Learning to grok: Emergence of in-context learning and skill composition in modular arithmetic tasks

Tianyu He [1 2] **Darshil Doshi** [* 1 2] **Aritra Das** [* 1] **Andrey Gromov** [1 2 3]

Abstract

Large language models can solve tasks that were not present in the training set. This capability is believed to be due to in-context learning and skill composition. In this work, we study the emergence of in-context learning and skill composition in a collection of modular arithmetic tasks. Specifically, we consider a finite collection of linear modular functions $z = a\,x + b\,y \mod p$ labeled by the vector $(a, b) \in \mathbb{Z}_p^2$. We use some of these tasks for pre-training and the rest for out-of-distribution testing. We empirically show that a GPT-style transformer exhibits a transition from in-distribution to out-of-distribution generalization as the number of pre-training tasks increases. We find that the smallest model capable of out-of-distribution generalization requires two transformer blocks, while for deeper models, the out-of-distribution generalization phase is *transient*, necessitating early stopping. Finally, we perform an interpretability study of the pre-trained models, revealing the highly structured representations in both phases; and discuss the learnt algorithm.

1. Introduction

Large language models (LLMs) can perform simple tasks that have not been present in the training set. This ability is usually achieved via in-context learning (Brown et al., 2020). More importantly, LLMs can perform an even larger variety of very complex tasks upon appropriate prompting or fine-tuning. The latter ability to perform complex tasks is usually attributed to the following mechanism. First, LLMs learn a large variety of simple tasks and, then, how to *compose* those skills to form very complex skills (Arora & Goyal, 2023). Furthermore, LLMs also exhibit "emergent capabilities" – a sudden emergence of a new complex skill as a function of scale (either model parameters, compute or data) (Wei et al., 2022; Ganguli et al., 2022). It is plausible that these sudden performance improvements are due to one or both of these mechanisms. For example, LLMs show grokking on algorithmic tasks (Power et al., 2022), which results from the model learning very structured representations (Liu et al., 2022; Gromov, 2023; Nanda et al., 2023). Once these representations emerge, the model abruptly learns how to perform the task.

In this work, we set out to examine skill composition both empirically and mechanistically. Inspired by the prior work that investigated emergence of in-context learning on linear regression tasks (Ahn et al., 2023; Raventos et al., 2023), we introduce a finite collection of discrete modular arithmetic tasks (Power et al., 2022) generalized to the in-context learning setting. Each task corresponds to learning a linear function $z = a\,x + b\,y$ mod p over $\mathbb{Z}_p$ from the examples provided in the context of the autoregressive model (AM). In the bi-variate case there are p^2 such functions labeled by the vector (a, b). The main objective of this algorithmic dataset is to probe *how* AM utilizes the tasks it has learnt during training to solve the new tasks.

Our analysis shows that the solution found by the AM after optimization is qualitatively different from the linear regression cases studied before (Ahn et al., 2023). In those cases, due to the continuous nature of the task, AM develops an emergent first-order optimization method that minimizes an emergent quadratic loss function. Furthermore, as it was shown in Ahn et al. (2023), a single linear attention layer can solve the regression problem, while adding extra layers and non-linearities slightly modifies the gradient descent. In the modular arithmetic case, AM first learns how to solve the pre-training tasks and later (assuming enough different tasks) develops a generalizing solution by combining the solved tasks.

Our main findings as well as the structure of the algorithmic dataset are illustrated on Fig. 1. Our main findings are: (i) there are **four** different phases in the end of pre-training depending on the number of tasks, $n_{\text{i.d.}}$, and number of examples per task, α. (ii) At inference time, there is a generalization transition in the number of few-shot examples,

[*]Equal contribution [1]Department of Physics, University of Maryland, College Park, MD 20740, USA [2]Condensed Matter Theory Center, University of Maryland, College Park, MD 20740, USA [3]Meta AI. Correspondence to: Tianyu He <tianyuh@umd.edu>.

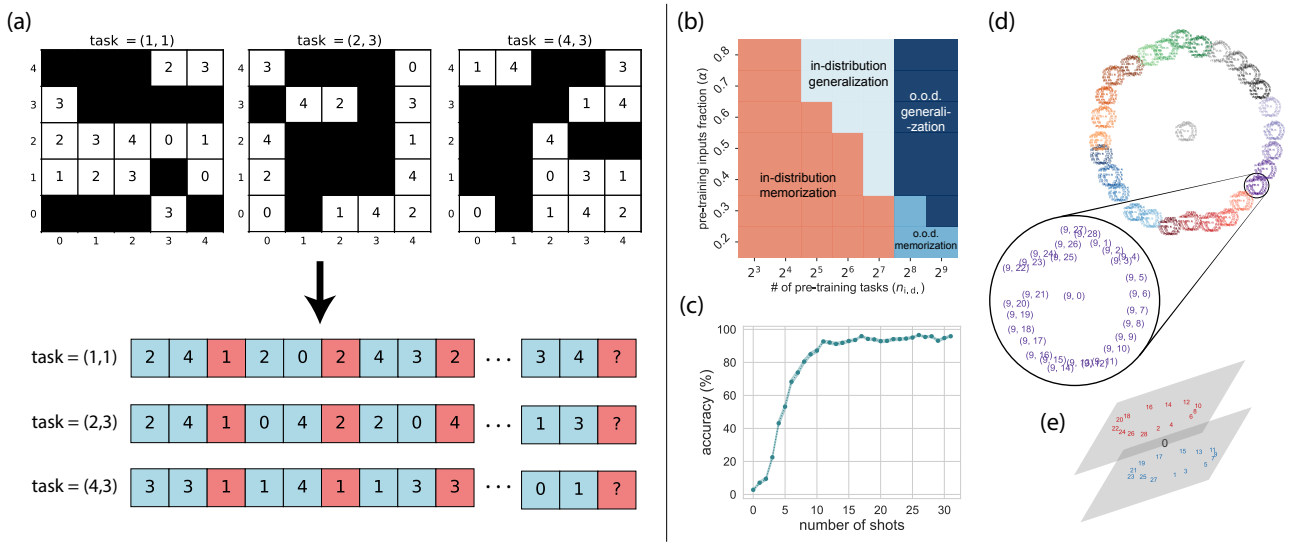


Figure 1: **(a)** The dataset. The tasks are labeled by vectors $(a, b) \in \mathbb{Z}_p^2$. Each table contains examples of $ax + by$ mod p. A fraction $1 - \alpha$ of the examples is blacked out. The remaining examples are flattened into a single "document" in the batch. Each document is organized as a collection of triples $(x, y, ax + by)$ for x, y from the training set (*i.e.* not blacked out in the table). To leverage training compute more effectively we predict not just the last element, but also other elements colored in red. This is similar to the traditional next-token prediction, with the main difference that we predict each third token (x and y do not have any correlation between each other). Every task appears exactly the same number of times in each batch. **(b)** Phase diagram for a six-layer model. We find four different phases. (1) in-distribution memorization: model does not generalize on x, y it has not seen for the training and test tasks. (2) in-distribution generalization: model generalizes on x, y it has not seen for the training tasks, but not for test tasks. (3) out-of-distribution memorization: model does not generalize on x, y it has not seen for test tasks, but completely generalizes on train tasks. (4) out-of-distribution generalization: model generalizes on all x, y for both train and test tasks. We focus on investigating phase (4) in more detail. **(c)** In-context sample complexity. Accuracy of the model in phase (4) as a function of the number of few-shot examples. **(d)** Representations developed by one of the attention heads in the first layer. These are projections of the embedding of a pair of numbers onto the two largest principal components (PCs) of the internal representation formed after passing through the attention layer and projection matrix. **(e)** The first 3 principal components of the embeddings partition the log_{27}-annotated numbers into separate planes of even and odd groups. The number 0 is special, and is sandwiched between the two planes.

as the number of examples grows, the models starts to generalize. This effect is somewhat similar to the transition in sample complexity for the modular arithmetic found in Power et al. (2022). (iii) model develops a striking circular representation for all of the tasks that naturally generalizes the circular representations found in the original work (Power et al., 2022). We further find that the deeper models are easier to optimize, but much harder to interpret. The optimization is discussed in more detail in the main text. Here we will highlight that optimization for these tasks is challenging and the AM tends to prefer the minimum that just solve a handful of tasks and memorize the training set. To avoid such minima we make sure that *every batch* contains equal number of tasks (meaning that no tasks is over- or under-represented in each batch). We further find that for larger models early stopping is necessary because the generalizing solution is transient.

We organize our paper as follows. Section 2 contains the literature review. In Section 3 we explain our notations and discuss the experimental details. In Section 4 we demonstrate empirically that the out-of-distribution ICL ability emerges as the number of training tasks increases. We also study the effects of model depth and task difficulty. In Section 5 we carefully examine a minimal setting, *i.e.* two-block transformer: we compare the representations learned in four different phases and show that in the generalizing phase the representations are highly structured and generalize the original modular addition case of Power et al. (2022).

2. Related Works

In-Context Learning (ICL) Brown et al. (2020) first demonstrated that large models performance improves substantially when a few examples of the task at hand are provided at inference time, in the prompt. Akyürek et al. (2023); Ahn et al. (2023); von Oswald et al. (2023) showed that the

AM implements emergent first-order optimization on an emergent objective function to solve linear regression tasks. Furthermore, Akyürek et al. (2023) showed that larger models learn to perform Bayesian estimation. Garg et al. (2023) demonstrated that transformers can learn several simple classes of functions in context. Kirsch et al. (2024) presented how task diversity and model size would affect the ICL performance for unseen tasks using a mixture of modified MNIST datasets. Raventos et al. (2023) investigated the relation between task diversity and out-of-distribution ICL ability on linear regression tasks. Lin & Lee (2024) identified two operating modes in ICL using a mixture of linear regression tasks, where for the first several shots, the model tries to figure out the correct task vector and later uses it to predict the correct results. Boix-Adserà et al. (2024) showed theoretically and experimentally that with enough pre-training data, a transformer model can perform abstract reasoning that a MLP cannot do. Guo et al. (2024) showed that transformers can use lower layers to memorize and upper layers to perform ICL in a feature regression setting. It was found in Singh et al. (2023) that ICL is a transient phase from the optimization point of view: it goes away once the model is over-trained. Hendel et al. (2023); Liu et al. (2024) showed that language models form in-context vectors, which can be extracted and used to control model predictions.

Modular Arithmetic Power et al. (2022) discovered Grokking, where models trained on modular arithmetic datasets have an abrupt change from random guessing to generalization on the test set way after the model memorized the training set. Gromov (2023); Nanda et al. (2023); Gu et al. (2024) showed that for modular addition tasks, models learned to map integers to Fourier features to solve modular arithmetic tasks. Liu et al. (2022) showed that grokking is related to learning highly structural features, and the grokking transition can be explained by a toy model. Zhong et al. (2023) showed that there is more than one algorithm that a model can implement to solve modular addition. Doshi et al. (2023) showed that corruption of the label does not prevent the models from finding a generalizing solution. Doshi et al. (2024) showed that MLP and transformer models can solve a specific family of modular polynomial tasks by bijectively mapping them to modular addition tasks.

Interpretability Elhage et al. (2021); Olsson et al. (2022) showed that transformers can form induction heads that predict the next token in a sequence by identifying and copying patterns from earlier in the sequence. With several indirect empirical evidences, they showed that those heads might constitute the core mechanism of ICL. Nichani et al. (2024) showed theoretically and empirically how disentangled transformers learn causal structures from in-context Markov chains by forming induction heads.

3. Preliminaries

Linear Modular Functions We consider modular arithmetic tasks of the form: $z_i^t = a^t x_i + b^t y_i \bmod p$. We will refer to the coefficients $(a^t, b^t) \in \mathbb{Z}_p^2$ as the *task vector*. The superscript $t \in [1, p^2]$ labels the p^2 possible tasks. We will refer to $(x_i, y_i) \in \mathbb{Z}_p^2$ as the *input vector*, which is labeled by the subscript $i \in [1, p^2]$.

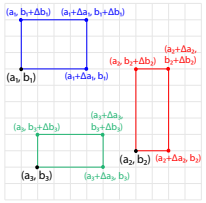
In-Context Learning with Transformers We use GPT-like transformers (Brown et al., 2020) with ReLU activation function and Rotary Positional Embedding (RoPE) (Su et al., 2023). The model has d consecutive blocks, H attention-heads, and embedding dimension d_{embed}. Each number is tokenized as an independent token. The pre-training is done following a slightly modified next-token prediction setup, with sequences of the form:

$$\boldsymbol{s}^t = (x_1 \quad y_1 \quad z_1^t \quad \cdots \quad x_{n_{\text{ctx}}} \quad y_{n_{\text{ctx}}} \quad z_{n_{\text{ctx}}}^t) \in \mathbb{Z}_p^{3 \times n_{\text{ctx}}}, \tag{1}$$

where n_{ctx} is the maximum number of in-context examples. The model is asked to predict *only* the labels $\{z_1^t, \cdots, z_{n_{\text{ctx}}}^t\}$. We emphasize that we do not explicitly provide the task vectors (a^t, b^t) to the model (see Fig. 1) – this information is implicit in the in-context labels $\{z_i^t\}$. In order for the model to generalize, it must determine the underlying task vector (a^t, b^t) from the few-shot examples.

Generalization There are two notions of generalization in this setup. (i) *In-distribution*: Generalization to *unseen* input vectors (x_i, y_i), but on task vector (a^t, b^t) the model has seen during pre-training. (ii) *Out-of-distribution*: Generalization to task vectors the model has *not* seen during pre-training. To clearly separate these regimes, we split the task vectors into in-distribution (i.d.) set $\mathcal{T}_{\text{i.d.}} := \{(a^t, b^t)\}_{\text{i.d.}}$ and out-of-distribution (o.o.d.) set $\mathcal{T}_{\text{o.o.d.}} := \{(a^t, b^t)\}_{\text{o.o.d.}}$. Similarly, we split the input vectors into train and test sets: $\mathcal{X}_{\text{train}} := \{(x_i, y_i)\}_{\text{train}}, \mathcal{X}_{\text{test}} := \{(x_i, y_i)\}_{\text{test}}$. This results in four distinct sets of sequences constructed from those sets; we name them $S_{\text{train}}^{\text{i.d.}}, S_{\text{test}}^{\text{i.d.}}, S_{\text{train}}^{\text{o.o.d.}}$ and $S_{\text{test}}^{\text{o.o.d.}}$. The set $S_{\text{train}}^{\text{i.d.}}$ is used for pre-training, while the other three sets are used for evaluations.

Pre-Training Task Selection and Sequence Design We always sample the pre-training task vectors $\mathcal{T}_{\text{i.d.}}$ in sets of 4, following the rectangular rule, shown in Figure 2(a). Additionally, each batch contains an equal representation from all the task vectors in the set $\mathcal{T}_{\text{i.d.}}$. Moreover, all the tasks share the same sequence of inputs. For example, a batch with two different task vectors $(a^{t_1}, b^{t_1}); (a^{t_2}, b^{t_2})$ and two distinct input sequences per task (resulting in four total sequences) is shown in Figure 2(b).



(a) Pre-training Task Selection with Rectagular Rule

$$\begin{pmatrix} x_{i_1} & y_{i_1} & z_{i_1}^{t_1} & x_{i_2} & y_{i_2} & z_{i_2}^{t_1} & \cdots \\ x_{j_1} & y_{j_1} & z_{j_1}^{t_1} & x_{j_2} & y_{j_2} & z_{j_1}^{t_1} & \cdots \\ x_{i_1} & y_{i_1} & z_{i_1}^{t_2} & x_{i_2} & y_{i_2} & z_{i_2}^{t_2} & \cdots \\ x_{j_1} & y_{j_1} & z_{j_1}^{t_2} & x_{j_2} & y_{j_2} & z_{j_2}^{t_2} & \cdots \end{pmatrix}$$

(b) Pre-training Sequence Design

Figure 2: Structured selection of tasks and sequences.

This structured approach creates a coherent signal from the sequences within each batch; ensuring that the model learns multiple task vectors with reasonable batch sizes. Alternatively, if the batches are sampled i.i.d., then the model is confused by the batch noise and cannot learn any tasks.

Detailed discussions on task selection and sequence design are presented in Appendix D.

Default Setting Unless stated explicitly, we will use $p = 29$, the number of heads $H = 4$, and embedding dimension $d_{\text{embed}} = 512$, with $n_{\text{ctx}} = 32$ in-context examples. All models are trained with AdamW optimizer (Loshchilov & Hutter, 2019) with batch size $B = 1024$ for 200k steps. We have also tied the embedding layer of the model with the readout layer.

4. Emergence of In-Context Learning and Task Composition

In this section, we demonstrate that a transformer model with depth $d \geq 2$ can develop ICL and out-of-distribution generalization on modular arithmetic tasks. We delve deeper into the two notions of generalization (i.d. and o.o.d.), and discuss the relevant factors. We find that the model's ability to generalize out-of-distribution is predominantly determined by the number of pretraining tasks $n_{\text{i.d.}}$.

4.1. Transition driven by the number of tasks

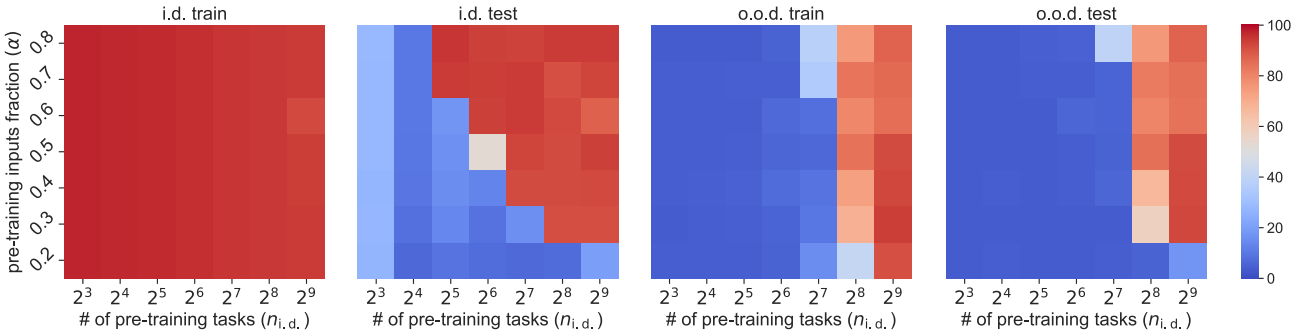
In Figure 3(a), we show the accuracy of $d = 6$ models vs the number of training tasks $n_{\text{i.d.}}$ and the number of few-shot examples quantified by the fraction of the total number of data points, α; on sets $S_{\text{train}}^{\text{i.d.}}$, $S_{\text{test}}^{\text{i.d.}}$, $S_{\text{train}}^{\text{o.o.d.}}$ and $S_{\text{test}}^{\text{o.o.d.}}$. The phase diagram in Figure 1 is constructed by merging the last shot accuracy version of these four diagrams shown in Figure 21(a) of Appendix G.

The ability of the model to generalize in-distribution increases with α, as can be seen by comparing the first two panels of Figure 3(a). This behavior is in correspondence with the original work on grokking, where the transition to generalizing solution is driven by the amount of data. Further, we observe that an increase in $n_{\text{i.d.}}$ enhances the in-context sample efficiency, *i.e.* the model generalizes at inference time with fewer few-shot examples. This indicates the onset of the transition from a task-memorizing solution to the one that generalizes out-of-distribution. The model switches to a new algorithmic way of solving the task and the solution is more few-shot-sample-efficient.

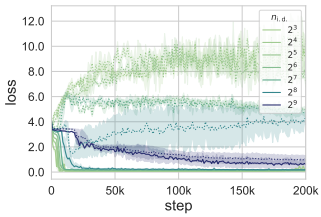
Shifting our focus to the last two panels of Figure 3(a), we see that when $n_{\text{i.d.}} \geq 256$, the model can solve new tasks that were absent in the training set. Notably, there appears to be a trade-off between memorization and generalization when the model attains this o.o.d. generalization ability. As the o.o.d. performance increases, the pre-training performance simultaneously degrades. This phenomenon indicates a shift in the algorithm implemented by the model. Prior to this transition, the model primarily needed to select possible vectors from the list of memorized tasks and apply them. However, post-transition, the model adopts a more universal approach to solve the task in-context. We emphasize, that the model learns to perform ICL in *both scenarios*. The difference lies in the approach to generalization. When the model can only generalize in-distribution it's task is to classify the sequence as one of the seen tasks or as unknown. Once it matches the sequence to one of the memorized task vectors, it does well for x, y pairs that only appear in the test set. However, as the number of tasks vectors grows the model fails to store them all and is forced to find a method of determining the task vector algorithmically at inference time. In that case model performs equally well on seen and un-seen tasks alike. In fact, the small two-layer model we study has such a low capacity that it entirely skips the in-distribution generalization phase and immediately jumps from pure memorization to out-of-distribution generalization.

Next, to further illustrate the effect of task diversity, we plot the pre-training accuracy (set $S_{\text{train}}^{\text{i.d.}}$) and the o.o.d. test accuracy (set $S_{\text{test}}^{\text{o.o.d.}}$) as a function of training steps (Figure 3(b, c)); for various values of $n_{\text{i.d.}}$. We observe a clear memorization-to-generalization transition as task diversity increases. Interestingly, for $n_{\text{i.d.}} = 2^8$, the ICL ability on $S_{\text{test}}^{\text{o.o.d.}}$ set exhibits non-monotonic behavior, where the o.o.d. performance rises and falls along the training process. This phenomenon is likely due to a competition between the memorizing and generalizing circuits inside the model. Note that this phenomenon is akin to the one analyzed in Singh et al. (2023), albeit with a different setup.

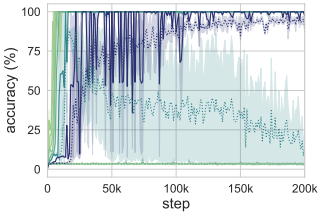
Further evidence supporting the two-circuit competition can be observed in panel (d) of Figure 3. The loss curves show a "monotonic $\rightarrow$ non-monotonic $\rightarrow$ monotonic" transition as the task diversity increases. With a minimal number of pre-training tasks, the model primarily engages in memorization, resulting in a monotonically increasing o.o.d. loss curve. As the number of pre-training tasks increases, the loss curve



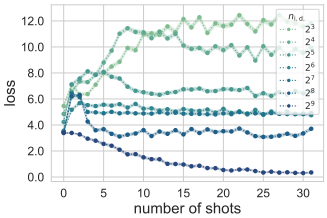
(a) Accuracy phase diagrams of all four sets



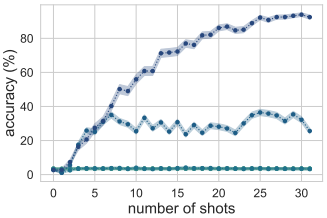
(b) Loss - step



(c) Accuracy - step



(d) Loss - # of shots



(e) Accuracy - # of shots

Figure 3: Phase diagram for the depth $d = 6$ models. **(a)** Accuracy (averaged over all shots) on all four sets used to plot the Figure 1 phase diagram, with an early stopping applied. Notably, in the regions when models generalize to o.o.d. sets, the pre-training performance degrades; **(b, c)** $\alpha = 0.6$ training accuracy and o.o.d. test accuracy (dotted line). For $n_{\text{i.d.}} = 2^8$, we notice that the o.o.d. generalization ability of the model first improves then degrades as we train longer; **(d, e)** $\alpha = 0.6$, loss and accuracy vs context length, measured on $S_{\text{test}}^{\text{o.o.d.}}$ at the end of training, where for $n_{\text{i.d.}} = 2^8$ case the ICL ability fades away.

exhibits non-monotonic behavior, indicating competition between two distinct neural circuits. This transient nature of o.o.d. generalization for $n_{\text{i.d.}} = 2^8$ is a peculiar case where memorization circuits are initially suppressed but eventually prevail. With substantial task diversity, the circuits responsible for generalization take over, culminating in a monotonic loss curve. Similar insights can be derived from examining the monotonicity of the accuracy curves in panel (e).

4.2. Effect of Model Size and Task Difficulty

A natural question to ask is if similar phenomena can be observed with different model sizes or task difficulties. Here we present our results with $d = 4$ and $d = 2$ in Figure 4 and leave the results for other prime p values in Appendix H.

When comparing phase diagrams in Figure 3 with Figure 4, we observe that those phase diagrams across different depths are qualitatively similar, where the o.o.d. generalization only emerges with a large enough number of pre-training tasks. As model capacity decreases, performance on both the pre-training set and the o.o.d. test set degrades. This is particularly evident in the $d = 2$ case, where the pre-training accuracy falls drastically as the model gains o.o.d. generalization ability.

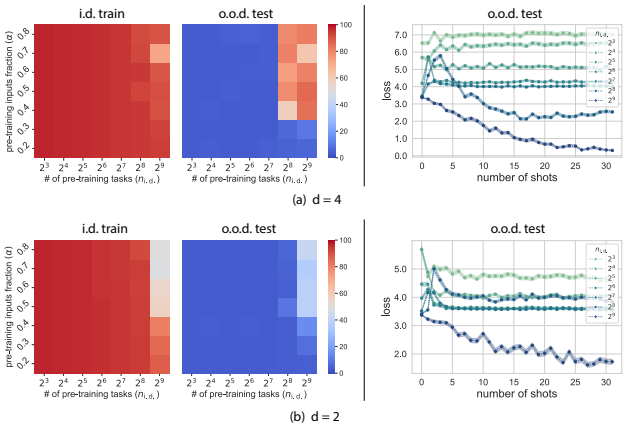


Figure 4: Phases of depth $d = 4$ and $d = 2$ models. With decreasing model capacity, the performance on both sets degrades. At the same time, the transient nature of ICL does not appear. **(a, b) from left to right:** accuracy phase diagrams on pre-training set $S_{\text{train}}^{\text{i.d.}}$ and on o.o.d. test set $S_{\text{test}}^{\text{o.o.d.}}$, with early stopping; loss and accuracy vs context length on o.o.d. test set $S_{\text{test}}^{\text{o.o.d.}}$ for $\alpha = 0.6$.

Interesting observations can be made by comparing loss and accuracy on the o.o.d. test set as a function of context length at the end of training. First, it is evident that as the model depth decreases, the 1-shot loss surge attributable to memorization becomes milder. Notably, for models with $n_{\text{i.d.}} = 2^9$, there is no loss surge in the 1-shot case across all three depths. Furthermore, the $d = 4$ model with $n_{\text{i.d.}} = 2^8$ behaves significantly differently from the corresponding one with $d = 2$ case, where the model fails to perform ICL for the o.o.d. test set. This is also distinct from the $d = 6$ case, where the model tends heavily toward memorization due to its excessive capacity. Instead, the $d = 4$ model manages to maintain a better balance between memorization and generalization at the end of pre-training. Consequently, the model has a 1-shot loss surge followed by a notable drop in ICL loss. This suggests that $d = 4$ optimally leverages the available model capacity to facilitate effective learning dynamics for o.o.d. generalization.

5. Interpretability

An important question to consider is what algorithm the model implements to achieve o.o.d. generalization. Here we propose such an algorithm and provide empirical evidence supporting our claims. For simplicity, consider the case with two-shot inference:

$$\begin{pmatrix} x_1 & y_1 & z_1^t & x_2 & y_2 & z_2^t & x & y & ? \end{pmatrix}$$

The model learns to (i) re-scale the in-context inputs and (ii) perform linear transformations (over $\mathrm{GF}(p)$). For example, in the above case, the model needs to find the two constants c_1 and c_2 such that

$$c_1(x_1,\ y_1) + c_2(x_2,\ y_2) = (x,\ y) \mod p. \tag{2}$$

Once the model has figured out the constants, the result can be simply computed with modular addition: $c_1 z_1^t + c_2 z_2^t \bmod p$ (Gromov, 2023; Nanda et al., 2023; Liu et al., 2022; Zhong et al., 2023). Notably, with sequences that provide more demonstrations, the model has the flexibility to combine in-context examples to derive the answer[1]. However, no matter which specific implementation the model chooses, solving linear systems in $\mathrm{GF}(p)$ to figure out those constants is a must. This requires the model to be able to combine in-context examples and perform modular arithmetic over them, which breaks down to the following three essential skills other than copying information across in-context examples:

I. *Modular Map*: Encode the tokens such that operations over $\mathrm{GF}(p)$ can be easily implemented

II. *Multiplication*: A necessary skill to rescale in-context examples like in Equation (2)

III. *Addition*: A necessary skill for combining two examples in Equation (2)

In Section 5.1, we demonstrate that while models of both depths, ($d = 2, 4$) learn skills I, II perfectly; the $d = 4$ model outperforms its $d = 2$ in skill III – namely *combining* the in-context examples. We attribute this disparity to the limited capacity of $d = 2$ models. Then, in Section 5.2, we show that there are special attention heads that implement the skills I and II. We explicitly show that the structure of these heads deteriorate as pre-training task diversity $n_{\text{i.d.}}$ decreases. We leave the discussion of embedding layers (shown in Figure 1(e)) to Appendix E.

5.1. Model Learns to Combine

We find telling signatures of Equation (2) upon analyzing model predictions with varying numbers of examples in context. Consider the 1-shot scenario, wherein only one demonstration is given to the model. This is equivalent to setting $c_2 = 0$ in Equation (2). In this case, one would expect that the model will only solve the task correctly for examples that obey $(x, y) = c_1(x_1, y_1)$ for some c_1. Indeed, this is exactly what we observe in Figure 5 – the model correctly predicts all the targets when the inputs (x, y) are re-scalings of the in-context input (x_1, y_1).

For few-shot scenarios, the model does two things. It can predict the targets correctly whenever the query input is proportional to any of the inputs of the previous demonstrations (Figure 5 row 1). Additionally and more importantly, it can correctly solve *substantially more* than just these cases (Figure 5 row 1,3). This results from the model's ability to *combine* in-context examples and use them for prediction. Evidence for this can be seen in the proliferation of red points in row 3 of Figure 5 as more examples are given in-context. The $d = 2$ model performs inferior to its $d = 4$ counterpart in this latter skill, hinting at an imperfect implementation of the algorithm and the partially correct choice of the set of constants $\{c_1, c_2, \cdots\}$.

5.2. Attention Heads Implement Essential Skills

In the previous section, we found "black-box" evidence suggesting that the model is implementing an algorithm akin to solving equation Equation (2). Now, we turn to "white-box" interpretability, to identify the components within the transformer model that are responsible for the essential skills outlined at the beginning of Section 5. In Figure 6(a, b) we analyze the important attention heads in a $d = 2$ model that generalizes o.o.d. We compare them with the heads from other $d = 2$ models that do *not* generalize o.o.d in Figure 6(c, d).

[1] In an autoregressive setting, the actual position that the model needs to output the prediction is shifted by one token, but this does not change our argument.

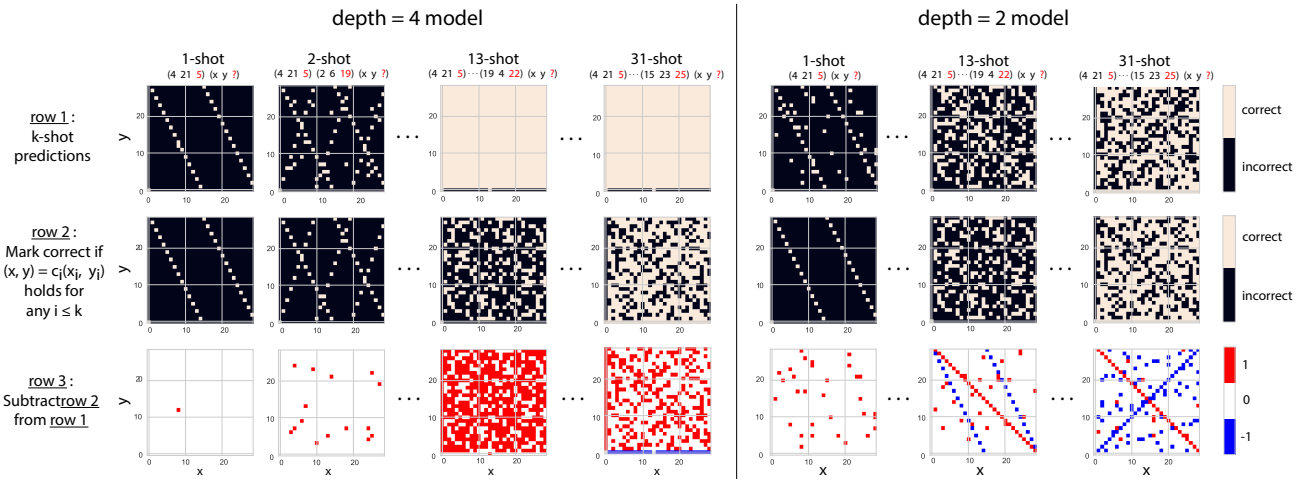


Figure 5: $d = 4$ and $d = 2$ models' performance on k-shot inference, on the grid of inputs $(x, y) \in \mathbb{Z}_p^2$ (task vector = $(6, 6)$). **row 1:** Models' predictions on o.o.d. task of the type $(x_1 \;\; y_1 \;\; z_1 \;\; \cdots \;\; x_k \;\; y_k \;\; z_k \;\; x \;\; y \;\; ?)$. **row 2:** Analytical plots showing all the possible examples that are re-scalings of previous examples. In other words, we mark the points white whenever $(x, y) = c_i(x_i, y_i)$ for some $i \leq k$. **row 3:** Subtract row 2 from row 1, by using correct=1 and incorrect=0. The red points correspond to the examples that the model manages to get right, but are *not* re-scalings of any of the previous examples. Whereas, blue points are the examples the model misses, despite being a re-scaling of a previous example. This row tells us about the model's ability to *combine* the re-scaled in-context examples. Note that $d = 4$ model readily learns to combine previous examples, while its $d = 2$ counterpart struggles due to its limited capacity.

In Figure 6(a), we show the attention head from layer 1 that implements skill I. In the top panel, we see that each query only pays attention to itself and the two preceding keys. This pattern likely stems from the fact that each example in the sequence contains three tokens x_i, y_i, z_i^t; and suggests that this head is mostly focused on the information within the example.

In the bottom panel, we perform principal component analysis (PCA) on the outputs of this head. Specifically, we feed a batched k-shot sequence of the form $(x_1 \;\; y_1 \;\; z_1 \;\; \cdots \;\; x_k \;\; y_k \;\; z_k \;\; x \;\; y \;\; z)$, where the first k inputs are fixed and the last input (x, y) is scanned over all p^2 possible pairs. We concatenate the resulting features from x and y, resulting in $p^2 \times 2d_{\text{embed}}$ batch of features – and perform PCA on this matrix. We project all these $2d_{embed}$ dimensional features onto the first two principal components. Annotating the pairs (x, y) with $(\log_{27} x, \log_{27} y)$[2], we find a "Clock-of-clocks" – where clocks of period 28 are themselves arranged in a bigger clock of period 28. Number 0 is located at the center of the clocks[3]. We observe similar clock-of-clocks for concatenated features from (x, z) and (y, z) as well. We refer the reader to Appendix E for further details.

[2]We use 27 as the base of logarithm, which is a primitive root of $\mathrm{GF}(29)$

[3]This is expected since $\log_{27} 0$ is mathematically ill-defined and needs special treatment.

In Figure 6(b), we analyze the head from layer 2 that effectively implements Skill II – it does so by building upon skill 1 and leveraging the features discussed above. The upper panel shows that the highly structured attention map that focuses on the current example as well as the examples preceding the current one. This pattern aligns with the step in our proposed algorithm where the model needs to compare (x, y) pairs across different examples in-context; and selectively re-scale some of them for later combination. Additionally, as the number of in-context demonstrations increases, the attention map indicates that the model begins to focus on multiple examples simultaneously. This suggests that the model might be employing a potential weighted average approach over these examples, as discussed earlier.

By conducting a PCA analysis similar to that in Figure 6(a), we also identify clocks when annotating examples in the $(\log_{27} x, \log_{27} y)$ format. Unlike the previously discussed pattern, the specifics of this "clock-of-clocks" arrangement vary depending on the position and the choice of task vector (a^t, b^t). This variability suggests that the head in question is utilizing information about the specific task from the context to appropriately rescale the equations.

We further highlight the importance of the signatures we find in the above paragraphs via comparison with a model that does *not* generalize o.o.d, in Figure 6(c, d). Note that as the number of pre-training tasks $n_{\text{i.d.}}$ decreases, the attention map similar to the one in panel (a) starts to be mosaicked;

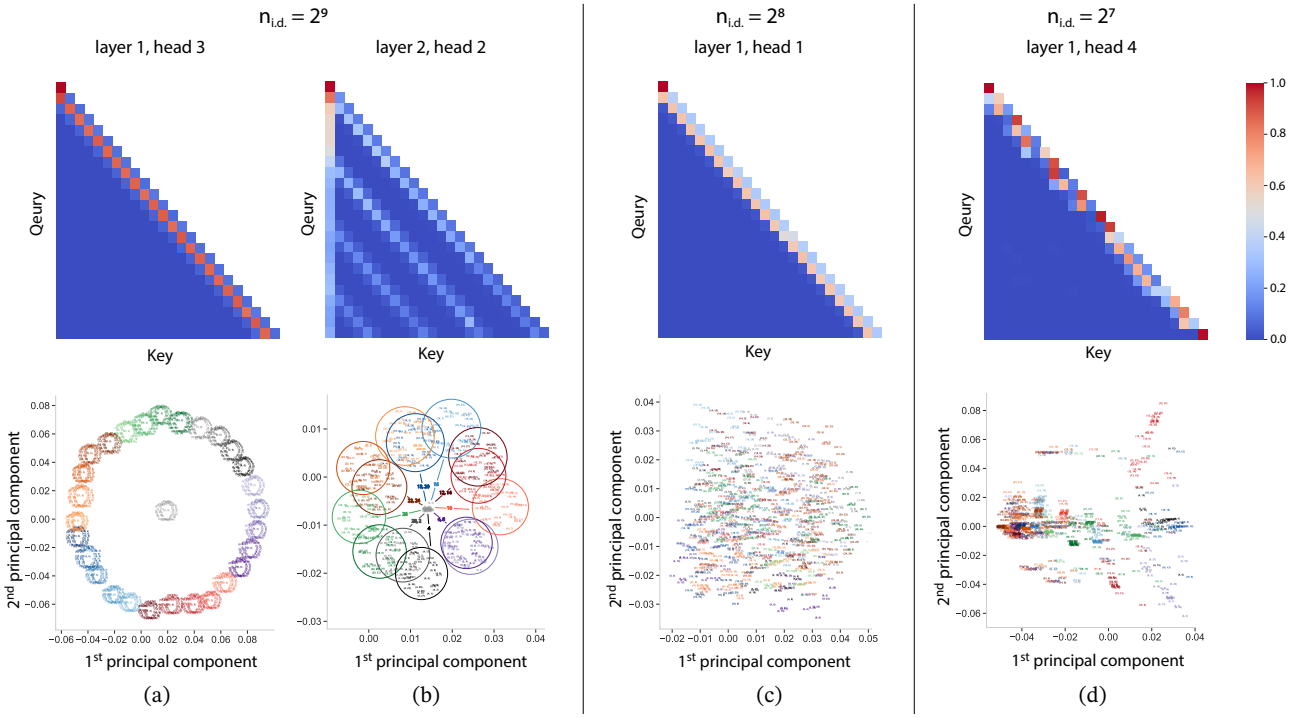


Figure 6: Models that generalize o.o.d. (left) exhibit more structured attention maps. Additionally, the top-2 principal components of the features from the corresponding heads also show more structured patterns. The features are computed for sequences with an o.o.d. task vector $(a^t, b^t) = (6, 6)$, loop over (x_i, y_i) at a specific shot while the previous parts of the sequence are fixed. We annotate each PCA plot with the $(\log_{27} x_i, \log_{27} y_i)$ mod p pairs. **(a)** The principal components form a circle of circles where the position of the outer circle is controlled by x_i. This pattern remains the same for different task vectors or the shot choices; **(b)** Only plotted pairs with even $\log_{27} x_i$, with each $\log_{27} x_i$ circled with different colors. The PCA pattern forms a similar double circle as those in (a), with the key difference that those circles depend on task vector choices and the shot choices; **(c, d)** Models without o.o.d. generalization ability. We pick heads from the first block that corresponds to the first column of (a). We see clearly the structure of attention maps and PCA patterns deteriorate as the task diversity decreases.

and at the same time the PCA projections lose their shape. As a result, those models also lose the ability to perform ICL on modular arithmetic out-of-distribution. In Appendix E.3, we perform pruning experiments on attention heads to further confirm the necessity of those "clock-of-clock" heads for the model to perform ICL on modular arithmetic.

From Figure 5, we infer that the $d = 2$ model has only partially acquired skill III, due to its limited capacity. On the other hand, $d = 4$ model is very good at combining equations via skill III, explaining its superior performance.

5.3. MLP and LayerNorm

The experiments in the previous subsection suggest that skill III may be implemented within the MLP layers. We attempted to search for signals supporting this assertion in similar vein as (Nanda et al., 2023; Gromov, 2023) but could not find conclusive evidence. However, we did find highly structured neuronal activations in certain MLP layers. In Figure 7, we show the MLP activations from layer 3 MLP of the $d = 4$ model on inputs of the type $(x_1 \;\; y_1 \;\; z_1 \;\; \cdots \;\; x_{31} \;\; y_{31} \;\; z_{31} \;\; x \;\; y)$. We find that each hidden neuron fires for specific, structured values of inputs (x, y). Consequently, we surmise that analysis of the MLP layers is both important and fruitful. Similarly, we could not detect any obvious signals in LayerNorm. We leave the careful study of MLP and LayerNorm for future work.

6. Discussion

We have investigated the emergence of ICL learning and skill composition in AMs on a novel algorithmic dataset. The dataset includes a large discrete set of modular arithmetic tasks and was specifically designed to force models to learn how to solve a variety of tasks. It consists of learning linear modular functions, where the model is expected to identify *and* perform a modular operation in-context. When the number of tasks is relatively small, the models simply

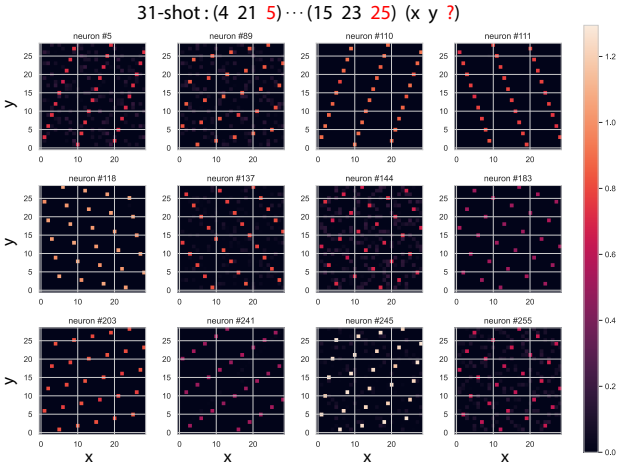


Figure 7: $d = 4$ model MLP activations from layer 3, on the grid of inputs $(x, y) \in \mathbb{Z}_p^2$ (task vector $= (6, 6)$). We extract the activations for the last token, with an input of the type $(x_1 \;\; y_1 \;\; z_1 \;\; \cdots \;\; x_{31} \;\; y_{31} \;\; z_{31} \;\; x \;\; y)$. Each plot corresponds to the output of a single neuron (post-ReLU) in the MLP layer, as a function of the inputs (x, y).

memorize the task vectors and classify the input vectors by the task vectors they have memorized. In this case, the models still have to learn how to perform the memorized tasks because they do generalize in-distribution and develop ICL capabilities. Once the number of training tasks becomes too large the models transition to a qualitatively different, algorithmic approach, where the task vector is determined at inference time.

Finally, we have investigated the learnt representations and showed that qualitatively different circuits are formed in different phases.

Limitations We have limited ourselves to algorithmic datasets. Without further investigation, it is difficult to say what lessons can be transported to the realistic language models, and what lessons are specific to the current setting. Mechanistic interpretability analysis of deeper models proved to be much more difficult than that of a smaller models. Consequently, we still do not understand the role of every part of the network in the deeper cases.

Acknowledgments

T.H. thanks Yue Xu and Dayal Singh Kalra for helpful discussions. A.G.'s work at the University of Maryland was supported in part by NSF CAREER Award DMR-2045181, Sloan Foundation and the Laboratory for Physical Sciences through the Condensed Matter Theory Center. The authors acknowledge the University of Maryland supercomputing resources (http://hpcc.umd.edu) made available for conducting the research reported in this paper.

Impact Statement

Our work involves small models and toy datasets. It does have societal consequences at large, beyond improving our understanding of certain aspects of deep learning.

References

Ahn, K., Cheng, X., Daneshmand, H., and Sra, S. Transformers learn to implement preconditioned gradient descent for in-context learning. In Oh, A., Neumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 45614–45650. Curran Associates, Inc., 2023. URL https://openreview.net/forum?id=LziniAXEI9.

Akyürek, E., Schuurmans, D., Andreas, J., Ma, T., and Zhou, D. What learning algorithm is in-context learning? investigations with linearmodels, 2023. URL https://openreview.net/forum?id=0g0X4H8yN4I.

Arora, S. and Goyal, A. A theory for emergence of complex skills in language models. *arXiv preprint arXiv:2307.15936*, 2023.

Boix-Adserà, E., Saremi, O., Abbe, E., Bengio, S., Littwin, E., and Susskind, J. M. When can transformers reason with abstract symbols? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=STUGfUz8ob.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Doshi, D., Das, A., He, T., and Gromov, A. To grok or not to grok: Disentangling generalization and memorization on corrupted algorithmic datasets. *arXiv preprint arXiv:2310.13061*, 2023.

Doshi, D., He, T., Das, A., and Gromov, A. On learning modular polynomials. In *ICLR 2024 Workshop on Bridging the Gap Between Practice and Theory in Deep Learning*, 2024. URL https://openreview.net/forum?id=QO0y9ysrgu.

Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., DasSarma, N., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

Ganguli, D., Hernandez, D., Lovitt, L., Askell, A., Bai, Y., Chen, A., Conerly, T., Dassarma, N., Drain, D., Elhage, N., et al. Predictability and surprise in large generative models. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, pp. 1747–1764, 2022.

Garg, S., Tsipras, D., Liang, P., and Valiant, G. What can transformers learn in-context? a case study of simple function classes. 2023.

Gromov, A. Grokking modular arithmetic, 2023. URL https://arxiv.org/abs/2301.02679.

Gu, J., Li, C., Liang, Y., Shi, Z., Song, Z., and Zhou, T. Fourier circuits in neural networks: Unlocking the potential of large language models in mathematical reasoning and modular arithmetic, 2024.

Guo, T., Hu, W., Mei, S., Wang, H., Xiong, C., Savarese, S., and Bai, Y. How do transformers learn in-context beyond simple functions? a case study on learning with representations. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ikwEDva1JZ.

Hendel, R., Geva, M., and Globerson, A. In-context learning creates task vectors, 2023.

Kirsch, L., Harrison, J., Sohl-Dickstein, J., and Metz, L. General-purpose in-context learning by meta-learning transformers, 2024.

Lin, Z. and Lee, K. Dual operating modes of in-context learning, 2024.

Liu, S., Ye, H., Xing, L., and Zou, J. In-context vectors: Making in context learning more effective and controllable through latent space steering, 2024.

Liu, Z., Kitouni, O., Nolte, N. S., Michaud, E., Tegmark, M., and Williams, M. Towards understanding grokking: An effective theory of representation learning. *Advances in Neural Information Processing Systems*, 35:34651–34663, 2022.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Nanda, N., Chan, L., Lieberum, T., Smith, J., and Steinhardt, J. Progress measures for grokking via mechanistic interpretability. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=9XFSbDPmdW.

Nichani, E., Damian, A., and Lee, J. D. How transformers learn causal structure with gradient descent, 2024.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Johnston, S., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. In-context learning and induction heads, 2022.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library, 2019.

Power, A., Burda, Y., Edwards, H., Babuschkin, I., and Misra, V. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

Raventos, A., Paul, M., Chen, F., and Ganguli, S. Pretraining task diversity and the emergence of non-bayesian in-context learning for regression. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=BtAz4a5xDg.

Singh, A. K., Chan, S. C., Moskovitz, T., Grant, E., Saxe, A. M., and Hill, F. The transient nature of emergent in-context learning in transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=Of0GBzow8P.

Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding, 2023.

von Oswald, J., Niklasson, E., Randazzo, E., Sacramento, J., Mordvintsev, A., Zhmoginov, A., and Vladymyrov, M. Transformers learn in-context by gradient descent, 2023.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

Zhang, F. and Nanda, N. Towards best practices of activation patching in language models: Metrics and methods. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Hf17y6u9BC.

Zhong, Z., Liu, Z., Tegmark, M., and Andreas, J. The clock and the pizza: Two stories in mechanistic explanation of neural networks. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S5wmbQc1We.

A. Experimental Details

A.1. Model and Training Hyperparameters

Architecture We used GPT-like architectures with Rotary Positional Embedding ($\theta = 10,000$) and ReLU activations. We fix the number of heads $H = 4$, embedding dimension $d_{\text{embed}} = 512$ and MLP widening factor 4 throughout every model. We use $d = 2, 4, 6$ throughout the paper. Embedding layers and the output layer are tied via weight tying.

Initialization All linear layers and embedding layer weights are sampled from Gaussian distribution $\mathcal{N}(0, 0.02^2)$ at initialization, with the exception that the last linear layer in each MLP is sampled from $\mathcal{N}(0, 0.02^2/2d)$. No bias is used in any layer.

Optimization and Schedule We trained most models using AdamW optimizer with learning rate $\eta = 1.5 \times 10^{-4}$, weight decay $\lambda = 2.0$, $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\epsilon = 10^{-8}$, batch size $B = 1024$, $n_{\text{ct}} = 32$ in-context examples for 200k steps, together with a 5% linear warmup starting from 0.01η and a cosine annealing towards the end to 0.1η. Weight decay is not applied to LayerNorm layers.

Hyperparameter Choice For $d = 2$ models we scanned learning rates $\eta \in \{7.5 \times 10^{-5}, 1.5 \times 10^{-4}, 3 \times 10^{-4}, 6 \times 10^{-4}\}$ and weight decay values $\lambda \in \{0.5, 1.0, 2.0, 5.0\}$. Then we transfer our hyperparameters to other depths. Benefiting from the extra scaling in the initialization of the last linear in MLP, we find that the hyperparameters perform well for other depths. For larger p values, we lowering down the learning rate to 10^{-4}.

A.2. Further Details of Each Plot in the Main Text

Figure 1 (b) Phase diagram constructed using $d = 6$ data shown in Figure 21, threshold for defining each phase is set to 75% for the corresponding set; (c) accuracy - number of shots curve for $d = 4$, $n_{\text{i.d.}} = 2^8$, $p = 29$ and $\alpha = 0.7$. (d, e) $d = 2$ model with $p = 29$, $n_{\text{i.d.}} = 2^9$ and $\alpha = 0.6$.

Figure 3 (a) Selected the best out of three random seeds with early stopping. (b, c) averaged over three random seeds with standard error labeled. All o.o.d. data are measured every $1,000$ step for 16 randomly sampled sequences along the pre-training. (d, e) Used checkpoint at the end of pre-training, averaged over 128 random sequences sampled from $S_{\text{test}}^{\text{o.o.d.}}$.

Figure 4 (a, b) Both phase diagrams are the best selected from three random seeds with early stopping. The loss - number of shots curves are plotted with $\alpha = 0.6$.

Figure 5 We use a $d = 2$ model trained with $\alpha = 0.6$ and $n_{\text{i.d.}} = 2^9$, and a $d = 4$ model trained with $\alpha = 0.8$ and $n_{\text{i.d.}} = 2^9$ with batch size $B = 1536$.

Figure 6 (a, b) Attention heads extracted form $d = 2$ model trained with $n_{\text{i.d.}} = 2^9$ and $\alpha = 0.6$. (c, d) Both models are trained with $\alpha = 0.6$.

B. Details of resources used

To generate each phase diagram and training curve, we used 90 GPU days on NVIDIA A100 40GB, with automatic mixed precision (BFloat16) and Flash Attention implemented in the PyTorch library(Paszke et al., 2019). During the exploring stage, we also used around another 90 GPU days. Most inferences were running on 1/7-NVIDIA A100 40GB GPU, of which the cost is negligible compared to the pre-training cost.

C. Table of log map for $p = 29$

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$\log_{27}(n)$	0^4	28	15	19	2	22	6	12	17	10	9	11	21	18	27

n	15	16	17	18	19	20	21	22	23	24	25	26	27	28
$\log_{27}(n)$	13	4	7	25	23	24	3	26	20	8	16	5	1	14

D. Task Selection and Sequence Design

Task Selection During the initial exploration phase, we observed that it was challenging for the model to learn multiple modular arithmetic tasks simultaneously. Typically, the loss would stagnate at a plateau indefinitely.

We hypothesize that when the model is trained on multiple modular arithmetic tasks, the strong bias inherent in each individual task may interfere significantly. When tasks are selected randomly, the resultant noise appears to inhibit the learning capabilities of the model, preventing it from acquiring any meaningful patterns or rules from the data.

Ultimately, we adopted a more structured approach to task selection as illustrated in Figure Figure 2, based on the following rationale: If the model is to learn a specific task vector (a^t, b^t), it would presumably be more straightforward if one of the two components – either a^t or b^t – has already learned by the model. This would leave the model with only the other component to decipher, which we assume is a comparatively simpler task. Thus, we decided to employ the rectangular rule for sampling task vectors, which we believe strategically reduces the complexity of the learning process by partially leveraging previously acquired knowledge.

In Figure 8, we show an ablation plot of phase diagrams with a randomly selected task vector collection for pre-training. We still use the special sequence design.

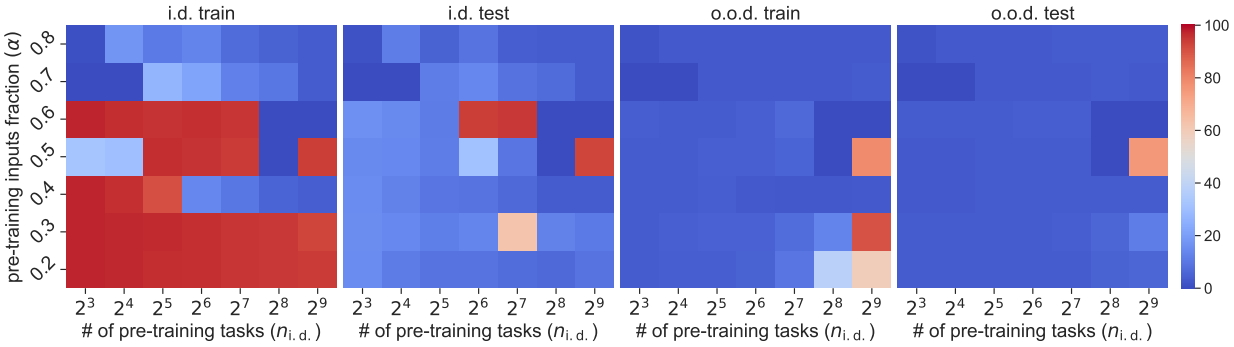


Figure 8: $d = 6$ phase diagram with task vectors randomly selected, selected the best results from two random seeds. Learning rate $\eta = 10^{-4}$.

Sequence Design Following a similar spirit, we use a balanced batch where sequences generated from all task vectors appear exactly the same number of times during the training. We further align the examples across sequences generated by different task vectors, which we believe reduces the chance of the model getting confused by the same input (x, y) appearing at different positions within the batch. Without this design, we could not make the model train.

Here, we also show the training curve per task in Figure 9, trained with all of our tricks. We see that the model first learned very few tasks and then eventually found its way out. For training without the task selection and sequence design, the loss typically plateau around ~ 3.0.

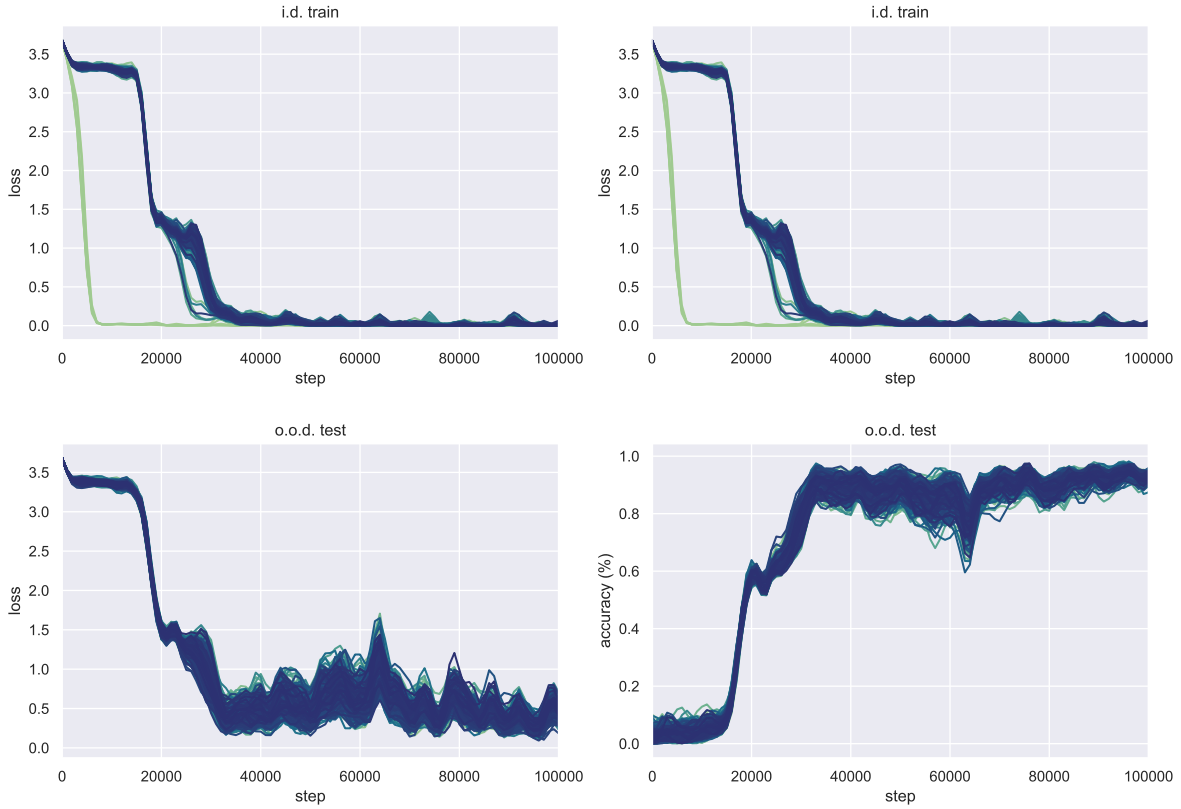


Figure 9: $d = 4$ training curve, with loss and accuracy measured per task for the last in context example. Trained with $n_{\text{i.d.}} = 2^8$ and $\alpha = 0.8$. There are six tasks learned almost instantly: $(5, 26)$, $(21, 23)$, $(28, 27)$, $(3, 9)$, $(10, 10)$, $(5, 9)$.

E. Additional Interpretability Results

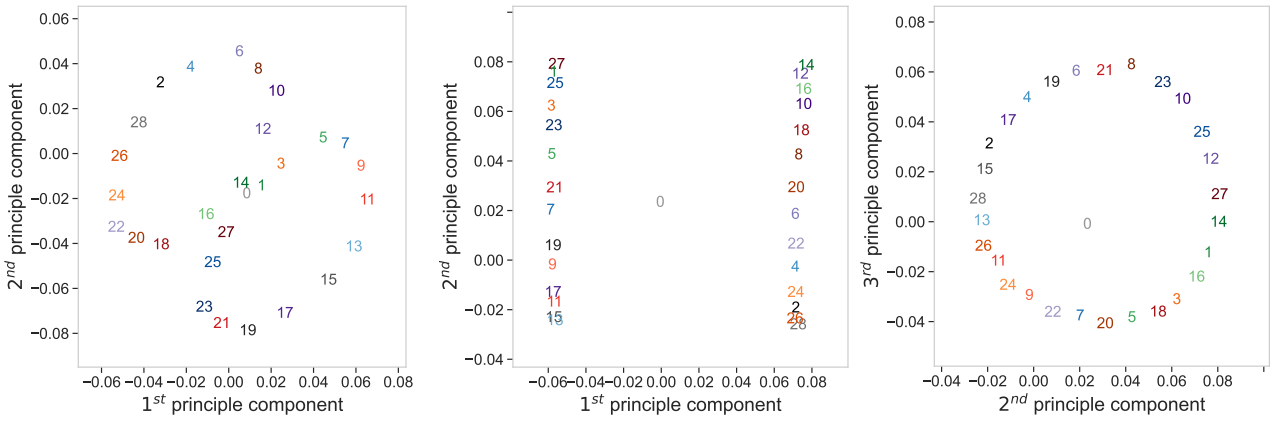
In this section, we show additional results on interpretability. Where we first show evidence in **??** that a $d = 4$ model also implements a version of the scale-and-combine algorithm we proposed in Section 5, and it can do better than the $d = 2$ model. Then, we continue our discussion on how the algorithm might be implemented inside the model. Importantly, This includes the role of embedding (Appendix E.1), the role of other attention heads (Appendix E.2), and finally, the role of MLP and LayerNorm (Section 5.3).

E.1. PCA over Embeddings

We begin the discussion with the role of embedding layers. By further examining different models, we find highly structured embedding from $d = 2$ models. $d = 4$ models, on the other hand, does not have such structured embedding layers.

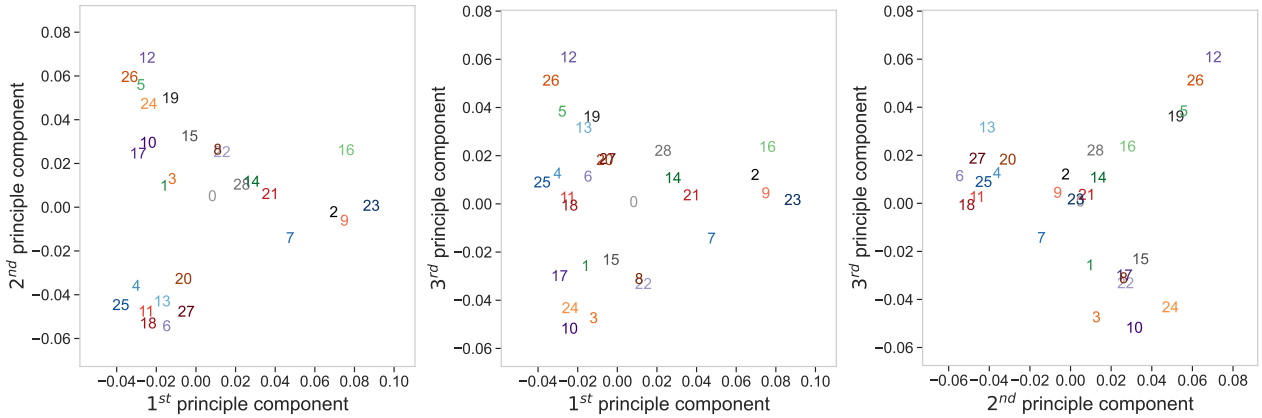
First we focus the embedding of $d = 2$ models. As shown in Figure 10, clearly, the logarithm of each number is split into even and odd groups, and each group forms one clock, which is a suitable embedding for doing modular multiplications. However, one should note that this will not obviate the importance of the head shown in Figure 6(a), as the model still needs a way to distinguish the same number that appears in the different positions in the context.

Curiously, we could not find such a structured embedding for the $d = 4$ model, as shown in Figure 11. However, as we will show in the next subsection, this non-structural embedding, together with the first layer, prepares a foundation for the essential heads in the latter layer to create a similar "clock-of-clock" feature as we shown in Figure 6 (a, b).



(a) seed 2 (b) seed 1

Figure 10: PCA over the embedding layers of $d = 2$ models with different random seeds. Each number is annotated by its $\log_{27}$ value.



(a) PCA over the embedding layer for top-3 principal components

Figure 11: PCA over the embedding layers of $d = 4$. Each number is annotated by its $\log_{27}$ value.

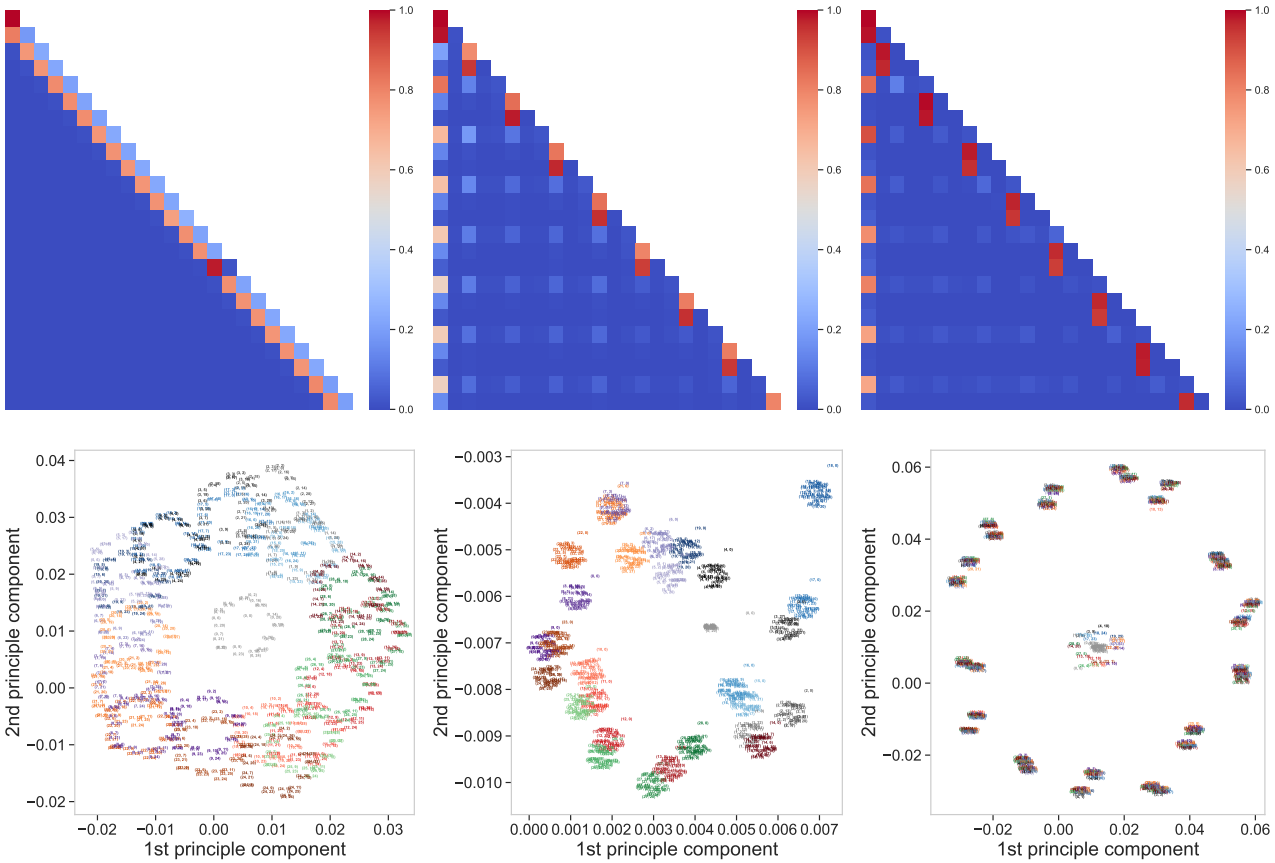
E.2. Attention Heads

E.2.1. $d = 4$ Model

To continue the story, we analyse the attention heads in different models. We first study the $d = 4$ model, where we also find a similar head with "clock-of-clocks" (Figure 6(a)). Moreover, two other heads put together are seemingly equivalent to Figure 6(b). We surmise that any model needs that solves modular arithmetic in context requires such heads.

From Figure 12(a), we see that the head still pays attention locally within three token positions. Importantly, it also creates a clock-of-clock while performing PCA over concatenated (x, y) features. The difference here is that the clock winds twice to go back to its origin. We believe that this factor of two differences in the period means that this head is effectively combing the row of the embedding layer of the $d = 2$ model and the role of the head in Figure 6(a). Overall, we do not yet know why the model needs to split the logarithm of numbers into even and odd groups.

From Figure 12(b, c), we find two heads that are structured but different from Figure 6(b). The PCA pattern forms clocks while annotated within logarithm space, but depends on the choice of the sequences. This hints that those two heads are trying to re-scale the examples by comparing them with the previous inputs. We think that the two can be combined to form a similar pattern as Figure 6(b).



(a) layer 2 head 4 (b) layer 3 head 1 (c) layer 3 head 4

Figure 12: PCA over the features of specific heads for $d = 4$ model. Each number is annotated by its $\log_{27}$ value. We plotted PCA over (a, b) (x, y) and (c) (y, z) concatenated features. Where the circle in (b, c) heavily depends on the sequence and (a) remains unchanged.

E.2.2. $d = 2$ Model

Next we focus on the $d = 2$ model. In Figure 13, we plot similar PCA to the one in Figure 6(a), but with different concatenated features from the same heads: (x, z) and (y, z). We again see clocks-of-clock, solidifying our argument that this head provides proper spaces for later heads to perform modular operations.

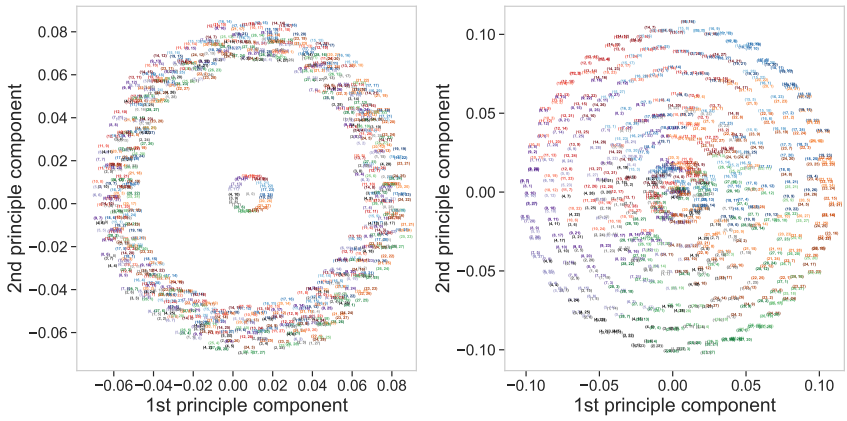
Finally, in Figure 14, we dump all the attention patterns we had for $d = 2$ models. The corresopnding PCA over concatenated (x, y) features are shown in Figure 15 (except for layer 1 head 4, for which we plot for (x, z)). The special head we choose here has a similar behavior to the one in Figure 6(a), with its main focus shifted by one position but still within two preceding tokens. We believe this head implements another variant of Equation (2) – it re-scales equations for the next layer, then helps them figure out how to combine them to predict new examples.

We have reasons to believe that the non-clock heads are not essential to the models' performance. However, we leave a careful exploration of this front for future work.

E.3. Prune Attention Heads

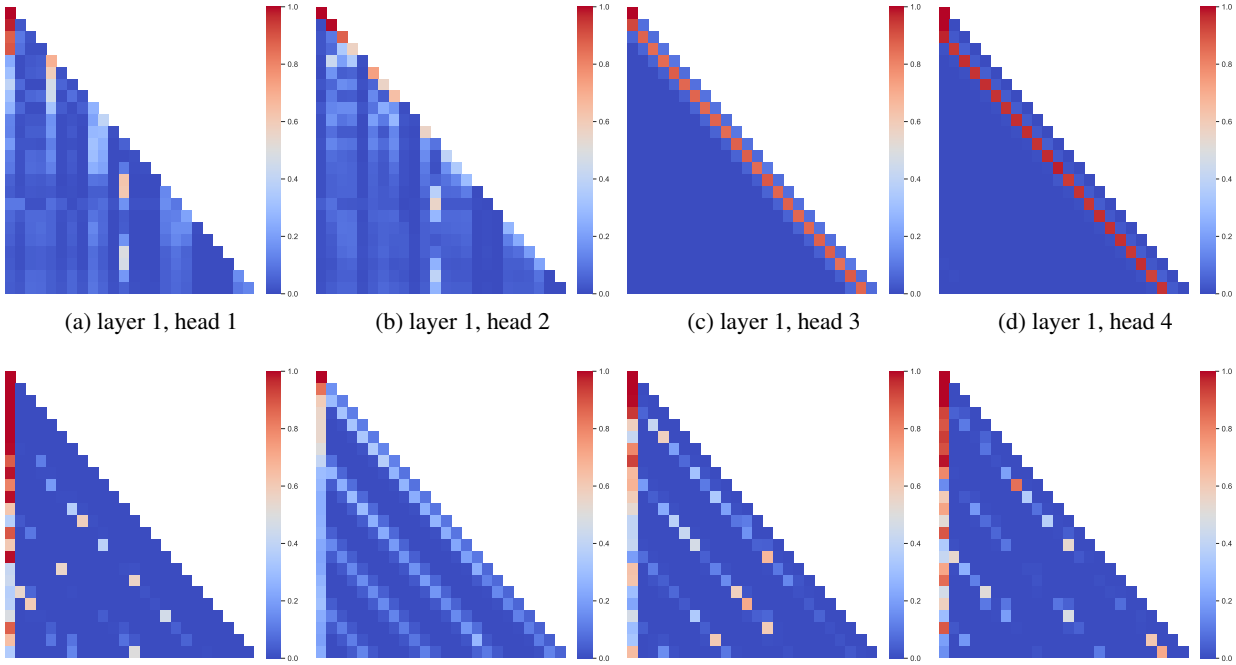
In Figures 6 and 12, we identified the "clock-of-clocks" arrangement through PCA analysis of attention head features. We posited that heads exhibiting these "clock-of-clocks" features are crucial to the model's performance.

To further validate the importance of these highly structured heads, we use the structural pruning method proposed by Zhang & Nanda (2024), where we prune attention heads by replacing their output before concatenation with the corresponding



(a) (x, z) (b) (y, z)

Figure 13: PCA over the same head in Figure 6(a), with feature concatenated differently. We annotate numbers with their corresponding logarithm.



(a) layer 1, head 1 (b) layer 1, head 2 (c) layer 1, head 3 (d) layer 1, head 4

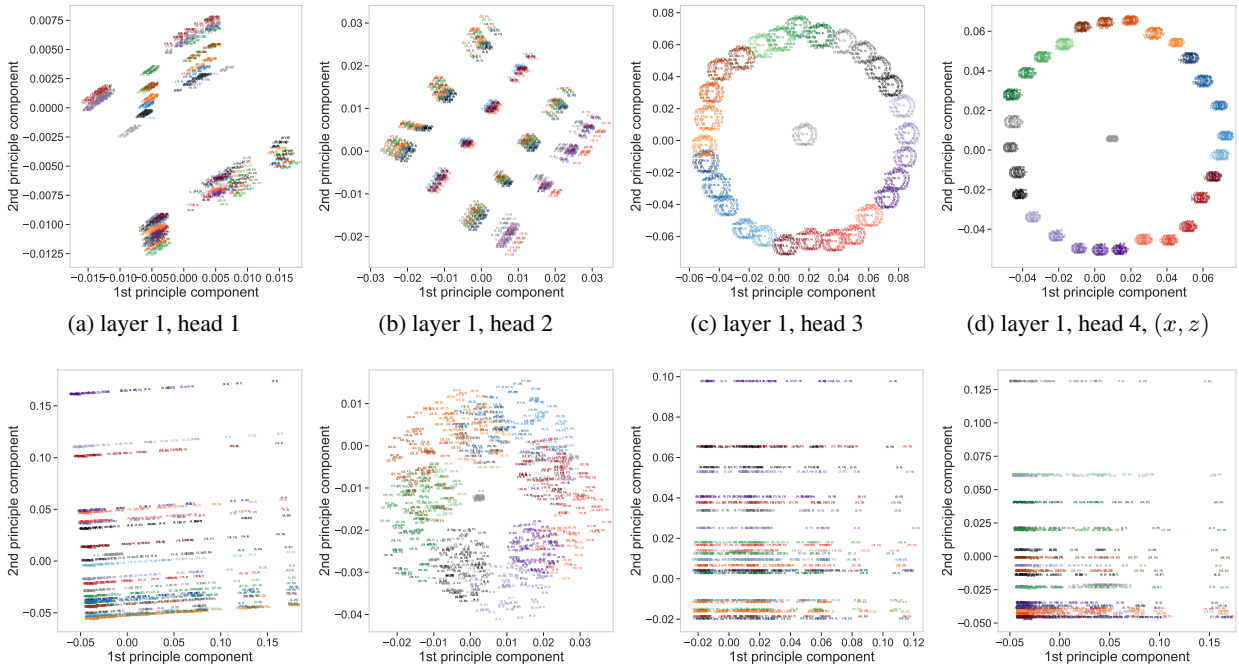
(e) layer 2, head 1 (f) layer 2, head 2 (g) layer 2, head 3 (h) layer 2, head 4

Figure 14: All attention heads in $d = 2$ model. Heads in panel (c) and (f) correspond to those heads in Figure 6 panel (a) and (b).

average outputs calculated using training sequences. Then we use the pruned models to conduct experiments similar to those in Figure 5. We apply this analysis to both the $d = 2$ and $d = 4$ models.

Our initial findings, as depicted in Figure 16, demonstrate that both the $d = 2$ and $d = 4$ models experience complete performance collapse when we prune their most critical attention heads, which is the layer 1 head 3 for case $d = 2$ and the layer 2 head 4 for case $d = 4$.

We further try to remove as many as possible heads from the model. As we show in Figure 17, for the $d = 4$ model, we only keep the three heads listed in Figure 12 from the first three layers while keeping all the heads in the last layer. Further



(a) layer 1, head 1 (b) layer 1, head 2 (c) layer 1, head 3 (d) layer 1, head 4, (x, z)

(e) layer 2, head 1 (f) layer 2, head 2 (g) layer 2, head 3 (h) layer 2, head 4

Figure 15: PCA for concatenated (x, y)/(x, z) features for all heads in $d = 2$ model.

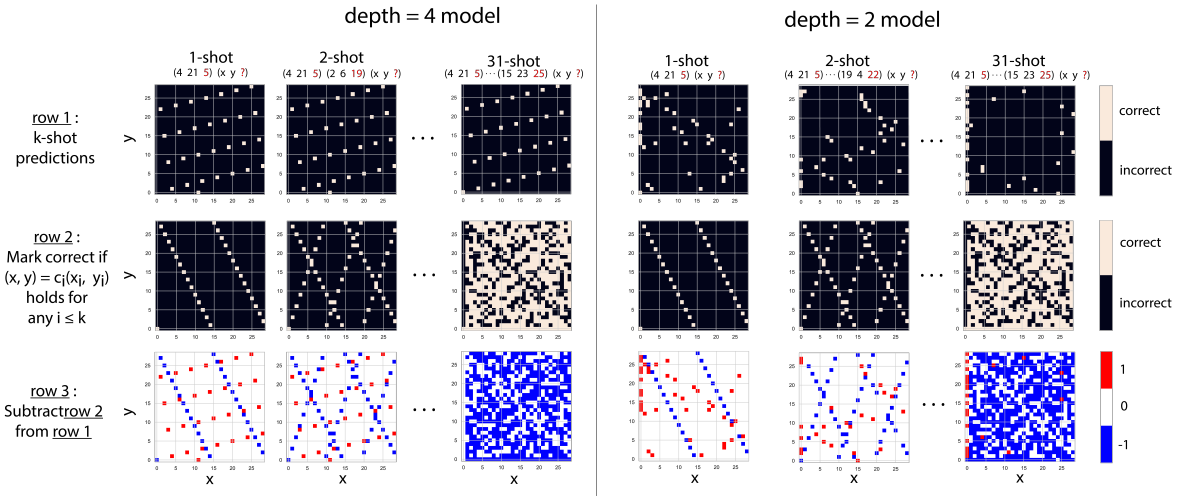


Figure 16: Performance of $d = 2$ and $d = 4$ models after removing the "clock-of-clock" heads.

pruning of any heads leads to significant performance changes. Not surprisingly, the model managed to maintain an almost perfect performance. In contrast, for the $d = 2$ model, we can only remove the heads 1 and 2 from the first layer; any further removal of the attention heads causes the performance to drop drastically.

E.4. Label Noise

To gain more insight into how the model combines the in-context examples, we introduce label-corruption in the in-context examples. In particular, we note the effect of (i) amount and (ii) position of label corruption on the model's performance. When we corrupt a *single* in-context example for $d = 4$ model, the model performance remains unaffected for longer sequences. This hints at that weighted average of the in-context inputs being used in model prediction. The $d = 2$ model,

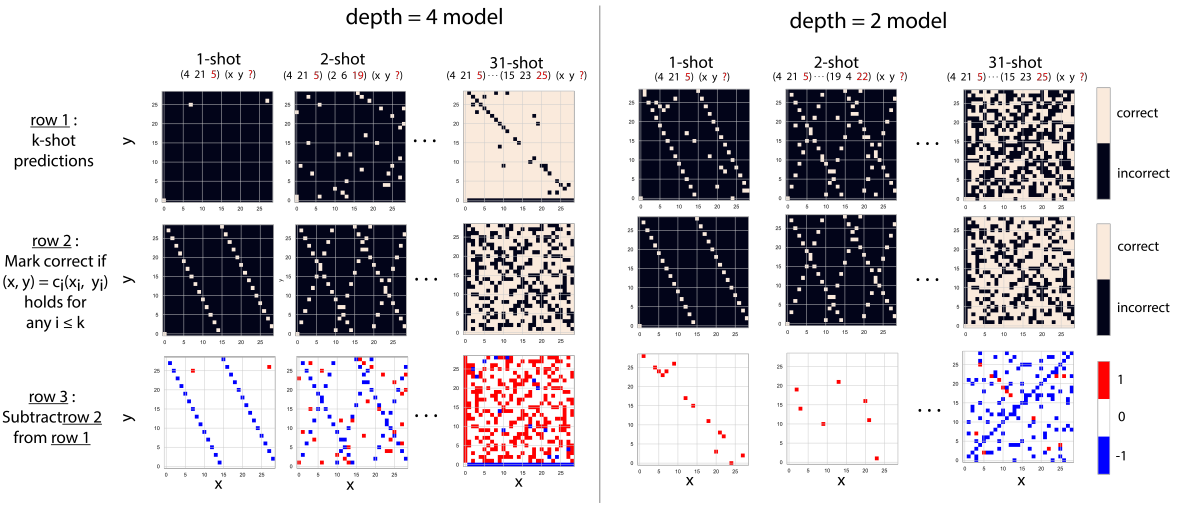
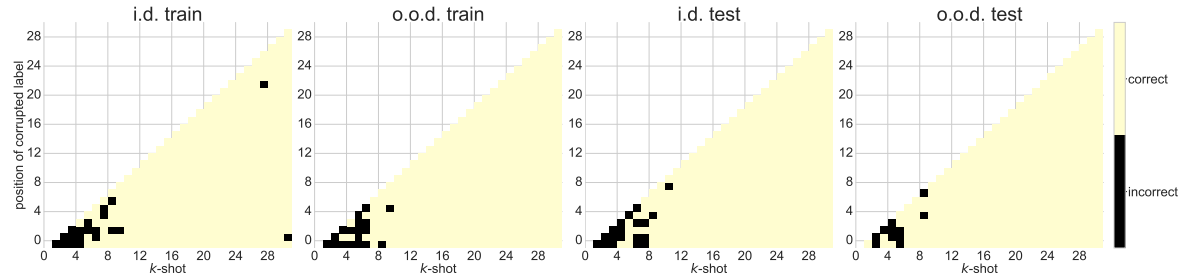


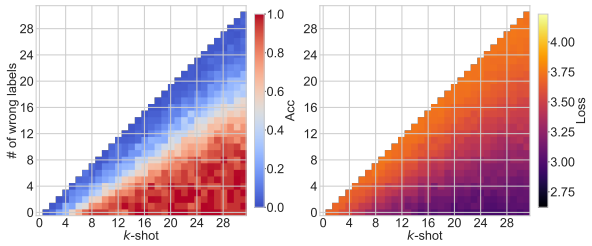
Figure 17: Performance of $d = 2$ and $d = 4$ models after removing most non-"clock-of-clock" heads. For the $d = 4$ model, we leave the last layer heads untouched, while for the $d = 2$ model, we removed heads 1 and 2 from the first layer.

however, did not show such resilience.

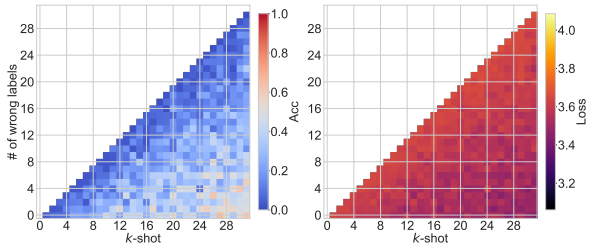
Next, we corrupt *multiple* in-context examples in random locations. We study the effect on model performance as the amount of corrupted labels increases. While the $d = 2$ model is easily overwhelmed, the $d = 4$ model is able to offer strong resistance even at $\sim 40\%$ label corruption, for long sequences. This behavior remains invariant with the change in task vector for the particular sequence, indicating the universality of the underlying algorithm necessary for o.o.d. generalization.



(a) single label corruption, $d = 4$



(b) multiple label corruption, $d = 4$

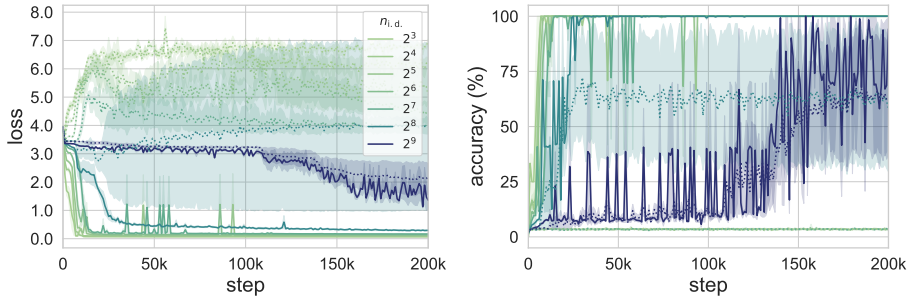


(c) multiple label corruption, $d = 2$

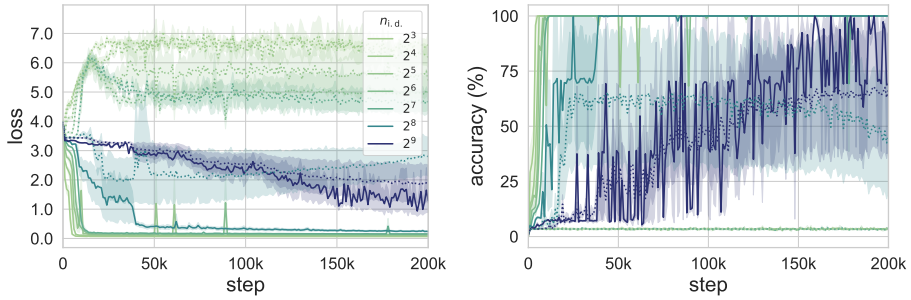
Figure 18: **(a)** Single label corruption: Performance of last token prediction as a function of changing position for a single label corruption. The x-axis shows different shots presented to the model, while the y-axis shows the position j where the label is corrupted $z_j \to z'_j$. The $d = 4$ model is remarkably robust for long sequences, indicating the use of an algorithm which is not sensitive to the position of preceding examples. Qualitatively similar plots hold for other task vectors as well. **(b, c)** Multiple label corruption, at random locations for $d = 4$ and $d = 2$ models respectively. The $d = 4$ model is more resilient to label corruption than the $d = 2$ model, implying that the algorithm the latter employs is imperfect due to limited capacity.

F. Additional Training Curves

We plot some selected training curves for $d = 4$ (Figure 19) and $d = 2$ (Figure 20) from Figure 4 phase diagrams. We see that even for $d = 4$, ICL can be a transient. With increased α or $n_{\text{i.d.}}$, the transient nature goes away.

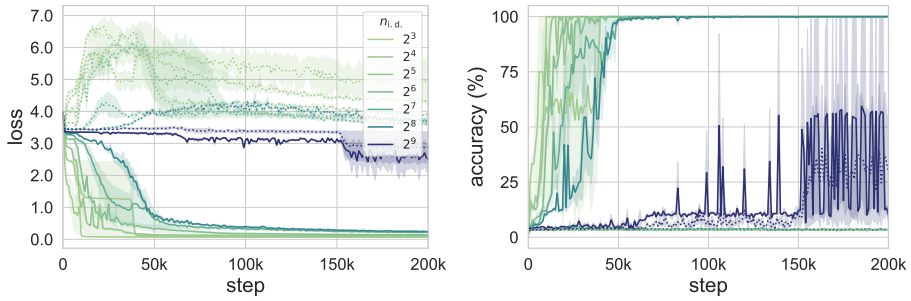


(a) $d = 4$, $\alpha = 0.7$

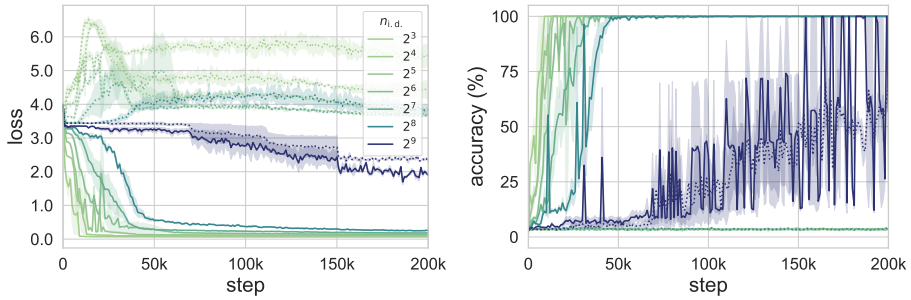


(b) $d = 4$, $\alpha = 0.6$

Figure 19: Training curves for $d = 4$, averaged over three random seeds.



(a) $d = 2$, $\alpha = 0.7$

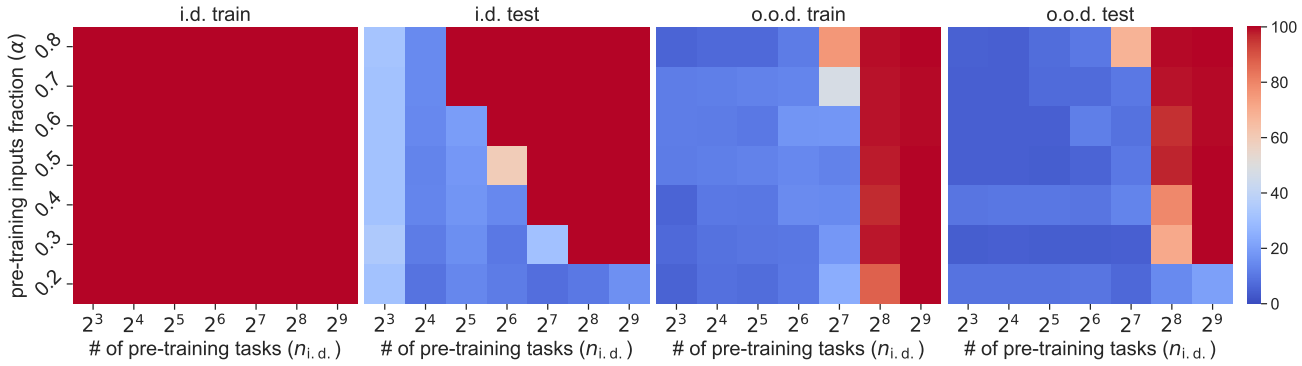


(b) $d = 2$, $\alpha = 0.6$

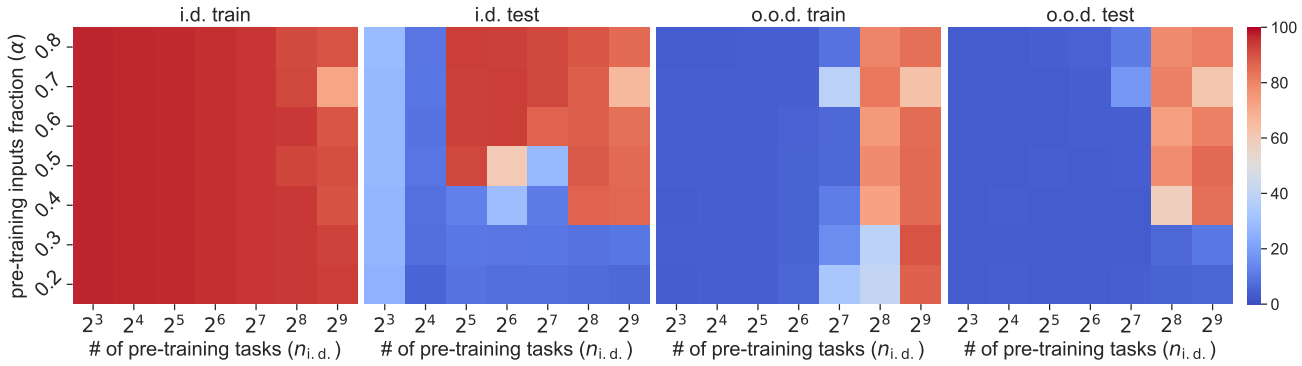
Figure 20: Training curves for $d = 2$, averaged over three random seeds.

G. Additional Phase Diagrams

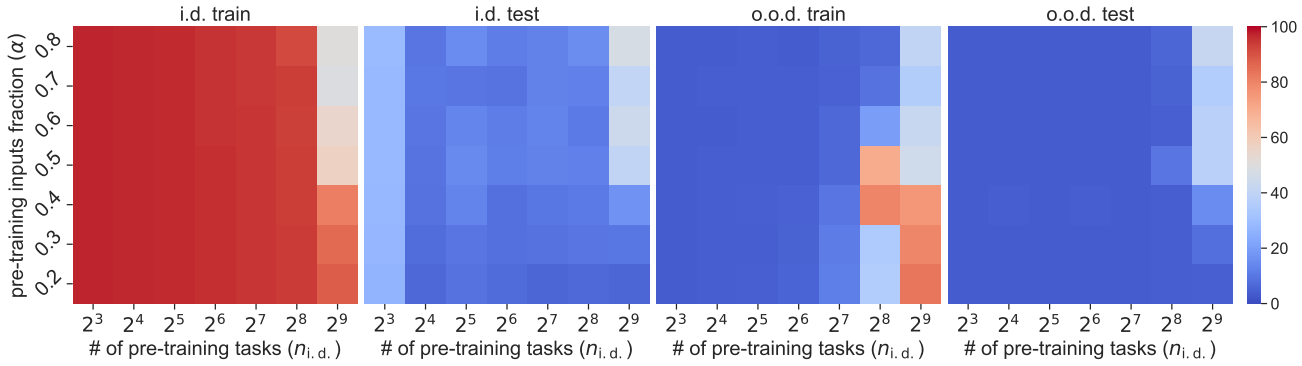
In Figure 21, we plotted detailed/extended versions of the phase diagrams shown in Figure 1/Figure 4. The four phases story we have shown in Figure 1 still hold for other depths.



(a) $d = 6$, last shot



(b) $d = 4$

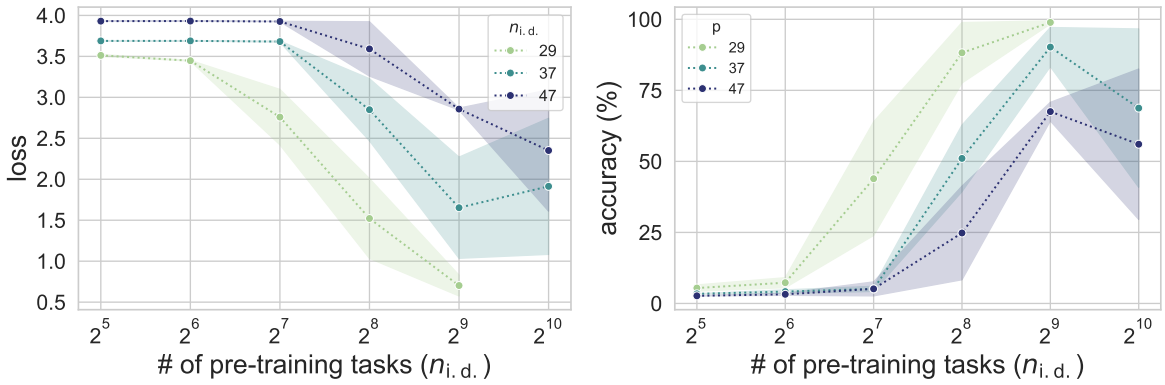


(c) $d = 2$

Figure 21: Phase diagrams on all four sets for $d = 4$ and $d = 2$.

H. Different Choice of p

In this section, we check the effect of varying task difficulties, i.e. the value of p. In Figure 22, we plotted o.o.d. generalization accuracy. Clearly as the task gets harder, the model needs to see more tasks to generalize out-of-distribution.



(a)

Figure 22: We compare the best performance for $d = 6$ models with different $n_{\text{i.d.}}$ values, averaged over three seeds. We use learning rate $\eta = 10^{-4}$ for $p = 37$ and $p = 47$, while keeping other hyperparameters the same as $p = 29$.