ATTENTION AS A HYPERNETWORK

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformers can under some circumstances generalize to novel problem instances whose constituent parts might have been encountered during training but whose compositions have not. What mechanisms underlie this ability for compositional generalization? By reformulating multi-head attention as a hypernetwork, we reveal that a composable, low-dimensional latent code specifies key-query specific operations. We find empirically that this latent code is predictive of the subtasks the network performs on unseen task compositions revealing that latent codes acquired during training are reused to solve unseen problem instances. To further examine the hypothesis that the intrinsic hypernetwork of multi-head attention supports compositional generalization, we ablate whether making the hypernetwork generated linear value network nonlinear strengthens compositionality. We find that this modification improves compositional generalization on abstract reasoning tasks. In particular, we introduce a symbolic version of the Raven Progressive Matrices human intelligence test which gives us precise control over the problem compositions encountered during training and evaluation. We demonstrate on this task how scaling model size and data enables compositional generalization in transformers and gives rise to a functionally structured latent space.

1 INTRODUCTION

Abstract reasoning is a cornerstone of human intelligence. Many intelligence tests in psychology measure intelligence using abstract reasoning tasks (e.g. Wechsler, 1958; Binet & Simon, 1961; Raven, 1962). Arising from the idea that combinatorial structures are central to human cognitive abilities, the ability of connectionist models to reason has been debated for decades (e.g. Rumelhart & McClelland, 1986; Fodor & Pylyshyn, 1988; Smolensky, 1991). However, with the success of neural networks at scale, the capacity for reasoning has seemingly come into reach (Huang & Chang, 2023). Arguably key to this success is the ability for *in-context learning* paired with the capacity to flexibly recombine learned skills and knowledge to unseen settings. While these capabilities emerge at least partially from training large models on huge datasets (Brown et al., 2020), it is not clearly understood how the resulting networks implement them - and why they still often fail.

A number of recent studies that aim to illuminate in-context learning abilities have found that transformers can learn to perform gradient-based optimization within sequence (Dai et al., 2023; Akyürek et al., 2023; von Oswald et al., 2023). They build on the insight that linear attention can be understood as a fast-weight programmer (Schmidhuber, 1992), that learns to construct an implicit model in-context, with weights given by a sum of input-dependent outer products (Schlag et al., 2021). Complementing this perspective, (Hendel et al., 2023; Todd et al., 2024) have found that in-context learning creates task vectors in the hidden activations of deeper layers of the network which summarize in-context information and modulate the forward computation. What is striking is that learning in-context can under some circumstances lead to *compositional generalization*, that is generalization to unseen combinations of previously observed constituents (An et al., 2023; Lake & Baroni, 2023; Hosseini et al., 2022), a capacity that neural networks notoriously struggle with (Srivastava et al., 2023; Press et al., 2023; Dziri et al., 2023). The mechanism behind this ability however is not well understood. Here we aim to shed light on this question.

Our key finding is that through the use of multiple heads, the attention mechanism of transformers is mathematically equivalent to a hypernetwork, a neural network that reconfigures the computation of another neural network as shown in Figure 1. This results in a novel interpretation of the attention scores along the head index as a compact latent code that specifies the operations the attention layer

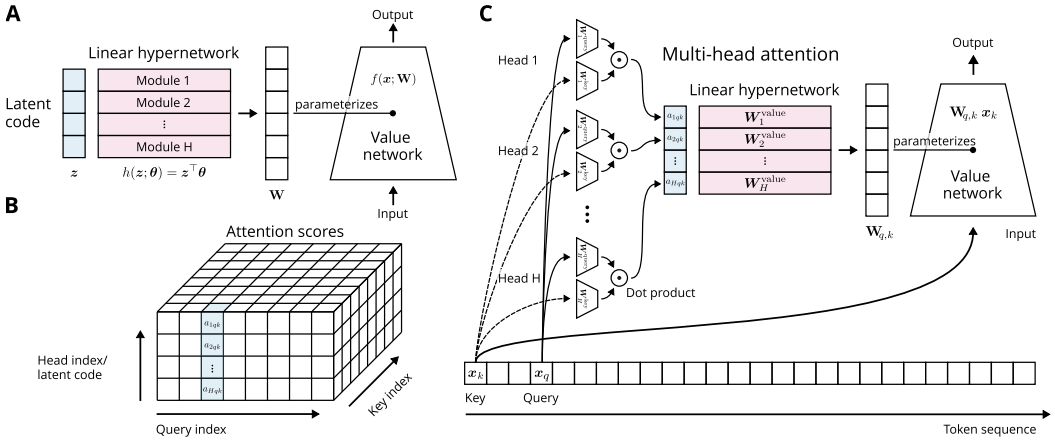

Figure 1: **Hypernetwork attention**. **A** A linear hypernetwork maps a latent code to a set of parameters that configure a value network to process the input. **B** The attention scores along the head index form the latent code of the hypernetwork. **C** Multi-head attention can be equivalently expressed as a linear hypernetwork that configures key-query specific computations of a linear value network.

applies to each input. Notably, we hypothesize that because multi-head attention shares the same hypernetwork per layer, the network is encouraged to reuse and recombine previously acquired operations.

In order to investigate whether this constitutes a faithful characterization of what transformers learn in practice, we study in-context learning tasks that require recomposing knowledge obtained during training. As one such task we develop a challenging abstract reasoning task based on the Raven Progressive Matrices human intelligence test (Raven, 1962) and show how multi-head attention develops a structured latent code that captures information about the operations implemented.

Our **main contributions** can be summarized as follows:

- We reformulate standard multi-head attention from a hypernetwork perspective revealing that network computations are specified by a low dimensional latent code.
- We show that scaling up model size and data enables transformers to compositionally generalize on abstract reasoning tasks and leads to a structured latent space that is predictive of network function.
- To test the hypothesis that the hypernetwork mechanism supports compositionality, we introduce a simple modification to multi-head linear attention that makes the value network nonlinear without introducing additional parameters finding that it improves compositional generalization.
- We introduce SRAVEN, a challenging Raven-based symbolic, abstract reasoning task whose difficulty can be controlled parametrically and which offers precise control over the problem compositions encountered during training in order to test for compositional generalization.

2 Attention as a hypernetwork

In this section, we will first briefly recapitulate hypernetworks (Ha et al., 2017) and standard multi-head dot-product attention (Vaswani et al., 2017) before showing how attention can equivalently be expressed as a hypernetwork. We will then introduce Hypernetwork Linear Attention (HYLA) as a simple modification of linear attention that renders the hypernetwork mechanism inside attention more expressive.

For notation we will use bold lower-case letters for vectors (e.g. $\boldsymbol{z}$), bold upper-case letters for matrices (e.g. $\mathbf{A}$) and bold, italic upper-case letters for learnable parameter matrices (e.g. $\boldsymbol{W}_V$).

2.1 Multi-head attention

A *hypernetwork* is a neural network $h(\boldsymbol{z};\boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$ that takes as input a *latent code* $\boldsymbol{z}$ and outputs parameters $\mathbf{W}$ that parameterize a *value network* $f(\boldsymbol{x};\mathbf{W})$ which then processes the

inputs (see Figure 1A). The latent code, $\boldsymbol{z}$, is typically low-dimensional and can be interpreted as a specification of the computation performed by the value network. As we will see in the following, multi-head attention can be viewed as a linear hypernetwork producing the weights for a linear value network. Note that this is different from the observation that linear attention can be understood as a fast weight programmer (Schlag et al., 2021) where the fast weights of a value network are constructed as a sum of outer products over the *key indices* instead of the *head indices*.

Self-attention[1] maps a sequence of inputs $\mathbf{X} \in \mathbb{R}^{T \times D}$ to a sequence of outputs $\mathbf{Y} \in \mathbb{R}^{T \times D}$. For each attention head $h \in \{1, \ldots, H\}$ the inputs are projected into keys $\mathbf{K}_h = \mathbf{X}\boldsymbol{W}_h^{\text{key}}$ and queries $\mathbf{Q}_h = \mathbf{X}\boldsymbol{W}_h^{\text{query}}$ using head-specific projection matrices $\boldsymbol{W}_h^{\text{key}}, \boldsymbol{W}_h^{\text{query}}, \boldsymbol{W}_h^{\text{value}} \in \mathbb{R}^{D \times D^{\text{head}}}$ with $D^{\text{head}} = \frac{D}{H}$. The attention matrix can then be obtained as

$$\boldsymbol{\mathcal{A}} = \sigma\left(\left[\tilde{\mathbf{A}}_1 \ \tilde{\mathbf{A}}_2 \ \ldots \ \tilde{\mathbf{A}}_H \right] \right) \text{ with } \tilde{\mathbf{A}}_h = \frac{\mathbf{Q}_h \mathbf{K}_h^\top}{\sqrt{D^{\text{head}}}} \tag{1}$$

where $\boldsymbol{\mathcal{A}} = (a_{h,q,k})_{h,q,k} \in \mathbb{R}^{H \times T \times T}$ stacks the head-specific unnormalized attention matrices $(\tilde{\mathbf{A}}_h)_h$ into a three-dimensional array and applies a normalization operation $\sigma(\cdot)$. For linear attention, $\sigma(\cdot) = \mathrm{Id}(\cdot)$ is simply the identity whereas for softmax attention $\sigma(\cdot) = \mathrm{Softmax}(\cdot)$ applies the softmax operation for each head and query index independently across the key indices. Considering a single query index $q \in \{1, \ldots, T\}$ and $\mathbf{x}_k$ the k-th row of $\mathbf{X}$, multi-head attention (MHA) can be equivalently expressed as a hypernetwork with a key-query specific latent code:

$$\begin{aligned}
\mathrm{MHA}_q(\mathbf{X}) :=& \boldsymbol{W}^{\text{out}} \bigoplus_{h=1}^{H} \sum_{k=1}^{T} a_{h,q,k} \ \boldsymbol{W}_h^{\text{value}} \mathbf{x}_k && (2)\\
=& \sum_{h=1}^{H} \boldsymbol{W}_h^{\text{out}} \sum_{k=1}^{T} a_{h,q,k} \ \boldsymbol{W}_h^{\text{value}} \mathbf{x}_k && (3)\\
=& \sum_{k=1}^{T} \left(\sum_{h=1}^{H} \underbrace{a_{h,q,k}}_{\text{latent code}} \ \underbrace{\boldsymbol{W}_h^{\text{out}} \boldsymbol{W}_h^{\text{value}}}_{\text{hypernetwork}} \right) \mathbf{x}_k && (4)\\
=& \sum_{k=1}^{T} \underbrace{\mathbf{W}_{q,k}}_{\text{value network}} \ \mathbf{x}_k && (5)
\end{aligned}$$

where $\oplus$ denotes concatenation along the leading axis and the $\boldsymbol{W}_h^{\text{out}} \in \mathbb{R}^{D^{\text{head}} \times D}$ are defined as the slices of $\boldsymbol{W}^{\text{out}}$ such that $\boldsymbol{W}^{\text{out}} = \bigoplus_{h=1}^{H} \boldsymbol{W}_h^{\text{out}}$. See Figure 1C for a diagram of the computation.

As a result, the key-query specific attention scores over the multiple heads form a latent code where the number of heads H corresponds to the *latent dimension* of a hypernetwork. Accordingly, the dot products between any key-query pair over all heads can be interpreted as an amortized inference process to configure the computation implemented in the linear value network.

The key significance of our finding is that it suggests that the use of multiple heads in attention allows it to implement specialized, reusable operations through the latent code of a hypernetwork. Notably, it has been theoretically and empirically shown that hypernetworks support compositional generalization (Schug et al., 2024), providing a possible explanation for similar capabilities observed in transformers (Lake & Baroni, 2023; An et al., 2023; Hosseini et al., 2022). Note that inside of multi-head attention the same hypernetwork is used for every key-query index. Specifically, the hypernetwork linearly combines the *same* matrices $(\boldsymbol{W}_h^{\text{out}}, \boldsymbol{W}_h^{\text{value}})_h$ according to the weights given by $a_{h,q,k}$. This incentivizes reuse of latent codes and their associated operations implemented through the value network.

2.2 Hypernetwork Linear Attention (HYLA)

If the perspective of multi-head attention as a hypernetwork indeed reflects how it operates in practice, it should be possible to modify multi-head attention in a way that further reinforces the hypernetwork

[1]We focus on self-attention but the same derivation holds for cross-attention where the keys and values are obtained as the output of an encoder and the queries come from a decoder.

mechanism. For instance both the hypernetwork and the value network in multi-head attention are simple linear networks and could be replaced by nonlinear networks. Furthermore the normalization operation $\sigma(\cdot)$ could be used to encode inductive biases such as competition or sparsity in the latent code by imposing structural constraints.

Here, we focus on a simple modification to linear attention where we make the value network nonlinear and normalize the latent code along the head index (instead of along the key index as in standard softmax attention). Specifically, we define Hypernetwork Linear Attention (HYLA) to use a single hidden layer value network with nonlinearity. This should increase the expressivity of the subfunctions that are learned and recomposed by the layer. Noting that the output and value projections, $\boldsymbol{W}_h^{\text{out}}, \boldsymbol{W}_h^{\text{value}}$, already form a deep linear network in standard multi-head attention (c.f. Eq. 4) this can be achieved without adding additional parameters as follows:

$$\text{HYLA}_q(\mathbf{X}) = \sum_{k=1}^{T} \left(\sum_{h=1}^{H} a_{h,q,k} \, \boldsymbol{W}_h^{\text{out}} \right) \phi \left(\sum_{h=1}^{H} a_{h,q,k} \, \boldsymbol{W}_h^{\text{value}} \mathbf{x}_k \right) \tag{6}$$

$$= \sum_{k=1}^{T} \mathbf{W}'_{q,k} \phi(\mathbf{W}_{q,k} \mathbf{x}_k), \tag{7}$$

where $\phi(x)$ is an element-wise nonlinearity; here we set it to be a ReLU, $\phi(x) = \max(0, x)$. Additionally, we use $\sigma(\cdot) = \text{RMSHead}(\cdot)$, to normalize the attention scores for each key and query index independently across the head indices using $\text{RMSNorm}(\boldsymbol{x}) = \frac{\boldsymbol{x}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}}$ (without adding a learnable scalar as in Zhang & Sennrich (2019)). This ensures that the parameters of the value network generated by the hypernetwork maintain the variance preserving properties of neural network initializations that have been found to be important for the stability of gradient-based training (Glorot & Bengio, 2010). The resulting HYLA-layer is a simple drop-in replacement to standard attention. Notably, since the normalization operation $\sigma(\cdot)$ is local to the query and key index, it requires no communication across keys as opposed to softmax attention. By making the value network more expressive, we allow the network to implement more complex operations for a given latent code which we hypothesize strengthens the ability for compositional generalization.

3 Compositional generalization on fuzzy logic functions

We start by developing a fuzzy logic task with compositional structure, inspired by a similar task previously considered by Rahaman et al. (2021). Learning such tasks in an in-context learning setting will allow us to study the ability of multi-head attention-based models to compositionally generalize and analyze the structure of their latent code when solving each task.

Given a set of L scalars $\{x_1, ..., x_L\}$ with $x_i \in [0, 1]$ we define fuzzy logic operators on the elements of this set as the Zadeh operators (Zadeh, 1965):

$$x_i \wedge x_j := \min(x_i, x_j), \quad x_i \vee x_j := \max(x_i, x_j), \quad \bar{x}_i := 1 - x_i \tag{8}$$

These operators have the property that for boolean inputs $x_i \in \{0, 1\}$ the outputs coincide with the outputs of the corresponding Boolean operation. Each task is then defined as a function in disjunctive normal form consisting of K terms such as for example for $L = 4$ and $K = 2$

$$f(x_1, x_2, x_3, x_4) = \underbrace{(x_1 \wedge x_2 \wedge x_3 \wedge x_4)}_{\text{Term 1}} \vee \underbrace{(\bar{x}_1 \wedge x_2 \wedge \bar{x}_3 \wedge x_4)}_{\text{Term 2}} \tag{9}$$

where each term is a conjunction of all variables with negations applied to some of them. In order to test for compositional generalization, we index each of the 2^L possible terms, generate all possible combinations of K terms and hold out a fraction of the combinations for out-of-distribution (OOD) testing, ensuring that their constituent terms have been encountered in some tasks during training.

For each task, we present N examples in-context (see Figure 2A) where each token has dimension $L + 1$ consisting of the concatenated inputs sampled uniformly from $[0, 1]$ and the corresponding target of the function for the current task. For the final token we mask out the target and use the output logits of the final token to predict the target, measuring the loss using the mean-squared error.

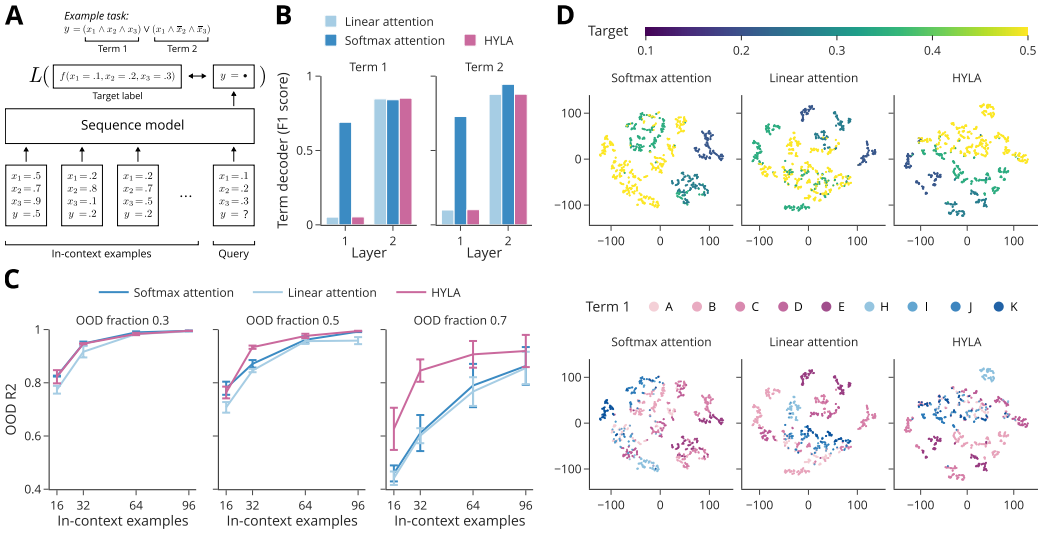

Figure 2: **Compositional generalization on fuzzy logic functions**. **A** We split fuzzy logic functions according to their constituent terms into train and out-of-distribution (OOD) set to measure compositional generalization in a sequence model that learns these functions in-context. **B** The latent code of the query token is predictive of the constituent terms underlying each task. Shown is the F1 score on unseen tasks of logistic regression classifiers for each layer and term, trained to predict the terms underlying each task based on the attention scores across the head index for the query attending to itself. **C** Task performance on unseen tasks reported as OOD R^2 for varying number of in-context examples and fraction of tasks held out during training. **D** tSNE visualization of the latent codes (attention scores across the head index for the query token attending to itself) colored according to the target label (top) and colored according to the first term of the fuzzy logic function of each task (bottom).

3.1 Compositional generalization

We first investigate how well multi-head attention based transformer models are able to learn our fuzzy logic tasks in-context and compositionally generalize to held out tasks. We vary the number of examples presented in context as well as the fraction of combinations of fuzzy logic terms held out from the training set. Figure 2C compares standard multi-head softmax attention and linear attention as well as our HYLA model in terms of their coefficient of determination (R^2) on the queries of held out tasks. We find that all models are able to solve the fuzzy logic task given sufficient examples are presented in context. As the fraction of held out tasks increases, compositional generalization performance starts decreasing for all models. Interestingly, HYLA is less affected by this decline than softmax and linear attention. We hypothesize that the more expressive value network allows to more efficiently represent the nonlinear terms being composed and thereby strengthens the inductive bias towards compositionality. Consistent with this interpretation, in-distribution the models show comparably small differences in performance (see Figure A3B+D). In Appendix A.1 we conduct additional model ablations that separate the contributions of the RMSHead normalization and the nonlinearity added to the value network, showing that both components in combination are required to obtain the observed performance improvements.

Our compositional split only contains novel combinations of *known* terms. A priori, it might be possible that the system learns to compose the basic fuzzy logic operations in a way that allows it to generalize to any fuzzy logic function, including those that contain novel combinations of *unknown* terms. However, testing on functions obtained as novel combinations of K unknown terms, we find that none of the models considered here is able to solve such a task (see Figure A3C).

3.2 Latent code structure

Next, we analyze the structure of the latent code for the different models solving the task. As suggested by Eq. 4, the attention scores for a given key-query pair across the head dimension configure the computation performed in the value network and accordingly we would expect it to

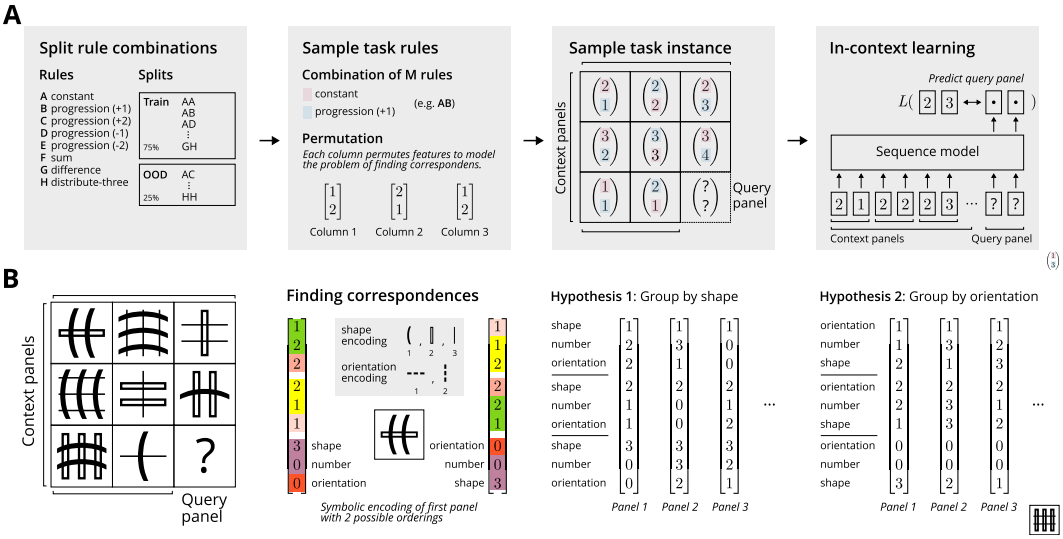

Figure 3: **SRAVEN**. **A** Illustration of SRAVEN task generation and the construction of the compositional generalization split. **B** Example problem instance illustrating a key challenge of the original Raven Progressive Matrices of *finding correspondences* (adapted from (Carpenter et al., 1990)). When attempting to solve this instance, different hypotheses over which figural elements are governed by a consistent rule across rows are possible. This is akin to different orderings of the symbolic features.

reflect the fuzzy logic function that needs to be implemented to correctly predict the output of the query token. To test this hypothesis, we collect the attention scores for the final query token attending to itself, obtaining a vector of dimension H for each layer and task. For this purpose we use the tasks held out from training, varying the query token while keeping inputs shown in-context fixed. In order to visualize the H-dimensional points we thereby obtain for each layer, we reduce the dimensionality using tSNE (Maaten & Hinton, 2008). The results of this analysis are shown in Figure 2D (for an extended figure showing all layers see Figure A2). In line with our hypothesis, the latent codes for each model form clusters. The structure of these clusters is only partially explained by the target value of the function for a given query value. Strikingly, coloring data points according to the fuzzy logic terms present underlying each function reveals that clusters in the latent codes closely correspond to the functions. Indeed, it is possible to decode the terms underlying each function from the latent code using a logistic regression classifier which is trained to predict the function terms given the latents and is evaluated on held out tasks as shown in Figure 2B.

4 SRAVEN: SYMBOLIC RAVEN

Next we will introduce an abstract reasoning task based on Raven Progressive Matrices (Raven, 1962) we refer to as SRAVEN. We develop this task to provide a challenging benchmark that specifically probes symbolic reasoning capabilities while giving us fine-grained control over the constituent parts of each task in order to assess compositional abilities and analyze latent code structure.

4.1 ABSTRACT REASONING BASED ON SYMBOLIC RAVEN PROGRESSIVE MATRICES

The original Raven Progressive Matrices test is a classic human intelligence test that probes the ability to induce abstract relations (Raven, 1962). It requires test subjects to generate and verify a potentially large number of hypotheses over rules that can parsimoniously explain a given problem instance (Carpenter et al., 1990). While there have been previous machine learning datasets inspired by Raven Progressive Matrices (Wang & Su, 2015; Zhang et al., 2019; Barrett et al., 2018; Hu et al., 2021), we seek to study a setting that is both symbolic, compositional and models an aspect which to the best of our knowledge has not been explicitly considered in prior benchmarks: we seek to model the key difficulty of Raven tasks referred to as *finding correspondences* by Carpenter et al. (1990) which requires to search through a large number of possible hypotheses. We discuss existing Raven-based benchmarks in Section 6.

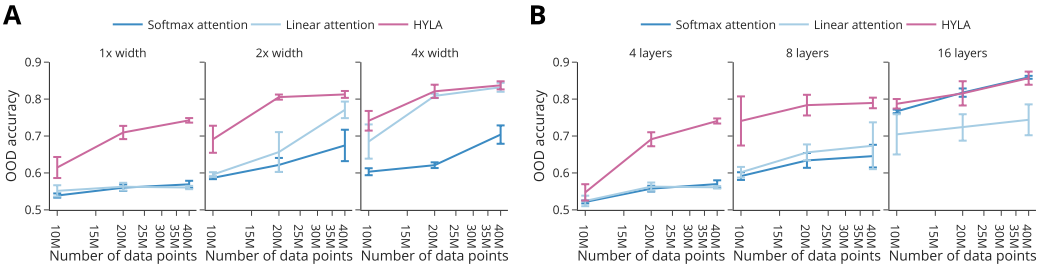

Figure 4: **Scaling data and model size on SRAVEN**. **A** Compositional generalization measured by OOD accuracy as a function of the number of training problem instances for different widths (scaling embedding and key-query-value dimensions) . **B** Same as A but increasing depth instead of width.

4.2 TASK DESCRIPTION

Figure 3A illustrates the generative process of SRAVEN. For each task, a matrix of eight context panels is presented and the goal is to predict the contents of the final query panel. In the original Raven Progressive Matrices Test, each panel typically displays a combination of geometrical shapes as for instance shown in Figure 3B. For our symbolic version, each panel is defined as a vector of integers of length M (in our experiments we set $M = 4$ unless stated otherwise). For each dimension $m \in \{1, 2, \ldots, M\}$ one of $R = 8$ possible rules is sampled as well as three corresponding sequences of three integers generated by the respective rule (see Figure 3A and Appendix B.3 for more details on the rules). We restrict rules to apply within each row while generally rules could also be applied both across columns and across rows. As a result we obtain nine panels where each row can be explained by the same set of underlying rules. In order to maintain a finite set of possible symbols both for the context panels and the query panel, we limit all tasks to contain only integers $\{0, 1, 2, \ldots, K-1\}$ (in our experiments we set $K = 8$). Arithmetic rules such as progression, sum or difference are therefore defined using modular arithmetic modulo K.

We use the first eight panels as context panels and present them sequentially to a sequence model which needs to predict each dimension of the final query panel independently. A prediction is considered correct if and only if all subpredictions were correct. In the original Raven task, a number of possible answer panels are presented alongside the context panels and test subjects are asked to pick the one which contains the correct answer (Raven, 1962). Since in our case the correct answer will consist of a combination of a finite set of known symbols, we can ask the agent solving the task to directly predict the missing symbols. Conveniently this helps avoid potential shortcuts for solving a task that might arise from a biased generative process to create multiple-choice answers as a popular prior Raven-based benchmark has been found to be affected by (Zhang et al., 2019; Hu et al., 2021).

One of the key features of SRAVEN is its compositional structure. Each task consists of a combination of a finite set of rule combinations. To systematically test whether a system trained on a subset of rule combinations can generalize to unseen combinations, we split all rule combinations into a train set and a test set. Unless noted otherwise, we hold out 25% of all possible rule combinations for evaluation in our experiments. Note that permutations of rule combinations (e.g. AB and BA) are considered to be the same and will only appear in one of the two sets. This ensures that at test time the agent is confronted with instances of tasks whose underlying combination of rules it has never encountered during training.

4.3 FINDING CORRESPONDENCES

Making the task symbolic instead of presenting images of geometric shapes comes with the advantage of a smaller input space allowing us to scale to larger model and data sizes in our experiments. However, it removes the visual representation learning step of discovering the right symbolic abstractions. We argue that while this is a difficult problem worth studying by itself, there is a another source of difficulty that arises even with access to symbolic abstractions. As Carpenter et al. (1990) argue in their theoretical account of the original Raven test, what makes many tasks difficult is the problem of *finding correspondences* of rules to feature dimensions.

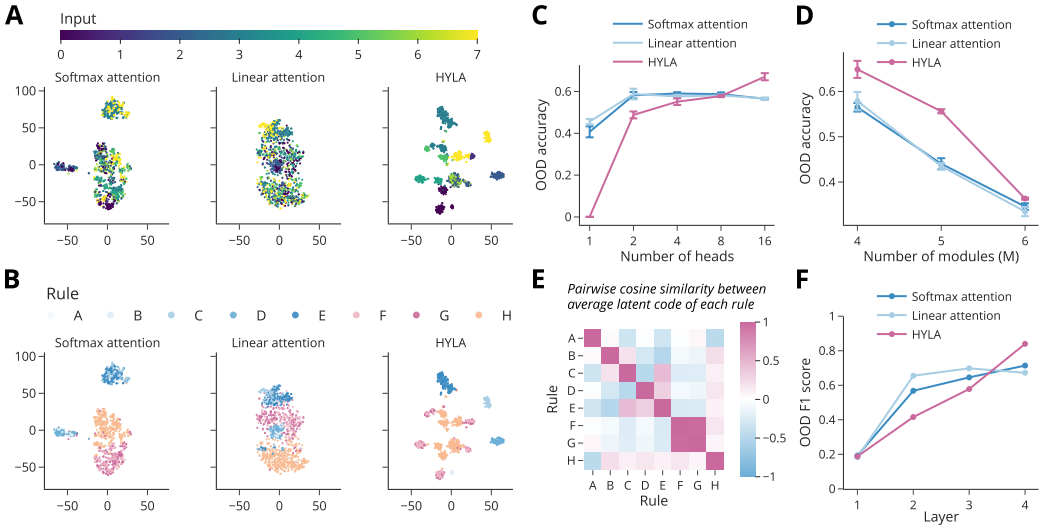

Figure 5: **Latent code structure of SRAVEN**. **A** tSNE visualizations of the final layer latent codes for the final query token colored by the magnitude of the predicted target value. **B** Same as A but colored by the ground-truth rule the model needs to apply to the query token to generated the correct prediction. **C** OOD accuracy for varying numbers of attention heads. For a single head the hypernetwork mechanism is absent which hampers OOD generalization. **D** The difficulty of SRAVEN can be parametrically controlled by varying the number of modules M. **E** Heatmap showing the pairwise cosine similarity between the average latent code of each rule for HYLA revealing how semantically related rules form clusters. For instance rule F (sum) and G (difference) are implemented with a very similar latent code indicating that the same code might be reused by flipping the sign of the operands. **F** Decoding performance of a logistic regression classifier trained to predict the ground-truth SRAVEN rule based on the latent code at the final query token of training tasks and evaluated on unseen OOD tasks revealing that the latent code is predictive of the implemented rule.

This is best illustrated when considering the example in Figure 3B showing an adapted version of Figure 5 of (Carpenter et al., 1990). When first attempting to solve this task, a test subject might hypothesize that there is a rule guiding the numerosity and orientations per shape. For example one might to try find a rule that governs all curved shapes, one rule that governs all lines and one rule that governs all rectangles. Translated into a symbolic encoding, this is akin to ordering symbols by the shape and trying to find rules along each dimension in this ordering. However, this approach turns out to not provide a parsimonious answer. The correct answer can instead be obtained by grouping the objects by their orientations, i.e. by finding a rule that applies to all horizontal objects and a rule that applies to all vertical objects. In the symbolic encoding this new correspondence of features amounts to a new ordering of the same symbolic encoding where symbols are now ordered by orientation and one needs to find rules guiding the numerosity and shape per orientation.

To model the property of finding correspondences in SRAVEN we similarly allow for permutations of the features. Specifically, we sample and apply a column-specific permutation of the features when generating each task such that the entries of the vectors encoding each panel are permuted with a consistent permutation for a given column across all rows. This makes finding the correct solution difficult since the number of possible hypotheses grows significantly (see Appendix B.1 for more details). Note that in principle a problem instance might have multiple valid but divergent answers making the task ill-posed. In the settings considered here this happens for less than half a percent of the problem instances. See Appendix B.2 for more details on handling such ambiguities.

4.4 RESULTS

Scaling model size and data enables compositional generalization. We first evaluate to what extent transformers can compositionally generalize on SRAVEN as we increase the number of problem instances trained on and scale up the models both in terms of width and depth. Figure 4 shows the results of these scaling experiments. Given sufficient data and model size, all model classes are able

to solve 80% of problem instances created from tasks held out during training. The inductive bias of making the value network more expressive by introducing a nonlinearity in HYLA seems to be beneficial for this task. Especially when presented with few problem instances and for small model sizes it outperforms both linear and softmax attention.

Disrupting the hypernetwork mechanism hurts compositional generalization. Varying the number of heads offers another way of manipulating the hypernetwork mechanism of multi-head attention. Specifically in the case of a single head the mechanism degenerates, only allowing the network to rescale the weights of the sole remaining value projection but no longer allowing it to compose multiple value projections, c.f. equation (4). In line with this argument, we observe a noticeably decrease in OOD performance in Figure 5C for the single head case.

The latent code is structured according to the rules of the task. We next analyze the latent code for the different models solving the task. To this end we collect the attention scores of the final query token attending to itself. Following the perspective of attention as a hypernetwork, we expect latent codes corresponding to the same rule to form clusters. Indeed, Figure 5B reveals a strikingly structured code for the final layer across models with clusters closely matching the rule to be inferred to correctly predict the query output for each given task. In comparison, Figure 5A colors the points in latent space by the magnitude of the inputs the query needs to attend to make the correct prediction. This tends to explain clusters more prominently in early layers as shown in the appendix in Figure A6 and for a 16 layer softmax transformer in Figure A7.

To obtain a better understanding of the semantics of the latent code, we explore the pairwise cosine similarity between average latent codes for each rule across tasks for HYLA. Figure 5E shows that semantically related rules like sum/difference can form clusters, suggesting that latent codes might even be reused to solve related task operations. Finally, we train a logistic regression classifier to predict the ground-truth SRAVEN rule based on the latent code at the final query token of in-distribution SRAVEN tasks and evaluate it on unseen OOD tasks. This reveals that especially for later layers the latent code is strongly predictive of the subfunctions performed by the network.

5 Language modeling

Multi-head attention has proven especially effective for language modeling. With language itself being highly compositional, it stands to question whether the hypernetwork mechanism inside of multi-head attention might play a part in explaining its success. To shed light on this question, we conducted language modeling experiments and evaluate the performance of HYLA in comparison to linear and softmax attention. We train decoder-only transformer models with 50M parameters autoregressively for 130 Billion tokens on the C4 dataset (Raffel et al., 2020). We find that strengthening the hypernetwork mechanism via HYLA improves performance over linear attention and performs closely to softmax attention despite being a linear attention variant itself. This is noteworthy considering the importance of softmax in language modeling hypothesized to be due to its role in binding and associative recall problems on which linear attention methods typically struggle (Arora et al., 2023; Olsson et al., 2022; Schlag et al., 2021). The fact that reinforcing the hypernetwork mechanism inside of multi-head attention as done by HYLA helps close the gap of linear attention to softmax attention suggests that the hypernetwork mechanism might be of practical relevance for understanding large-scale transformer models.

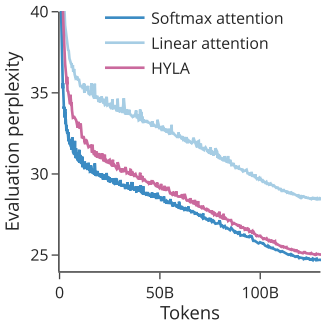

Figure 6: **Language modeling**. Performance of autoregressively trained decoder-only transformer models with 50M parameters over the course of training on the C4 dataset with 130B tokens (Raffel et al., 2020).

6 Related work

The role of multiple heads in attention. Prior work that has studied the question why the attention mechanism benefits from multiple heads has counterintuitively found that it is possible to prune

almost all but one head in some layers after training, sacrificing only relatively small amounts of performance (Voita et al., 2019; Michel et al., 2019). It has therefore been speculated that equipping the attention mechanism with multiple heads primarily aids stability during training (Liu et al., 2021). The hypernetwork perspective of attention offers an alternative account, suggesting that multiple heads create a way of configuring compositional computations in the value network. Seen from this perspective, the observation of singular heads dominating could point to a connection to the phenomenon of module collapse often observed when training modular systems in practice (Shazeer et al., 2017; Kirsch et al., 2018; Rosenbaum et al., 2018; Mittal et al., 2022).

Compositional generalization. Consistent with our observation on scaling model size on the SRAVEN task, (Hosseini et al., 2022) find that as pretrained large language models are scaled, their ability to compositionally generalize on in-context semantic parsing tasks improves. Similarly, outside the in-context learning regime, (Furrer et al., 2021) have demonstrated that pretrained large language models outperform many architectures specialized towards compositional generalization. Still, to what extent the ability for compositional generalization extends beyond the training distribution remains openly debated (Srivastava et al., 2023; Press et al., 2023; Dziri et al., 2023).

Raven-based tasks. A number of Raven-inspired tasks have been introduced to assess abstract reasoning capabilities of neural networks (Wang & Su, 2015; Zhang et al., 2019; Barrett et al., 2018; Hu et al., 2021). Common to all these variants including our version is an underlying generative task model based on a finite number of possible rules that are applied on one or more of the feature dimensions for each panel. Different from our version, these variants render the problem instances as images using simple geometrical shapes resulting in larger datasets and computational demand. Still we argue that the so generated tasks are not necessarily harder than the tasks of SRAVEN, given that SRAVEN models an additional difficulty of *finding correspondences* as detailed in Section 4.3 and its difficulty can be controlled parametrically; for instance by increasing the number of features.

7 DISCUSSION

We have proposed a novel decomposition of multi-head attention revealing that it can be interpreted as a hypernetwork that composes value networks specific to each key-query pair. Consistent with this perspective, we find empirically that the attention scores over the heads for each key-query pair form a functionally structured space, identifying reusable subfunctions in two abstract reasoning tasks. Furthermore, modifying multi-head attention to strengthen the hypernetwork mechanism improves compositional generalization on these tasks.

Adopting the hypernetwork perspective more broadly, multi-head attention can be interpreted as a particular choice for the level of granularity at which the hypernetwork parameterizes the value network: The value networks are *key-query specific* and are followed by a pooling step that sums the key-query specific outputs over the key index. In principle, other levels of granularity are possible. For instance the hypernetwork could parameterize a *query specific* value network which subsumes the aggregation over the key index. In the extreme case, it could directly parameterize the full sequence operation potentially facilitating the re-use of sequence-level operations. This also offers an interesting connection to attention-based graph neural networks (Veličković et al., 2018; Shirzad et al., 2023). Classically, a non-causal single layer transformer is seen as a fully connected graph where each token corresponds to a node and aggregation is done via attention (Battaglia et al., 2018). Our derivation suggests an alternative interpretation where the message function is a hypernetwork that subsumes the attention weights while the aggregation becomes the sum operator. Given the crucial role that aggregation plays in graph neural networks (Rosenbluth et al., 2023; Dudzik et al., 2023), a natural question is whether different pooling operators should be considered more generally for multi-head attention.

Limitations. We have focused our analysis on models trained from scratch in order to give us fine-grained control over what data is encountered during training. Many interesting behaviors emerge in large-scale models pretrained on diverse tasks. Investigating whether the resulting models form a similarly structured latent code is an interesting avenue for future work.

Ethics statement. This paper conducts foundational research aiming to illuminate the mechanisms of the widely used multi-head attention layer. While we foresee no immediate negative societal impact, we hope that it may improve our understanding of this widely deployed technology.

Reproducibility statement. To ensure the reproducibility of our work, we are providing the code for all our experiments as part of the supplementary material. We further expand upon the experimental details explained in the main text in Appendix C.

REFERENCES

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? Investigations with linear models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=0g0X4H8yN4I.

Shengnan An, Zeqi Lin, Qiang Fu, Bei Chen, Nanning Zheng, Jian-Guang Lou, and Dongmei Zhang. How Do In-Context Examples Affect Compositional Generalization? In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11027–11052, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.618. URL https://aclanthology.org/2023.acl-long.618.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and Improving Recall in Efficient Language Models, December 2023. URL https://arxiv.org/abs/2312.04927v1.

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, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola. The DeepMind JAX Ecosystem, 2020. URL http://github.com/deepmind.

David Barrett, Felix Hill, Adam Santoro, Ari Morcos, and Timothy Lillicrap. Measuring abstract reasoning in neural networks. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 511–520. PMLR, July 2018. URL https://proceedings.mlr.press/v80/barrett18a.html.

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks, October 2018. URL http://arxiv.org/abs/1806.01261. arXiv:1806.01261 [cs, stat].

Lukas Biewald. Experiment Tracking with Weights and Biases, 2020. URL https://www.wandb.com/.

Alfred Binet and Theophile Simon. The development of intelligence in children. 1961. Publisher: Appleton-Century-Crofts.

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.

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 (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.

P. A. Carpenter, M. A. Just, and P. Shell. What one intelligence test measures: a theoretical account of the processing in the Raven Progressive Matrices Test. *Psychological Review*, 97 (3):404–431, July 1990. ISSN 0033-295X. URL https://psycnet.apa.org/record/1990-27436-001.

Damai Dai, Yutao Sun, Li Dong, Yaru Hao, Shuming Ma, Zhifang Sui, and Furu Wei. Why Can GPT Learn In-Context? Language Models Secretly Perform Gradient Descent as Meta-Optimizers. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 4005–4019, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.247. URL https://aclanthology.org/2023.findings-acl.247.

Andrew Joseph Dudzik, Tamara von Glehn, Razvan Pascanu, and Petar Veličković. Asynchronous Algorithmic Alignment with Cocycles. In *The Second Learning on Graphs Conference*, 2023. URL https://openreview.net/forum?id=ba4bbZ4KoF.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and Fate: Limits of Transformers on Compositionality, June 2023. URL http://arxiv.org/abs/2305.18654. arXiv:2305.18654 [cs].

Jerry A. Fodor and Zenon W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1-2):3–71, March 1988. ISSN 00100277. doi: 10.1016/0010-0277(88)90031-5. URL https://linkinghub.elsevier.com/retrieve/pii/0010027788900315.

Daniel Furrer, Marc van Zee, Nathan Scales, and Nathanael Schärli. Compositional Generalization in Semantic Parsing: Pre-training vs. Specialized Architectures, September 2021. URL http://arxiv.org/abs/2007.08970. arXiv:2007.08970 [cs].

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

David Ha, Andrew M. Dai, and Quoc V. Le. HyperNetworks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkpACe1lx.

Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2023. URL http://github.com/google/flax.

Roee Hendel, Mor Geva, and Amir Globerson. In-Context Learning Creates Task Vectors, October 2023. URL http://arxiv.org/abs/2310.15916. arXiv:2310.15916 [cs].

Arian Hosseini, Ankit Vani, Dzmitry Bahdanau, Alessandro Sordoni, and Aaron Courville. On the Compositional Generalization Gap of In-Context Learning. In Jasmijn Bastings, Yonatan Belinkov, Yanai Elazar, Dieuwke Hupkes, Naomi Saphra, and Sarah Wiegreffe (eds.), *Proceedings of the Fifth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pp. 272–280, Abu Dhabi, United Arab Emirates (Hybrid), December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.blackboxnlp-1.22. URL https://aclanthology.org/2022.blackboxnlp-1.22.

Sheng Hu, Yuqing Ma, Xianglong Liu, Yanlu Wei, and Shihao Bai. Stratified Rule-Aware Network for Abstract Visual Reasoning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35 (2):1567–1574, May 2021. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v35i2.16248. URL https://ojs.aaai.org/index.php/AAAI/article/view/16248.

Jie Huang and Kevin Chen-Chuan Chang. Towards Reasoning in Large Language Models: A Survey. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 1049–1065, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.67. URL https://aclanthology.org/2023.findings-acl.67.

Plotly Technologies Inc. Collaborative data science, 2015. URL https://plot.ly. Place: Montreal, QC Publisher: Plotly Technologies Inc.

Louis Kirsch, Julius Kunze, and David Barber. Modular Networks: Learning to Decompose Neural Computation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/310ce61c90f3a46e340ee8257bc70e93-Paper.pdf.

Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In Eduardo Blanco and Wei Lu (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 66–71, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-2012. URL https://aclanthology.org/D18-2012.

Brenden M. Lake and Marco Baroni. Human-like systematic generalization through a meta-learning neural network. *Nature*, 623(7985):115–121, November 2023. ISSN 1476-4687. doi: 10.1038/s41586-023-06668-3. URL https://www.nature.com/articles/s41586-023-06668-3. Publisher: Nature Publishing Group.

Liyuan Liu, Jialu Liu, and Jiawei Han. Multi-head or Single-head? An Empirical Comparison for Transformer Training, June 2021. URL http://arxiv.org/abs/2106.09650. arXiv:2106.09650 [cs].

Peter J. Liu, Roman Novak, Jaehoon Lee, Mitchell Wortsman, Lechao Xiao, Katie Everett, Alexander A. Alemi, Mark Kurzeja, Pierre Marcenac, Izzeddin Gur, Simon Kornblith, Kelvin Xu, Gamaleldin Elsayed, Ian Fischer, Jeffrey Pennington, Ben Adlam, and Jascha-Sohl Dickstein. NanoDO: A minimal Transformer decoder-only language model implementation in JAX., 2024. URL http://github.com/google-deepmind/nanodo.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts, May 2017. URL http://arxiv.org/abs/1608.03983. arXiv:1608.03983 [cs, math].

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization, January 2019. URL http://arxiv.org/abs/1711.05101. arXiv:1711.05101 [cs, math].

Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. ISSN 1533-7928. URL http://jmlr.org/papers/v9/vandermaaten08a.html.

Paul Michel, Omer Levy, and Graham Neubig. Are Sixteen Heads Really Better than One?, November 2019. URL http://arxiv.org/abs/1905.10650. arXiv:1905.10650 [cs].

Sarthak Mittal, Yoshua Bengio, and Guillaume Lajoie. Is a Modular Architecture Enough? In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=3-3XMModtrx.

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, September 2022. URL http://arxiv.org/abs/2209.11895. arXiv:2209.11895 [cs].

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and Narrowing the Compositionality Gap in Language Models, May 2023. URL http://arxiv.org/abs/2210.03350. arXiv:2210.03350 [cs].

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Nasim Rahaman, Muhammad Waleed Gondal, Shruti Joshi, Peter Vincent Gehler, Yoshua Bengio, Francesco Locatello, and Bernhard Schölkopf. Dynamic Inference with Neural Interpreters. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=IUjt25DtqC4.

John C. Raven. *Advanced Progressive Matrices, Set II*. H. K. Lewis, London, 1962.

Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing Networks: Adaptive Selection of Non-Linear Functions for Multi-Task Learning. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=ry8dvM-R-.

Eran Rosenbluth, Jan Toenshoff, and Martin Grohe. Some Might Say All You Need Is Sum, May 2023. URL http://arxiv.org/abs/2302.11603. arXiv:2302.11603 [cs].

David E Rumelhart and James L McClelland. On learning the past tenses of English verbs. 1986. Publisher: Cambridge, MA: MIT Press.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear Transformers Are Secretly Fast Weight Programmers, June 2021. URL http://arxiv.org/abs/2102.11174. arXiv:2102.11174 [cs].

Jürgen Schmidhuber. Learning to Control Fast-Weight Memories: An Alternative to Dynamic Recurrent Networks. *Neural Computation*, 4(1):131–139, 1992. doi: 10.1162/neco.1992.4.1.131.

Simon Schug, Seijin Kobayashi, Yassir Akram, Maciej Wo\lczyk, Alexandra Proca, Johannes von Oswald, Razvan Pascanu, João Sacramento, and Angelika Steger. Discovering modular solutions that generalize compositionally. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=H98CVcX1eh.

Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J. Sutherland, and Ali Kemal Sinop. Exphormer: Sparse Transformers for Graphs. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 31613–31632. PMLR, July 2023.

Paul Smolensky. Connectionism, Constituency and the Language of Thought. In Barry M. Loewer and Georges Rey (eds.), *Meaning in Mind: Fodor and His Critics*. Blackwell, 1991.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, Agnieszka Kluska, Aitor Lewkowycz, Akshat Agarwal, Alethea Power, Alex Ray, Alex Warstadt, Alexander W. Kocurek, Ali Safaya, Ali Tazarv, Alice Xiang, Alicia Parrish, Allen Nie, Aman Hussain, Amanda Askell, Amanda Dsouza, Ambrose Slone, Ameet Rahane, Anantharaman S. Iyer, Anders Johan Andreassen, Andrea Madotto, Andrea Santilli, Andreas Stuhlmüller, Andrew M. Dai, Andrew La, Andrew Lampinen, Andy Zou, Angela Jiang, Angelica Chen, Anh Vuong, Animesh Gupta, Anna Gottardi, Antonio Norelli, Anu Venkatesh, Arash Gholamidavoodi, Arfa Tabassum, Arul Menezes,

Arun Kirubarajan, Asher Mullokandov, Ashish Sabharwal, Austin Herrick, Avia Efrat, Aykut Erdem, Ayla Karakaş, B. Ryan Roberts, Bao Sheng Loe, Barret Zoph, Bart\lomiej Bojanowski, Batuhan Özyurt, Behnam Hedayatnia, Behnam Neyshabur, Benjamin Inden, Benno Stein, Berk Ekmekci, Bill Yuchen Lin, Blake Howald, Bryan Orinion, Cameron Diao, Cameron Dour, Catherine Stinson, Cedrick Argueta, Cesar Ferri, Chandan Singh, Charles Rathkopf, Chenlin Meng, Chitta Baral, Chiyu Wu, Chris Callison-Burch, Christopher Waites, Christian Voigt, Christopher D. Manning, Christopher Potts, Cindy Ramirez, Clara E. Rivera, Clemencia Siro, Colin Raffel, Courtney Ashcraft, Cristina Garbacea, Damien Sileo, Dan Garrette, Dan Hendrycks, Dan Kilman, Dan Roth, C. Daniel Freeman, Daniel Khashabi, Daniel Levy, Daniel Moseguí González, Danielle Perszyk, Danny Hernandez, Danqi Chen, Daphne Ippolito, Dar Gilboa, David Dohan, David Drakard, David Jurgens, Debajyoti Datta, Deep Ganguli, Denis Emelin, Denis Kleyko, Deniz Yuret, Derek Chen, Derek Tam, Dieuwke Hupkes, Diganta Misra, Dilyar Buzan, Dimitri Coelho Mollo, Diyi Yang, Dong-Ho Lee, Dylan Schrader, Ekaterina Shutova, Ekin Dogus Cubuk, Elad Segal, Eleanor Hagerman, Elizabeth Barnes, Elizabeth Donoway, Ellie Pavlick, Emanuele Rodolà, Emma Lam, Eric Chu, Eric Tang, Erkut Erdem, Ernie Chang, Ethan A. Chi, Ethan Dyer, Ethan Jerzak, Ethan Kim, Eunice Engefu Manyasi, Evgenii Zheltonozhskii, Fanyue Xia, Fatemeh Siar, Fernando Martínez-Plumed, Francesca Happé, Francois Chollet, Frieda Rong, Gaurav Mishra, Genta Indra Winata, Gerard de Melo, Germán Kruszewski, Giambattista Parascandolo, Giorgio Mariani, Gloria Xinyue Wang, Gonzalo Jaimovitch-Lopez, Gregor Betz, Guy Gur-Ari, Hana Galijasevic, Hannah Kim, Hannah Rashkin, Hannaneh Hajishirzi, Harsh Mehta, Hayden Bogar, Henry Francis Anthony Shevlin, Hinrich Schuetze, Hiromu Yakura, Hongming Zhang, Hugh Mee Wong, Ian Ng, Isaac Noble, Jaap Jumelet, Jack Geissinger, Jackson Kernion, Jacob Hilton, Jaehoon Lee, Jaime Fernández Fisac, James B. Simon, James Koppel, James Zheng, James Zou, Jan Kocon, Jana Thompson, Janelle Wingfield, Jared Kaplan, Jarema Radom, Jascha Sohl-Dickstein, Jason Phang, Jason Wei, Jason Yosinski, Jekaterina Novikova, Jelle Bosscher, Jennifer Marsh, Jeremy Kim, Jeroen Taal, Jesse Engel, Jesujoba Alabi, Jiacheng Xu, Jiaming Song, Jillian Tang, Joan Waweru, John Burden, John Miller, John U. Balis, Jonathan Batchelder, Jonathan Berant, Jörg Frohberg, Jos Rozen, Jose Hernandez-Orallo, Joseph Boudeman, Joseph Guerr, Joseph Jones, Joshua B. Tenenbaum, Joshua S. Rule, Joyce Chua, Kamil Kanclerz, Karen Livescu, Karl Krauth, Karthik Gopalakrishnan, Katerina Ignatyeva, Katja Markert, Kaustubh Dhole, Kevin Gimpel, Kevin Omondi, Kory Wallace Mathewson, Kristen Chiafullo, Ksenia Shkaruta, Kumar Shridhar, Kyle McDonell, Kyle Richardson, Laria Reynolds, Leo Gao, Li Zhang, Liam Dugan, Lianhui Qin, Lidia Contreras-Ochando, Louis-Philippe Morency, Luca Moschella, Lucas Lam, Lucy Noble, Ludwig Schmidt, Luheng He, Luis Oliveros-Colón, Luke Metz, Lütfi Kerem Senel, Maarten Bosma, Maarten Sap, Maartje Ter Hoeve, Maheen Farooqi, Manaal Faruqui, Mantas Mazeika, Marco Baturan, Marco Marelli, Marco Maru, Maria Jose Ramirez-Quintana, Marie Tolkiehn, Mario Giulianelli, Martha Lewis, Martin Potthast, Matthew L. Leavitt, Matthias Hagen, Mátyás Schubert, Medina Orduna Baitemirova, Melody Arnaud, Melvin McElrath, Michael Andrew Yee, Michael Cohen, Michael Gu, Michael Ivanitskiy, Michael Starritt, Michael Strube, Michal Swedrowski, Michele Bevilacqua, Michihiro Yasunaga, Mihir Kale, Mike Cain, Mimee Xu, Mirac Suzgun, Mitch Walker, Mo Tiwari, Mohit Bansal, Moin Aminnaseri, Mor Geva, Mozhdeh Gheini, Mukund Varma T, Nanyun Peng, Nathan Andrew Chi, Nayeon Lee, Neta Gur-Ari Krakover, Nicholas Cameron, Nicholas Roberts, Nick Doiron, Nicole Martinez, Nikita Nangia, Niklas Deckers, Niklas Muennighoff, Nitish Shirish Keskar, Niveditha S. Iyer, Noah Constant, Noah Fiedel, Nuan Wen, Oliver Zhang, Omar Agha, Omar Elbaghdadi, Omer Levy, Owain Evans, Pablo Antonio Moreno Casares, Parth Doshi, Pascale Fung, Paul Pu Liang, Paul Vicol, Pegah Alipoormolabashi, Peiyuan Liao, Percy Liang, Peter W. Chang, Peter Eckersley, Phu Mon Htut, Pinyu Hwang, Piotr Milkowski, Piyush Patil, Pouya Pezeshkpour, Priti Oli, Qiaozhu Mei, Qing Lyu, Qinlang Chen, Rabin Banjade, Rachel Etta Rudolph, Raefer Gabriel, Rahel Habacker, Ramon Risco, Raphaël Millière, Rhythm Garg, Richard Barnes, Rif A. Saurous, Riku Arakawa, Robbe Raymaekers, Robert Frank, Rohan Sikand, Roman Novak, Roman Sitelew, Ronan Le Bras, Rosanne Liu, Rowan Jacobs, Rui Zhang, Russ Salakhutdinov, Ryan Andrew Chi, Seungjae Ryan Lee, Ryan Stovall, Ryan Teehan, Rylan Yang, Sahib Singh, Saif M. Mohammad, Sajant Anand, Sam Dillavou, Sam Shleifer, Sam Wiseman, Samuel Gruetter, Samuel R. Bowman, Samuel Stern Schoenholz, Sanghyun Han, Sanjeev Kwatra, Sarah A. Rous, Sarik Ghazarian, Sayan Ghosh, Sean Casey, Sebastian Bischoff, Sebastian Gehrmann, Sebastian Schuster, Sepideh Sadeghi, Shadi Hamdan, Sharon Zhou, Shashank Srivastava, Sherry Shi, Shikhar Singh, Shima Asaadi, Shixiang Shane Gu, Shubh Pachchigar, Shubham Toshniwal, Shyam Upadhyay, Shyamolima Shammie Debnath, Siamak Shakeri, Simon Thormeyer, Simone Melzi, Siva Reddy, Sneha Priscilla Makini, Soo-

Hwan Lee, Spencer Torene, Sriharsha Hatwar, Stanislas Dehaene, Stefan Divic, Stefano Ermon, Stella Biderman, Stephanie Lin, Stephen Prasad, Steven Piantadosi, Stuart Shieber, Summer Misherghi, Svetlana Kiritchenko, Swaroop Mishra, Tal Linzen, Tal Schuster, Tao Li, Tao Yu, Tariq Ali, Tatsunori Hashimoto, Te-Lin Wu, Théo Desbordes, Theodore Rothschild, Thomas Phan, Tianle Wang, Tiberius Nkinyili, Timo Schick, Timofei Kornev, Titus Tunduny, Tobias Gerstenberg, Trenton Chang, Trishala Neeraj, Tushar Khot, Tyler Shultz, Uri Shaham, Vedant Misra, Vera Demberg, Victoria Nyamai, Vikas Raunak, Vinay Venkatesh Ramasesh, vinay uday prabhu, Vishakh Padmakumar, Vivek Srikumar, William Fedus, William Saunders, William Zhang, Wout Vossen, Xiang Ren, Xiaoyu Tong, Xinran Zhao, Xinyi Wu, Xudong Shen, Yadollah Yaghoobzadeh, Yair Lakretz, Yangqiu Song, Yasaman Bahri, Yejin Choi, Yichi Yang, Yiding Hao, Yifu Chen, Yonatan Belinkov, Yu Hou, Yufang Hou, Yuntao Bai, Zachary Seid, Zhuoye Zhao, Zijian Wang, Zijie J. Wang, Zirui Wang, and Ziyi Wu. Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=uyTL5Bvosj.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with Rotary Position Embedding. *Neurocomputing*, 568:127063, February 2024. ISSN 0925-2312. doi: 10.1016/j.neucom.2023.127063. URL https://www.sciencedirect.com/science/article/pii/S0925231223011864.

Eric Todd, Millicent Li, Arnab Sen Sharma, Aaron Mueller, Byron C. Wallace, and David Bau. Function Vectors in Large Language Models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=AwyxtyMwaG.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *International Conference on Learning Representations*, 2018.

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 (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 5797–5808, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1580. URL https://aclanthology.org/P19-1580.

Johannes von Oswald, Eyvind Niklasson, Maximilian Schlegel, Seijin Kobayashi, Nicolas Zucchet, Nino Scherrer, Nolan Miller, Mark Sandler, Blaise Agüera y Arcas, Max Vladymyrov, Razvan Pascanu, and João Sacramento. Uncovering mesa-optimization algorithms in Transformers, September 2023. URL http://arxiv.org/abs/2309.05858. arXiv:2309.05858 [cs].

Ke Wang and Zhendong Su. Automatic generation of Raven's progressive matrices. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pp. 903–909, Buenos Aires, Argentina, July 2015. AAAI Press. ISBN 978-1-57735-738-4. URL https://www.ijcai.org/Proceedings/15/Papers/132.pdf.

David Wechsler. The measurement and appraisal of adult intelligence. *Academic Medicine*, 33(9): 706, 1958. Publisher: LWW.

L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, June 1965. ISSN 0019-9958. doi: 10.1016/S0019-9958(65)90241-X. URL https://www.sciencedirect.com/science/article/pii/S001999586590241X.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

Chi Zhang, Feng Gao, Baoxiong Jia, Yixin Zhu, and Song-Chun Zhu. RAVEN: A Dataset for Relational and Analogical Visual REasoNing. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5312–5322, 2019. doi: 10.1109/CVPR.2019.00546.

Algorithm 1 Multi-head softmax attention

Input: x
Require: `emb_dim`, `heads`, `h_dim`

q = **Dense**(shape=(`heads`, `h_dim`), axis=-1)(x)
k = **Dense**(shape=(`heads`, `h_dim`), axis=-1)(x)
v = **Dense**(shape=(`heads`, `h_dim`), axis=-1)(x)

a = **einsum**(`"...qhd,...khd->...hqk"`, q, k)
a = **softmax**(a, axis=-1)

z = **einsum**(`"...hqk,...khd->...qhd"`, a,v)

y = **Dense**(features=`emb_dim`, axis=(-2, -1))(z)

return y

Algorithm 2 Hypernetwork linear attention

Input: x
Require: `emb_dim`, `heads`, `h_dim`

q = **Dense**(shape=(`heads`, `h_dim`), axis=-1)(x)
k = **Dense**(shape=(`heads`, `h_dim`), axis=-1)(x)
v = **Dense**(shape=(`heads`, `h_dim`), axis=-1)(x)

a = **einsum**(`"...qhd,...khd->...hqk"`, q, k)
a = **rms_norm**(a, axis=-3)

z = **einsum**("...hqk,...khd->...qkd", a, v)
z = **relu**(z)
z = **einsum**(`"...hqk,...qkd->...qhd"`, a, z)
y = **Dense**(features=`emb_dim`, axis=(-2, -1))(z)

return y

Figure A1: Pseudocode comparing multi-head softmax attention to hypernetwork linear attention. Differences between the two are highlighted in yellow.

A Additional results

A.1 Model ablations

Compared to linear attention, HYLA introduces three modifications: (1) normalization of the attention scores across the head indices using RMSHead, (2) applying the attention-weighted sum operation also to the output projection and (3) inserting a nonlinearity in the value network (c.f. the pseudocode for HYLA shown in Figure 1). To delineate the contributions of these modifications, we run an ablation study on the fuzzy logic task and SRAVEN shown in Table A1. We find that while simply applying RMSHead to linear attention (Linear Attention + RMSHead) and also the attention-weighted sum operation without nonlinearity (HYLA $-$ nonlinearity) both by themselves can slightly boost performance, the full HYLA model generally performs best.

Increasing depth of the value network. To further test our hypothesis that it is the hypernetwork mechanism inside multi-head attention which improves compositional generalization, we explore making the value network configured by the hypernetwork deeper, adding an additional layer as follows,

$$\text{HYLA}_q^+(\mathbf{X}) = \sum_{k=1}^{T} \left(\sum_{h=1}^{H} a_{h,q,k} \, \boldsymbol{W}_h^{\text{out}} \right) \phi \left(\sum_{h=1}^{H} a_{h,q,k} \, \boldsymbol{W}_h^{\text{value}'} \phi \left(\sum_{h=1}^{H} a_{h,q,k} \, \boldsymbol{W}_h^{\text{value}} \mathbf{x}_k \right) \right), \quad (10)$$

where $\boldsymbol{W}_h^{\text{value}'} \in \mathbb{R}^{D^{\text{key}} \times D^{\text{key}}}$ is a second layer value matrix which increases the number of parameters of this model. Consistent with our hypothesis, this model also demonstrates strong OOD performance as reported in Table A1.

Replacing RMSHead normalization with the softmax. We perform an additional ablation where we replace the RMSHead normalization of HYLA with the standard softmax that normalizes across keys but not across heads. As shown in Table A1 this model performs poorly.

Replacing feedforward layers with mixture of experts. Sparsely gated mixture of experts (MoE) have been proposed as a replacement for the feedforward layer in the transformer block (Shazeer et al., 2017). In this model, each token is routed to a sparse subset of k out of E experts

$$f_{\text{MoE}}(\boldsymbol{x}) = \sum_{i=1}^{E} \text{Softmax}(\text{Top_k}(\boldsymbol{W}^{\text{gate}} \boldsymbol{x})) \cdot \boldsymbol{W}_i^{\text{up}} \phi(\boldsymbol{W}_i^{\text{down}} \boldsymbol{x}).$$

Intuitively, this could allow individual experts to specialize on specific subtasks that can be composed by the router. However, it is not well understood if MoEs improve compositional generalization of

transformers in practice. Here, we perform additional experiments where we replace the feedforward layers with MoEs on the SRAVEN and the fuzzy logic task reported in Table A1. While we find moderate general improvements, these improvements seem to be similar to other increases of model capacity such as increasing the width/depth as discussed in Section 4.4.

Table A1: **Model ablations.** Left column reports the OOD R2 score on the fuzzy logic task studied in Figure 2 for a sequence length of 32 and 70% of tasks held out from training. Right column reports the OOD accuracy on the SRAVEN task studied in Figure 4 for a 4-layer transformer trained on 20M problem instances. We report the standard error over 3 seeds.

	Fuzzy logic: OOD R2	SRAVEN: OOD accuracy
Softmax attention	63.28 ± 2.31	56.56 ± 1.05
Linear attention	59.89 ± 5.22	56.30 ± 1.11
HYLA	81.13 ± 7.77	69.13 ± 1.90
Linear attention + RMSHead	68.93 ± 5.10	59.01 ± 2.50
Linear attention + RMSHead + nonlinear value	56.91 ± 4.25	55.38 ± 0.32
HYLA − RMSHead	81.27 ± 7.88	62.53 ± 5.73
HYLA − nonlinearity	84.54 ± 5.32	56.54 ± 0.21
HYLA − nonlinearity − RMSHead	79.42 ± 7.43	56.33 ± 1.41
HYLA − RMSHead + softmax	70.98 ± 7.08	38.21 ± 2.98
HYLA + deep value network	87.65 ± 4.54	66.59 ± 0.08
HYLA + softmax	82.47 ± 4.22	62.21 ± 3.08
Softmax attention + MoE	63.83 ± 6.07	57.68 ± 1.31
Linear attention + MoE	60.09 ± 6.16	64.34 ± 4.3
HYLA + MoE	82.37 ± 6.55	66.94 ± 3.94

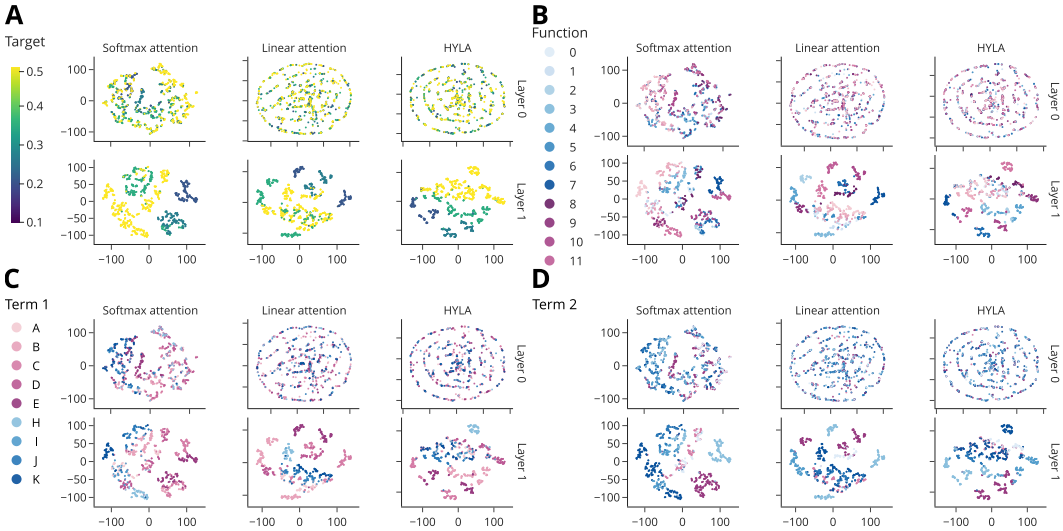

Figure A2: tSNE visualization of the attention scores across the head index for the query token on the fuzzy logic task for a sequence length of 32 and 70% of tasks held out from training colored according to (A) target label (B) fuzzy logic function, (C) term 1 and (D) term 2.

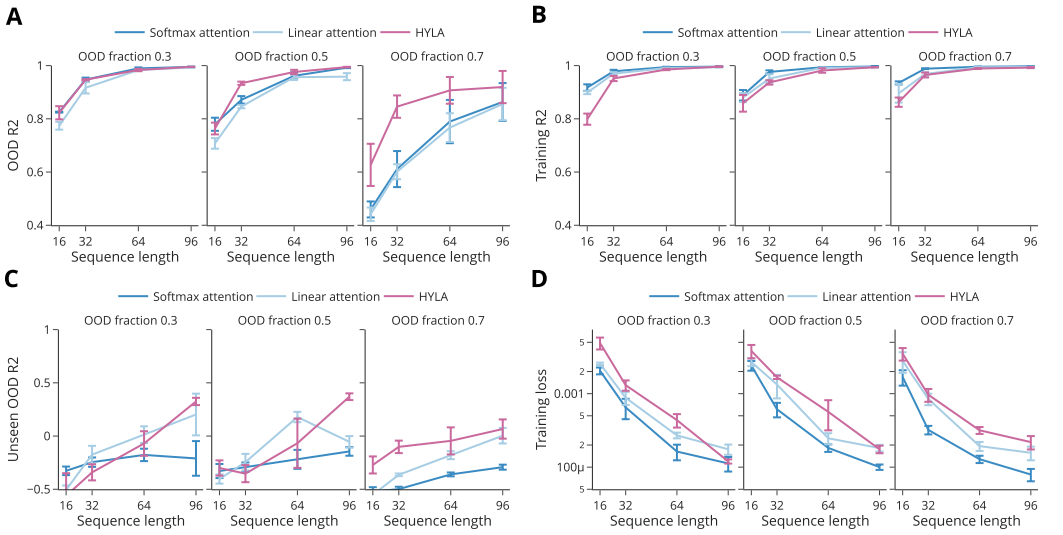

Figure A3: **Fuzzy logic task performance metrics.** Various task performance metrics on the fuzzy logic task for varying number of in-context examples and fraction of tasks held-out during training. **A** Coefficient of determination (R^2) on tasks consisting of unseen combinations of seen terms (same as Figure 2**C**) **B** Coefficient of determination (R^2) on training tasks. **C** Coefficient of determination (R^2) on tasks containing terms not encountered during training. As expected all methods fail in this setting. **D** Training mean squared error loss.

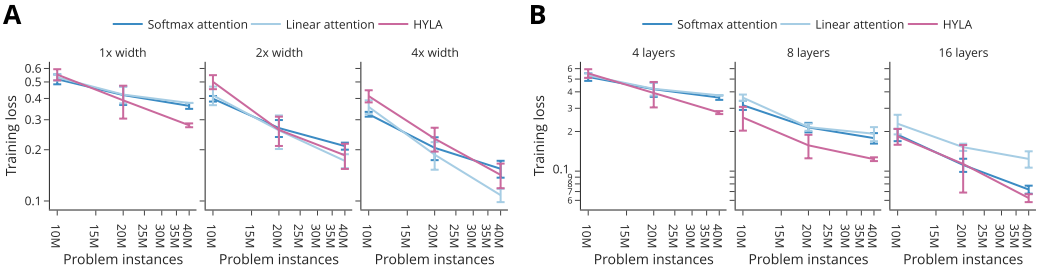

Figure A4: **Training loss when scaling data and model size on SRAVEN.** **A** Training loss as a function of the number of training problem instances for different widths (scaling embedding and key-query-value dimensions). **B** Same as A but increasing depth instead of width.

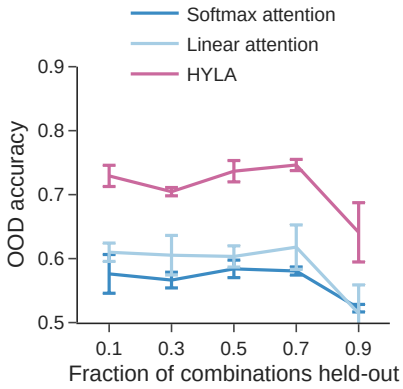

Figure A5: **Varying the fraction of OOD tasks on SRAVEN.** OOD accuracy on SRAVEN as varying fractions of tasks specified by their rule combinations are held-out during training.

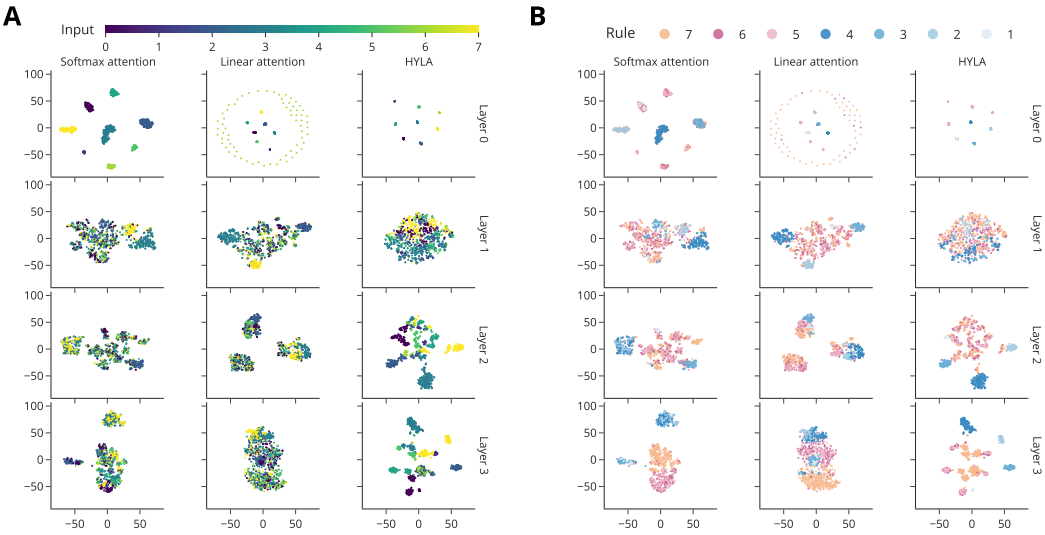

Figure A6: **Latent code on SRAVEN.** **A** tSNE visualization of the latent code predicting the final query token colored by the magnitude of the predicted target value for 4-layer transformers trained on 40M problem instances. **B** Same as A but colored by rule.

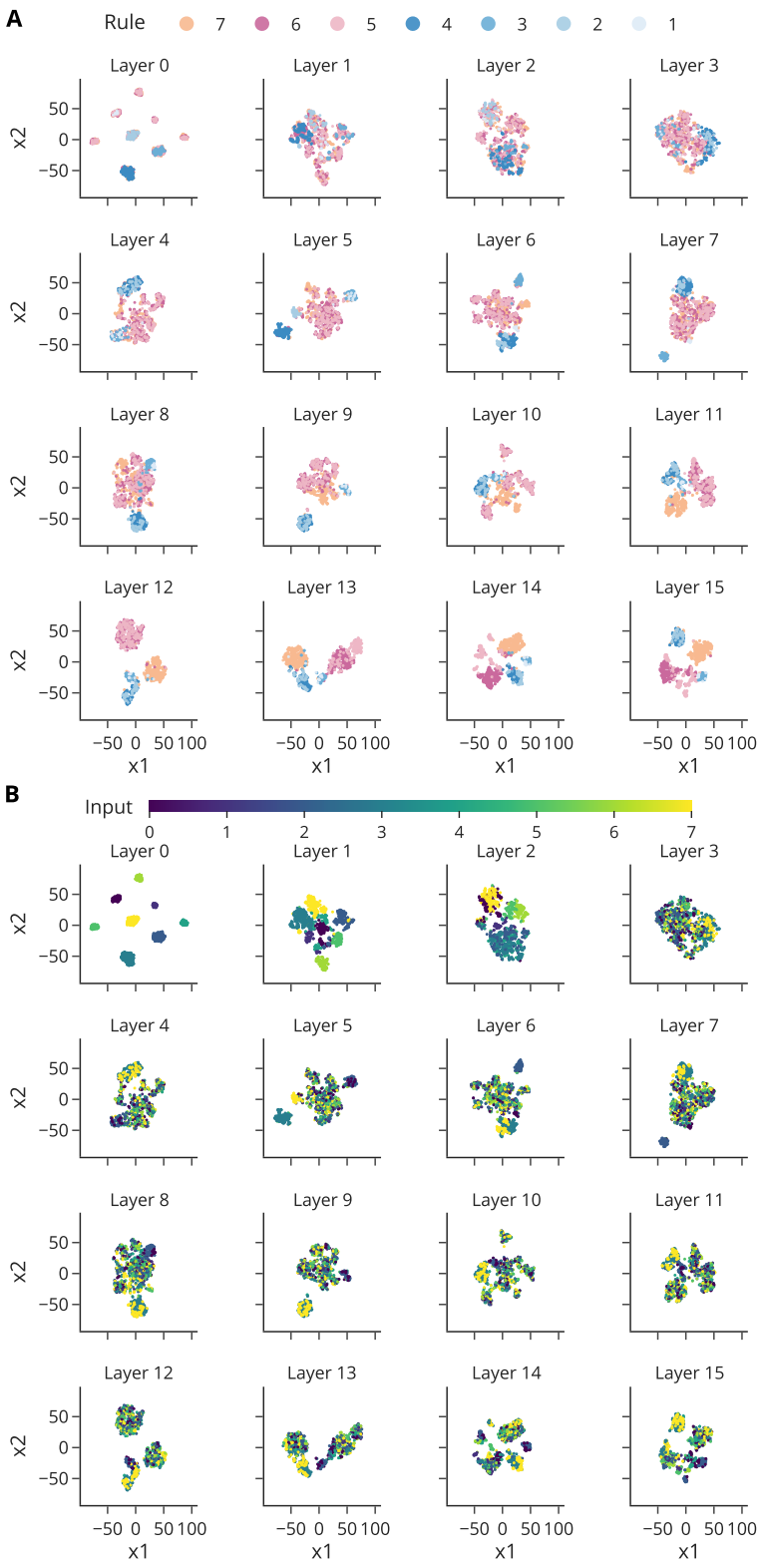

Figure A7: **Latent code visualization in SRAVEN for a softmax transformer of depth 16**. **A** tSNE visualization of the latent code predicting the final query token colored by the rule to be inferred for a softmax transformer of depth 16 trained on 40M problem instances. **B** Same plot as in A but colored by the magnitude of the predicted target value.

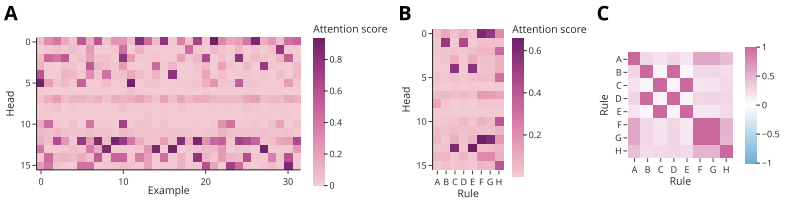

Figure A8: **Attention scores of a softmax transformer on SRAVEN. A** Random sample of the attention scores / latent codes for the final key-query pair on OOD SRAVEN tasks in the final layer of a softmax transformer (same model as shown in Figure A6). **B** Average latent codes per rule for the same model as in A **C** Cosine similarity between the average latent codes.

B SRAVEN

B.1 NUMBER OF POSSIBLE TASKS AND INSTANCES

In the general case, the number of possible tasks in SRAVEN is a product of the number of possible rule combinations $\binom{R+M-1}{M}$ and the number of possible unique permutations $((M!)^{G-1} - M)$ where G is the number of columns of the grid (in Raven-based tasks typically $G = 3$). Figure A9A shows the number of possible SRAVEN tasks and Figure A9B shows the number of possible SRAVEN problem instances. Note specifically that the latter is orders of magnitudes larger than the number of training examples in our experiments for $M = 4$, $K = 8$.

B.2 AMBIGUITY

In principle, a problem instance generated given a set of rules and permutations might have an ambiguous answer. That is there is at least one other set of rules and permutations that fit all context panels but prescribe a different final query panel. One possibility is to check for such examples when generating task instances and reject ambiguous ones. This process is prohibitively expensive as it requires a search over the exponentially many possible permutations. We therefore resort to a Monte-Carlo estimation of the fraction of ambiguous instances. Table A2 shows the results for various parameterizations of the SRAVEN task. In the setting we consider here, $(M = 4, K = 8)$, the probabilities are negligible and we do not take measures to filter out ambiguous examples given the computational overhead it produces in the data generation process.

B.3 RULES

`sraven` contains the following rules:

1. `constant`: Each row consists of a random but fixed integer in the range 1 to K.
2. `progression (+1)`: The first element of each row is sampled uniformly at random and incremented by 1 modulo K for each successive column.
3. `progression (+2)`: The first element of each row is sampled uniformly at random and incremented by 2 modulo K for each successive column.
4. `progression (-1)`: The first element of each row is sampled uniformly at random and decremented by 1 modulo K for each successive column.
5. `progression (-2)`: The first element of each row is sampled uniformly at random and decremented by 2 modulo K for each successive column.
6. `addition`: Two elements are sampled uniformly at random for each row and added modulo K to obtain the last column.
7. `subtraction`: Two elements are sampled uniformly at random for each row and subtracted modulo K to obtain the last column.
8. `distribute three`: Three elements are sampled uniformly at random and presented in three independently sampled random permutations for each row.

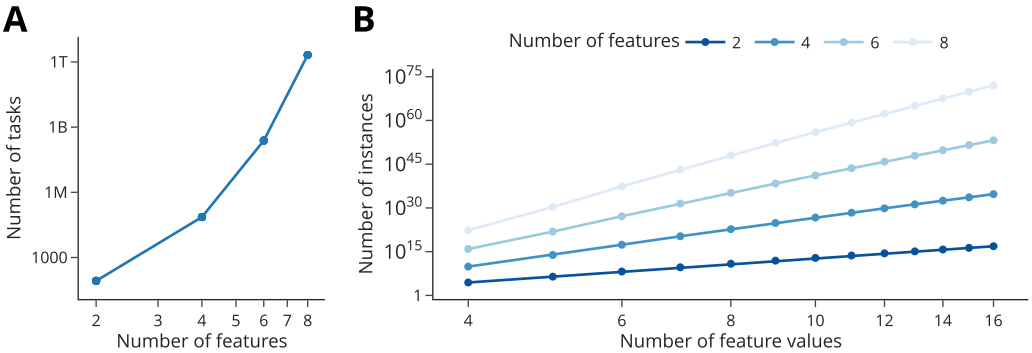

Figure A9: **A** Number of possible SRAVEN tasks as a function the number of features (M). **B** Number of possible SRAVEN problem instances as a function of the number of features (M) and the number of feature values (K).

Table A2: Monte Carlo estimation of fraction of ambiguous examples over 4096 problem instances $\pm$ standard error. Bold used to denote the setting used for all experiments in this paper.

Number of features (M)	Number of feature values (K)	Estimated fraction of ambiguous examples
4	4	0.0642 ± 0.0038
4	**8**	$\mathbf{0.0032 \pm 0.0009}$
4	16	0.0005 ± 0.0003

C EXPERIMENTAL DETAILS

Architecture. For all models we use a standard decoder-only transformer architecture where each block is structured as

$$\mathbf{Z} = \texttt{MultiHeadAttention}(\texttt{LayerNorm}(\mathbf{X})) + \mathbf{X}$$
$$\mathbf{Y} = \texttt{FeedForward}(\texttt{LayerNorm}(\mathbf{Z})) + \mathbf{Z}.$$

`MultiHeadAttention` is either linear attention, softmax attention or hypernetwork linear attention (HYLA) and uses T5-style relative positional embeddings (Raffel et al., 2020). For the `FeedForward` layer, we employ a standard single-hidden layer MLP with GeLU nonlinearity applied to each position in the sequence independently. For our language modeling experiments we use rotary positional encodings (Su et al., 2024).

Tokenization. For the fuzzy logic task we directly use the concatenated input examples as tokens. Similarly for SRAVEN we use one-hot encodings of the integer inputs as the input tokens. In both cases the input tokens are mapped into the embedding space of the transformer using a fully connected dense layer and another fully connect dense layer is used to produce the output logits. For the language modeling experiment we use the default sentencepiece tokenizer (Kudo & Richardson, 2018) of NanoDo (Liu et al., 2024) with a vocabulary size of 32000 and use the transposed embedding matrix to produce the logits.

Optimization. We use the AdamW optimizer (Loshchilov & Hutter, 2019) with linear warmup starting from a learning rate of 0 followed by a cosine decay learning rate scheduler (Loshchilov & Hutter, 2017) that decays the learning rate to 0.1 times the base learning rate. We exempt biases and LayerNorm parameters from weight decay. In the fuzzy logic task we use the mean-squared error on the logits of the query token whereas in SRAVEN we use the softmax cross-entropy loss on the logits of all query tokens. The language modeling task uses an autoregressive softmax cross-entropy loss with causal masking applied to the attention mechanism as well as mixed-precision training.

Hyperparameters. For all tasks and models we perform a grid search over the learning rate, weight decay and warmup steps. We report the search grid as well as all other hyperparameters in Table A3.

Table A3: Hyperparameters for all tasks and methods. Lists of values indicates that a grid search over all points contained in the list have been conducted and for each method the optimal combination has been picked. The width factor multiplies the embedding dimension, key-query-value dimension and MLP dimension.

Parameter	Fuzzy logic	SRAVEN	Language modeling
`batch_size`	128	128	256
`num_layers`	2	4 / 8 / 16	6
`width_factor`	1	1 / 2 / 4	1
`emb_dim`	128	128	512
`kqv_dim`	16	64	64
`mlp_dim`	256	256	2048
`num_heads`	8	16	8
`learning_rate`	$[0.001, 0.003]$	$[0.0003, 0.001]$	$[0.001, 0.003]$
`weight_decay`	$[0.03, 0.1]$	$[0.1, 0.3]$	$[0.1, 0.03]$
`warmup_steps`	100	$[1000, 3000]$	1000

D ADDITIONAL DETAILS

D.1 COMPUTE RESOURCES

We used a Linux workstation with two Nvidia RTX 3090 GPUs with 24GB of memory each for development and conducted hyperparameter searches and experiments using 1 Linux server with 4 Nvidia RTX 3090 GPUs as well as a Slurm cluster equipped with Nvidia RTX 4090 GPUs. With a single GPU of these two GPU models, a single run of the fuzzy logic task takes around $2-3$ minutes, a single run on SRAVEN between $20-200$ minutes. For the language modeling experiments, we used 16 Cloud TPU v5e with a complete run taking 72-100 hours. In total, it takes around 24 GPU hours to reproduce all fuzzy logic results on our hardware, around 10 GPU days to reproduce all our SRAVEN results and approximately 163 TPU days to reproduce our results on C4.

D.2 SOFTWARE AND LIBRARIES

For the results obtained in this paper we built on free and open-source software. We implemented our experiments in Python using JAX (Bradbury et al., 2018, Apache License 2.0), Flax (Heek et al., 2023, Apache License 2.0), NanoDo (Liu et al., 2024, Apache License 2.0) and the Deepmind Jax Ecosystem (Babuschkin et al., 2020, Apache License 2.0). We utilized WandB (Biewald, 2020, MIT license) to monitor the progress and results of experiments, and Plotly (Inc, 2015, MIT license) for generating the plots.