Learning dynamics of emergent in-context learning in transformers

Aaditya K Singh

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Gatsby Computational Neuroscience Unit

University College London

August 24, 2025

Abstract

Transformer neural networks demonstrate a remarkable capacity for in-context learning (ICL). Typically contrasted with in-weights learning (IWL), in-context learning is the ability of a neural network to generalize to a new inputs and tasks at test time, *without* any weight updates. This emergent ability, first seen in large language models (LLMs) trained at internet scale, has led to much of the utility of LLMs in modern society, through chatbots and coding assistants.

Despite its importance, this ability was remarkably poorly understood, likely due to the scale needed to achieve it. In this thesis, I study emergent in-context learning by first identifying a smaller-scale setup where ICL emerges, despite not being explicitly trained for. I find that non-uniform data distributional properties are crucial for ICL emergence.

Even with these properties however, overtraining (as is common with modern LLMs) can lead to ICL fading away. In other words, emergent ICL is a transient phenomenon, which emphasizes the need for a better understanding of dynamics and begs the question: Why does ICL emerge if only to give way to in-weights strategies?

To tackle these dynamics, I developed optogenetics-inspired methods to causally intervene on dynamics. These methods enabled a deeper understanding of the abrupt induction circuit emergence (the key driver of ICL abilities) as the simultaneous learning of three smoothly evolving sub-circuits. Equipped with this better understanding, I found that the transient emergence of ICL arises due to co-operative interactions with asymptotic, largely in-weights strategies, which share two of the three sub-circuits. The transience is caused by competitive interactions:

asymptotic strategies ”crowd out” the earlier ICL.

In sum, these findings indicate that ICL abilities should rather be thought of as existing on a *spectrum* with competitive and cooperative interactions governing their tradeoff through learning.

Impact

This work enhanced our understanding of one of the most critical abilities of modern large language models (LLMs): in-context learning (ICL). This emergent ability is what enables LLMs to adapt to different users and a wide variety of tasks, thus making them such useful assistants. By identifying ICL as a possibly transient phenomenon, this work emphasizes possible dangers of the current scaling paradigm in artificial intelligence: as we scale models further, desirable abilities they currently possess are not guaranteed to persist.

This work starts by identifying key data distributional properties that may lead to emergent ICL. While these properties (e.g., burstiness) are naturally occurring in language, they may or may not be in other domains of interest. Identifying these properties paves the path to future work towards enhanced ICL in domains beyond language, by (potentially artficially) incorporating such data distributional properties.

In understanding the subsequent transience of emergent ICL, I created a series of minimal mathematical models that capture key parts of a complex neural network's learning dynamics. Key substrates, such as tensor products and multiplicatively interacting loss terms, enable flexible modeling of the dynamics of ICL emergence and transience. Such mathematical models may enhance our understanding of the largely inscrutable workings of deep neural networks, especially in the realm of the dynamics of capability emergence and transience. Specifically, while dynamics of deep transformer neural networks lie outside what is analytically tractable, these minimal mathematical models *can* be analyzed mathematically, with insights that transfer to larger scales.

Beyond ICL, the enhanced understanding of dynamical phenomena may extend to other strategies (and their interactions during training) in deep neural networks. For example, the coopetition phenomena identified represents a marked shift in how strategies are thought to trade off through learning: not only do they compete for capacity within a network, but they may cooperate and help each other emerge. Such *backwards hysteresis*, whereby later, asymptotic strategies may influence earlier, transient strategies, complements existing notions of path dependence (earlier points affecting later points).

Beyond the field of artificial intelligence, this work shows the benefits of shared methods across neuroscience and artificial intelligence. The open-sourced artificial optogenetics framework created in this thesis is a key example of how an analysis method from neuroscience (optogenetics) can inspire more rigorous experimentation in artificial intelligence. Specifically, the causal interventions on dynamics enabled by "clamping" can reveal sub-circuits in a way that prior correlational approaches (such as progress measures) do not. As modern AIs approach human-level intelligence, such transfer of methods from neuroscience to AI may enable more rigorous study, and hopefully safer development.

Finally, as a resource to the community, we open-source all code needed for reproducing this thesis at: `https://github.com/aadityasingh/icl-dynamics`.

Dedication

I would like to dedicate my thesis to Felix Hill, my co-advisor through most of my time at the Gatsby Unit, until his passing in December 2024.

Felix was there from the beginning, and really transformed the way I thought about research. My experience in machine learning research before meeting him had been... mixed. Working with Felix was refreshingly different: caring about ideas and good research over papers, and prioritizing people over the projects they worked on. When he agreed to be my co-advisor after my internship at DeepMind ended, I was overjoyed. In the coming months, our weekly meetings were always a source of inspiration, and honestly, his enthusiasm for research was just so infectious.

Our field has truly lost a great light, and I can only hope that some of Felix lives on in the people lucky to be touched by him. For my part, I hope to keep spreading his enthusiasm for research, his brilliant ideas, and his care for those he mentored. Thank you Felix for being my hero; I would not be the researcher I am without you.

Acknowledgements

The shoulders of giants may give us height, but the hands of loved ones give us balance, direction, and the strength to climb higher still.

I'd like to start by thanking my family—Pradeep, Poonam and Kritika—for their constant support throughout my thesis. Whether it be celebrating a good result, or hopping on a last minute flight to help support me in London when I fell ill, they've always been there for me. My dad's work ethic has always served as an inspiration to me, and we'd often keep each other company as we co-worked till 2am for the fun of it during my trips home. My mom's ever-present love, care, and belief in me got me through the hardest times.

I've also been extremely grateful to have many, many supportive friends. Thank you to Joshua Park and Albert Yue for our weekly, virtual board game nights where we'd alternate between cooperative and competitive games (coopetition surely describes our dynamics). Thank you to Jeffery Yu for our yearly Europe trips that were one of my main ways of recharging. Thank you Erica Weng and Cindy Yang for all the amazing music recs that got me through my coding sessions. Thank you to Eric Qian and Elle Yang for the stimulating conversations; both your loves of learning has always been such an inspiration to me. Thank you Yu Yang for library chats and your listening ear. Thank you Kushal Tirumala for our oddly timed calls and your sincerity. Thank you Lovish Madaan for countless late night work sessions (with ping pong breaks, of course) in London. And a very special thank you to Kira Düsterwald who was with me since Day 1. I never expected to form such a close friendship in grad school, but thank you for being such a pillar of support during my time in London.

In London, I was also lucky to be part of a thriving research community at the Gatsby Unit and Sainsbury Wellcome Centre. Thank you to Jai Bhagat, Marta Sartori, Liang Zhou, Clementine Domine, Peter Orbanz, I-Chun Lin and countless others for making my time in the building so special. Thank you to the Saxe Lab (Jirko Rubruck, Verena Klar, Sam Liebana, and ... it's too big to list everyone but I mean everyone!) for making every Thursday such a fun day. A special thank you to my fellow in-context learners (Jin Lee, Yedi Zhang, Sara Dragutinovic, Basile Confavreaux, and Nishil Patel) for making the last few months of the PhD so much fun collaborating.

Speaking of collaboration, this PhD would not have happened if not for countless amazing people across academia and industry. Thank you Ted Moskovitz for being my "buddy-in-the-building" as we did all of our projects together. Thank you Andrew Lampinen for your mentorship during my internship at DeepMind, and all our research chats since—I always find myself thinking about things in a new way after we chat. Thank you DJ Strouse for showing me how much fun pairing can be, the right way to do conferences (feat. Dan Roberts), and [redacted CoT] so much more. A special thank you to Stephanie Chan, who's been there from the beginning of this ICL journey. Your research insights, mentorship, and above all, care for those around you has been such an inspiration for me.

Through my academic journey, I've been lucky to have amazing teachers and mentors who have made me who I am. Most recently, I was lucky to be supervised by Andrew Saxe. I'm so glad I chose to move to London to pursue a PhD with him. I can't think of how anyone could be a better advisor—from our weekly research chats, to always being there when I needed to talk, to being understanding of all my random side projects (including the one that turned into this thesis!). I still don't fully know how he deals with all the context switching, yet still stay so relaxed and provides such insight. I always walk away from our chats feeling more motivated. I've learned so much from him, and am eternally grateful to have been in his lab for the past 4 years.

UCL Research Paper Declaration Form: referencing the doctoral candidate's own published work(s)

Paper 1

1. For a research manuscript that has already been published (if not yet published, please skip to section 2):

   (a) What is the title of the manuscript? Data Distributional Properties Drive Emergent In-Context Learning in Transformers

   (b) Please include a link to or DOI for the work: `https://proceedings.neurips.cc/paper_files/paper/2022/hash/77c6ccacfd9962e2307fc64680fc5ace-Abstract-Conference.html`

   (c) Where was the work published? NeurIPS 2022

   (d) Who published the work? Curran Associates, Inc.

   (e) When was the work published? 2022

   (f) List the manuscript's authors in the order they appear on the publication: Stephanie Chan, Adam Santoro, Andrew Lampinen, Jane Wang, **Aaditya Singh**, Pierre Richemond, James L. McClelland, Felix Hill

   (g) Was the work peer reviewed? Yes

   (h) Have you retained the copyright? Yes

(i) Was an earlier form of the manuscript uploaded to a preprint server (e.g., medRxiv)? **Yes.**

If 'Yes', please give a link or DOI: `https://arxiv.org/abs/2205.05055`

If 'No', please seek permission from the relevant publisher and check the box below:

☐ I acknowledge permission of the publisher named under 1(d) to include in this thesis portions of the publication named as included in 1(c).

2. For a research manuscript prepared for publication but that has not yet been published (if already published, please skip to section 3):

   (a) What is the current title of the manuscript?

   (b) Has the manuscript been uploaded to a preprint server (e.g., medRxiv)?

   (c) Where is the work intended to be published?

   (d) List the manuscript's authors in the intended authorship order:

   (e) Stage of publication:

3. For multi-authored work, please give a statement of contribution covering all authors (if single-author, please skip to section 4):

   - Stephanie Chan: Led the work, from conception to conducting experiments and overall driving progress, to writing.
   - Adam Santoro, Andrew Lampinen, Jane Wang, **Aaditya Singh**, Pierre Richemond, James L. McClelland: Collaborated closely from start to finish of the project, suggesting experiment directions, providing feedback, and helping write.
   - Felix Hill: Supervised the project start to finish.

4. In which chapter(s) of your thesis can this material be found? Primarily Introduction, Chapter 2, and Conclusion, though elements from this first work carried into subsequent papers (and corresponding chapters).

E-Signatures confirming that the information above is accurate (This form should be co-signed by the supervisor/senior author unless this is not appropriate, e.g., if the paper was a single-author work):

Candidate: [illegible] Date: Aug. 24, 2025

Supervisor: Andrew Saxe Date: Aug. 24, 2025

Paper 2

1. For a research manuscript that has already been published (if not yet published, please skip to section 2):

   (a) What is the title of the manuscript? The Transient Nature of Emergent In-Context Learning in Transformers

   (b) Please include a link to or DOI for the work: `https://proceedings.neurips.cc/paper_files/paper/2023/hash/58692a1701314e09cbd7a5f5f3871cc9-Abstract-Conference.html`

   (c) Where was the work published? NeurIPS 2023

   (d) Who published the work? Curran Associates, Inc.

   (e) When was the work published? 2023

   (f) List the manuscript's authors in the order they appear on the publication: **Aaditya Singh**, Stephanie Chan, Ted Moskovitz, Erin Grant, Andrew Saxe, Felix Hill

   (g) Was the work peer reviewed? Yes

   (h) Have you retained the copyright? Yes

   (i) Was an earlier form of the manuscript uploaded to a preprint server (e.g., medRxiv)? **Yes.**

   If 'Yes', please give a link or DOI: `https://arxiv.org/abs/2311.08360`

   If 'No', please seek permission from the relevant publisher and check the box below:

☐ I acknowledge permission of the publisher named under 1(d) to include in this thesis portions of the publication named as included in 1(c).

2. For a research manuscript prepared for publication but that has not yet been published (if already published, please skip to section 3):

   (a) What is the current title of the manuscript?

   (b) Has the manuscript been uploaded to a preprint server (e.g., medRxiv)?

   (c) Where is the work intended to be published?

   (d) List the manuscript's authors in the intended authorship order:

   (e) Stage of publication:

3. For multi-authored work, please give a statement of contribution covering all authors (if single-author, please skip to section 4):

   - **Aaditya Singh**: Co-led the work, from the original observation and motivation, to planning and executing experiments, and writing the paper.
   - Stephanie Chan: Co-led the work, providing the initial codebase (from our previous work) and assisting with running experiments, as well as writing.
   - Ted Moskovitz: Active collaborator for the work, assisting with some coding and experiments.
   - Erin Grant: Collaborator, earlier in ideation of the work, contributing to some initial code.
   - Andrew Saxe, Felix Hill: Co-supervised the work, from start to finish.

4. In which chapter(s) of your thesis can this material be found? Primarily Introduction, Chapter 3, and Conclusion, though elements from this work carried into subsequent papers (and corresponding chapters).

E-Signatures confirming that the information above is accurate (This form should be co-signed by the supervisor/senior author unless this is not appropriate, e.g., if the paper was a single-author work):

Candidate: [illegible] Date: Aug. 24, 2025

Supervisor: Andrew Saxe Date: Aug. 24, 2025

Paper 3

1. For a research manuscript that has already been published (if not yet published, please skip to section 2):

   (a) What is the title of the manuscript? What needs to go right for an induction head? A mechanistic study of in-context learning circuits and their formation

   (b) Please include a link to or DOI for the work: `https://dl.acm.org/doi/10.5555/3692070.3693925`

   (c) Where was the work published? ICML 2024

   (d) Who published the work? JMLR.org

   (e) When was the work published? 2024

   (f) List the manuscript's authors in the order they appear on the publication: Aaditya K. Singh, Ted Moskovitz, Felix Hill, Stephanie C.Y. Chan, Andrew M. Saxe

   (g) Was the work peer reviewed? Yes

   (h) Have you retained the copyright? Yes

   (i) Was an earlier form of the manuscript uploaded to a preprint server (e.g., medRxiv)? **Yes.**
   If 'Yes', please give a link or DOI: `https://arxiv.org/abs/2404.07129`

   If 'No', please seek permission from the relevant publisher and check the box below:

☐ I acknowledge permission of the publisher named under 1(d) to include in this thesis portions of the publication named as included in 1(c).

2. For a research manuscript prepared for publication but that has not yet been published (if already published, please skip to section 3):

   (a) What is the current title of the manuscript?

   (b) Has the manuscript been uploaded to a preprint server (e.g., medRxiv)?

   (c) Where is the work intended to be published?

   (d) List the manuscript's authors in the intended authorship order:

   (e) Stage of publication:

3. For multi-authored work, please give a statement of contribution covering all authors (if single-author, please skip to section 4):

   - **Aaditya Singh**: Led the work, from start to finish: ideas, analysis, code, methodology, and writing.
   - Ted Moskovitz, Felix Hill: Collaborators during early stages of the work, contributing to some code and providing feedback and encouragement.
   - Stephanie Chan, Andrew Saxe: Co-supervised the work, from start to finish.

4. In which chapter(s) of your thesis can this material be found? Primarily Introduction, Chapter 4, and Conclusion, though elements from this first work carried into subsequent papers (and corresponding chapters).

E-Signatures confirming that the information above is accurate (This form should be co-signed by the supervisor/senior author unless this is not appropriate, e.g., if the paper was a single-author work):

Candidate: [illegible] Date: Aug. 24, 2025

Supervisor: Andrew Saxe Date: Aug. 24, 2025

Paper 4

1. For a research manuscript that has already been published (if not yet published, please skip to section 2):

   (a) What is the title of the manuscript? Strategy Coopetition Explains the Emergence and Transience of In-Context Learning

   (b) Please include a link to or DOI for the work: https://arxiv.org/abs/2503.05631

   (c) Where was the work published? ICML 2025

   (d) Who published the work? JMLR.org

   (e) When was the work published? 2025

   (f) List the manuscript's authors in the order they appear on the publication: Aaditya K. Singh, Ted Moskovitz, Sara Dragutinovic, Felix Hill, Stephanie C.Y. Chan, Andrew M. Saxe

   (g) Was the work peer reviewed? Yes

   (h) Have you retained the copyright? Yes

   (i) Was an earlier form of the manuscript uploaded to a preprint server (e.g., medRxiv)? **Yes.**

   If 'Yes', please give a link or DOI: `https://arxiv.org/abs/2503.05631`

   If 'No', please seek permission from the relevant publisher and check the box below:

   ☐ I acknowledge permission of the publisher named under 1(d) to include in this thesis portions of the publication named as included in 1(c).

2. For a research manuscript prepared for publication but that has not yet been published (if already published, please skip to section 3):

   (a) What is the current title of the manuscript?

   (b) Has the manuscript been uploaded to a preprint server (e.g., medRxiv)?

(c) Where is the work intended to be published?

(d) List the manuscript's authors in the intended authorship order:

(e) Stage of publication:

3. For multi-authored work, please give a statement of contribution covering all authors (if single-author, please skip to section 4):

    - **Aaditya Singh**: Led the work, from start to finish: ideas, analysis, code, methodology, and writing.
    - Ted Moskovitz, Felix Hill: Collaborators during early stages of the work, contributing to some code and providing feedback and encouragement.
    - Sara Dragutinovic: Collaborated during later stages of the work, contributing some come and providing feedback on early drafts.
    - Stephanie Chan, Andrew Saxe: Co-supervised the work, from start to finish.

4. In which chapter(s) of your thesis can this material be found? Primarily Introduction, Chapter 5, and Conclusion.

E-Signatures confirming that the information above is accurate (This form should be co-signed by the supervisor/senior author unless this is not appropriate, e.g., if the paper was a single-author work):

Candidate: [illegible] Date: Aug. 24, 2025

Supervisor: Andrew Saxe Date: Aug. 24, 2025

Paper 5

1. For a research manuscript that has already been published (if not yet published, please skip to section 2):

    (a) What is the title of the manuscript?

    (b) Please include a link to or DOI for the work:

(c) Where was the work published?

(d) Who published the work?

(e) When was the work published?

(f) List the manuscript's authors in the order they appear on the publication: Andrew Kyle Lampinen, Stephanie C. Y. Chan, Aaditya K. Singh, Murray Shanahan

(g) Was the work peer reviewed?

(h) Have you retained the copyright?

(i) Was an earlier form of the manuscript uploaded to a preprint server (e.g., medRxiv)?

If 'Yes', please give a link or DOI:

If 'No', please seek permission from the relevant publisher and check the box below:

☐ I acknowledge permission of the publisher named under 1(d) to include in this thesis portions of the publication named as included in 1(c).

2. For a research manuscript prepared for publication but that has not yet been published (if already published, please skip to section 3):

(a) What is the current title of the manuscript? The broader spectrum of in-context learning

(b) Has the manuscript been uploaded to a preprint server (e.g., medRxiv)? Yes

(c) Where is the work intended to be published? TMLR

(d) List the manuscript's authors in the intended authorship order: Andrew Kyle Lampinen, Stephanie C. Y. Chan, Aaditya K. Singh, Murray Shanahan

(e) Stage of publication: Under review

3. For multi-authored work, please give a statement of contribution covering all authors (if single-author, please skip to section 4):

    - Andrew Lampinen: Led the work (position paper), from conception to writing.
    - Stephanie Chan, **Aaditya Singh**: Collaborated closely throughout in discussions and in providing feedback.
    - Murray Shanahan: Advised the project.

4. In which chapter(s) of your thesis can this material be found? Primarily Introduction and Conclusion, though elements may be present throughout.

E-Signatures confirming that the information above is accurate (This form should be co-signed by the supervisor/senior author unless this is not appropriate, e.g., if the paper was a single-author work):

Candidate: [illegible]	Date: Aug. 24, 2025
Supervisor: Andrew Saxe	Date: Aug. 24, 2025

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past few years, deep neural networks have gone from being a topic of research, to being something that hundreds of millions of people go to everyday with their questions (OpenAI, 2024). The defining aspect of these systems is their ability to learn. For models that make predictions based on an input sequence, the ability to learn can manifest in two ways: longer-term learning from previous sequences, as well as short-term learning from existing structure within an input sequence. For example, as you read this thesis, you understand the meanings of most individual words from your prior experience reading English text (past sequences), but for novel words (e.g., "coopetition"), you will use the definitions provided in this document (this input sequence). This short-term learning ability is commonly referred to as *in-context learning* (ICL), and its presence in modern large language models (LLMs) is one of the main reasons for their general usefulness.

The remarkable finding that motivated this work is that ICL can actually be *emergent* (Brown et al., 2020). Namely, to extract the most value during longer-term learning, it may be in the best interest of a human or model to "learn-to-learn." For example, instead of memorizing the definition of every word in the English language from prior experience, a human learns to learn from definitions provided within a document.

In this thesis, we study why ICL may emerge when training to predict tokens in sequences of data.

1.1 Background

1.1.1 Supervised learning

One of the first large-scale successes of machine learning was that of AlexNet (Krizhevsky et al., 2012), an image classification system that outperformed existing approaches by a large margin by learning features from data, rather than relying on human-engineered features. These features were learned through supervised learning using a deep neural network.

Supervised learning refers to learning the parameters θ of a function $f_\theta(x)$ given some labeled data $\mathscr{D} = \{(x_i, y_i)\}_{i=1}^N$. Specifically, supervised learning operates by minimizing a loss function $\mathbb{E}_{x\sim\mathscr{D}}[\mathscr{L}(y, f_\theta(x))]$ with respect to the parameters θ. Deep neural networks refer to a specific family of functions and parametrizations that are highly expressive (Hornik et al., 1989) yet still amenable to gradient-based learning (Rumelhart et al., 1988). In practice, these loss functions are optimized using variants of stochastic gradient descent (SGD) by sampling from $\mathscr{D}$, calculating $\frac{d\mathscr{L}}{d\theta}$, and performing a gradient step in that direction.

Notably, supervised learning is *slow*, often requiring thousands or millions of gradient steps before $f_\theta(x)$ does something reasonable. However, the impressive aspect of supervised learning is that it scales remarkably well (Kaplan et al., 2020; Hoffmann et al., 2022): namely, when using either a larger network (i.e., a higher dimensional θ), or training for longer, the loss keeps decreasing. From the first demonstrations with Alexnet (Krizhevsky et al., 2012), to modern large language models (OpenAI, 2023), this scaling paradigm has driven progress for over a decade.

1.1.2 In-context learning

As traditional supervised learning is slow, researchers have sought to instill deep neural networks with a capacity for quick adaptation. This field is broadly known as "meta-learning."

Meta-learning involves an inner and outer learning process. Outer learning typically acts on sequences of inputs, while inner learning acts within a sequence.

The inner loop is incentivized to learn from the sequence, while the outer loop is incentivized to enhance the ability of the inner loop to learn: namely, learn to learn.

While some approaches to meta-learning involve rapid weight updates as the inner loop (Finn et al., 2017), here we focus on memory-based meta-learning (Santoro et al., 2016; Wang et al., 2016; Ortega et al., 2019), in which the inner learning process occurs within the memory (or activations) of the deep neural network. Typically, such meta-learning involves artificially constructing sequences that force the outer loop to learn to learn—for example, Santoro et al. (2016) construct sequences of images with randomized labels per sequence (but consistent within a sequence), forcing the outer loop (which updates the weights) to learn-to-learn how to map images to labels.

Recently, Brown et al. (2020) found that such memory-based meta-learning—which they termed *in-context learning* (ICL), a term we will use henceforth—can emerge without constructing an explicit inner-outer loop setup. Specifically, Brown et al. (2020) used supervised learning to train transformer sequence models to predict the next token. Each input to the model is a sequence of L tokens, and can be thought of as $L-1$ different supervised examples (predicting the k-th token using the previous $k-1$ tokens, for $2 \leq k \leq L$). Somehow, after training on an internet-scale amount of text, large scale transformer models learn to learn: they can solve unseen language tasks when presented with a few, concatenated input-output examples earlier in the sequence (e.g., classifying the sentiment of various sentences as positive or negative, except having to output “bar” instead of positive and “foo” instead of negative (Wei et al., 2023)). This mysterious emergence is one of the key motivations for our work here.

Importantly, these ICL abilities in LLMs emerge in conjunction with more classical supervised learning (e.g., being able to complete the phrase “The capital of France is ____” with “Paris.”), which we will refer to as *in-weights learning* (IWL), to represent that this information is stored in the weights of the model, rather than inferred from context.

1.1.3 Transformer neural networks

As emergent ICL was first observed in transformer language models, we largely restrict our study here to transformer neural networks. First introduced by Vaswani et al. (2017), transformers neural networks consist of stacked, alternating layers of self-attention (SA) blocks and multilayer perceptron (MLP) blocks, with residual connections throughout. Here, we will sketch out the basic forward pass of a transformer operating on an input sequence of L (discrete) tokens.

First, each token is embedded through a (learned) embedding matrix: $e_i = W_E t_i$, where t_i is the one hot vector for the token at position i.

Then, these embeddings are fed through a series of stacked, alternating residual layers:

$$\begin{aligned} H_{l,SA} &= H_{l-1,MLP} + SA(H_{l-1,MLP}) \\ H_{l,MLP} &= H_{l,SA} + MLP(H_{l,SA}), \end{aligned}$$

where $H_{0,MLP}$ is defined as the $d \times L$ matrix of input embeddings for all L tokens being fed through the model. The parameters for all layers will be different, below we use re-use some letters for simplicity of exposition.

Before diving deeper into the specifics of implementation, we note that the residual connections enable a view of the forward pass of a transformer as reading out from and writing into a *residual stream* (Elhage et al., 2021) per token. We will see how this framing can be useful as we continue.

SA blocks: A (modern) multi-headed self-attention block $SA_\theta(X)$ performs the following operation:

$$\begin{aligned} Z &= \text{LayerNorm}_{w,b}(X) \\ SA_\theta(X) &= \sum_{h=1}^{H} \underbrace{W_O^h W_V^h Z}_{\text{OV circuit}} \cdot \underbrace{\text{softmax}\left(\frac{Z^\top (W_K^h)^\top W_Q^h Z}{\sqrt{d}}\right)}_{\text{QK circuit}}, \end{aligned}$$

where $\theta = \{w, b, W_Q, W_K, W_V, W_O\}$ and the softmax applies over columns. The LayerNorm (Ba et al., 2016) is applied independently per token, with two d dimensional parameters: w, b. There are 4 $d \times d$ weight matrices: W_Q, W_K, W_V, W_O. The self-attention block is said to have H heads (and thus called "multi-headed" when $H > 1$), with $W_\cdot^h$ defined as the $\frac{d}{H} \times d$ matrix comprised of rows $(h-1)\frac{d}{H}$ to $h\frac{d}{H}$. Typically, we choose H to be a divisor of d to make sure all heads have the same integer dimension.

Note that the above form differs slightly from the original formulation by Vaswani et al. (2017). Specifically, we use "pre-LayerNorm" (as opposed to post-), as is common in modern LLMs (Touvron et al., 2023a). We also write the equations with a projection matrix for each head (W_O^h), following Elhage et al. (2021), as opposed to the more classical form where W_O applies to a concatenated vector across heads (these are equivalent).

Before continuing to the MLP layers, we make a few comments on the form of multi-headed self-attention. First, the output of a layer consists of additive contributions from individual heads—thus, each head can be viewed as independently updating the residual stream. Second, each head can be viewed as performing two operations:

- QK circuit: Determines attention from a given token to other tokens. Outputs an $L \times L$ matrix of attention scores, where each column sums to 1.
- OV circuit: Transforms each token's representations "individually"

The QK circuit determines how information can transfer from other tokens' residual stream to this token's residual stream. The OV circuit determines what information is transferred (via $W_O W_V$). Finally, we note that in practice we use decoder-only (also known as "causal") transformers, where we mask attention scores so that tokens can only attend to previous tokens.

MLP blocks: The multi-layer perceptron blocks operate on a per token basis. Namely, $MLP_\theta(X) = \text{concat}[MLP_\theta(x_i)]$, where x_i are the columns of X. The forward pass $MLP_\theta(x_i)$ can be written as:

$$z_i = \text{LayerNorm}_{w,b}(x_i)$$
$$MLP_\theta(x_i) = W_2\text{act}(W_1 z_i + b_1) + b_2,$$

where $\theta = \{w, b, W_1, b_1, W_2, b_2\}$. Again, there is LayerNorm with two d dimensional parameters: w, b. The weight matrices are W_1, W_2, with dimensions $d \times 4d$ and $4d \times d$, respectively.[1] The bias vectors b_1, b_2 have dimension $4d$ and d, respectively. While the activation function *act* varies from model to model (Shazeer, 2020), we use GeLU (Hendrycks and Gimpel, 2016), following the original transformer (Vaswani et al., 2017). Note that, as the MLP applies per token, the same weights are used on every token individually. MLP layers can be viewed as enriching a token's residual stream, without transferring information from other tokens (since the MLP applies per token).

After passing through the stacked, alternating self-attention and MLP layers, the final residual is passed through an unembedding layer to output (per token). For discrete tokens, this has the form $W_U\text{LayerNorm}(x_i)$, where x_i are the columns of $H_{L,MLP}$. These output "logits" are typically trained using a cross entropy loss function to match the correct one-hot token at a given position.

Positional encodings: A key aspect of transformers is their use as *sequence* models. However, in our exposition so far, much of the computation of the transformer is agnostic to sequence order (with the exception of the causal attention mask). To impart positional information to the model, pracitioners use *positional encodings*. The original positional encodings (Vaswani et al., 2017) were absolute and additive: An index-specific positional vector was added to the input embedding for each token. These positional encodings could be a fixed (vector) function of the index ("sinusoidal positional embeddings") or learned (as additional parameters of the model). Since then, a variety of positional encoding schemes have cropped up, including relative additive and relative multiplicative encodings (Su et al., 2021). While po-

[1] The specific choice of 4 here is known as the expansion factor. Modern language models commonly use 4 (Touvron et al., 2023a).

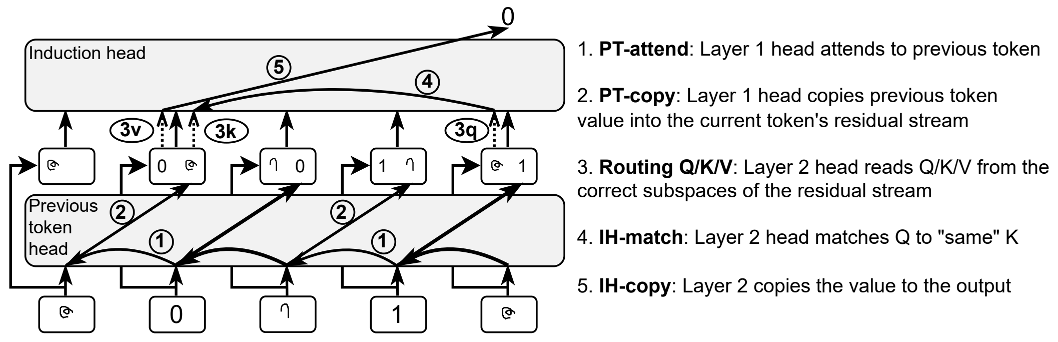

0
Induction head
5
4
3v
3k
3q
Previous token head
2
1
0
1
1. PT-attend: Layer 1 head attends to previous token
2. PT-copy: Layer 1 head copies previous token value into the current token's residual stream
3. Routing Q/K/V: Layer 2 head reads Q/K/V from the correct subspaces of the residual stream
4. IH-match: Layer 2 head matches Q to "same" K
5. IH-copy: Layer 2 copies the value to the output

Figure 1.1: Schematic of an induction circuit, adapted from Singh et al. (2024), with the 5 key computations performed annotated across the two layers. Example sequence is of the form A 0 B 1 A, where the network should output 0 by in-context-learning the mapping from characters to labels.

sitional encodings are not a focus of our work, we experiment with some effects in later sections. Modern transformers typically use RoPE (relative multiplicative embeddings), following Su et al. (2021).

1.1.4 ICL in transformers: Induction Circuits

Equipped with the basic architecture of a transformer, a natural question arises: what set of weights gives rise to ICL behaviors? Olsson et al. (2022) find that a remarkably simple 2-layer circuit seems to explain much ICL behavior: the induction circuit.

An induction circuit is a two-layer circuit (Figure 1.1) comprised of a "previous token head" in an earlier layer, followed by the eponymous "induction head". Previous token heads are attention heads responsible for attending to the previous token (Step 1) and copying it into the attending token's residual stream (Step 2). Induction heads then perform a match-and-copy operation, looking for a match (Step 4) between a query derived from the current token and key derived from the output of the previous token head (Step 3). Induction heads then copy forward the value of the token being attended to from the residual stream (Step 5) as the output.

Olsson et al. (2022) found that induction circuits emerge in a sharp drop in the loss (which we will term a "phase change") coinciding with a model's ability to do ICL. Paired with causal ablations, this evidence strongly points to induction circuits being the source of many in-context learning abilities. Presumably, neither previous token heads nor induction heads on their own are useful for minimizing loss, yet their interaction in an induction circuit yields a surprisingly useful and general ability: ICL.

1.2 Towards understanding *why* ICL emerges

Prior work (Brown et al., 2020) has led to the remarkable finding that ICL emerges when training to predict the next token on internet-scale text (Section 1.1.2). We refer to this ability as *emergent* since, in contrast to prior work in meta-learning (Santoro et al., 2016), no explicit outer loop was constructed that would "necessitate" in-context learning. Olsson et al. (2022) took this finding and made great strides towards the "what" of ICL emergence: Induction circuits emerge in a phase change, and may be responsible for ICL behaviors in transformers. However, a key question remains: Why does ICL emerge when training on data that does not explicitly require it?

We start to answer this question by noticing that classical supervised learning datasets typically assumes various notions of uniformity (e.g., in Imagenet classification each image). However, the next token prediction task instead operates on natural language, a highly non-uniform data distribution. For example, the marginal distribution of words is known to follow the skewed Zipfian (power law) distribution (Zipf, 1949; Piantadosi, 2014; Smith et al., 2018), with rare words appearing in bursts (Sarkar et al., 2005; Alvarez-Lacalle et al., 2006; Neuts, 2007; Altmann et al., 2009; Serrano et al., 2009; Lambiotte et al., 2013). Could these data distributional properties be driving the emergence of in-context learning?

We tackle this question by creating synthetic data setups (Chapter 2) where we can carefully ablate or add back in each of these properites of natural language. We find a host of data distributional properties, inspired by those of natural language, that are necessary for ICL emerge (Chan et al., 2022a): Namely, ablating them reverts transformers to learning IWL strategies to solve the task. Specifically, based on the set of data properties, we observe 3 behavioral regimes:

1. Transformer learns IWL strategies.

2. Transformer learns ICL strategies briefly, then transitions to IWL strategies.

3. Transformer learns ICL strategies.

At first glance, it would seem that the answer to *why* ICL emerges (case 3) is simply that certain data properties incentivize it. Namely, transformers employing ICL strategies achieve better (asymptotic) loss on data with these specific distributional properties.

While that is one hypothesis, there is an important alternate hypothesis that explains the observations: What if these three cases are actually all the *same* behavior? ICL emerges and then gives way to IWL (case 2 above), with data properties modulating the *timescale* (i.e., in case 3 the timescale is just longer) rather than the regime? In some cases, IWL is quick to show up, so we do not see any ICL (case 1), and in others IWL is very slow to show up, making it seem that ICL is persistent (case 3 above). We refer to this as the "transient ICL" hypothesis.

Such hypotheses became especially important to consider, given the emerging practice of *overtraining* LLMs (Touvron et al., 2023a), concurrently to our first work. Overtraining refers to training smaller models for longer and longer, as performance continues to increase (i.e., next token prediction loss continues to decrease). If the transient ICL hypothesis were true, it would indicate an especially pernicious danger: training LLMs for longer and longer may lead to critical ICL abilities fading away.

We test the "transient ICL" hypothesis in our synthetic data setup, and remarkably, find evidence supporting it (Chapter 3) across a range of settings, an original finding (Singh et al., 2023) that has since been reproduced in a variety of settings by other groups (Anand et al., 2024; He et al., 2024; Park et al., 2024; Nguyen and Reddy, 2024). This transient nature of ICL motivates a *dynamical* understanding of ICL, focusing on *how* it emerges and fades away.

1.3 Towards understanding *how* ICL emerges

After uncovering ICL transience, we first sought to understand why ICL fades away, given that it is a useful strategy to solve the task. Our initial evidence, in large scale setups, suggests that competitive interactions with IWL drive ICL transience (Section 3.6), a finding that later work has followed up on and added credence to (Park et al., 2024; Nguyen and Reddy, 2024). To understand the interaction dynamics conclusively however, we realized we first needed to build a toolkit for causal studies of dynamics.

Inspired by the foundational work of Olsson et al. (2022) in isolating circuits responsible for ICL behaviors (Section 1.1.4), we decided to take a mechanistic approach. Prior work in mechanistic intepretability has often shied away from dynamics due to added complexity of the time axis (Nanda, 2023a), with methods such as *progress measures* only recently being popularized (Nanda et al., 2023) and used for ICL (Reddy, 2023). That said, methods such as progress measures only offer a *correlational* view on dynamics, as they track how various quantities evolve through training but do not intervene directly on network learning. Analogously, older mechanistic interpretability works use linear probes to read out activations correlated with network behavior, yet these methods have been shown to lead to incorrect conclusions, when tested against more rigorous *causal* ablation methods (Morcos et al., 2018; Belinkov, 2021; Hayne et al., 2022).

We introduce *clamping* as a method to causally intervene on dynamics. We first establish its usefulness in understanding the emergence of ICL: Namely, elucidating the underlying interactions that give rise to the stereotyped phase change that is indicative of induction circuit emergence (Singh et al., 2024). Specifically, we find that 3 interacting sub-circuits all need to "go right" at the same time, which is what leads to the phase change (Section 4.4).

We then apply *clamping* to tackling the original question of why ICL emerges, if only to give way to IWL-based strategies asymptotically. From our first work, it is clear that data properties play a role in incentivizing emergence, yet not in the sense of making ICL asymptotically preferred. We find a surprising strategy, context-constrained in-weights learning (CIWL), is actually asymptotically preferred on bursty data. The earlier emergence of ICL is driven by *cooperative* interactions with this CIWL strategy (as shown by clamping analyses), before eventual *compet-*

itive interactions "crowd out" ICL, leading to its transience. This dynamical understanding, in combination with the data properties found earlier, finally answers the question of why ICL emerges (transiently), even when training on data that does not explicitly require it.

1.4 Implications

Our work emphasizes that some of the most critical abilities of modern neural networks may in fact be *dynamical phenomena*. Alongside this discovery, we pioneer methods for studying such dynamical phenomena: *clamping* (for mechanistic investigations), as well as a minimal mathematical modeling toolkit that reproduces key dynamics. While the dynamics we study are in the setup of emergent in-context learning, the phenomena themselves (sudden emergence through phase changes, transience, cooperative and competitive interactions) are quite general, and we hope the understanding thus extends to other capabilities.

In the modern scaling paradigm, the two main axes are training a larger model or training for longer (or more recently, on longer context data, for the new paradigm of reasoning models (OpenAI, 2024c; DeepSeek-AI et al., 2025)). Our work sheds light on the types of dynamics that may influence the appearance and disappearance of critical strategies in the case of training for longer. As neural network models become increasingly powerful, we hope this understanding will also help the field of AI safety in understanding how abilities may emerge (or disappear) in future larger scale systems through the course of training.

Chapter 2

Data distributional properties drive emergent in-context learning in transformers

This chapter is based on:

> Stephanie Chan, Adam Santoro, Andrew Lampinen, Jane Wang, **Aaditya Singh**, Pierre Richemond, James McClelland, and Felix Hill. Data distributional properties drive emergent in-context learning in transformers. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, A*dvances in Neural Information Processing Systems*, volume 35, pages 18878–18891. Curran Associates, Inc., 2022.

This was our first work on understanding why ICL may be emergent, without being explicitly trained for, and introduces the key synthetic setups and data properties we use for much of the rest of the work. My contributions to this work were during ideation, iterating on experiments, providing feedback, and helping in writing of the paper.

2.1 Introduction

Large transformer-based language models show an intriguing ability to perform *in-context learning* (Brown et al., 2020). This is the ability to generalize rapidly from a few examples of a new concept on which they have not been previously trained, without gradient updates to the model. In-context learning is a special case of few-shot learning in which the output is conditioned on examples from a 'context', and where there are no gradient updates. It contrasts with *'in-weights' learning* (IWL), which is the standard setting for supervised learning – this is slow (requiring many examples), and depends on gradient updates to the weights. Earlier work in the context of 'meta-learning' showed how neural networks can perform few-shot learning without the need for weight updates (Santoro et al., 2016; Wang et al., 2016; Vinyals et al., 2016). To achieve this, the researchers explicitly designed the training regime to incentivize in-context learning, a process sometimes called 'meta-training'. In the case of transformer language models, however, the capacity for in-context learning is *emergent*. Neither the model's transformer architecture nor its learning objective are explicitly designed with in-context learning in mind.

Here, we consider the question of how transformer language models are able to acquire this impressive ability, without it being explicitly targeted by the training setup or learning objective. The emergence of in-context learning in language models was observed as recurrent models were supplanted by transformers, e.g. in GPT3. Was the novel architecture the critical factor behind this emergence? In this work we explore this possibility, as well as a second: that a capacity for in-context learning depends on the *distributional qualities of the training data.*

This hypothesis was inspired by the observation that many natural data sources – including natural language – differ from typical supervised datasets due to a few notable features. For example, natural data is temporally 'bursty". That is, a given entity (word, person, object, etc) may have a distribution that is not uniform across time, instead tending to appear in clusters (Sarkar et al., 2005; Alvarez-Lacalle et al., 2006; Neuts, 2007; Altmann et al., 2009; Serrano et al., 2009; Lambiotte et al., 2013). Natural data also often has the property that the marginal distribution across entities is highly skewed, following a Zipfian (power law) distribution with a long tail of infrequent items (Zipf, 1949; Piantadosi, 2014; Smith et al., 2018). Finally, the 'meaning' of entities in natural data (such as words in natural language) is often dynamic rather than fixed. That is, a single entity can have multiple possible interpretations (polysemy and homonymy, in language) and multiple entities can map to the same interpretation (synonymy, in language), usually in a context-dependent way. The combination of these properties may result in training data that occupies some middle-ground between the data used in canonical supervised learning and that used for few-shot meta-training.

In particular, standard supervised training typically consists of item classes that recur with uniform regularity, and with item-label mappings that are fixed throughout training – these properties allow a model to gradually learn over time, by encoding information into its weights, e.g. via gradient descent. By contrast, few-shot or in-context meta-training generally involves training a model directly on specially crafted sequences of data where item classes only recur and/or item-label mappings are only fixed *within episodes* – they do not recur and are not fixed across episodes (Santoro et al., 2016; Vinyals et al., 2016). Naturalistic data, such as language or first-person experience, has characteristics of both of these data types. As in supervised training, items (words) do recur, and the relationship between an entity and its interpretation (or meaning) is fixed, to some degree at least. At the same time, the skewed and long-tailed distribution of natural data means that some entities recur very frequently while a large number recur much more rarely. Importantly, however, these rare items are often bursty, making them disproportionately likely to occur multiple times within a given context window, somewhat like a sequence of 'meta-training' data. We can also see the dynamic relationship between entities and their interpretation (epitomized by synonyms, homonyms, and polysemy, in the case of language) as weaker versions of the completely dynamic item-label mappings that are used in few-shot meta-training, where the mappings are randomly permuted on every episode.

In this paper, we experimentally manipulated the distributional properties of the training data and measured the effects on in-context few-shot learning. We performed our experiments over data sequences sampled from a standard image-based few-shot dataset (the Omniglot dataset; Lake et al., 2015). At training, we fed each model (such as a transformer or recurrent network) with input sequences of Omniglot images and labels, varying the natural data-inspired distributional properties of choice. At evaluation, we assessed whether these properties gave rise to in-context learning abilities.

Our results showed that, indeed, in-context learning emerges in a transformer model only when trained on data that includes both burstiness and a large enough set of rarely occurring classes. We also tested two instantiations of the kinds of dynamic item interpretation observed in natural data – having many labels per item as well as within-class variation. We found that both interventions on the training data could bias the model more strongly towards in-context learning. The models we tested typically exhibited a tradeoff between rapid in-context learning vs. relying on information that was stored through slow, gradient-based updates ("in-weights" learning). However, we found that models could simultaneously exhibit *both* in-context learning and in-weights learning when trained on a skewed marginal distribution over classes (akin to the Zipfian distribution of natural data).

At the same time, architecture is also important. Unlike transformers, recurrent models like LSTMs and RNNs (matched on number of parameters) were unable to exhibit in-context learning when trained on the same data distribution. It is important to note, however, that transformer models trained on the wrong data distributions still did fail to exhibit in-context learning. Thus, attention is not all you need – architecture and data are both key to the emergence of in-context learning.

2.2 Experimental Setup

2.2.1 The training data

To investigate the factors that lead to in-context few-shot learning, we created training and evaluation sequences using the Omniglot dataset (Lake et al., 2015, MIT License), a standard image-label dataset for few-shot learning. Omniglot consists of 1623 different character classes from various international alphabets, with each class containing 20 handwritten examples. Using the Omniglot dataset allowed us to apply evaluation procedures that are standard in the study of few-shot learning.

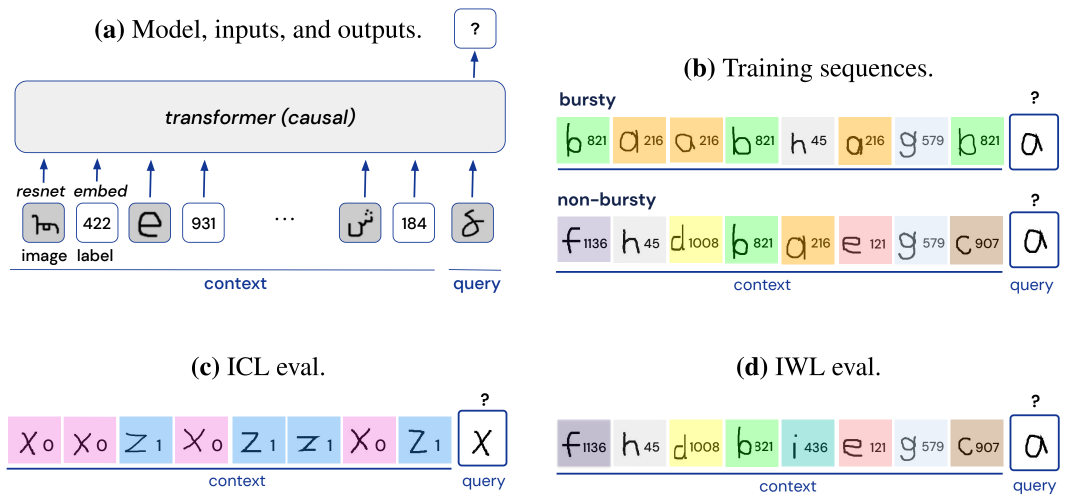


Figure 2.1: Experimental setup of the Omniglot sequence prediction task, for this chapter, as described in Section 2.2. In these example sequences, we add colors and use only Latin characters for visualization purposes. **(a)** For each experiment, a transformer model is trained on sequences of image-label pairs. The model is trained to minimize the loss on predicting the label corresponding to the final 'query' image. **(b)** In training, image-label mappings are fixed across sequences, in contrast to few-shot meta-training. The training data consist of a mix of 'bursty' and 'non-bursty' sequences. Bursty sequences, featuring multiple occurrences of the same classes, can be solved by learning labels across sequences (in-weights learning), or referring back to the context (in-context learning). Non-bursty sequences were composed of i.i.d. images. **(c)** To evaluate few-shot in-context learning, the model is presented with a standard few-shot sequence. The holdout image classes were never encountered in training, and are randomly assigned to labels $\{0,1\}$. Thus the model must use the context to predict the query label. We refer to this as the "ICL eval." **(d)** To evaluate in-weights learning, the model is presented with sequences where the labels are the same as in training. However, the query class does not appear in the context. Thus, the model must used information stored in weights to predict the query label. We refer to this as the "IWL eval."

The few-shot challenge is to classify an example of a character class that was never seen in training, based only on a few examples of that class and some alternate classes.

The training data consisted of sequences of images and labels (Figure 2.1b). The first 16 elements of each sequence comprised the 'context', and consisted of 8 image-label pairs (where each image was always followed immediately by its corresponding label). The final element was the 'query' image, and the aim of the model was to predict the correct label for the query.

Images were allowed to recur throughout training, and the integer label for each image class was unique and fixed across training, as in typical supervised datasets. We emphasize that this is a major departure from conventional few-shot training, where item-label mappings are completely novel on each episode, or the items themselves are novel on each episode.

In our standard experiments, we trained the model on a mixture of 'bursty' and 'non-bursty' sequences. In the bursty sequences, the query class appeared 3 times in the context. There are many possible ways to quantify or instantiate burstiness (e.g., Sarkar et al., 2005; Alvarez-Lacalle et al., 2006; Neuts, 2007; Altmann et al., 2009; Serrano et al., 2009; Lambiotte et al., 2013), but the 'bursty' sequences in our experiment were designed to reflect the within-context burstiness that is observed in e.g. language. To prevent the model from simply outputting the most common label in the sequence, a second image-label pair also appeared 3 times in the context. For the non-bursty sequences, the image-label pairs were drawn randomly and uniformly from the full Omniglot set. We can continuously vary the overall degree of burstiness in a dataset by changing the proportion of 'bursty' vs 'non-bursty' sequences.

2.2.2 The model

We trained 12-layer decoder-only Transformer models Vaswani et al. (2017) with absolute, additive sinusoidal positional encodings and a hidden dimension of $d_{\text{model}} = 64$. There were 8 heads per layer, an MLP expansion factor of 4 was used, and LayerNorm Ba et al. (2016) was applied at the start of each block. To embed Omniglot images, we jointly learned a resnet encoder He et al. (2015) with two blocks per group and channels per group (16, 32, 32, 64). Labels were embedded using a standard embedding layer. All models were trained on a softmax cross-entropy loss on the prediction for the final (query) image using the Adam optimizer Kingma and Ba (2015) with a learning rate schedule consisting of a linear warmup up to a maximum learning rate of 3e-4 at 4000 steps, followed by an inverse square root decay to the maximum of $5e5$ steps. The batch size (for each step) was 32 sequences. Error bars in all plots are calculated over 3 to 5 runs, depending on the plot.

2.2.3 The evaluation data

We evaluated trained models on two types of sequences, to measure (1) in-context learning and (2) in-weights learning. As in the training sequences, the evaluation sequences also consisted of 8 pairs of ‘context’ image and label tokens, followed by a single ‘query’ image token.

To measure a trained model’s ability for in-context few-shot learning, we used a standard few-shot setup (the “ICL eval”). The context consisted of a random ordering of 2 different image classes with 4 examples each, and the query was randomly selected from one of the two image classes (a ‘4-shot 2-way’ problem, in few-shot nomenclature). Unlike in training, where the labels were fixed across all sequences, the labels for these two image classes were randomly re-assigned for each sequence. One image class was assigned to 0, and the other to 1 (Figure 2.1c). Because the labels were randomly re-assigned for each sequence, the model must use the context in the current sequence in order to make a label prediction for the query image. For all the results presented in this chapter with the ICL eval, we use holdout image classes that were never seen in training, though results are generally similar if using classes that were seen in training (with slightly higher accuracies across the board – see Appendix A.1).

Although the model is always required to perform a full multi-class classification over all possible output labels (as in training), few-shot accuracy is computed by considering the model outputs only for the two labels seen in the few-shot se-

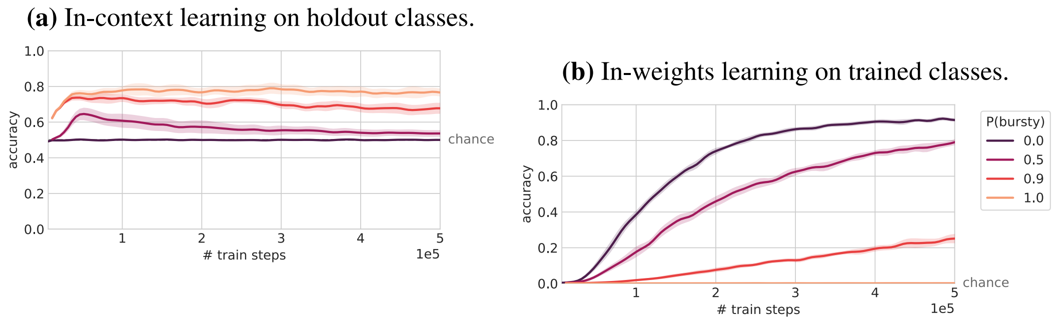


Figure 2.2: Effects of burstiness. $P(\text{bursty})$ indicates the proportion of training sequences that were bursty vs non-bursty. Models are evaluated on the two types of evaluation sequences, over the course of training. Burstiness in the training data increases in-context learning, and decreases in-weights learning.

quence (0 and 1), with chance at $1/2$. This ensures that performance above chance cannot be due to e.g. randomly selecting one of the labels from the context.

To measure in-weights learning of trained classes in a model, evaluation sequences consisted of image classes that were selected uniformly without replacement, with the same labels that were used in training (Figure 2.1d). Because the image classes were forced to be unique within each sequence, the query had no support in the context. Thus, the only way for a model to correctly predict the label was to rely on information stored in the model weights. For this problem, where the correct query label could be any of the labels seen in training, chance was usually $1/1600$.

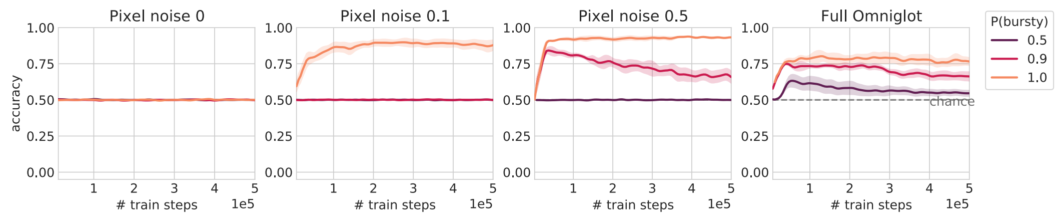
2.3 Results

Across all the experiments in this section (Figs 2.2-2.3), we also evaluated in-context learning on training classes (rather than holdout classes), again randomly assigning the classes to labels 0 and 1 (rather than using the ones seen in training). Evaluations looked similar in all cases, with only slightly higher performance.

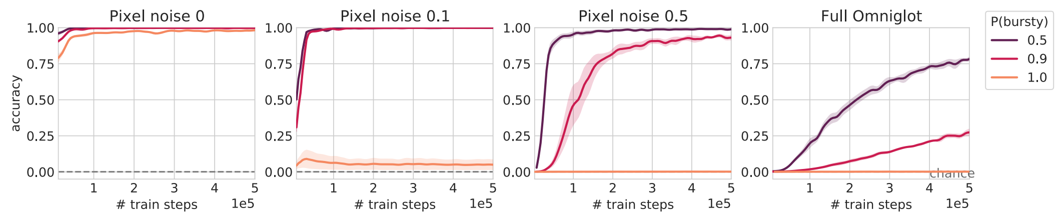
2.3.1 Burstiness is essential for emergent ICL

We vary levels of burstiness in the training data by varying the proportion of bursty vs non-bursty sequences in the training data (as described in Section 2.2.1). These experiments replicate the finding that transformers can acquire in-context few-shot learning even without explicit meta-training – notably, p(bursty) > 0 is required for

(a) In-context learning on holdout classes.



(b) In-weights learning on trained classes.



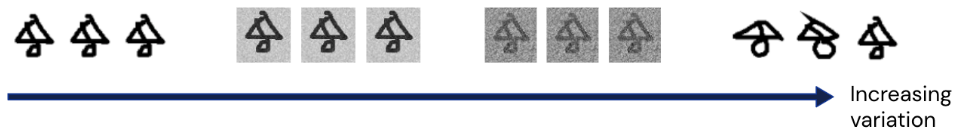


Figure 2.3: Effects of in-class variation. When we increase the in-class variation (from left to right), in-context learning tends to increase **(a)** while in-weights learning decreases **(b)**. Both effects are nonetheless upper-bounded by the difficulty of in-class generalization, with the 'Full Omniglot' problem being more difficult than the rest. For the 'Full Omniglot' experiments, each class contained the full set of 20 Omniglot exemplars per class. For the remaining experiments, each consisted of only a single Omniglot exemplar image, with varying levels of Gaussian pixel noise.

any ICL to emerge. They further show that, as hypothesized, the model displays better in-context learning with more burstiness in training (Figure 2.2a). We also see that in-context learning trades off against in-weights learning – greater burstiness simultaneously leads to lower weight-based learning (Figure 2.2b). Interestingly, the models can in some cases lose an initial bias towards in-context learning, moving towards in-weights learning over the course of training.[1]

2.3.2 In-class variation is necessary for emergent ICL

As mentioned earlier, the Omniglot dataset consists of 20 exemplars per class, which means multiple images are mapped to the same label. This effect is anal-

[1]In Chapter 3, we will see that this is actually true in all cases, just with varying timescales, but here we continue with the experiments in our first work.

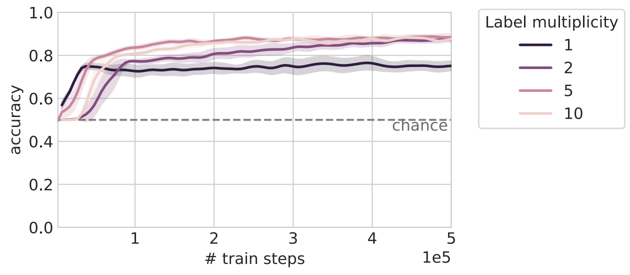


Figure 2.4: Dynamic meanings improve in-context learning. Increasing the number of labels per class ('label multiplicity') increases in-context learning.

ogous to how in natural language, multiple words could have the same meaning. We consider the importance of this "in-class variation" here by considering versions of our dataset with reduced variation, or synthetic (as opposed to naturalistic) variation.

In Figure 2.3, we show that removing in-class variation, by making each class only consist of 1 exemplar, leads to ICL disappearing and only IWL being learned. We then add in-class variation in a more controlled fashion, by adding pixel-level Gaussian noise, which leads to ICL reappearing. We notice an interesting interaction between in-class variation and burstiness: At low levels of in-class variation, we need P(bursty)=1 to see any ICL. As in-class variation is increased, progressively lower levels of burstiness are needed to elicit ICL, but again we see that P(bursty)=1 is necessary for what-appears-to-be persistent ICL. Overall, these results show how that in-class variation is crucial for the emergence of ICL.

Generally, this finding is somewhat surprising, as in-class variation makes ICL more difficult as well (matching an exemplar to any exemplar from its class, rather than the same exemplar in the case of 1-exemplar-per-class).[2] We suggest that in-class variation makes IWL "even more" difficult, thus making ICL emerge more strongly. In other words, in-class variation preferentially hampers in-weights learning more than it hampers in-context learning.

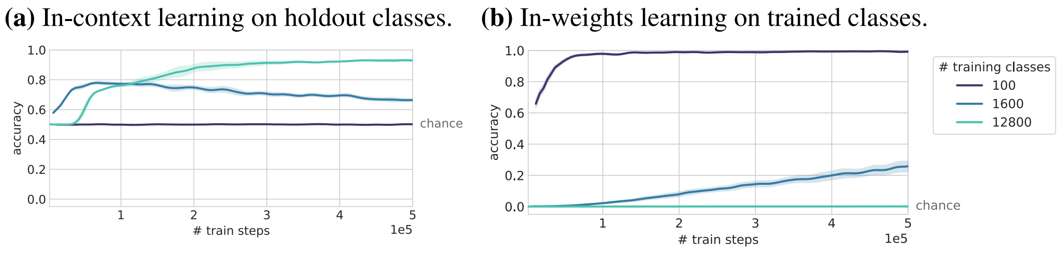


Figure 2.5: Effects of number of classes. Increasing the number of training classes improves in-context learning, while reducing in-weights learning.

2.3.3 Label mutliplicity necessitates ICL

A variant of the above experiments would be to instead introduce label multiplicity, inspired by dynamic meanings of words: instead of multiple words that have the same meaning, as in Section 2.3.2, a single word in natural language will often have multiple meanings, with the specific meaning being inferred from context (Gale et al., 1992).

We experimented with this idea by assigning each image class to multiple possible labels and, in the data sequences, randomizing the label shown after each image from among the possible labels. If a class appeared more than once in the same sequence, the label was consistent for all presentations within that sequence (to match natural language). In Figure 2.4, we see that increasing the 'label multiplicity' (the number of labels per class) also increases in-context learning. Again, burstiness was fixed for these experiments at $p(\text{bursty}) = 0.9$.

These findings are not as surprising as the in-class variation results, as label mutliplicity creates sequences where ICL is *necessary* to minimize loss: a pure IWL strategy would have no way of disentangling the multiplicity of labels, as the only way to do so is by using context.

2.3.4 A large number of rare classes is necessary for emergent ICL

Next, we consider the effect that vocabulary size (in our setup, the number of classes) may have on in-context learning. As we increase the number of classes

[2]We revisit this observation in Chapter 5 with more controls.

from 100 to 1600 (and correspondingly decrease the frequency of each class), we see improvement of in-context learning (Figure 2.5a). As before, we also see an accompanying decrease in in-weights learning (Figure 2.5b). This accords with our hypothesis about the importance of having a long tail in the distribution, or a large vocabulary. Note that the bias against in-weights learning cannot be explained by the number of exposures to each class – even controlling for the number of exposures, the model trained with 1600 classes is much slower to achieve similar levels of in-weights learning. Importantly, we need both burstiness and a large number of classes for in-context learning to emerge (as evidenced by all curves showing p(bursty)=0.9, yet ICL not emerging if only using 100 classes).

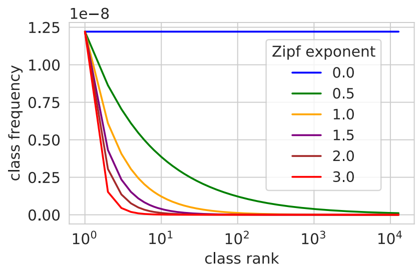
In order to further increase the number of classes beyond the 1623 available in the original Omniglot dataset, we rotated ($0°, 90°, 180°, 270°$) and flipped (left-right) the images, obtaining $8\times$ more image classes. We ensured that the holdout set did not include transformed versions of train images. Training on these 12800 classes further improved in-context learning (and reduced in-weights learning) (Figure 2.5).

2.3.5 Zipfian marginal distributions enable co-existent ICL and IWL

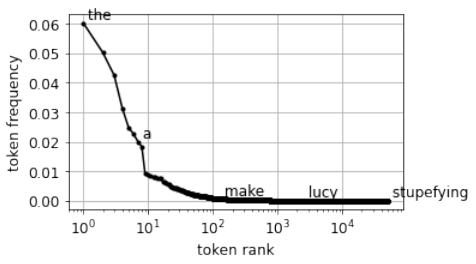
In the previous section, we saw a consistent tradeoff between in-context learning and in-weights learning – no models could maintain both. However, it is useful for a model to have both capabilities – to remember information about classes that will re-appear in evaluation, while also being able to perform rapid in-context learning on new classes that appear only in holdout. Large language models certainly do have both of these capabilities. How might we achieve this?

For all prior experiments, the training data were marginally distributed uniformly over classes, even if the data were non-uniform in other ways. I.e., each class was equally likely to appear, marginalizing across the dataset. We postulated that we might achieve both types of learning in the same model by instead training on marginally-*skewed* distributions. In this case, some classes appear very commonly, while most classes appear very rarely. Many natural phenomena such as

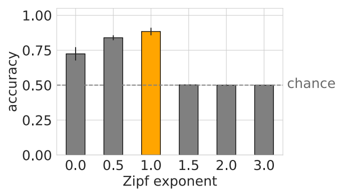
(a) Examples of Zipfian distributions.



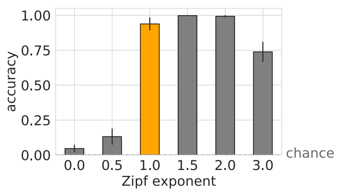
(b) Distribution of tokens in a natural language corpus.



(c) In-context learning on holdout classes.



(d) In-weights learning on common classes.



(e) In-weights learning on rare classes.

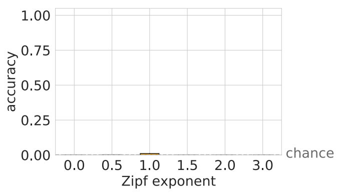


Figure 2.6: Effects of training on Zipfian (rather than uniform) marginal distributions over classes. **(a)** Examples of Zipfian distributions with varying exponents. **(b)** The distribution of tokens in an example English-language corpus. In **(c-e)**, bars indicate mean evaluation accuracy in the window [400k, 500k] steps of training. **(c)** As we increase the Zipf exponent, i.e. increasing the skew on the class distribution, we see a decrease in in-context learning. **(d)** In-weights learning of the 10 most common classes, in contrast, increases with more skew. With uniform training (Zipf exponent = 0), the model exhibits only in-context learning and not in-weights learning. However, if we train on skewed distributions, there is a sweet spot where both in-context learning and in-weights learning can be maintained at a high level in the same model (Zipf exponent = 1, for this particular training regime). Coincidentally, a Zipf exponent of 1 corresponds approximately to the skew in many natural languages. **(e)** Rare items from training are never memorized (performance is at chance for all Zipf exponents).

word distributions take this form, and are classically described as a Zipfian (power law) distribution (Zipf, 1949):

$$p(X = x) \propto \frac{1}{x^{\alpha}} \tag{2.1}$$

Here, X is the rank of the class (e.g. 1 for the most common class), and the exponent $\alpha \in [0, \infty)$ determines the degree of skew. Figure 2.6a shows some examples of Zipfian distributions with various exponents. Figure 2.6b shows an example of token distributions in English (from the Brown corpus; Francis and Kucera, 1979).[3] This type of skewed distribution could allow a model to learn common classes in its weights, while the long tail of rare classes simultaneously induces an ability for in-context learning.

To test this hypothesis, we trained on Zipfian distributions, varying the Zipf exponent and hence the degree of skew. We used the same training sequences as before, with 12800 classes and $p(\text{bursty}) = 0.9$. Our results are shown in Figs 2.6c-e. We evaluate in-weights learning separately on common classes (the 10 classes seen most often in training) and on rare classes (the remaining classes). When there is no skew, all classes are relatively rare, and we see high levels of in-context learning but no in-weights learning. Increasing the skew leads to the loss of in-context learning and increased in-weights learning of common classes.[4] In between the two extremes, we observe a sweet spot at Zipf exponent = 1, where the model maintains high levels of both in-context learning and in-weights learning of common classes. Intriguingly, natural languages are best described by a Zipfian distribution with an exponent of approximately 1 (Piantadosi, 2014). Note though that the sweet spot for simultaneously maintaining in-weights and in-context learning in transformers may differ, depending on the training regime.

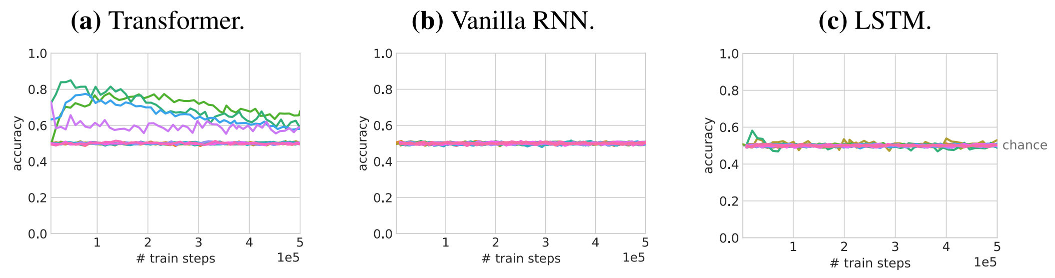


Figure 2.7: In-context learning in transformers vs. recurrent architectures. We compare architectures while holding fixed the number of layers, hidden layer size, and number of parameters. Only a transformer is able to attain in-context learning; the Vanilla RNN and LSTM never perform above chance. One run was performed for each set of hyperparameters in a hyperparameter sweep. Each color denotes one run, but not any particular hyperparameter values.

2.3.6 Data is not everything – architecture matters too

To investigate whether these results are specific to transformer models, we performed similar experiments using recurrent sequence models. For these models, we simply replaced the transformer with either a vanilla recurrent neural network (RNN; David E. Rumelhart et al., 1985) or a long short-term memory network (LSTM; Hochreiter and Schmidhuber, 1997). We used the same training sequences as before, with 1600 classes and $p(\text{bursty}) = 0.9$. We also used the same image and label encoders, and cross-entropy classification loss. The recurrent models were matched to the transformer for depth, number of parameters, and hidden layer size. We performed a comprehensive hyperparameter search for all models (see Appendix A.2 for details).

In these experiments, we see that the recurrent models are never able to achieve in-context learning, despite the parity in training setup (Figure 2.7). Interestingly, the transformer actually outperforms the recurrent models on in-weights learning as well (see Figure A.2 in Appendix A.2), indicating that we cannot explain these results by proposing that recurrent models are simply more biased towards in-weights learning than transformers.

[3]Plot generation adapted from https://gist.github.com/fnielsen/7102991

[4]We see decreased in-weights learning for Zipf exponent = 3, because that level of skew leads to extreme focus on a tiny number of classes (e.g. the three most common classes form 97% of the data).

2.4 Discussion

In summary, we find that both data and architectures contribute significantly to the emergence of in-context learning in transformers.

We identify several features of training data that can promote in-context learning – burstiness, dynamic meaning (as instantiated by in-class variation or multiple labels per class), and the number and rarity of training classes. These data properties allow in-context learning to emerge despite differing significantly from the data used in standard few-shot meta-training, in that we allow items and item-label mappings to recur throughout training. These properties are also central features of natural data including language, and thus may explain the remarkable emergence of in-context learning in large language models without explicit meta-training.

Notably, we observed three key regimes of behavior, depending on data properties:

1. IWL-only: ICL does not emerge at all. Properties that induce: p(bursty) = 0, or no in-class variation, or small number of classes (100).

2. Transient ICL: ICL emerges, then gives way to IWL. Properties that induce: p(bursty) = 0.5 or 0.9 with 1600 classes.

3. ICL-only: ICL emerges strongly and seems to persist. Properties that induce: p(bursty) = 1, in-class variation, 12800 classes

Looking back, some of these make sense – when training on p(bursty) ¡ 1, the network *must* learn some IWL strategies to perform well on the non-bursty sequences, which could explain Case 2. The fact that ICL still emerges briefly, presumably due to the presence of bursty data, is interesting. We will explore these ideas further in the next chapter.

Beyond identifying drivers of ICL emergence in larger models trained on natural language, this work demonstrates a tractable synthetic setting for studying ICL. We will continue to use the Omniglot task introduced here, with the ICL-inducing properties of burstiness, in-class variation, and a large number of classes, in our subsequent work.

2.4.1 Broader implications

This newfound understanding of data-distributional properties driving in-context learning emergence may also help us design and collect datasets to achieve in-context learning in domains *outside* of language, an area of ongoing research (e.g. Finn et al., 2017; Wang et al., 2016; Hill et al., 2020). Given that reinforcement learning environments are generally designed to be uniformly distributed (Chan et al., 2022b), or that supervised datasets are frequently rebalanced to have *more uniform* distributions (Chawla et al., 2002; Van Hulse et al., 2007; Katharopoulos and Fleuret, 2019), we may be missing an opportunity to endow non-language models with a powerful capability. We may need to consider data distributions more carefully when pre-training in non-language domains, as well. For example, recent work has shown that pre-training on language data was useful for offline reinforcement learning, but pre-training on vision data was not (Reid et al., 2022) – could this difference be due to the non-uniform, structured distribution of the language data?

Since publishing our work, others have followed up and found exactly this: Raparthy et al. (2023) find that training on sequences of bursty episodes can lead to in-context learning of new sequential decision making tasks.

Chapter 3

The transient nature of emergent in-context learning in transformers

This chapter is based on:

> **Aaditya Singh**, Stephanie Chan, Ted Moskovitz, Erin Grant, Andrew Saxe, and Felix Hill. The transient nature of emergent in-context learning in transformers. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 27801–27819. Curran Associates, Inc., 2023.

Building on the three cases identified in the previous chapter, I asked: What if these 3 regimes are actually the same "transient ICL" case, with data properties simply modulating the relative speeds of ICL and IWL emergence? I find, surprisingly, that this does seem to be the case.

My contributions to this work were comprehensive: coming up with the original hypothesis, proposing experiments to verify it, extending these experiments to show generality of the effect, and writing of the paper.

3.1 Introduction

Concurrent to our first work, a complementary body of work studied emergence in massive models trained directly on organic web-scale data, with the takeaway that impressive capabilities such as ICL are more likely to emerge in large models trained on more data (Kaplan et al., 2020; Rae et al., 2021; Chowdhery et al., 2022; Wei et al., 2022). However, this reliance on massive models poses serious practical challenges, such as how to innovate rapidly, how to train energy-efficiently and in low-resource settings, and how to run efficiently at deployment. Thus, a growing line of research (Fu et al., 2023; Huang et al., 2022; Eldan and Li, 2023) has focused on achieving comparable performance (including emergent ICL) in smaller transformer models.

Overtraining is currently the favored approach for training small yet performant transformers. These small models are trained on more data, possibly with repetition, beyond what is prescribed by scaling laws and compute budget (Hoffmann et al., 2022; Touvron et al., 2023a). At its core, overtraining relies on an assumption that is implicit in most (if not all) recent explorations of ICL in LLMs: *persistence*. That is, it is assumed that once a given model has been trained sufficiently for an ICL-dependent capacity to emerge, that capacity will be retained as training progresses, provided the training loss continues to decrease.

Here, we show that the general assumption of persistence is false. We build on our findings in Chapter 2, and find that rather than data properties controlling which regime a transformer lands in (IWL-only, Transience ICL, ICL-only), data properties simply control the timescale. In other words, ICL is generally transient when it emerges, and our prior evidence (e.g., p(bursty)=1 in Section 2.3.1) pointing at “ICL-only” cases simply did not train long enough to see the transience.

We find this transience occurs across a range of model sizes (Section 3.4.1), dataset sizes (Section 3.4.2), and dataset types (Section 3.4.3), though we do identify some properties (Section 3.4.4) that can postpone transience. In Section 3.5, we study ways to alleviate this risk, and show that regularization may offer a path to persistent ICL. In Section 3.6, we present preliminary evidence that ICL transience is caused by competition with IWL circuits.

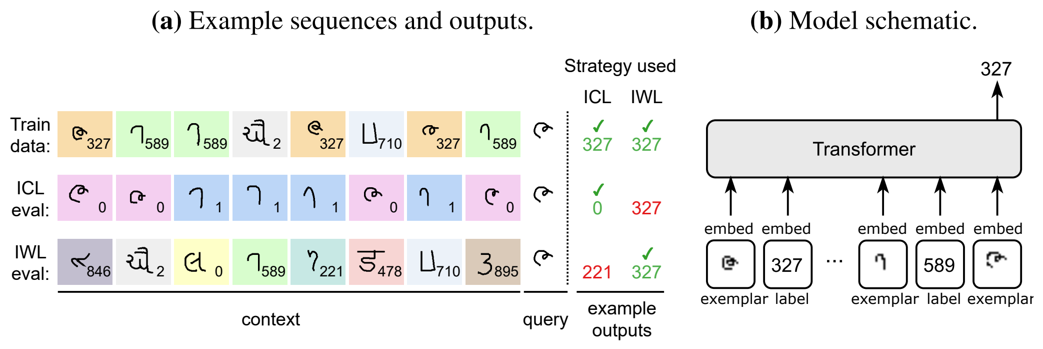


Figure 3.1: An overview of our setup. (a) Example sequences during training, ICL evaluation, and IWL evaluation. Example outputs are colored green when correct and red when incorrect. Note that for train sequences, ICL and IWL strategies *both* result in the correct answer. On ICL eval sequences, the IWL prediction is incorrect. ICL is required to remap exemplars to the randomized 0 or 1 labels. On IWL eval sequences, there are no matching exemplar-label pairs in context, so IWL is necessary. In the language of Chapter 2, we use the same evaluators, and always train in the p(bursty)=1 setting, to not advantage IWL unfairly (if p(bursty) $<$ 1, only IWL strategies can achieve perfect accuracy on the training data). (b) Model schematic. Training and evaluation focuses on the predicted label for the final exemplar.

3.2 Experimental Setup

3.2.1 Dataset construction

Our setup largely matches that of our previous work (see Section 2.2), using sequences of Omniglot (Lake et al., 2015) image-label pairs as training data. There are a few notable differences:

- We only consider p(bursty)=1 for training sequences, as depicted in Figure 3.1.
- To explore the generality of our results, we also extend to exemplars derived from language model token embeddings (rather than Omniglot). We use pre-trained token embeddings from the LLaMa family (Touvron et al., 2023a) of large language models, which use a vocabulary size of 32,000 tokens. We extract the input embedding matrices from all four open-source LLaMa 1 models. We then subset and cluster (Johnson et al., 2019) these token embeddings to form 3,200 classes of 5 tokens each. For each LLaMa model

size, we construct two such datasets, with different clustering seeds. The full procedure is described in Appendix B.1.

- To make sure that results are not confounded by a failure to generalize to out-of-distribution exemplars, we default to constructing evaluation sequences out of in-distribution training classes. Note that the sequences in both ICL eval and IWL eval are still out-of-distribution (see Figure 3.1a): in the former, the labels do not match those from training (being re-assigned to 0-1), and in the latter, the sequences are not bursty (while all train sequences are bursty).

3.2.2 Model and training details

For our transformer model, the "default" settings consist of the same model as that from Section 2.2.2: 12 layers, with an embedding dimension of 64 and an additive, sinusoidal positional encoding scheme (Vaswani et al., 2017). Each element of the sequence is passed through an embedder, either a standard embedding layer for the label tokens, a ResNet encoder for the exemplar images, or a simple linear layer for exemplars created from LLaMa tokens (Figure 3.1b). All embedders are trained jointly with the network.

In our experiments, we vary multiple aspects of this architecture and observe the effect on the transience of ICL. All experiments were run with 2 seeds and used the Adam optimizer (Kingma and Ba, 2015) (with default parameters of $\beta_1 = 0.9, \beta_2 = 0.999$) and a learning rate schedule with a linear warmup up to a maximum learning rate of 3e-4 at 4,000 steps, followed by an inverse square root decay. We did not observe qualitative differences in behavior between seeds – shaded areas in all figures indicate standard deviation around the mean. We ran most experiments for 5e7 iterations, but truncated some early when signs of transience were clear.

We note that, despite the long training times, all our experiments are still in the < 1 epoch regime, due to the combinatorial nature of our dataset. We estimate there are over $1,600 \times 1,599 \times (1,598 \times 1,597/2) \times (8!/(3! \times 3!)) \times (20^9) = 1.87 \times 10^{27}$ possible training sequences, of which our model sees a maximum of $32 \times 5 \times 10^7 = 6.4 \times 10^8$.

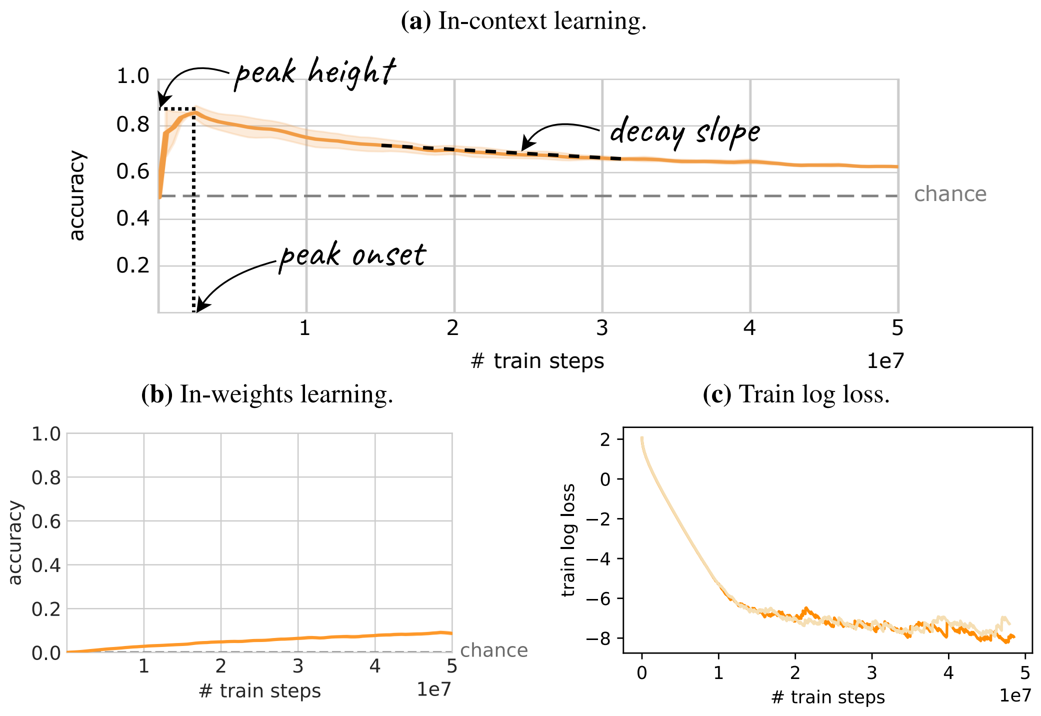


Figure 3.2: In-context learning is transient, shown for our “default” settings: 12 layers, embedding dimension of 64, trained on 1,600 classes, with 20 exemplars per class. All training sequences are bursty (see Figure 3.1a for details). We previously found these settings to strongly incentivize ICL, but did not observe ICL transience (see Figure 2.2a), as we did had not trained long enough. (a) ICL evaluator accuracy. (b) IWL evaluator accuracy. We note that, while accuracy on train sequences is 100%, accuracy on the IWL evaluator is very slowly increasing, as the test sequences are out-of-distribution. (c) Training log loss. Two colors indicate two seeds used for experiments.

L2 regularization, when used, was implemented by adding the squared weights of the model (excluding batch norm parameters) to the loss term.

3.3 In-context learning is transient

Figure 3.2a shows that ICL is transient even when using data-distributional settings we previously found to strongly incentivize ICL. We annotate various aspects of the ICL curve, such as the peak height, peak onset, and decay slope. In Section 2.3.1, we had stopped training at 5e5 training steps, before peak onset. Training for much longer in an identical setting leads to the disappearance of ICL. Notably, ICL is slowly replaced by IWL (as seen by the steadily increasing accuracy on the IWL evaluator, Figure 3.2b), all while the training loss continues to decrease (Figure 3.2c).

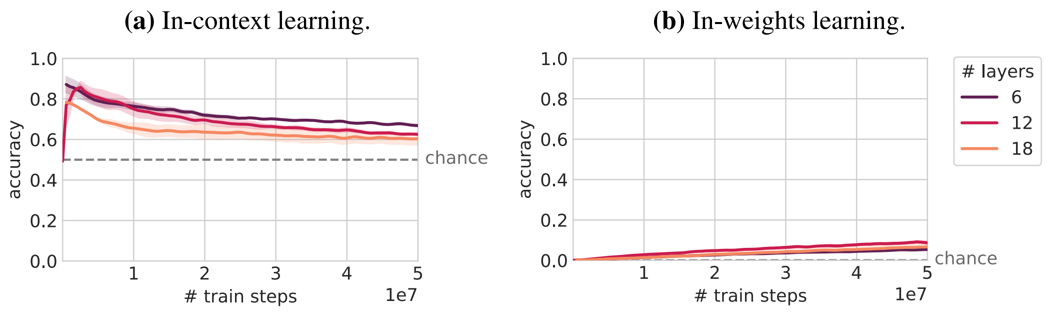


Figure 3.3: (a-b) ICL is transient regardless of model depth, with no clear trend of peak height or peak onset. Decay slopes are roughly similar across model sizes.

This result is especially surprising since we train in the p(bursty)=1 setting, where ICL should be "sufficient" to fully solve the training task (since the context always contains a relevant exemplar-label mapping that can be used). Next, we will see it extend across a range of settings.

3.4 In-context learning is transient across settings

3.4.1 Effect of model size

We investigate whether the ICL transience effect is modulated by model size, given the current prevalence of extremely large transformer models. We find that there is no consistent effect of model depth on ICL transience (Figure 3.3a-b). In fact, deeper models can lead to lower peak heights (e.g., the 18-layer model has a lower peak height than the 12-layer model). Notably, the decay slope is similar across different model depths, indicating that the ICL is unlikely to persist simply by scaling model depth.

We also experiment with the width of the model in Section 3.6, which sheds some light on the causes of ICL transience.

3.4.2 Scaling dataset size

Next, we investigate the effects of dataset size on ICL transience. There are two primary ways of varying the dataset size: increasing the number of exemplars per class (a form of increasing in-class variation), or increasing the number of classes.

We previously found that in-class variation was crucial for ICL to emerge (Section 2.3.2), and we find a similar trend here. When the number of exemplars per

(a) In-context learning. **(b)** In-weights learning.

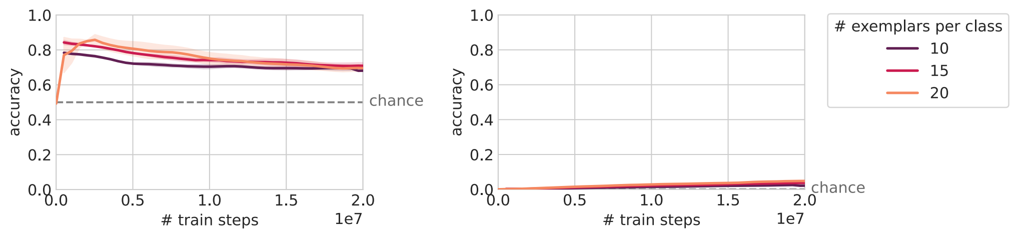


Figure 3.4: In-class variation improves ICL, as we saw previously and again see here by the higher peak heights, but ICL is nonetheless transient across settings.

(a) In-context learning. **(b)** In-weights learning.

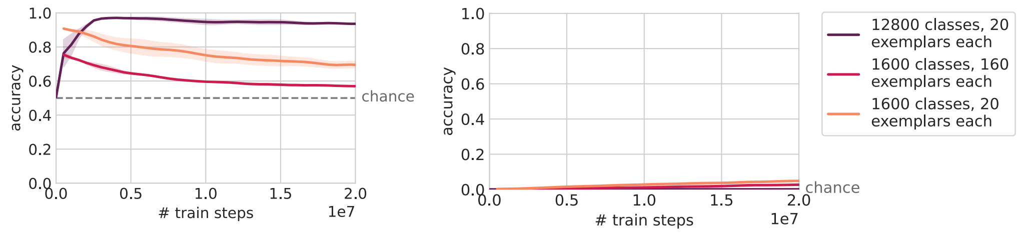


Figure 3.5: Effect of dataset size on ICL transience. Increasing the number of classes leads to higher ICL peak height and a more gradual decay slope. Despite the number of exemplar-label pairs being the same in the purple and red curves, we note that the ICL and IWL behavior is very different when we increase the number of classes (purple) v.s. the in-class variation (red). Also, note that chance level performance for IWL differs slightly: 1/12,800 for the other curve (purple), and 1/1,600 for the 1,600 classes lines (red and orange).

class is increased (from 10 to 15 to 20) while keeping the number of classes fixed, we see an increase in ICL peak height (Figure 3.4a). However, despite higher peak heights, we also see steeper decay slopes, indicating that increasing in-class variation is unlikely to make ICL persistent.

When increasing the number of classes, we again find a similar trend (to Section 2.3.4) with peak heights increasing (Figure 3.5a). In this case, we also find that decay slopes become more gradual. To better illustrate the differences between increasing in-class variation and increasing the number of classes, we do another experiment where the total number of exemplar-label pairs is fixed to $12,800 \times 20 = 1,600 \times 160 = 256,000$, with one run having more classes and less in-class variation, and another run having fewer classes and more in-class variation. We find that increasing the number of classes is a more effective way to promote ICL (higher peak height, shallower decay slope) than increasing the in-class variation, perhaps due to each class appearing less often during training. This result also

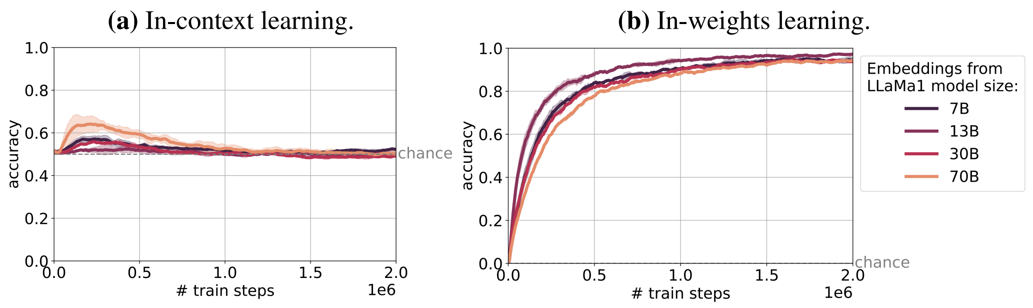


Figure 3.6: ICL is transient across a range of input datasets, derived from clusterings of LLaMa 1 token embeddings from the four different sizes. Error bars indicate standard error across two possible datasets derived from each model's embedding matrix. Further results can be found in Appendix B.1.

indicates that, beyond a certain point, in-class variation actually hurts ICL performance (lower peak height, steeper decay slope, as seen in Figure 3.5a). To conclude, ICL remains transient as we increase the number of classes, but training for longer and longer becomes necessary to make it go away, given the shallower decay slope.

3.4.3 Extending to language model token embeddings

To explore the generality of our results, we replaced the image exemplars with exemplars derived from language-model token embeddings (procedure described fully in Appendix B.1). On all of these token-embedding derived datasets, we find that whenever ICL emerges, it is subsequently transient (Figure 3.6). Furthermore, the peak height is lower than in our other experiments. In terms of timing, the peak onset as well as subsequent transience occurs on a quicker timescale (note the shorter x-axis scale in Figure 3.6). These results also further corroborate our findings on dataset size (Section 3.4.2), as the constructed token embedding-label dataset is smaller ($3,200 \times 5 = 16,000$ exemplar-label pairs, as opposed to $1,600 \times 20 = 32,000$ pairs in our Omniglot experiments), so we would expect lower heights and sharper decay slopes. Overall, these experiments are a promising step towards connecting our findings to natural language, and demonstrate that the transience phenomenon is not limited to image-based exemplar-label datasets.

3.4.4 Changing the data distribution

In Section 2.3.5, we saw that switching to a Zipfian marginal distribution could lead to co-existence of ICL and IWL. Could using a Zipfian marginal distribution also

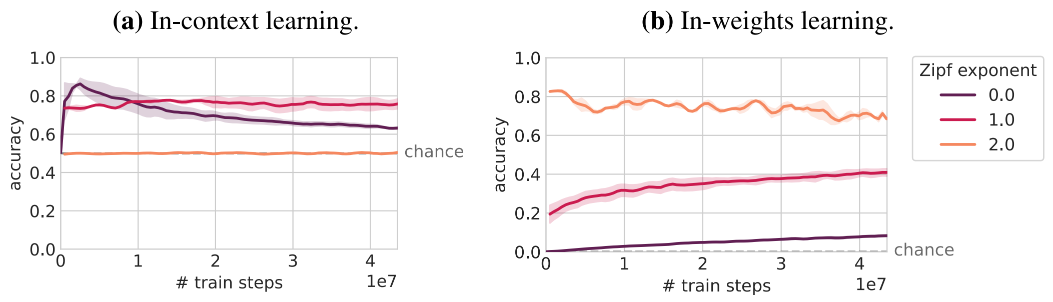


Figure 3.7: ICL transience as a function of dataset skew, implemented across classes in accordance with a Zipfian distribution. (a) A moderate Zipf exponent of 1 leads to a significant delay of ICL peak onset followed by a shallow decay slope, implying models would need to be trained for even longer to get full transience. However, an extreme level of skew (Zipf exponent 2) leads to the complete loss of ICL altogether. (b) IWL, evaluated on the same skewed distribution over query classes as in the training data (thus high accuracy on Zipf exponent 2 is not unexpected, as the IWL test set is dominated by a small number of classes).

play a similar role in reducing the transience of ICL?

Indeed, in Figure 3.7, we find this to be the case. We implemented different levels of skew across data classes according to a Zipfian distribution $p(X = x) \propto x^{-\alpha}$ (x corresponds to the rank of an input class—e.g., $\alpha = 0$ corresponds to a uniform distribution). At Zipf exponent $\alpha = 1$, we find a severely delayed ICL peak onset, thereby delaying the subsequent transience, and a shallow decay slope following this delayed onset (similar to that seen with a large number of classes in Figure 3.5a). These results our consistent with our earlier results – stopping training early, when using skewed distributions, can lead to a combination of ICL and IWL in the same network (as evidenced by the increased IWL at Zipf exponent $\alpha = 1$ as compared to Zipf exponent $\alpha = 0$, Figure 3.7b). However, it is important to note that if the Zipfian exponent is too extreme (e.g., $\alpha = 2$), ICL does not emerge at all, and the network uses IWL to solve the task. We conclude that moderately skewed Zipfian data does not eliminate transience, but mitigates it by necessitating training for an even longer time to observe transience (similar to when there is a large number of classes).

3.5 Regularization may eliminate ICL transience

If we consider that ICL is a more generally useful solution than IWL, then ICL transience is, in a sense, the *opposite* of the grokking phenomenon, where a model

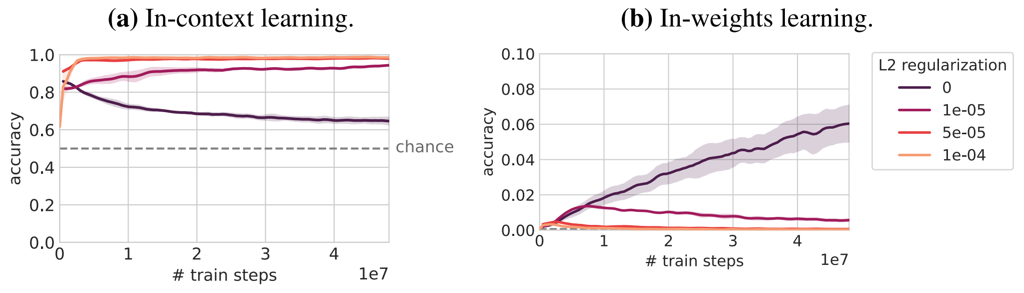


Figure 3.8: L2 regularization potentially eliminates ICL transience. Increasing regularization decreases ICL transience, at least up to a point. Regularization also seems to lead instead to IWL transience. (Note that the y-axis in Figure 3.8b is zoomed in so that IWL transience is visible.)

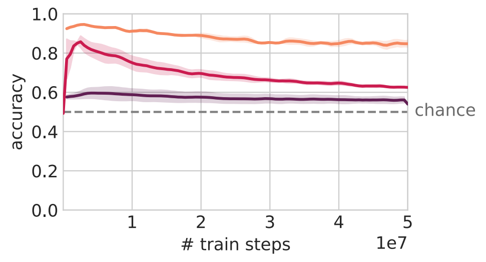
changes from a less to a more general solution after long periods of training (Power et al., 2022). Regularization (or implicit regularization) may help to drive grokking (Nanda et al., 2023; Thilak et al., 2022), with some work even showing an impact of dataset size as well (Liu et al., 2023), analogous to what we find above. Thus, would we see regularization mitigating the ICL transience phenomenon?

We see in Figure 3.8 that this is indeed the case, with L2 regularization seeming to even eliminate ICL transience entirely, at least on the timescales that we tested. Without any regularization, we see transient ICL as before. However, as we add increasing levels of L2 regularization, we see that the decay slope of ICL performance goes to 0. On the other hand, we actually observe some transience of in-weights learning (purple and maroon curves, Figure 3.8b). These results indicate that the ICL circuit is a lower norm solution than the IWL circuit. Thus, regularization may offer a path to persistent ICL. However, with values of regularization that are too high (> 0.0001), the network is over-regularized and ICL transience returns, but, notably, IWL accuracy never rises above zero either (see Appendix B.2 for more train loss curves and more extreme values of regularization).

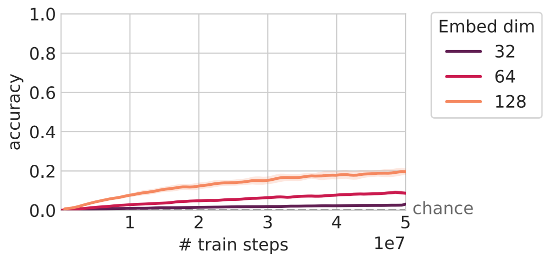
3.6 ICL transience may be caused by competition with IWL circuits

We performed initial investigations to understand the origins of ICL transience: why does ICL fade when the IWL solution emerges? Do the ICL and IWL circuits compete with each other for resources, for example, in the transformer's residual

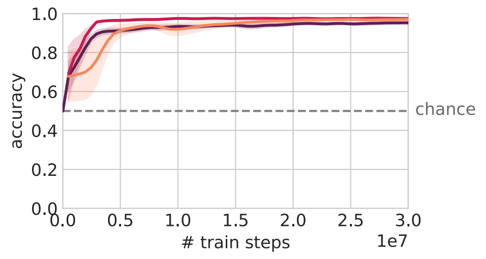
(a) In-context learning.



(b) In-weights learning.



(c) In-context learning, when training on data that only supports in-context learning.



(d) In-weights learning, when training on data that only supports in-weights learning.

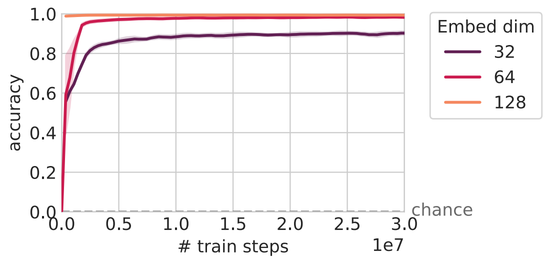


Figure 3.9: (a-b) We find a trend that wider models (larger embedding sizes) have higher ICL peak heights and more gradual decay slopes. (c-d) However, when we trained on data that allowed us to specifically elicit each type of learning, we found that embedding size was only consistently helpful for enhancing IWL, indicating that mitigation of ICL transience is likely due to decreased interaction between the two circuits, as opposed to an enhancement of ICL capabilities.

stream (Elhage et al., 2021)? If so, we would expect wider transformers with larger embedding size (and therefore greater residual stream capacity) to be less afflicted by ICL transience. In Figure 3.9a, we see that this is the case: Greater model embedding sizes lead to higher peak heights and gentler decay slopes. However, one confound to consider is that larger embedding may just independently enhance ICL abilities (as they enhance IWL abilities, Figure 3.9b), rather than reduce competition between circuits.

To better understand how scaling the embedding dimension enhances ICL and IWL, we trained models on data that could be solved via ICL only or IWL only, rather than being solvable by both strategies. For ICL-only training, we used the same type of sequences used in our ICL evaluator, where the network *must use ICL* to solve the query-prediction task. For IWL-only training, we used the same type of sequences used in our IWL evaluator, where the network *must use IWL* to solve the query-prediction task. Figure 3.9d shows that larger embedding dimensions

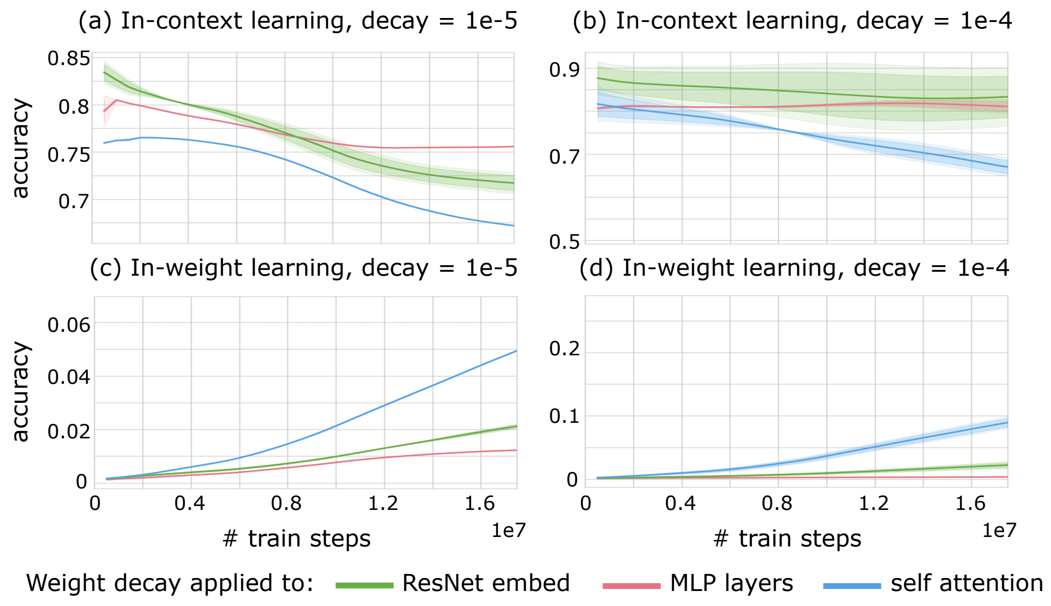


Figure 3.10: Selectively applying weight decay to different sets of network weights reveals that IWL relies on MLP layers. When we apply weight decay only to self-attention layers (blue), ICL is still transient. When we only apply weight decay to MLP layers, ICL transience is mitigated (red). These results can be interpreted in light of prior work indicating that in-weights information is stored in MLP layers (Geva et al., 2021; Meng et al., 2023). By selectively penalizing this behavior, we enable ICL to persist, thus providing convergent evidence that – when no weight decay is applied – ICL fades due to competition with IWL circuits.

enhance the model's ability to perform IWL, but do not consistently help ICL (Figure 3.9c). Thus, these results indicate that the increased peak height and shallower decay slope in Figure 3.9a are likely due to decreased competitive interactions with IWL circuits, rather than an enhancement of ICL capabilities.

3.6.1 Convergent evidence via selective regularization

We arrived at convergent evidence by building on the regularization experiments from Section 3.5. We applied weight decay selectively to either the ResNet embedding layers, the MLP layers, or the self-attention layers. When we applied weight decay to only the self-attention layers, ICL was still transient. When we applied weight decay to the MLP or ResNet layers, however, ICL transience was mitigated (Figure 3.10). These results can be interpreted in light of prior work indicating that in-weights information is stored in MLP layers (Geva et al., 2021; Meng et al., 2023). By selectively penalizing this behavior, we enable ICL to persist, thus providing convergent evidence that – when no weight decay is applied – ICL fades due to competition with IWL circuits.

3.7 Related Work

3.7.1 Prior work in synthetic settings

This work builds on several papers that seek to understand in-context learning by analyzing small networks and/or controlled data sources. Xie et al. (2021) consider an implicit definition of ICL (defined in terms of the model's loss) and show that the effect can be framed as a form of Bayesian inference. Our earlier work (Chan et al., 2022a) shows via experimentation with controlled synthetic datasets that, e.g., the bursty and Zipfian nature of text data could play an important role in the emergence of ICL. Many other works consider how ICL can be used to learn certain function classes using synthetic data, and how ICL may be implementing a form of gradient descent to accomplish this (Garg et al., 2022; Akyürek et al., 2022; von Oswald et al., 2022).

As with the aforementioned work, here we do not experiment directly with billion-parameter transformers trained on web-scale corpora. Rather, we use the setup we have shown previously to provide a useful testbed (Chapter 2).

3.7.2 Connections to larger models

Nonetheless, our work should be of relevance to studies that aim to derive ICL in increasingly small and compact text-based language models. By training language models of many different sizes on different amounts of data, Hoffmann et al. (2022) identified a relationship between the amount of training and model size that appeared compute-optimal with respect to the training loss. This correspondence motivated Chinchilla, a 70B-parameter language model whose downstream performance (on tasks that require ICL) was comparable to far larger models. More recently, Touvron et al. (2023a) took this further, "overtraining" smaller models to achieve lower training loss according to a given, inference-time, compute budget. Perez et al. (2021) point out that in such work, tasks that require ICL are typically included in the models' validation set in order to optimise hyperparameters directly for ICL. Our results indicate that, without appropriate validation and early stopping, training for a long time may eventually lead to ICL gradually disappearing, perhaps even without recognition from model developers. A large number of "classes" or Zipfian data distributions (both properties of natural language (Piantadosi, 2014)) can mitigate this by delaying the disappearance. Moreover, regularization may completely eliminate this risk. Indeed, Hoffmann et al. (2022) find that the use of optimizers with weight decay (Loshchilov and Hutter, 2017) leads to models that perform better, but this benefit only manifests at later stages of training (perhaps when ICL would otherwise decay).

Relatedly, Wei et al. (2023) find that "larger models do in-context learning differently": Specifically, it would seem that larger models exhibit a more general form of ICL, being able to adapt to flipped labels in context. As larger models are relatively less overtrained compared to smaller models, the transience of emergent ICL could provide a possible explanation for such phenomena: smaller models could be starting to lose some of their most general ICL capabilities, while larger models would need to be trained much longer to see similar effects.

3.7.3 Reproductions in other settings

Since the publication of our original work, numerous works have reproduced the transience phenomenon in a range of different settings, from modular arithmetic (He et al., 2024) to natural language (Anand et al., 2024). More recent work (Park et al., 2024; Nguyen and Reddy, 2024) has even expanded on our hypothesis of competition being the main cause of ICL fading away, lending more evidence to this claim.

3.8 Discussion

The key insight of this chapter is the empirical finding that, if in-context learning (ICL) emerges in a transformer network, it may not necessarily persist as the model continues to be trained. That is, while ICL is often emergent, it may be in many cases also be transient (Section 3.3). Importantly, this identifies emergent ICL as a *dynamical* phenomenon, rather than *asymptotic* one, motivating a different lens of study than previously taken and opening up some interesting questions:

Why does ICL emerge at all, if it is not asymptotically preferred? Since GPT-3 (Brown et al., 2020), the emergence of ICL has become an expected phenomenon in large-scale transformer models. Our experiments underline how curious it is that ICL emerges in such models given that it does not seem to be asymptotically preferred. We explore this question further in our subsequent work (Chapter 5).

Given that ICL does emerge, why does it fade in favor of IWL? On our training data, either ICL or IWL can reach good performance. While we do not know for certain why ICL emerges in a network, the question remains of why it fades if it is a viable strategy to solve the training task. Our results in Section 3.6 suggest that part of the explanation may lie in a *competition* between IWL and ICL circuits for resources in the transformer residual stream (Elhage et al., 2021). Under the assumption that IWL is asymptotically preferred, this competition would explain why ICL fades after emerging.

Why is IWL asymptotically preferred over ICL, when both solve the task? Induction heads (Olsson et al., 2022) are a possible circuit that may be responsible for ICL, and comprise of two interacting components: finding a match in-context, then copying some token forward. This "matching" operation requires matching any of 20 exemplars to each other. On the other hand, IWL relies on learning the exemplar-label mapping in-weights, which we show is feasible in Figure **??**. IWL could be seen as matching an exemplar to a given label, which is a 20-to-1 mapping, instead of a 20-to-20 mapping for ICL. The latter may be more challenging, though quicker to learn fuzzily (due to the algorithmic simplicity of ICL compared to IWL), which may give rise to the transient appearance of ICL before the "more reliable" IWL dominates asymptotically. Our results about "too much" in-class variation hurting ICL (Section 3.4.2) lends is consistent with this hypothesis. We explore this direction further in our subsequent work (Section 5.8).

How do we get ICL and IWL to co-exist asymptotically in the same model? Large language models clearly show a co-existence of ICL and IWL. In Section 2.3.5, we found that training on Zipfian data allows the network to perform ICL on rare classes, but learn common classes in-weights. Regularization was the best strategy we found to make ICL persistent, but it leads to little-or-no IWL. Given our results, the only effective way currently to get co-existence of ICL and IWL appears to be stopping training at the right time. Future work could investigate other factors that may restore co-existent ICL and IWL, even asymptotically.

Chapter 4

What needs to go right for an induction head? A mechanistic study of in-context learning circuits and their formation

This chapter is based on:

> **Aaditya K. Singh**, Ted Moskovitz, Felix Hill, Stephanie C. Y. Chan, and Andrew M. Saxe. What needs to go right for an induction head? A mechanistic study of in-context learning circuits and their formation. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024

Given that emergent ICL should be viewed as a *dynamical phenomenon*, we build a toolkit for causal interventions on dynamics and elucidate the underlying subcircuits that need to "go right" for ICL to emerge. These interacting subcircuits are responsible for the classic *phase change* (Olsson et al., 2022) associated with ICL emergence. My contributions to this work were comprehensive: building the artificial optogenetics framework, inventing the relevant methods, iterating on the toy model, conducting all the experiments, and writing the paper.

4.1 Introduction

In the previous chapter, we recast emergent ICL as a dynamical phenomenon, motivating a better understanding of the dynamics of its formation. Prior work (Olsson et al., 2022) has shown the induction circuit (Section 1.1.4) to be responsible for in-context learning. Notably, induction circuits (and corresponding ICL abilities) emerge in a *phase change* – a sudden drop of the loss. Prior work has yet to understand these dynamics – the closest is the work of Reddy (2023), who begin to address this question using progress measures. Popularized by Nanda et al. (2023), progress measures track correlational relationships between intermediate activations during training. Given the pitfalls of correlational approaches in mechanistic interpretability (Morcos et al., 2018; Belinkov, 2021), we instead pioneer a causal approach, termed *clamping*, which allows us to directly determine the circuits and dynamics that are causally affecting formation. Through these clamping experiments, we bring to light a new set of mechanisms governing learning dynamics.

Specifically, we take inspiration from the field of optogenetics in neuroscience, which allows precise causal manipulations of neural activity (Boyden et al., 2005), and develop a novel framework for causally modifying activations throughout model training. These causal-through-training clamping analyses allow us to easily study interactions between subcircuits and isolate the underlying factors that drive induction circuit formation. We utilize this framework to analyze induction circuit formation in transformers. To focus in on understanding the dynamical phenomenon, we restrict our attention in this chapter to 2-layer attention-only models (the standard setting for studying induction circuits; Olsson et al. (2022)) trained on a task that *requires* ICL (similar to the "ICL-only" data in Section 3.6).

As with any work towards a mechanistic understanding of dynamics, we first provide a thorough investigation of the induction circuits in a network trained on this task, shedding light on the dependence (and independence) of induction heads upon each other (Section 4.3). We find that multiple induction heads form, contributing additively to minimize the loss. Furthermore, we observe emergent redundancy, despite not applying regularization techniques such as dropout (Hinton et al., 2012), mirroring findings on larger-scale language models (Michel et al., 2019; Voita et al., 2019). We also find the wiring between induction heads and previous token heads to be many-to-many, rather than one-to-one. Next, we focus on the dynamics of formation (Section 4.4), where we use clamping (causal manipulations of activations throughout training) to identify three, smoothly evolving underlying subcircuits whose interaction may be causing the seemingly discontinuous phase change. Specifically, the third key evolving subcircuit is responsible for copying input labels to the output (highlighted in blue, Figure 4.1a), a function that was often believed to be easy (with most prior work focusing on the "match" operation instead, where an induction circuit has to find the right token to attend to). Finally, we show how data properties influence the timing of the phase change, and how this shift in induction circuit formation can be better understood by individually understanding the data-dependent formation of each of our identified subcircuits.

As a resource to the community, we open-source our artificial optogenetics framework as part of this overarching work's codebase at `https://github.com/aadityasingh/icl-dynamics`, providing a tool for modifying activations throughout training and conducting further causal analyses of transformers' circuit elements. Our work and tooling represents an important step in mechanistic understanding of training dynamics, applied here to induction circuit formation, and we hope it spurs further progress on understanding how different computations in transformers are learned.

4.2 Methods

4.2.1 Experimental setup

We train transformer models on a few-shot learning (FSL) task that necessitates in-context learning. As in our prior work, sequences are series of exemplar-label pairs, followed by a query exemplar (see Figure 4.1b). Exemplars come from the Omniglot dataset (Lake et al., 2015). Most of our experiments here use a simplified dataset with a random set of $C = 50$ classes and $E = 1$ exemplar per class, which we

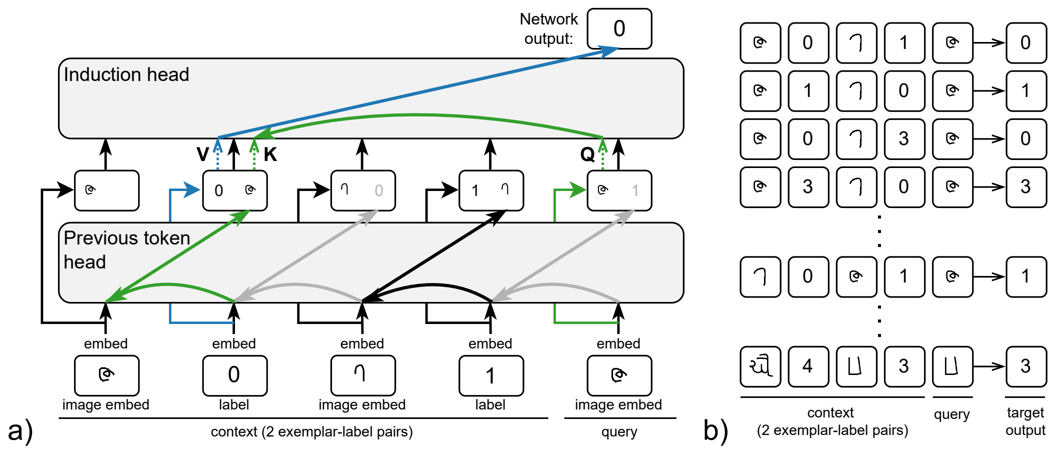


Figure 4.1: **a)** Schematic of an induction circuit, similar to Figure 1.1, except with "matching" (green) and "copying" (blue) operations highlighted. We also gray out the previous token head operation for copying label tokens. This behavior does not robustly emerge as it is not necessary since we only train on sequences of a fixed length (5 tokens long). **b)** Example training sequences built from the Omniglot dataset and inspired by classical few-shot meta-training (like the "ICL-only" data from Section 3.6). The context consists of two exemplar-label pairs, where the exemplars are from different classes. The query exemplar comes from the same class as one of the exemplars in context. The in-context labels are randomly chosen. Every exemplar can appear with every possible label in every possible position, forcing the transformer to use ICL to minimize the training loss. Validation sequences either use held out class exemplars or held out pairs of labels.

found sufficient to elicit the relevant phenomena (in this ICL-only setting). To obtain input embeddings, we feed Omniglot images through an ImageNet-pretrained, frozen Resnet18 encoder (He et al., 2015; Russakovsky et al., 2015).[1]

To isolate FSL capabilities, we use the standard meta-training setup where labels are randomized for each sequence. Specifically, each sequence can be viewed as a 2-way, 1-shot classification problem (see Figure 4.1b).[2] Labels are randomly selected from L one-hot labels. To obtain input embeddings, we use a standard learnt embedding layer.

In our default setting[3] ($C = 50, E = 1, L = 5$), we have a total of 78400 unique

[1]This is a departure from our previous work, which jointly trained a Resnet encoder for images. We made this change for simplicity, and since we found it did not affect results of the previous chapter.

[2]Note that, as a result, sequences are shorter than in our previous work (5 tokens instead of 17). We found this simplification useful for more mechanistic analysis.

[3]In Section 4.5, we vary these parameters to more deeply study the dependence of induction head formation on data properties.

training sequences (Appendix C.1).

Model details We train causal, 2-layer attention-only transformer models, as used by prior work (Olsson et al., 2022), to study induction circuit formation. Similar to our prior work, we only train to minimize the cross entropy loss on the last token – the predicted label for the query exemplar. We use RoPE (Su et al., 2021) for positional encoding[4] and a model dimension of 64, with 8 heads per layer. As is common for transformers outside of interpretability work, we are able to use LayerNorm (Ba et al., 2016) when reading in from the residual stream to an attention block – LayerNorm is often excluded in interpretability work because it complicates standard analyses (Elhage et al., 2021), but our causal framework works by modifying activations instead of inspecting weights or assuming linearity, circumventing the issues.

For optimization, we use Adam (Kingma and Ba, 2015) from the `optax` library (DeepMind et al., 2020), with parameters $\beta_1 = 0.9, \beta_2 = 0.999$. We use a constant learning rate of $1e-5$ and a batch size of 32 sequences.

Evaluation. To test generalization, we consider two test sets over held-out classes and relabelings. Specifically, for "test (exemplars)", we evaluate performance on a random, held-out set of $C_{test} = 100$ classes, which verifies the generality of the "match" operation of induction circuits. For "test (relabel)", we hold out and test on a fixed percentage (20%) of pairs of labels. This ensures that, while all labels are seen during training, not all pairs are. This test set verifies the generality of the "copy" operation of induction circuits.

4.2.2 An artificial-optogenetics framework

One of the contributions of this chapter is a novel training and analysis framework that easily exposes activations, and allows causal manipulations throughout training (as opposed to only after training, as in prior work). The framework enables a wide range of analyses—in this work, we use it to conduct targeted analyses of induction circuit formation via clamping. Our codebase is implemented in Equinox/JAX

[4]This is also different from our prior work, which used absolute positional encodings. We decided to switch to RoPE given its rising popularity in larger models.

```
def example_pattern_preserving_ablation(model, sequence, preserve, ablate):
  # preserve - list of (layer, head) pairs to preserve patterns on
  # ablate - list of (layer, head) pairs to ablate
  base_cache = model.call_with_all_aux(sequence, cache=None, cache_mask=None)
  cache = jax.tree_map(jnp.zeros_like, base_cache)
  cache_mask = jax.tree_map(partial(jnp.zeros_like, dtype=bool), base_cache)
  # preserve head patterns
  for layer, head in preserve:
    cache[layer][head]['attn_pattern'] = base_cache[layer][head]['attn_pattern']
    cache_mask[layer][head]['attn_pattern'] = True
  # perform other ablation, example: turning off heads
  for layer, head in ablate:
    cache[layer][head]['out'] = 0
    cache_mask[layer][head]['out'] = True
  return model(sequence, cache=cache, cache_mask=cache_mask)
```

Figure 4.2: Example pseudocode demonstrating a pattern preserving ablation using our framework.

(Kidger and Garcia, 2021; Bradbury et al., 2018), natively supports up to 50x speed-ups with `jax.jit`,[5] and is open-source.

In contrast to the standard practice of having modules with a forward function implemented as `__call__`, we have an underlying `call_with_all_aux` method which returns a pytree of all intermediate activations. The `__call__` method wraps `call_with_all_aux` and returns just the output. During the training process, `__call__` is used. For analysis, `call_with_all_aux` can be easily used to expose all intermediate activations.

To allow for editing/ablating activations throughout training (or just post-hoc, as done in prior work), `call_with_all_aux` accepts `cache` and `cache_mask` arguments, which have the same shape as the output pytree of `call_with_all_aux`. These caches can be used to insert activations into the network, e.g. clamping activations at particular values during training. Combined with the functional, automatic differentiation of JAX, these caches can be input- and model-dependent, while still allowing for proper gradient routing. For

[5]Specficially, on the hardward used for this chapter (a 2018 MacBookPro with a 6-Core Intel Core i7 processor and 16GB of RAM), standard training runs were sped up from $\approx$ 330 minutes to $\approx$ 7 minutes using `jax.jit`, which our framework natively supports. Though PyTorch had recently added `torch.compile` to impart similar functionality (PyTorch, 2022), interpretability frameworks based in PyTorch at the time of this work were not compatible with `torch.compile` due to their use of hooks (Cooney, 2023).

example, Figure 4.2 shows how easy it is to implement the "pattern-preserving[6] ablation" of prior work (Olsson et al., 2022). Importantly, our framework allows implementing such causal manipulations *throughout training*, which enables us to isolate the *dynamics* of subcircuit formation (Section 4.4.2). While prior work[7] was restricted to conducting mechanistic analyses on checkpoints from training, providing only correlational evidence, our framework allows *causal* interventions on learning dynamics (see Section 4.4.3).

4.3 Induction head emergence and diversity

We first establish that training transformers as described in Section 4.2.1 leads to ICL abilities and induction head emergence. Figure 4.3a shows loss dynamics. Initially, the transformer reaches a plateau (perhaps indicative of a saddle point) where its accuracy is $\approx$50% (a loss of $\log 2$). This corresponds to the network having learned to randomly select from the two labels in context,[8] rather than the full set of all possible labels ($L = 5$). Then, a sudden phase change in the loss leads to near-0 loss and near-perfect accuracy on the task (99.99%). At this point, we also observe strong generalization to unseen exemplars and unseen label pairs, indicating the transformer has learnt a general ICL mechanism and is not simply memorizing the training sequences.

We find that the phase change in the loss corresponds to the formation of induction circuits. We measure the "induction strength" of a head, a common progress measure, as a difference in attention weights from the query token: [attention to the correct in-context label token] - [attention to the incorrect in-context label token]. For example, on the sequence "A 0 B 1 A", this would be the difference [attention weight from 5th token to 2nd token] - [attention weight from 5th token to 4th token]. We see that the emergence of induction heads corresponds to the phase change in

[6]Here "patterns" are used to refer to the post-softmax attention scores from one token to others.

[7]We note that some prior work in vision (Ranadive et al., 2023) or masked language models (Chen et al., 2024) has attempted to causally manipulate learning, but primarily through the use of regularizing losses. Our framework permits more direct interventions (on activations) throughout training.

[8]The model tends to place equal weight on the two labels, rather than randomly putting high weight on one.

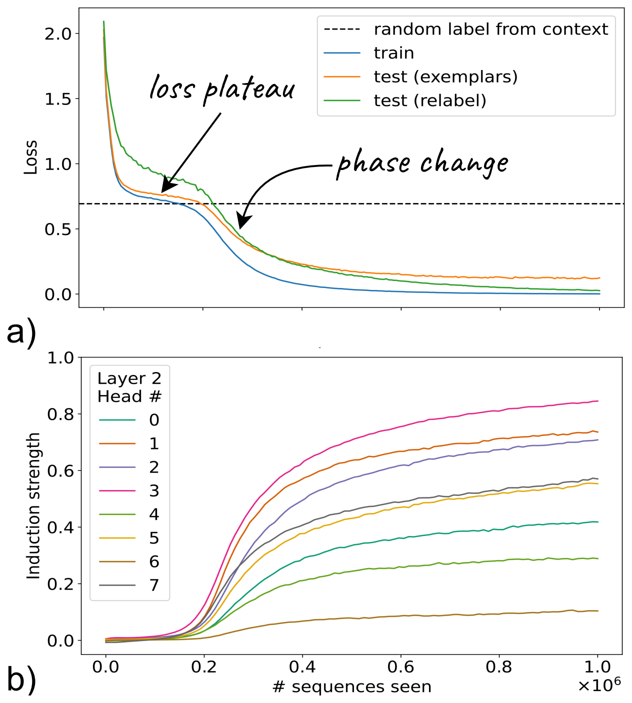


Figure 4.3: **a)** Train and test loss curves. Transformers exhibit strong generalization to unseen classes (orange) and label pairs (green). The loss dynamics reveal a plateau (which may be indicative of a saddle point), where the model is randomly guessing between the two labels present in context (so it has 50% accuracy, instead of the chance level of 20% when there are $L = 5$ labels). Then, there is a phase change in the loss which corresponds to the formation of induction circuits, reproducing the finding of Olsson et al. (2022). **b)** Induction head strength for each Layer 2 head plotted over time. Induction head strength is defined as the attention weight given to the correct label token minus that to the incorrect token. All heads appear to have some induction-like behavior, with Head 3 being the strongest and emerging first.

the loss.

To verify that Layer 2 induction heads instead of Layer 1 heads are primarily contributing to task performance, we use our artificial optogenetics framework to ablate the residual connection between Layer 1 heads and the output. We do this by setting Layer 2 activations (attention patterns and values vectors) to those from an un-ablated forward pass (keeping induction circuits intact) and then ablating Layer 1 head outputs. Such an ablation leads to a negligible change in loss and accuracy (which drops by 0.01% to 99.98%), indicating that Layer 1 heads are not responsible for task performance. When we apply this ablation across checkpoints, we see that earlier in training, Layer 1 heads are contributing to minimizing the loss (perhaps

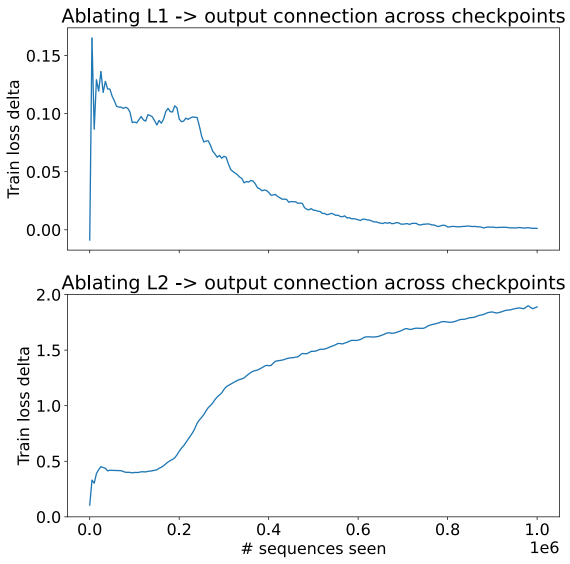


Figure 4.4: **a)** Change in loss after ablating connection from Layer 1 heads to output (unembedding) layer. Lower change indicates that Layer 1 head to output connections are less important. The ablation was applied independently to each checkpoint from the training run shown in Figure 4.3 and can thus be viewed as a progress measure. We can see that before the phase change, Layer 1 heads are contributing to the loss, but as induction heads form and the loss goes to 0 at the end of training, Layer 1 heads do not contribute to the output in a direct way. **b)** Same as a), but instead the connection from Layer 2 heads to output (unembedding) layer is ablated. Through training, importance of Layer 2 heads is increasing, especially after the phase change that corresponds to induction head formation.

through the use of skip-trigrams; Elhage et al., 2021), but no longer towards the end (Figure 4.4).

4.3.1 The additive nature of induction heads

One of the notable features of Figure 4.3b is the fact that many heads become strong induction heads. What purpose do all these heads serve? Are they necessary to solve the task?

To answer these questions, we consider two types of ablations of the fully trained network (Figure 4.5a). The first type of ablation is similar to prior mechanistic work (Olsson et al., 2022), where we ablate a given head and observe the effect on network performance (triangles, Figure 4.5a). However, we found that this type of ablation was not very informative, and even potentially misleading, as

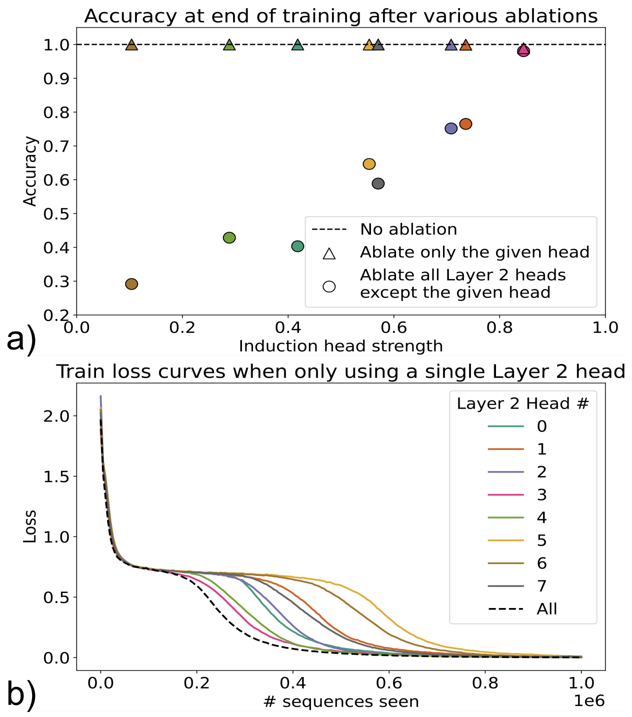


Figure 4.5: **a)** Effect of various ablations on accuracy. Ablating any single head (triangles) leads to virtually no decrease in task performance, with the exception of Head 3, which leads to a 1% decrease. Ablating all but a specific head (circles) isolates how useful that specific head is, which correlates well to the induction strength (x-axis), the metric from Figure 4.3b. Importantly, ablating Head 3 (pink triangle) performs very similar to ablating everything except Head 3 (pink circle), which indicates the other heads function additively, and together can make up for the deletion of Head 3. **b)** Training loss curves when training from scratch with only a single head from Layer 2 active (and the rest ablated). Black dotted line is the loss profile from the training run in Figure 4.3. Colors chosen to match Figure 4.3b. Each Layer 2 head on its own can learn to solve the task, though the timing of the phase change shifts and learning is slower.

ablating any head except the strongest leads to an almost unobservable decrease in task performance. Next, we considered an ablation inspired by work on head pruning (Michel et al., 2019), where all Layer 2 heads are turned off and only one head is turned on (circles, Figure 4.5a). This ablation allows us to identify the positive contribution of each head to the task. [9] As seen in Figure 4.5a, the latter ablation (circles) reveals a clearer picture of the importance of induction heads, showing

[9]There are some intuitive connections to the commonly used *logit lens* (nostalgebraist, 2020). However, we emphasize that our method relies on causal ablations only and takes into account LayerNorm, which is often a thorn for interpretability research. We measure task performance with the ablation applied, instead of a change in logits.

the correlation between induction head strength (which is calculated using attention scores), and task performance.

We see that despite Head 3 being able to mostly solve the task on its own (achieving an accuracy of 98%), other single heads can still achieve strong performance. Furthermore, heads seem to have an additive effect, most clearly demonstrated by comparing the pink triangle (which corresponds to ablating Head 3) and pink circle (which corresponds to only keeping Head 3). Both achieve around 98% accuracy, indicating that the other, weaker heads together can make up for the loss of Head 3. These results echo similar findings in head pruning (Michel et al., 2019) and layer-wise redundancy of language models (McGrath et al., 2023).[10] Our analysis builds on this work and cautions against the use of purely knock-out ablations in mechanistic analyses, which may overlook redundancies and incorrectly dismiss network pieces as not implementing a certain function.

4.3.2 Networks use additional capacity for faster training, even if it is not necessary

Given the additive qualities observed above, a natural question is—does the network need multiple induction heads to *learn* to solve the task? We know that we can ablate many heads and not hurt performance *at the end of training*, but perhaps these heads play a crucial role *during training*? To answer these question, we use our artificial optogenetics framework to ablate all Layer 2 heads but one *throughout training*. We do this for each Layer 2 head and show train loss curves in Figure 4.5b (all networks trained from the same random initialization). We find that final networks attain the same near-perfect train performance as the network trained with 8 heads in Layer 2 (black dotted line in Figure 4.5b, blue line in Figure 4.3), but with slightly slower dynamics (delayed phase change timing). Thus, it seems the network may make use of the additional capacity during training to learn faster, even though it is not strictly necessary for the task.

These experiments connect to notions of lottery tickets[11] (Frankle and Carbin,

[10]Such redundancy is also a common observation in neuroscience (Hennig et al., 2018).

[11]Lottery tickets typically refer to sub-networks sufficient to solve the task determined largely by

2018; Nanda, 2023b). Specifically, we note that different Layer 2 heads exhibit phase changes at different times (but all eventually learn to solve the task). Head 3 is the "quickest to learn" in Figure 4.5b, which may also be the reason it becomes the strongest when training with all heads (Figure 4.3b)—perhaps heads are racing to minimize the loss, similar to the mechanism in Saxe et al. (2022). Once Head 3 emerges, though, the ordering of phase changes in Figure 4.5b does not match emergence in the full network (Figure 4.3b)—e.g., Head 1 is the second to emerge and second strongest, but the third slowest when trained on its own. Additive interactions could be causing Head 1 to be learned sooner, due to the presence of Head 3 (we further explore such interactions in Appendix C.2).

4.3.3 Previous token heads influence induction heads in a many-to-many fashion

Prior work (Olsson et al., 2022) has stressed the two-layer nature of induction circuits, where induction heads in later layers rely on the output of previous token heads in earlier layers to attend to the correct token. By inspecting attention patterns, we identify three previous token heads in Layer 1: Heads 1, 2, and 5. Through a series of ablation analyses, we find that each of these heads is enough to elicit above-chance accuracy in at least one of the strongest induction heads in Layer 2 (see Figure C.3b). These results indicate a many-to-many wiring between previous token heads and induction heads, with previous token heads operating redundantly (just as induction heads operate redundantly to solve the task). Full details are provided in Appendix C.3.

4.4 Three interacting subcircuits give rise to the phase change in induction head formation

We now turn our attention to the dynamics of induction circuit formation. Prior work (Jermyn and Shlegeris, 2022) indicates that reverse-S phase changes in the loss are often due to multiple interacting components. Below, we delineate the necessary

initialization.

computations that comprise an induction circuit (calling back to Section 1.1.4), then study their formation dynamics in isolation using our artificial optogenetics framework.

4.4.1 Terminology for this section

For clarity, we isolate and define the 5 key computations involved in an induction circuit, as illustrated in Figures 1.1.4 and 4.6c. In Section 4.4.2, we will show that these 5 computations can be grouped into the 3 primary interacting subcircuits.

Step 1 (PT-attend) Layer 1 head attends to previous token.

Step 2 (PT-copy) Layer 1 head copies previous token value into the current token's residual stream.

Step 3 (Routing Q/K/V) Layer 2 head reads Q/K/V[12] from the correct subspaces of the residual stream: Q from the residual, K from the output of the Layer 1 head, and V from the residual or from other Layer 1 heads.[13]

Step 4 (IH-Match) Layer 2 head matches Q to "same" K.

Step 5 (IH-copy) Layer 2 copies the value to the output.

Steps 1 and 2 comprise what is canonically known as a previous token head, while Steps 3-5 form the induction head. The "match operation" (Figure 4.1a, blue) consists of Steps 1, 2, 3qk, 4. The "copying operation" (Figure 4.1a, green) consists of Steps 3v, 5.

4.4.2 Clamping computations to understand the causal effects of subcircuits on dynamics

Now that we have defined these computations, we ask: how do the learning dynamics of each of these computations causally influence the learning dynamics of the full model? Can we explain seemingly discontinuous qualitative phase changes, such as the induction circuit formation, in terms of subcircuits with smoother, exponential learning dynamics?

To motivate our analysis, we extend the toy model from Jermyn and Shlegeris

[12]Q/K/V are the query, key, and value in the attention computation (Section 1.1.3.

[13]Unlike Olsson et al. (2022), we observe non-trivial V-composition in our networks. See Appendix C.3 for details.

(2022) to the case of three interacting vectors.[14] Specifically, we have three vectors **a**, **b**, and **c** that are learnt using a simple mean-squared error loss, corresponding to learning their true values ($\mathbf{a}^*$, $\mathbf{b}^*$, and $\mathbf{c}^*$) via a tensor product:

$$\begin{aligned}\mathscr{L}(\mathbf{a},\mathbf{b},\mathbf{c}) &= \frac{1}{2}\,||\mathbf{a}^* \otimes \mathbf{b}^* \otimes \mathbf{c}^* - \mathbf{a} \otimes \mathbf{b} \otimes \mathbf{c}||_F^2 \\ &= \frac{1}{2}\sum_{i,j,k}\left(a_i^* b_j^* c_k^* - a_i b_j c_k\right)^2\end{aligned}$$

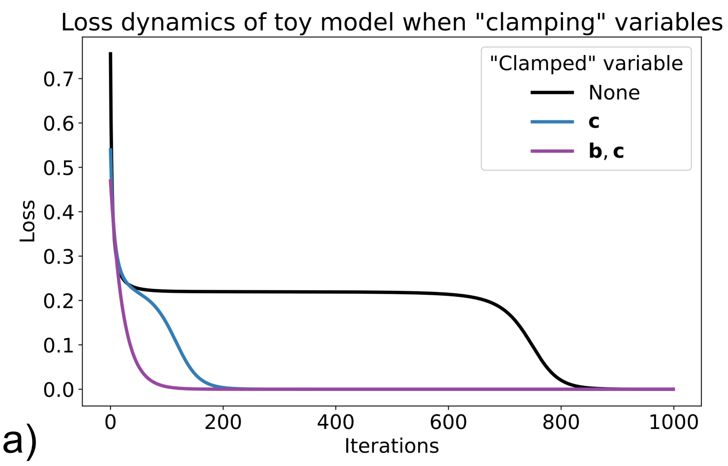
The interaction between the evolution of **a**, **b**, and **c** gives rise to a phase change in dynamics (black, Figure 4.6a) following a loss plateau caused by a saddle point (see proof in Appendix C.4). If we clamp **c** to its final value, learning is faster (blue, Figure 4.6a), but the co-evolution of **a** and **b** still results in a phase change. If we clamp **b** *and* **c** (purple, Figure 4.6a), we isolate a smooth exponential loss curves corresponding to the formation dynamics of **a**. This toy model gives us the intuition that phase changes may be caused by two or more subcircuits that, when learned on their own, evolve exponentially, but when co-evolving, induce phase changes. Furthermore, these subcircuits could be isolated by clamping all the other interacting components throughout training.

We apply these insights from the toy model to understanding the key underlying subcircuits in an induction circuit. Specifically, we aim to isolate subcircuits where each individual subcircuit does not exhibit a phase change in its learning dynamics, like variables $\mathbf{a}, \mathbf{b}$, and **c** in the toy model. We restrict ourselves to considering the induction circuit with induction head Layer 2 Head 3, a minimal setting where we observe a phase change in the loss dynamics (see Figure 4.5b). In Section 4.5, we will show how understanding these subcircuits can help us understand the data-dependence of phase change timing, affirming the importance of discovering smoothly-learned underlying subcircuits.

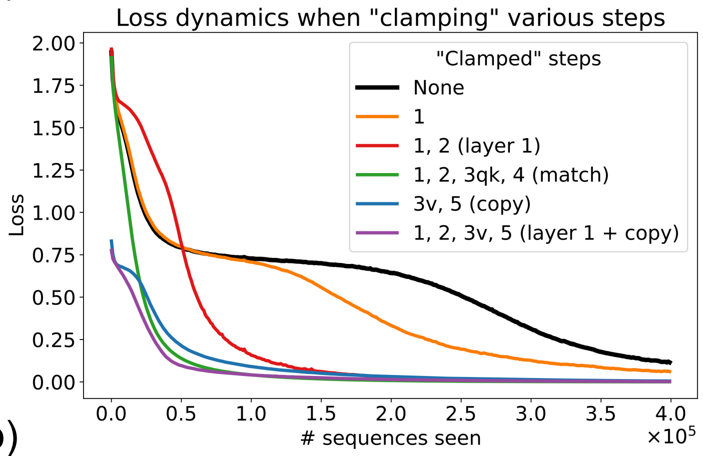
To isolate these subcircuits, we iteratively clamp the computations delineated in Section 4.4.1 at the activation level,[15] using our artificial optogenetics frame-

[14]Our tensor product formalism for analyzing sub-circuits could be extended beyond 3. We chose 3 as we ended up finding 3 primary subcircuits contributing to induction circuit formation. We also assume that $\mathbf{a}^*, \mathbf{b}^*, \mathbf{c}^* \neq \mathbf{0}$.

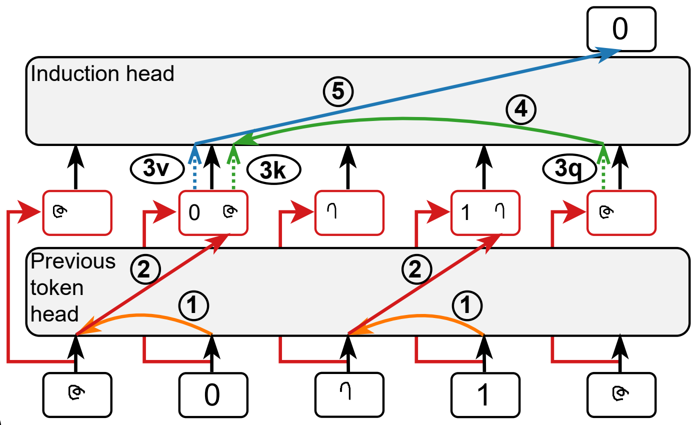
[15]Specifics of our implementation are provided in Appendix C.5.2 and why fixing weights to those



a)



b)



c)

Figure 4.6: **a)** Loss dynamics when clamping various variables in the toy model presented in Section 4.4.2. Black shows the learning dynamics when no variable is clamped. Only when all other interacting components ($\mathbf{b}, \mathbf{c}$) are clamped does the loss curve become exponential. **b)** Loss dynamics when clamping various computations outlined in Section 4.4.1. Black shows the training dynamics of the full network with nothing clamped. **c)** Induction circuit schematic (from Figure 4.1a), with computation steps labeled. Arrow colors chosen to illustrate which steps are additionally clamped.

work. Results are shown in Figure 4.6b. We start by clamping Step 1 (PT-attend), the previous token head attention pattern (an oft-used progress measure for IH formation). We train a network where one[16] of the Layer 1 heads is clamped from the start of training to a perfect PT-attend pattern. We see the loss dynamics still exhibit a phase change (orange), with a similar profile, indicating the rest of the computations (2-5) contain interacting subcircuits. Next, we clamp the full Layer 1 computation (PT-attend *and* PT-copy) and ensure that the superimposed residual and output of Layer 1 are disentangled (see Appendix C.5.2 for details). We find that the dynamic profile of the loss shifts substantially (red). While a phase change is still observed, it is much sharper, lasting only $\approx 7.5e4$ iterations (compared to $2e5$, black). This suggests that the formation of Layer 1 attention+output circuits is likely a key sub-component, but there are still interacting sub-components in Steps 3, 4, and 5.

We take this analysis further and clamp the entire match operation (green in Figures 4.1a, Figure 4.6), so only the copy operation (blue, Figure 4.1a) is being learned. We see this loss profile is smoothly exponential, suggesting there are no sub-components hidden within it. We then do the opposite, clamping the copy operation and focusing on the dynamics of match operation formation. We see a small saddle point followed by a quick phase change (lasting $\approx 0.5e5$ iterations), indicating that the match operation should be decomposed further. We do so by clamping both layer 1 and copy components (Steps 1, 2, 3v, 5), which results in a more exponential loss profile (purple, Figure 4.6).

Given these results, we believe the phase change is primarily determined by three interacting subcircuits:

Subcircuit A: Layer 1 Attending to previous token and copying it forward. Comprised of PT-attend and PT-copy (Steps 1 and 2).

Subcircuit B: IH QK Match Matching queries to keys in the induction head. Comprised of Routing Q, Routing K and IH-match (Steps 3qk and 4).

---

at the end of training is not sufficient (see Appendix C.5.1).

[16]We also conducted experiments where multiple previous token heads were provided and observed similar loss dynamics. See Appendix C.5.2.

Subcircuit C: Copy Copying of input label to output. Comprised of Routing V and IH-copy (Steps 3v and 5).

Prior work has often hypothesized the first and second components; our analysis verifies these intuitions and critically identifies the third interacting component through the use of *clamping* (causal manipulations of activations throughout training).

4.4.3 Contrasting clamping to progress measures

Here, we briefly contrast conclusions one may obtain from progress measures (Nanda et al., 2023; Reddy, 2023) to those from clamping. Progress measures track quantities through the normal training process, often using causal ablations of intermediate network checkpoints.[17] Despite this causal nature per checkpoint, we note that progress measures are actually *correlational* metrics, as the learning dynamics of the quantities they track may or may not be influencing the learning dynamics of the overall network. As such, we find our clamping method (Section 4.4.2) to be more useful for uncovering underlying subcircuits.

We first illustrate this with the toy model explored in Section 4.4.2. To track the evolution of "subcircuit" $\mathbf{a}$, we might consider various progress measures. The first problem we run into is that $\mathbf{a}, \mathbf{b}, \mathbf{c}$ are only specified up to scalars – if $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ is a solution, so is $(-\mathbf{a}, -\mathbf{b}, \mathbf{c})$. We find this to be an interesting toy analog to the rotations problem described in Appendix C.5.1. Given this, we might consider tracking the squared cosine distance between $\mathbf{a}$ and $\mathbf{a}^*$. Another progress measure we may consider, which offers a direct analog to clamping, is the loss if $\mathbf{b} = \mathbf{b}^*, \mathbf{c} = \mathbf{c}^*$.[18] Note the difference between this progress measure and clamping is that with the progress measure, we are considering the evolution of $\mathbf{a}$ during normal training and setting $\mathbf{b} = \mathbf{b}^*, \mathbf{c} = \mathbf{c}^*$ at every checkpoint. With clamping, we set $\mathbf{b} = \mathbf{b}^*, \mathbf{c} = \mathbf{c}^*$ throughout training and then consider the evolution of $\mathbf{a}$.

Results are shown in Figure 4.7. We find that both progress measures consid-

[17]Our results in Figure 4.3b and Figure 4.4 are examples of progress measures, with the latter involving a causal ablation per checkpoint.

[18]Specifically, we consider $\mathbf{b} = \pm\mathbf{b}^*, \mathbf{c} = \pm\mathbf{c}^*$ and pick the one that yields lower loss (this choice is made to avoid the scaling issue).

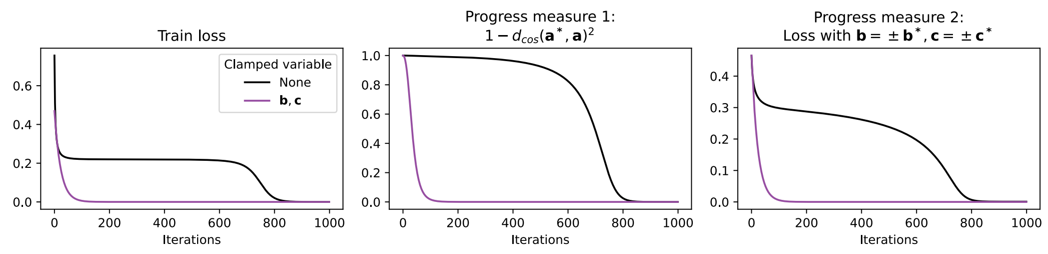


Figure 4.7: Progress measures vs clamping in a toy model. Black curve indicates standard training, as in Figure 4.6a. Leftmost plot shows the training loss. Right two plots show two different progress measures for the learning of a. We note that clamping (purple curve, leftmost plot) gives a clearer signal of the relevant subcircuit than progress measures in standard training (black curves, right two plots).

ered, cosine distance to true value and loss assuming $\mathbf{b}, \mathbf{c}$ fixed to those at the end of training, exhibit a phase change when observed in isolation (black curves, right two columns, Figure 4.7). On the other hand, as seen in Figure 4.6a and reproduced in the leftmost plot of Figure 4.7 in purple, the loss curve when training with clamping is smoothly exponential and easily interpretable.

We extend results from the toy model to our setting, and consider progress measures corresponding to two of our clamped experiments: that of clamping the previous token attention pattern (clamp Step 1, orange curve, Figure 4.6) and that of clamping the whole match operation (clamp Steps 1, 2, 3qk, and 4, green curve, Figure 4.6). We construct analogous progress measures by perfecting either the previous token head attention pattern or induction head attention pattern for each checkpoint in standard training and considering the loss. Figure 4.8 shows the results, with a reproduction of the relevant curves from Figure 4.6 for a side-by-side comparison. We can see that the progress measure is informative in the sense that it identifies the copy subcircuit (Steps 3v, 5) as quicker to learn than the subcircuit consisting of Steps 2, 3qkv, 4, and 5. However, when just looking at progress measures, both subcircuits appear to still emerge in a phase change, missing out on the crucial distinction between the two: the copy subcircuit (Steps 3v, 5) does not contain interacting components, whereas the other subcircuit (Steps 2, 3qkv, 4, and 5) does. These results show the benefit of the clamping approach for underlying exponentially forming subcircuits whose dynamics of formation causally affect the

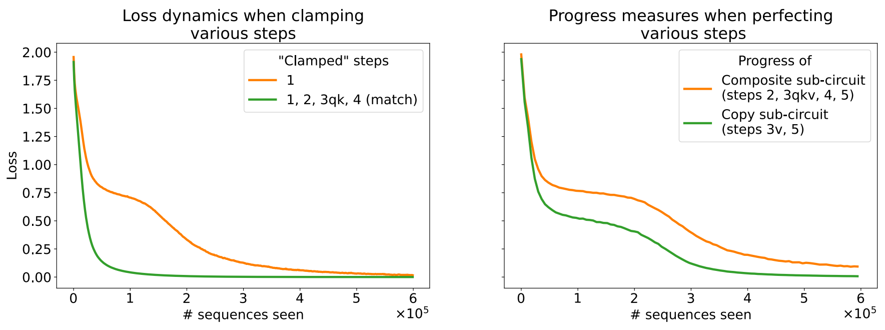


Figure 4.8: (Left) Our clamping method applied to studying formation of composite subcircuit consisting of Steps 2, 3qkv, 4, 5 (orange) and copy subcircuit consisting of Steps 3v, 5 (green). (Right) Progress measures tracking these same circuits. We can see that the clamping analysis more clearly shows the difference in formation dynamics of these two subcircuits, whereas the progress measure shows a phase change for both.

learning dynamics of the full network.

To conclude this section, we discuss pros and cons of progress measures versus our clamping approach. Progress measures have the benefit of only requiring access to network checkpoints after training, reducing the load of the mechanistic interpretability researcher. For cases of large language models, where training data is often not disclosed even for open-source models (Touvron et al., 2023b), this could be especially appealing. Furthermore, as each progress measure only involves forward passes on checkpoints, they may be easier to iterate on. On the other hand, progress measures, as demonstrated above, provide largely a correlational understanding of subcircuits giving rise to dynamical phenomena. Clamping experiments, though more complex, provide a more causal understanding of learning dynamics. We hope that our results and open-source artificial optogenetics framework encourages more researchers to consider these types of experiments.

In terms of scaling to larger models, the hope would be that isolated subcircuits evolve quickly enough that the cost is low enough. Specifically, one can view clamping experiments as requiring about 3x the cost of progress measures per iteration (due to the forwards and backwards passes). Thus, if the number of iterations needed to train clamped networks is substantially lower than that of the full network, clamping experiments may actually be cheaper than progress measures.

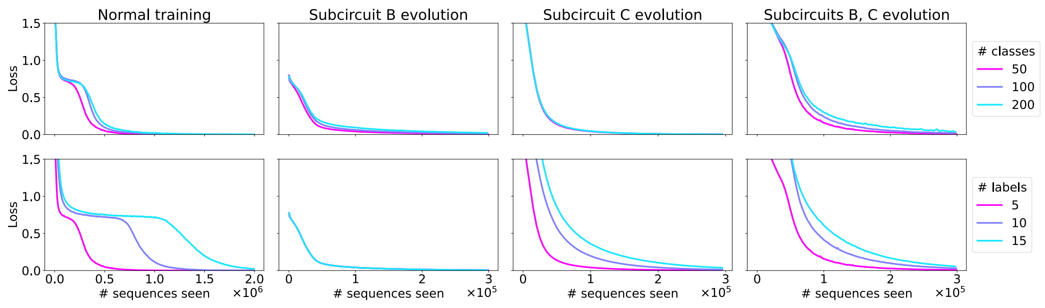


Figure 4.9: Data-dependent learning dynamics of the induction circuit and assorted subcircuits. Top row shows curves as the # of classes is increased, bottom row shows curves as the # of labels is increased. Left-most column shows training loss over time without any clamps on subcircuits. Middle two columns looks at the evolution of individual subcircuits identified in Section 4.4.2, by clamping the other two subcircuits. Right-most column shows evolution of a composite circuit (by only clamping Subcircuit A). Middle two plots clearly show how different subcircuits depend on different data properties, and how this can explain the overall difference in learning dynamics.

4.5 Subcircuits can explain data-dependent shifts in phase change timing

To demonstrate the relevance of the subcircuits we identify through clamping, we turn towards explaining changes in the timing of the phase change as we modify various data properties. Previous work (including our own; Chapter 2) has shown that such data properties can affect the dynamics and emergence of ICL (Reddy, 2023; Raparthy et al., 2023; Yu et al., 2023). Specifically, we consider two possible variations: increasing the number of classes C or the number of labels L.

In Figure 4.9, we find that increasing both the number of classes and number of labels lead to a delay in the formation of induction heads, mostly seen as a shift in the timing of the phase change (leftmost column, Figure 4.9). This may be because the task becomes more challenging due to the higher data diversity. To better understand what leads to this delay, we look at the effect of these data properties on the formation of some of the subcircuits identified in Section 4.4, specifically Subcircuits B and C (middle two columns, Figure 4.9). We find that increasing the number of classes makes learning IH QK Match (Subcircuit B) harder while not increasing the difficulty of Copy (Subcircuit C), thus implying that a delay in Subcircuit B formation causes the delayed phase change. On the other hand, we find that when increasing the number of labels, IH QK Match is learnt as quickly, while copying becomes way harder, indicating that a delay in Subcircuit C formation causes the delayed phase change. These results make intuitive sense—more classes may necessitate learning a higher precision QK matching operation (Subcircuit B), and more labels may necessitate learning a higher precision copying operation (Subcircuit C). Our analysis enables this rigorous decomposition of the phase change of the full model into delays of formation of interacting subcircuits.

To emphasize the importance of considering these subcircuits individually, we consider the joint evolution of both Subcircuits B and C (which we can track by clamping Subcircuit A, analogous to the red line in Figure 4.6b). This would correspond to just looking at an induction head's formation, when a network is provided with previous token heads (and disentangled outputs of Layer 1). We find that both # classes and # labels seems to affect these dynamics, thus not providing as much clarity on how these different properties might differentially affect learning dynamics. Decomposing into the individual subcircuits that we identified in Section 4.4.2 provides a better understanding of how data properties differentially increase the difficulty of induction circuit formation, leading to delayed phase changes.

Finally, we note that learning of the individual subcircuits is substantially faster than that of the overall dynamics. By considering data-dependent effects on individual subcircuits, we may be able to predict network behavior without having to train for long amounts of time, waiting for a phase change.

4.6 Related tooling for mechanistic interpretability

As a key part of this chapter is introducing a new open-source, JAX-based framework for mechanistic interpretability analyses, we briefly mention some other related frameworks here.[19] Primary open-source frameworks include Transformer-

[19]We limit here to the frameworks existing at the time of this work. Since publishing, we note that new frameworks such as `penzai` (Johnson, 2024) and `mishax` (Kramár, 2024), that are JAX-based, have also appeared.

Lens (Nanda and Bloom, 2022), `nnsight` (Fiotto-Kaufman)[20], and `pyvene` (Wu et al., 2024). All of these frameworks are built on PyTorch, allow studying and manipulating activations from model checkpionts, and natively support many open-source LLMs. In some cases (Fiotto-Kaufman), they even allow intervention on gradients, but it is unclear if any of these frameworks can be used to manipulate activations *during training with proper propagation of gradients*, which is a crucial and unique feature of our framework. Furthermore, while our library does not support as large a breadth of use cases, we hope the vibrant open-source JAX community will find a mechanistic interpretability framework based in JAX and natively supporting compilation useful (including major speedup benefits), similar to the uptake of the `Rust_circuit` library (Goldowsky-Dill et al., 2023) by Rust users. The only JAX framework at the time of completing this chapter was Tracr (Lindner et al., 2023), which allows researchers to quickly construct transformers that implement certain RASP programs: a complementary focus from TransformerLens, `nnsight`, `pyvene`, `Rust_circuit` and our JAX-based artificial optogenetics framework.

4.7 Discussion

In this chapter, we focused on understanding the dynamics of ICL emergence – specifically, the formation of induction circuits. To do this, we simplified down to a task that requires ICL, and we switched to using two-layer attention-only models, in line with prior mechanistic work (Olsson et al., 2022).

Through our analysis, we discovered multiple answers to what does and does not need to "go right" for an induction circuit. We found that induction heads operate additively, with multiple heads used to learn the ICL task more quickly despite not being necessary to solve it. This theme of redundancy carried to previous token heads, which we found to exhibit a many-to-many wiring pattern to induction heads. Next, we used the novel *clamping* methodology to identify three smoothly-evolving subcircuits whose interaction may explain the phase change in induction

[20]We note that an updated, more expansive version of `nnsight` was released (Fiotto-Kaufman et al., 2024) after the work in this chapter was completed.

circuit formation. Furthermore, this better understanding helped us explain data-distribution-dependent changes in phase change timing. Of specific interest here was the quicker timelines of subcircuit formation when isolated, which could serve as useful intuition for understanding the effects various data ablations in large model training may have on the learning dynamics, without having to train full networks on each data ablation from scratch. In terms of specific subcircuits we identified, our work demonstrated the crucial role of the copy subcircuit in explaining data-distributional dependent delays in phase change timing (bottom row, Figure 4.9). This added understanding may have implications for practitioners when selecting a vocabulary size for LLMs: while larger vocabulary sizes are often preferred due to their increased compression ratio (and thus longer effective context), they may make copying more challenging, thereby delaying induction head formation.

Beyond our specific results on induction heads, our work here contributes a JAX-based artificial optogenetics framework which supports compilation (yielding nearly a 50x speedup on our hardware, see Appendix **??**) and can be used to modify activations *during training with proper gradient flow*, a methodology we term *clamping*. This framework permits causal analyses of learning dynamics, whereas prior work mostly applied causal analyses to end networks or checkpoints from normal training (Section 4.4.3). We view our work as taking the first steps with this approach, and are excited to see how future work may combine our clamping methodology with other causal mechanistic interpretability techniques, such as path patching (Wang et al., 2022; Conmy et al., 2023) or causal mediation analysis (Vig et al., 2020; Geiger et al., 2021; Cao et al., 2021; Geva et al., 2023; Finlayson et al., 2021), throughout training for better understanding of transformer circuit learning dynamics.

In the next chapter, we will apply this artificial optogenetics framework, new-found knowledge of subcircuits, and clamping methodology to understand the dynamics of ICL emergence and transience in the setups from our previous work (Chapters 2 and 3), where ICL is not necessary for solving the task.

Chapter 5

Strategy coopetition explains the emergence and transience of in-context learning

This chapter is based on:

> **Aaditya K. Singh**, Ted Moskovitz, Sara Dragutinovic, Felix Hill, Stephanie C. Y. Chan, and Andrew M. Saxe. Strategy coopetition explains the emergence and transience of in-context learning, 2025. URL `https://arxiv.org/abs/2503.05631`

Equipped with a toolkit for understanding dynamics (from Chapter 4), we tackle the question of why ICL emerges at all, if only to fade asymptotically. We find a surprising hybrid asymptotic strategy, context-constrained in-weights learning (CIWL), that has remarkable *cooperative* interactions with ICL, leading to its transient emergence. My contributions to this work were comprehensive: conceiving the ideas and hypotheses, conducting all experiments, developing the minimal mathematical model, and writing the paper.

5.1 Introduction

In Chapter 3, we found that ICL is often transient, motivating a need to understand the *dynamics* of ICL emergence (and transience). ICL is often viewed to be in *competition* with other strategies (as suggested in Section 3.6 and supported by subsequent external work (Park et al., 2024; Nguyen and Reddy, 2024)), such as IWL, with the tradeoff thought to be modulated by data properties (Chapter 2), model size (Wei et al., 2023), and/or training time (Chapter 3). While competition may explain why ICL gives way to other strategies through the course of training, the question remains: why does ICL emerge in the first place (if only to fade away)?

In this chapter, we aim to extend the *mechanistic* understanding of ICL, which currently focuses on induction heads (Olsson et al., 2022) and their emergence dynamics (Chapter 4), to a richer dynamical setting involving multiple strategies cycling in and out over the course of learning. To do so, we reproduce our key transience result with a 2-layer attention-only transformer, enabling mechanistic study. We extend our behavioral evaluators (from Sections 2.2.3 and 3.2.1) and find that the asymptotic strategy after the disappearance of ICL is not pure in-weights learning. Rather, it is a surprising hybrid strategy that we term context-constrained in-weights learning (CIWL, Section 5.5). The implementation of CIWL takes the form of skip-trigrams (Elhage et al., 2021) distributed across multiple heads in a form of superposition (Elhage et al., 2022). Perhaps even more remarkably, we find that even though CIWL dominates over ICL asymptotically, both strategies *share* critical sub-circuits (Section 5.6.1), indicating cooperative dynamics between these seemingly competitive mechanisms—a phenomenon we term "strategy coopetition." We borrow the term "coopetition" from game theory, where it describes situations where competitors simultaneously cooperate and compete with each other.[1] Cooperation enables the emergence of ICL (despite it not being asymptotically preferred), while competition leads to its eventual transience (as we found previously in Section 3.6). Notably, ICL emergence can only occur when CIWL is not fully formed (Section 5.6.4), which may explain the tradeoff we saw in Chapter 2: under certain data properties, CIWL forms too "quickly," preventing ICL emergence.

We formalize our intuitions into a minimal mathematical model capturing coopetition dynamics and explaining the transience behavior. Our model moti-

[1]A classic historical examples is Hollywood studios in the early 20th century, who competed aggressively for talent and audiences, but also collaborated to establish industry-wide standards and jointly negotiate with labor unions.

vates further experiments to modulate the tradeoff between strategies through learning, like reducing the asymptotic bias towards one strategy via data properties. For the case of ICL vs CIWL, we find that matching context and query exemplars removes the asymptotic bias towards CIWL and leads to the persistence of the "faster" ICL strategy (relating to some of the prior intuitions we had in Sections 2.3.2,3.4.2, and 3.8).

This chapter represents a step forward in understanding how different strategies trade off during learning, through mechanistic investigations on the transience of ICL.

5.2 Experimental setup

5.2.1 Training details

Like Chapter 4, here we train 2-layer attention-only transformers (Vaswani et al., 2017; Elhage et al., 2021) to enable mechanistic study. We use $d_{model} = 64$, with 8 heads per layer and learned absolute positional embeddings.[2] We used the Adam optimizer (Kingma and Ba, 2015) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, a learning rate of 10^{-5}, and a batch size of 32 sequences. All models were trained in JAX (Bradbury et al., 2018), with code available in the Github repo: `https://github.com/aadityasingh/icl-dynamics`.

5.2.2 Dataset

For the task, we use a simplified version of the task in Section 3.2.1, merging with some of the simplifications in Section 4.2.1. Specifically, we reduce the context to just two exemplar-label pairs, one from the same class as the query, one from a distractor class (this maintains the p(bursty=1) property). Exemplar-label mappings are fixed through training (as in Chapters 2 and 3), allowing the model to use ICL or IWL strategies. All experiments in this chapter use a random 12800 classes for training, using rotations and flips to augment Omniglot (Lake et al., 2015) classes as first introduced in Section 2.3.4. In Appendix D.1.3, we also considered using

[2]We considered various alternative architectures, such as those with MLPs or RoPE (Su et al., 2021), in Appendix D.1.

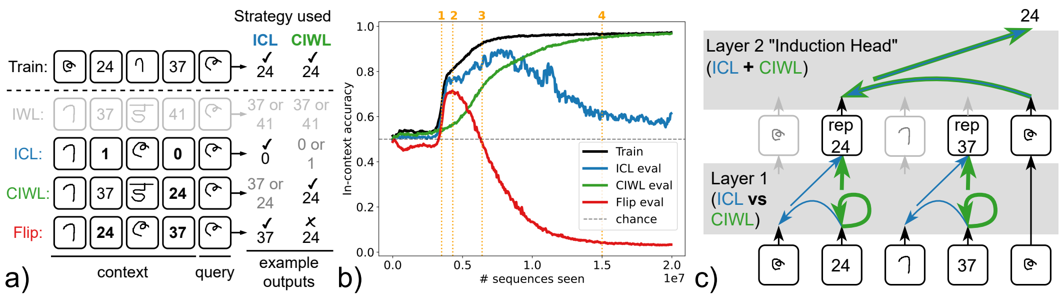


Figure 5.1: **(a)** Example sequences seen during training and evaluation. Training data is "bursty", enabling both in-context and in-weights strategies (the context always contains an exemplar from the same class as the query, but also exemplar-label mappings are fixed throughout training). Evaluation sequences (below dotted line) build on those from Section 2.2.3 and are designed to measure the presence of different strategies. ICL relies on the exemplar-label mapping in context. IWL depends solely on in-weights information. CIWL requires the correct label in context, but not the query exemplar. The Flip evaluator measures the balance between ICL and CIWL (1.0 means pure ICL, 0.0 means pure CIWL). Bolding indicates OOD exemplar-label pairings. Grayed outputs indicate random selection between the two in-context labels. **(b)** Accuracy on sequences from (a), over the course of training. "In-context accuracy" is computed by restricting the network's outputs to the two labels present in context—this ensures the same chance level (0.5) for all plotted evaluators. ICL transience is clearly visible in blue. IWL is not shown, as we found little-to-no IWL in the networks (Section 5.4). We annotate four points: 1. the formation of Layer 2 circuits, the canonical "induction head"; 2. ICL strategy dominates network output, as evidenced by peak in the Flip evaluator (red); 3. CIWL strategy matches strength of ICL, as indicated by 50% performance on Flip evaluator; 4. CIWL strategy dominates network output, leading Flip evaluator (red) to be 0 and CIWL evaluator (green) to be 1. **(c)** Illustration of competitive (Layer 1) and cooperative (Layer 2) interactions we find between ICL and CIWL strategies. Both strategies are present in varying amounts through training, as represented by the varying line weights in the Layer 1 circuits: when Layer 1 acts as previous token heads, the network exhibits ICL, but when Layer 1 heads attend to self, the network exhibits CIWL. Crucially, the computation in Layer 2 remains largely unchanged after its initial formation, despite the strategy switch from ICL to CIWL.

different #'s of classes or exemplars, observing similar modulations to Chapter 3 for the duration, timing, and magnitude of the transience effect (even with the 2-layer attention-only model).

5.2.3 Evaluators

To measure in-context and in-weights strategies, we consider four out-of-distribution evaluation sets (Figure 5.1a).

In the "ICL" evaluator, we replace labels from training with 0 and 1. This invalidates any exemplar-label mappings previously stored in weights. To perform above chance, the model is instead forced to use the mappings provided *in context*, i.e. an *in-context learning (ICL)* strategy.

In the "IWL" evaluator, the query exemplar (and its label) does not appear in the two context pairs, thus preventing the model from using context to perform the task. This forces the model to use knoweldge stored in weights, i.e. a pure in-weights learning (IWL) strategy. Performance on this evaluator barely rises above chance (Appendix 5.4).

The "CIWL" evaluator is like the IWL evaluator, except that while a matching exemplar to the query is not in context, the correct label *is* in context. We created this evaluator as we noticed that accuracy on the IWL evaluator stays low, even when training 12-layer networks in Section 3.3 (Figure 3.2b). We realized that IWL sequences are out-of-distribution, as when training on p(bursty)=1 data (as we do here and in Chapter 3) every training sequence contains the correct label in context. This evaluator explicitly permits a *mixed* strategy where in-context *label information* can be combined with in-weights information. We refer to the strategy that achieves above chance on this evaluator (but chance on the IWL evaluator) as *context-constrained in-weights learning (CIWL)*. It requires the correct label token in context, but not the full exemplar-label pairing.

Finally, the "Flip" evaluator can be seen as testing for the model's preference between ICL and CIWL strategies. The two context exemplars have their labels flipped relative to training (e.g. if exemplars X and Y were trained with label mappings X:24 Y:25, the in-context mappings would instead be X:25 Y:24 for this evaluator). The query comes from one of those two classes. If the network prefers the ICL strategy, accuracy on this evaluator would be 1. If the network prefers CIWL, accuracy would be 0, as it would instead output the label that was paired with the query exemplar during training. This evaluator is especially useful in measuring which strategy is *dominant* at each point in training (even when both strategies are present in some form).

5.3 Verifying usefulness of the small model

Figure 5.1b shows a reproduction of the key transience phenomenon (Chapter 3) in the simplified setting of this chapter (with a 2-layer attention-only model, and using sequences of length 5 tokens for training and evaluation as detailed in Section 5.2). Figure 5.2 shows an extended figure (with even longer training). ICL emerges and then disappears, as evidenced by above-chance performance on the ICL evaluator (blue). The disappearance of ICL corresponds to a rise in accuracy on the CIWL evaluator (green), indicating that the network is somehow using *just the label information* from context, in combination with some form of in-weights information, to get the right answer.

Notably, there is a significant period of time where ICL and CIWL co-exist (from $\sim$3e6 to $\sim$2e7 sequences seen). During this time, the balance between the two strategies shifts from ICL to CIWL, as evidenced by the decrease in the Flip evaluator (red). Early in training, ICL dominates (annot. 2). Then, ICL and CIWL are roughly balanced (annot. 3), before ICL fades and CIWL dominates (annot. 4). This corresponds to a switch in the network's behavior, which first bases its output on the exemplar-label mappings from context, and then eventually to the exemplar-label mappings from training (in combination with the label provided in context).

5.4 Networks barely exhibit *pure* in-weights learning (IWL)

In the previous section, we did not consider pure in-weights strategies, since it turns out they barely contribut to network behavior.

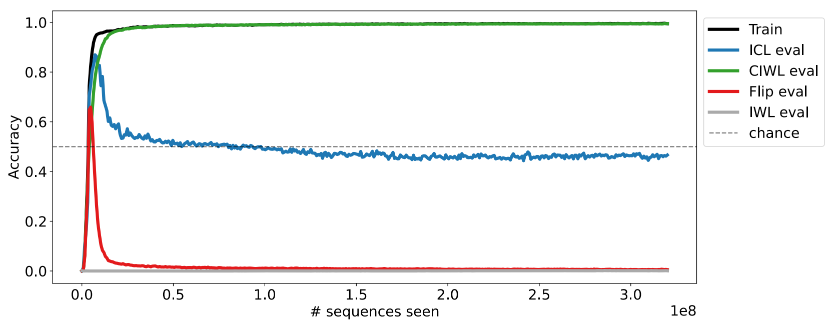


Figure 5.2: Same run as Figure 5.1b, but run 16x as long. We plot accuracy on the y-axis here (across all 12800 labels) as opposed to in-context accuracy to more compellingly demonstrate saturation of CIWL (green) and minimal learning of IWL (gray). This plot lets us include that networks are in fact not learning a pure IWL mechanism, but rather a context-constrained one which requires the correct label in context.

Figure 5.2 depicts accuracy on a pure IWL evaluator (Figure 5.1a, gray), which differs from the CIWL evaluator in that the correct label token does not appear in context. As in Figure 3.2b, we find that networks perform relatively poorly on this evaluator, instead employing a CIWL strategy (as evidenced by the delta between green and gray curves in Figure 5.2).

That said, the performance on the IWL evaluator at the end of Figure 5.2 is 0.08%, which is above chance level (which would be $1/12800 \approx 0.008\%$), indicating that the network may have picked up a tiny amount of pure IWL (or it could be noise). This prompted us to investigate further, and we did uncover a very minor in-weights mechanism.

Specifically, if we ablate all attention heads (so that the network is just the embedding $\rightarrow$ unembedding connection), we find that in-context accuracy on CIWL sequences is 62.46% (compared to 98.74% with no ablation). With this no-heads ablation, the network cannot actually attend to any tokens in context, so the "in-context accuracy" can be interpreted as "how likely is the model to output the correct class over a given random class." This value (62.46%) being above chance (50%) indicates the model is able to do this comparison somewhat accurately using just the embedding $\rightarrow$ unembedding pathway.

To make sure this pure IWL mechanism is not playing a key role, we consider the opposite ablation: zero-out the embedding $\to$ unembedding pathway, and leave the rest of the network unchanged. We do this by using a pattern-and-value preserving ablation, using two passes through the network. In the first pass, no intervention is used, and patterns and values are cached. In the second pass, we 0 out the embeddings, but use the cached activations for the attention heads, having the intended effect. This ablation leads to a mere 2% drop in accuracy (98.74% to 96.72%), indicating that the embedding $\to$ unembedding pathway is quite auxiliary compared to the computations performed by the attention heads. Moving forwards, we will focus on the computation of the attention heads, but note here that networks do a small amount of *pure* IWL, even when trained on “bursty data.”

5.5 The asymptotic “context-constrained in-weights learning” (CIWL) mechanism

Prior work (e.g., Anand et al., 2024; Nguyen and Reddy, 2024; Chan et al., 2024) has contrasted ICL to “pure” IWL strategies that are completely independent of context. However, we found above that the dominant asymptotic mechanism in our networks is dependent on in-weights information but also *context-constrained*, requiring the presence of the correct label in context, perhaps more analogous to notions of task recognition (e.g., Wang et al., 2024). Here, we uncover a possible mechanism for these strategies, in our simplified setting.

5.5.1 Layer 2 heads are skip-trigram-copiers

While label copying might appear to be a separate additive mechanism, we actually find that CIWL is implemented through the use of attention heads in Layer 2 (L2), which act as skip-trigram-copiers (Elhage et al., 2021). This mechanism attends to the correct label, and then copies it forward to the output: “... [label] ... [query] $\to$ [label].”

Figures 5.3a,b show attention patterns on the CIWL evaluator. At the end of training, the model correctly attends from the query exemplar to the in-context label that was paired with the query during training (regardless of which exemplar

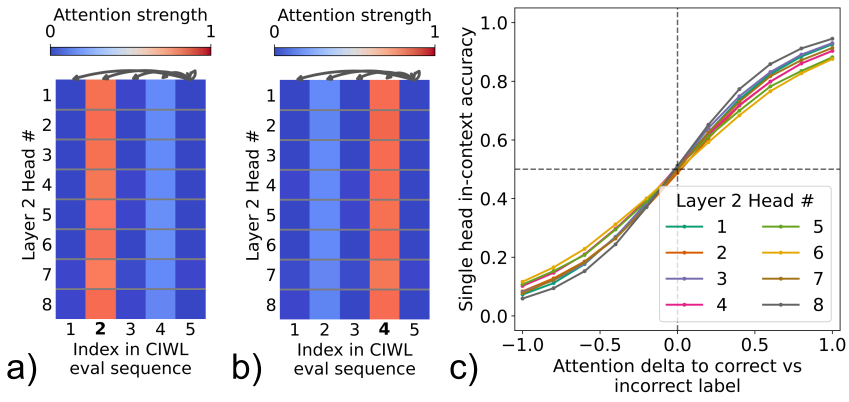


Figure 5.3: CIWL strategy is implemented via skip-trigram-like mechanisms in Layer 2 (L2), with substantial K- and V- composition to Layer 1 (L1). **(a)** Average attention patterns for L2 heads at the end of training. Attention is measured from the query token (index 5) to each token in context. It is computed over CIWL sequences where the correct label is at **index 2**. We see that, at the end of training, L2 heads attend to the correct label, regardless of what exemplar it is paired with in context. **(b)** Same as (a), except with CIWL sequences where the correct label is at **index 4**. **(c)** Task performance as a function of clamped attention delta to correct vs. incorrect label token, calculated over 5000 CIWL sequences, when only the given head is active. CIWL accuracy increases as attention to the correct label increases.

that label currently appears with in context). We find such trends on in-distribution training sequences as well as OOD CIWL and Flip evaluators. Skip-trigram heads only appear in L2, with Layer 1 (L1) heads not contributing directly to the output: ablating L1 $\rightarrow$ output connections, via the pattern preserving method used in Figure 4.4, results in a negligible drop in accuracy (98.7% to 98.5%) on the CIWL evaluator.

The second part of this mechanism is the copying forward of the label from context to output. We demonstrate this through the use of causal ablations: we clamp the attention pattern for a given head to give a certain weight to the "correct" label token and the remaining weight to the "incorrect" label token. When considering each L2 head in isolation,[3] we find that modulating this weight is nearly perfectly correlated with output predictions (Figure 5.3c), causally indicating that each L2 head is serving to copy input labels to the output. The combination of attending

[3]This is performed by taking the output of a single L2 head after our clamp, and feeding it through the rest of the network, similar to the logit lens (nostalgebraist, 2020) but taking into account the pre-unembedding LayerNorm.

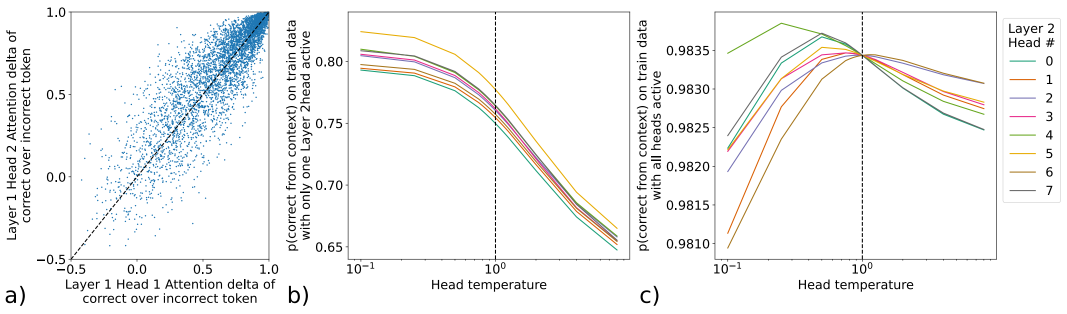


Figure 5.4: Preliminary evidence of superposition. **(a)** While average attention patterns across heads look identical (Figures 5.3a,b), patterns on individual points can be quite different (e.g., shown for Head 1 and 2, where each point in the plot represents attention deltas of these two heads for a single evaluation sequence). **(b)** Decreasing the temperature of any Layer 2 head's attention operation, if it is the only Layer 2 head active, increases performance. **(c)** The same temperature decrease does not as consistently affect performance if all heads are active, indicating non-additive interactions.

to the correct label and copying it forward give rise to the CIWL mechanism.

5.5.2 Layer 2 heads may be acting in superposition

Our results so far on the CIWL mechanism have some curious features:

1. All heads appear to be doing the same thing on average (Figures 5.3a,b).
2. All heads are *imperfect* on average—attention to the correct token never converges to 1 (instead plateauing around 0.8).

Here, we lay out preliminary evidence for how these heads may be acting in *superposition*. Our hypothesis was inspired by Elhage et al. (2022), who find that transformer MLP hidden layers may represent sparse features in superposition. The skip trigrams our network is learning are sparse (each sequence only requires comparing 2), and numerous (12800 classes), while the network weights that can be used to represent these are limited (eight 8-dimensional heads).

First, we find that while heads have the same attention patterns on average, their patterns on a given sequence can be quite different (Figure 5.4a). This would point to some form of distributed computation (which is hidden if only considering average patterns).

The most compelling evidence we find is by considering the interactions between different heads, and their imperfect average attention. If we consider a head in isolation (with all the other heads ablated), we can ask if the imperfection is “optimal.” Specifically, we consider artificially decreasing the temperature of the softmax attention on this head,[4] to see if its performance[5] is better or worse. We see in Figure 5.4b that uniformly decreasing the temperature improves performance, which begs the question: Why does the network not learn to do this?

The answer comes from considering how heads interact. If we leave all heads active and then decrease the temperature of a head, we find that overall performance often *goes down*. Each head on its own gets better at solving the task, but the inter-head interactions can often make the network overall worse! These non-additive interactions hint at a tight coupling between heads, and the notion that heads are meaningfully exploiting their dynamic range of attention (instead of the simplified, binary notion of “heads learn to attend to the correct token”). These results are in stark contrast to the additivity of heads we observed previously (Section 4.3.1), perhaps due to the much larger dataset being trained on (12800 classes instead of 50, and 20 exemplars per class instead of 1). We suggest this difference in head structure is due to a form of superposition.

We conclude by noting that this suggestion is somewhat speculative, backed by preliminary evidence. We hope this heads-in-superposition hypothesis, and preliminary evidence supporting it, can motivate further work on understanding individual attention head function.

5.5.3 Layer 1 engages in K- and V- composition, despite not being necessary

While L1 heads do not directly influence network output, ablating them completely (without preserving patterns and values in L2) leads to a big drop in performance. This indicates substantial composition (Elhage et al., 2021) between the two layers

[4]Note when conducting these experiments, we artificially also only allow the head to attend to label tokens. We know this is what they do in practice, and we wanted to avoid high temperature performance from being artificially hurt by increased attention to non-label tokens.

[5]By performance here, we mean performance on the task if only this head is active. Alternatively, this value can be viewed as a readout of the contribution of this head to the output..

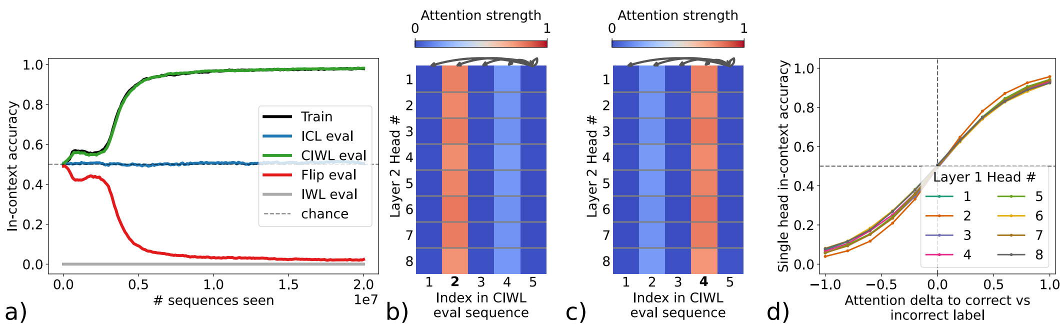


Figure 5.5: A 1-layer transformer trained on our bursty training data can learn skip-trigram-copiers, just as well as the 2-layer model. **(a)** Metrics indicate that the model learns a CIWL strategy. Notably, ICL does not emerge, as we would expect for a 1-layer transformer (Elhage et al., 2021). **(b-c)** Analogous plot to Figures 5.3a,b showing that all heads learn attention from query token (index 5) to the correct token in-context (index 2 or 4 for b, c respectively). **(d)** Analogous plot to Figure 5.3c showing that all heads are copiers.

(i.e. L2 attention heads are dependent on inputs from L1), despite the fact that skip-trigrams can be implemented with just a single layer. Figure 5.5 shows that a 1-layer model can learn CIWL using skip-trigram-like circuits, in line with prior work that describes 1-layer mechanisms for skip-trigrams (Elhage et al., 2021).

To dig into the specific composition between the two layers, we consider various ablations of L1 head outputs on the CIWL evaluator data, while preserving some combination of values, keys, or queries in L2. Recall that accuracy at the end of training on this evaluator is 98.7%, with chance level being 50%. When we preserve everything but values, performance drops to 59.3%, indicating L1 outputs influence L2 values in a crucial way (known as V-composition).[6] When we preserve everything but keys, performance drops to 57.5%, indicating K-composition. When we preserve everything but queries, performance drops to 95.4%, indicating a very small amount of Q-composition (if at all). Thus, the primary reliance of L2 on L1 is in the calculation of L2 *keys and values*, begging the question: Given that a one-layer network could implement skip-trigram-copiers, why does this observed K- and V- composition between layers emerge?

[6]As defined by Elhage et al. (2021), the degree of k-, q-, and v- composition is the degree that a previous layer contributes to keys, queries, and values (respectively) of a following layer.

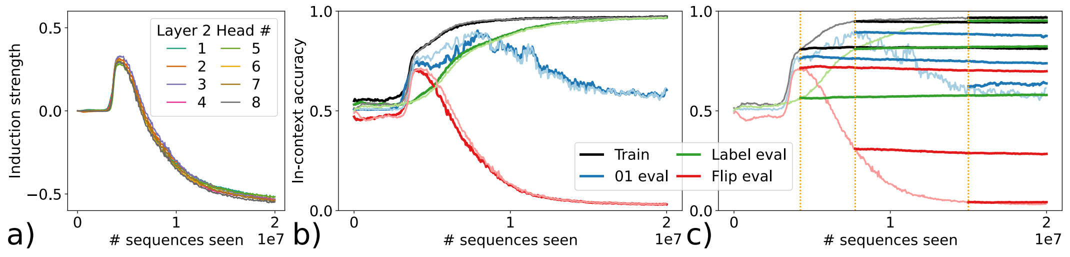


Figure 5.6: The transience of emergent ICL is driven by changes in the function of Layer 1. **(a)** Average induction strength (attention delta to correct ICL token vs incorrect ICL token) on Flip eval data of each Layer 2 head, through the course of training. We see induction circuits emerge then flip, matching the end-of-training attention patterns shown in Figure 5.3. **(b)** For each available checkpoint, we fix Layer 2 weights to be those from the end of training, and plot performance on each of our evaluators. Using the Layer 2 weights from the end of training (darker curves) reproduces the original behavior (lighter curves; matches Figure 5.1b) at all points in training after the initial flat portion. This indicates that Layer 2 is not meaningfully changing during the transition from ICL to CIWL. **(c)** For each available checkpoint, we fix the Layer 1 weights to those from a specific checkpoint (marked by the dotted orange vertical lines). After the Layer 1 weights are fixed in this way (darker curves), network behavior does not change, as evidenced by the flat lines on all the data we considered.

5.6 ICL emerges, despite not being asymptotic, due to cooperative interactions with CIWL

Induction circuits (which typically underpin ICL strategies, c.f., Olsson et al. (2022)) also show characteristic K- and V- composition, and seem to be present during the transient emergence of ICL in our experiments (Figure 5.6a). Could the K- and V- composition for the two strategies be related? Do these seemingly competing strategies actually *share* circuit elements? Could that be why ICL emerges, despite not being asymptotically preferred? In this section, we find the answer is, remarkably, yes.

5.6.1 The ICL to CIWL transition is explained by L2 reuse and L1 dynamics

Induction circuits are typically comprised of a "previous token" head in the 1st layer that copies information from the previous token into the next token, and an "induction head" in the 2nd layer that uses the 1st layer to find tokens preceded by the present token. This enables a common form of ICL: "[A*][B*] ... [A] $\rightarrow$ [B]"

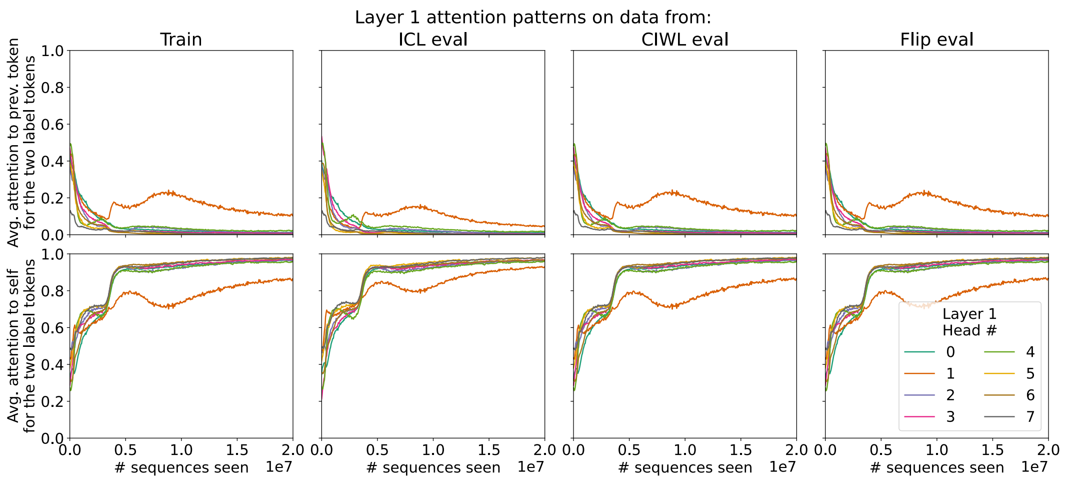


Figure 5.7: Per-Layer-1-head average attention patterns over different data subsets through the course of training. Top row depicts average attention from a label token (at index 2,4) to the previous token (index 1, 3, respectively). Bottom row depicts average attention from a label token (at index 2, 4) to itself. Most of the attention on these tokens is made up of these two pieces (attention to previous and attention to self), though tokens can attend to all previous tokens (standard causal transformer setup). Attention patterns look similar on similar subsets, indicating that Layer 1 heads operate similarly on our out of distribution evaluators. Layer 1 Head 1 appears to have transient previous token behavior, suggesting it may be partly responsible for the emergence of ICL.

(Olsson et al., 2022).

Surprisingly, we find here that the L2 induction heads in the ICL induction circuits are re-used by CIWL with little change. The transition from ICL to CIWL is implemented by a transition in the role of *L1 heads*, which switch from previous-token-attention to attending-to-self (Figure 5.1c).[7]

This is supported by attention patterns in L1 (Figure 5.7) and L2 (Figures 5.3a,b), and also by two sets of additional experiments, inspired by the notion of progress measures (Nanda et al., 2023).[8] We take each checkpoint from training, and then fix subsets of the network to the weights from a different checkpoint from the same run. In Figure 5.6c, we show that fixing the first half of the network (em-

[7]Algorithmically, we believe these heads are essentially learning to map label tokens to prototypical embeddings. It is hard to directly verify such a hypothesis, given the heads' rotation invariance and observed superposition (Appendix 5.5.2), but our evidence for the semantic stationarity of L2 points to such an algorithm.

[8]Despite the potential pitfalls of progress measures discussed earlier (Section 4.4.3), they suffice for our purposes here as we wish only to comment on the role of L1 and L2 heads in naturalistic training. We establish relevant causal relationships through clamping in Section 5.6.3.

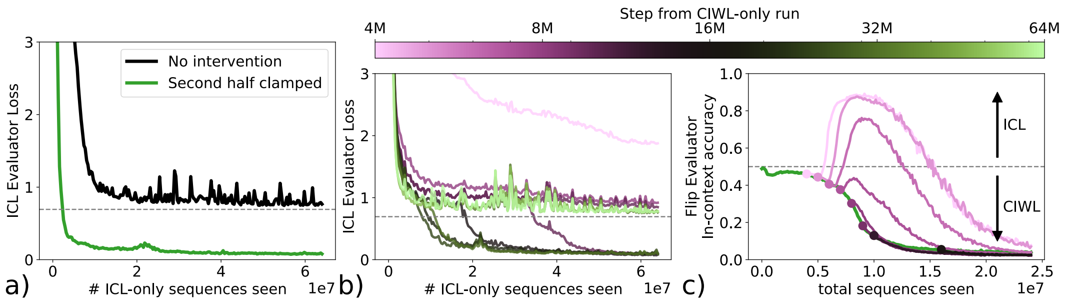


Figure 5.8: ICL emergence is enabled due to *cooperative* interactions with CIWL. **(a)** In this plot, we train networks on "ICL-only" data, i.e. where ICL is a viable strategy but CIWL is not. Without any interventions (black), we hit a loss plateau that greatly slows learning (reminiscent of the lengthening plateaus in Figure 4.5). However, if we replace the Layer 2 and unembedding weights with those from end-of-training on our standard training data (green), the network learns quickly. We thus see that these weights, which were part of a CIWL strategy, are reusable for learning ICL. **(b)** We further consider using Layer 2 + unembedding weights from different checkpoints of a "CIWL-only" run, and then training on "ICL-only" data. Early and late checkpoints lead to no learning of ICL, but middling checkpoints (from 9.8M to 31.5M, identified via binary search) do enable ICL learning. **(c)** We continued training different checkpoints from the "CIWL-only" run on our standard training data. Once CIWL has formed (later checkpoints), ICL does not re-emerge even when switching to the bursty data (which otherwise permits ICL).

bedding + L1) to that of a checkpoint at a given point (orange dotted lines) leads to very little change in network behavior after that point (flattening of the dark lines as compared to the lighter lines). Conjointly, if we fix the second half of the network (L2 + unembedding) to the checkpoint from the end of training (Figure 5.6b), we see little difference in behavior after the initial phase change, which indicates that Layer 2 changes are not meaningfully affecting the network after that point.

Notably, this means the canonical "induction heads" (in L2) remain, but they become part of the computational strategy CIWL rather than ICL, due to the change in L1. These results connect to a recently emerging notion that few-shot ICL (Brown et al., 2020) may lie on a spectrum of ICL abilities (Lampinen et al., 2024; Yin and Steinhardt, 2025), which we now show may be connected on the mechanistic level via shared subcircuits.

5.6.2 Learning the ICL strategy is *enhanced* by the availability of asymptotic CIWL

Why does ICL emerge at all, if it is not asymptotically preferred? We find that ICL's emergence may actually be enabled by ICL's shared L2 subcircuit with CIWL. Thus, shared subcircuits may lead to *cooperative interactions*, on top of known competitive interactions (Section 3.6, as well as Park et al. (2024)).

To show this, we train networks on "ICL-only" data, where we randomize the exemplar-label mappings across contexts, but keep the mappings consistent within each context. This data can only be solved by ICL, and not by CIWL (since there are no fixed mappings to learn in weights). [9] When training from scratch, we see the network struggle to learn (Figure 5.8a, black). Yet when we previously trained on bursty data (where CIWL *was* a viable strategy), we saw ICL emerge quite quickly (Figure 5.1b). Combined, these form a striking result—ICL actually emerges more easily when it is not the only viable strategy! Presence of an asymptotic CIWL mechanism may be enabling ICL emergence.

To further test this hypothesis, we again trained on ICL-only data, but clamped the weights of the second half of the network (L2 + unembedding), using those from the end of training from our "standard run" on bursty data, after the network has converged to a CIWL strategy. Through this causal intervention on dynamics (an example of *clamping*, Section 4.4.2), we show that ICL can emerge quite rapidly (Figure 5.8a, green) with the help of the CIWL weights for Layer 2. These findings connect to the notion that loss plateaus may arise from multiple sub-circuits needing to go right (Section 4.4). When the Layer 2 subcircuits are provided, ICL emerges quickly.

5.6.3 ICL is "close to" the path towards CIWL

One confound in the previous experiment is that CIWL weights at the end-of-training were likely influenced by the earlier presence of ICL (a possible path dependence). Here, we consider a stricter version of the experiment.

[9]This resembles the setup used in Chapter 4, except with 12800 classes and 20 exemplars per class.

We start by training networks on “CIWL-only” data: sequences drawn from the same distribution as the CIWL evaluator, which can be solved by CIWL but not ICL. We then take checkpoints from different points in training of this CIWL-only run, and use the L2 + unembedding weights to repeat the experiments of Section 5.6.2; namely, clamped training on ICL-only data. Essentially, we are asking if the L2 weights when learning a CIWL strategy, without influence from ICL, can still enable ICL strategies.[10] Surprisingly, we find that the answer is, for a brief period, yes.

Figure 5.8b shows training curves of our clamped networks on ICL-only data. Each curve is colored based on the iteration from the CIWL-only run that the clamped weights come from. We see that there is a region from about 10M to 32M sequences seen where ICL can emerge. After this point, L2 weights likely specialize too strongly on a CIWL strategy (as they are being trained on data that only permits such strategies), and thus can no longer be re-used for ICL.

5.6.4 If CIWL is fully formed, ICL will not emerge

To understand why and when ICL emerges, a final critical factor is that CIWL is “slow” to emerge on certain data distributions, relative to ICL. If this were not the case, the competitive interactions might dominate and CIWL might prevent ICL from ever emerging (as is the case under various data properties, as we found in Chapter 2 and reproduce in this smaller-scale setup in Appendix D.1.3).

To show more directly that ICL cannot emerge once CIWL has fully formed, we take various checkpoints from the “CIWL-only” run, and continue training but with bursty data (Figure 5.8c). When initializing from early “CIWL-only” checkpoints, we see a transient emergence of ICL before CIWL again dominates. At later checkpoints, CIWL persists, with no emergence of ICL. This indicates that ICL could only emerge at the earlier training times because CIWL was not yet fully formed.

[10]Note that we use the L2 weights of a CIWL-run from *the same initialization* to avoid issues related to learning the right rotations (Appendix C.5.1)

5.6.5 Strategy coopetition

Taken together, these results reveal a surprisingly rich interaction between ICL and CIWL: ICL is enabled by CIWL, yet also competing with CIWL for resources in the network. The competition between ICL and CIWL we previously suggested in Section 3.6 is now mechanistically observed in Layer 1. In contrast, Layer 2 exhibits cooperative interactions that lead ICL to emerge, despite not being asymptotic. This emergence dynamic has a few requirements:

1. **Useful**: ICL's ability to reduce loss on "bursty data."

2. **On the way**: ICL is "close" to the path of a 2L network learning an asymptotic CIWL strategy.

3. **Fast**: The CIWL strategy emerging "slowly" enough to allow the "faster" ICL to make a transient appearance.

The first factor follows from the dataset design. The second requirement is supported by Sections 5.6.1-5.6.3. The third requirement is supported by Section 5.6.4, our earlier work (Chapter 3), and related work since the initial finding of transience (Nguyen and Reddy, 2024).

5.7 A toy model of strategy racing + coopetition

To crystallize these intuitions, we propose a minimal mathematical model that captures the competitive and cooperative interactions at play. Specifically, we consider learning four vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$ via gradient descent on the following loss function, where $\mathbf{a}^*, \mathbf{b}^*, \mathbf{c}^*, \mathbf{d}^*$ are the true values:

$$
\begin{aligned}
&\mathscr{L}(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d}) = \\
&\left(\underbrace{||\mathbf{a}^* \otimes \mathbf{b}^* \otimes \mathbf{c}^* - \mathbf{a} \otimes \mathbf{b} \otimes \mathbf{c}||_F^2}_{\text{Mechanism 1 (ICL) Loss}} + \mu_1 \right) && (5.1) \\
&\times \left(\underbrace{||\mathbf{d}^* \otimes \mathbf{b}^* \otimes \mathbf{c}^* - \mathbf{d} \otimes \mathbf{b} \otimes \mathbf{c}||_F^2}_{\text{Mechanism 2 (CIWL) Loss}} \right) && (5.2) \\
&+ \alpha \underbrace{||\mathbf{a} \otimes \mathbf{d}||_F^2}_{\text{Competition}}, && (5.3)
\end{aligned}
$$

where $\alpha \geq 0$ is a parameter modulating the strength of competition,[11] and $\mu_1 \geq 0$ modulates the relative asymptotic preference of mechanism 2 over mechanism 1. If $\alpha, \mu_1 > 0$, loss is minimized when $\mathbf{a} = \mathbf{0}, \mathbf{b} = \mathbf{b}^*, \mathbf{c} = \mathbf{c}^*, \mathbf{d} = \mathbf{d}^*$.

This loss function builds on the minimal model of phase change dynamics we proposed in Section 4.4.2 – specifically, we use that model as the dynamical model for a single mechanism. Our full model has *multiple* circuits (ICL and CIWL) in competition, racing to minimize loss (Section 5.6.4), but also cooperating (Sections 5.6.1-5.6.3) via shared sub-circuits $(\mathbf{b}, \mathbf{c})$.

We model these race dynamics using two mechanisms that interact multiplicatively (lines (5.1) and (5.2)). If the offset $\mu_1 = 0$, then minimizing either the loss of Mechanism 1 or 2 will lead to 0 loss. By setting $\mu_1 > 0$, we can enforce an asymptotic preference for one mechanism over the other. Competition is added via the second term (line (5.3)), which drives $\mathbf{a}$ or $\mathbf{d}$ to 0 in order to achieve 0 loss (if $\alpha > 0$). This corresponds to the network pressure we observe for Layer 1 to be part of an ICL or CIWL strategy. To model cooperative interactions in Layer 2, we re-use vectors $\mathbf{b}, \mathbf{c}$ in both mechanisms. Finally, to model the relative speeds of the two mechanisms, we use different dimensions for $\mathbf{a}$ and $\mathbf{d}$ (larger vectors are slower to learn).

[11] An alternate interpretation is that α is a fixed version of a Lagrange multiplier for satisfying the constraint that $\mathbf{a}$ or $\mathbf{d}$ is 0.

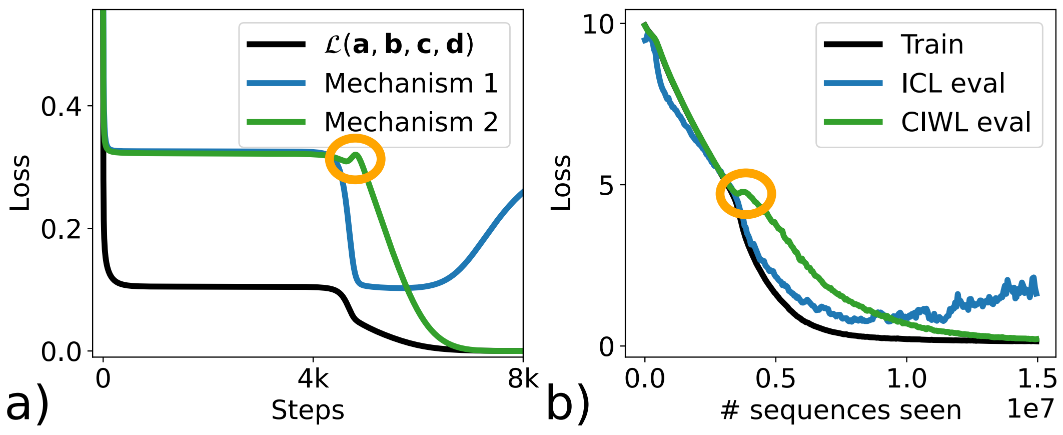


Figure 5.9: Minimal mathematical model captures key phenomena of strategy racing and coopetition. **(a)** Toy model dynamics. **(b)** Loss of real transformer, corresponding to Figure 5.1b. Notably, both curves exhibit transience behavior in Mechanism 1 (ICL). Intriguingly, the toy model also captures nonmonotonicity in the emergence of Mechanism 2 (CIWL), highlighted in orange.

Putting it all together, we simulate the toy model dynamics numerically for the setting of $dim(\mathbf{a}) = dim(\mathbf{b}) = dim(\mathbf{c}) = 20, dim(\mathbf{d}) = 160, \mu_1 = 0.1, \alpha = 0.1$. We show results in Figure 5.9a, with additional seeds and details in Appendix D.2. Our simulation shows a transience of Mechanism 1, where its loss first quickly drops with the initial phase change, but then goes back up, indicating that **a** is nearly learned but then eventually goes back to **0**. This return to **0** is caused by the asymptotically preferred **d** emerging on a slower timescale and the competition term ($\alpha > 0$). These emergence and subsequent transience dynamics mimic those of ICL, with sub-circuits $\mathbf{b}, \mathbf{c}$ representing the shared Layer 2 "induction heads."

Curiously, the toy model dynamics showed a small divot in the formation of Mechanism 2, corresponding to CIWL (highlighted in orange, Figure 5.9a). Specifically, when Mechanism 1 is nearly fully learned ($\sim 5k$ iterations), there is a mild *worsening* of Mechanism 2, before the eventual learning that causes transience of Mechanism 1 and asymptotic dominance of Mechanism 2. When we reexamined the loss curves of our actual transformer, (whose corresponding in-context accuracies are plotted in Figure 5.1b), we found a similar divot in the loss on CIWL evaluator (Figure 5.9b)! While we do not fully understand what leads to this brief divot,[12] we believe this mirroring between model and empirics lends credence to

[12]We speculate that as Mechanism 1 (ICL) forms, the competition forces a brief regression to

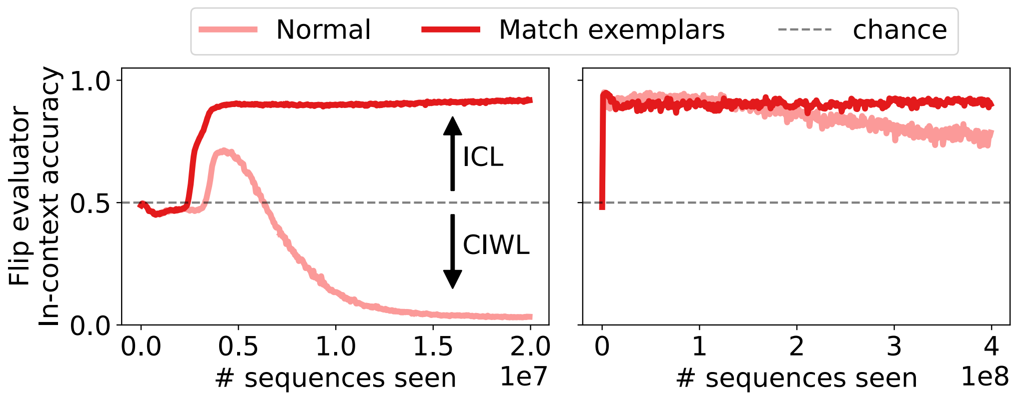


Figure 5.10: Matching context and query exemplars leads to persistent ICL in small (left) and large (right) models, as evidenced by the darker red line converging to accuracy one on the flip evaluator. Larger models are trained with the setup of Chapter 3, with the small exception of using pre-extracted Resnet embeddings (as done in Chapter 4 and this chapter). The timescale of transience is much longer in larger models (lighter line, right plot), so we do not see a full fading of ICL. Notably, the matched exemplars run shows no indicating of turning downwards.

this simple mathematical model.

5.8 ICL can be made asymptotically preferred

Equipped with a better understanding of transience, we return to a revised version of the question from Chapter 2: Are there data properties that incentivize the *asymptotic* emergence of ICL, but do not require it? We tackle this question using intuition from our toy model.

Specifically, if $\mu_1 = 0$, the faster Mechanism 1 (corresponding to ICL) would win asymptotically (Figure D.6). Thus, when two strategies are possible, one can be made preferred asymptotically if it is 1) faster and 2) as good asymptotically. We have already established that ICL is faster to emerge than CIWL (Section 5.6.4) when training on our bursty data, so all that remains is to see if we can find data properties that make ICL "not worse" at the task than CIWL.

The key difference in the mechanisms we find for ICL and CIWL is that CIWL matches query exemplars to *static labels*, while ICL matches to a *variable context exemplar*. This variation may make the "matching" harder for ICL (as postulated

Mechanism 2 (CIWL), which is slowly learning in parallel.

in Section 3.8). We ablate this by considering data where the relevant context exemplar is the exact same as the query exemplar.[13] In Figure 5.10, we find that training on such data makes ICL asymptotic—it persists after its initial emergence! Furthermore, this finding extends to the larger-scale setup considered in Chapter 3 as well,[14] indicating that the intuitions from the toy model may extend to larger networks.

In combination with our findings in Chapter 2, this result would point toward data properties that not only lead to the emergence of ICL—bursty data, with high diversity (# of classes) and variation (# of exemplars) across sequences—but its *persistence* as well: strong correlations in-context—exact matching exemplars. Overall, we found this intervention, motivated by our minimal mathematical model, a compelling illustration of how a better dynamical understanding may lead to data interventions that can control the cycling in and out of strategies throughout training.

5.9 Rejected hypotheses for the transience of emergent ICL

In the scientific process of figuring out what may cause ICL emergence, given that it is not asymptotically preferred, we considered (and subsequently rejected) many alternative hypotheses. In the spirit of sharing the path used to arrive at a conclusion, we share some of these hypotheses here.

5.9.1 Hypothesis 1: Earlier ICL is *necessary* for CIWL to emerge

Given the experiments above, this hypothesis already seems a bit suspicious—in Figure 5.5 and Section 5.6.4 we show that CIWL can be learned without earlier ICL emergence. But what if somehow bursty data requires ICL to emerge first in order

[13]Note: there are still 20 exemplars per class, making the exemplar-label mapping many-to-one (leaving CIWL the same), it is just that the bursty context exemplar exactly matches the query exemplar (making ICL more effective).

[14]Specifically, 12-layer transformers, with MLPs, trained on bursty sequences of 8 exemplar-label pairs, with the 3 bursty context exemplars matching the query in our intervention.

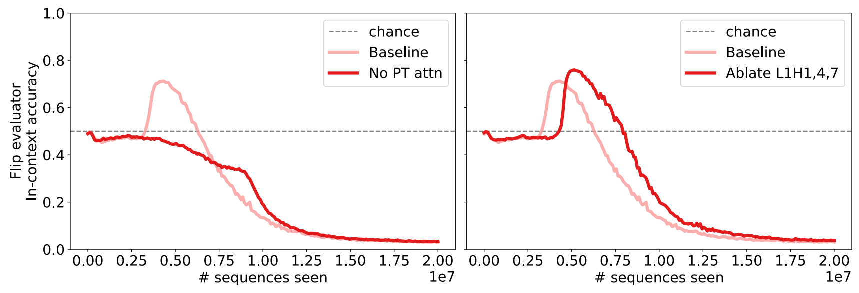


Figure 5.11: Various experiments to reject alternative hypotheses for why ICL emerges, despite not being asymptotic. Recall that accuracy of 1 on the Flip evaluator means ICL is dominant, and accuracy of 0 means CIWL is dominant. **(a)** ICL is not necessary for CIWL emergence. Darker line indicates evolution from a training run where Layer 1 heads were prevented from attending to the previous token, effectively disabling ICL. Lighter line is from standard training (same as Figure 5.1b). This clamp has relatively little effect on the dynamics of CIWL emergence. **(b)** ICL is likely not an initialization artifact. Darker line indicates evolution from a training run where Layer 1 heads 1, 4, and 7 (which play a crucial role in ICL in standard training) are ablated. While ICL emergence is slightly delayed, it still occurs, and in fact transience is a bit slower as well.

to reach a CIWL solution?

We test (and reject) this hypothesis by considering an experiment where we prevent Layer 1 heads from attending to the previous token, and then training on bursty data. This experiment is a causal intervention on dynamics (i.e., clamping), as it prevents ICL emergence but lets other strategies develop naturally. We show results in Figure 5.11a. Notably, we get similar curves to standard training, with the exception that ICL just does not emerge (since it was blocked). Thus, we reject the hypothesis that transient ICL is *necessary* to arrive at asymptotic CIWL.

5.9.2 Hypothesis 2: ICL emerges due to a lottery ticket

Another idea may be that ICL simply emerges since typical transformer initializations just happen to be "close" to an ICL solution—so rather than the dynamics leading the model through an ICL solution transiently, it just starts near one and the transience we observe is more of an "initialization effect." Though not quite the same, the notion of lottery tickets (Frankle and Carbin, 2018) is what inspired this hypothesis.

Given our earlier experiments, this hypothesis does feel a bit suspicious as well—for example, we find that ICL transience is quite robust over initialization seeds (Appendix D.1.1). To offer more evidence against this hypothesis, we again consider a clamping intervention:

If ICL is transient due to initialization effects, this would imply that the circuits being used in ICL are already "largely present" at the start of training. Through ablations on checkpoints from standard training, we identify that (for the main run we analyze throughout the text and presented in Figure 5.1b) the Layer 1 heads contributing previous-token behavior (Figure 5.7) to ICL are heads 1, 4, and 7. We establish this by first inspecting attention patterns, and then ablating these heads in checkpoints from around the peak performance of ICL ($\sim$3e6 to $\sim$1.5e7). Such an ablation on these checkpoints eliminates ICL behavior (as verified by performance on the ICL evaluator dropping to chance level).

We then re-train the model from scratch, with the same initialization, but with these Layer 1 heads clamped off *throughout training*. Presumably, this would eliminate any "already-existing-at-initialization" ICL circuits. Doing so minorly delays ICL emergence (Figure 5.11b), but does not prevent it. This indicates that their is strong dynamical pressure for ICL to emerge, regardless of initialization, providing evidence against this hypothesis.

5.10 Discussion

In this work, we finally get to the bottom of the question of why ICL emerges, despite not being asymptotically preferred. We uncovered an unexpected asymptotic strategy, CIWL, which is a hybrid of classical ICL and IWL. We found that *cooperative interactions* between CIWL and ICL enable emergent ICL, with *competitive interactions* eventually leading to ICL transience. Along with identifying shared mechanistic underpinnings, we codify these intuitions in a minimal mathematical model, which motivates further experiments and the creation of a synthetic setting where ICL emerges *and persists*, despite not being necessary to solve the task.

Overall, we hope our work serves to deepen the understanding of ICL and how it may interact with other strategies through the course of training. Beyond ICL, we view our work as a case study demonstrating a few themes that are important to keep in mind, for those of us who wish to understand model behaviors:

1. Models often learn *surprising and counterintuitive* strategies, even for simple tasks.

2. Models are highly *dynamical* and we cannot assume that their strategies remain constant over training, even if they appear stable at a given time.

3. These training dynamics are affected by a kind of *backwards hysteresis*, where later strategies can affect the development of earlier strategies.[15]

4. Alternate strategies are not always strictly in competition and can exhibit *coopetition* by boosting each other.

[15]We note that this backwards hysteresis complements notions of path dependence, such as the CIWL strategy being influenced by ICL earlier, for which correlates have been found in larger-scale language models (Yin and Steinhardt, 2025).

Chapter 6

Conclusion

This work was heavily inspired by the emergence of in-context learning (ICL), the ability to learn from context without weight updates, in large transformer models trained to predict the next token on internet-scale corpora (Brown et al., 2020). The simple question of why ICL emerges when not explicitly trained for led us down a winding path: In Chapter 2, we identified necessary data properties, inspired by those present in natural language, for ICL to emerge. That work enabled a smaller scale setup for studying ICL. In Chapter 3, we pushed the limits of this setup and found that, even when trained on data where ICL should be able to achieve perfect accuracy, ICL eventually *fades away* when overtraining transformers. These results motivated a *dynamical* understanding of ICL emergence, which we pursued through a mechanistic lens. In Chapter 4, we built key methodology for studying dynamics, and identified key subcircuits that need to "go right" for ICL to emerge. We then applied this enhanced understanding to the original question of ICL emergence on data where it is not explicitly required. In Chapter 5, we finally answered this question and found that *cooperative* interactions with asymptotic strategies (which tend to be a hybrid of in-context and in-weights learning) enable ICL emergence, with *competitive* interactions later in training leading to transience. This tradeoff is modulated by data properties, connecting back to our initial results in Chapter 2. Overall, this work serves to further the understanding of emergent ICL in transformers as a dynamical phenomenon, and answers the key question of why ICL may emerge (and eventually disappear) in transformers, even when not explicitly trained for.

6.1 Connecting to emerging literature on ICL

Concurrent to our work, the field of study around in-context learning has grown. A recent emerging notion in the field has been that ICL abilities actually lie on a *spectrum* Lampinen et al. (2024), rather than being as discrete as few-shot in-context learning vs. in-weights learning. In Chapter 5, we find mechanistic evidence of this idea, in many ways. The asymptotic mechanism on our Omniglot task is a context-constrained in-weights strategy, relying on in-weights learned mappings from exemplars to labels combined with labels from context, in a way that is strikingly similar to notions of task recognition in larger models.

Task recognition dates back to some of the first nuances found regarding ICL in large scale models (Wei et al., 2023; Wang et al., 2024; Min et al., 2022; Lin and Lee, 2024). Essentially, instead of learning a new task in-context, the few-shot exemplars are thought to help an LLM "retrieve" the relevant task from its prior knowledge (stored in-weights). Recent work (Yin and Steinhardt, 2025) suggests that canonical ICL circuits may even transition to serving this function over the course of training, a possible large-scale analog of our results on ICL transience.

Notably, our work also finds that task recognition and task learning are not as "distinct" as they may seem – we find that ICL and CIWL actually *share* critical sub-circuits, including the eponymous "induction head." These results add to the notion that ICL abilities lie on a spectrum, from things as simple as matching tenses, to coreference resolution, to few-shot learning, and possibly even learning new languages in-context (Gemini Team, 2024). We look forward to increased understanding of how different training paradigms (Touvron et al., 2023a) and architectures may lead to models lying on different points of this spectrum (Wei et al., 2023), rather than the binary of whether or not a model "can do ICL."

6.2 Connecting to mechanistic interpretability

Our work also takes heavy inspiration from mechanistic interpretability in transformers (Elhage et al., 2021; Olsson et al., 2022) and contributes back through its focus on dynamics. While prior work (Nanda et al., 2023) has mainly used progress measures, a correlational lens on dynamics, we innovate *clamping*, a method for causal interventions on dynamics. Given the history of correlation vs causal methods in mechanistic interpretability (Morcos et al., 2018; Belinkov, 2021), we hope that this methodology enables more rigorous study of dynamics, which we anticipate to be of increasing importance.

Beyond methods, our later work also connects to the rich emerging literature on superposition Elhage et al. (2022). While superposition is primarily studied in MLP layers (Bricken et al., 2023) or the residual stream (Lindsey et al., 2025), we present preliminary findings of superposition in attention heads, which could motivate similar, sparse autoencoder, approaches for attention heads.

6.3 Motivating a stronger focus on dynamics

Finally, we view our work as primarily advocating for a deeper look at *dynamical phenomena* in neural networks. Most analysis focuses on end networks, or at best, static checkpoints from training (Biderman et al., 2023), which we believe may miss out on key interactions. Our initial work in Chapter 2 started as such: we considered data properties inspired by natural language and asked if they led to ICL emergence, when training for a fixed amount of time. We quickly realized that the *dynamics* were actually key to understanding ICL emergence (Chapter 3), and ended up finding remarkably rich “coopetitive” interactions between a spectrum of ICL capabilities (Chapter 5). We view our work on ICL here as a case study and are excited by a future where dynamical understanding of phenomena is prioritized.

That said, understanding the dynamics of neural networks, which are already canonically “black boxes,” presents its own set of challengs. Many works focus on simplifying down networks, primarily through architectural changes (e.g., Reddy (2023); Zhang et al. (2025)), to enable building theories of learning. While useful, we advocate for a second approach: that of building minimal mathematical models that capture key dynamical phenomena, without having to model the underlying system in detail.

We were inspired to this approach by classical mechanics – while it leaves much on the table (i.e., strictly speaking it is "incorrect" due to relativity), simple equations like Newton's second law are surprisingly powerful models of movement in the natural world. We hope that similar models can be built of the movement of neural network weights through the course of training.

We view our minimal models as laying groundwork in this direction. For example, simply learning a set of vectors via gradient descent can lead to remarkably rich phenomena: Loss functions defined on tensor products (Section 4.4.2) implement an 'AND' operation, necessitating vectors to be learned concurrently and giving rise to phase changes. Yet when mechanisms with scalar losses are composed via a product (Section 5.7), they instead implement an 'OR' operation, enabling either of two strategies to emerge. By composing such simple operations, remarkably rich phenomena (e.g., phase changes, coopetition) can be modeled.

An interesting intermediate between our "bottom-up" approach to dynamics, and the "top-down" approach of simplifying networks, is the field of deep neural networks (Saxe et al., 2014). Recent works in this field (Saxe et al., 2022) extend the analysis to non-linear networks where multiple pathways "race" to solve the task, with pathways containing the most *shared substructure* win and can predict the wiring of a non-linear ReLU network. We see many intuitive connections between such works and our own, with strategies racing against each other (ICL vs CIWL), emerging and trading off through training due to shared substructure (induction heads). Such intermediate levels of analysis could serve as the path for connecting minimalistic mathematical models to phenomena observed in real transformers, even at large scale (Yin and Steinhardt, 2025).

Appendix A

Appendix for Chapter 2

A.1 Results on ICL eval with in-distribution exemplars

Figure A.1 shows results of evaluating in-context learning on classes that were seen in training, rather than on holdout classes (the standard evaluation setting for few-shot learning, as described in Section 2.2.3. The pattern of results is very similar between the two settings, with just slightly higher performance when evaluating on training classes.

A.2 Hyperparameter sweep for RNNs

A.2.1 Architectural details

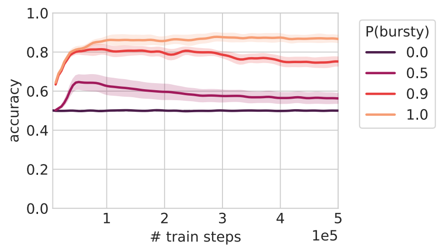
Hyperparameter sweep:

- Max learning rate: 15 samples log-uniform distributed over the range [1e-5, 0.1]
- Num warmup steps: 15 samples log-uniform distribution over the range [1, 10000]

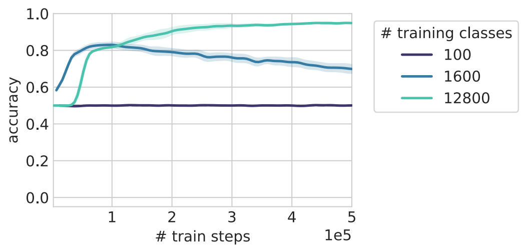
We performed 15 runs for each architecture (Transformer with 2 or 12 layers, LSTM with 2 or 12 layers, Vanilla RNN with 2 or 12 layers), i.e. 90 runs total.

Parameter counts:

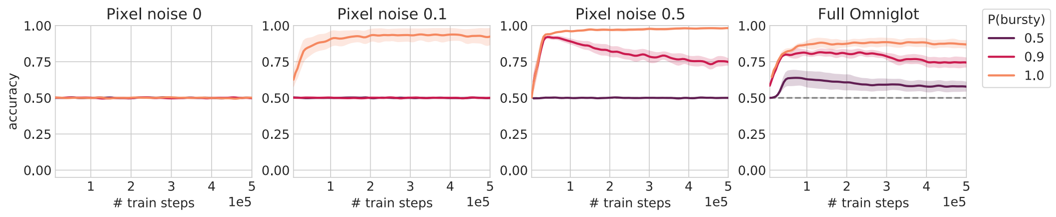
(a) Burstiness.



(b) Num training classes*



(c) Within-class variation.



(d) Dynamic meanings*

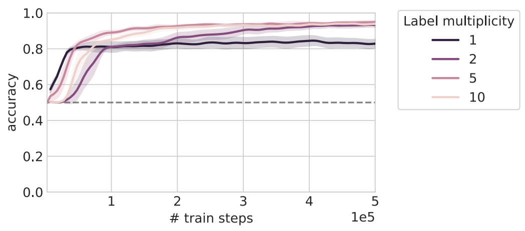


Figure A.1: In-context learning accuracy, evaluated on classes that were observed in training, rather than holdout classes. Patterns of results are very similar to those shown in the main text, with overall slightly higher performance when evaluated on training classes.

- Transformer with 12 layers: 831,479
- LSTM with 12 layers: 627,959
- Transformer with 2 layers: 331,639
- LSTM with 2 layers: 297,719

A.2.2 In-weights learning

Transformers exhibited similar or slightly higher in-weights learning than the recurrent models (Figure A.2), indicating that their superior in-context learning performance (as seen in Figure 2.7) cannot simply be explained by a bias towards

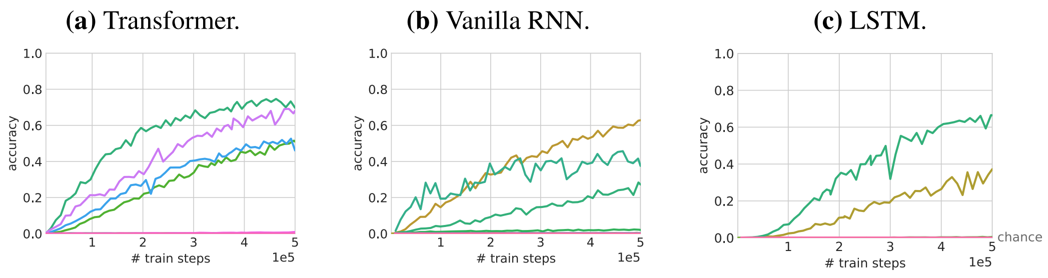


Figure A.2: In-weights learning in transformers vs. recurrent architectures. We compare architectures while holding fixed the number of layers, hidden layer size, and number of parameters. One run was performed for each set of hyperparameters in a hyperparameter sweep. Each color denotes one run, but not any particular hyperparameter values.

in-context learning and against in-weights learning.

Appendix B

Appendix for Chapter 3

B.1 Construction of LLaMa-based exemplar-label datasets

In this section, we provide more detail as to the procedure used to construct the fixed embedding datasets used in Section 3.4.3.

B.1.1 Constructing datasets

We save the token embedding matrix for each LLaMa v1 model. These have dimensions 32,000 $\times d_{model}$, where $d_{model} = 4096, 5120, 6656, 8192$ for the 7B, 13B, 30B, and 70B, LLaMa models, respectively. To go from these raw embeddings to "classes" that we can use in our setup, we perform the following steps:

1. Subselect: Many tokens correspond to individual bytes or characters from non-English languages. We observed that the statistics of these token embeddings are often different than the others, so we remove these tokens. Specifically, we use tokens 259 to 29,870, inclusive, for a total of 29,612 tokens.

2. Cluster: We cluster the token embeddings using K-means clustering and cosine distance using the FAISS library (Johnson et al., 2019). As FAISS does not offer a valid minimum number of points per cluster, we first cluster into 2400 (or 4800) clusters, then pick all clusters that have more than 10 (or 5) points. This gives us at least 1,600 (or 3,200) clusters. We randomly pick the actual 1,600 (or 3,200) clusters.

3. Select from cluster: Many of these clusters contain more than 10 (or 5) points. To maximize in-class variation, we pick the furthest 10 (or 5) points from the cluster centroid. Thus, we have our 1,600 classes of 10 examples, or 3,200 classes of 5 examples.

Note that step 2 is highly dependent on the random seed used to initialize cluster centroids. We run the above procedure for all 4 LLaMa models on 2 random seeds.

B.1.2 Example clusters

If we map the embeddings back to tokens, we can get a sense of what various clusters may represent. Many clusters cover similar tokens, but some clusters can be more semantic or even multilingual! We show some extracted clusters (for 3,200 classes, 5 embeddings per class, on the 70B LLaMa) below:

```
_erm | _room | rm | _Room | _rm
_utter | _extremely | _partially | _entirely | Comple
hover | _mysqli | gz | RGB | rgb
async | _predicate | Observer | _deleg | _delegate
ive | ous | _cable | rable | ible
FLAG | _flag | _Flag | flag | Flag
_specified | _provided | _supplies | _supply | _supplied
ring | _dent | dent | _rent | RENT
_Derby | _Dit | _pedig | _ort | _Ort
_Britain | _England | _Madrid | _Spanish | Espagne
```

B.1.3 The difficulty of emergent ICL

We observed that ICL emergence was more sensitive to model initialization when using LLaMa token embeddings as exemplars. An example seed where ICL did not emerge at all (for 3,200 classes, 5 exemplars per class) is shown in Figure B.1. However, in all cases where ICL did emerge (even to a small degree), we found it to be transient (Figures 3.6, B.2). Furthermore, the results presented in Section 3.4.3 are averaged over two different clustering seeds (see step 2 above), which yield different output datasets. ICL is transient across all 8 datasets tested.

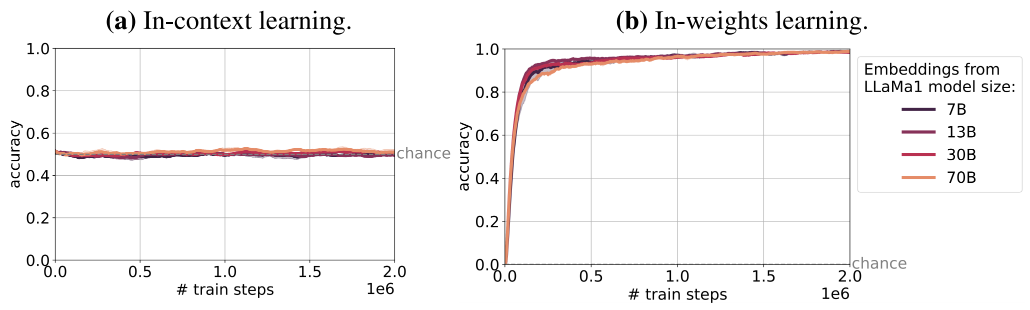


Figure B.1: A model initialization seed for which no ICL emerges on LLaMa-derived datasets

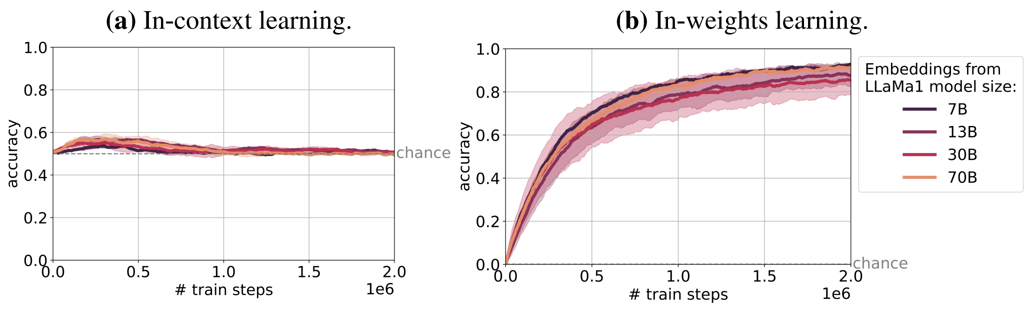


Figure B.2: Mild emergence and subsequent transience of ICL on a model trained on the LLaMa derived datasets with 1,600 classes and 10 exemplars per class.

At 1,600 classes, with 10 exemplars per class, we found mild ICL emergence, as shown in Figure B.2. We found that 3,200 classes were necessary to get any strong ICL, which is shown in Figure 3.6.

B.2 Additional regularization results

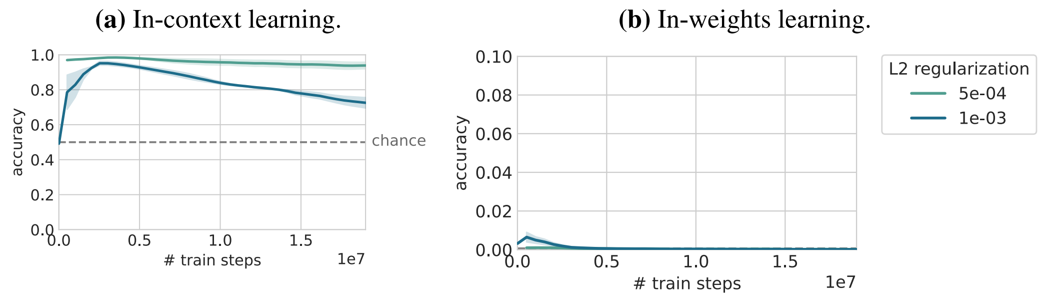


Figure B.3: For extreme values of L2 regularization (>1e-04), the model seems to be over-regularized: We observe a return to ICL transience, as well as poor performance on IWL.

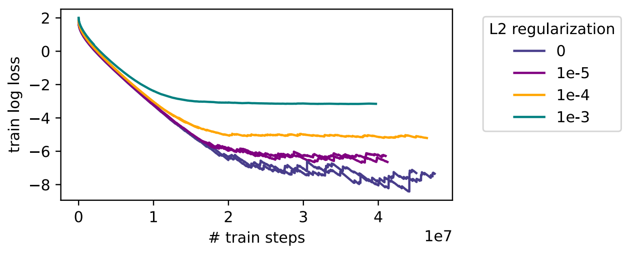


Figure B.4: Total train loss (including L2 penalty) for a sampling of experiments. Regularization seems to make the training loss smoother, as well as causing train loss to plateau at a higher value (rather than continuing to drop).

Appendix C

Appendix for Chapter 4

C.1 Dataset size calculation

For a dataset of C classes, E exemplars, L labels, and assuming a fraction F of relabelings being used for training, our total dataset size is

$$\binom{C}{2} \cdot F \cdot \binom{L}{2} \cdot E^3 \cdot 2 \cdot 2 \cdot 2 = 2FE^3C(C-1)L(L-1) \Rightarrow 1.6C(C-1)L(L-1)$$

for our settings where $E = 1$ and $F = 0.8$ are fixed.

C.2 The "residual heads" hypothesis

In Section 4.3.1, we provide evidence that induction heads function additively with redundancy across heads. Combining this insight with the ordering of head emergence in Figure 4.3b, we hypothesized that other induction heads form to minimize the residual of the loss left after earlier heads form (e.g. Head 3). We didn't include this hypothesis in the main thesis as we never fully got to the bottom of it, and it's not necessary for the main narrative. We include it here as we find evidence supporting and opposing it and so we hope these mixed results can serve as a launch point for further study.

One simple way to test this hypothesis would be to consider the mistakes made by the individual Layer 2 heads, when all other Layer 2 heads are turned off. Let's consider Head 2 and Head 3 for example. Head 3 on its own achieves an accuracy of 98%, outputting the wrong answer for 1553 of the 78400 training sequences. Head

Accuracy	Head 0	Head 1	Head 2	Head 4	Head 5	Head 6	Head 7
On the subset	24.08%	64.78%	71.67%	32.97%	56.41%	16.61%	46.10%
Overall	40.32%	76.51%	75.12%	42.85%	64.65%	29.16%	58.85%

Table C.1: Accuracy when ablating all but a given Layer 2 head on the subset of problems that are incorrect when ablating all but Layer 2 Head 3 (the one with highest induction score). Second row contains the accuracies over the whole dataset for comparison (which matches the circles in Figure 4.5a).

Accuracy	Head 0	**Head 1**	Head 2	Head 4	Head 5	Head 6	Head 7
On the subset	24.14%	**86.21%**	20.69%	22.41%	50.00%	0.00%	44.83%
Overall	50.01%	82.80%	82.73%	62.10%	65.42%	28.46%	70.70%

Table C.2: Same as Table C.1, except we also "perfect" each induction head's match circuit (as done in Figure C.1), so we can view this table as only considering the mistakes in each induction head's copy circuit. There are only 58 examples that the "perfected match" layer 2 Head 3 gets wrong, so this table only considers the accuracy of other "perfected match" heads on these 58 examples. Note the overall accuracies in this Table match the squares in Figure C.1.

2's accuracy is 75%. If Head 2 is fitting to the residual, we might expect Head 2's accuracy on the sequences that Head 3 fails to solve to be above its baseline of 75%, meaning it's preferentially stronger on the problems where Head 3 is mistaken. On the other hand, it could be that the sequences missed by Head 3 are simply "harder", in which case we would expect an accuracy less than its baseline. In Table C.1, we find that the latter is more likely to be the case, so this piece of evidence is *against* the residual heads hypothesis.

Next, we present some evidence that supports the hypothesis. Namely, we consider perfecting each induction head's match circuit (by setting its attention pattern to the perfect pattern that gives attention 1 to the correct token and 0 to all other tokens), which allows us to quantify the quality of its copy circuit. Then, we can examine the mistake rate of the copy circuit for different labels, and see these values across heads. Our results are summarized in Figure C.1. Specifically, we do find some specialization of output classes: Head 3 has a higher mistake rate for copying labels 1 and 3, while heads 1 and 5 have virtually 0 mistake rate on these, indicating that they may have specialized to these examples. This result *supports* the "residual heads" hypothesis.

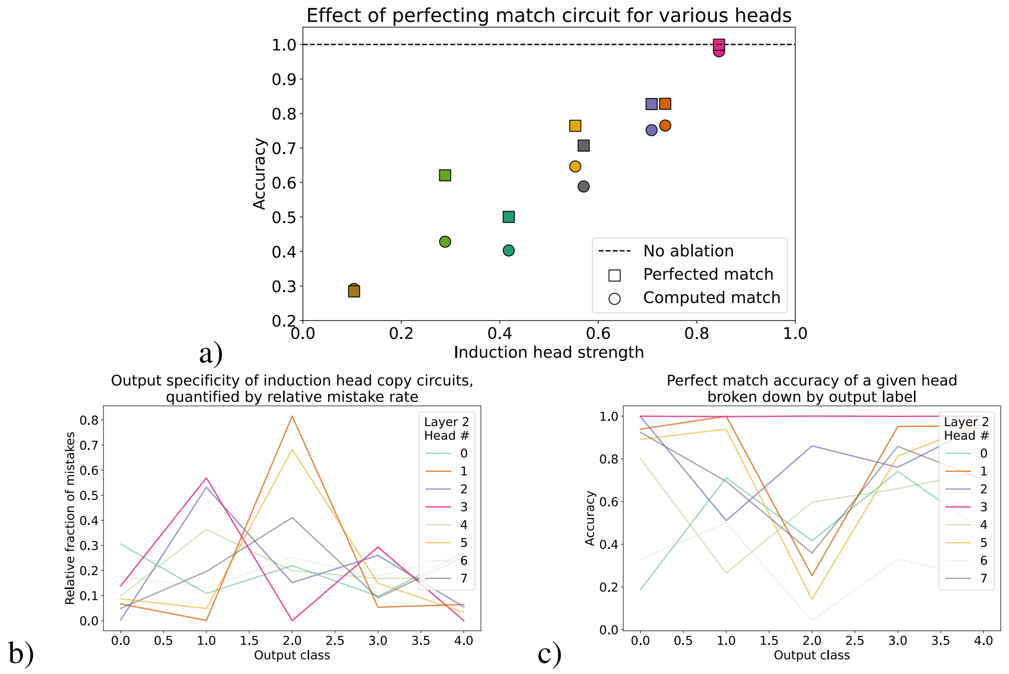


Figure C.1: a) Induction heads are imperfect copiers. We consider the network from the end of training in Figure 4.3 and perform the ablate-all-but-head-X ablation used for the circles in Figure 4.5. We then use our artificial optogenetics framework to "perfect" each head's attention pattern (so it would have an induction strength of 1). While this increases the accuracy for most heads, the accuracy is still far from perfect, indicating that inducion heads at the end of training are imperfect copiers. **b)** We dive deeper and find label dependence in the imperfect copying. Specifically, here we plot the relative mistake rate by output label for each head. Line opacity is equal to induction head strength, to emphasize the more prominent induction heads. Interestingly, Head 3 is relatively worse at copying labels 1 and 3, but heads 1 and 5 compensate for this. Notably, in Figure 4.3, heads 1 and 5 appear after Head 3. We postulate that heads 1 and 5 are thus compensating for the mistakes that Head 3 makes. Future work could further explore such a "residual heads" hypothesis in more depth. **c)** For completeness, we include accuracy vs output class. We note that Head 3 with a "perfect" attention pattern overall does make very few mistakes, as we'd expect given its high accuracy in a). Notably, what plot b shows is that Head 3 with a "perfect" attention pattern, when it does make mistakes, tends to do so on classes 1 and 3.

If we repeat the analysis of Table C.1, we find some promising evidence in this “perfected match” condition (Table C.2). These results suggest that, when it comes to the copying subcircuit, induction heads forming later in training preferentially focus on copying labels that earlier formed induction heads are worse at. As a result, these earlier heads do not have to “fix their mistakes” as much as if they were trained on their own, connecting to our results in Section 4.3.2. This result is *inconclusive* with respect to the “residual heads” hypothesis, as it only pertains to Head 1, and not the other heads.

Finally, we build off the result in Figure 4.5b, and try to take the argument further. Given that Head 3 is the quickest to learn and does in fact emerge first in the full network (Figure 4.3b), we consider training networks where all but *two* heads are ablated: Head 3 and each of the other 7 heads (one at a time). Loss curves are shown in Figure C.2. By training with Head 3 and just one other head, we see loss curves are way closer to the actual training curve. However, the ordering of phase changes still does not match the ordering of emergence and strengths in Figure 4.3b, perhaps due to higher order interactions (e.g., between three heads or more). Given the current results though, this piece of evidence is *against* the “residual heads” hypothesis.

Given all these pieces of evidence, for and against, we cannot convincingly argue for or against the “residual heads” hypothesis. We hope this investigation (including negative results) can inform future work researching this hypothesis.

C.3 The role of Layer 1 heads

To establish the causal role of Layer 1 heads to individual induction heads in Layer 2, we did extensive ablation analyses.

Earlier, we found that Layer 1 heads do not substantially contribute to the output logits. To see if they only compose to Layer 2 heads’ attention patterns (the match subcircuit), we consider a “pattern preserving” ablation (Olsson et al., 2022): the outputs of Layer 1 heads are completely ablated, but the attention patterns in Layer 2 are fixed to what they would’ve been had the Layer 1 heads not been ab-

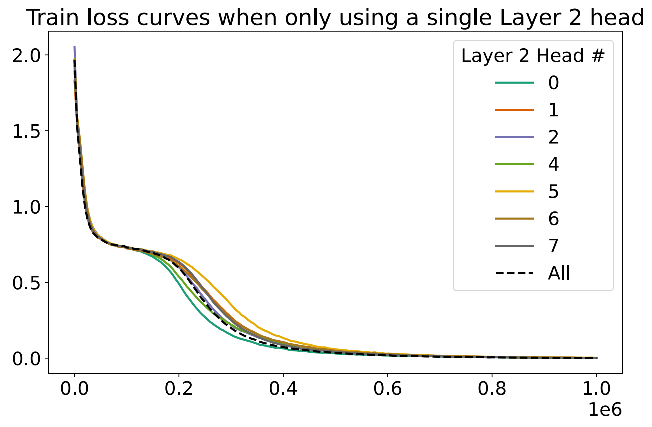


Figure C.2: A follow-up to Figure 4.5b where we consider training with only a pair of Layer 2 heads active: Head 3 and one other head. Interestingly, loss curves snap to almost the true training loss curve. The relative ordering of phase changes is not indicative of the order of emergence (see Figure 4.3).

lated. We implement this using our artificial optogenetics framework by using two passes through the network. In the first, nothing is ablated. The Layer 2 attention patterns from this pass are then put into the `cache`. On the second pass through the network, Layer 1 outputs are ablated, and the `cache` is used to restore the patterns. We find that accuracy drops substantially (down to 55%), indicating substantial V-composition in our networks. To study V-composition, we introduce an analogous "value-preserving" ablation where instead of caching attention patterns, Layer 2 value vectors are cached. When using just a "value-preserving" ablation of all Layer 1 heads (without preserving patterns), accuracy drops substantially (down to 11%). These results indicate that Layer 1 heads are contributing to both the match subcircuit (patterns) and the copy subcircuit (values). Of the two, the role in the match circuit does seem to be "more important", which aligns with prior work (Olsson et al., 2022).

To verify the observed previous token (PT) heads, we next consider an ablation of these three heads with pattern-preserving (which should have no effect if they are PT heads) or value-preserving (which should have a huge effect, since patterns in Layer 2 will get messed up with these heads deleted). Likewise, we consider both types of ablations of the other heads (Layer 1 heads 0,3,4,6,7). Results are summarized in Table C.3. We can clearly see that ablating heads 1,2,5 while preserving

Ablated heads	Type of ablation: Pattern-preserving	Value-preserving
Layer 1 Heads 1,2,5 (PT heads)	99.85%	69.87%
Layer 1 Heads 0,3,4,6,7 (rest)	34.27%	99.32%

Table C.3: Accuracy after Layer 1 head ablations of various sets of heads. All Layer 2 heads remain active.

patterns has little effect, indicating these heads primarily compose with the match subcircuit. Likewise, we find the other heads primarily participate in V-composition for the copy subcircuit.

Finally, we wish to establish the connectivity between Layer 1 and Layer 2 heads. To be specific to Layer 2 head, we repeat each experiment 8 times, once for each Layer 2 head active (and the others are ablated). As in Section 4.3.1, we first consider ablating a single previous token head. Results are shown in Figure C.3a. We find that when ablating single heads, only Head 2 seems to have a strong effect. We believe this result is again suffering from the redundancy across heads, so the importance of heads 1 and 5 is masked. To account for this, we instead consider ablating all but a given Layer 1 head. In this case, each square in the heatmap can be seen as one path through the network, with only one head in each layer used. Results are shown in Figure C.3b. From these plots, it's clear that there are multiple active paths in the network:

1. Layer 1 Head 1 $\rightarrow$ Layer 2 Head 3
2. Layer 1 Head 2 $\rightarrow$ Layer 2 Heads 1, 2, 3
3. Layer 1 Head 5 $\rightarrow$ Layer 2 Heads 1 and 3

These results indicate a *many-to-many wiring diagram.*

We'd also like to take a moment here to emphasize the use of activation-level ablation analyses, as opposed to metrics calculated on weights. Composition score

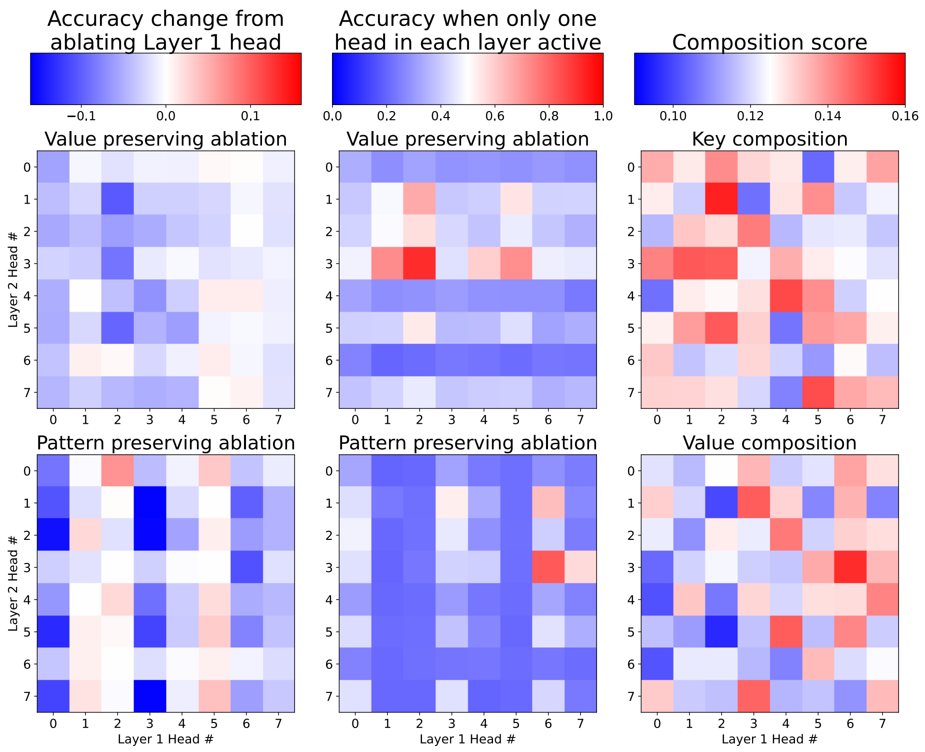


Figure C.3: Layer 1 (L1) $\rightarrow$ Layer 2 (L2) head connectivity analyses (three different methods as the three columns). **Left:** Effect of ablating a single L1 head. Each column corresponds to the L2 head that is kept active. We plot the deviation compared to if the given L1 head was not ablated, to emphasize positive/negative differences (here darker blue would mean more influential). Top plot preserves values, so results will be most affected by L1 heads that contribute to the match subcircuit (previous token heads). From this, we see that L1 Head 2 is an important previous token head for many of the L2 heads. The bottom plot preserves patterns, so results will be least affected by L1 heads that only contribute to the match subcircuit (previous token heads). When ablating heads 1, 2, and 5, performance improves, indicating that previous token heads might be interfering with the network in some cases due to (suboptimal) V-composition. **Middle:** Performance of a single pair of heads (one from L1, and one from L2) when preserving L2 values (left) or patterns (right). We plot the raw accuracy to show how strong each path through the network is. The color scheme highlights points (in red) that go above simple context copying strategies (which has an accuracy of 50%). We see that L1 Head 2 $\rightarrow$ L2 Head 3 is the strongest path (when preserving values), leading to a performance of 90%. The top plot indicates that L1 heads 1, 2, and 5 play a strong role in computing patterns. The bottom plot indicates that L1 Head 6 has the most important V-composition to L2 heads. **Right:** For comparison, we include the composition score of L1 head output projections to key and value input matrices in L2. We again use a bwr colorscheme to highlight pairs with higher than average composition scores. While the main forms of composition are still visible (K-composition from L1 Head 2 to L2 Head 1/3, and V-composition from L1 Head 6 to L2 Head 3), we note that the results are overall noisier. This suggests that the activation modification approach used in the middle column, made easy to implement by our artificial optogenetics framework, may provide clearer signals than weight-based approaches on circuit wiring.

(Elhage et al., 2021; Nanda, 2022) is one such weight-based metric, calculated as:

$$\text{score}\left(L_1H_i \to L_2H_j : Q/K/V\right) = \frac{||W_O^{L_1H_i} W_{Q/K/V}^{L_2H_j}||_F}{||W_O^{L_1H_i}||_F ||W_{Q/K/V}^{L_2H_j}||_F}$$

It aims to measure the connection strength between heads in different layers by considering the matrices they use to write output and read input from the residual stream. In Figure C.3c, we plot the composition scores of Layer 1 heads to key and value matrices of Layer 2 heads. Though the most noteworthy cases of composition are recovered (such as the importance of Layer 1 Head 2 for K-composition, and Layer 1 Head 6 for V-composition), overall we find the results to be much harder to interpret than those using our activation-level ablation analyses (Figure C.3b).

C.4 Proof of saddle point in toy model

We include a brief proof that $\mathbf{a} = \mathbf{b} = \mathbf{c} = \mathbf{0}$ is a saddle point for the toy model, leading to the long loss plateau before the phase change.

Recall, the loss being minimized is:

$$\mathscr{L}(\mathbf{a},\mathbf{b},\mathbf{c}) = \frac{1}{2}\sum_{i,j,k}\left(a_i^* b_j^* c_k^* - a_i b_j c_k\right)^2,$$

and we assume $\mathbf{a}^*, \mathbf{b}^*, \mathbf{c}^* \neq \mathbf{0}$.

We then have:

$$\begin{aligned}
\frac{\partial \mathscr{L}(\mathbf{a},\mathbf{b},\mathbf{c})}{\partial a_i} &= -\sum_{j,k}\left(a_i^* b_j^* c_k^* - a_i b_j c_k\right) b_j c_k = 0, \\
\frac{\partial \mathscr{L}(\mathbf{a},\mathbf{b},\mathbf{c})}{\partial b_j} &= -\sum_{i,k}\left(a_i^* b_j^* c_k^* - a_i b_j c_k\right) a_i c_k = 0, \\
\frac{\partial \mathscr{L}(\mathbf{a},\mathbf{b},\mathbf{c})}{\partial c_k} &= -\sum_{i,j}\left(a_i^* b_j^* c_k^* - a_i b_j c_k\right) a_i b_j = 0,
\end{aligned}$$

if $\mathbf{a} = \mathbf{b} = \mathbf{c} = \mathbf{0}$. Moving forward, we will abbreviate derivations using the symmetry between a, b, c.

The value of the loss function at this point is:

$$\mathscr{L}(\mathbf{0}) = \frac{1}{2}\sum_{i,j,k}\left(a_i^* b_j^* c_k^*\right)^2$$

While we know this point is neither a global maximum (∞) nor a global minimum (0) of the loss function, we now need to show it's a saddle point (and not a local minimum/maximum).

C.4.1 Hessian test inconclusive for 3 or more interacting variables

A first attempt to show it's a saddle point (and not a local minimum/maximum) would be to use the determinant of the Hessian. We note that such a proof does work in the two vector case (the determinant of the Hessian is negative in that case), but for the 3-variable case, the determinant of the Hessian is also 0. We show this below:

The Hessian can be seen as a 3×3 block matrix:

$$\mathscr{H} = \begin{pmatrix} H_{aa} & H_{ab} & H_{ac} \\ H_{ba} & H_{bb} & H_{bc} \\ H_{ca} & H_{cb} & H_{cc} \end{pmatrix},$$

and we can work out the individual terms below:

$$\frac{\partial^2 \mathscr{L}(a,b,c)}{\partial a_i^2} = \sum_{j,k}(b_j c_k)^2 = 0$$
$$\frac{\partial^2 \mathscr{L}(a,b,c)}{\partial a_i \partial a_{i'}} = 0, i \neq i'$$

so $H_{aa} = H_{bb} = H_{cc} = \mathbf{0}$. For the cross terms, we have:

$$\frac{\partial^2 \mathscr{L}(a,b,c)}{\partial a_i \partial b_j} = \sum_k 2a_i b_j c_k^2 - a_i^* b_j^* c_k^* c_k = 0$$

Here, the presence of c_k forces the these partials to 0, making the Hessian a matrix of all 0's. Note in the two variable case (if we remove **c**), the Hessian would have positive off-diagonal elements and it becomes easy to show the determinant is negative.

C.4.2 Directions of positive and negative slope

Since the Hessian test doesn't work, we can instead attempt to identify two directions, one in which the function will increase in a small neighborhood and one in which it will decrease in a small neighborhood. These are easy to construct by picking i', j', k' s.t. $a_{i'^*} b_{i'^*} c_{i'^*} \neq 0$ (such indices are guaranteed to exist since $\mathbf{a}^*, \mathbf{b}^*, \mathbf{c}^* \neq \mathbf{0}$).

Specifically, consider the direction of $\mathbf{d}_- = \langle 0, ..., 0, a^*_{i'}, 0, ..., 0, b^*_{j'}, 0, ..., 0, c^*_{k'}, 0, ..., 0\rangle$. We can see that a small $\varepsilon > 0$[1] perturbation along this direction will reduce the loss:

$$\begin{aligned}
\mathscr{L}(\mathbf{0}+\varepsilon\mathbf{d}_-) &= \frac{1}{2}\sum_{i,j,k}\left(a^*_i b^*_j c^*_k - a_i b_j c_k\right)^2 \\
&= \frac{1}{2}\left(a^*_{i'} b^*_{j'} c^*_{k'} - \varepsilon^3 a^*_{i'} b^*_{j'} c^*_{k'}\right)^2 + \frac{1}{2}\sum_{i\neq i', j\neq j', k\neq k'}\left(a^*_i b^*_j c^*_k\right)^2 \\
&= \frac{1}{2}\left(a^*_{i'} b^*_{j'} c^*_{k'}\right)^2 (\varepsilon^6 - 2\varepsilon^3) + \frac{1}{2}\sum_{i,j,k}\left(a^*_i b^*_j c^*_k\right)^2 \\
\Rightarrow \lim_{\varepsilon\to 0}\mathscr{L}(\mathbf{0}+\varepsilon\mathbf{d}_-) - \mathscr{L}(\mathbf{0}) &= \frac{1}{2}\left(a^*_{i'} b^*_{j'} c^*_{k'}\right)^2 (\varepsilon^6 - 2\varepsilon^3) \approx -\varepsilon^3\left(a^*_{i'} b^*_{j'} c^*_{k'}\right)^2 < 0
\end{aligned}$$

Thus, a direction with a negative directional derivative exists.

Similarly, we consider $\mathbf{d}_+ = \langle 0, ..., 0, -a^*_{i'}, 0, ..., 0, b^*_{j'}, 0, ..., 0, c^*_{k'}, 0, ..., 0\rangle$. We

[1]Note, the positivity assumption is not necessary. In fact, if we allow ε to take either sign, we can just use one direction to see the saddle point (due to the factor of ε^3). However, for the case of an even number of components (e.g., $N = 4$ instead of $N = 3$), we do need two separate directions (since ε^N will always be positive if N is even). To make this proof general, we thus provide the two directions and focus on positive ε.

can see that a small $\varepsilon > 0$ perturbation along this direction will increase the loss:

$$\begin{aligned}
\mathscr{L}(\mathbf{0}+\varepsilon\mathbf{d}_+) &= \frac{1}{2}\sum_{i,j,k}\left(a_i^*b_j^*c_k^* - a_ib_jc_k\right)^2 \\
&= \frac{1}{2}\left(a_{i'}^*b_{j'}^*c_{k'}^* + \varepsilon^3 a_{i'}^*b_{j'}^*c_{k'}^*\right)^2 + \frac{1}{2}\sum_{i\neq i',j\neq j',k\neq k'}\left(a_i^*b_j^*c_k^*\right)^2 \\
&= \frac{1}{2}\left(a_{i'}^*b_{j'}^*c_{k'}^*\right)^2(\varepsilon^6+2\varepsilon^3) + \frac{1}{2}\sum_{i,j,k}\left(a_i^*b_j^*c_k^*\right)^2 \\
\Rightarrow \lim_{\varepsilon\to 0}\mathscr{L}(\mathbf{0}+\varepsilon\mathbf{d}_+) - \mathscr{L}(\mathbf{0}) &= \frac{1}{2}\left(a_{i'}^*b_{j'}^*c_{k'}^*\right)^2(\varepsilon^6+2\varepsilon^3) \approx \varepsilon^3\left(a_{i'}^*b_{j'}^*c_{k'}^*\right)^2 > 0
\end{aligned}$$

Thus, we've identified two directions, one which increases the loss and one which decreases the loss, indicating that $\mathbf{a} = \mathbf{b} = \mathbf{c} = \mathbf{0}$ must be a saddle point. Note, if we find the directional derivative in either of these directions, it's still 0 (due to the factor of ε^3 not vanishing, even if we divide by ε to get the derivative)—we simply show the derivation to show the differing sign. We also note that this argument extends beyond 3 interacting vectors.

This argument can also give some intuition as to why the saddle point is "harder to escape" in the case of 3 interacting vectors (as evidenced by the longer loss plateau in the black curve versus the blue curve in Figure 4.6a). Here, we have a factor of ε^3 rather than ε^2 (in the two vector case). The intuition extends: if the number of components is increased further (beyond 3), the saddle point will get harder and harder to escape.

C.5 Implementation details of clamping for induction head formation

In Section 4.4.2, we detail the methodology and inspiration for clamping various subcircuits. Here, we detail the specifics of implementation of each of the clamped steps.

C.5.1 Clamping weights vs activations: Complications due to routing

We first demonstrate that clamping isn't as easy as just fixing weights during training. Recall that we want to clamp computation steps (Section 4.4.1) that comprise an induction circuit. There are many ways (in terms of weights) the network can implement some of these computation steps, as many of them involve reading from and/or writing into a subspace of the residual stream (e.g., Steps 2 and 3). Arbitrary rotations could be applied to these subspaces.[2] The specific subspaces used appear to be initialization dependent, as shown in Figure C.4.

We consider fixing the weights of Layer 1 to match those of networks pre-trained on our task, but from different initializations. Specifically, we train networks on our task from different initialization seeds (which we will refer to as seeds[3] 5, 6, 7). All networks learn induction circuits, with multiple previous token heads appearing in Layer 1. We then train a network initialized from seed 5 with the residual stream after Layer 1 clamped to the residual stream after Layer 1 of one of the fully trained networks (equivalent to fixing all the weights up to this point). Note that these activations are fixed for a given input, so gradients are only flowing to Layer 2 of the model (as desired, since we are clamping Layer 1 to be an externally "perfect" Layer 1 based on weights). The clamped Layer 1 is theoretically perfectly performing Steps 1 and 2. However, we can see that there is substantial variance when the weights used are from the same seed as that used for the Layer 2 initialization (red, Figure C.4) versus when the weights used are from a different seed (blue or green, Figure C.4). These results point to the importance of Step 3, routing, and tell us that fixing weights may not be the best way to clamp subcircuits as it may be confounded by the specific methodology used (specifically, which network the weights come from).

[2]This has been referred to the as *identification failures* by some (Shalizi, 2024).

[3]These values are arbitrary and are the first three we tried. We didn't use seed 0 as this was the seed for our data sampler.

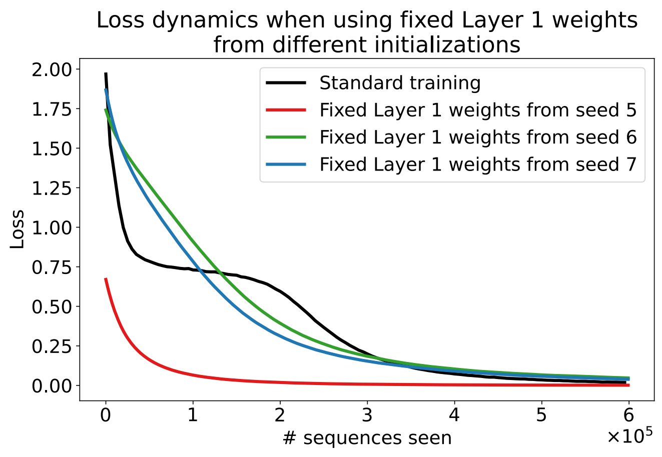


Figure C.4: Loss curves when fixing Layer 1 weights to be those found at the end of training with various seeds. We see that when using the Layer 1 weights from the end of training off of the same initialization seed (seed 5) learning is way quicker, indicating that specific implementations of computations 1 and 2 in Section 4.4.1 are tightly coupled to initialization. Furthermore, these results suggest that fixing weights is not the ideal form of clamping as it is very susceptible to correlations between the chosen weights and network initialization.

C.5.2 Activation clamps that we used

Building off this insight, we instead implement our clamps at the activation level. While this does restrict the set of useful clamps we may apply, all the clamps used in Section 4.4.2 can be expressed at the activation level. We detail each clamp below:

Clamping Step 1 (PT-attend) We fix the attention pattern of a Layer 1 head to always attend to its previous token (with post-softmax value 1), corresponding to clamping Step 1 from Section 4.4.1. In Section 4.4.2, we clamped Layer 1 Head 2's attention pattern. In Figure C.5, we show the effect of this clamp applied to different sets of Layer 1 heads. We find that overall there is little difference, unless all heads are clamped, in which case learning is severely slowed. We hypothesize that some Layer 1 heads speed up learning of copy circuits (which is consistent with the findings of V-composition in Appendix C.3) – by forcing all Layer 1 heads to have the previous token attention pattern, we slow this mechanism. We didn't investigate this further, however, as this was not the focus of our work, and instead used a clamp on just one previous token head for our main experiments.

Clamping Steps 1, 2 (layer 1) To implement this clamp while taking into account the concerns related to routing from Appendix C.5.1, we consider how Layer 1

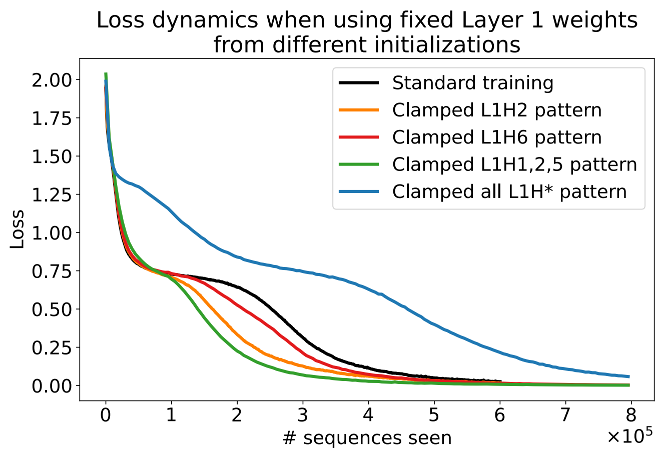


Figure C.5: Effect of different implementations of the previous token attention pattern clamp (clamping computation Step 1, using the terminology of Section 4.4.1). We see that most versions of the clamp have little effect on dynamics (just slight left or right shifts in the phase change), with the exception of clamping all heads in Layer 1. We suspect this may be due to some Layer 1 heads V-composing with Layer 2 heads to make copy circuits easier to learn.

outputs are used by Layer 2. We utilize two forward passes through the network. In the first, we set the input to Layer 2 to be equivalent to the output from previous token heads and perfect query routing. For example, for the sequence “A 0 B 1 A”, this would look like “A A 0 B A”, where we note that the first token cannot attend to a previous token (hence it still has ‘A’), the next 3 tokens attend to previous tokens (so ‘0 B 1’ becomes ‘A 0 B’), and the query token still routes from the input (‘A’). We directly pass the embeddings of these tokens to Layer 2, and cache the *patterns*. Then, we do a second forward pass using these patterns, except now we clamp the input to Layer 2 to match the input to the network (“A 0 B 1 A”) which is what the value routing should be reading from. We note that this clamp is making the task a bit easier for the network as the vectors that Routing QK needs to read from and Routing V needs to read from are not additively superimposed, but these routing steps need to learn to read from the correct subspaces. We opted for this option as it makes it even more telling that there are some subcircuit interactions in the remaining steps after applying this clamp.

Clamping Steps 1, 2, 3qk, 4 (match) We implement this by fixing the attention pattern of a Layer 2 head (we picked Layer 2 Head 3 as described in Section 4.4.2) to be that of a perfect induction head – 1 to the correct label token and 0 to all other tokens. The aim of this clamp is to make sure the correct label is matched to (thus isolating the copy operation), which this implementation suffices to do.

Clamping Steps 3v, 5 (copy) We implement this clamp by considering the attention logits to the two labels in-context at a given induction head (we picked Layer 2 Head 3 as described in Section 4.4.2) and setting the output logits to them. For example, on the sequence "A 0 B 1 A", the attention logits from the query token 'A' to the label tokens '0' and '1' would be the output logits for labels 0 and 1. The other output logits are set to $-1e9$. The aim of this clamp is to make sure labels are perfectly copied (thus isolating the match operation), which this implementation suffices to do.

Clamping Steps 1, 2, 3v, 5 We implement this clamp by combining the clamp for Steps 1, 2 (layer 1), and the clamp for Steps 3v, 5 (copy). As each of these clamps are disjoint, it's easy to apply both, serving the intended purpose.

C.6 Reproducibility across model initializations

Our code, as well as notebooks reproducing findings on multiple initialization seeds (5 seeds) can be found at `https://github.com/aadityasingh/icl-dynamics`. We found little to no differences when training seed (responsible for data ordering) was varied. Here, we summarize qualitatively the results that differ slightly across init seeds and the ones that don't (rather than providing $4 \cdot 6 = 24$ new figures).

Figure C.6 summarizes a reproduction of some of the results of Section 4.3.1. Specifically, strongest heads (determined via induction strength) are mostly able to solve the task. We find that the connection to the *neural race* (Saxe et al., 2022) mostly holds across seeds (first discussed in Section 4.3.2, though the correlation is not as strong as other metrics related to induction strength. Our main results on sub-circuit discovery (Figure 4.6) hold across seeds. The scaling result (Figure 4.9) reproduces across most seeds, though sometimes in standard training we see learning with 15 labels to be as quick as learning with 10 labels. For further details, we

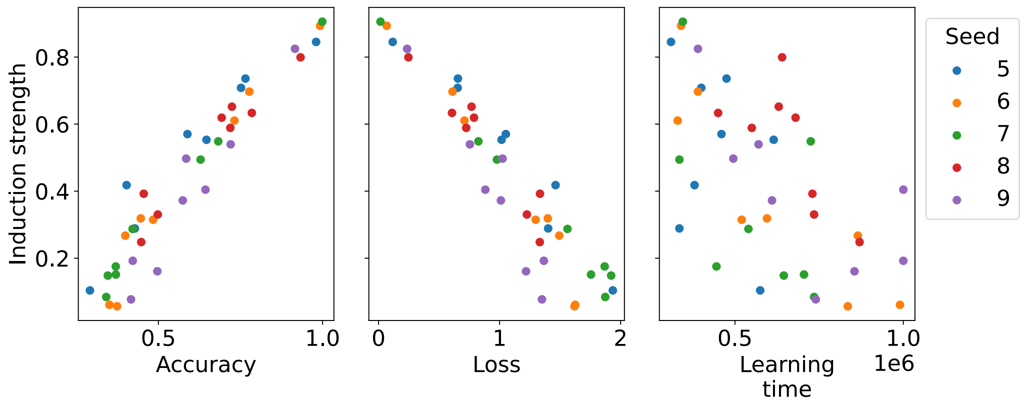


Figure C.6: Reproducing some of the results of Section 4.3.1 across seeds. Specifically, we plot the induction strength of all 8 L2 heads across 5 seeds (40 points) versus various other metrics on individual heads: accuracy and loss if all other L2 heads are ablated (analogous to circles in Figure 4.5a), as well as the “learning time”, which we define as the number of sequences seen such that the loss drops below $0.4 \cdot \log 2$ when training with a single head (a summary statistic for Figure 4.5b). As we can see, induction head strength is well correlated with accuracy and loss (lower loss is better, and corresponds to stronger heads) across seeds. We see that learning time (and thus, connections to the neural race reduction (Saxe et al., 2022)) is correlated to strength across seeds (faster learning times correlate to stronger heads), but not as strongly. Future work could investigate these ideas of circuits “racing” to minimize the loss further.

refer readers to the notebooks in our codebase.

Appendix D

Appendix for Chapter 5

D.1 Extended behaviors seen in small settings

In this section, we provide additional results on alternative setups we considered—specifically, using other types of positional embeddings, interaction with data properties considered in Chapter 2, and potentially using MLP layers in a 2L network (with absolute positional embeddings).

All our plots in this section feature accuracy on the IWL evaluator as well. Note that in-context accuracy on this evaluator is meaningless, as the correct label does not appear in context. As a result, we just plot the direct accuracy (for which chance level would be $\frac{1}{\#classes}$). Most of the times, this accuracy stays at chance level, though we note settings where it seems to rise.

Runs in this section are also often run for longer than our main experiments—we truncated the plots in the main paper to focus on the relevant phenomena. Here, we present the full timecourses we ran in case it's useful for future researchers.

D.1.1 Different random seeds

To confirm our reproduction, we experimented over a few seeds (for both model initialization and data generation/ordering)—see Figure D.1. We found data seed to make little-to-no difference. Model intialization can change the exact timing of transience, but the general profiles of all the curves are the same.

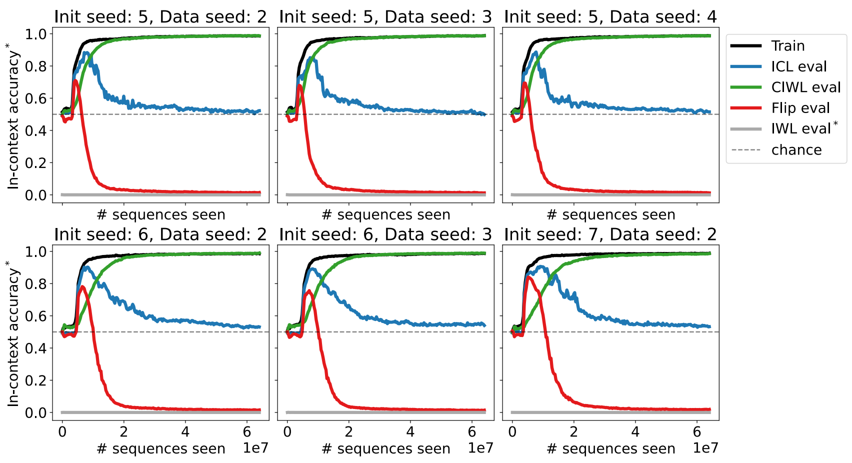


Figure D.1: Reproduction of our main setup (Section 5.2) over random seeds. Top left figure is same as Figure 5.1b.

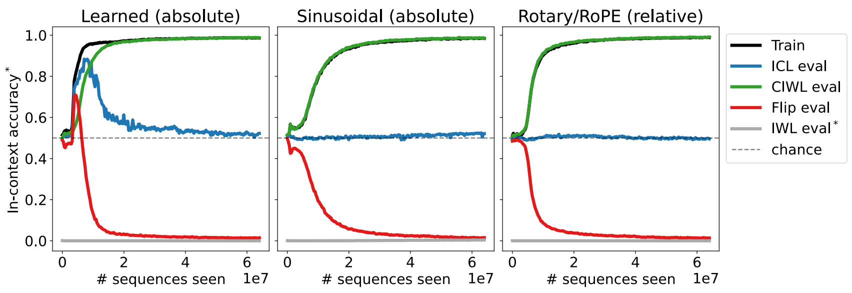


Figure D.2: Changes in behavior when using different positional embedding schemes. Left figure is the same as Figure 5.1b. Note that learned absolute positional embeddings are needed to get ICL emergence in a 2L attention-only transformer on our task.

D.1.2 Other positional embeddings

We considered absolute sinusoidal positional embeddings (introduced by Vaswani et al. (2017) and used in the our earlier work (Chapters 2 and 3) as well as rotary positional embeddings (RoPE, introduced in (Su et al., 2021) and used in Chapter 4). However, we found that, for two-layer attention only networks, learned absolute positional embeddings were needed to reproduce the emergence and transience of ICL. While larger networks don't require this added flexibility, we found it necessary to elicit emergent ICL, which we speculate may be due to the difficulty of learning previous token heads in Layer 1 otherwise (Olsson et al., 2022).

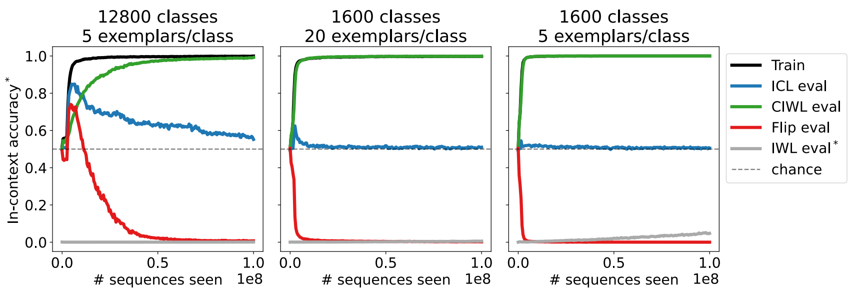


Figure D.3: Changes in behavior under different data properties.

D.1.3 ICL does not emerge with lower classes or fewer exemplars

We consider different #'s of classes and different #'s of exemplars. In Section 2.3.4, we found that higher numbers of both incentivized ICL emergence, and in Section 3.4.2, we found that more classes slowed down transience. In Figure D.3, we find that larger number of classes are necessary for ICL to dominate over CIWL for a time (Flip evaluator accuracy > 0.5). We still see smaller transience effects with 1600 classes, as long as 20 exemplars are used. When reducing both (1600 classes, 5 exemplars), we see no ICL emergence, and even some asymptotic strengthening of pure in-weights mechanisms.

These findings match what we saw previously on 12-layer models, and we hypothesize that when # of classes is lower, the CIWL mechanism emerges "too quickly" for ICL to show up (see Section 5.6.4).

D.1.4 Behavior with MLPs

We also considered the use of 2-layer transformers with MLPs as well, as are typically used in practice (Vaswani et al., 2017). We use a standard 4x expansion factor and GeLU activations (Hendrycks and Gimpel, 2016). Results in Figure D.4.

While our main setting (12800 classes, 20 exemplars) seems to have similar trends, we observed transience and resurgence in other settings (12800 classes, 5 exemplars), which make it difficulty to claim asymptotic behaviors in these small models with MLP. We thus restricted our main analysis to the attention-only models, as is common for mechanistic work on transformers (Elhage et al., 2021; Olsson et al., 2022).

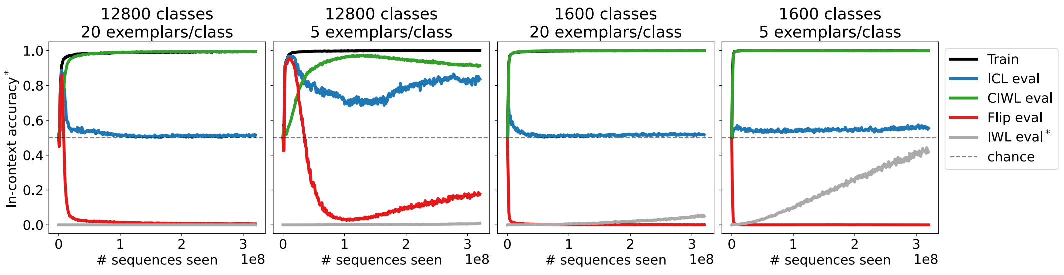


Figure D.4: Changes in behavior when training 2-layer transformers with MLPs. In our main setting (left), we see similar trends, but when training on 12800 classes with 5 exemplars each, we get transient and resurgent ICL (unlike the corresponding run without MLPs, Figure D.3, left) In the rightmost plot, we see significantly increased pure in-weights learning at low numbers of classes and exemplars (as compared to Figure D.3), which may be due to the hypothesized role of MLPs in such pathways (Geva et al., 2021; Meng et al., 2023).

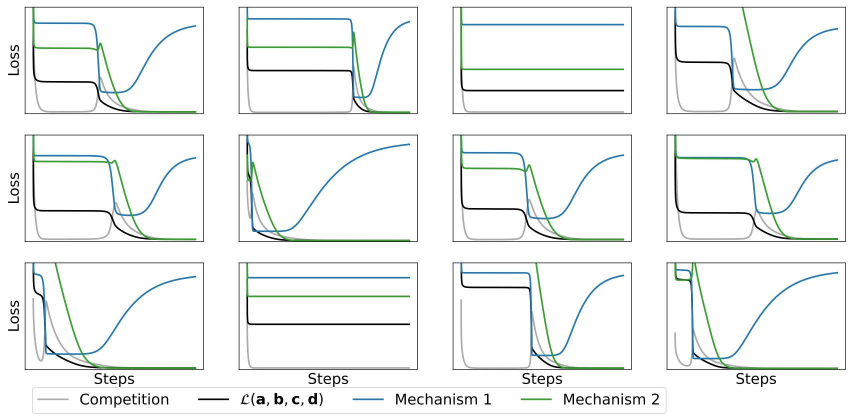


Figure D.5: Simulations of the toy model over 12 seeds, using the same settings as those for Figure 5.9: $dim(\mathbf{a}) = dim(\mathbf{b}) = dim(\mathbf{c}) = 20, dim(\mathbf{d}) = 160, \mu_1 = 0.1, \alpha = 0.1$. We additionally show the competition term in gray.

D.2 Additional details and seeds on toy model

All vectors in our toy model (Section 5.7) are initialized randomly with each element being drawn from $\mathcal{N}(0,1)$. Vectors evolve through gradient descent, with a learning rate of 1. In practice, we use a scaled Frobenius norm in the loss function, where we normalize for the size of the tensors in each expression, to avoid needing to tune the learning rate. Mathematically, this rescaling is isomorphic to a corresponding setting of μ_1, α and learning rate.

In Figure D.5, we show simulations on 12 random seeds (to supplement the single seed shown in Figure 5.9). While the exact timing in magnitudes of the

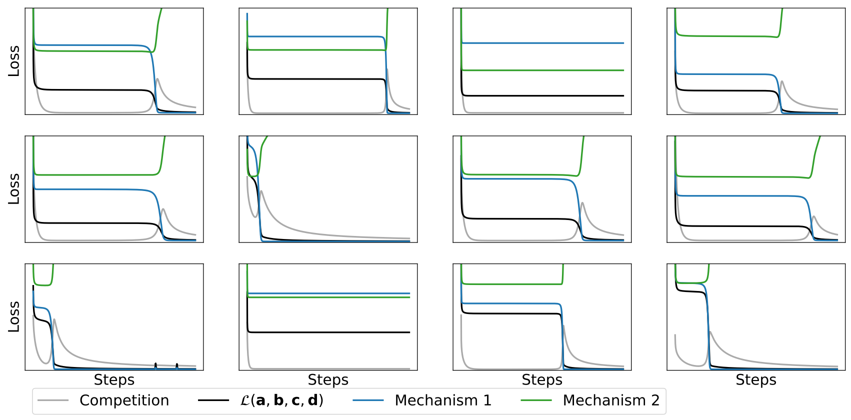


Figure D.6: Simulations of the toy model over 12 seeds, using the settings: $dim(\mathbf{a}) = dim(\mathbf{b}) = dim(\mathbf{c}) = 20, dim(\mathbf{d}) = 160, \mu_1 = 0, \alpha = 0.1$. Without the asymptotic bias $\mu_1 = 0$, the faster Mechanism 1 gets learned first and dominates. $\mathbf{d} \to \mathbf{0}$ due to the competition term, which leads to increasing loss of Mechanism 2 (green).

curves can shift around, the overall dynamical profile (transience of Mechanism 1, the divot in Mechanism 2 formation) is largely preserved. For some seeds, we found that the vectors did not exit the loss plateau (likely caused by a saddle point in the loss landscape, analogous to that in the simpler model, see Appendix C.4) for the duration of our simulation.

In Figure D.6, we show simulations on 12 random seeds with the same settings as Figure D.5, except that $\mu_1 = 0$ instead of $\mu_1 = 0.1$. Eliminating the asymptotic bias for Mechanism 2 gets rid of transient behavior, with the faster mechanism (Mechanism 1) emerging and dominating. These intuitions motivated our experiments in Section 5.8.

Bibliography

OpenAI. ChatGPT, 2024. URL chat.com.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(89)90020-8. URL `https://www.sciencedirect.com/science/article/pii/0893608089900208`.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning representations by back-propagating errors*, page 696–699. MIT Press, Cambridge, MA, USA, 1988. ISBN 0262010976.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

OpenAI. GPT-4 Technical Report, 2023. URL `https://arxiv.org/abs/2303.08774`.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850. PMLR, 2016.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Pedro A Ortega, Jane X Wang, Mark Rowland, Tim Genewein, Zeb Kurth-Nelson, Razvan Pascanu, Nicolas Heess, Joel Veness, Alex Pritzel, Pablo Sprechmann, et al. Meta-learning of sequential strategies. *arXiv preprint arXiv:1905.03030*, 2019.

Jerry Wei, Jason Wei, Yi Tay, Dustin Tran, Albert Webson, Yifeng Lu, Xinyun Chen, Hanxiao Liu, Da Huang, Denny Zhou, et al. Larger language models do in-context learning differently. *arXiv preprint arXiv:2303.03846*, 2023.

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

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023a.

Noam Shazeer. Glu variants improve transformer, 2020. URL `https://arxiv.org/abs/2002.05202`.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2016. URL `https://arxiv.org/abs/1606.08415`.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *CoRR*, abs/2104.09864, 2021. URL `https://arxiv.org/abs/2104.09864`.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

Aaditya K. Singh, Ted Moskovitz, Felix Hill, Stephanie C. Y. Chan, and Andrew M. Saxe. What needs to go right for an induction head? a mechanistic study of in-context learning circuits and their formation. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

George Kingsley Zipf. *Human behavior and the principle of least effort: An introduction to human ecology*. Ravenio Books, 1949.

Steven T. Piantadosi. Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic bulletin & review*, 21(5):1112–1130, October 2014. ISSN 1069-9384. doi: 10.3758/s13423-014-0585-6. URL `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4176592/`.

Linda B. Smith, Swapnaa Jayaraman, Elizabeth Clerkin, and Chen Yu. The Developing Infant Creates a Curriculum for Statistical Learning. *Trends in Cognitive Sciences*, 22(4):325–336, April 2018. ISSN 1364-6613. doi: 10.1016/j.tics.2018.02.004. URL `https://www.sciencedirect.com/science/article/pii/S1364661318300275`.

Avik Sarkar, Paul H. Garthwaite, and Anne De Roeck. A Bayesian mixture model for term re-occurrence and burstiness. In *Proceedings of the Ninth Conference on Computational Natural Language Learning - CONLL '05*, page 48, Ann Arbor, Michigan, 2005. Association for Computational Linguistics. doi: 10.3115/1706

543.1706552. URL `http://portal.acm.org/citation.cfm?doid=1706543.1706552`.

E. Alvarez-Lacalle, B. Dorow, J.-P. Eckmann, and E. Moses. Hierarchical structures induce long-range dynamical correlations in written texts. *Proceedings of the National Academy of Sciences*, 103(21):7956–7961, May 2006. doi: 10.1073/pnas.0510673103. URL `https://www.pnas.org/doi/abs/10.1073/pnas.0510673103`. Publisher: Proceedings of the National Academy of Sciences.

Marcel F. Neuts. The burstiness of point processes. *Stochastic Models*, April 2007. doi: 10.1080/15326349308807275. URL `https://www.tandfonline.com/doi/abs/10.1080/15326349308807275`. Publisher: Marcel Dekker, Inc.

Eduardo G. Altmann, Janet B. Pierrehumbert, and Adilson E. Motter. Beyond Word Frequency: Bursts, Lulls, and Scaling in the Temporal Distributions of Words. *PLoS ONE*, 4(11):e7678, November 2009. ISSN 1932-6203. doi: 10.1371/journal.pone.0007678. URL `https://dx.plos.org/10.1371/journal.pone.0007678`.

M. Angeles Serrano, Alessandro Flammini, and Filippo Menczer. Modeling Statistical Properties of Written Text. *PLOS ONE*, 4(4):e5372, April 2009. ISSN 1932-6203. doi: 10.1371/journal.pone.0005372. URL `https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0005372`. Publisher: Public Library of Science.

Renaud Lambiotte, Lionel Tabourier, and Jean-Charles Delvenne. Burstiness and spreading on temporal networks. *The European Physical Journal B*, 86(7):320, July 2013. ISSN 1434-6028, 1434-6036. doi: 10.1140/epjb/e2013-40456-9. URL `http://arxiv.org/abs/1305.0543`. arXiv:1305.0543 [physics, q-bio].

Stephanie Chan, Adam Santoro, Andrew Lampinen, Jane Wang, Aaditya Singh, Pierre Richemond, James McClelland, and Felix Hill. Data distributional properties drive emergent in-context learning in transformers. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 18878–18891. Curran Associates, Inc., 2022a. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/77c6ccacfd9962e2307fc64680fc5ace-Paper-Conference.pdf`.

Aaditya Singh, Stephanie Chan, Ted Moskovitz, Erin Grant, Andrew Saxe, and Felix Hill. The transient nature of emergent in-context learning in transformers. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 27801–27819. Curran Associates, Inc., 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/file/58692a1701314e09cbd7a5f5f3871cc9-Paper-Conference.pdf`.

Suraj Anand, Michael A. Lepori, Jack Merullo, and Ellie Pavlick. Dual Process Learning: Controlling Use of In-Context vs. In-Weights Strategies with Weight Forgetting, May 2024. URL `http://arxiv.org/abs/2406.00053`. arXiv:2406.00053 [cs] version: 1.

Tianyu He, Darshil Doshi, Aritra Das, and Andrey Gromov. Learning to grok: Emergence of in-context learning and skill composition in modular arithmetic tasks, June 2024. URL `http://arxiv.org/abs/2406.02550`. arXiv:2406.02550 [cond-mat, physics:hep-th, stat].

Core Francisco Park, Ekdeep Singh Lubana, Itamar Pres, and Hidenori Tanaka. Competition Dynamics Shape Algorithmic Phases of In-Context Learning, December 2024. URL `http://arxiv.org/abs/2412.01003`. arXiv:2412.01003 [cs].

Alex Nguyen and Gautam Reddy. Differential learning kinetics govern the transition from memorization to generalization during in-context learning, November

2024. URL http://arxiv.org/abs/2412.00104. arXiv:2412.00104 [cs].

Neel Nanda. 200 cop in mi: Analysing training dynamics, 2023a. URL https://www.alignmentforum.org/posts/hHaXzJQi6SKkeXzbg/200-cop-in-mi-analysing-training-dynamics.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability, 2023.

Gautam Reddy. The mechanistic basis of data dependence and abrupt learning in an in-context classification task, 2023.

Ari S. Morcos, David G. T. Barrett, Neil C. Rabinowitz, and Matthew Botvinick. On the importance of single directions for generalization, 2018. URL https://arxiv.org/abs/1803.06959.

Yonatan Belinkov. Probing classifiers: Promises, shortcomings, and advances, 2021. URL https://arxiv.org/abs/2102.12452.

Lucas Hayne, Abhijit Suresh, Hunar Jain, Rahul Kumar, and R. McKell Carter. Much easier said than done: Falsifying the causal relevance of linear decoding methods, 2022. URL https://arxiv.org/abs/2211.04367.

OpenAI. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024c. URL https://arxiv.org/abs/2412.16720.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL `https://arxiv.org/abs/2501.12948`.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29, 2016.

Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266): 1332–1338, 2015. doi: 10.1126/science.aab3050. URL `https://www.science.org/doi/abs/10.1126/science.aab3050`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

William A. Gale, Kenneth W. Church, and David Yarowsky. One Sense Per Discourse. In *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23-26, 1992*, 1992. URL `https://aclanthology.org/H92-1045`.

W. Nelson Francis and Henry Kucera. Brown Corpus Manual, 1979. URL `http://korpus.uib.no/icame/brown/bcm.html`.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. Technical report, 1985. URL `https://apps.dtic.mil/dtic/tr/fulltext/u2/a164453.pdf`.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. Conference Name: Neural Computation.

Felix Hill, Olivier Tieleman, Tamara von Glehn, Nathaniel Wong, Hamza Merzic, and Stephen Clark. Grounded Language Learning Fast and Slow. *arXiv:2009.01719 [cs]*, October 2020. URL `http://arxiv.org/abs/2009.01719`. arXiv: 2009.01719.

Stephanie C. Y. Chan, Andrew K. Lampinen, Pierre H. Richemond, and Felix Hill. Zipfian environments for Reinforcement Learning. *arXiv:2203.08222 [cs]*, March 2022b. URL `http://arxiv.org/abs/2203.08222`. arXiv: 2203.08222.

N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16:321–357, June 2002. ISSN 1076-9757. doi: 10.1613/jair.953. URL `http://arxiv.org/abs/1106.1813`. arXiv: 1106.1813.

Jason Van Hulse, Taghi M. Khoshgoftaar, and Amri Napolitano. Experimental perspectives on learning from imbalanced data. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pages 935–942, New York, NY, USA, June 2007. Association for Computing Machinery. ISBN 978-1-59593-793-3. doi: 10.1145/1273496.1273614. URL `https://doi.org/10.1145/1273496.1273614`.

Angelos Katharopoulos and François Fleuret. Not All Samples Are Created Equal: Deep Learning with Importance Sampling. *arXiv:1803.00942 [cs]*, October 2019. URL `http://arxiv.org/abs/1803.00942`. arXiv: 1803.00942.

Machel Reid, Yutaro Yamada, and Shixiang Shane Gu. Can Wikipedia Help Offline Reinforcement Learning? *arXiv:2201.12122 [cs]*, January 2022. URL `http://arxiv.org/abs/2201.12122`. arXiv: 2201.12122.

Sharath Chandra Raparthy, Eric Hambro, Robert Kirk, Mikael Henaff, and Roberta Raileanu. Generalization to New Sequential Decision Making Tasks with In-Context Learning, December 2023. URL `http://arxiv.org/abs/2312.03801`. arXiv:2312.03801 [cs].

Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

Yao Fu, Hao Peng, Litu Ou, Ashish Sabharwal, and Tushar Khot. Specializing smaller language models towards multi-step reasoning. *arXiv preprint arXiv:2301.12726*, 2023.

Yukun Huang, Yanda Chen, Zhou Yu, and Kathleen McKeown. In-context learning distillation: Transferring few-shot learning ability of pre-trained language models, 2022.

Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english?, 2023.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets. Technical Report arXiv:2201.02177, arXiv, January 2022. URL `http://arxiv.org/abs/2201.02177`. arXiv:2201.02177 [cs] type: article.

Vimal Thilak, Etai Littwin, Shuangfei Zhai, Omid Saremi, Roni Paiss, and Joshua Susskind. The Slingshot Mechanism: An Empirical Study of Adaptive Optimizers and the Grokking Phenomenon, June 2022. URL `http://arxiv.org/abs/2206.04817`. arXiv:2206.04817 [cs, math].

Ziming Liu, Eric J. Michaud, and Max Tegmark. Omnigrok: Grokking beyond algorithmic data, 2023. URL `https://arxiv.org/abs/2210.01117`.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories, 2021.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt, 2023.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. *arXiv preprint arXiv:2111.02080*, 2021.

Shivam Garg, Dimitris Tsipras, Percy S Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 30583–30598. Curran Associates, Inc., 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/c529dba08a146ea8d6cf715ae8930cbe-Paper-Conference.pdf`.

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models. *arXiv preprint arXiv:2211.15661*, 2022.

Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. *arXiv preprint arXiv:2212.07677*, 2022.

Ethan Perez, Douwe Kiela, and Kyunghyun Cho. True few-shot learning with language models, 2021.

Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017. URL `http://arxiv.org/abs/1711.05101`.

Edward S Boyden, Feng Zhang, Ernst Bamberg, Georg Nagel, and Karl Deisseroth. Millisecond-timescale, genetically targeted optical control of neural activity. *Nature Neuroscience*, 8(9):1263–1268, September 2005. ISSN 1097-6256, 1546-1726. doi: 10.1038/nn1525. URL `https://www.nature.com/articles/nn1525`.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL `http://arxiv.org/abs/1207.0580`.

Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *CoRR*, abs/1905.10650, 2019. URL `http://arxiv.org/abs/1905.10650`.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1580. URL `https://aclanthology.org/P19-1580`.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

DeepMind, Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Laurent Sartran, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Miloš Stanojević, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola. The DeepMind JAX Ecosystem, 2020. URL `http://github.com/google-deepmind`.

Patrick Kidger and Cristian Garcia. Equinox: neural networks in JAX via callable PyTrees and filtered transformations. *Differentiable Programming workshop at Neural Information Processing Systems 2021*, 2021.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

PyTorch. We just introduced pytorch 2.0 at the #pytorchconference, introducing torch.compile!, 2022. URL `https://x.com/PyTorch/status/1598708792598069249?s=20`.

Alan Cooney. [proposal] improve performance with torch.compile, 2023. URL `https://github.com/neelnanda-io/TransformerLens/issues/413`.

Omkar Ranadive, Nikhil Thakurdesai, Ari S Morcos, Matthew Leavitt, and Stéphane Deny. On the special role of class-selective neurons in early training, 2023.

Angelica Chen, Ravid Shwartz-Ziv, Kyunghyun Cho, Matthew L. Leavitt, and Naomi Saphra. Sudden drops in the loss: Syntax acquisition, phase transitions, and simplicity bias in mlms, 2024.

nostalgebraist. interpreting gpt: the logit lens, 2020. URL `https://www.alig` nmentforum.org/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens.

Thomas McGrath, Matthew Rahtz, Janos Kramar, Vladimir Mikulik, and Shane Legg. The hydra effect: Emergent self-repair in language model computations, 2023.

Justin A Hennig, Matthew D Golub, Parker J Lund, Patrick T Sadtler, Emily R Oby, Kristin M Quick, Stephen I Ryu, Elizabeth C Tyler-Kabara, Aaron P Batista, Byron M Yu, and Steven M Chase. Constraints on neural redundancy. *eLife*, 7: e36774, 2018. doi: 10.7554/eLife.36774. URL https://elifesciences.org/articles/36774.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Training pruned neural networks. *CoRR*, abs/1803.03635, 2018. URL http://arxiv.org/abs/1803.03635.

Neel Nanda. 200 concrete open problems (COP) in MI: Analysing training dynamics, 2023b. URL https://www.alignmentforum.org/posts/hHaXzJQi6SKkeXzbg/200-cop-in-mi-analysing-training-dynamics.

Andrew M. Saxe, Shagun Sodhani, and Sam Lewallen. The neural race reduction: Dynamics of abstraction in gated networks, 2022.

Adam Jermyn and Buck Shlegeris. Multi-component learning and s-curves, 2022. URL https://www.alignmentforum.org/posts/RKDQCB6smLWgs2Mhr/multi-component-learning-and-s-curves.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar

Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023b.

Keunwoo Peter Yu, Zheyuan Zhang, Fengyuan Hu, and Joyce Chai. Efficient In-Context Learning in Vision-Language Models for Egocentric Videos, November 2023. URL `http://arxiv.org/abs/2311.17041`. arXiv:2311.17041 [cs].

Daniel D. Johnson. Penzai + Treescope: A toolkit for interpreting, visualizing, and editing models as data. *ICML 2024 Workshop on Mechanistic Interpretability*, 2024.

János Kramár. Mishax: a utility library for mechanistic interpretability research, 2024. URL `https://github.com/google-deepmind/mishax`.

Neel Nanda and Joseph Bloom. Transformerlens. `https://github.com/neelnanda-io/TransformerLens`, 2022.

Jaden Fiotto-Kaufman. nnsight: The package for interpreting and manipulating the internals of deep learned models. . URL `https://github.com/JadenFiotto-Kaufman/nnsight`.

Jaden Fiotto-Kaufman, Alexander R Loftus, Eric Todd, Jannik Brinkmann, Caden Juang, Koyena Pal, Can Rager, Aaron Mueller, Samuel Marks, Arnab Sen Sharma, Francesca Lucchetti, Michael Ripa, Adam Belfki, Nikhil Prakash, Sumeet Multani, Carla Brodley, Arjun Guha, Jonathan Bell, Byron Wallace, and

David Bau. Nnsight and ndif: Democratizing access to foundation model internals. 2024. URL `https://arxiv.org/abs/2407.14561`.

Zhengxuan Wu, Atticus Geiger, Aryaman Arora, Jing Huang, Zheng Wang, Noah D. Goodman, Christopher D. Manning, and Christopher Potts. pyvene: A library for understanding and improving PyTorch models via interventions. 2024. URL `arxiv.org/abs/2403.07809`.

Nicholas Goldowsky-Dill, Chris MacLeod, Lucas Sato, and Aryaman Arora. Localizing model behavior with path patching, 2023.

David Lindner, János Kramár, Matthew Rahtz, Thomas McGrath, and Vladimir Mikulik. Tracr: Compiled transformers as a laboratory for interpretability. *arXiv preprint arXiv:2301.05062*, 2023.

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small, 2022.

Arthur Conmy, Augustine N. Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability, 2023.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart M. Shieber. Causal mediation analysis for interpreting neural NLP: the case of gender bias. *CoRR*, abs/2004.12265, 2020. URL `https://arxiv.org/abs/2004.12265`.

Atticus Geiger, Hanson Lu, Thomas Icard, and Christopher Potts. Causal abstractions of neural networks. *CoRR*, abs/2106.02997, 2021. URL `https://arxiv.org/abs/2106.02997`.

Nicola De Cao, Leon Schmid, Dieuwke Hupkes, and Ivan Titov. Sparse interventions in language models with differentiable masking. *CoRR*, abs/2112.06837, 2021. URL `https://arxiv.org/abs/2112.06837`.

Mor Geva, Jasmijn Bastings, Katja Filippova, and Amir Globerson. Dissecting recall of factual associations in auto-regressive language models, 2023.

Matthew Finlayson, Aaron Mueller, Sebastian Gehrmann, Stuart M. Shieber, Tal Linzen, and Yonatan Belinkov. Causal analysis of syntactic agreement mechanisms in neural language models. *CoRR*, abs/2106.06087, 2021. URL `https://arxiv.org/abs/2106.06087`.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy models of superposition. *Transformer Circuits Thread*, 2022. URL `https://transformer-circuits.pub/2022/toy_model/index.html`.

Bryan Chan, Xinyi Chen, András György, and Dale Schuurmans. Toward Understanding In-context vs. In-weight Learning, October 2024. URL `http://arxiv.org/abs/2410.23042`. arXiv:2410.23042 [cs].

Xiaolei Wang, Xinyu Tang, Wayne Xin Zhao, and Ji-Rong Wen. Investigating the pre-training dynamics of in-context learning: Task recognition vs. task learning, 2024. URL `https://arxiv.org/abs/2406.14022`.

Andrew Kyle Lampinen, Stephanie C. Y. Chan, Aaditya K. Singh, and Murray Shanahan. The broader spectrum of in-context learning, 2024. URL `https://arxiv.org/abs/2412.03782`.

Kayo Yin and Jacob Steinhardt. Which attention heads matter for in-context learning?, 2025. URL `https://arxiv.org/abs/2502.14010`.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*, 2022.

Ziqian Lin and Kangwook Lee. Dual operating modes of in-context learning, 2024. URL `https://arxiv.org/abs/2402.18819`.

Gemini Team. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024. URL `https://arxiv.org/abs/2403.05530`.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E Burke, Tristan Hume, Shan Carter, Tom Henighan, and Christopher Olah. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. https://transformer-circuits.pub/2023/monosemantic-features/index.html.

Jack Lindsey, Wes Gurnee, Emmanuel Ameisen, Brian Chen, Adam Pearce, Nicholas L. Turner, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. On the biology of a large language model. *Transformer Circuits Thread*, 2025. URL `https://transformer-circuits.pub/2025/attribution-graphs/biology.html`.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. Pythia: A suite for analyzing large language models across training and scaling, 2023.

Yedi Zhang, Aaditya K. Singh, Peter E. Latham, and Andrew Saxe. Training dynamics of in-context learning in linear attention, 2025. URL `https://arxiv.org/abs/2501.16265`.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks, 2014. URL `https://arxiv.org/abs/1312.6120`.

Neel Nanda. A comprehensive mechanistic interpretability explainer & glossary, Dec 2022. URL `https://neelnanda.io/glossary`.

Cosma R. Shalizi. ”attention”, ”transformers”, in neural network ”large language models”, 2024. URL `http://bactra.org/notebooks/nn-attention-and-transformers.html#identification`.