixing function $\mathcal{M}$ that yields conditioning information $h = \mathcal{M}(z)$, we can train a conditional generative model and define a notion of capabilities relevant to our work as follows.

Definition 13. ***(Capability.)*** *A concept class $\mathcal{C}$ denotes the set of concept vectors $z_{\mathcal{C}}$ such that a subset of dimensions of these vectors are fixed to predefined values. Classes $\mathcal{C}$ and $\mathcal{C}'$ are said to differ in the k^{th} concept if $\forall z \in z_{\mathcal{C}}$, there exists $z' \in z_{\mathcal{C}'}$ with $z[k] \neq z'[k]$ and $z[i] = z'[i]$ for $i \neq k$. We say a model possesses the "capability to alter the k^{th} concept" if for any class $\mathcal{C}$ whose samples were seen during training, the model can produce samples from class $\mathcal{C}'$ that differs from $\mathcal{C}$ in the k^{th} concept.*

Intuitively, the definition above comments on whether the model can flexibly manipulate concepts of classes seen during training to produce samples from classes that were not seen, i.e., classes that are entirely out-of-distribution. As an example, consider a concept space with `shape`, `color`, and `size` as concepts. If `shape` and `color` are fixed to `circle` and `blue` respectively, we get the class of `blue circles`; i.e., $\forall z \in z_{\texttt{blue circles}}$, the first and second dimension respectively take on values that correspond to the shape `circle` and color `blue`. Then, given a conditional diffusion model that was shown `blue circles` during training, we will say this model possesses the capability to alter the concept `color` if it can produce samples from concept classes with the same shape as `circles`, but different colors (e.g., `red` or `green circles`). Analyzing learning dynamics in the concept space will thus provide a direct lens into the model's capabilities as they are acquired.

We also note that the definition above is not dependent on the precise manner via which the model is prompted to elicit an output, i.e., it need not be the case that the conditioning information h that is used for training the model is used to evaluate the model capability. In fact, in our experiments, we will try alternative protocols such as intervening on the latent representations to show that substantially before the model can be prompted using h to generate samples from an OOD concept class, it can generate samples from said class via such latent interventions. To this end, we also define a measure that assesses how much learning signal the data provides towards disentanglement of a concept, and hence learning of a capability to manipulate it.

Definition 14. ***(Concept Signal.)*** *The concept signal σ_i for a concept z_i measures the sensitivity of the data-generating process to change in the value of a concept variable, i.e., $\sigma_i = \left|{}^{\partial \mathcal{G}(z)}/_{\partial z_i}\right|$.*

Intuitively, if the training objective is factorized at the granularity of concepts, concept signal indicates how much the model would benefit from learning about a concept. For example, consider a diffusion model trained using the MSE loss with conditioning $h = z$ to predict the noise added to an image $\mathcal{G}(z)$. σ_i is then merely the component of the loss representing how much change in conditioning h yields changes in concept z_i, as it is represented in the image. Concept signal will thus serve as a crucial knob in our experiments to analyze learning dynamics in concept space and the order in which concepts are learned.

4.3.1 Experimental and Evaluation Setup

Before proceeding further, we discuss our experimental setup that concretely instantiates the formalization above. Our data-generating process is motivated by prior work on disentanglement [202, 206, 305, 306, 307, 197, 308] and involves concept classes defined by three concepts, each with two values: `color` $= \{\text{red}, \text{blue}\}$, `size` $= \{\text{large}, \text{small}\}$, and `shape`$=\{\text{circle}, \text{triangle}\}$. In Sec. 4.4.1 and Sec. 4.4.4, we use two attributes: `color` (with red labeled as `0` and blue as `1`) and `size` (large labeled as `0` and small as `1`). We generate a total of 2048 images for each class, with objects randomly positioned within each image. We train models on classes `00` (`large red circles`), `01` (`large blue circles`), and `10` (`small red circles`), shown as blue nodes in Fig. 4.2, and test using class `11` (`small blue circles`), shown as pink nodes, to evaluate a model's ability to manipulate concepts and generalize OOD (see App. C.1.1 for further details). In Sec. 4.5, we will restrict to two attributes, `shape`$=\{\text{circle}, \text{triangle}\}$ and `color` $= \{\text{red}, \text{blue}\}$, and study the effect of noisy conditioning, i.e., what happens when concepts are correlated in the conditioning information h due to some non-linear mixing function. For all experiments, we use a variational diffusion model [335] to generate $3 \times 32 \times 32$ images conditioned on vectors h (see App. C.1.2 for further training details).

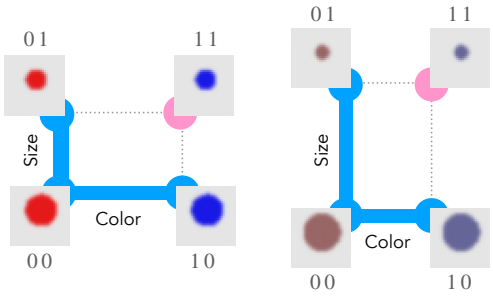


Figure 4.2: **Concept spaces with different concept values see different concept signal.** Consider a concept space comprised of concepts `size` and `color`. (Left) The `color` separation between the classes is stronger than the `size` separation, resulting in a stronger concept signal in the `color` dimension. (right) The `size` separation between the classes is stronger, thus resulting in a stronger signal for `size`.

Evaluation Metric. To assess whether a generated image matches the desired concept class without human intervention, we follow literature on disentangled representation learning [196,

202, 203, 204, 205, 206, 207, 290] and train classifier probes for individual concepts using the diffusion model's training data. The probe architecture involves a U-Net [336] followed by an average pooling layer and n MLP classification heads for the n concept variables. See App. C.1.2 for further details.

4.4 Learning Dynamics in Concept Space

4.4.1 Concept Signal Determines Learning Speed

We first demonstrate the utility of concept signal as a tool to gauge at what rate the model learns a concept and the capability to manipulate it. To this end, we develop controlled variants of our data-generating process by changing the level of concept signal of each concept and train diffusion models conditioned with the latent concept vector z on them. We primarily focus on concepts `color` and `size`, altering their concept signal by respectively adjusting the RGB contrast between `red` and `blue` and the size difference between `large` and `small` objects (see App. C.1.1 for details). We define the speed of learning each concept as inverse of the number of gradient steps required to reach 80% accuracy for class `11`, i.e., the OOD class that requires learning the capability to manipulate concepts as seen during training. Results comparing different concept signals are shown in Fig. 4.3. For both `color` and `size`, we observe that *concept signal dictates the speed at which individual concepts are learned*. We also find that when different concepts have varying strengths of concept signals, this leads to differences in the learning speed for each concept.

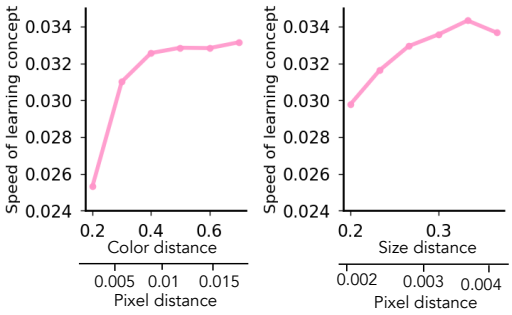


Figure 4.3: **Concept signal determines learning speed.** The speed of concept learning as an inverse of the time in gradient steps when the separation in color (left) and size (right) between different classes increases. Concept learning is faster when pixel differences among concept class and hence concepts are larger.

4.4.2 Concept Signal Governs Generalization Dynamics

We next examine the model's learning dynamics in concept space under various levels of concept signal for the concepts `color` and `size`. For completeness, we evaluate a model's ability to generalize both in-distribution (ID) and OOD, but only the latter is deemed learning of a capability to manipulate a concept. Results are shown in Fig. 4.4, with panel (a) showing

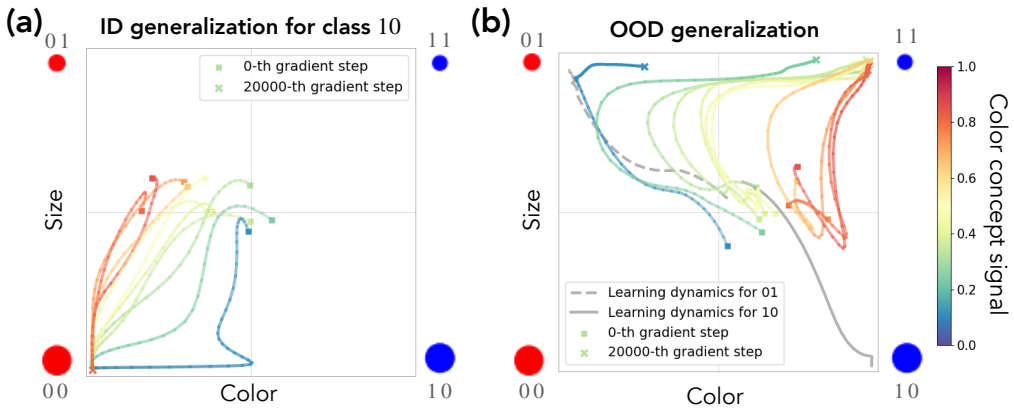


Figure 4.4: **Concept signal governs generalization dynamics.** (a) Learning dynamics in the concept space for in-distribution concept class `00` (bottom left). (b) Learning dynamics for out-of-distribution (OOD) concept class `11` (top right). We plot the accuracy for `color` on the x-axis and `size` on the y-axis. The [0,1) normalized `color` concept signal level is color coded. Two trajectories for `01` and `10` are shown to illustrate *concept memorization*.

dynamics for learning to generate samples from ID class `00` and panel (b) showing dynamics for OOD class `11`. Results on OOD generalization show a form of *concept memorization*, which we define as the phenomenon where *the model's generations on an OOD conditioning h are biased towards the training class that helps define the strongest concept signal*. For example, for the unseen conditioning `11`, the generations are more alike `01` when the concept signal is stronger for `size` (e.g., blue curve in Fig. 4.4(b)) and alike `10` when the signal is stronger for `color` (e.g., red curve). Interestingly, we observe that for settings with high imbalance in concept signals, e.g., the blue curve in Fig. 4.4 (b), the endpoint of *concept memorization* is very biased towards one training class, here `01`, delaying its out-of-distribution (OOD) generalization. For the learning trajectories of all classes, see Fig. C.3 in App. C.3.2.

Broadly, our results imply that an early stopped text-to-image model can witness *concept memorization* and hence simply associate an unseen conditioning to the nearest concept class when asked to generate OOD samples (see Kang et al. [337] for a similar result in LLMs). However, given sufficient time, the model will disentangle concepts underlying the data-generating process and learn to generate entirely novel, OOD samples. For futher evidence in this vein, we also confirm our results across more general scenarios, including with the real-world CelebA dataset (App. C.3.3) and using three concept variables: `color`, `background color`, and `size` (App. C.3.4).

4.4.3 Towards a Landscape Theory of Learning Dynamics

Fig. 4.4 indicates models undergo phases of understanding of concepts at different stages of training. In fact, an intriguing property of trajectories shown in Fig. 4.4 (b) is that there is a sudden turn in the learning dynamics from *concept memorization* to *OOD generalization* (e.g., see the top-left quadrant in Fig. 4.4 (b)). To investigate this further, we propose a minimal toy model that captures the geometry of model's learning trajectories shown in that figure. Specifically, we use the following dynamics equation, $d(t) = \tilde{z} + (\hat{z} - \tilde{z}) \cdot \frac{1}{1+e^{-(t-\hat{t})}}$, where $\hat{z}$ is the target point we want to get to and $\tilde{z}$ is the initial, "biased" target. For example, consider the case with `color` and `size` concepts in Fig. 4.4 (b). The model's generated samples are more alike class `01` and biased towards $(\tilde{z}_1, \tilde{z}_2) = (0, 1)$ when the size concept signal σ_2 is stronger than σ_1; and to `10`, $(\tilde{z}_1, \tilde{z}_2) = (1, 0)$ when the color concept signal σ_1 is stronger than σ_2. We define $(\hat{z}_1, \hat{z}_2)$ as the target values or directions we want the learning to head towards (e.g., $(1, 1)$ for OOD generalization). Based on this framework, we can derive the following energy function.

$$\frac{d\mathbf{z}}{dt} = -\nabla_{\mathbf{z}} L, \;\; L(z_1, z_2) = \begin{cases} \frac{1}{2a}\left(d(t-\hat{t}_1) - z_1\right)^2 + \frac{a}{2}(1-z_2)^2 & \text{if } \sigma_1 > \sigma_2, \\ \frac{1}{2a}(1-z_1)^2 + \frac{a}{2}\left(d(t-\hat{t}_2) - z_2\right)^2 & \text{otherwise.} \end{cases} \tag{4.1}$$

Here, a is determined by the difference $|\sigma_1 - \sigma_2|$ and $\hat{t}_1$ and $\hat{t}_2$ denote the times when the model learns concepts z_1 and z_2, respectively. Fig. 4.5 illustrates the simulated trajectories for classes `00` and `11`, based on Eq. 4.1. Panels (a) and (b) correspond to classes `00` and `11`, respectively. We define the actual target points $(\hat{z}_1, \hat{z}_2)$ as $(0, 0)$ for class `00` and $(1, 1)$ for class `11`. For the initial targets $(\tilde{z}_1, \tilde{z}_2)$, we set both values to $(0, 0)$ for class `00`. For class `11`, the targets are set to $(1, 0)$ when $\sigma_1 > \sigma_2$ and to $(0, 1)$ when $\sigma_1 < \sigma_2$. *We find our toy model effectively captures the actual learning dynamics for both in-distribution (Fig. 4.3(b)) and out-of-distribution (OOD) concept classes (Fig. 4.3(c)).* Notably, our simulation accurately replicates the two types of curves: clockwise (blue trajectory in Fig. 4.3(b)) and counter-

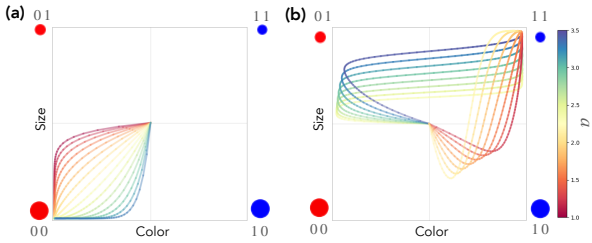


Figure 4.5: **A Phenomenological Model of Learning Dynamics in Concept Space.** Using Eq. 4.1, we simulate the learning trajectory for concept class `00` in panel (a) and the OOD class `11` in panel (b). Initially, target values are set at $(0, 1)$ or $(1, 0)$ based on the concept signal strengths for color or size, respectively. As the model progressively learns each concept, the target values gradually shift towards $(1, 1)$. This simple toy model accurately reproduces the observed curves in Fig. 4.3(c), which arise from *concept memorization*.

clockwise (red trajectory).

An important conclusion that follows from the results above is that the network's learning dynamics can be precisely decomposed into two stages, hence yielding the sudden turns seen in trajectories in Fig. 4.4. *We hypothesize there is a phase change underlying this decomposition and the model acquires the capability to alter concepts at this point of phase change.* We investigate this next.

4.4.4 Sudden Transitions in Concept Learning Dynamics

The concept space visualization of learning dynamics observed in Fig. 4.4 (b) and our toy models analysis in Fig. 4.5 indicate that there exists a phase in which the model departs from concept memorization and disentangles each of the concepts, but still produces incorrect images. We claim that at the point of departure, the model has in fact already disentangled concepts underlying the data-generating process and acquired the relevant capabilities to manipulate them, hence yielding a change of direction in its learning trajectory. However, naive input prompting is insufficient to elicit these capabilities and generate samples from OOD classes, giving the impression the model is not yet "competent". This then leads to the second phase in the learning dynamics, wherein an alignment between the input space and underlying concept representations is learned. We take the model corresponding to the second from left curve (the green curve) in Fig. 4.4 (b) to investigate this claim in detail. Specifically, given intermittent checkpoints along the model's learning trajectory, we use the following two protocols for prompting the model to produce images corresponding to the class `11` (`blue`, `small`). See App. C.2 for further details.

1. **Activation Space: Linear Latent Intervention.** Given conditioning vectors h, during inference we add or subtract components that correspond to specific concepts (e.g., $h_{\texttt{blue}}$).
2. **Input Space: Overprompting.** We simply enhance the color conditioning to values of higher magnitude, e.g. (0.4, 0.4, 0.6) to (0.3, 0.3, 0.7).

Fig. 4.6 shows the accuracy for five independent runs under: (a) naive input prompting, (b) linear latent interventions, and (c) overprompting. In Fig. 4.6 (a), we observe that some runs can produce samples from the target concept class (`blue`, `small`) with $\sim 100\%$ accuracy after around 8,000 gradient steps, while other runs fail to do so. However, in Fig. 4.6 (b, c), we find alternative protocols for prompting the model can *consistently* elicit the desired outputs much earlier than input prompting, e.g., at around as early as 6,000 gradient steps. *This indicates the model does possess the capability to alter concepts and generalize OOD!* Furthermore, we note that different protocols enable elicitation of the capability *at approximately the same number of gradient steps, irrespective of the seed, and that this is precisely the point of sudden*

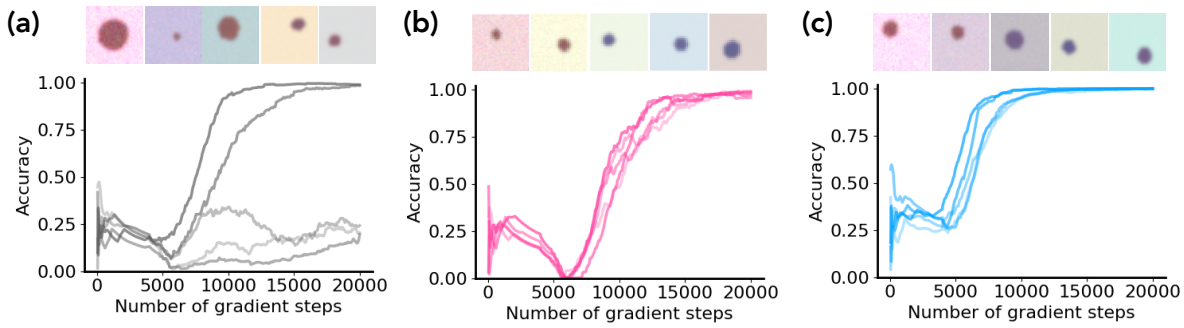


Figure 4.6: **Emergence of hidden capabilities.** We plot accuracy as a function of gradient steps for five different runs, using three different protocols for prompting the model to generate outputs for OOD concept classes. (a) The baseline, naive prompting protocol; (b) linear latent intervention, applied in the activation space; and (c) overprompting, akin to an intervention on the input space.

turn in the learning dynamics in Fig. 4.4! Interestingly, experiments with Classifier Free Guidance (CFG) [338] show that CFG only becomes effective after this transition (Fig. C.7).

We further explore the second phase of learning in Appendix C.3.6 by patching the embedding module used for processing the conditioning information from the final checkpoint to an intermediate one. Our results show that when the final checkpoint does enable use of naive input prompting for eliciting a capability, the embedding module can be patched to an intermediate checkpoint and we can retrieve the desired output at approximately the same time that alternative prompting protocols start to work well. This suggests the second phase of learning primarily involves aligning the input space to intermediate representations that enable eliciting the model capabilities. Overall, our results yield the following hypothesis.

Emergence of Hidden Capabilities

Hypothesis. ***(Emergence of Hidden Capabilities.)*** *We hypothesize Generative models possess hidden capabilities that are learned suddenly and consistently during training, but naive input prompting may not elicit these capabilities, hence hiding how "competent" the model actually is.*

4.5 Effects of Underspecification on Learning Dynamics

In the results above, we use conditioning information that perfectly specifies concepts underlying the data-generating process, i.e., $h = z$. In practice, however, instructions are underspecified and one can thus expect correlations between concepts in the condi-

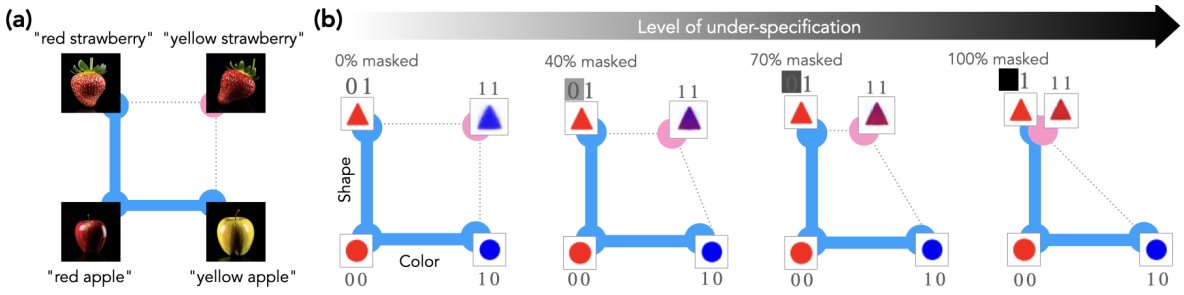


Figure 4.8: **Underspecification and Concept Learning.** (a) The state-of-the-art generative models [7] erroneously produces a `red strawberry` (top right corner) for the prompt "`yellow strawberry`". (b) Without underspecification in the training data, a model F accurately learns the concepts of shape and color, successfully generalizes to the unseen node `blue triangle` (leftmost). As masks are applied to the word `red` for the prompt `red triangle`, concept signal for `triangle` increasingly starts to correlate with the concept `red`. This causes the output images to change from blue to purple as the level of masking increases (panels left to right). Eventually, the color dimension for `triangle` collapses, biasing the model towards generating solely `red triangles` (rightmost).

tioning information extracted from those instructions [171, 339, 340, 341, 342]. For example, images of a `strawberry` are often correlated with the color `red` (see Fig. 4.8(a)). Correspondingly, unless a text-to-image model is shown explicit data stating "`red strawberry`" or images of non-red strawberries, the model's ability to disentangle the concept `color` from the concept `strawberry` may be impeded (see generations for "`yellow strawberry`" in Fig. 4.8). Motivated by this, we next investigate the effects of using underspecified conditioning information on a model's ability to learn concepts and capabilities to manipulate them.

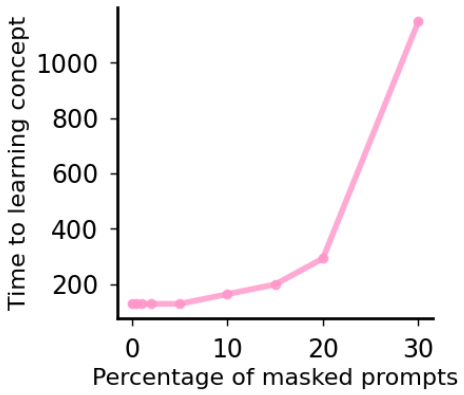


Figure 4.7: **Underspecification delays out-of-distribution (OOD) generalization.** The number of gradient steps required to reach accuracy above 0.8, as the percentage of masked prompts increases. A higher proportion of masked prompts slows down the speed of concept learning.

Experimental setup. The data generation and evaluation process follows the protocol described in Sec. 4.3.1. We select `color` (red and blue) and `shape` (circle and triangle) as the concepts, drawing an analogy to the "`yellow strawberry`" example. To simulate underspecification, we randomly select training samples that have a specific combination of shape and color (e.g., "`red triangle`"). We then mask the token representing the color ("`red`") and train the model on three concept classes {`00`, `01`, `10`}, represented by blue nodes in Fig. 4.8, with some

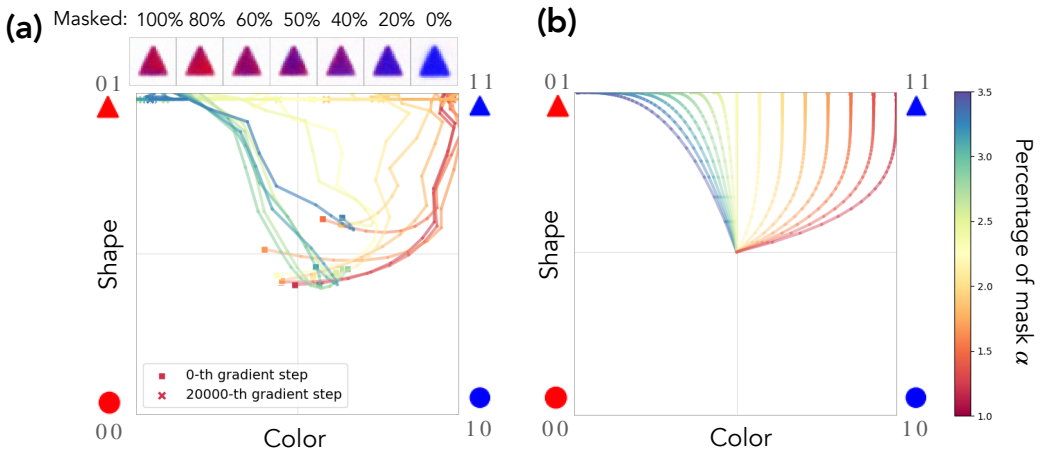


Figure 4.9: **Underspecification hinders out-of-distribution (OOD) generalization.** (a) The learning dynamics with varying levels of prompt masking, from 0% to 100%, and the generated images. At 0% masking (top right image), the model correctly produces an image of `blue triangle` from the prompt "`blue triangle`." As the masking increases (from right to left), the images gradually shift towards the incorrect color, `red`. (b) The simulation of the learning dynamics under underspecification in concept space based on Eq. 4.2. Our toy model replicates a trained network's learning dynamics.

prompts masked. We test using class `11` (pink node) with no prompts masked, to see if the model has learned disentangled concepts and can generalize OOD.

Underspecification Delays and Hinders OOD generalization. Fig. 4.7 shows how underspecification (masked prompts) affects the speed of concept learning. We see that as the percentage of masked prompts increases, the speed of learning a concept decreases, suggesting that underspecification leads to slower learning of concepts. Further, Fig. 4.9 (a) shows models' learning dynamics in concept space at varying levels of underspecification. With 0% masking, the model accurately produces an image of `blue triangle` (see Fig. 4.8(b)). However, as the percentage of masking increases, the color of generated images shifts from `blue` to `purple` (middle), and finally `red`. This demonstrates that when prompts are masked, the model's understanding of shape `triangle` becomes intertwined with color `red`; even when `blue` is specified in the prompt, the dynamics are biased towards `red`.

Toy Model of Learning Dynamics with Underspecification. When prompts are masked (i.e., underspecification occurs), target values for the concept variables are shifted: e.g., in our setup, with no mask applied, the target directions for the class `11` is $(1, 1)$. When the word "`red`" in "`red triangle`" is fully masked, the target shifts to $(0, 1)$. Assuming this shift is linear with respect to the percentage of masked prompts α, we can derive the

following energy function.

$$\frac{d\mathbf{z}}{dt} = -\nabla_{\mathbf{z}} L,\ L(z_1, z_2) = \Big((1 - s\alpha) - z_1\Big)^2 + \Big(1 - z_2\Big)^2. \tag{4.2}$$

In the above, parameter s represents the impact of underspecification. Fig. 4.9 (b) shows the simulation of model behavior in the concept space according to Eq. 4.2 ($s = 0.01$). As the masking level increases, the target directions shift from $z_2 = 1$ in the top right corner to $z_2 = 0$ in the top left corner. Our simulated dynamics thus match well with the model's learning dynamics shown in Fig. 4.9 (a).

Influence of Underspecification on Emergence of Hidden Capabilities. Following Sec. 4.4.4, we also explore whether it is possible to elicit the desired outputs from a model trained with underspecified data. Fig. 4.10 shows the accuracy results over five runs (a) without using any prompting method, and (b) using over-prompting. In both scenarios, the percentage of masking is set at 50%. Results clearly demonstrate that with over-prompting, the model achieves 100% accuracy after approximately 1,000 gradient steps, whereas without over-prompting, it fails in three out of five runs even after 2,000 gradient steps. These findings confirm that capability can develop prior to observable behavior, even in cases of underspecification.

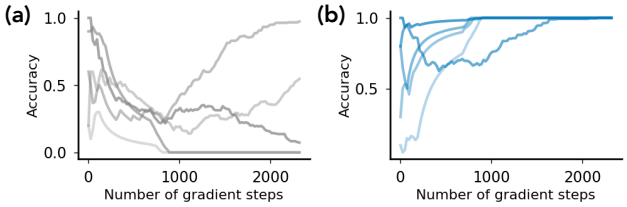


Figure 4.10: **Underspecification and Hidden Capabilities.** We use the over-prompting protocol from Sec. 4.4.4 to assess if the capabilities to enable OOD generalization are learned before naive input prompting. (a) Accuracy for OOD generalization across five different seed runs under naive prompting; (b) Accuracy under over-prompting, with a fixed masking percentage of 50%.

4.6 Additional Results on Natural Data

In this section, we aim to assess whether our findings generalize beyond the synthetic data domains defined before. To this end, we use the CelebA dataset [343], which contains fine-grained attributes corresponding to concepts like `Gender`, `With Hat`, and `Smiling`. We analyze two experimental settings.

- **Using concepts `Gender` and `With Hat`.** Results for the first setting are provided in Fig. C.4, wherein we find that indeed the concept learning dynamics discovered for the relatively simple concepts from prior sections generalizes. For example, we see there is first a phase of concept memorization, wherein the model learns to generate images of

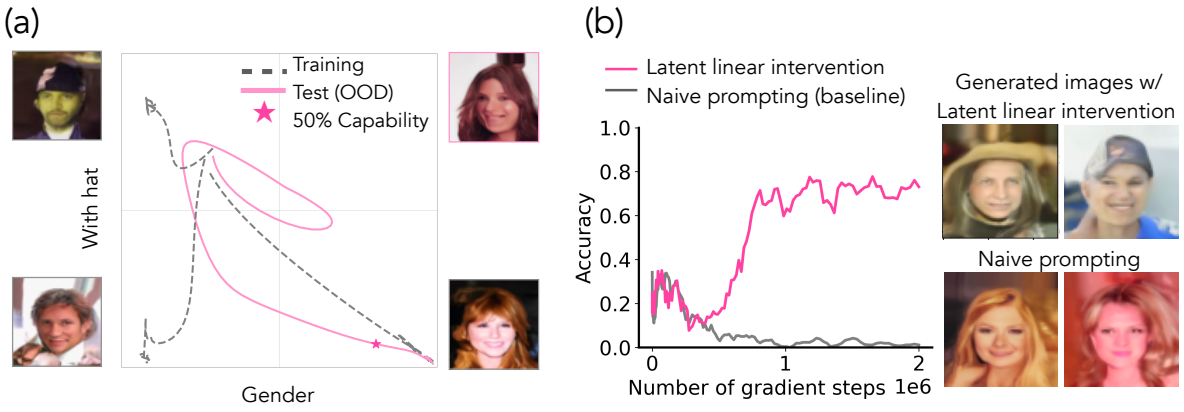


Figure 4.11: **Validating our findings on CelebA.** (a) Analyzing the learning dynamics for concepts `Gender` and `With Hat`, we find the training run gets stuck in concept memorization: converging near (likely) the closest class of `Females`, `With Hat`. (b) Performing latent interventions however clearly demonstrates that the model is indeed *capable* of generalizing out-of-distribution, but is not *performant* when evaluated via naive prompting.

`Female`, `Smiling`. However, as training proceeds, there is a sudden turn in the learning dynamics and the model finally learns to generalize out-of-distribution.

- **Using concepts `Gender` and `Smiling`.** In the second setting, under the given compute budget, we find the model never learns to generalize out-of-distribution (see Fig. 4.11). However, when we perform latent interventions on the model's representations, we are in fact able to force the model to produce images from the class `Female`, `With Hat`. These results show that the model is indeed *capable* of generalizing out-of-distribution, but is not *performant* when evaluated via naive prompting.

4.7 Discussion

Why study the concept space? One might ask the question of why the concept space could be useful beyond synthetic datasets. In Fig. C.2, we show the loss, accuracy, and training trajectory in concept space of a model from Fig. 4.6. The time at which the model acquires the capability to manipulate `size` and `color` independently is not evident from either the loss or accuracy curves. However, in concept space, one can directly see the divergence of the trajectory from *concept memorization*. Benchmarking generative models is a challenging task, still often involving humans in the loop [344, 132]. Our concept space framework suggests that benchmarking out-of-distribution (OOD) generalization can potentially be reduced to monitoring the learning trajectory in concept space.

Concept Learning vs. Grokking. We make a distinction between our delayed elicitation of capabilities versus grokking [173]. The common aspect is that the measured performance on a test set is delayed from the train set, as seen in Fig. C.2. However, we deal with out-of-distribution test data, which is different from the setups like modular arithmetic or polynomial regression in which grokking is usually explored [173, 345, 346, 174, 347]. Second, work on hidden progress measures show that in fact a model is continuously building representations towards the mechanism to solve the specified task [174, 210]; however, our results find that even at the level of latent representations, there is a sudden emergence whereby the model learns the capability to manipulate concepts underlying the data-generating process and generalize OOD.

Is Concept Learning a Phase Transition? In Sec. 4.4.4, we have demonstrated that before the capability to manipulate a concept is learned, the desired outputs cannot be generated irrelevant of the protocol used for prompting the model. Moreover, we showed that different protocols yield desired outputs at precisely the same time, and this time is also independent of model initialization. We thus hypothesize that: *concept learning is a well-controlled phase transition in model capability, but the ability to elicit this capability via a predefined single input prompting can be arbitrarily delayed, e.g., by the level of the concept signal.*

CHAPTER 5

In-Context Learning Dynamics via Random Binary Sequences

5.1 Introduction

Large language models (LLMs) achieve high performance on a wide variety of tasks they were not explicitly trained for, demonstrating complex, emergent capabilities [9, 137, 101, 348, 10, 239, 27, 138, 301, 349, 350, 351]. In-context learning (ICL) describes how task-specific patterns of behavior are elicited in LLMs by different prompts (or *contexts*) [9, 10, 352, 353, 354, 355, 356, 357, 358, 359]. Although no weight updates occur in ICL, different input contexts can activate, or re-weight, different capabilities of an LLM, analogous to updating model parameters to learn representations in traditional gradient descent learning methods [360, 361, 362, 363, 8, 364]. However, two prompts that are seemingly equivalent to humans can evoke very different behaviors in LLMs [68, 69, 357]. ICL makes interpreting what representations in an LLM are being evoked by a given input more challenging than with task-specific language models, where the space of capabilities is more restricted and well-defined.

Our central motivation is to interpret latent *concepts* underlying complex behaviors in LLMs by analyzing in-context learning behavioral dynamics, without directly observing hidden unit activations or re-training models on varied datasets. Inspired by computational approaches to human cognition [365, 366, 367, 368], we model and interpret latent concepts evoked in LLMs by different contexts without observing or probing model internals. This approach, which we call *Cognitive Interpretability*, is a middle ground between shallow test-set evaluation benchmarks on one hand [214, 369, 356, 370, 371, 372, 373, 374, 375, 376] and mechanistic neuron- and circuit-level understanding of pre-trained model capabilities on the other [174, 377, 378, 379, 351, 380, 381, 317, 382, 85, 383, 210, 345, 384] (also see App. D.3). Unlike related work of feature interpretability in vision neural networks, we analyze ICL and

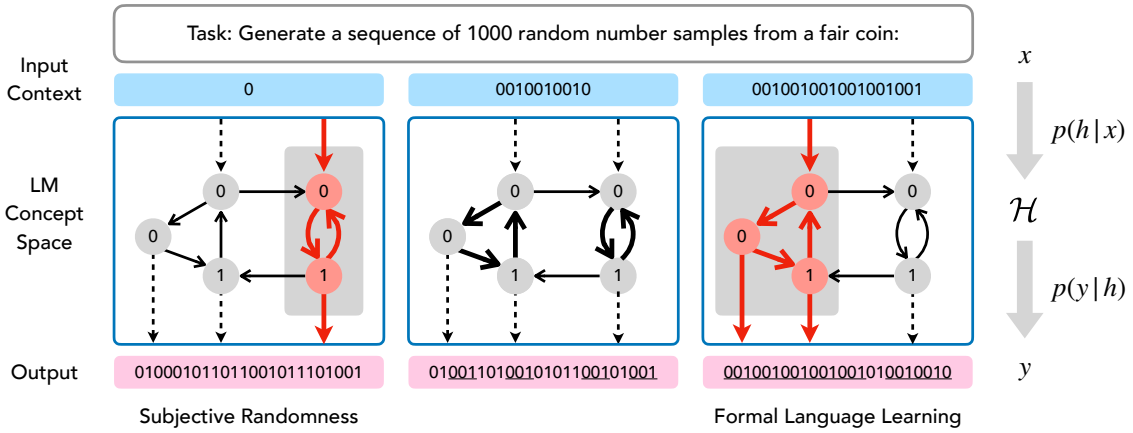


Figure 5.1: **Overview of our modeling framework.** Given a pre-trained Large Language Model, we systematically vary input context prompts, in this case $x = (001)^n$ with $n = \{2, 4\}$ (001001, 001001001001). LLM outputs y (Bottom) significantly as a function of x (Top), based on some unknown latent concept space embedded in the LLM. We model a subset of the algorithms embedded in the LLM's latent concept space $\mathcal{H}$ that are invoked by ICL (Middle), to predict the LLM outputs y. With very little context (Left), GPT-3.5+ generates subjectively random sequences, whereas with enough context matching a simple formal language ($x = 001001001001 = (001)^4$), it begins deterministically repeating patterns in x.

text generation dynamics over output sequences, and unlike mechanistic work studying ICL, we observe learning dynamics in black box input-output behavior of state-of-the-art LLMs.

Specifically, we focus on interpreting latent concepts in LLMs in the domain of random sequences of binary values (Fig. 5.1). Random binary sequences are a minimal domain that has been studied extensively in statistics, formal language theory, and algorithmic information theory [385, 386, 387]. We use this domain to systematically test few-shot learning as a function of context length $|x|$ across different binary token sequences x. Moreover, language generation trajectories over binary sequences can also be analyzed and visualized more easily than typical user-chatbot interaction trajectories [388, 389] since the token-by-token branching factor is only two. Random binary sequences have also been a target domain in cognitive science (specifically, *subjective randomness*), where researchers have studied the mechanisms and concepts that underlie how people generate random binary sequences or evaluate the randomness of a sequence [390, 391, 392].

Similar to computational cognitive theories of human concept learning [393, 394, 367, 395, 396, 397], we argue that ICL can be interpreted as underspecified program induction, where there is no single "correct" answer; instead, an LLM should appropriately re-weight latent algorithms. The domain of random sequences reflects this framing, in contrast to other behavioral evaluation methodologies, in that there is no correct answer to a random number generation or judgment task (Fig. 5.3). If the correct behavior is to match a target random

process, then the best way to respond to the prompt *Generate N flips from a fair coin* is a uniform distribution over the tokens `Heads` and `Tails`, instead of a specific ground truth token sequence. If instead the correct behavior is to match how humans generate random sequences, the LLM must represent a more complex algorithm that matches the algorithm used by people. Overall, we make the following contributions in this work.

- **Evaluating ICL as Bayesian hypothesis selection in a practical setting (Sec. 5.2).** We find evidence of in-context learning operating as Bayesian hypothesis selection in widely used LLMs, in contrast to prior work that trains smaller toy neural networks from scratch.
- **Subjective Randomness as a domain for studying ICL dynamics (Sec. 5.3).** We introduce a new domain for studying ICL—*subjective randomness*—which is rich enough to be theoretically interesting, but simple enough to enable analysis of complex behavioral patterns.
- **Sharp transitions in abilities to in-context learning dynamics (Fig. 5.4, 5.5).** We systematically analyze in-context learning dynamics in LLMs without observing or updating model weights, demonstrating sharp changes in model behaviors. This is a minimal, interpretable example of how the largest and most heavily fine-tuned LLMs can suddenly shift from one pattern of behavior to another during text generation, and supports theories of ICL as hypothesis selection.

5.2 In-Context Learning as Hypothesis Selection

Bayesian inference is a key methodological tool of cognitive modeling, and recent work has framed in-context learning as Bayesian inference over models [358, 398, 399]. The output tokens y an LLM produces given input tokens x can be modeled as a posterior predictive distribution $p(y|x)$ that marginalizes over latent concepts c in the LLM: $p(y|x) = \int_{c \in \mathcal{C}} p(y|c)\ p(c|x)$. In other words, a context x will activate latent concepts c in an LLM according to their posterior probability $p(c|x)$, which the model marginalizes over to produce the next token y by sampling from the posterior predictive distribution. This inference process takes place in network activation dynamics, without changing model weights.

In our experiments, we assume a hypothesis space $\mathcal{H}$ that approximates the latent space of LLM concepts used when predicting the next token, i.e.,

$$p(y|x) = \sum_{h \in \mathcal{H}} p(y|h)\ p(h|x).$$

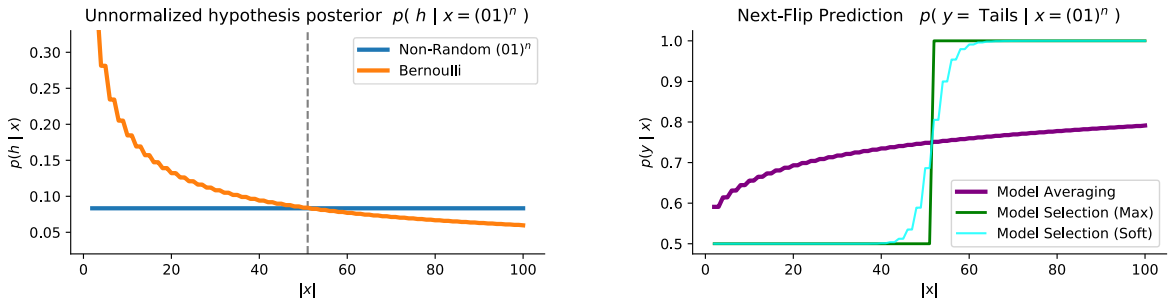


Figure 5.2: **Sharp changes in the predictive distribution $p(y|x)$ suggest hypothesis selection** (Left) Here we show an illustrative example of a simple Bayesian model with two hypotheses in $\mathcal{H}$: a random Bernoulli hypothesis, and a deterministic Non-Random hypothesis for the concept $(\texttt{01})^n$. When hypothesis posteriors $p(h|x)$ cross as a function of additional data, i.e., larger $|x|$, the predictive distribution $p(y|x)$ for hypothesis selection suddenly transitions from one pattern of behavior to another (Right). With hypothesis averaging, instead there is a steady, gradual change in $p(y|x)$ (see App. D.4 for further explanation). We find analogous S-shaped curves in ICL with LLM predictive distributions $p(y|x)$ in GPT-3.5+, suggesting hypothesis selection rather than hypothesis averaging (Fig. 5.4, 5.5). These phase changes are not predicted by theories of ICL as linear regression [8] or few-shot learning [9, 10], where loss steadily decreases with more context examples in x.

We assume that an LLM operates over next-token prediction only, and so each generated token y_{t-1} will be appended to the context when the following token is sampled, i.e., $x_t = (x_{0,\ldots,t-1}, y_{t-1})$. With this framework, we can understand ICL dynamics through the lens of hypothesis search. An LLM might not compute the sum in the above equation by taking a weighted average of latent capabilities. An LLM might perform Bayesian hypothesis selection, instead of hypothesis averaging, where for any given x, only a single latent concept h approximates the posterior predictive distribution: $h^* = \arg\max_h p(h|x), \quad p(y|x) = p(y|h^*)$. We provide a visual demonstration of how Bayesian hypothesis selection leads to sharp changes in the predictive distribution $p(y|x)$ with a toy model in Fig. 5.2.

Cognitive scientists have used Bayesian predictive and posterior distributions to model learning dynamics in human adults and children, as well as in non-human animals [365, 395, 397]. A discrete hypothesis space in a probabilistic model can give clear and meaningful explanations to learning patterns analogous to mode collapse and phase changes in deep learning. When behavior suddenly shifts from one pattern, or mode, of behavior to another, this can be understood as one hypothesis coming to dominate the posterior $p(h|x)$ as x grows in scale. This theory can explain conceptual shifts observed when children learn intuitive theories of domains such as intuitive physics [396] and learning to count [367]. Such cognitive analysis of behavioral shifts as generated from a shift in posterior probability between one hypothesis to another parallels recent work on learning dynamics in training

and prompting LLMs. Sharp behavior changes have been observed when training neural networks, corresponding to the formation of identifiable mechanisms such as modular addition circuits or induction heads [174, 228, 384, 400], and can be interpreted as Bayesian hypothesis selection [401]. Also see App. D.1, D.3.

To our knowledge, we provide the first demonstration of such sharp behavior changes in ICL, particularly with state-of-the art blackbox LLMs. The minimal model used by Xie et al. [358], as well as other few-shot learning work [9], show linearly or logarithmically decreasing loss curves in ICL, rather than S-shaped curves with stable plateaus as we find. Michaud et al. [247] propose that neural network knowledge and skills may be quantized into discrete chunks, and so training might consist of many such phase changes which disappear into a smooth curve when test loss is averaged over many samples. We propose that In-Context Learning may also consist of such sudden phase (or behavior) changes, as well as other complex learning dynamics, which may be explained as search over a discrete latent concept space. Such phase changes may occur in typical few-shot learning domains [10, 9], where smooth, continuously decreasing loss curves might hide complex ICL dynamics corresponding to shifts in the latent algorithms driving an LLMs behavior.

5.3 Algorithmic and Subjective Randomness

Next, we describe the domain of randomness explored in this work, and provide relevant background from computational cognitive science. To build intuition, consider: which sequence of coin flips is more random, `TTHTHHTH` or `HTTTTTTT`? While the odds of a fair coin producing either sequence are the same (1 in 256), most people say that the first sequence "looks more random". In a bias termed *the Gambler's Fallacy*, people reliably perceive binary sequences with long streaks of one value as 'less random', and judge binary sequences with higher-than-chance alternation rates as 'more random' than truly random sequences [390]. In addition to not using simple objective measures when judging the randomness of sequences, people also have a hard time generating sequences that are truly random [402, 392, 403, 404, 405].

Cognitive scientists studying *Subjective Randomness* model how people perceive randomness, or generate data that is subjectively random but algorithmically non-random, and have related this problem to algorithmic information theoretic definitions of randomness. One way to study this is to ask people whether a given data sequence was more likely to be generated by a Random process or a Non-Random process (Fig. 5.3). `HTTTTTTT` might be described with a shorter program—*one `H` followed by seven `T`s* – than `HTTHTHHT`, which cannot be compressed as well—*`H` followed by two `T`s, then one `H`, then one `T`, then two `H`s, then one `T`*. Viewing this task as program induction, a natural prior $p(h)$ over program hypotheses assigns

higher weights to programs with shorter description lengths, or lower complexity. Similar to our description of hypothesis selection, or hypothesis search, compared to hypothesis averaging, the space of non-random programs can be simplified to a search for the single shortest program consistent with the data x. This search is equivalent to computing the Kolmogorov complexity of a sequence $K(x)$ [387] and has motivated the use of "simplicity priors" in a number of domains in computational cognitive science [393, 395, 367].

Following previous work [366, 392], here we define subjective randomness of a sequence as the ratio of likelihood of that sequence under a random versus non-random model, i.e., $\texttt{randomness}(x) = \log P(x|\texttt{random}) - \log P(x|\texttt{non-random})$. The likelihood given a truly random Bernoulli process will be $p(x|\texttt{random}) = 2^{-|x|}$, since each of $2^{|x|}$ binary sequences of length $|x|$ has equal probability. The non-random likelihood $p(x|\texttt{non-random}) = 2^{-K(x)}$ denotes the probability of the minimal description length program that generates x, equivalent to Bayesian hypothesis selection: $P(x|\texttt{non-random}) = \max_{h \in \mathcal{H}} p(x|h)\ p(h)$. In theory, the space $\mathcal{H}$ of non-random models encompasses every possible computation; in this work, we study a small subset of this space that includes formal languages and probabilistic models inspired by psychological models of human concept learning and subjective randomness [392, 406, 407]. For further background, see App. D.11. We specifically focus on Bernoulli processes, regular languages, Markov chains, and a simple memory-constrained probabilistic model as candidates for $\mathcal{H}$.

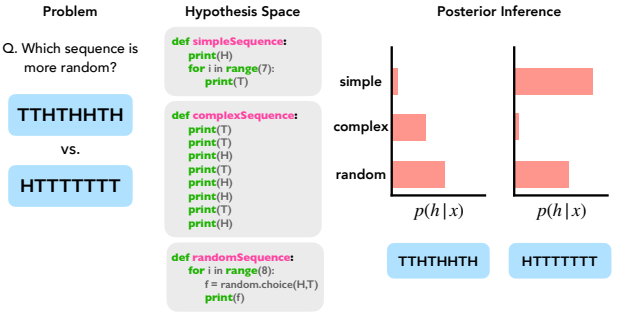


Figure 5.3: **Determining whether a sequence is random can be viewed as search for a simple program that could generate that sequence.** HTTTTTTT can be described with a short deterministic program `simpleSequence`, and has a higher $p(h)$ according to a simplicity prior, compared to TTHTHHTH and `complexSequence`. Both sequences can be generated by `randomSequence`, but with lower likelihood $p(x|h)$.

5.4 Experiments

Our experiments aim to evaluate how LLMs operate in the domain of Subjective Randomness. Specifically, our goal is to understand whether ICL and text generation dynamics suggest hypothesis selection, or hypothesis search, over a discrete concept space. We first evaluate whether there is an emergent algorithm for generating subjectively random sequences, and if so, whether it is similar to cognitive models of human subjective randomness (Fig. 5.6,

Q: Generate a sequence of 1000 random samples from a weighted coin, with {1 - p}% probability of Heads and {p}% probability of Tails.

A: [{sequence}, _____

Q: Is the following sequence of coin flips generated by a random process with no pattern, or are they generated by a non-random algorithm? [{sequence}]

A: The sequence was generated by a _____

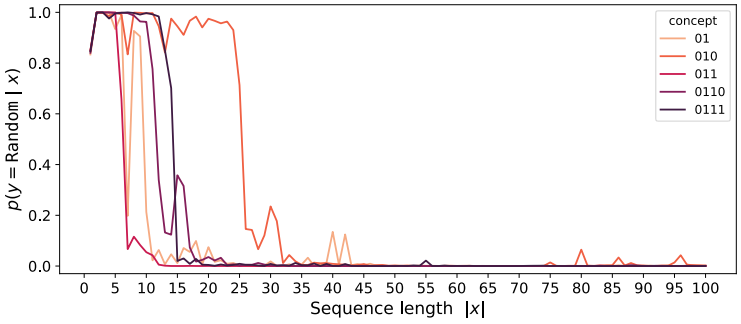


Figure 5.4: **Prompt templates and GPT-3.5 Randomness Judgments** (Left) Prompt templates used for the Randomness Generation (top) and Judgment (bottom) tasks. {`sequence`} is substituted with a list of context flips x, e.g. `Heads`, `Tails`, `Tails`, `Heads`. {`p`} is substituted with the probability of Tails, and {`1 − p`} with the probability of Heads. (Right) In Randomness Judgment tasks, we observe sharp transitions in the predictive distribution $p(y = \texttt{random}|x)$ for GPT-3.5, from high-confidence in x being generated by a random process, to high confidence in a non-random algorithm, as additional data x is provided from the same concept. See App. D.7 for additional concepts.

5.7). We use open-ended text generation experiments (Fig. 5.4, Left) and analyze binary sequences produced by LLMs, and compare LLM sequences to a Bernoulli distribution and our cognitively inspired [390, 407] Window Average model. Second, we evaluate whether ICL dynamics in searching over random and non-random hypotheses demonstrate ICL operating as hypothesis averaging, where learning is smooth and continuous, or as hypothesis selection, or hypothesis search, where sudden, dramatic changes can occur during learning.

Randomness Generation and Judgment Tasks In order to assess text generation dynamics and in-context concept learning, we evaluate LLMs on random sequence **Generation** tasks, analyzing responses according to simple interpretable models of *Subjective Randomness* and *Formal Language Learning*. In these tasks, the model generates a sequence y of binary values, or *flips*, comma-separated sequences of `Heads` or `Tails` tokens. We also analyze a smaller set of randomness **Judgment** tasks, with the prompt: *Is the following sequence of coin flips generated by a random process with no pattern, or are they generated by a non-random algorithm?* In both cases, y is a distribution over tokens with two possible values. Outputs y consist of a single `Random` or `Non` token, indicating whether the sequence was generated by a random process with no correlation, or some non-random algorithm. We analyze dynamics in LLM-generated sequences y simulating a weighted coin with specified $p(\texttt{Tails})$, with $|x| \approx 0$. An initial flip '`Heads`' is used so the model's flips will be formatted consistently. Prompt context in these experiments includes a specific probability, shown in Fig. 5.4, where the last _ marks where the model begins generating tokens y. We collect 200 output sequences

y for each LLM at each $P(Tails) \in [.05, .1, .2, .3, .4, .49, .5, .51, .60, .70, .80, .90, .95]$, cropping output tokens to $|y| = 50$ to limit cost and for simplicity. Temperature parameters are set to 1, and other parameters follow OpenAI API defaults; see App. D.2 for more details.

In the Judgment task, x includes the prompt question and the full sequence of flips. We systematically vary prompt context x by varying the number of flips, denoted $|x|$. We test few-shot learning in LLMs by evaluating output behavior on specific bit sequences, for example $(\texttt{01})^n$ with varying n (e.g. "`Heads`, `Tails`, `Heads`, `Tails`, ..."), as well as zero-shot language generation dynamics when x is empty and $|x| = 0$ (in practice, we initialize x with a single flip to help the LLM match the correct format). Unlike typical cases of few-shot learning, where high-variance samples ('shots') are drawn from a broad category such as "chain-of-thought solutions" [10, 9], all data in a sequence x corresponds to a single unobserved program.

Subjective Randomness Models We compare LLM-generated sequences to a ground truth "random" Bernoulli distribution with the same mean ($\mu = \overline{y}_{\text{LLM}}$), to a simple memory-constrained probabilistic model, and to Markov chains fit to model-generated data y. The Gambler's Fallacy is the human bias to avoid long runs of a single value (*'Heads'* or *'Tails'*) consecutively in a row [390, 406]. Hahn and Warren [407] theorize that the Gambler's Fallacy emerges as a consequence of human memory limitations, where 'seeming biases reflect the subjective experience of a finite data stream for an agent with a limited short-term memory capacity'. We formalize this as a simple *Window Average* model, which tends towards a specific probability p as a function of the last w flips: $p(y|x) = \max(0, \min(1, 2p - \overline{x}_{t-w,\ldots,t}))$.

Sub-Sequence Memorization and Complexity Metrics Bender et al. [408] raise the question of whether LLMs are 'stochastic parrots' that simply copy data from the training set. We test the extent to which LLMs generate subjectively random sequences by arranging memorized patterns of sub-sequences, rather than according to a more complex algorithm as our Window Average model represents. To measure memorization, we look at the distribution of unique sub-sequences in y. If an LLM is repeating common patterns across outputs, potentially memorized from the training data, this should be apparent in the distribution over length K sub-sequences. Since there are deep theoretical connections between complexity and randomness [409, 410], we also consider the complexity of GPT-produced sequences. Compression is a metric of information content, and thus of redundancy over irreducible complexity [411, 412], and neural language models have been shown to prefer generating low complexity sequences [360]. As approximations of sequence complexity, we evaluate the distribution of Gzip-compressed file sizes [413] and inter-sequence Levenshtein distances [414].

Formal Language Learning Metrics In our Formal Language Learning analysis, x is a subset of regular expression repetitions of short token sequences such as $x \in (\texttt{011})^n$,

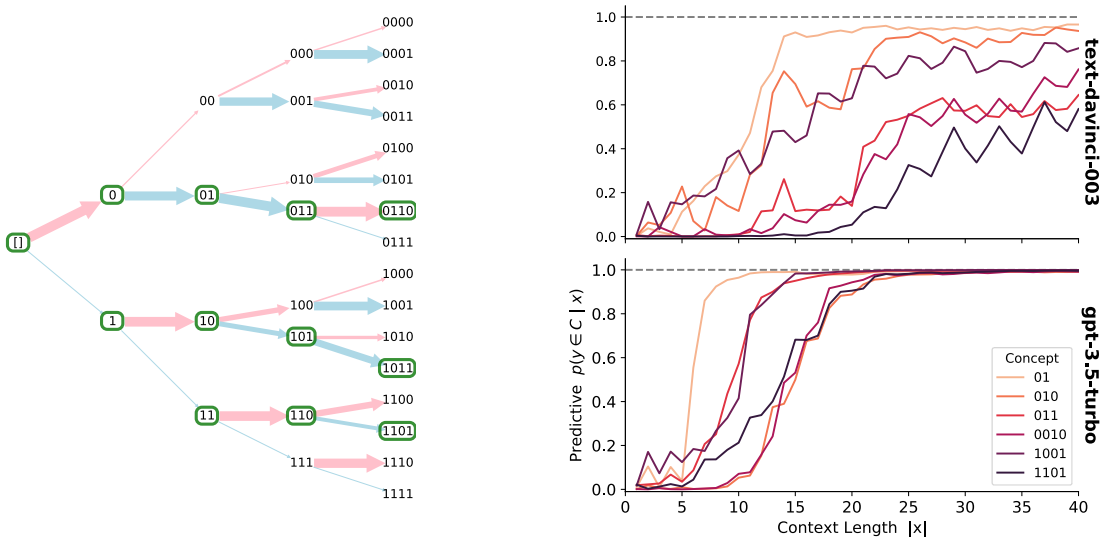


Figure 5.5: **With enough context, GPT-3.5 learns simple formal language concepts, transitioning from generating subjectively random sequences to only generating values matching the concept** (Left) GPT-3.5 next-token predictive distribution $p(y|x)$ visualized as a binary tree, where red arrows correspond to *Heads* (0), blue arrows to *Tails* (1), and nodes matching the target concept $C = (011)^n$ are dark green. $p(y|x)$ changes with varying $|x|$, here $|x| = 39$ and next-token predictions strongly follow the concept C. Also see Fig D.7, D.8 in Appendix. (Right) In-context learning dynamics for simple formal languages $x = (\texttt{HTH})^n$ $(\texttt{010})$ and $x = (\texttt{HTT})^n$ $(\texttt{011})$ as a function of context length $n = |x|$. Prediction accuracy computed as the total probability mass of $p(y|x)$ assigned to valid continuations of the formal language x, as a function of prediction depth $d = |y|$ and context length $|x|$. Curves shown are for depth $d = 6$, where only 3 out of 64 possible length-6 binary sequences y match concept C. Note: bottom model is `gpt-3.5-turbo-instruct-0914`. Also see App. D.6.

where longer sequences x correspond to larger n. This enables us to systematically investigate in-context learning of formal languages, as $|x|$ corresponds to the amount of data for inducing the correct program (e.g. $(\texttt{011})^n$) from Bernoulli processes, as well as the space of all possible algorithms. A Markov chain of sufficiently high order k can implicitly represent a regular language in its weights (see Fig. 5.6, Right). However, the number of parameters required is exponential in Markov order k, and weights are difficult to interpret. For example, a 3-order Markov chain fit to data $x = (\texttt{011})^n$ will represent $(\texttt{011})^n$ corresponds to high weights in the conditional probability distribution for $p(\texttt{0}|\texttt{11})$, $p(\texttt{1}|\texttt{01})$, $p(\texttt{1}|\texttt{10})$, and low values for other conditional probabilities. To match the goal of the Generation task, we use a variation of the prompt in Fig. 5.4: *'Generate a sequence of 1000 samples that may be from a fair coin with no correlation, or from some non-random algorithm'*

In Randomness Judgment tasks, we assess formal concept learning dynamics by changes

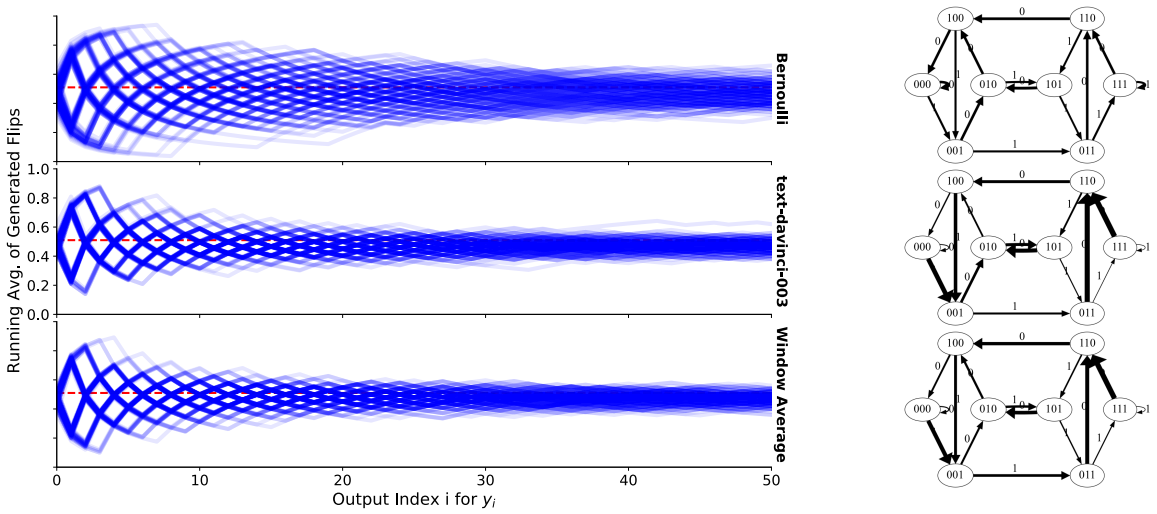


Figure 5.6: **GPT-3.5 generates subjectively random binary sequences that deviate from a Bernoulli process.** (Left) Running averages for flip sequences generated by each model, where 0 denotes '`Heads`' and 1 denotes '`Tails`'. Compared to a Bernoulli process (top), sequences generating using GPT (middle) and those of our Window Average model (bottom) stay closer to the mean, repeating the same patterns more often. (Right) Empirical conditional probabilities for a third-order Markov Chain fit to sequences y generated by GPT-3.5 `text-davinci-003`, a Bernoulli process centered at the mean of GPT sequence $\overline{y}$, and our Window Average model ($w = 5$). In the simulated Bernoulli process, edges are fairly uniform; edges for GPT-3.5 and the Window Average model are non-uniform, but similar to one another.

in the predictive distribution $p(y = \texttt{random} | x = C^{|x|})$, i.e., the probability of the final token being *'Random'* vs. *'Non'*, as a function of context length $|x|$. The method for measuring concept learning dynamics in Randomness Generation tasks requires computing $p(y|x)$ over possible binary flip sequences y, instead of y being a single token as with Judgment tasks. In binary sequence Generation tasks, we estimate the predictive probability $p(y_t \in C | x, y_{0...,t-1})$ assigned to a given regular language concept C by computing the total probability mass of $p(y|x)$ for all unique token sequences $y_{0...d}$ up to some depth d, such that the y exactly matches a permutation of the concept. If we assume that tokens in x and y have two possible values, 0 or 1, the probability table for $p(y_t | y_{0,...,t-1})$ is a weighted binary tree (see Fig. 5.5, Left) with depth d, where edges represent the next-token probability $p(y_t|y_{t-1})$, and paths represent the probability of a sequence of tokens. For example, with $C = (\texttt{011})^n$, there will be 3 paths $y_{0...d}$ corresponding to the permutations $(\texttt{011})^n, (\texttt{101})^n, (\texttt{110})^n$, out of 2^d possible sequences y, and so: $p(y \in (\texttt{011})^n \mid x) = p(y = \text{'0, 1, 1, 0, 1, 1, ...'} \mid x) + p(y = \text{'1, 1, 0, 1, 1, 0, ...'} \mid x) + p(y = \text{'1, 0, 1, 1, 0, 1, ...'} \mid x)$. We use the notation $p(y \in C \mid x)$ in Fig. 5.5 (Right) to refer to this predictive probability of y matching a concept C. In both

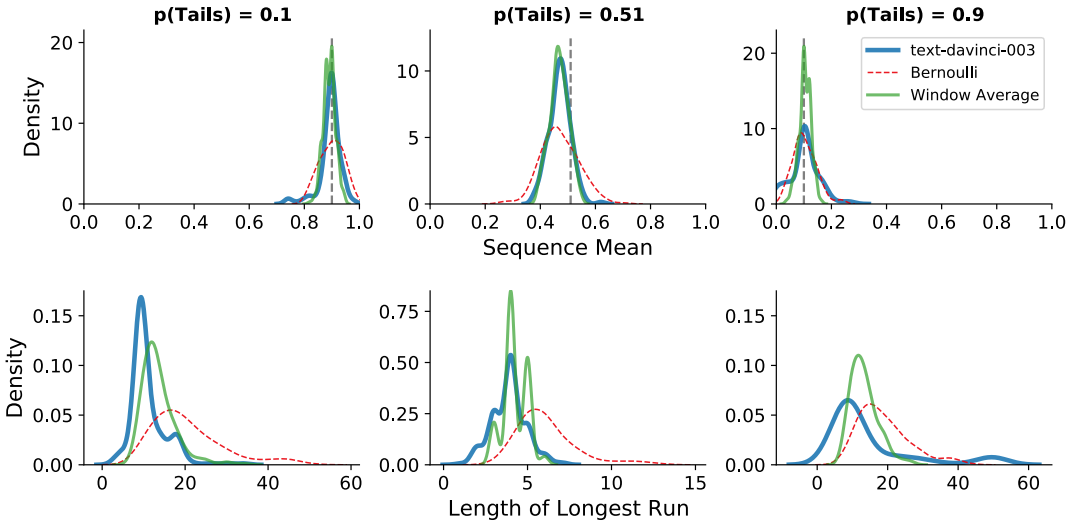


Figure 5.7: **GPT-3.5 shows a Gambler's fallacy bias of avoiding long runs.** (Top) Distribution of mean values of flip sequences ($\mu = \frac{1}{T}\sum_t y_t$) generated by GPT-3.5 (`text-davinci-003`) with the specified p(`Tails`), compared with a Bernoulli process and our Window Average model with the same mean as the GPT-3.5 flips. Flips generated by GPT approximately follow the expected mean p(`Tails`), but have lower variance than a Bernoulli distribution. (Bottom) Length of the longest run for each sequence, where a run is a sub-sequence of the same value repeating. In this case, we see a clear bias in GPT-3.5 to avoid long runs, with a similar pattern across all values of p(`Tails`) despite the x-axis changing in scale.

Judgment and Generation tasks for formal concept learning, we use next-token prediction probabilities $p(y_t|y_{0,\ldots,t-1}, x)$ directly from the OpenAI API, for cost efficiency and since this is equivalent to repeated token samples.

5.5 Results

5.5.1 Subjectively Random Sequence Generation

In the GPT3.5 and higher ('InstructGPT') models—`text-davinci-003`, ChatGPT (`gpt-3.5-turbo`, `gpt-3.5-turbo-instruct`) and GPT-4—we find an emergent behavior of generating seemingly random binary sequences (Fig. 5.6). *This behavior is controllable*, where different p(`Tails`) values leads to different means of generated sequences $\overline{y}$. However, the distribution of sequence means, as well as the distribution of the length of the longest runs for each sequence, deviate significantly from a Bernoulli distribution centered at $\overline{y}$, analogous to the Gambler's Fallacy bias in humans (see Fig. 5.7). *Our Window Average model with a window size of* $w = 5$ *partly explains both biases*, matching GPT-generated sequences more

closely than a Bernoulli distribution. This result demonstrates that ICL enables selection of *latent concepts in an LLM that operate as algorithms acting on data*. Our next experiment, on sub-sequence memorization, aims to further characterize this algorithm and investigate the alternative hypothesis that this behavior is generated by memorizing specific instances from the training distribution.

For brevity, in the following sections we primarily analyze the behavior of `text-davinci-003`, referring to it simply as 'GPT-3.5', and GPT models including more recent LLMs as 'GPT-3.5+'. Our cross-LLM analysis shows that `text-davinci-003` is controllable with $p(\texttt{Tails})$, with a bias towards $\overline{y} = .50$ and higher variance in sequence means (though lower variance than a Bernoulli process). Both versions of ChatGPT we analyze (versions `gpt-3.5-turbo-0301` and `0613`) demonstrate similar behavior for $p(\texttt{Tails}) < .5$, but behave erratically with higher $P(\texttt{Tails})$ and the majority of sequences y converge to repeating '`Tails`'. Both versions of GPT-4 show stable, controllable subjective randomness behavior, where sequence means $\overline{y}$ closely follow the specified $p(\texttt{Tails})$, but have even lower variances than sequences generated by `text-davinci-003`. Earlier models do not show subjective randomness behavior, with `text-davinci-002` and `text-davinci-001` being heavily biased and uncontrollable, and `text-curie-001` generates sequences with $\overline{y} = .50$ regardless of $p(\texttt{Tails})$. Also see App. D.8.

5.5.2 Sub-Sequence Memorization and Complexity

We find significant differences between the distributions of sub-sequences for GPT-3.5 - generated sequences and sequences sampled from a Bernoulli distribution (see Fig. 5.8). This difference is partly accounted for with a Window Average model with a window size $w = 5$, indicating an algorithmic explanation for model generated sequences; although GPT repeats certain longer sub-sequences, for example length-20 sub-sequences, that are far longer than 5. However, the majority of sub-sequences have very low frequency, and though it may be possible that these sub-sequence are memorized, we believe it is unlikely that these sub-sequences were in the training set, since we find thousands of unique length-k (with varying k) sub-sequences generated at various values of $P(\texttt{Tails})$. This indicates that GPT-3.5 combines dynamic, subjectively random sequence generation with distribution-matched memorization. Nonetheless, we do note that further experiments are needed to further ground this claim.

We also highlight that with a specification of $P(\texttt{Tails}) = 50\%$, but not 49%, 51% or other values, sequences y generated by GPT-3.5+ are dominated by repeating '`Heads`, `Tails`, `Heads`, `Tails`, ...'. This pattern is consistent across variations of the prompts listed in Fig. 5.4, including specifying 'fair' or 'unweighted' instead of a 'weighted coin', and produces

a visible kink in many cross-$p(\texttt{Tails})$ metrics (Fig. D.17, D.20, D.21). Due to this, in Fig. 5.7 we show results for $P(\texttt{Tails}) = 51\%$. Across three metrics of sequence complexity—number unique sub-sequences, Gzip file size, and inter-sequence Levenshtein distance (see Fig. D.21, D.20 in Appendix)—we find that *GPT-3.5+ models, with the exception of ChatGPT, generate low complexity sequences*, showing that structure is repeated across sequences and supporting contemporary results [360, 412]. For metrics mean Levenshtein distance and number of unique sub-sequences, ChatGPT generates higher complexity sequences than chance. We speculate that this phenomenon might be explained in future work by a simple cognitive model: if the LLM makes a sequence more subjectively random by avoiding repeating sub-sequences, like a human might, then it will produce sequences with higher-than-chance complexity overall.

5.5.3 Distinguishing Formal Languages from Randomness

GPT-3.5 sharply transitions between behavioral patterns, from generating subjectively random values to generating non-random sequences that perfectly match the formal language (Fig. 5.5). We observe a consistent pattern of formal language learning in `gpt-3.5-turbo-instruct-0914` generating random sequences where predictions $p(y|x)$ of depth $d \geq 4$ are initially random with small $|x|$, and have low $p(y \in C|x)$ where C is a given concept. This follows whether the prompt describes the process as samples from "a weighted coin" or "a non-random-algorithm". ICL curves for longer concepts are smoother for `text-davinci-003` and do not converge to $p(y|x) \approx 1$ as they do for `gpt-3.5-turbo-instruct-0914`. In this work we do not explore the reason for this distinction or study when this breakdown occurs; however, we do provide analysis of longer concepts in Randomness Judgment experiments, which are more cost-efficient (App. D.7).

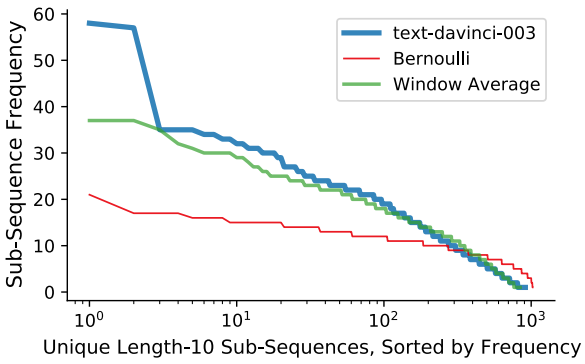


Figure 5.8: **GPT repeats certain sub-sequences more often than chance.** We find that GPT-3.5 repeats certain sub-sequences more often than is predicted by our Window Average model. While the Window Average model (green) generates fewer unique sub-sequences than a Bernoulli process (red), this does not account for the bias in GPT-3.5 (`text-davinci-003`, in blue) to repeat many of the same sub-sequences. This disparity increases with longer sub-sequences (see Fig. D.18 in Appendix).

We also find *sharp phase changes in GPT-3.5 behavioral patterns in Randomness Judgment*

tasks across 9 binary concepts (Fig. 5.4). These follow a stable pattern of being highly confident in that the sequence is Random (high $p(y = \texttt{random}|x)$ when x is low, up to some threshold of context at which point it rapidly transitions to being highly confident in the process being non-random. Transition points vary between concepts, but the pattern is similar across concepts (also see Fig. D.9, D.10 in Appendix). These transitions support our theory of ICL as hypothesis selection, or discrete hypothesis search, rather than hypothesis averaging, as in Fig. 5.2.

5.6 Discussion

We find strong patterns of random number Generation and Judgment in LLMs, including an emergent ability to generate low-variance, 'subjectively' random binary sequences, that can be partly explained with a memory-constrained moving window model. This behavior can also partly be explained as simple memorization and compression, but we observe and provide indicative evidence for algorithmic behavior that seems to go beyond mere parroting. We also find striking behavioral phase changes when GPT-3.5 learns to distinguish simple formal language concepts from randomness. This supports our framework of understanding ICL as Bayesian hypothesis selection and as underspecified program induction, and demonstrates how LLM behavioral patterns can change non-linearly with ICL as a function of context data.

Our work is an initial step in a direction which may prove to be highly impactful: if new capabilities can suddenly, rather than gradually, emerge after a single additional data point is presented in-context, this presents major challenges to AI safety and LLM interpretability. LLM capabilities may suddenly emerge or disappear with new prompts, or with more 'shots', and LLM behavior may suddenly and dramatically shift over the course of a single user interaction. These phase changes may be common in ICL, unobserved by research in few-shot learning that averages over many samples when considering in-context learning curves, instead of considering individual learning trajectories as we do.

Future work may characterize concept learning dynamics underlying other LLM capabilities, including applications in usability and safety (App. D.3). If ICL is analogous to hypothesis search in human cognition, a compelling research direction would be to further understand the defining features and limits of in-context learning dynamics using theories and tools from computational cognitive science.

Part 3: Limitations of Fine-tuning for Removing Model Vulnerabilities

Once an undesirable behavior or vulnerability is identified, we must fix it. The primary tool used for this purpose in contemporary literature is fine-tuning via different objectives on a downstream dataset that demonstrates the desired behavior. For example, consider an LLM making gender biased predictions in some task where gender is irrelevant. One may collect expert data corresponding to unbiased predictions for this task and fine-tune their model on this dataset [228]. If data corresponding to biased and undesirable predictions is also available (it can in fact be generated using the model), one can use a contrastive learning objective that promotes the probability of desirable generations while suppressing that of undesirable ones [415]. Alternatively, one may collect several outputs to a query and ask human raters to describe which outputs they prefer. A reward model can be trained on this preference dataset and then the language model can be fine-tuned to maximize the return received from the reward model [144, 416, 143]. The shared thread in all these protocols is the use of fine-tuning with a small learning rate and on a much smaller dataset than the one used for pretraining; the difference is the precise objective used for fine-tuning.

The success of (chat)GPT and related models [18, 17, 19] clearly demonstrates the protocols above can reduce a model's tendency to produce undesirable outputs. However, accidents are best described via safety in the worst-case, not average-case, scenarios. Thus, instead of evaluating whether the model avoids use of undesirable behavior on an average interaction that is similar in structure to the data seen during fine-tuning, the question that warrants being asked is whether the behavior has been entirely deleted from the model: when we systematically go out-of-distribution with respect to the fine-tuning data's distribution, does the model avoid engaging in undesirable behaviors? The goal of this part of the dissertation is to (i) make progress on the question above by providing a thorough formalization of the notion of going "systematically out-of-distribution" in the language of causal inference and (ii) provide a detailed empirical account of how, at a mechanistic level, the mechanisms underlying undesired behaviors continue to persist in the model after fine-tuning.

Contributions of Part 3

- **Formalizing a notion of mechanistic similarity and demonstrating fundamental limitations of fine-tuning in synthetic vision tasks**. We propose a theoretical framework to characterize whether two models that behave the same, are mechanistically the same, i.e., whether they rely on the same features present in an input to make their predictions. We connect our definition to fine-tuning via the lens of mode-connectivity—a phenomenon in neural networks that shows independently trained models can be interpolated along continuous paths between their parameters without decrease in task performance [73, 74]. We show when these paths are linear in nature, models are mechanistically dissimilar in relatively trivial manners. Since fine-tuning with small learning rates generally produces linearly mode-connected models, our work demonstrates the inability of fine-tuning to alter or delete undesirable mechanisms from a model.
- **Mechanistically analyzing the effects of fine-tuning on procedurally defined tasks**. While our work above argues fine-tuning can only trivially alter pretraining mechanisms, it is unclear how it elicits the substantially different behaviors we see after protocols like RLHF [143]. In this chapter, we put forth and show empirical verification for the hypothesis that a model learns all relevant mechanisms during pretraining (both desirable and undesirable ones), and fine-tuning produces localized changes—which we call "wrappers"—that minimally alter the pretraining mechanisms to disallow the elicitation of undesired behaviors. Due to their minimalistic nature, however, going even slightly out of distribution leads the wrappers to not be invoked. Further, fine-tuning on a downstream dataset can sample efficiently remove these wrappers (e.g., in 5–50 iterations!).

Author Roles

This part is based on a paper published at ICML 2023 [85], co-authored with Eric Bigelow, Robert Dick, David Krueger, and Hidenori Tanaka; and another that was published at ICLR 2024 [87], co-authored with Samyak Jain, Robert Kirk, Robert Dick, Hidenori Tanaka, Edward Grefenstette, Tim Rocktäschel, and David Krueger. In the first work, I led the problem definition, formalization, theory, design of experimental setup alongside Eric Bigelow, execution of the experiments, and paper writing. In the second work, I defined the broader problem based on my prior work, led its formalization, design of experimental setup, and paper writing. Experiments were conducted in collaboration with Samyak and Robert Kirk.

CHAPTER 6

Mechanistic Mode Connectivity

6.1 Introduction

Loss landscapes of modern deep neural networks (DNNs) have been shown to contain infinitely many global minimizers that are equally reachable via standard gradient-based optimization techniques [417, 418, 419, 420, 421, 422]. Recent work finds intriguing geometrical constraints relating these minimizers [423, 424, 425, 426, 427, 428], showing them to be **connected** via a single, continuous manifold that emerges as a result of overparameterization. The existence of such connected sets of solutions has been heavily corroborated in literature on **mode connectivity** [73, 74, 429, 430, 431], which, quite surprisingly, shows that the paths connecting global minimizers obtained via standard training pipelines are *relatively simple* (e.g., linear or quadratic). In parallel, several papers have demonstrated that different models trained on a task can perform radically differently at test time [432, 433, 434]. This behavior can be partially ascribed to models learning to utilize rather dissimilar attributes of an input for making their predictions [435, 436, 437, 438]. For example, in most vision datasets, backgrounds are correlated with object categories—a sampling bias [439, 440]. Consequently, a model can infer the correct label of an object by

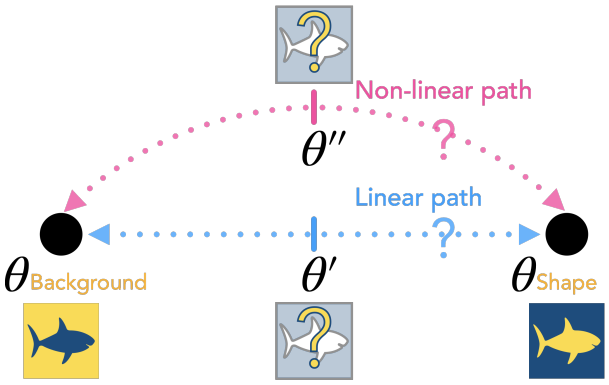


Figure 6.1: **Mechanistic Lens on Mode connectivity.** Consider two sets of parameters that minimize loss using background $\theta_{\text{Background}}$ and object shape θ_{Shape} as the input attributes for prediction, respectively. Are such *mechanistically dissimilar* minimizers connected via paths of low loss in the landscape? Does the dissimilarity of these mechanisms affect the simplicity of their connectivity paths? Can we exploit this connectivity to switch between minimizers that use our desired mechanisms?

learning **mechanisms** to identify either its background or its shape; however, only models that rely on shape are likely to generalize robustly [441, 442, 443]. Thus, despite models of both types being equally performant on a given dataset, the exact mechanisms they use for making their predictions disallows for us to consider them equally useful.

This work: We argue prior literature analyzing connectivity properties in DNN loss landscapes has ignored the influence of the exact mechanisms a model implements for performing a task (see Fig. 6.1). In fact, due to inherent tendencies in the training pipelines of modern DNNs towards learning simple functions [444, 445, 446, 11, 447], minimizers identified via training on the same dataset often exhibit similar biases [11, 448]. Such **similarity** in the models' prediction mechanisms may influence the identifiability of simple connectivity patterns in the loss landscape, such as the ones observed in prior work. Importantly, it has remained unclear if ***mechanistically dissimilar*** *models, e.g., ones that rely on background and ones that rely on shape, exhibit connectivity at all.* Beyond a better scientific understanding of DNN loss landscapes, knowledge of such geometric properties relating mechanistically dissimilar minimizers can possibly lead to practical insights for designing post-hoc, sample-efficient *fine-tuning* strategies that allow switching to minimizers that follow our desired predictions mechanisms. Motivated by questions above, we make the following contributions.

- **Defining a notion of *mechanistic similarity* (§6.3).** We characterize mechanistic similarity of two models via systematic interventions on the data-generating process, claiming similarity if the models are *invariant* to the same set of interventions. Our definition is motivated to account for the specific attributes of an input (e.g., shape vs. background) a model relies on for making predictions. When analyzed in the context of *spurious* attributes, i.e., attributes that statistically but not causally correlate with desired predictions (see Fig. 6.3), our definition leads to a characterization of DNN loss landscapes that is relevant to challenges of robustness [432, 449, 450].

- **Characterizing connectivity properties of mechanistically (dis)similar models (§6.5).** Our analysis shows that *if two models lack linear connectivity in the landscape (up to architectural symmetries), they must be mechanistically dissimilar*; that is, existence of loss barriers on the linear path between two models implies they have learned different invariances (see Fig. 6.4, 6.5). Our results especially hold implications for naïve fine-tuning of a pretrained network, which often yields models linearly connected with the original pretraining solution [451]. Specifically, if a model has learned to rely on spurious attributes during pretraining, our results imply mere fine-tuning on some "clean" dataset may not improve its robustness. We augment these first steps towards a mechanistic characterization of loss landscapes with extensive empirical verification

over a broad variety of settings, including different datasets, architectures, connectivity paths, and training strategies.

- **Exploiting lack of linear connectivity to efficiently alter a model's mechanisms (§6.6).** Based on our analysis, we propose a method, *Connectivity-Based Fine-Tuning (CBFT)*, that exploits lack of linear connectivity between mechanistically dissimilar models to induce models that differ in specific prediction mechanisms (§6.6). Extensive experiments on synthetic datasets show CBFT is more effective than recent methods [452, 15, 16] at reducing a model's tendency to rely on spurious attributes for making its predictions.

6.2 Preliminaries: Mode Connectivity

Intuitively, mode connectivity along a path implies moving along that path does not witness *barriers* in error or loss. We formalize this below, in line with prior work [429, 73, 74, 430, 453, 454]. Consider a neural network $f : \mathbb{R}^n \times \mathbb{R}^d \to [K]$ that takes n-dimensional inputs $x \in \mathcal{X} \subset \mathbb{R}^n$, has parameters $\theta \in \mathbb{R}^d$, and produces an output $f(x;\theta) \in [K]$, where $[K]$ denotes the set $\{1, 2, \ldots, K\}$. We say θ "induces the model" $f(.;\theta)$. A model's loss on a dataset $\mathcal{D} \in \mathcal{X} \times [K]$ for set of parameters θ is denoted using $\mathcal{L}(f(\mathcal{D};\theta))$; θ is called a minimizer of the loss on that dataset if $\mathcal{L}(f(\mathcal{D};\theta)) < \epsilon$, where ϵ is some small scalar. Note that we primarily focus on minimizers obtained using SGD. We denote a continuous path between two sets of parameters θ_1, θ_2 as $\gamma_{\theta_1 \to \theta_2}(t)$, where $\gamma_{\theta_1 \to \theta_2}(0) = \theta_1$ and $\gamma_{\theta_1 \to \theta_2}(1) = \theta_2$.

Definition 15. ***(Mode Connectivity.)*** *Minimizers θ_1, θ_2 corresponding to a dataset $\mathcal{D}$ are called mode connected along the path $\gamma_{\theta_1 \to \theta_2}(t)$ if moving along the path never yields barriers. Formally, $\forall\, t \in [0, 1]$, $\mathcal{L}(f(\mathcal{D}, \gamma_{\theta_1 \to \theta_2}(t))) \leq t \cdot \mathcal{L}(f(\mathcal{D};\theta_1)) + (1-t) \cdot \mathcal{L}(f(\mathcal{D};\theta_2))$.*

As mentioned in §6.1, prior work shows mode connectivity is exhibited in modern DNNs' loss landscapes along rather simple paths. We focus on the following two families:

(i) **Linear:** $\gamma_{\theta_1 \to \theta_2}(t) = (1-t)\theta_1 + t\theta_2$ and

(ii) **Quadratic:** $\gamma_{\theta_1 \to \theta_2}(t) = (1-t)^2\theta_1 + 2t(1-t)\theta_{12} + t^2\theta_2$.

In the above, θ_{12} denotes a set of parameters that is explicitly optimized to identify a quadratic path connecting θ_1 and θ_2; notably, then, quadratic paths are a function of the data used for identifying them (see App. E.3.1 for further discussion). The goal for analyzing quadratic paths, beyond the fact that they were analyzed in prior work, is that if there is a hill disconnecting two valleys of low loss, then a quadratic path would allow one to "go around the hill" without increasing loss.

Entezari et al. [430] recently hypothesized that accounting for permutation symmetry* of DNN architectures [455] in fact leads to observance of linear connectivity between any two linearly *disconnected* minimizers obtained using SGD; prior work [430, 456, 431] extensively probed and corroborated this claim empirically. To demonstrate the robustness of our results, we also assess whether accounting for permutation symmetry leads to linear connectivity between mechanistically dissimilar models. Specifically, we follow the "activation matching" algorithm used by Ainsworth et al. [431] and call these paths *Linear (permuted)*.

6.3 Defining Mechanistic Similarity

To analyze whether models that rely on different mechanisms for making their predictions exhibit mode connectivity, we must first define a notion of mechanistic similarity between two models. For this purpose, we argue two models are mechanistically similar if they utilize the same attributes of an input to make their predictions (e.g., shape or background). This can be assessed by transforming an input to alter some attribute of interest and thereafter checking if the two models under consideration make the same predictions on these transformed inputs. By using transformations that embody task-relevant vulnerabilities, this definition can be made practically well-motivated. For example, by choosing background randomization as an input transformation, we can assess whether two models rely on the (often) spurious attribute of background to make their predictions (Fig. 6.1).

To formalize the intuition above, we describe a generative model of data that can represent input transformations in a general manner. Specifically, we follow prior literature on disentanglement [207, 457, 458, 308, 197] and Independent Component Analysis (ICA) [306, 305, 459, 460], and assume that there is a latent space $\mathcal{Z} \subset \mathbb{R}^m$, with z sampled from a factorizable distribution, $P(z) = \prod_i P(z_i)$, such that each z uniquely maps to samples in the dataset via a generative process $\mathcal{G} : \mathcal{Z} \to \mathcal{X} \times [K]$, i.e., $(x, y) := \mathcal{G}(z)$. If $\mathcal{G}_X$, $\mathcal{G}_Y$ define the components of $\mathcal{G}$ producing x and y, the uniqueness of z amounts to assuming invertibility of $\mathcal{G}_X(.)$, denoted as $\mathcal{G}_X^{-1} : \mathcal{X} \to \mathcal{Z}$. Using the notations above, we can model input transformations as counterfactuals generated via systematic interventions on the data-generating process, similar to Besserve et al. [296, 461].

Definition 16. ***(Unit Interventions and Counterfactuals.)*** *A unit intervention $\mathcal{A}_i^{\alpha_i} : \mathcal{Z}_i \times \mathcal{Z}_i \to \mathcal{Z}_i$ on the data-generating process $\mathcal{G}$ is the alteration of the i^{th} dimension of a latent vector z by setting it to a predefined scalar $\alpha_i \in \mathcal{Z}_i$. Meanwhile, a counterfactual*

*Note that DNNs exhibit several architectural symmetries and a more general statement would account for all such symmetries, as done by [454]. However, symmetries beyond permutations are unlikely to play a critical role in analysis of mode connectivity of SGD based minimizers (see App. E.4 for details).

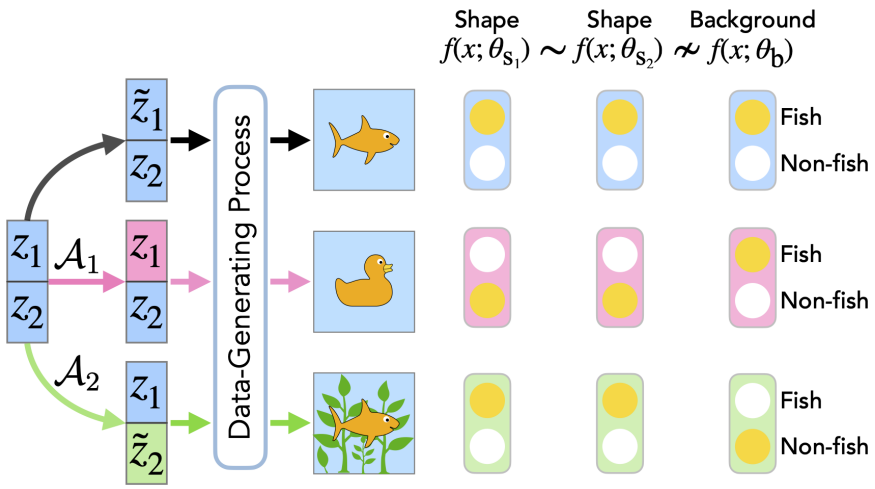


Figure 6.2: **Mechanistic Similarity:** We define mechanistic similarity of two models based on how they respond to unit interventions on the data-generating process, i.e., interventions on specific dimensions of the latent vector z; e.g., $\mathcal{A}_1$ (shape) and $\mathcal{A}_2$ (background) in the figure. Here, yellow circles represent the prediction of a given model (column) on a counterfactual image (row). Models whose predictions are invariant to the same set of interventions (denoted $\theta_1 \sim \theta_2$) are termed mechanistically similar.

process $\mathcal{E} : \mathcal{X} \times \mathcal{Z}_m \times \cdots \times \mathcal{Z}_1 \to \mathcal{X}$ *transforms a sample* x *by changing its corresponding latent vector* $z = \mathcal{G}_X^{-1}(x)$ *via a set of unit interventions* $\widehat{\mathcal{A}} := \{\mathcal{A}_i^{\alpha_i}\}_{i=1}^m$ *and mapping it back to the input space, i.e.,* $\mathcal{E}(x; \widehat{\mathcal{A}}) = \mathcal{G}_X \circ \mathcal{A}_m^{\alpha_m} \circ \cdots \circ \mathcal{A}_1^{\alpha_1} \circ \mathcal{G}_X^{-1}(x)$.[†]

Broadly, unit interventions describe systematic manipulations of the latent space of a generative process, while counterfactuals describe mapping of these manipulations to the observable data space. Note that due to independence of latent dimensions, our definition of unit interventions easily composes and can model other notions of interventions [198, 175]. Combined with counterfactuals, unit interventions are thus sufficient to model any general input transformations in a formal manner and can be used to characterize the input attributes a network is sensitive to, as shown next.

Definition 17. ***(Invariance.)*** *We say* $f(.;\theta)$ *is invariant to unit intervention* $\mathcal{A}_i$ *if counterfactuals generated by* $\mathcal{A}_i$ *do not increase its loss, i.e.,* $\mathcal{L}(f(\mathcal{D};\theta)) = \mathbb{E}_{\alpha \in \mathcal{Z}_i} \mathcal{L}(f(\mathcal{E}(\mathcal{D}; \mathcal{A}_i^{\alpha}); \theta))$.

Proposition 1. ***(Exhaustiveness of Unit Interventions.)*** *If* $f(.;\theta)$ *is invariant to unit interventions* $\mathcal{A}_i$ *and* $\mathcal{A}_j$*, it must be invariant to their composition. Further, lack of invariance to* $\mathcal{A}_i$ *or* $\mathcal{A}_j$ *precludes invariance to their composition.*

[†]We slightly abuse notation and assume that unit interventions corresponding to all latent dimensions need *not* be mentioned in $\widehat{\mathcal{A}}$: if a dimension is unmentioned, then its value is unmodified.

The above statement shows that studying a model's response to individual unit interventions is sufficient to characterize which attributes of the data a model is using for making its predictions: if a model is invariant to a set of unit interventions, it must be invariant to their composition too; similarly, lack of invariance to a unit intervention is sufficient to preclude invariance to all counterfactuals produced by the composition of that intervention and a set of invariant interventions. This result thus helps us circumvent the need for assessing a model's sensitivity to all possible combinations of interventions to fully characterize it. We are now ready to define mechanistic similarity.

Definition 18. ***(Mechanistic Similarity.)*** *Consider a set of unit interventions $\widehat{\mathcal{A}} := \{\mathcal{A}_i\}$, where $i \in [m]$. For parameters θ, denote the subset of interventions that $f(.;\theta)$ is invariant to as $\mathcal{I}(\theta) \subset \widehat{\mathcal{A}}$. Then, $f(.;\theta_1)$ and $f(.;\theta_2)$ are called mechanistically similar if $\mathcal{I}(\theta_1) = \mathcal{I}(\theta_2)$.*

Fig. 6.2 illustrates mechanistic similarity in an intuitive manner. Formally, given a set of independent transformations (instantiated by use of unit interventions), we say two models are mechanistically similar if they exhibit invariance to the same set of interventions. Our definition shares motivation with the idea of *prediction mismatch*, which involves assessing the number of distinct examples two models produce different predictions on, and has been used in prior work to analyze properties such as calibration and catastrophic interference [462, 463, 464, 465]. In contrast, mechanistic similarity is based on assessment of the number of distinct interventions on the data-generating process to which two models are simultaneously invariant. This makes mechanistic similarity more appropriate for problems involving distribution shifts and robustness, where modeling the data-generating process is of crucial importance [160]. We next extend the definition of mode connectivity to account for mechanistic similarity of two models.

Definition 19. ***(Mechanistic Connectivity.)*** *Consider two minimizers θ_1 and θ_2 of loss $\mathcal{L}(f(\mathcal{D};\theta))$ on a dataset $\mathcal{D}$. Let $\mathcal{E}(\mathcal{D}) := \{\mathcal{E}(\mathcal{D}; \mathcal{A}_i^{\alpha_i \sim \mathcal{Z}_i})\}_{i=1}^m$ denote a set of counterfactual datasets designed by applying unit interventions $\mathcal{A}_i$ to all points in dataset $\mathcal{D}$, where intervention assignments α_i are chosen uniformly from the respective range of values $\mathcal{Z}_i$. Then, θ_1 and θ_2 are called mechanistically connected along the path $\gamma_{\theta_1 \to \theta_2}(t)$ if, for all counterfactual datasets, they are minimizers that exhibit mode connectivity.*

Essentially, if two minimizers exhibit mechanistic connectivity, then there exists a path such that moving along it does not yield increase in loss on the counterfactual dataset described by any pre-defined intervention; that is, all points on the path induce mechanistically similar models. Meanwhile, if two minimizers induce mechanistically dissimilar models, moving along any path between them will necessarily involve a change in the mechanisms used for

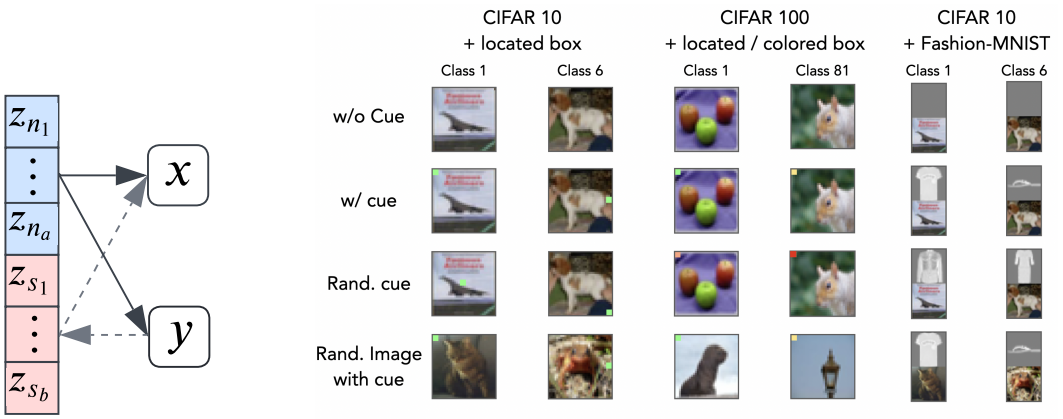


Figure 6.3: **Data-Generating Process (left).** We augment the natural latents $\{z_n\}$ of a data-generating process with a set of synthetic latents $\{z_s\}$. The attributes induced in the input by these synthetic latents are called *cues*. Conditioning (grey, dotted line) the value of a synthetic latent on the target label (y), we can induce correlation between its corresponding cue and the desired model output. If the cue is made easily separable, a DNN will preferentially learn mechanisms to use the cue for making its predictions [11] (see also training curves in App. E.2). **Synthetic Datasets (right).** Following the protocol above, we embed synthetic cues in three existing datasets: (1) CIFAR-10 with 3×3 box cues whose locations depend on the target label; (2) CIFAR-100 with 3×3 box cues colored according to the first digit of the object label, and located according to the second digit; and (3) Dominoes [11], where CIFAR-10 images are concatenated with Fashion-MNIST images of the same class. We analyze counterfactual datasets that involve removing the cue (*w/o Cue*), keeping it (*w/ cue*), randomizing it (*Rand. cue*), or randomizing the natural image (denoted *Rand. image*). These counterfactuals help us ascertain the extent to which a model relies on natural vs. spurious attributes.

making predictions. If this change yields increase in loss on an intermediate point on the path between two minimizers, then it is harmful for the distribution shift described by the corresponding intervention. Mechanistic connectivity is defined to succinctly capture this behavior and characterize the connectivities of mechanistically (dis)similar models.

6.4 Setup for a Mechanistic Evaluation

Before proceeding further, we discuss how we construct mechanistically dissimilar models and assess mechanistic connectivity between them. This allows us to interleave our formal results with experimental verification and demonstrate the validity of our claims in context.

Designing mechanistically dissimilar models. To design models that use different mechanisms for making predictions, we design easily manipulable synthetic datasets that

contain multiple viable discriminative attributes. Specifically, our data-generating process is illustrated in Fig. 6.3 and involves augmenting the natural generative process with synthetic latent variables that are conditioned on the target label. We refer to the attributes induced in the input by such latents as *cues*. By intentionally designing cues that are easily separable, we can exploit the *simplicity bias* of modern DNNs and force our models to preferentially utilize these cues over natural attributes for making their predictions [11]. Training curves for different models are shown in App. E.2 and clearly demonstrate that the process above yields mechanistically dissimilar models: models trained with high correlation between cue and target label rely only on the cue for making predictions, showing invariance to natural attributes; models trained without cues are invariant to them. Importantly, such low-complexity cues can be viewed as stand-ins for spurious or shortcut attributes that are commonplace in realistic settings [439, 442], allowing us to determine whether minimizers that induce models reliant on spurious vs. non-spurious attributes are connected in the landscape.

Generating counterfactuals for analyzing mechanistic connectivity. A primary need for our mechanistic analysis of mode connectivity is the ability to generate counterfactuals via unit interventions. To that end, we highlight that the data-generating process defined above is easy to unit-intervene on. Specifically, since the natural attributes and the synthetically embedded cue are controlled by independent latents, the following counterfactual datasets can be generated via valid unit intereventions: (i) *w/ Cue*: identity intervention that does not alter the cue; (ii) *w/o Cue*: removes the cue from the image; (iii) *Rand. Cue*: randomizes the cue to break its correlation with the target label (e.g., uniformly changing location of the box in the CIFAR-10 with box cue dataset); and (iv) *Rand. Image*: randomizes the natural attributes by altering the underlying image, while keeping the cue intact (e.g., replacing plane with cat). These counterfactuals are especially interesting since they allow us to assess how much a model relies on natural attributes found in the source image vs. our synthetically embedded, spurious cues for making its predictions (see Fig. 6.3).

6.5 Mechanistic Analysis of Mode Connectivity

We now demonstrate how mechanistic similarity of two models affects their connectivity patterns in the landscape. We start with the following proposition, which is a trivial implication that follows from the results of Nguyen et al.[426, 428] and shows mechanistically dissimilar models can indeed be mode connected.

Proposition 2. ***(Mode Connectivity under Mechanistic Dissimilarity) [Informal]*** *Assume θ_1, θ_2 are minimizers of the loss on a dataset $\mathcal{D}$ and induce mechanistically dissimilar*

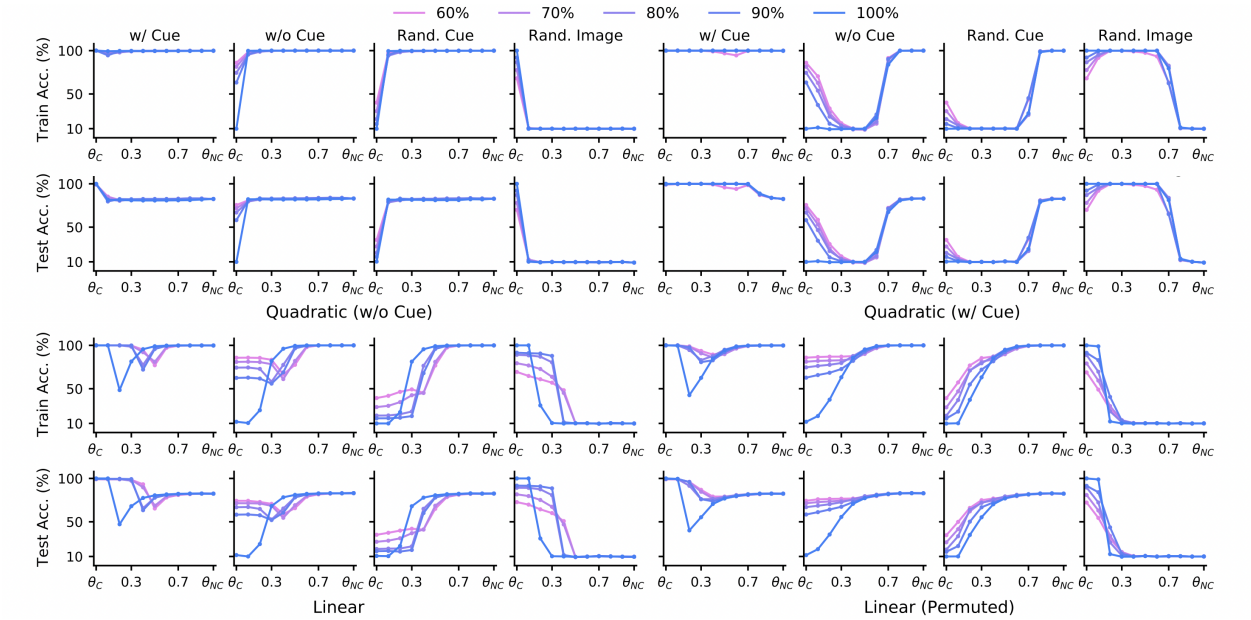
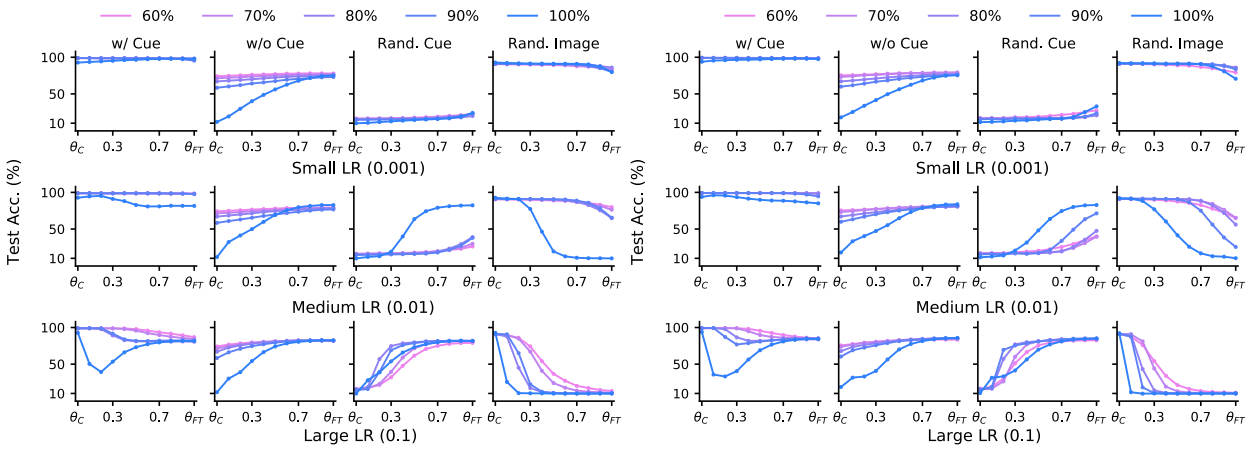


Figure 6.4: **Non-Linear Mode Connectivity of Mechanistically Dissimilar Models.** We train ResNet-18 models on our synthetic CIFAR-10 datasets with and without box-cues (denoted θ_{C} and θ_{NC}, respectively). We evaluate quadratic and linear connectivity paths; quadratic paths identified using both data with and w/o cues are analyzed. Line colors denote proportion of the training data with synthetic cues. Plot titles denote evaluation data (see Fig. 6.3), including data where either the cue is present (w/ Cue), absent (w/o Cue), randomized (Rand. Cue), or the underlying image is randomized (Rand. Image). As shown, θ_{NC} yields the same performance upon randomization of the cue, while the performance of θ_{C} decreases substantially; i.e., the two minimizers induce mechanistically dissimilar models. We see: (i) quadratic paths can be easily identified to mode connect mechanistically dissimilar models; (ii) linear paths cannot be identified, even after permutations; and (iii) mechanistic connectivity is unfounded. See App. E.7 for similar results on other settings and loss curves.

models. Given sufficient overparameterization, there exists a continuous path along which the minimizers are mode connected.

That is, even if two minimizers of loss on a dataset $\mathcal{D}$ induce models that rely on completely distinct mechanisms, *there necessarily exists a continuous path along which the two minimizers exhibit mode connectivity.*

Note, however, the claim above does not yet address the simplicity of these connectivity paths, which is empirically observed to be surprisingly high for minimizers retrieved from the same dataset. To investigate whether this property also holds for mechanistically dissimilar models, we train VGG-13 and ResNet-18 models on the synthetic datasets described in §6.4. We analyze accuracy on counterfactual datasets (see Fig. 6.3) along quadratic and linear paths (see Eq. 6.2), including quadratic paths identified using data with/without cues, linear paths,



(a) VGG. (b) ResNet18.

Figure 6.5: **Analyzing Pre-trained vs. Fine-Tuned Models: Lack of Linear Connectivity implies Mechanistic Dissimilarity.** We train VGG-13 and ResNet-18 models on our synthetic CIFAR-10 dataset with box-cues and perform naïve fine-tuning on data without cues for 100 epochs using different initial learning rates (LR) and a step-decay schedule. Corresponding models are denoted θ_{C} and θ_{FT}; line colors denote proportion of dataset with synthetic cues; titles denote evaluation datasets, similar to Fig. 6.4. We plot test accuracy as a function of location on the linear paths (after permutation). Using a large learning rate or enforcing perfect correlation between the cue and label induces loss barriers along the linear path, i.e., linear mode connectivity does not hold. Simultaneously, the models respond differently to counterfactuals, i.e, they are mechanistically dissimilar and not connected. For a small/medium learning rate, we notice θ_{FT} remains linear mode connectivity θ_{C} on data with cues. Simultaneously, we see the corresponding models responding similarly on counterfactuals and are mechanistically similar. See App. E.8 for similar results on other datasets, models, and loss curves.

and linear (permuted) paths. Results for ResNet-18 are shown in Fig. 6.4 and remaining are deferred to App. E.7. Interestingly, we find minimizers that induce mechanistically dissimilar models can be mode connected via fairly simple paths as well: *we see we can identify quadratic, but not linear, mode connectivity paths for two mechanistically dissimilar models.* In fact, we conjecture that lack of linear connectivity between two models is intricately related to their mechanistic similarity.

Conjecture 1. ***(Lack of Linear Connectivity implies Mechanistic Dissimilarity)*** *If two minimizers θ_1 and θ_2 of the loss $\mathcal{L}(f(\mathcal{D};\theta))$ on a dataset $\mathcal{D}$ cannot be linear mode connected (up to architectural symmetries), their induced models $f(.;\theta_1), f(.;\theta_2)$ must be mechanistically dissimilar.*

In App. E.6, we show the claim above holds true for a 1-hidden layer model on a simplified

data-generating process inspired by our setup. Here, we show extensive empirical evidence of its validity in more complex settings. In particular, we follow the experimental protocol of Neyshabur et al. [451], who demonstrate that a pretrained model exhibits linear mode connectivity on the original pretraining dataset before and after fine-tuning on another target dataset. We thus train VGG-13 and ResNet-18 models on our synthetic datasets with (partially) predictive cues and then fine-tune them on data without cues. Results on CIFAR-10 with box cues are shown in Fig. 6.5; App. E.8 has additional results. We see that *when linear mode connectivity does not hold, the fine-tuned models behave differently on counterfactuals*, i.e., are mechanistically dissimilar to the pretrained model. For example, the models before and after fine-tuning using a large learning rate do not exhibit linear mode connectivity; correspondingly, the fine-tuned models exhibit clear invariance to cue attributes, while the pretrained models do not. Similarly, under perfect correlation between labels and cue attributes, fine-tuned models are not linear mode connected with their pretrained counterparts, and exhibit different behavior on counterfactuals (even for small initial learning rates). We note this latter, specific instance of success in altering the pretrained model's mechanisms via fine-tuning is a result of the model being rendered entirely invariant to natural attributes during pretraining (see App. E.2); consequently, the model lacks any transferable mechanisms for the target data distribution and hence the mechanisms necessarily have to change to fit the new dataset.

A practical takeaway of our results above is that *naïve fine-tuning can fail to alter the mechanisms learned by a model during pretraining.* While large learning rates can help overcome this limitation, they are likely to heavily distort features learned during pretraining [16], rendering pretraining obsolete and the sample complexity of fine-tuning similar to that of training from scratch [29]. This indicates that for fine-tuning to be useful, pretraining must be performed with care to ensure desirbale mechanisms relevant to downstream tasks are learned. If incorrect mechanisms, such as ones that rely on spurious attributes, are learned, mere fine-tuning on some "clean" dataset will be insufficient to alter the model's behavior, as hinted at by results in few recent works [466, 467, 468, 469]. Intriguingly, this latter strategy of fine-tuning on a clean dataset forms the basis of several recent methods on improving DNNs' robustness [15, 452, 16, 470]: such methods fine-tune some/all layers of a pretrained model on a *minimal* dataset that is known to not contain the spurious attribute we want to reduce the model's reliance on. In the next section, we perform a thorough counterfactual evaluation on our synthetic datasets to assess if such methods can actually alter a model's behavior.

6.6 Altering a Model's Mechanisms Efficiently

In this section, our goal is to show that our newfound understanding of DNN loss landscapes from a mechanistic perspective (see Fig. 6.6) can be used to devise a sample-efficient strategy that allows targeted altering of a model's mechanisms. We primarily see the results below as further corroboration of our analysis in §6.5.

6.6.1 Connectivity-Based Fine-Tuning (CBFT)

As defined in our work, mechanistic dissimilarity corresponds to lack of shared invariances. Our results in §6.5 demonstrate that lack of linear connectivity between two models implies they will be mechanistically dissimilar. *A valid strategy for altering a model's mechanisms then involves moving the model to a region in the landscape that does not exhibit linear connectivity to the current minimizer.* Of course, we specifically want the unshared invariance to correspond to ignoring of the spurious attribute (denoted C) that we desire to reduce the model's reliance on. For this purpose, we follow prior works and assume access to a *minimal* dataset $\mathcal{D}_{\text{NC}}$ that does *not* contain the attribute C. Note that this setting is not similar to the often used setup in domain adaptation, where the original training dataset (denoted $\mathcal{D}_{\text{C}}$ here) and the novel dataset, $\mathcal{D}_{\text{NC}}$, are assumed to be pairs of images in different environments.

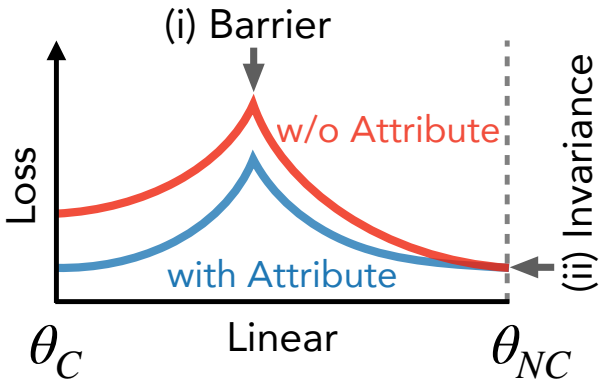


Figure 6.6: **Clues for altering a model's mechanisms.** Given a discriminative attribute C, the loss landscape along the linear path connecting a model invariant to the attribute (θ_{NC}) versus a model that relies on the attribute (θ_C) generally shows (i) a loss barrier along the path and (ii) invariance at the endpoint corresponding to θ_{NC}, i.e., $\mathcal{L}(\theta_{NC}, D_{NC}) = \mathcal{L}(\theta_{NC}, D_C)$.

In the following, we use $\mathcal{D}^i$ to denote the subset of examples in dataset $\mathcal{D}$ belonging to the i^{th} class in a K-class classification problem, $\gamma_{\theta \to \theta_{\text{C}}}(t)$ to denote the linear path between a set of parameters θ and the pretraining solution θ_{C}, and $f_r(x; \theta)$ to denote the model's representation for an input x at the penultimate layer. Let $\mathcal{N}_{\text{Tr}}$ denote the Truncated Gaussian [471] with mean/std of 0.5 that is constrained to the range $[0, 1]$. Our method,

Connectivity-Based Fine-Tuning (CBFT), involves minimizing the following loss:

$$\begin{aligned}
&\mathcal{L}_{\text{CBFT}} = \mathcal{L}_{\text{CE}}(f(\mathcal{D}_{\text{NC}};\theta), y) + \mathcal{L}_{\text{B}} + \frac{1}{K}\mathcal{L}_{\text{I}}, \text{ where} \\
&\mathcal{L}_{\text{B}} = \mathbb{E}_{t\sim\mathcal{N}_{\text{Tr}}}|\lambda_{\text{B}} - \mathcal{L}_{\text{CE}}(f(\mathcal{D}_{\text{C}};\gamma_{\theta\to\theta_{\text{C}}}(t)), y)| \text{ and} \\
&\mathcal{L}_{\text{I}} = \sum_{k=1}^{K} \left\| \mathbb{E}_{x\in\mathcal{D}_{\text{C}}^{k}}(f_r(x;\theta)) - \mathbb{E}_{\tilde{x}\in\mathcal{D}_{\text{NC}}^{k}}(f_r(\tilde{x};\theta)) \right\|_2^2.
\end{aligned} \tag{6.1}$$

Here $\mathcal{L}_{\text{CE}}$ denotes the cross-entropy loss and promotes learning of correct labels on the minimal dataset $\mathcal{D}_{\text{NC}}$, while $\mathcal{L}_{\text{B}}$, $\mathcal{L}_{\text{I}}$ instantiate the two principles discussed in Fig. 6.6: $\mathcal{L}_{\text{B}}$ denotes a "barrier loss" that randomly samples a point on the linear path between θ, θ_C and maximizes the loss at this point up to an upper bound λ_{B} (=1 in all our experiments) and $\mathcal{L}_{\text{I}}$ denotes an invariance loss that promotes reducing the distance between class-average representations on $\mathcal{D}_{\text{NC}}$ and $\mathcal{D}_{\text{C}}$. Overall, $\mathcal{L}_{\text{B}}$ helps CBFT find a set of parameters θ that does not exhibit linear connectivity to θ_C, while $\mathcal{L}_{\text{I}}$ helps CBFT pick a solution that is (approximately) invariant to attribute C. We emphasize that since the cross entropy loss can be made arbitrarily large, using the hyperparameter λ_{B} is important. We also note that using class-average representations to learn (approximately) invariant representations has the advantage of not requiring access to simultaneous pairs of samples in different environments, i.e., ones with and without the spurious attributes [472, 473].

Evaluating CBFT: We empirically validate the effectiveness of CBFT by using our synthetic datasets from §6.4 as a benchmark. We compare CBFT against naïve fine-tuning, Last-Layer Re-Training (LLR) [15, 452], and Linear Probe plus Fine-Tuning (LPFT) [16] (see App. E.2.2 for implementation details). Results are reported in Tab. 6.1. We see that while the baselines perform well on clean data, *they do not yield desired behavior on counterfactual datasets*: e.g., they achieve high accuracy even if we randomize the image, indicating that they are more sensitive to the cue. In contrast, we see that beyond just performing well on clean data, CBFT models show the desired behaviors: sensitivity to randomization of the image and invariance to spurious attributes. These results suggest CBFT successfully alters a model's mechanisms and provides further corroboration to the claim that lack of linear connectivity implies mechanistic dissimilarity between two models (see Conj. 1). We also provide detailed ablations for CBFT in App. E.5 and find both losses, $\mathcal{L}_{\text{B}}$ and $\mathcal{L}_{\text{I}}$, are important for getting the desired results: the barrier loss helps induce a mechanistically dissimilar model, while the invariance loss helps select the mechanisms we desire.

Table 6.1: **Evaluating CBFT.** We train ResNet-18 models on our synthetic CIFAR-10, CIFAR-100, and Dominoes dataset with different proportions of samples with cue features and fine-tune them using 2500 "clean" samples from a dataset without any cues. Test accuracies (%) on counterfactual test datasets with No Cue (NC), with Cue (C), Randomized Cue (RC), and Randomized Image (RI) are reported (mean of three seeds). We compare our method, Connectivity-Based Fine-Tuning (CBFT), with several baselines: Fine-tuning with a medium/small learning rate ($FT_{M/S}$), LLR [15], and LPFT [16]. $\sim$ denotes invariance is desirable, i.e., accuracy should be similar to that on NC; $\uparrow/\downarrow$ indicate higher/lower accuracy is desirable; best results are in bold. We generally see that all baselines yield large degradations in the absence of cues, and even achieve very high accuracy when the underlying image is randomized. Meanwhile, CBFT is able to break reliance on cues, inducing representations that are completely invariant to their presence.

	60% Cue data				70% Cue data				80% Cue data				90% Cue data			
C-10	NC$^{\uparrow}$	C$^{\sim}$	RC$^{\sim}$	RI$^{\downarrow}$	NC$^{\uparrow}$	C$^{\sim}$	RC$^{\sim}$	RI$^{\downarrow}$	NC$^{\uparrow}$	C$^{\sim}$	RC$^{\sim}$	RI$^{\downarrow}$	NC$^{\uparrow}$	C$^{\sim}$	RC$^{\sim}$	RI$^{\downarrow}$
FT_M	**75.7**	98.4	23.6	83.4	**75.8**	98.6	27.7	78.6	**71.3**	97.7	37.6	63.6	67.2	95.4	49.6	46.6
FT_S	75.8	98.7	17.5	90.1	74.9	98.8	16.3	91.1	69.9	98.4	15.7	90.9	64.7	97.9	15.3	90.7
LLR	71.6	95.1	36.3	57.1	70.9	95.8	29.9	65.8	65.1	81.8	27.0	53.2	59.3	70.7	24.6	40.7
LPFT	70.6	88.1	21.0	70.7	69.6	87.3	18.7	72.5	64.4	63.8	18.8	48.0	59.7	56.6	19.8	37.8
CBFT	74.1	**71.5**	**73.4**	**8.75**	73.2	**69.2**	**72.3**	**8.60**	70.0	**70.0**	**69.5**	**9.68**	**67.9**	**72.5**	**68.1**	**13.1**
C-100	NC$^{\uparrow}$	C$^{\sim}$	RC$^{\sim}$	RI$^{\downarrow}$	NC$^{\uparrow}$	C$^{\sim}$	RC$^{\sim}$	RI$^{\downarrow}$	NC$^{\uparrow}$	C$^{\sim}$	RC$^{\sim}$	RI$^{\downarrow}$	NC$^{\uparrow}$	C$^{\sim}$	RC$^{\sim}$	RI$^{\downarrow}$
FT_m	**44.4**	99.2	12.8	85.3	**40.3**	99.6	12.3	89.8	33.6	99.0	11.4	90.5	25.2	79.2	9.79	57.9
FT_s	43.1	99.6	10.3	93.6	38.2	99.7	10.5	95.7	32.5	99.6	10.4	97.0	24.5	39.4	4.87	30.9
LLR	35.5	99.2	12.1	89.0	31.5	98.6	11.3	89.6	25.3	96.7	10.6	89.4	18.9	75.1	9.1	58.7
LPFT	35.1	93.2	10.3	82.3	31.1	90.2	9.89	78.5	25.6	89.6	9.70	80.8	18.7	**28.6**	**4.42**	**19.6**
CBFT	42.7	**65.0**	**36.4**	**14.6**	38.5	**66.7**	**34.7**	**21.2**	**34.6**	**69.3**	**23.0**	**27.9**	**28.5**	72.9	23.2	46.0
Dom.	NC$^{\uparrow}$	C$^{\sim}$	RC$^{\sim}$	RI$^{\downarrow}$	NC$^{\uparrow}$	C$^{\sim}$	RC$^{\sim}$	RI$^{\downarrow}$	NC$^{\uparrow}$	C$^{\sim}$	RC$^{\sim}$	RI$^{\downarrow}$	NC$^{\uparrow}$	C$^{\sim}$	RC$^{\sim}$	RI$^{\downarrow}$
FT_m	**77.4**	96.8	43.8	56.1	**76.6**	96.6	42.7	58.7	**74.1**	95.7	41.7	61.3	**68.8**	95.1	40.0	57.5
FT_s	76.4	96.9	37.5	62.4	76.8	96.6	32.5	66.5	73.2	96.4	30.8	67.7	67.3	95.2	31.2	65.6
LLR	74.6	94.4	39.8	53.0	73.9	93.2	36.3	54.7	70.8	84.8	33.1	46.6	63.3	77.0	31.2	39.0
LPFT	73.2	92.5	38.0	51.8	72.7	88.0	34.8	50.9	69.4	34.8	33.1	39.1	61.2	60.8	31.2	26.6
CBFT	72.0	**64.9**	**67.5**	**9.9**	71.5	**70.0**	**59.2**	**12.1**	70.8	**69.7**	**65.9**	**11.9**	67.2	**68.7**	**61.5**	**14.9**

6.7 Related Work

Mode connectivity. Existence of a single, continuous manifold connecting global minimizers was first claimed by Freeman and Bruna [424] and Nguyen [426], and empirically discovered in concurrent works under the title of "mode connectivity" by Garipov et al. [73] and Draxler et al. [74]. A geometrical characterization of this manifold was provided by Simsek et al. [423], who showed the manifold is *primarily* composed of affine subspaces. Connectivity properties of neural networks have been used for designing and analyzing algorithms for several practically relevant applications, such as ensembling [453, 474, 475, 476], network pruning [429, 430], optimization [477], adversarial robustness [478], and multi-task/continual learning [479, 480]. During the course of this work, we became aware of the contemporary work by Juneja et al. [481]. Therein, the authors use NLP datasets designed by McCoy et

	Linear Mode	Nonlinear Mode	Linear Mech.	Nonlinear Mech.
Mech. Similar	✓	✓	✓	✓
Mech. Dissimilar	✗*	✓	✗	✗

Table 6.2: **Summarizing our Findings.** ✓, ✗ respectively indicate whether there always exist paths along which mechanistically (dis)similar models identified using gradient-based optimization can exhibit the type of connectivity specified in the column title. * denotes there are exceptional, but primarily theoretical, cases where the connectivity definition can hold (see App. E.6).

al. [482] to perform an empirical analysis similar to ours, finding that models that lack linear connectivity show different generalization behaviors, relying on different attributes of an input to make their predictions. Our work further formalizes this result: we show lack of linear connectivity implies mechanistic dissimilarity. The results by Juneja et al. thus provide further corroboration for our claims on a different modality.

Fine-tuning and Model Editing. Fine-tuning is a well-established practice in deep learning. The most basic fine-tuning method is to treat the pretrained model as an initialization, and continue training with new data. A variant is to train only a subset of parameters, such as the final classification layer [15, 452], possibly fine-tuning the entire model after that [16, 470]. A related application to fine-tuning, model editing has become quite popular recently and approaches for the same generally aim to make a targeted change to a model's factual knowledge [483, 484, 485]. For instance, Sinitsin et al. [485] give the example of correcting a model's prediction error on a particular example without changing its predictions on other examples. Prior work on model editing aims to make changes that are "local" in input space, e.g., only affecting the model's "understanding" of who the current prime minister of the UK is. CBFT shares this motivation of "targeted" alteration of a model; however, instead of altering the model's factual knowledge, the overarching goal of CBFT is to make changes to the specific rules or mechanisms the model implements to make its predictions (see [486] for a discussion on distinction between rule vs. exemplar / factual inference strategies). Specifically, CBFT aims to make a model invariant to features that it was not already invariant to (or vice versa), without changing any of its other learned invariances.

6.8 Discussion

Depending on the mechanisms they learn for making their predictions, neural networks trained on a specific data distribution can nonetheless differ vastly in their behavior when evaluated on other distributions. This realization prompted us to perform a mechanistic characterization of connectivity properties in the loss landscape of neural networks. Our proposed notion of *mechanistic similarity* instantiates the idea as shared invariances, and helps extend the prior notion of mode connectivity to account for mechanistic similarity. Our analysis reveals several surprising findings (see Tab. 6.2): (i) mechanistically dissimilar minimizers can be mode connected via relatively simple, but non-linear, paths; (ii) linear mode connectivity of two minimizers is intricately related to the mechanistic similarity of their induced models; (iii) naïve fine-tuning can fail to eliminate spurious attributes learned during pretraining; and (iv) finding linearly disconnected regions in the landscape enables sample-efficient alteration of a model's mechanisms.

6.9 Future Work

Future work can involve use of counterfactual generators based on modern generative models [487] to extend our synthetic data experiments and corroborate our claims in naturalistic settings. Especially the effectiveness of CBFT at removing undesirable mechanisms in a model warrants further investigation. Further, our proposed formalism, which currently operates an aggregate, dataset level can plausibly be extended towards a per-sample analysis. Theoretical characterization in other model architectures will also be useful, as our current model is extremely simplistic (1-hidden layer). Tools for going beyond can be performing a per-layer analysis by assuming a random initialization for remaining layers [488].

We also believe our analysis can be useful to reason about benefits and limitations of recent averaging-based ensembling methods [489, 476, 490, 491]. Specifically, note that our claims do not preclude possible linear connectivity of mechanistically dissimilar models: in fact, any two solutions of the linear system of equations $y = Wx$ can be interpolated regardless of their prediction mechanisms (hence the * in Tab. 6.2). However, as we show in App. E.6 in a simplified setup, these different mechanisms should be of similar "complexity" to enable linear connectivity (e.g., mechanisms corresponding to linearly separable attributes). In the context of our fine-tuning results, this implies naïve fine-tuning can work well on a target distribution only if the desired mechanism is of similar complexity to the mechanism for identifying the spurious attribute (which would possibly imply it finds a spurious attribute again); otherwise, a loss barrier must be surmounted for successful learning on the target

distribution. This suggests that pretraining should aim to promote learning of a variety of expressive prediction mechanisms.

CHAPTER 7

Mechanistic Analysis of Fine-Tuning on Procedurally Defined Tasks

7.1 Introduction

Large language models (LLMs) pretrained on huge, web-crawled text datasets demonstrate extremely general capabilities [100, 137, 9, 20]. This has led to the current paradigm of machine learning, where practitioners often use model adaptation protocols such as fine-tuning to achieve unprecedented performance on a broad range of downstream tasks [30, 492, 493, 494, 35]. Relatedly, the generality of an LLM's capabilities implies the model also learns to exhibit several undesirable behaviors, e.g., producing sensitive, biased, or toxic outputs in the pursuit of completing a task [495, 496, 288, 497, 498, 499, 500]. Fine-tuning with different training objectives has again seen immense usage in mitigating such "unsafe" capabilities, serving as an integral component of current state-of-the-art *alignment* approaches like RLHF [143, 501, 502, 144, 503].

Given its ubiquity in the design of both performant and safely deployable models, a natural question emerges: precisely how does fine-tuning influence a pretrained model's capabilities to adapt to a downstream dataset (see Fig. 7.1)? The generality of an LLM's capabilities opens the possibility that fine-tuning protocols merely identify the most relevant capabilities and amplify their use for a given set of inputs, while inhibiting the use of other capabilities. Arguably, results on jailbreaking alignment-finetuned LLMs via adversarially generated prompts to elicit undesirable behavior support this hypothesis [504, 72, 505, 506, 507]; however, a precise study to establish the phenomenology of fine-tuning remains absent from the literature. It therefore remains unclear how pernicious this problem is.

Motivated by the above problem, we perform an extensive analysis of the effects of fine-tuning on a pretrained model's capabilities in *controlled settings* where we can use mechanistic interpretability tools to understand precisely what is happening to the model's underlying capabilities. Specifically, we focus on the following two setups: (i) compiled transformer

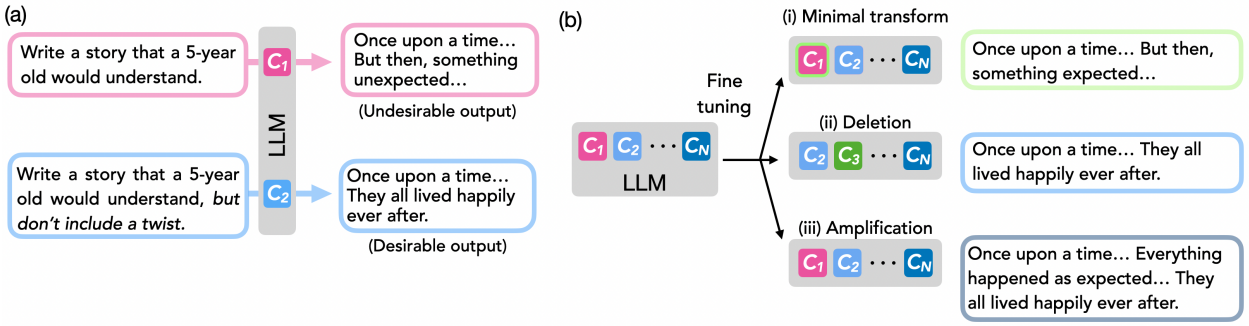


Figure 7.1: **How does fine-tuning alter a model's capabilities?** (a) Pretraining on huge, web-crawled datasets leads to LLMs learning several capabilities that can justifiably process an input. The figure shows this using an illustrative query, "write a story a 5-year old would understand." Via careful prompting, the desired answer can be retrieved, indicating both desired and undesired capabilities exist in an LLM. (b) Upon fine-tuning, e.g., to avoid use of undesirable capabilities, we hypothesize that three explanations are possible: (i) a minimal transformation of the original capability is learned, e.g., a negation of the original capability; (ii) the undesirable capability is deleted altogether; or (iii) the use of another relevant capability is amplified.

models based on the *Tracr library* [508, 509], which allows encoding specific computational programs into a transformer, and (ii) procedurally generated setups involving *probabilistic context-free grammars (PCFGs)* [385, 232], a formal model designed to capture syntactic properties of natural and programmatic languages that has recently served as a test-bed for mechanistically understanding language models [12, 510, 511]. While Tracr allows us to analyze models with perfectly encoded capabilities, models trained on PCFGs allow evaluation of the effects of design choices involved in the pretraining pipeline. Fine-tuning these models via the often-used protocol of further training a pretrained model on a downstream dataset with a sufficiently small learning rate, we make the following findings.

- **Fine-tuning alters pretraining capabilities by minimally transforming them.** We find that when a relevant pretraining capability is present, the fine-tuned model learns a minimally transformed version of it. We call the transformed portion a *wrapper*.
- **Wrappers are generally very localized.** We show that the wrappers transforming a model's pretraining capabilities are often extremely localized: e.g., via mere pruning of a few weights or neurons, we show the model can start to reuse its pretraining capability and unlearn how to perform the downstream task. Relatedly, we find that via a simple linear probe, we are still able to retrieve outputs expected from the pretrained model.
- **Reverse fine-tuning to "revive" a capability.** In scenarios where upon fine-tuning a model *behaviorally* seems to not possesses a capability, we find that further fine-tuning

the model on a subset of pretraining data leads to a sample-efficient "revival" of the capability. We corroborate these results in a realistic setup using the TinyStories dataset [14].

7.2 Related Work

Fine-tuning in the "foundation model" era. Fine-tuning large-scale foundation models pretrained on huge datasets, such as LLMs [137, 9] or large vision models [22, 264], has become the norm in most domains of machine learning. Accordingly, several fine-tuning methods have been proposed in recent years, e.g., instruction fine-tuning [138, 512, 142], parameter-efficient fine-tuning [513, 514, 515], low-rank adaptation [516, 517, 105], and weight averaging [518, 519]. The diversity of these protocols makes fine-tuning a general, umbrella term for related methods used to adapt a pretrained model to elicit its most relevant capabilities. *For precision, we restrict this work to fine-tuning protocols that continue training of a pretrained model on a smaller downstream dataset at a learning rate that is often one to three orders of magnitude lower than the average pretraining one.* Such protocols are widely used in practice, e.g., in instruction fine-tuning [138].

Understanding fine-tuning. A few papers theoretically analyze fine-tuning [520, 521, 522, 523, 16] under strong assumptions such as relatively simple model classes (e.g., linear functions) or a kernel view of deep learning, which, as shown by Yang et al. [524], trivializes the notion of feature transfer in fine-tuning / transfer learning (though see Malladi et al. [525] for a notable exception). Prior works have also evaluated the effects of fine-tuning via the lens of mode connectivity [481, 85], behavioral evaluations [466], and intrinsic dimensionality of the loss landscape [526]. In contrast, we aim to provide a mechanistic analysis of how fine-tuning changes model capabilities. Relevant recent work [527, 528] claim that fine-tuning is unlikely to alter a model's capabilities and support this claim by providing a behavioral analysis of linear regression tasks and of realistic models via a novel prompting strategy.

Model interpretability via synthetic tasks. Several recent works have focused on *mechanistically* understanding how Transformers learn synthetic language generation tasks, such as learning formal grammars and board games [12, 529, 530, 260, 236, 195, 234, 531, 532]. The goal of such works, including ours, is not necessarily to provide accurate explanations for the success of LLMs, but to develop concrete hypotheses that can be used to develop grounded experiments or tools for understanding their behavior. For example, in a recent work, Allen-Zhu et al. [533, 534] use a synthetically designed setup to develop hypotheses for how "knowledge" about an entity is stored in a pretrained model, showing such knowledge can often be manipulated via relatively simple linear transformations. Similarly, Okawa et al. [86]

use a procedurally defined multimodal dataset to demonstrate that emergent capabilities seen in neural networks are partially driven by the compositional nature of real world data. In another work, Zhou et al. [531] utilize Tracr compiled Transformers to hypothesize and demonstrate that if primitive operations involved in a formal algorithm can be implemented by a model, length generalization if practically feasible. Feng et al. [535] use context-free grammars to demonstrate stepwise inference allows Transformers to solve problems that require dynamic programming.

7.3 Defining our notion of capabilities

For precision and to motivate our experimental setup, we first discuss the notion of capabilities that we aim to capture for analyzing how fine-tuning alters a model (see Tab. F.1 for a summary of notations). We use an idealized definition to communicate our primary intuition and emphasize that we do not expect all capabilities in a pretrained model will act as perfectly as the definition necessitates. However, for the procedural tasks used in this work, our idealized notion is fairly representative. The definitions are motivated by standards of empirical work in language models, as discussed in context below.

Let $\mathcal{D}_{\texttt{PT}}$ denote a dataset sampled from a distribution $\mathcal{P}_{\text{X}}$ over the domain X. We will assume the domain X can itself be factorized into two domains X_I and X_D. Correspondingly, a sample $x \in \text{X}$ can be divided into a tuple of variables $(x_i \in \text{X}_I, x_d \in \text{X}_D)$, where x_i identifies which capability a model *should use* to process the information encoded by the variable x_d. The goal of this decomposition is to capture the idea that instruction-tuned language models enforce decomposition of inputs into an instruction (the specific task a user wants the model to perform) and information (the precise data the user wants said task to be performed upon). That is, given a string, there are multiple tasks the user can a general-purpose system like an LLM to perform; the instruction part of the input clarifies which specific task is to be performed, i.e., *which capability the model should use to process the input.*

We denote the identifier of capability c using i_c. Pretraining on $\mathcal{D}_{\texttt{PT}}$ yields us an $\texttt{L}$-layer model $\texttt{M}(.)\colon \text{X} \to \text{Y}$, where often $\text{Y} = \text{X}$ for language models. Let $\texttt{Read}_l(\texttt{M}(.))$ denote the action where a linear layer is trained on intermediate outputs at layer l of model $\texttt{M}(.)$ using $\mathcal{D}_{\texttt{PT}}$. Under this setup, we define a capability as follows.

Definition 20. (Capability.) *Define a surjective map $f_{\mathcal{C}} : \text{X}_{\mathcal{D}} \to \text{Y}_{\mathcal{C}}$, where $\text{Y}_{\mathcal{C}} \subset \text{Y}$. Let $\text{X}_{\mathcal{C}} \subset \text{X}$ be a sub-domain s.t. $\forall\, x \in X_{\mathcal{C}}$, the capability identifier variable is the same, i.e., $x_i = \texttt{i}_{\mathcal{C}}$. Then, we say the model $\texttt{M}(.)$ "possesses a capability $\mathcal{C}$" if for all $x \in \text{X}_{\mathcal{C}}$, $\exists\, l \leq \texttt{L}$ s.t. $\texttt{Read}_{\texttt{l}}(M(x)) = f_{\mathcal{C}}(x_d)$.*

A linear readout at an intermediate layer is used in the definition above to emphasize that the notion of a capability need not correspond to only input–output behavior. Further, the definition is restricted to a sub-domain of the overall input space, which we argue is important to define a system's capabilities. For example, one can claim an 8-bit adder circuit possesses the capability to perform addition, but, technically, this is true only over the domain of 8-bit numbers; for inputs with more than 8-bit precision, the circuit will see an overflow error, generating an incorrect but syntactically valid output. Similarly, an LLM may possesses the capability to identify the sentiment of a passage of text in a specific language, but *possibly* fail when inputs in a different language are shown. Such structured failures imply claiming the existence of a capability should account for the input domain.

We next consider how the fine-tuning distribution $\mathcal{P}_{\mathrm{X}}^{\mathtt{FT}}$ over the domain X can interact with capabilities exhibited in the pretrained model. Our goal here is to capture the fact that a large-scale pretraining corpora is likely to have non-zero probability under the fine-tuning distribution, i.e., it is unlikely that a pretrained model will lack *any* capability relevant to the downstream task. This motivates a notion of "relevance of a capability". Specifically, let $\mathcal{D}_{\mathtt{FT}} \sim \mathcal{P}_{\mathrm{X}}^{\mathtt{FT},\mathtt{E}}$ denote the downstream dataset used for fine-tuning, where $\mathcal{P}_{\mathrm{X}}^{\mathtt{FT},\mathtt{E}}$ is the empirical distribution that captures a subset of the support with non-zero probability in the distribution $\mathcal{P}_{\mathrm{X}}^{\mathtt{FT}}$.

Definition 21. (Relevance of a Capability.) *Assume the capability $\mathcal{C}$ in a pretrained model can be transformed to a map $\mathtt{g} \circ f_{\mathcal{C}}$ via fine-tuning on $\mathcal{D}_{\mathtt{FT}}$, where $|\mathcal{D}_{\mathtt{FT}}| \ll |\mathcal{D}_{\mathtt{PT}}|$, such that for all $x \sim \mathcal{P}_{\mathrm{X}}^{\mathtt{FT},\mathtt{E}}$, the correct output is produced. Then, if for all $x \sim \mathcal{P}_{\mathrm{X}}^{\mathtt{FT}}$, $\mathtt{g} \circ f_{\mathcal{C}}$ yields the correct output, we claim capability $\mathcal{C}$ is* ***strongly relevant*** *to the fine-tuning task; else, we call it* ***weakly relevant****.*

For example, a *weakly relevant* capability can involve the ability to recognize a spurious attribute that the model can learn to exploit to perform well on the fine-tuning dataset, without enabling generalization to the overall distribution that the fine-tuning dataset is sampled from. Meanwhile, a *strongly relevant* capability is one that extracts a causally relevant feature for that task (see Fig. 7.2 for an example). When a weakly relevant pretraining capability is available, we empirically observe that we can *often* identify

	Identification of negation words (Weakly relevant)	Sentiment recognition (Strongly relevant)
Complete the passage while maintaining its narrative.	The movie was definitely not bad. I really can't believe they continue to make these. They're just a waste of resources!	The movie was definitely not bad. More character building could have helped develop the plot, but I loved the twists and action!

Figure 7.2: **Capability Relevance.** Consider the task of completing a passage while maintaining its narrative. Herein, the ability to recognize the sentiment of a text will be deemed *strongly relevant* and the ability to recognize negative words *weakly relevant*. Such words are often spuriously correlated with a negative sentiment.

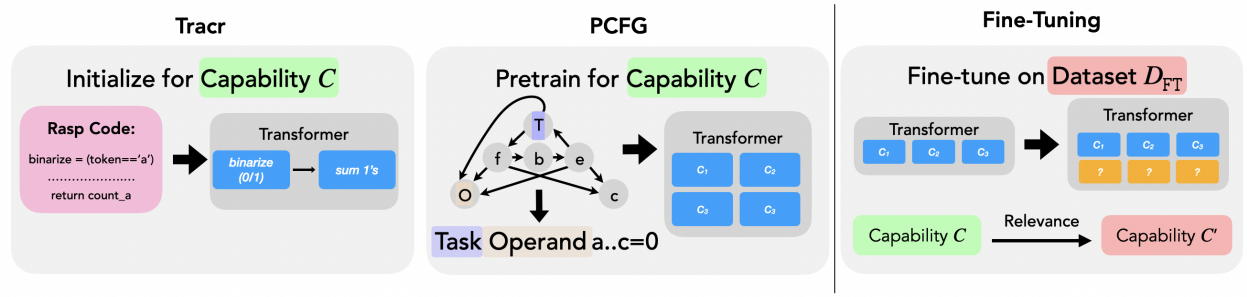


Figure 7.3: **Experimental setup.** We primarily analyze two setups: (i) Tracr "compiled" models with predefined capabilities and (ii) models trained to learn capabilities defined via a PCFG, following prior work [12]. During fine-tuning, we train the model on a dataset $\mathcal{D}_{\text{FT}}$ that promotes learning of a capability $\mathcal{C}'$. We randomly embed spurious attributes in the fine-tuning dataset that correlate with features extracted by a pretraining capability $\mathcal{C}$ to operationalize capability relevance.

specific components in the latter half of the model (e.g., neurons or layers) that seem to implement the transform $\mathtt{g}$ in Def. 21. In such cases, we call $\mathtt{g}$ a "wrapper" and $\mathtt{g} \circ \mathcal{C}$ a "wrapped capability". If we intervene on the model by either removing the wrapper or training it to forget the wrapper, we find the model starts to perform well on the pretraining task again. In such cases, we say the pretraining capability has been "revived".

7.4 Building Capable Models: Tracr and PCFGs

We next describe the setup used in this work for analyzing how fine-tuning alters a model's capabilities (see Fig. 7.3 and App. F.2). Due to lack of clarity on what capabilities a language model possesses or what training data it has seen, we primarily focus on procedurally defined setups that enable clear interpretability. To understand how the relevance of a capability affects fine-tuning, we randomly embed a predefined spurious attribute into the fine-tuning dataset. Specifically, the attribute correlates with the features extracted by the pretraining capability—if the attribute is "simple" enough, the model preferentially exploits it to reduce the downstream loss [11, 536].

Compiled capabilities with Tracr. For a fully controllable system, we use the recently proposed Tracr library [508]. Tracr enables "compiling" a transformer model with a set of predefined computational primitives over a string of characters from the English alphabet. Accordingly, we define a set of capabilities as a Tracr program and compile it into a Transformer via Tracr (see App. F.2.1 for a detailed pipeline). The model is then fine-tuned on a downstream task to which the compiled capability may either be weakly or strongly relevant.

While we analyze two tasks in this setup, for the main body of the chapter, we focus on only the following one.

- **Counter:** Compile the capability to count the number of occurrences of a token $\texttt{O}_{\texttt{PT}}$ in a string into the model; fine-tune to count occurrences of another token $\texttt{O}_{\texttt{FT}}$. If $\texttt{r}(x, \texttt{O})$ denotes the number of occurrences of a token $\texttt{O}$ in a string x, the spurious correlation is defined by enforcing a constant difference in token occurrences, i.e., $\texttt{r}(x, \texttt{O}_{\texttt{FT}}) - \texttt{r}(x, \texttt{O}_{\texttt{PT}}) = \texttt{q}$.

As an example, note that in the Counter setup, the model can exploit its pretraining capability and get the correct output on the fine-tuning dataset by merely adding $\texttt{q}$ to the count of $\texttt{O}_{\texttt{PT}}$ tokens. This *wrapped capability* will however perform poorly on samples without the correlation.

Learned capabilities with PCFGs. In this setup, capabilities are "learned", akin to practical situations. This allows us to probe the effects of different pretraining design choices, e.g., the distribution of the pretraining data. Specifically, we follow recent work [12] and train a minGPT model [537] via autoregressive training on probabilisitc context-free grammars (PCFGs), a formal model of language that captures syntactic properties. Broadly, the data-generating process involves a tree traversal (see Fig. 7.3), starting from an initial root node and randomly choosing and navigating a set of production rules of the grammar from start/intermediate nodes to intermediate/terminal nodes, stopping only when a terminal node is reached. The terminal nodes reached by all paths starting at the root node will be concatenated to define a string x from the grammar (see Appendix for more details). We prepend special tokens $\texttt{T}$ and $\texttt{O}$, called "task family" and "operand" tokens, that specify a certain task must be performed on the string x; e.g., count the occurrences (a task family) of a certain token (operand) in a string. Overall, a specific pair of the task family and operand tokens instantiates a task in our setup. The ground truth output of this task and a special token indicating that the output should be produced at the next position are appended at the end of the string in the training data (see App. F.2.2 for further details and Fig. F.4 for an example). Our experiments thus involve the following steps.

(i) Pretrain on a set of task families. Every sample begins with the task family and operand tokens to specify the task. This ensures different tasks do not "conflict" (assign different labels to the same input), since, by construction, they have non-overlapping support.

(ii) Fine-tune on a task which may or may not have been included during pretraining.

(iii) Evaluate how this fine-tuning affects the model.

The data-generating process involves a uniform prior over task family tokens; meanwhile, the set of operand tokens seen during pretraining, denoted $\{\texttt{O}_{\texttt{PT}}\}$, have a multinomial sampling prior. Specifically, the probability of sampling a specific token $\texttt{O}_{\texttt{PT}} \in \{\texttt{O}_{\texttt{PT}}\}$ under task $\texttt{T}$ is denoted $\mathcal{P}_{\texttt{T}}(\texttt{O}_{\texttt{PT}})$. If this probability is low, the model may not learn the relevant capability to perform the task specified by the special tokens. While we analyze the effect of fine-tuning in two broad setups, using a model pretrained on five distinct task families relating to counting and indexing elements of a string, we focus on only the following one in the main body.

- **Counter:** We intentionally reuse this task to demonstrate the effects of compilation of a capability via Tracr versus learning the capability via PCFGs. Specifically, the model is pretrained to learn to count the occurrences of tokens from a *set* of operand tokens $\{\texttt{O}_{\texttt{PT}}\}$ and is fine-tuned to exclusively count occurrences of a token $\texttt{O}_{\texttt{FT}} \in \{\texttt{O}_{\texttt{PT}}\}$. By making the sampling probability of $\texttt{O}_{\texttt{FT}}$ tokens high during pretraining, we can make the model preemptively performant on the downstream task; this allows us to model the notion of capability relevance.

7.5 Results and Discussion

In the following sections, we provide several results indicating that fine-tuning rarely elicits meaningful changes to pretraining capabilities. To this end, we borrow several protocols commonly used in the field of mechanistic interpretability for our analysis (see Fig. 7.4), specifically **network pruning** [538, 81], **attention map visualizations** [539, 540, 541], and **probing classifiers** [542, 543, 544, 545]. We use multiple tools for all experiments since each tool, individually, is known to suffer from pitfalls [546, 547, 548, 379, 549]. Demonstrating our claims consistently hold true across a diverse set of tools improves our conclusions' robustness to pitfalls of a specific tool. Additionally, we propose a methodology called **reverse fine-tuning** (`reFT`), wherein one takes a pretrained model, fine-tunes it on a downstream dataset, and then fine-tunes it again in the "reverse" direction, i.e., on a dataset sampled from the original

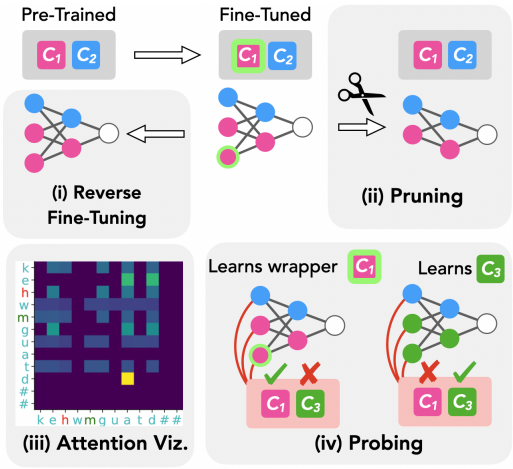


Figure 7.4: **Analysis protocols.** We analyze how fine-tuning affects a pretrained model's capabilities by (i) Reverse Fine-tuning, (ii) network pruning, (iii) attention visualization, and (iv) probing classifiers. We use (ii)—(iv) to show fine-tuning often yields wrapped capabilities. For further evidence, we use (i) and (ii) and find we can "revive" the original capabilities, i.e., the model starts performing well on the pretraining task again. See App. F.4 for precise details.

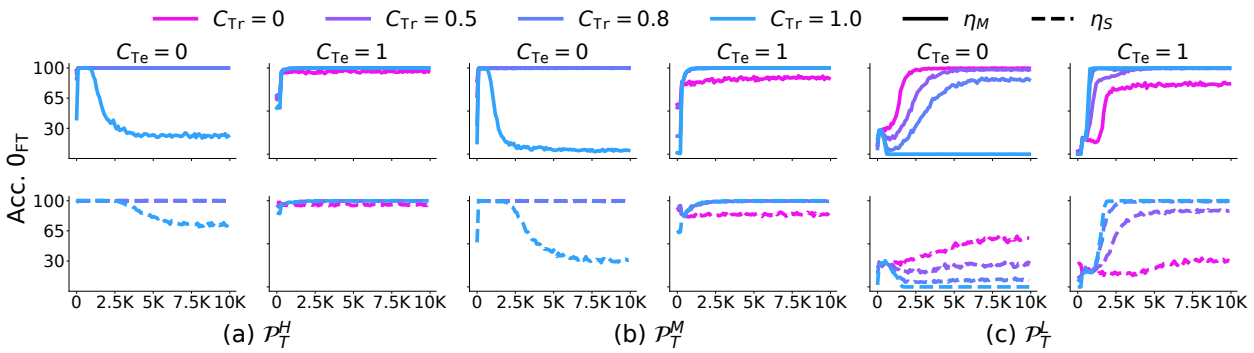


Figure 7.5: **Fine-tuning accuracy w.r.t. number of training iterations.** We vary the probability of sampling the token $\texttt{0}_{\texttt{FT}}$ in the pretraining data and the spurious correlation in the fine-tuning datasets. When the prior is sufficiently high (a, b), we find the model learns to perform well on the downstream task. Meanwhile, if the prior is low (c), the model learns the downstream task only if a high enough learning rate is used and the spurious correlation is imperfect. This indicates the ability to extract information relevant for the downstream task is likely to be exploited during fine-tuning.

pretraining distribution. We argue if the behavior corresponding to a pretraining capability is retrieved in a few steps of `reFT` , fine-tuning did not meaningfully alter said capability *.

We primarily focus on the learned capabilities setup of PCFG counter in the main content, relegating most results on compiled capabilities with Tracr to App. F.7 and other results with PCFGs to App. F.8—findings remain consistent across all settings. In the PCFG counter setup, the model is pretrained, amongst other tasks, to count occurrences of tokens from the set $\{\texttt{0}_{\texttt{PT}}\} = \{\texttt{a}, \texttt{b}, \texttt{c}\}$ in a given string; during fine-tuning, the model is trained to count occurrences of $\texttt{0}_{\texttt{FT}} = \texttt{b}$. Here, the spurious correlation is defined by enforcing count of `b` to be 1 more than that of `a`. The probability a datapoint sampled from the train or test fine-tuning dataset contains a spurious correlation is denoted $C_{\texttt{Tr}}$ and $C_{\texttt{Te}}$, respectively. Here, $C_{\texttt{Tr}} \in \{0.0, 0.5, 0.8, 1.0\}$ and $C_{\texttt{Te}} \in \{0.0, 1.0\}$. We use three sets of sampling probabilities of the task operands in the pretraining data: $\mathcal{P}_{\texttt{T}}^L = (0.999, 0.001, 0.000)$, $\mathcal{P}_{\texttt{T}}^M = (0.9, 0.1, 0.0)$, or $\mathcal{P}_{\texttt{T}}^H = (0.5, 0.3, 0.2)$. These priors indicate a low/medium/high probability of sampling $\texttt{0}_{\texttt{FT}}$. We use the following learning rates (LR) for fine-tuning: $\eta_M = \eta_{\texttt{PT}}/10^1$ and $\eta_S = \eta_{\texttt{PT}}/10^2$, where $\eta_{\texttt{PT}}$ is the *average* pretraining learning rate.

7.5.1 Behavioral assessment of fine-tuning

We first evaluate the model's learning dynamics during fine-tuning (see Fig. 7.5 and Tab. F.4). When the pretraining prior has low probability of sampling the token $\texttt{0}_{\texttt{FT}}$, we see the fine-tuned

*this claim can likely be formalized further using results from causal representation learning [550, 551]

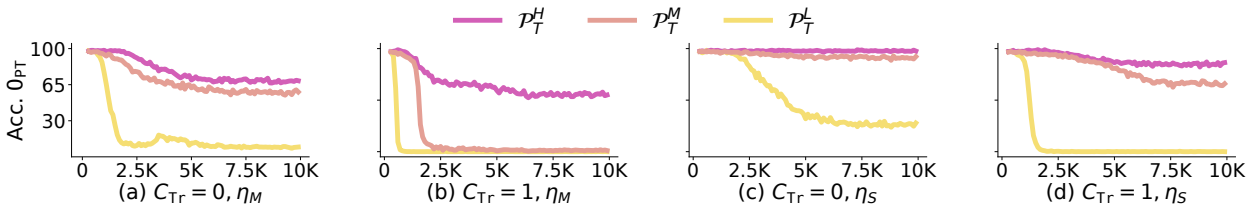


Figure 7.6: **Impact of sampling prior on the pretraining task's accuracy as fine-tuning is performed.** We plot accuracy on the pretraining task w.r.t. fine-tuning iterations. When the sampling prior of the $\mathtt{O}_{\mathtt{FT}}$ is low during pretraining, the pretraining task accuracy quickly plummets, especially if the spurious correlation is high; having a high sampling prior mitigates this behavior. This *indicates* pretraining capabilities are affected the most when they are weakly relevant.

model performs well only when the spurious correlation is present, i.e., $C_{\mathtt{Te}} = 1$. As the sampling probability is increased, however, we observe this behavior significantly changes. In particular, even if the model is fine-tuned for a high value of $C_{\mathtt{Tr}}$, albeit less than 1, it starts to perform well on the test data regardless of the presence of the spurious attribute. Note that the performance is not high to begin with, indicating the ability to count $\mathtt{O}_{\mathtt{FT}}$ was learned during fine-tuning; however, having a sufficiently large sampling probability for the token during pretraining leads the model to avoid the spurious correlation. This indicates a pretraining ability to extract information relevant for the downstream task is likely to be exploited during fine-tuning. This is further corroborated by the results in Fig. 7.6, where we observe that when the spurious correlation is present in the fine-tuning data, accuracy on the pretraining task is affected the most if sampling prior of the target token was low during pretraining. We next analyze these results mechanistically.

7.5.2 Pruning / Probing fine-tuned models indicates learning of wrapped capabilities

Our results above *indicate* the model exploits its weakly relevant capabilities, i.e., the capability that helps exploit any spurious correlation, to solve the downstream task. We hypothesize that, at a mechanistic level, the model exploits the weakly relevant capability by learning a *wrapper* over it. To evaluate this, we analyze the models fine-tuned with a low sampling prior via network pruning and linear probing (see App. F.4 for setup details). Specifically, we prune the fine-tuned models to find the most salient weights for reducing loss on the pretraining task of counting $\mathtt{O}_{\mathtt{PT}}$. If the model learns a wrapper on this capability, the neurons we find should correspond to this wrapper, such that deleting them recovers the capability to count that token. As shown in Fig. 7.7, we find this is indeed the case—in a

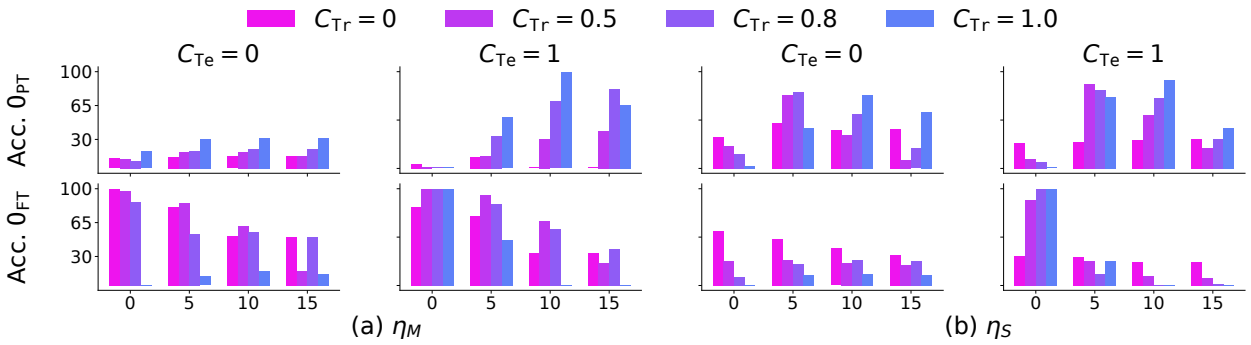


Figure 7.7: **Pruning a few neurons is sufficient to retrieve pretraining task accuracy.** We plot accuracy w.r.t. number of neurons pruned to improve performance on the pretraining task. We see when a small learning rate is used for fine-tuning, the pretraining task's performance improves after just 5–15 neurons are pruned (top), while the fine-tuning task's performance reduces correspondingly (bottom). We argue these neurons serve as a wrapper to minimally alter the weakly relevant pretraining capability and exploit the spurious correlation present in the fine-tuning data.

setup with weak relevance of capabilities, pruning a very small number of neurons is sufficient to revive the ability to perform well on the original task of counting O_{PT}. To assess this further, we train a linear probe on the residual output of every block of the transformer model and determine whether the count of O_{PT} can be accurately computed via the fine-tuned model. As shown in Fig. 7.8, in the presence of spurious correlations, a linear probe can retrieve the count of the token O_{PT}, indicating intermediate outputs relevant to the pretraining capability are still being produced by the fine-tuned model. This observation is particularly evident when a smaller learning rate is used, which is common in practice. Overall, these results show that when a weakly relevant capability is present in the pretrained model, a *wrapper*, i.e., a localized transformation of the pretraining capability, is learned during fine-tuning.

7.5.3 `reFT` enables "revival" of pretraining capabilities

To further corroborate our claims, we use a model fine-tuned to count O_{FT} and reverse fine-tune it to re-learn the ability to count O_{PT}. As a baseline, we also report a protocol called `Scr. + FT`, wherein the model is initialized with parameters pretrained to count O_{FT} and then fine-tuned to count O_{PT}. Note that this baseline and the `reFT` protocol differ in their initialization state: former is initialized with parameters pretrained to count O_{FT}, while latter is initialized with parameters pretrained to count O_{PT} and fine-tuned to count O_{FT}. Results are shown in Fig. 7.9. We see the model starts to perform well on the pretraining task even if a small learning rate is used for `reFT` , i.e., even minimally changing the fine-tuned model's

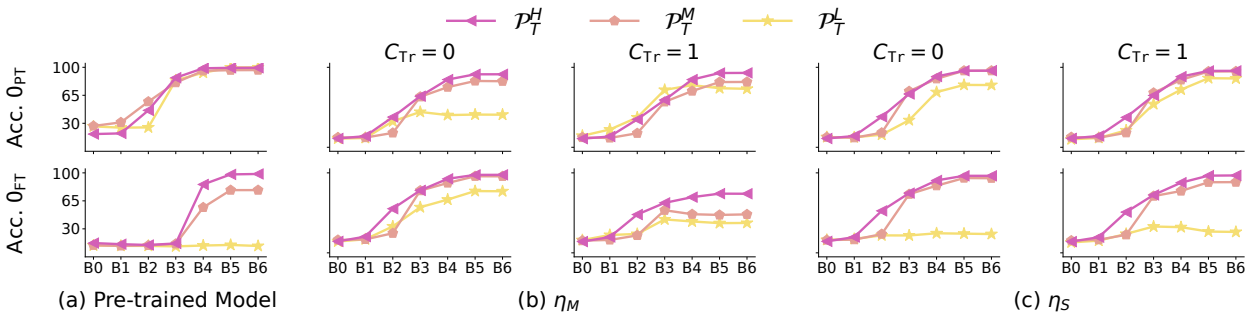


Figure 7.8: **Probing the presence of pretraining (top) and fine-tuning (bottom) capabilities.** We plot probe accuracy versus the index of the block in the Transformer model. $C_{\mathtt{Te}}$ is set to 0. The pretrained model (left) acts as a baseline for the trend of performance through the model's blocks. In most scenarios, we find we can infer the count of $0_{\mathtt{PT}}$ with a similar trend as the pretrained model (left). A drop in performance is observed only when learning rate η_M is used with a weakly relevant capability (low sampling prior). This indicates pretraining capabilities continues to persist upon fine-tuning.

parameters is sufficient to regain the pretraining capability! Further, increasing the sampling prior of $0_{\mathtt{FT}}$ accelerates this behavior. This indicates that when a strongly relevant capability is present, the model essentially amplifies its use, but does not catastrophically affect the pretraining capability itself; meanwhile, with a weakly relevant capability (low sampling prior during pretraining), even though the performance is initially poor on the pretraining task, in relatively few iterations (compared to baseline), the accuracy becomes perfect.

7.5.4 Attention map visualizations further corroborate the wrappers hypothesis

As we noted before, the results discussed above remain consistent across other experimental setups for both Tracr and PCFG models. However, by construction, Tracr yields particularly interpretable attention maps, allowing us to directly visualize the effects of fine-tuning. We thus analyze the attention maps of a Tracr model on the Counter task described in Sec. 7.4. Results are shown in Fig. 7.10. The original Tracr compiled model serves as a baseline and clearly demonstrates that all tokens only attend the pretraining target token, $0_{\mathtt{PT}} = a$. Upon fine-tuning to count $0_{\mathtt{FT}} = b$, we find the model clearly continues to pay attention to $0_{\mathtt{PT}}$ if a small learning rate is used. A larger learning rate is, however, able to alter the model computation, but only if the pretraining capability is not weakly relevant to the fine-tuning task, i.e., when $C_{\mathtt{Tr}} = 0$; otherwise, we again find the model continues to pay attention to the pretraining target.

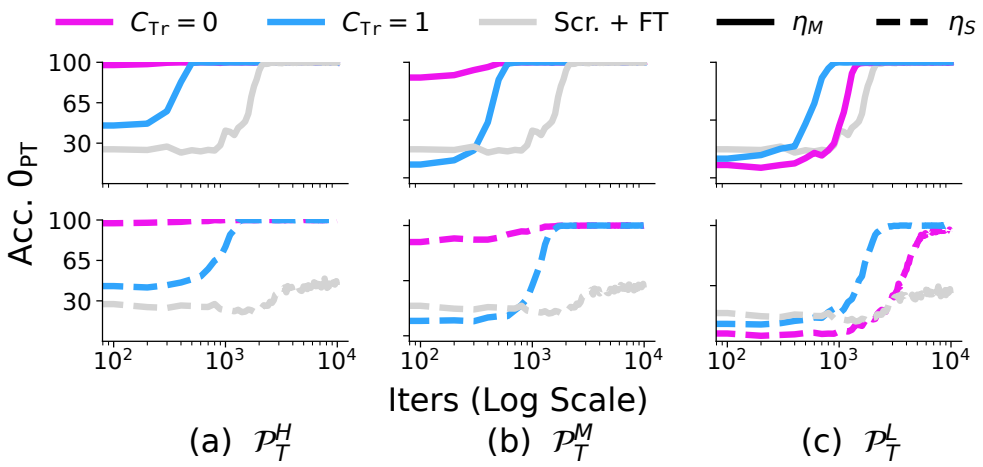


Figure 7.9: **Reverse Fine-Tuning:** We set $C_{\texttt{Te}}$ to be 0 to test if the model performs well regardless of a spurious correlation. Models are fine-tuned for 10K iterations. We observe that when a strongly relevant capability is present (a, b), the model very quickly (0.1–1K iterations) starts to perform well on the task via `reFT` , even if behavior relevant to the capability ceased during pretraining (e.g., when $C_{\texttt{Tr}}$ is 1). Meanwhile, when the model possesses a weakly relevant capability (c), this "revival" is *comparatively* slower (3K iterations). In contrast, the `Scr.` + `FT` baseline only reaches perfect accuracy at 4.5K iterations and when using a larger learning rate, i.e., η_M.

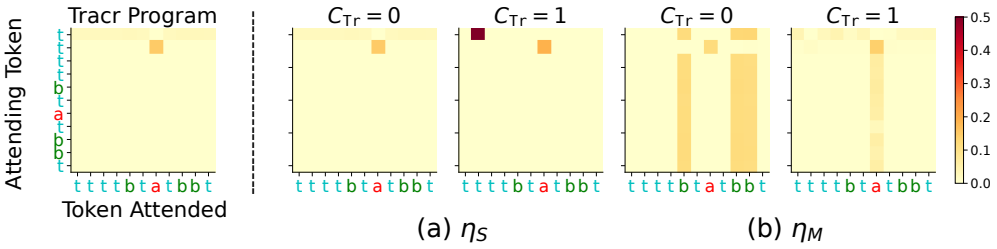


Figure 7.10: **Visualizing attention maps of fine-tuned Tracr models.** Leftmost panel shows the Tracr compiled model's attention map on the counter task. Upon fine-tuning under different spurious correlations, we see the model continues to pay attention to the pretraining target $O_{\text{PT}} = a$. Only when a large enough learning rate and zero spurious correlation is used, there is a change in the attention pattern.

7.5.5 Validating our Hypotheses on TinyStories

To give additional strength to our results, we perform an analysis using more realistic language models trained on the TinyStories-Instruct dataset [14] (see App. F.2.3 for an example). These models are able to follow specific instructions to write coherent English stories over multiple paragraphs. We perform experiments analogous to the `reFT` and probing experiments in the previous sections, but now explicitly focus on whether fine-tuning can *delete* capabilities

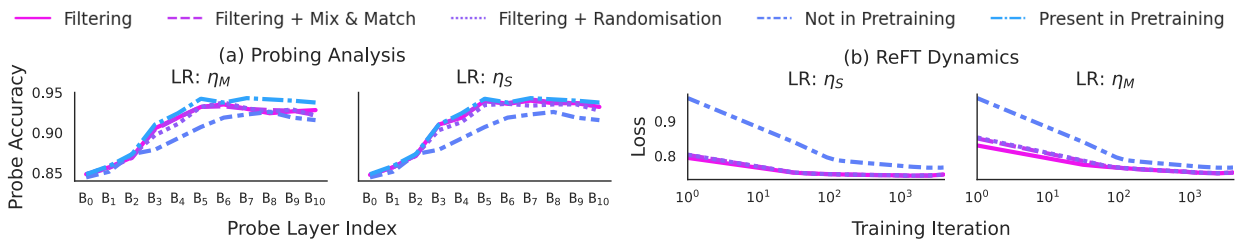


Figure 7.11: **Validation on TinyStories.** Models are trained to produce stories with several features (e.g., `foreshadowing`) and fine-tuned via different protocols to not produce stories with a specific, predefined feature (specifically, `twists`) (see App. F.6 for details). We call this latter feature "forbidden". **Left:** We probe the existence of this feature at intermediate Transformer blocks. Probe accuracy on models pretrained with or without `twist` data (*Present/Not in Pretraining*, respectively) act as upper and lower bounds on the expected accuracy, and are plotted for ease of comparison. Regardless of the fine-tuning protocol (Filtering, Filtering + Randomization, Filtering + Mix & Match), for the lower LR, no protocol removes a meaningful amount of information and a similar but less strong trend holds for the higher LR. **Right:** We plot the loss during reverse fine-tuning (`reFT`) to again produce stories with the forbidden feature. Fine-tuned models' loss goes down very quickly (30–300 iterations) compared to baselines (which never reach the same loss; also see Tab. 7.1). Both these results indicate the capability of identifying the forbidden feature, a necessary capability for story modelling, continues to persist after fine-tuning.

present in pretrained models. Models are pretrained to generate stories with specific story features (such as containing a `twist`, `foreshadowing`, or `bad ending`) and fine-tuned to *not* generate stories with a specific feature (`twist`) (see App. F.6 for details on the protocols).

We probe these models to detect the deleted feature from the intermediate model outputs in Fig. 7.11, where the dynamics of loss during `reFT` on learning to generate stories with the deleted feature are also shown. We also report the percentage of stories with the deleted feature generated by models during `reFT` in Table 7.1, where the generated stories are processed by a fine-tuned GPT-3.5 classifier to predict if the story contains the deleted feature (see App. F.6 for details). Overall, we find that "deleted" capabilities can be easily and sample-efficiently recovered (compared to the baseline), i.e., stories with that feature can be generated again, regardless of the fine-tuning protocol used. These results support our hypotheses that fine-tuning only minimally alters pretrained model capabilities.

7.6 Discussion

In this work, we show that fine-tuning generally alters pretrained model via small, localized changes that only minimally transform their capabilities. We perform our analysis both with

existing mechanistic interpretability tools as well as our proposed `reFT` method. Our results pave the way for future work both understanding how fine-tuning works in more realistic settings with larger models, as well as developing methods beyond fine-tuning that alter pretrained model capabilities more substantially, particularly deleting unsafe capabilities.

In this work, we show that fine-tuning generally alters pretrained model via small, localized changes that only minimally transform their capabilities. We perform our analysis both with existing mechanistic interpretability tools as well as our proposed `reFT` method. Our results pave the way for future work both understanding how fine-tuning works in more realistic settings with larger models, as well as developing methods beyond fine-tuning that alter pretrained model capabilities more substantially, particularly deleting unsafe capabilities. In this work, we show that fine-tuning generally alters pretrained model via small, localized changes that only minimally transform their capabilities. We perform our analysis both with existing mechanistic interpretability tools as well as our proposed `reFT` method. Our results pave the way for future work both understanding how fine-tuning works in more realistic settings with larger models, as well as developing methods beyond fine-tuning that alter pretrained model capabilities more substantially, particularly deleting unsafe capabilities.

Deletion Type	%Twist at Iteration			
	0	30	300	3000
`F` (η_M)	44%	81%	81%	82%
`F` + `R` (η_M)	12%	56%	69%	75%
`F` + `MM` (η_M)	31%	88%	50%	75%
`F` (η_S)	69%	88%	75%	94%
`F` + `R` (η_S)	12%	44%	81%	81%
`F` + `MM` (η_S)	50%	81%	62%	81%
Not in `PT`	12%	31%	44%	81%

Table 7.1: **TinyStories `reFT` Analysis**. We report the percent of generations with a `twist` during reverse fine-tuning for the twist capability. `F`, `R`, and `MM` stand for our three fine-tuning protocols: Filtering, Randomisation and Mix & Match (see App. F.6 for details). Regardless of learning rate and protocol, models relearn to generate stories with `twist` more sample-efficiently than the control model pre-trained on data w/o twists and fine-tuned to generate them (Not in `PT`).

7.7 Follow-Up Work

Since the publishing of the paper associated with this chapter, several papers have come out that either build on or further corroborate the claims of this chapter. For example, performing a similar analysis as ours on natural language data, Lee et al. [552] show that DPO, an algorithm for fine-tuning with a contrastive objective, also does not remove undesirable behaviors from a GPT-2 model. The mechanistic explanation identified in their work is similar in spirit as well: fine-tuning performs localized alterations to the model to elicit the

desired behavior as minimally as possible, consequently failing to show this behavior as soon as an out of distribution input is used.

The results and protocol of `reFT` has seen wide usage to demonstrate limitations of alignment protocols such as RLHF [143, 416] in practical scenarios, such as via fine-tuning GPT-4 models [553, 554, 555, 556]. These works extensively corroborate our claims on practically relevant settings.

An important limitation of all works listed above, and the chapter itself, is that the *downstream* tasks evaluated are fairly simple. For example, often the output of the task is expected to be just one token[†]. In this sense, these papers capture more classical tasks (e.g., sentiment classification) and are limited in their analysis of open-ended generation. Our results on TinyStories do address this limitation; in that setting, the model is expected to perform open-ended story generation during both pretraining and downstream fine-tuning.

Another limitation of the work presented in chapter is that while we demonstrated how fine-tuning affects a model's pretraining mechanisms, we did not sufficiently explore whether these results can explain the lack of robustness of language models under adversarially designed inputs (e.g., jailbreaks [72]); see App. F.5.2 for a few experiments that we did conduct. To address this, in a follow up work [91], we precisely investigate how adversarial inputs exploit the minimalistic changes induced via safety fine-tuning. In brief, we build on our formal grammars setup from this chapter and show that there is a precise signature in a model's intermediate representations that indicates whether the given input is safe or not. This signature turns out to be a rank-1 vector that, given a set of safe versus unsafe samples, can be precisely identified. Importantly, the signature is what triggers the learned wrappers. We also confirm our findings on broader settings by using "open-weights" models, specifically Llama-2 and Llama-3 [140, 21]. Beyond extending the results from this chapter, this follow-up work provides further evidence corroborating its claims in open-ended generation settings.

[†]Note that the pretraining task in this chapter was still language modeling, i.e., it goes beyond production of a single token; only the downstream task was relatively simple.

CHAPTER 8

Conclusion, Limitations, and Future Work

As stated in Chapter 1, the goal of this dissertation was to challenge the use of risk-centric frameworks for regulation of modern, extremely open-ended, and increasingly capable AI systems. Across Parts 1–3, we followed a model-systems approach to make a series of arguments inline with this challenge.

What did we learn?

Since risk regulation requires one preemptively know what capabilities to characterize a system for, if there is sudden learning of capabilities—or, emergence—then such preemptive listing can become difficult (if not entirely infeasible). Part 1 aims to (i) demonstrate there are indeed emergent phenomena in neural network training, and (ii) develop a better understanding of which capabilities show emergent learning. Our results show capabilities that are compositional in nature will exhibit emergent learning curves (Chapter 2). We also find appropriating models of phase transitions from other fields may at times make it feasible to preemptively identify at what scale a capability will emerge (Chapter 3). However, defining such models might be difficult in general.

In Part 2, we discuss hurdles to evaluations of generative models, which are important for thorough risk assessments and hence risk-centric regulation. We first argue that beyond the "unknown unknowns", i.e., capabilities we might not expect the model to exhibit, emergent capabilities also exhibit a component of "unknown knowns". That is, even when we know which capabilities the model might exhibit, defining perfect evaluations to monitor them can be difficult, as models learn capabilities much before input space prompting allows us to elicit them (Chapter 4). In this part, we also discuss how input underspecification renders exhaustive characterizations and establishment of use standards difficult. We propose a specific explanation for this purpose: models perform Bayesian hypothesis selection to identify which capability processes an input, leading to sudden change in their behaviors

under minimal alteration of the input (Chapter 5).

Finally, assuming we have identified a vulnerability through evaluations and would like to fix it, the current state-of-the-art protocols for this purpose are grounded in fine-tuning an objective function that describes user preferences. Motivated by this, in Part 3, we aim to better understand how fine-tuning alters a model. Our results show that at a mechanistic level, fine-tuning is unlikely to delete mechanisms learned by a model during pretraining (Chapter 6). To make these results more precise, in Chapter 7, we characterize what changes fine-tuning makes to models trained on formal grammars and open-ended story generation tasks. Results show that fine-tuning yields minimalistic wrappers over base capabilities that can be easily removed via interventions such as pruning of a few neurons or fine-tuning on the pretraining data for a few steps.

Despite the exhaustively analyzed results listed above, it is worth emphasizing the presented work suffers from limitations possible from use of synthetically defined abstractions of the target problem, instead of focusing on natural domains directly. To address this, we provided several real world examples that yield indicative evidence that our results do generalize to more natural settings. We further discussed follow-up and contemporary work that has built on the contributions of this dissertation to verify our claims in natural settings: e.g., we discussed the use of GPT-4 fine-tuning API to confirm our claims on reverse fine-tuning [553]. Nonetheless, there is much scope left to demonstrate validity of our claims in broader settings. For example, defining a notion of concepts and compositionality to study multiplicative emergence in natural text-to-image settings would make our claims more robust. The primary hurdle here is lack of data with sufficient, precisely annotated concepts that can be used to train models until compositional generalization, hence enabling characterization of multiplicative emergence and growth of competence before performance. While we provide preliminary results on this front using the limited, but naturalistic, CelebA dataset [343], alternative datasets such as Kubric [557] can be worth experimenting with.

Where do we go from here?

The work presented in this dissertation is a step towards identifying challenges in defining regulation frameworks for AI, specifically targeting risk regulation frameworks [63]. The findings also lead to a set of takeaways that are worth highlighting.

- **Towards a decomposition-centric evaluation framework.** For capabilities we believe our model might exhibit at some scale, the presented findings indicate one should consider decomposing the broader capability into relatively simpler, atomic capabilities

and evaluating the model for them (Chapter 2). Monitoring a model's learning of this decomposition can serve as a signal for when the model has learned all relevant components needed to exhibit the capability we are actually interested in. In fact, we highlight a recently released benchmark follows this principle: meant to evaluate a model's autonomous replication abilities, this benchmark provides a decomposition of the overall task and recommends evaluation for capabilities corresponding to these narrower tasks [558].

- **Capability evaluations must go beyond input-output behavior analysis with predefined benchmarks.** Our findings demonstrate alternative protocols, e.g., latent interventions, can elicit model capabilities that naive input-space prompting may be unable to (Chapter 4). If the aim of an evaluation is to judge whether the model possesses a capability, use of such alternative protocols may be warranted. While latent interventions are still more of an art than science [559], research investment in such protocols might enable more rigorous stress testing for the existence of capabilities in a model.

- **Importance of counterfactual evaluations.** We find that fine-tuning can give the *impression* of deleting behaviors from a model; however, when evaluated under systematically defined counterfactuals, results can turn out to be substantially different. When designing evaluations, intentionally defining stylistically restructured formats that preserve the semantic meaning of an input can enable behavioral evaluations that assess whether a model has changed at a mechanistic level (as indicated by Defn. 18). More broadly, it is important that counterfactual evaluations become a common practice.

- **Alternatives to Fine-tuning are needed.** Presented findings and several contemporary works corroborating them show fine-tuning yields brittle changes to the model (Chapters 6 and 7). To address this, we must identify alternative protocols for altering a model's behavior. Promising work by Korbak et al. [560] argues preference data should be used during pretraining to alter a model's behavior. This solution does not give us a posthoc intervention that can fix a vulnerability identified in the model, but may be a worthwhile option when pretraining from scratch is feasible (e.g., in the training run of a new member of the model family). Another alternative is inference-time interventions [561], which advocate for altering a model's generated output by identifying how a model represents a behavior in its latent representation space. If we can identify similar inference-time protocols to influence model generations towards or away a broad spectrum of behaviors, the generations can then be steered towards preferred outputs regardless of what input is fed in. In this sense, inference-time interventions may enable

"worst-case" robustness, compared to fine-tuning protocols that are likely to achieve merely "average-case" robustness by maximizing, on average, an objective function that approximates our preferences.

- **Regulation must get more nuanced.** The presented findings were geared towards critiquing regulation frameworks solely grounded in risk assessment. However, it is worth emphasizing that *for harms we can preemptively define and accurately measure, risk regulation is indeed a decent framework.* The issue is that the framework is too rigid, for it does not adapt to novel harms as they are discovered. To this end, we claim more nuanced regulation frameworks that draw on several well-defined tools, not just a single one, are needed. As a trivial example, Tort law and liability models can be mixed with risk regulation to ensure (i) harms that are well-known can be avoided; (ii) there is a path to relief for harms that occur post deployment; and (iii) companies are incentivized to perform stress testing for capabilities beyond what current risk standards suggest, e.g., a frontier AI company should run evaluations for speculative capabilities they expect may emerge at larger scales and can cause harms, but as of yet have not been seen in state-of-the-art models. Another exciting framework that is worth highlighting is based on insurance liability [562, 563]. The framework is yet to see further development, but can be impactful for more catastrophic risks.