Neural Attentive Circuits

Martin Weiss[*,1,6] **Nasim Rahaman**[*,1,2] **Francesco Locatello**[3] **Chris Pal** [1,6,7]
Yoshua Bengio[1,5,7] **Bernhard Schölkopf**[2] **Nicolas Ballas** [†,4] **Erran Li**[†,3]

[*,†] Equal contribution, random order.

[1]Mila, Quebec AI Institute [2] Max Planck Institute for Intelligent Systems, Tübingen
[3] AWS AI [4] Meta AI [5] Université de Montréal [6] Polytechnique Montréal
[7] Canada CIFAR AI Chair

Abstract

Recent work has seen the development of general purpose neural architectures that can be trained to perform tasks across diverse data modalities. General purpose models typically make few assumptions about the underlying data-structure and are known to perform well in the large-data regime. At the same time, there has been growing interest in modular neural architectures that represent the data using sparsely interacting modules. These models can be more robust out-of-distribution, computationally efficient, and capable of sample-efficient adaptation to new data. However, they tend to make domain-specific assumptions about the data, and present challenges in how module behavior (i.e., parameterization) and connectivity (i.e., their layout) can be jointly learned. In this work, we introduce a general purpose, yet modular neural architecture called Neural Attentive Circuits (NACs) that jointly learns the parameterization and a sparse connectivity of neural modules without using domain knowledge. NACs are best understood as the combination of two systems that are jointly trained end-to-end: one that determines the module configuration and the other that executes it on an input. We demonstrate qualitatively that NACs learn diverse and meaningful module configurations on the Natural Language and Visual Reasoning for Real (NLVR2) dataset without additional supervision. Quantitatively, we show that by incorporating modularity in this way, NACs improve upon a strong non-modular baseline in terms of low-shot adaptation on CIFAR and Caltech-UCSD Birds dataset (CUB) by about 10 percent, and OOD robustness on Tiny ImageNet-R by about 2.5 percent. Further, we find that NACs can achieve an 8x speedup at inference time while losing less than 3 percent performance. Finally, we find NACs to yield competitive results on diverse data modalities spanning point-cloud classification, symbolic processing and text-classification from ASCII bytes, thereby confirming its general purpose nature.

1 Introduction

General purpose neural models like Perceivers [29] do not make significant assumptions about the underlying data-structure of the input and tend to perform well in the large-data regime. This enables the application of the same model on a variety of data modalities, including images, text, audio, point-clouds, and arbitrary combinations thereof [29, 28]. This is appealing from an ease-of-use perspective, since the amount of domain-specific components is minimized, and the resulting models can function well *out-of-the-box* in larger machine learning pipelines, e.g., AlphaStar [28].

At the same time, natural data generating processes can often be well-represented by a system of sparsely interacting independent mechanisms [41, 46], and the Sparse Mechanism Shift hypothesis

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

stipulates that real-world shifts are often sparse when decomposed, i.e., that most mechanisms may remain invariant [47]. Modular architectures seek to leverage this structure by enabling the learning of systems of sparsely interacting neural modules [3, 20, 44, 45]. Such systems tend to excel in low-data regimes, systematic generalization, fast (sample-efficient) adaptation to new data and can maintain a larger degree of robustness out-of-distribution [52, 3, 4, 21]. However, many of these models make domain-specific assumptions about the data distribution and modalities [4] – e.g., Mao et al. [37] relying on visual scene graphs, or Andreas et al. [3] using hand-specified modules and behavioral cloning. As a result, such models are often less flexible and more difficult to deploy.

In this work, we propose Neural Attentive Circuits (NACs), an architecture that incorporates modular inductive biases while maintaining the versatility of general purpose models. NACs implement a system of sparsely and attentively interacting neural modules that pass messages to each other along *connectivity graphs* that are learned or dynamically inferred in the forward pass and subject to priors derived from network science [25, 6, 18]. NACs demonstrate strong low-shot adaptation performance and better out-of-distribution robustness when compared to other general purpose models such as Perceiver IOs [28]. Besides scaling linearly with input size (like Perceivers), we show that by pruning modules, the computational complexity of NACs can be reduced at inference time while preserving performance. In addition, we propose a conditional variant where the graph structure and module parameterization is conditioned on the input. This enables conditional computation [10, 9], where the functions computed by neural modules and the sparse connectivity between modules is determined only at inference time. A qualitative analysis on Natural Language and Visual Reasoning for Real (NLVR2) shows that conditional NACs can learn connectivity graphs that are meaningfully diverse.

Figure 1 shows the schematics of the proposed architecture, with its two core components: the **circuit generator** and the **circuit executor**. The circuit generator produces the configuration over modules, which we call a circuit design, defining (a) the connectivity pattern between the neural modules, and (b) instructions that condition the computation performed by each module. The circuit design may either be conditioned on (part of) the sample (in the case of conditional NACs), or it may simply be learned over the course of training via gradient descent (in the case of unconditional NACs). The circuit executor consumes the circuit design and an input sample to perform inference.

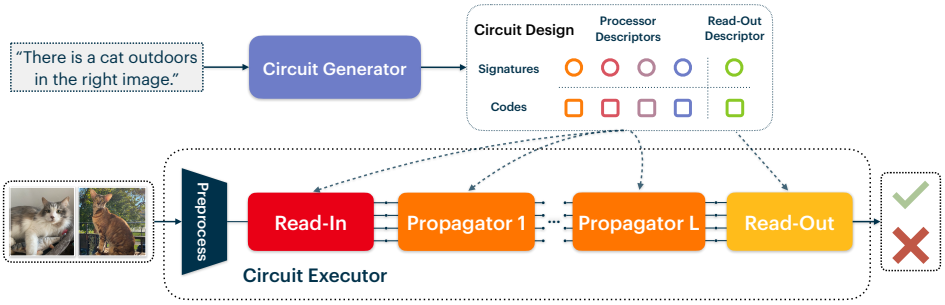


Figure 1: **Circuit Generator and Executor.** We show NAC applied to a natural language and visual reasoning task where text conditions the circuit generator and the program executor consumes images.

Our contributions are as follows. **(a)** We propose Neural Attentive Circuits (NACs), a general purpose neural architecture that jointly learns the module parameterization and connectivity end-to-end and gracefully supports more than one thousand sparsely interacting modules. **(b)** We demonstrate the general-purpose nature of NACs by training it on a selection of diverse data domains covering natural images, 3D point-clouds, symbolic processing, text understanding from raw ASCII bytes, and natural-language visual reasoning. **(c)** We quantitatively evaluate the out-of-distribution robustness and few-shot adaptation performance of NACs. We find that in the low-shot regime, NACs can outperform Perceiver IOs by approximately 10% on 8-Way CIFAR and CUB-2011. Further, the proposed model yields roughly 2.5% improvement on TinyImageNet-R, an out-of-distribution test-set for TinyImageNet based on ImageNet-R [24]. **(d)** We explore the adaptive-computation aspect of NACs, wherein the computational complexity of a trained model can be reduced at inference time by roughly 8 times, at the cost of less than 3% loss in accuracy on Tiny-ImageNet. **(e)** We qualitatively demonstrate that it is possible to condition the circuit design (i.e., configuration over modules) on the input. On NLVR2, we use the text modality of the sample to condition the circuit design, which is

then applied to the image modality of the same sample to produce a result. We find that connectivity graphs generated by sentences that involve similar reasoning skills have similar structures.

2 Neural Attentive Circuits

This section describes Neural Attentive Circuits (NACs), a neural architecture that can consume arbitrary (tokenized) set-valued inputs including images and text. NACs are a system of neural modules that learn to interact with each other (see Figure 2). Module connectivity is learned end-to-end and is subject to regularization that encourages certain patterns (e.g., sparsity, scale-freeness and formation of cliques). In the following, we first describe two key building blocks of the proposed architecture. We then introduce the circuit executor that infers an output given some input and a circuit design. We finally introduce the circuit generator which produces said circuit design.

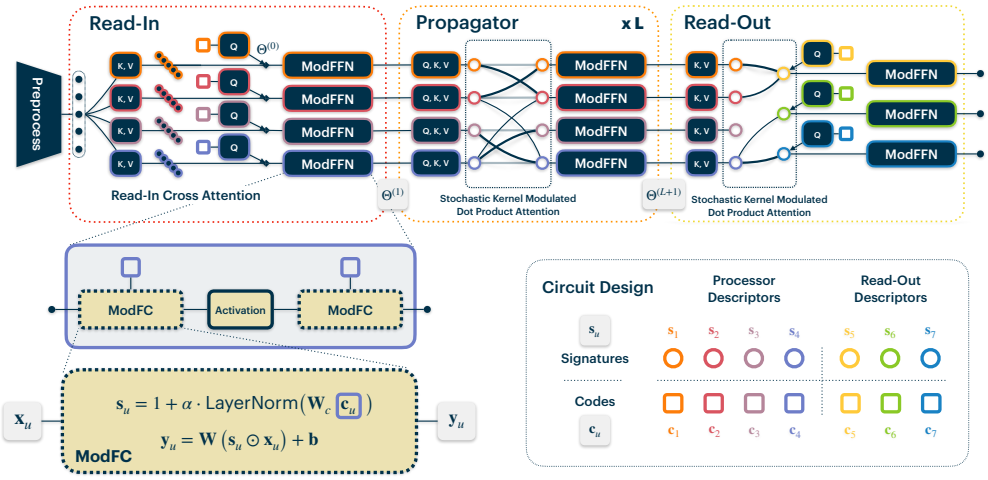


Figure 2: **Schematics of Neural Attentive Circuits.** *Top:* a NAC with $U_p = 4$ processor modules and $U_o = 3$ read-out modules. *Bottom right:* each module has a unique signature and code (colored circles and boxes). The signatures determine which modules interact and the code given to a module conditions the computation it performs (via ModFFNs and ModFCs). *Bottom left:* visual description of ModFFNs and ModFC.

Circuit Design. Consider a NAC (Figure 2) with U modules, of which U_p are *processor modules* (responsible for the bulk of the computation) and U_o are *read-out modules* (responsible for extracting model outputs). By circuit design, we refer to the set of U descriptors that condition the module connectivity and computation carried out at each module. Each descriptor comprises three vectors $(\boldsymbol{s}_u, \boldsymbol{c}_u, \boldsymbol{\theta}_u^{(0)})$: a signature vector $\boldsymbol{s}_u \in \mathbb{R}^{d_{sig}}$, a code vector $\boldsymbol{c}_u \in \mathbb{R}^{d_{code}}$ and a module initial state $\boldsymbol{\theta}_u^{(0)} \in \mathbb{R}^{d_{model}}$. The set of signature vectors $\{\boldsymbol{s}_u\}_u$ determines the connectivity between modules while the code vector $\boldsymbol{c}_u$ conditions the computation performed by module at index u. Further, the vector $\boldsymbol{\theta}_u^{(l)} \in \mathbb{R}^{d_{model}}$ tracks the state of the module u before the l-th layer, and by $\Theta^{(l)}$ we denote the set of all such states (over modules). Each module updates its state as the computation progresses in depth (see below). Finally, processor module signatures and codes are shared across depth.

ModFC and ModFFN. The Modulated Fully-Connected (ModFC) layer is the component that enables a module to condition its computation on a code. It replaces the fully-connected layer and supports a multiplicative conditioning of the input vector $\boldsymbol{x}$ by the *code vector* $\boldsymbol{c}$. It can be expressed

$$\boldsymbol{y} = \text{ModFC}(\boldsymbol{x}; \boldsymbol{c}) = \boldsymbol{W}(\boldsymbol{x} \odot (1 + \alpha\, \text{LayerNorm}(\boldsymbol{W}_c \boldsymbol{c}))) + \boldsymbol{b}, \tag{1}$$

where $\odot$ denotes element-wise multiplication, $\boldsymbol{W} \in \mathbb{R}^{d_{out}} \times \mathbb{R}^{d_{in}}$ is a weight matrix, $\boldsymbol{b} \in \mathbb{R}^{d_{out}}$ is a bias vector, $\boldsymbol{c} \in \mathbb{R}^{d_{code}}$ is the conditioning code vector, $\boldsymbol{W}_c \in \mathbb{R}^{d_{in}} \times \mathbb{R}^{d_{code}}$ is the conditioning weight matrix, and α is a learnable scalar parameter that controls the amount of conditioning. Setting $\alpha = 0$ removes all conditioning on $\boldsymbol{c}$, and we generally initialize $\alpha = 0.1$. Multiple such ModFC layers (sharing the same code vector $\boldsymbol{c}$) and activation functions can be stacked to obtain a ModFFN

layer. Like in transformers, multiple copies of ModFCs and ModFFNs are executed in parallel. But unlike in transformers, each copy is conditioned by a unique learnable code vector and therefore performs a different computation. Consequently, though the computation performed in each module is different, the total number of parameters does not noticeably increase with the number of modules. See a clarifying example in Appendix B.

SKMDPA. Stochastic Kernel Modulated Dot-Product Attention (SKMDPA) is a sparse-attention mechanism that allows the modules to communicate. Given two modules at index i and j, the distance between their signatures $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$ determines the probability with which these modules can interact via dot-product attention – smaller distance implies higher likelihood that the modules are allowed to communicate. Like vanilla dot-product attention, SKMDPA operates on a set of query $\boldsymbol{q}_i$, key $\boldsymbol{k}_j$, and value $\boldsymbol{v}_j$ vectors, where i indexes queries and j indexes keys and values. But, in addition, each query $\boldsymbol{q}_i$ and key $\boldsymbol{k}_j$ is equipped with a signature embedding vector, which may be learned as parameters or dynamically inferred. These signatures are used to sample a kernel matrix K [1] via

$$K_{ij} \sim \text{Concrete}(P_{ij}, \tau) \text{ with } P_{ij} = \exp\left[\frac{-d(\boldsymbol{s}_i, \boldsymbol{s}_j)}{\epsilon}\right], \tag{2}$$

where d is some distance metric, ϵ is a bandwidth (a hyper-parameter), Concrete is the continuous relaxation of the Bernoulli distribution [36] with sampling temperature τ, and the sampling operation $\sim$ is differentiable via the reparameterization trick. K_{ij} is likely to be close to 1 if the distance between s_i and s_j is small, and close to 0 if not. As $\tau \to 0$, the kernel matrix K_{ij} becomes populated with either 1s or 0s. Further, if P_{ij} is ensured to be sufficiently small on average, K_{ij} is a sparse matrix. In practice, we set τ to a small but non-zero value (e.g. 0.5) to allow for exploration. Now, where $A_{i \leftarrow j}$ is the dot-product attention score $\boldsymbol{q}_i \cdot \boldsymbol{k}_j / \sqrt{d}$ between the query $\boldsymbol{q}_i$ and key $\boldsymbol{k}_j$, the net attention weight $W_{i \leftarrow j}$ and the output $\boldsymbol{y}_i$ are

$$W_{i \leftarrow j} = \text{softmax}_j\left[A_{i \leftarrow j} + \log \hat{K}_{ij}\right] \text{ where } \hat{K}_{ij} = K_{ij} / (\delta + \textstyle\sum_j K_{ij}) \text{ and } \boldsymbol{y}_i = \sum_j W_{i \leftarrow j} \boldsymbol{v}_j. \tag{3}$$

Here, $W_{i \leftarrow j}$ denotes the weight of the *message* $\boldsymbol{v}_j$ passed by element j to element i, and δ is a small scalar for numerical stability. We note that if $K_{ij} \approx 0$, we have $W_{i \leftarrow j} \approx 0$, implying if K_{ij} is sparse, so is $W_{i \leftarrow j}$. The proposed attention mechanism effectively allows the query $\boldsymbol{q}_i$ to interact with the key $\boldsymbol{k}_j$ with a probability that depends on the distance between their respective signatures. In this work, this distance is derived from the cosine similarity, i.e., $d(\boldsymbol{s}_i, \boldsymbol{s}_j) = 1 - \text{CosineSimilarity}(\boldsymbol{s}_i, \boldsymbol{s}_j)$. We note the connection to Lin et al. [34], where the attentive interactions between queries and keys are dropped at random, albeit with a probability that is not differentiably learned.

2.1 Circuit Executor

The circuit executor takes the circuit design and executes it on an input to make a prediction. It has four components: **(a)** a tokenizer to convert the input (e.g., images or text) to a set of vectors, **(b)** a *read-in* layer which allows the processor modules to attentively read from the inputs, **(c)** a sequence of *propagator layers* which iteratively update the processor modules states through a round of communication and computation, **(d)** *read-out* layers that enable the read-out modules to attentively poll the final outputs of the processor modules.

Tokenizer. The tokenizer is any component that converts the input to the overall model to a set of representation vectors (with positional encodings, where appropriate), called the input set X. It can be a learned neural network (e.g., the first few layers of a ConvNet), or a hard-coded patch extractor. *Role:* The tokenizer standardizes the inputs to the model by converting the said inputs to a set of vectors.

Read-In. The read-in component is a read-in attention followed by a ModFFN. Read-in attention is a cross-attention that uses the initial processor module states $\Theta_p^{(0)}$ to generate a query vector and uses the inputs X to generate keys and values. Unlike typical cross-attention [32, 29], we use ModFCs instead of FCs as query, key and value projectors. Where $\boldsymbol{x}_j$ is an element of X, we have:

$$\boldsymbol{q}_u = \text{ModFC}(\boldsymbol{\theta}_u^{(0)}, \boldsymbol{c}_u) \qquad \boldsymbol{k}_{uj} = \text{ModFC}(\boldsymbol{x}_j, \boldsymbol{c}_u) \qquad \boldsymbol{v}_{uj} = \text{ModFC}(\boldsymbol{x}_j, \boldsymbol{c}_u) \tag{4}$$

$$\hat{\boldsymbol{y}}_{\boldsymbol{u}} = \sum_j \frac{\boldsymbol{q}_u \cdot \boldsymbol{k}_{uj}}{\sqrt{d}} \boldsymbol{v}_{uj} \qquad \boldsymbol{y}_u = \text{ModFC}(\hat{\boldsymbol{y}}_{\boldsymbol{u}}, \boldsymbol{c}_u) \qquad \boldsymbol{\theta}_u^{(1)} = \text{ModFFN}(\boldsymbol{y}_u, \boldsymbol{c}_u) \tag{5}$$

Each query $\boldsymbol{\theta}_u^{(0)}$ sees its own set of keys $\boldsymbol{k}_{uj}$ and values $\boldsymbol{v}_{uj}$, owing to the conditioning on the code $\boldsymbol{c}_u$. The read-in attention is followed by U_p copies of a two layer ModFFN each conditioned on a code $\boldsymbol{c}_u$. The read-in outputs a set of vectors $\boldsymbol{\theta}_u^{(1)} \in \Theta_p^{(1)}$. *Role:* The read-in attends to the inputs via a cross-attention mechanism that scales linearly with the size of the input.

Propagator Layers. After the read-in operation, propagator layers sequentially update the state of each processor module L times. Each propagator layer implements a round of module communication via SKMDPA, followed by the computation of U_p copies of a ModFFN. Both operations are conditioned on their processor module descriptor codes. The l-th propagator (where $l = 1, \ldots, L$) ingests the set of states $\Theta_p^{(l)}$ and outputs another set $\Theta_p^{(l+1)}$ of the same size. *Role:* A propagator layer updates each processor module state vector by applying learned stochastic attention and module computation.

Read-Out. The read-out component is a SKMDPA-based cross-attention mechanism (read-out attention) followed by a ModFFN, and it is responsible for extracting the output from the processor module states after the final propagator layer. Read-out modules have their own signatures and codes parameters which are different from the processor modules. The read-out attention uses the initial states $\Theta_o^{(0)}$ of these modules as queries, and the final states of the processor modules $\Theta_p^{(L+1)}$ to obtain keys and values. Finally, we note that in vanilla classification problems, we found it useful to use several read-out modules. Each read-out module produces a confidence score, which is used to weight their respective additive contribution to the final output (e.g., classification logits). *Role:* The read-out is responsible for attending to the final processor module states to produce an overall output.

2.2 Circuit Generator

Recall that the circuit generator produces a circuit design that specifies the connectivity between the modules (via signatures) and the computation performed by each module (via codes). There are two ways in which the circuit generator can be instantiated, as described below.

Unconditional Circuit Generator. In the first variant (unconditional), the circuit design is induced by signatures $\boldsymbol{s}_u$ and codes $\boldsymbol{c}_u$ that are freely learnable parameters (where u indexes the module descriptor). The initial state $\boldsymbol{\theta}_u^{(0)}$ is obtained passing $\boldsymbol{c}_u$ through a two layer MLP. We refer to the overall model as an unconditional NAC, or simply a NAC. *Role:* The unconditional circuit generator implements a mechanism to learn the circuit design via free parameters with gradient descent.

Regularization via Graph Priors. In the unconditional setting, we observe a task-driven pressure that collapsed all signatures $\boldsymbol{s}_u$ to the same value (see Appendix A). This results in a graph that is fully connected, i.e., all modules interact with all other modules, thereby forgoing the benefits of sparsely interacting modules. To counteract this pressure, we impose a structural prior on the learned graph via a regularization term in the objective, which we call the Graph Structure Regularizer. The graph prior is expressed as a target link probability matrix and the extent to which the prior is satisfied is obtained comparing the closest permutation of P_{ij} (the generated link probability matrix) to the target values. This regularizer encourages the graph to have certain chosen properties, while still preserving the flexibility afforded by learning the graph structure. We experiment with several such properties, including scale-freeness [6] (leading to a few well connected nodes or *hubs*, and a heavy tail of less connected nodes), and a planted partition [15] one based on the stochastic block model [25] (encouraging modules to group themselves in *cliques*, where communication within cliques is less constrained than that between cliques). In Section 4, we will find that some graph properties do indeed yield better out-of-distribution performance, whereas others perform better in-distribution. Figure 3 shows the class of graphs we experiment with in this work, and Appendix A contains the details. *Role:* The Graph Structure Regularizer enforces sparsity by preventing the module connectivity graph from collapsing to an all-to-all connected graph, while inducing graph properties that may unlock additional performance depending on the task.

Conditional Circuit Generator. In the second variant (conditional), each sample is assigned a different circuit design. In this case, the circuit generator takes as input the sample X (or a part thereof), and outputs a set of signatures and codes $\{\boldsymbol{s}_u(X), \boldsymbol{c}_u(X)\}_u$. This setting is particularly interesting for multi-modal data, where the circuit generator may ingest one modality and the circuit executor may consume another. In this work, we implement the conditional circuit generator as a simple cross attention mechanism, where learned parameters are used as queries (one per module) and elements of X as keys and values [32]. We refer to the resulting model as conditional NACs,

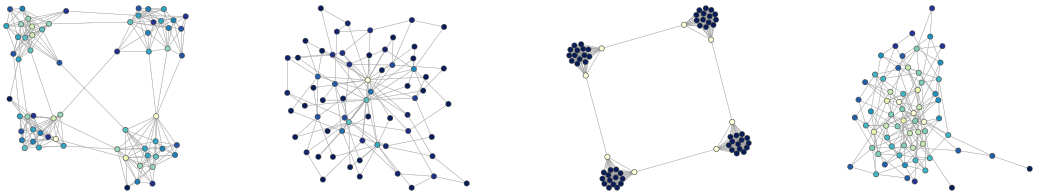


Figure 3: Samples from **Graph priors** explored here. Node colors represent degree (# of edges). **Left**: Planted-partition [15, 25]. **Center-left**: Scale-free [7, 6]. **Center-right**: Ring-of-Cliques [38]. **Right**: Erdos-Renyi [17].

while noting that the circuit generator could in principle be implemented by other means (e.g., auto-regressive models and GFLowNets [12]) in future work. *Role:* The conditional circuit generator relies on the sample when producing a circuit design. This way, the connectivity between modules and the computation performed by each module can be dynamically changed for each sample.

In summary, we have introduced Neural Attentive Circuits (NACs) and its conditional variant, a general purpose neural architecture of attentively interacting modules. We discussed the two key components of NACs: **(a)** a circuit generator that *generates* a circuit design determining which modules connect to which others (via signatures) and the function performed by each (via codes), and **(b)** a circuit executor that *executes* the configuration on an input to infer an output.

3 Related work

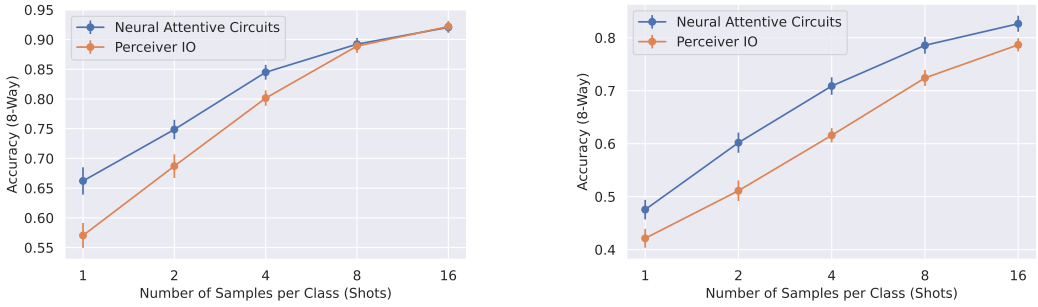
General Purpose Models. A recent line of work develops *general-purpose models* that make few assumptions about the data domain they are applied to [29, 28, 23]. Of these models, Perceiver IO [28] is most similar to NACs, in that **(a)** they both rely on a cross-attention to both read (tokenized) set-valued inputs into the model and extract set-valued output from the model, and **(b)** their computational complexity scales linearly with the input size. However, unlike Perceiver IO (PIO), **(a)** the connectivity between NAC modules is not all-to-all, but sparse in a learned way, and **(b)** the computation performed by NAC's equivalent of a PIO latent is conditioned at all layers via the ModFC component.

Modular Inductive Biases. Modular neural architectures are known to yield models that are capable of systematic generalization and fast adaptation [5, 2]. Of these architectures, Neural Interpreters (NIs) are most similar to NACs, in that they both use the framework of signatures and codes, where signatures determine the computational pathway and codes condition individual modules. However, there are four key differences: **(a)** unlike in NIs, the routing of information in NACs is stochastic, **(b)** the mechanism to condition on codes (ModFC) is different, and **(c)** the computational complexity and memory requirements of NIs scales quadratically with the size of the input, whereas in NACs, this scaling is linear, and **(d)** NIs are only known to work with up to 8 modules, whereas NACs can support more than a thousand modules on a single GPU (i.e., without model parallelism).

Neuro-symbolic Models. Neuro-symbolic models have shown strong systematic generalization on synthetic datasets such as CLEVR [30] and SQOOP [5]. However, models of this class such as the Neuro-Symbolic Concept Learner [37] and Neural Module Networks [3] are not general-purpose, using semantic parsers and domain-specific modules. In contrast, NACs are general-purpose and can jointly learn a sparse layout and parameterization for more than one thousand homogeneous modules.

4 Experiments

In this section, we empirically investigate Neural Attentive Circuits (NACs) in five problem settings with different modalities: image classification, point cloud classification, symbolic processing (ListOps [49]), text classification from raw bytes [49], and multi-modal reasoning over natural language and images [48]. We compare against baselines from the literature, as well as those that we train under controlled settings. The questions we ask are the following: **(a)** Does the inductive bias of sparsely interacting modules improve NACs out-of-distribution robustness and fast (few-shot) adaptation to new data? **(b)** How important is it that these modules be capable of modelling different functions (via ModFC layers), and that the connectivity between modules is learned (via SKMDPA)? **(c)** Do the modules strongly co-adapt to each other, or can the system still function with some modules



(a) CUB 8-Way

(b) CIFAR 8-Way

Figure 4: Few-Shot Adaptation Results on CUB and CIFAR. On each dataset, we fine-tune ImageNet pretrained NAC and PIO models to classify images into 8 classes and vary the number of examples ("shots") per class from 1 to 16. In the low-shot transfer regime (e.g., 1-4 samples per class), we find that NAC compares favorably with the baseline PIO, suggesting that the modular inductive biases in the former can aid fast adaptation.

pruned at inference time? **(d)** In the sample-conditional setting, are there interesting patterns in the generated circuit designs? **(e)** Are NACs general-purpose?

4.1 Image Classification

In this section, we conduct a quantitative evaluation of NACs on image classification problems. We use Tiny-ImageNet as our primary test-bed. Tiny-ImageNet is a miniaturized version of ImageNet[16], with smaller dataset size and lower image resolution. It allows for compute efficient experimentation, but makes the overall classification task significantly more challenging, especially for larger models that can easily overfit [33]. To evaluate out-of-distribution (OOD) robustness in this regime, we use Tiny-ImageNet-R, which is a down-sampled subset of ImageNet-R(enditions) [24]. It contains roughly 12000 samples categorized in 64 classes (a subset of Tiny-ImageNet classes), spread across multiple visual domains such as art, cartoons, sculptures, origami, graffiti, embroidery, etc. For low-shot adaptation experiments, we also train a NAC on ImageNet, in order to compare with a pretrained baseline. Additional details can be found in Appendix C.

Baselines. For a certain choice of hyper-parameters, NACs can simulate Perceiver IOs (PIOs) [28], making the latter the natural baseline for NACs. For all Tiny-ImageNet runs for both NACs and PIOs, we use the same convolutional preprocessor that reduces the 64×64 input image to a 8×8 feature map. For ImageNet training, we use the same convolutional preprocessors as we do for Tiny-ImageNet. Please refer to Appendix C for additional details.

Pre-training. We pretrain all models on Tiny-ImageNet for 400 epochs, and perform model selection based on accuracy on the corresponding validation set. We evaluate the selected models on Tiny-ImageNet-R. For few-shot adaptation experiments, we use the official ImageNet pretrained weights for a 48-layer deep PIO [28], which yields 82.1% validation accuracy. To compare few-shot adaptation performance with this model, we train a 8-layer deep NAC with a scale-free prior graph on full ImageNet for 110 epochs, which yields 77% validation accuracy with 1024 modules. Additional details and hyperparameters can be found in Appendix C.

Few-Shot Adaptation. Bengio et al. [11] proposed that models that learn a good modularization of the underlying data generative process should adapt in a sample-efficient manner to unseen but related data-distributions. Motivated by this reasoning, we measure NACs few-shot adaptation performance and compare with a non-modular PIO baseline. To this end, we fine-tune ImageNet pre-trained NACs and PIOs on small numbers of samples – between 8 and 128 in total – from two different datasets: CUB-2011 (Birds) and CIFAR. The results shown in Figure 4 support the hypothesis that the modular inductive biases in NACs help with sample-efficient adaptation, and we show the few-shot experiments with varying numbers of classes in the Appendix with similar results.

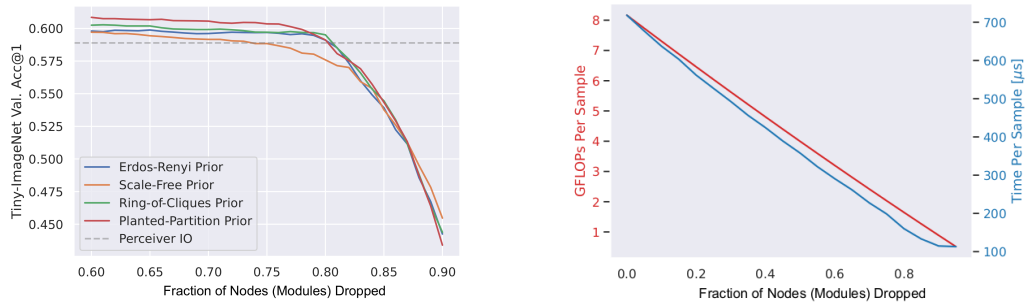
Ablations. In this experiment (Table 1), we start with a Perceiver IO and cumulatively add design elements to arrive at the NAC model. We evaluate the IID and OOD performances and make two key observations. **(a)** We find that adding ModFC layers does not significantly affect IID generalization, but improves OOD generalization. Moreover, using a static Barabasi-Albert attention graph (via frozen signatures) reduces the IID performance while further improving OOD performance. Jointly

learning the connectivity graph and module codes yields the largest improvements. Learning the graph allows to recover the IID performance, and the OOD performance is further improved. **(b)** The choice of a graph prior exposes a trade-off between IID and OOD performance. The scale-free prior yields the best IID performance, whereas the ring-of-cliques prior yields the best OOD performance.

	IID (Tiny-ImageNet Validation)		**OOD** (Tiny-ImageNet-R)	
	Acc@1	Acc@5	Acc@1	Acc@5
Perceiver IO	58.89	80.09	17.57	37.16
+ ModFC	58.91	79.43	18.01	37.05
+ Sparse graph (not learnable)	58.31	80.06	18.50	37.92
NAC with Scale-Free Prior	**60.76**	80.86	19.52	38.52
NAC with Planted-Partition Prior	60.71	**81.34**	19.42	39.84
NAC with Ring-of-Cliques Prior	60.54	81.18	**20.03**	**40.44**
NAC with Erdos-Renyi Prior	60.33	81.08	19.83	39.32

Table 1: **Ablation and Graph Results.** In this experiment, we start with a Perceiver IO and cumulatively add design elements to arrive at the NAC model. At each stage, we train and observe the effect on generalization. We find that adding ModFC layers alone does not have a clear impact on either IID or OOD generalization. However, OOD is then clearly improved by adding a sparse kernel sampled from a frozen prior distribution. Below the midrule, we evaluate NACs performance with several different graph prior regularizers. We see that learning the graph via SKMDPA allows the model to improve both its IID and OOD performance.

Effect of Dropping Modules. In this experiment, we seek to measure the extent to which the NACs modules depend on other modules to function. To this end, we drop modules in order of their connectivity (i.e., least connected modules are dropped first) and measure the performance of the model on Tiny-ImageNet along the way. We perform this experiment for models trained with different types of graph priors, and show the result in Figure 5. We make two observations: **(a)** it is possible to remove a large fraction (e.g., more than 80%) of the modules at little cost to accuracy, obtaining a $\sim 8\times$ speedup in the process, and **(b)** different graph priors offer different amounts of robustness towards dropped modules. The former observation is particularly interesting from a computational efficiency perspective (Figure 5b), since dropping modules increases throughput and decreases memory requirements at inference-time without additional implementation overhead. The latter observation suggests a way in which the choice of a graph prior can affect the properties of the NACs model. We provide additional details in Appendix F, where Figure 15 compares the effect of sparsification on NACs (with different graph priors) and Perceiver IO.



(a) Validation accuracy as modules are dropped.

(b) Inference speed as modules are dropped.

Figure 5: **Adaptive Computation**. We train models to convergence and then sparsify them at inference time and evaluate their validation accuracy. In both plots, the x-axis shows the number of modules dropped at inference time, and the Y-axis shows the accuracy of the model on Tiny-ImageNet after sparsification at inference time. We observe that the Neural Attentive Circuit (NAC) is much more robust than Perceiver IO to sparsification at inference time, and can drop over 80% of its circuits before its performance drops below Perceiver IO. As a result, NACs can be adapted at inference time to a wide range of computational budgets.

4.2 Point Cloud Classification

We evaluate the NAC architecture on a point cloud classification benchmark without additional data augmentation, geometric inductive biases or self-supervised pre-training [14]. Comparably, Perceiver IO obtained 77.4%, Hierarchical Perceivers achieved 75.9% without self-supervised pre-training, and when trained with self-supervised pre-training Hierarchical Perceivers obtained 80.6% (cf. Table 7 of [14]). We trained a NAC without specialized tokenization or pre-training to achieve a test accuracy of 83%, demonstrating that the sparse inductive prior was quite effective in this setting. The original Perceiver [29] relies on additional data augmentation beyond that used in [14] to obtain 85% test accuracy. We do not know how well the original Perceiver would perform without the additional augmentations, but expect it to be similar to Perceiver IO.

Method	Test Accuracy
Hierarchical Perceiver (No MAE) [14]	75.9%
Perceiver IO [14]	77.4%
Hierarchical Perceiver (with MAE) [14]	80.6%
NAC (ours) (No MAE)	**83.0%**

Table 2: **Point Cloud Classification.** In this experiment, we compare against results reported in Carreira et al. [14]. In that work, the authors were able to improve over Perceiver IO results through a masked auto-encoding (MAE) pre-training step. We find that, without pre-training, NACs with a Scale-Free sparse prior graph are able to achieve superior test accuracy.

4.3 Symbolic Processing on ListOps and Text Classification from Raw Bytes

In this set of experiments, we evaluate the ability of NACs to reason over raw, low-level inputs. We use two tasks to this end: a symbolic processing task (ListOps) and a text-classification task from raw ASCII bytes. The former (ListOps) is a challenging 10-way classification task on sequences of up to 2,000 characters [39]. This commonly used benchmark [49] tests the ability of models to reason hierarchically while handling long contexts. The latter is a binary sentiment classification task defined on sequences of up to 4000 bytes. In particular, we stress that the models ingest raw ASCII bytes, and **not** tokenized text as common in most NLP benchmarks [49]. This tests the ability of NACs to handle long streams of low-level and unprocessed data.

The results are presented in Table 3. We find that NACs are competitive, both against a general-purpose baseline (our implementation of Perceiver IO) and other third-party baselines, including full-attention transformers and the Linformer. This confirms that NACs are indeed general purpose, and that general-purpose architectures can be effective for handling such data.

4.4 Sample-Conditional Circuit Design Generation

In the previous sections, we quantitatively analyzed the performance of a NAC model equipped with a connectivity graph that is learned, but fixed at inference time. In this section, we investigate sample-conditional circuit generation without regularizing the graph structure. Concretely, we study the circuit designs generated by a conditional NAC trained to perform a multi-modal reasoning task. In our experiments with conditional NACs, the circuit generator was implemented by a simple cross attention layer followed by a feed forward network (FFN), and a self-attention layer followed by

	Test Accuracy	
Method	ListOps [39]	Text Classification [49]
Full Attention Transformers [49]	37.13%	65.35%
Linformer [49]	37.38%	56.12%
Perceiver IO (our impl.)	39.70%	66.50%
NAC (ours)	**41.40%**	**68.18%**

Table 3: **ListOps and Text Classification Results.** In this experiment, we train a Perceiver IO and a NAC from scratch on two tasks from the the Long Range Arena [49]: ListOps symbolic processing and text classification from ASCII bytes. We observe that NAC excels in processing long-range dependencies in this setting.

two FFNs run in parallel, see Appendix G for further details. We use Natural Language for Visual Reasoning *for Real* (NLVR2) dataset [48] which contains natural language sentences grounded in images. The task is to determine whether a natural language statement holds true for a given pair of photos. Solving this task requires reasoning about sets of objects, making comparisons, and determining spatial relations. Accordingly, we would expect that the circuit generator creates circuit designs that are qualitatively different for different logical relations.

Training. We condition the circuit generator on natural language captions, and the executor outputs a binary label conditioned both on the image pairs and on the output of the circuit generator. Given that our goal of understanding how conditional NACs perform reasoning, we use a (frozen) pre-trained CLIP [42] backbone to pre-process both texts and images. We select CLIP because it allows us to measure zero-shot performance out-of-the-box (53% validation accuracy, details in Appendix G), and observe that conditional NACs improve on it (obtaining approximately 64% validation accuracy).

Analysis. We select a trained model and analyze the behaviour of its circuit generator. In Figure 6, we visualize the 2D embeddings of the link probability matrix P (see Equation 2), as well as a selection of inferred connectivity graphs for simple (non-compound) natural language statements that capture a subset of the semantic phenomena found in Suhr et al. [48] (see Appendix G). To obtain the former, we flatten the lower-triangular part of the link probability matrix P_{ij} into a vector. We first reduce this $\sim$50k-dimensional vector to 64 dimensions via PCA [31]. Subsequently, TSNE [51] is used to embed the 64 dimensional vector in 2 dimensions. To obtain the plots of module connectivity graphs, we use a force-based graph visualization subroutine in the network analysis library NetworkX [22], wherein the attractive force between two modules is inversely proportional to the distance of their respective signatures. We draw an edge between modules if they are more than 50% likely to be connected with each other, and the module colors are consistent over all shown graphs. **Key observations: (a)** there is diversity in the generated graph in the conditional setting, even though we do not use a regularizing prior like we did in the image classification task; **(b)** different sentence types are assigned to different graph structures. Further, the generated graphs share a similar structure with two large cliques, with nodes that bridge them.

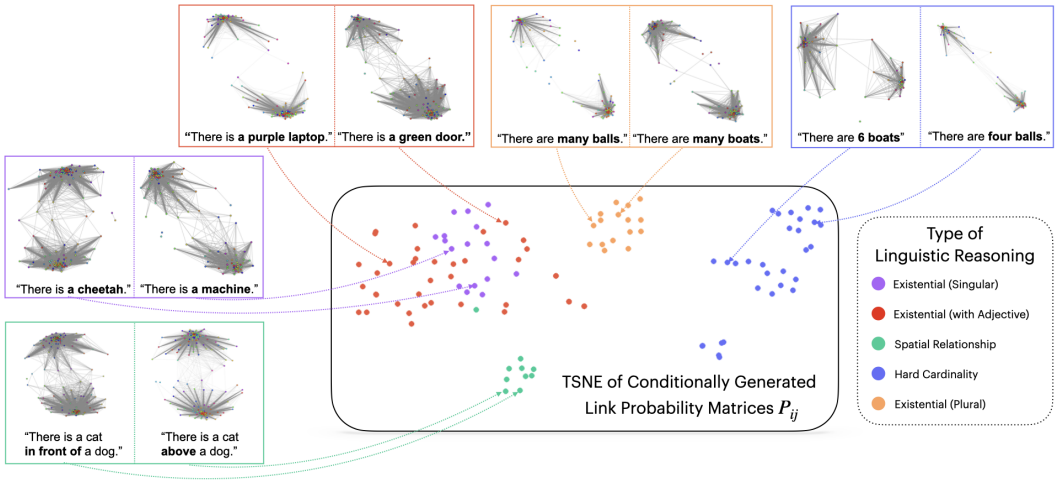


Figure 6: **Link Probability Matrix Embeddings**. This figure shows a TSNE [51] embedding of the generated graphs provided different kinds of text input. We color-code text inputs based on the type of reasoning that it requires; here we show *hard cardinality* which includes a precise number of objects, *soft cardinality* which includes words like "many" or "few", *existential* which only indicates that an object is present, and spatial relationships which use a prepositional phrase to denote a physical relationship. We see by the clustering that the low-dimensional embedding space of the generated graph captures aspects of the reasoning task. We also note that all learned graphs are similarly structured with two large cliques, but the specific arrangement varies widely.

In conclusion, we have shown a framework in Neural Attentive Circuits (NACs) for constructing a general-purpose modular neural architecture that can jointly learn meaningful module parameterizations and connectivity graphs. We have demonstrated empirically that this modular approach yields improvements in few-shot adaptation, computational efficiency, and OOD robustness.

Acknowledgements. This work was partially conducted while Nasim Rahaman was interning at Meta AI. Chris Pal and Martin Weiss thank NSERC for support under the COHESA program and IVADO for support under their AI, Biodiversity and Climate Change Thematic Program. Nasim Rahaman and Bernhard Schölkopf was supported by the German Federal Ministry of Education and Research (BMBF): Tübingen AI Center, FKZ: 01IS18039B, and by the Machine Learning Cluster of Excellence, EXC number 2064/1 - Project number 390727645.

References

[1] D. Achlioptas, F. McSherry, and B. Schölkopf. Sampling techniques for kernel methods. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 335–342, Cambridge, MA, USA, 2002. MIT Press.

[2] Ferran Alet, Erica Weng, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Neural relational inference with fast modular meta-learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/b294504229c668e750dfcc4ea9617f0a-Paper.pdf`.

[3] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Deep compositional question answering with neural module networks. *CoRR*, abs/1511.02799, 2015. URL `http://arxiv.org/abs/1511.02799`.

[4] Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron C. Courville. Systematic generalization: What is required and can it be learned? *CoRR*, abs/1811.12889, 2018. URL `http://arxiv.org/abs/1811.12889`.

[5] Dzmitry Bahdanau, Harm de Vries, Timothy J. O'Donnell, Shikhar Murty, Philippe Beaudoin, Yoshua Bengio, and Aaron C. Courville. CLOSURE: assessing systematic generalization of CLEVR models. *CoRR*, abs/1912.05783, 2019. URL `http://arxiv.org/abs/1912.05783`.

[6] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[7] Albert-László Barabási and E. Bonabeau. Scale-free networks. *Scientific American*, 288 (60-69), 2003. URL `http://www.nd.edu/~networks/PDF/Scale-Free%20Sci%20Amer%20May03.pdf`.

[8] Robert G. Bartle. *The Elements of Integration and Lebesgue Measure*. John Wiley & Sons, New York, 1995.

[9] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.

[10] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[11] Yoshua Bengio, Tristan Deleu, Nasim Rahaman, Nan Rosemary Ke, Sebastien Lachapelle, Olexa Bilaniuk, Anirudh Goyal, and Christopher Pal. A meta-transfer objective for learning to disentangle causal mechanisms. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=ryxWIgBFPS`.

[12] Yoshua Bengio, Tristan Deleu, Edward J. Hu, Salem Lahlou, Mo Tiwari, and Emmanuel Bengio. Gflownet foundations. *CoRR*, abs/2111.09266, 2021. URL `https://arxiv.org/abs/2111.09266`.

[13] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers, 2020. URL `https://arxiv.org/abs/2005.12872`.

[14] Joao Carreira, Skanda Koppula, Daniel Zoran, Adria Recasens, Catalin Ionescu, Olivier Henaff, Evan Shelhamer, Relja Arandjelovic, Matt Botvinick, Oriol Vinyals, Karen Simonyan, Andrew Zisserman, and Andrew Jaegle. Hierarchical perceiver, 2022. URL `https://arxiv.org/abs/2202.10890`.

[15] Anne Condon and Richard M. Karp. Algorithms for graph partitioning on the planted partition model. 18:116–140, 1999.

[16] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[17] P. Erdös and A. Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290, 1959.

[18] Paul Erdős, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[19] Daniel Glasscock. What is a graphon? 2016. doi: 10.48550/ARXIV.1611.00718. URL `https://arxiv.org/abs/1611.00718`.

[20] Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. *CoRR*, abs/1909.10893, 2019. URL `http://arxiv.org/abs/1909.10893`.

[21] Nitish Gupta, Kevin Lin, Dan Roth, Sameer Singh, and Matt Gardner. Neural module networks for reasoning over text. *CoRR*, abs/1912.04971, 2019. URL `http://arxiv.org/abs/1912.04971`.

[22] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[23] Curtis Hawthorne, Andrew Jaegle, Cătălina Cangea, Sebastian Borgeaud, Charlie Nash, Mateusz Malinowski, Sander Dieleman, Oriol Vinyals, Matthew Botvinick, Ian Simon, Hannah Sheahan, Neil Zeghidour, Jean-Baptiste Alayrac, João Carreira, and Jesse Engel. General-purpose, long-context autoregressive modeling with perceiver ar, 2022. URL `https://arxiv.org/abs/2202.07765`.

[24] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, Dawn Song, Jacob Steinhardt, and Justin Gilmer. The many faces of robustness: A critical analysis of out-of-distribution generalization. *CoRR*, abs/2006.16241, 2020. URL `https://arxiv.org/abs/2006.16241`.

[25] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

[26] Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[27] Drew A Hudson and Christopher D Manning. Compositional attention networks for machine reasoning. In *International Conference on Learning Representations*, 2018.

[28] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, Olivier Hénaff, Matthew M. Botvinick, Andrew Zisserman, Oriol Vinyals, and João Carreira. Perceiver io: A general architecture for structured inputs and outputs, 2021. URL `https://arxiv.org/abs/2107.14795`.

[29] Andrew Jaegle, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and João Carreira. Perceiver: General perception with iterative attention. *CoRR*, abs/2103.03206, 2021. URL `https://arxiv.org/abs/2103.03206`.

[30] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross B. Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. *CoRR*, abs/1612.06890, 2016. URL `http://arxiv.org/abs/1612.06890`.

[31] I.T. Jolliffe. *Principal Component Analysis*. Springer Verlag, 1986.

[32] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer. *CoRR*, abs/1810.00825, 2018. URL http://arxiv.org/abs/1810.00825.

[33] Seung Hoon Lee, Seunghyun Lee, and Byung Cheol Song. Vision transformer for small-size datasets, 2021. URL https://arxiv.org/abs/2112.13492.

[34] Zehui Lin, Pengfei Liu, Luyao Huang, Junkun Chen, Xipeng Qiu, and Xuanjing Huang. Dropattention: A regularization method for fully-connected self-attention networks. *CoRR*, abs/1907.11065, 2019. URL http://arxiv.org/abs/1907.11065.

[35] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *arXiv preprint arXiv:2201.03545*, 2022.

[36] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *CoRR*, abs/1611.00712, 2016. URL http://arxiv.org/abs/1611.00712.

[37] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *CoRR*, abs/1904.12584, 2019. URL http://arxiv.org/abs/1904.12584.

[38] Atsushi Miyauchi and Yasushi Kawase. Ring of cliques network. 1 2016. doi: 10.1371/journal.pone.0147805.g002. URL https://plos.figshare.com/articles/figure/_Ring_of_cliques_network_/1642440.

[39] Nikita Nangia and Samuel R. Bowman. Listops: A diagnostic dataset for latent tree learning, 2018. URL https://arxiv.org/abs/1804.06028.

[40] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. Film: Visual reasoning with a general conditioning layer. *CoRR*, abs/1709.07871, 2017. URL http://arxiv.org/abs/1709.07871.

[41] Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of Causal Inference: Foundations and Learning Algorithms*. The MIT Press, 2017. ISBN 0262037319.

[42] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. *CoRR*, abs/2103.00020, 2021. URL https://arxiv.org/abs/2103.00020.

[43] 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. *CoRR*, abs/1910.10683, 2019. URL http://arxiv.org/abs/1910.10683.

[44] Nasim Rahaman, Anirudh Goyal, Muhammad Waleed Gondal, Manuel Wuthrich, Stefan Bauer, Yash Sharma, Yoshua Bengio, and Bernhard Schölkopf. S2rms: Spatially structured recurrent modules. *CoRR*, abs/2007.06533, 2020. URL https://arxiv.org/abs/2007.06533.

[45] Nasim Rahaman, Muhammad Waleed Gondal, Shruti Joshi, Peter V. Gehler, Yoshua Bengio, Francesco Locatello, and Bernhard Schölkopf. Dynamic inference with neural interpreters. *CoRR*, abs/2110.06399, 2021. URL https://arxiv.org/abs/2110.06399.

[46] Bernhard Schölkopf. Causality for machine learning. *CoRR*, abs/1911.10500, 2019. URL http://arxiv.org/abs/1911.10500.

[47] Bernhard Schölkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Toward causal representation learning. *Proceedings of the IEEE*, 109(5):612–634, 2021.

[48] Alane Suhr, Stephanie Zhou, Iris Zhang, Huajun Bai, and Yoav Artzi. A corpus for reasoning about natural language grounded in photographs. *CoRR*, abs/1811.00491, 2018. URL http://arxiv.org/abs/1811.00491.

[49] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena : A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=qVyeW-grC2k`.

[50] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

[51] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL `http://www.jmlr.org/papers/v9/vandermaaten08a.html`.

[52] Kan Xu, Hamsa Bastani, and Osbert Bastani. Robust generalization of quadratic neural networks via function identification, 2022. URL `https://openreview.net/forum?id=Xx4MNjSmQQ9`.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Detailed discussion in the appendix.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] In the appendix.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We provide the detailed information about our model architecture and source code in the supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We provide the detailed
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Where applicable; see also supplementary material.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We have provided this information in the supplementary material.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A] We used widely tackled academic datasets with clear license information in their respective publications as cited in our work.
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Graph Priors

A.1 Graph Structure Regularizer

In this section, we describe in detail the graph structure regularizer, as introduced in Section 2.2.

Overview of the Objective. The graph structure regularizer applies on the link probability matrix P_{ij} introduced in Equation 2, and ensures that it matches a *canonical* link probability matrix (see below) of an arbitrary prior random graph $P_{ij}^{(0)}$, up to a permutation of the node labels. Where σ_* is this permutation (see below), the overall regularization objective given by

$$\mathcal{L}_{\text{graph}} = \sum_{i \neq j} \left(P_{ij} - P^{(0)}_{\sigma_*(i)\sigma_*(j)} \right)^2 . \tag{6}$$

Here we use the mean-squared error, but other distance metrics (e.g. KL divergence) are possible. Further, we exclude the diagonals (where $i = j$) because $P_{ii} = 1$ by definition, since $d(\boldsymbol{s}_i, \boldsymbol{s}_i) = 0$ in Equation 2. Given this overall objective, we now discuss how we obtain its key components: the prior $P_{ij}^{(0)}$ and the permutation σ_*.

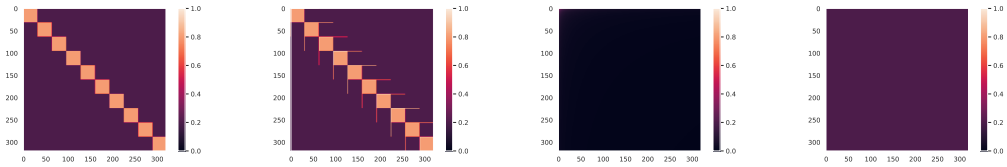
Prior Link Probabilities. Before defining the prior link probability matrix $P_{ij}^{(0)}$, we informally introduce the notion of graph functions, or *graphons* (a formal yet accessible treatment can be found in [19]). A graphon is a mathematical tool for describing certain classes of random graphs where the node labels are exchangeable, i.e. graphs where the permutation over nodes does not matter. It enables one to sample the probability that two nodes labeled i and j in a randomly sampled graph are connected, which in turn is used to sample whether or not there is an edge between the nodes i and j in the random graph.

Mathematically, a graphon $W : [0, 1] \times [0, 1] \to [0, 1]$ is a symmetric, Lebesgue measurable [8] function mapping from the unit square $[0, 1]^2$ to the unit interval $[0, 1]$. To sample a random graph with U nodes, we first draw U samples from the uniform distribution on the unit-interval $\mathcal{U}([0, 1])$, and call these $r_1, \cdots, r_u, \cdots, r_U$. Now, the probability that there is an edge between the nodes i and j in the sampled random graph is given by $W(r_i, r_j)$. We observe that r_u being sampled (for all u) establishes the exchangeability of nodes.

To define the prior link probability matrix $P_{ij}^{(0)}$, we select some canonical ordering of nodes (more on this below), e.g. $r_u = {}^{u}/_{U-1}$ where $u \in \{0, \cdots, U-1\}$, and define:

$$P_{ij}^{(0)} = W(r_i, r_j) \tag{7}$$

In words, $P_{ij}^{(0)}$ is simply the graphon W sampled on a grid.



(a) SBM (b) Ring of Cliques (c) Scale-free (d) Erdos-Renyi

Figure 7: **Sampled Graphons.** The quantity $P_{ij}^{(0)}$ for the graphs considered in this work, for $U = 320$.

As to the choice of a graphon W, we consider several options, including stochastic block models (SBMs), ring-of-cliques, scale-free model, and the Erdös-Renyi model – Figure 7 shows the corresponding sampled graphons. The simplest of these corresponds to the Erdös-Renyi model [18], where all pairs of nodes are equally likely to be connected, in which case $W(r_i, r_j) =$ constant. For the scale-free network, we use the following graphon with parameter β (which we set to 0.5):

$$W(r_i, r_j) := \frac{U^\beta}{16} (r_i + 1)^{-\beta} (r_j + 1)^{-\beta} \tag{8}$$

Permutation over Graph Nodes. Recall that we obtained $P_{ij}^{(0)}$ by ordering the nodes in a canonical but arbitrary manner. However, permuting the node labels does not change the relevant properties of the underlying graph. To account for this, we allow for arbitrary permutations of node labels in Equation 6, meaning that the modules (labeled by indices i and j in P_{ij} in Equation 6) must not adhere to the canonical order defined for $P_{ij}^{(0)}$. However, this exposes us to a difficult challenge of inferring the correct permutation over node labels (recall that there are $U!$ such permutations where $U \approx \mathcal{O}(1000)$). We approach this heuristically by leveraging the Hungarian method to obtain an approximate solution to the search over all $U!$ possible permutations.

To this end, let $\sigma : \{0, \cdots, U-1\} \to \{0, \cdots, U-1\}$ be some permutation (i.e. a bijection) over the set of node labels. Let C be a cost matrix, whose entries C_{vw} measure the cost associated with mapping $\sigma(v) \mapsto w$. We define this as:

$$C_{vw} = \sum_i \left(P_{vi} - P_{wi}^{(0)} \right)^2 \tag{9}$$

The cost matrix is fed to a linear assignment problem solver (e.g. Hungarian method, accesible via `scipy.optimize.linear_sum_assignment`) to obtain the permutation σ_*. An implementation detail is that this solver is supported only on CPUs, necessitating a device transfer to move C_{vw} to the CPU. However, in the unconditional setting, this matching must happen only once per iteration (and not once per sample in the batch, like in e.g. DETR [13]); consequently, the overhead is not significant and we observe good GPU utilization.

A.2 Illustration of Learned Graphs

We visualize the learned graphs in Figure 8.

B ModFFN and SKMDPA

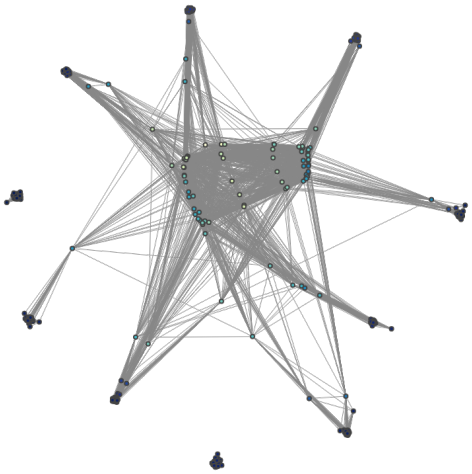
B.1 ModFFN Parameter Example

The majority of ModFFN parameters are shared between modules, but a small number of parameters (the code vectors, in the unconditional case) are not. Let's consider a concrete example — a 2 layer deep ModFFN that takes in $d_{in} = 384$ dimensional inputs and returns $d_{out} = 384$ dimensional outputs, with a hidden layer size of $d_h = 1536$. Let us assume that the number of modules is U, each associated with a code vector of dimension $d_c = 384$. Let us further assume that the total number of such ModFFNs in the network is $L = 8$. The total number of parameters in all ModFFNs is given by: $L \cdot (2d_{in} \cdot d_h + 2 \cdot d_c \cdot d_h) + U \cdot d_c = 1.89 \times 10^7 + 384 \cdot U$. The multiplier 2 is due to the fact that each FFN has two layers. We can observe that the number of parameters increases only very modestly with the number of modules U. It would take more than 10000 modules before the contribution due to code vectors starts becoming noticeable.

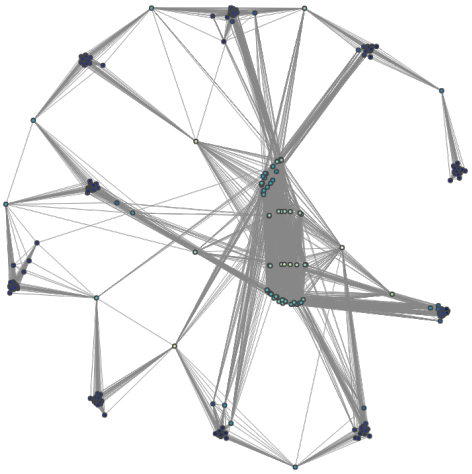
B.2 SKMDPA

The notion of a module in NACs generalizes that of a latent vector (a row in the latent array) defined in Perceiver IO [28]. Recall from Section 2 (Circuit Design) that each module in NACs can be described by three vectors: a signature vector, a code vector, and an initial state. The initial state exactly corresponds to a latent vector in Perceiver IO, and therefore, the number of modules exactly equals the number of latent vectors in Perceiver IO. However, we note that there is no equivalent of signature and code vectors in Perceiver IO.

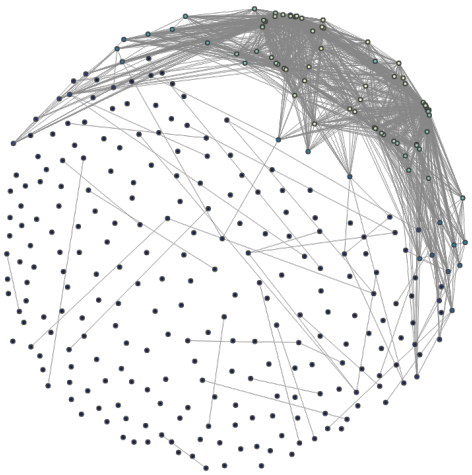
In this context, SKMDPA in NACs is functionally a drop-in replacement for the transformer-based attention in Perceiver IO. Like transformer-based attention, it also consumes U input vectors and outputs U output vectors. Unlike transformer-based attention where all input vectors can influence all output vectors, only some input vectors can influence some other output vectors in SKMDPA. Which vector influences which other vector is specified by the corresponding signatures (recall that there are exactly U of these), which are also learned.



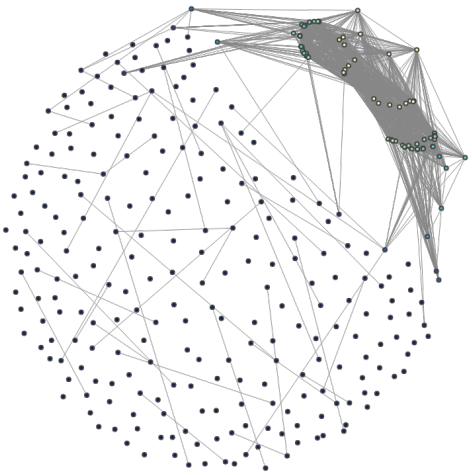
(a) SBM



(b) Ring of Cliques



(c) Scale-free



(d) Erdos-Renyi

Figure 8: **Visualization of the Learned Graphs.** We draw a an edge between two nodes i and j if the corresponding probability P_{ij} (see Equation 2) is larger than 0.5. Further, the distances between any pair of nodes in the visualized graphs are optimized to be proportional to the distance between their corresponding signatures.

C Pre-training Details

Data Augmentation. We used a standard data augmentation regimen [50, 33] when pretraining on (Tiny-)ImageNet for both NACs and PIOs.

Convolutional Preprocessor. For Tiny-ImageNet, the image tokenization pipeline was the same across both models. For each, we downsample twice: first by a factor of 4 (via a convolutional layer with kernel size and stride of 4, followed by a layernorm), and then by another factor of two (via another convolutional layer with kernel size and stride of 2, followed by a layernorm). Between the two downsampling blocks, we include a ConvNeXt [35] block with a single convolutional layer (with a kernel size of 7 and no stride) followed by two point-wise linear layers. Each of the linear layers in the ConvNeXt block is preceeded by a layernorm, and there is a GELU activation after the first linear layer. The output of the preprocessor is a feature-map of size 8×8 for Tiny-ImageNet and 28×28 for ImageNet. We concatenate a positional encoding to this feature-map before feeding it to the model. These encodings are learnable vectors, each with 64 dimensions. **Convolutional Preprocessor.** For Tiny-ImageNet, the image tokenization pipeline was the same across both models. For each, we downsample twice: first by a factor of 4 (via a convolutional layer with kernel size and stride of 4, followed by a layernorm), and then by another factor of two (via another convolutional layer with kernel size and stride of 2, followed by a layernorm). Between the two downsampling blocks, we include a ConvNeXt [35] block with a single convolutional layer (with a kernel size of 7 and no stride) followed by two point-wise linear layers. Each of the linear layers in the ConvNeXt block is preceeded by a layernorm, and there is a GELU activation after the first linear layer. The output of the preprocessor is a feature-map of size 8×8 for Tiny-ImageNet and 28×28 for ImageNet. We concatenate a positional encoding to this feature-map before feeding it to the model. These encodings are learnable vectors, each with 64 dimensions.

Hyperparameters for Tiny-ImageNet. Table 4 shows all hyperparameters we used to train NACs and PIOs on Tiny-ImageNet. For Perceiver IO, we adapt a popular third-party implementation[1]. The learning rate and weight decay were tuned individually for each model. We also note that the weight decay is neither applied to parameters with dimension less than or equal to 1, nor codes and signatures.

Hyperparameters for ImageNet. Table 5 shows all hyperparameters we used to train NACs on ImageNet. We use a pretrained PIO model produced by [28] and made available in Huggingface's transformers repository[2].

D Limitations & Future Work

Future work. Modular, general-purpose neural architectures present many opportunities for future work. These include (1) incorporating train-time sparsity where each input sample or token only updates a small fraction of the modules for one training step, (2) adding sparse kernels to the implementation to support sparsification, (3) conditioning the configuration of modules based on the task (perhaps in addition to sample-conditional generation), (4) thorough analysis of scaling laws associated with such architectures, (5) development of a principled approach to performing sparse updates to modules, (6) learn a predictive model that identifies a subnetwork of modules for conditional computation, and (7) train modules to solve certain hard-coded problems and learn how to wire them.

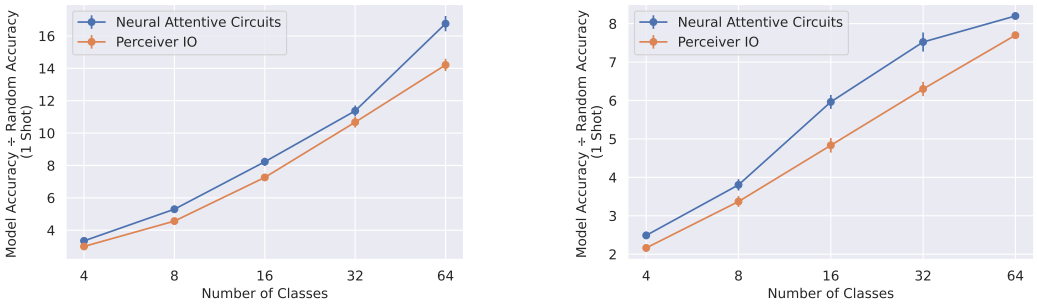
Limitations. In addition, this work has several limitations. First, pre-training of NAC on large-scale natural language datasets (such as C4 [43]) and on large-scale multi-modal datasets was determined to be out of scope. Future work may build on this architecture and determine the best way to perform this large-scale pre-training, and the incorporation of self-supervised objectives. Second, while our approach for learning the circuit generator via signatures and a learned cross attention mechanism was sufficient for validating that NACs can learn sample-conditional module configurations, we expect that further experimentation with auto-regressive models and GFLowNets [12]) will yield better configurations. Third, an analysis of the scaling properties of NACs as compared to homogenous general-purpose modules would likely be useful for those seeking to compare against this model-class with access to different amounts of compute resources.

[1] `https://github.com/lucidrains/perceiver-pytorch`

[2] `https://huggingface.co/docs/transformers/model_doc/perceiver`

Table 4: Hyperparameters for training on Tiny-ImageNet.

Hyperparameter	**Neural Attentive Circuit**	**Perciever IO**
Batch size	1024	1024
Number of epochs	400	400
Augmentation pipeline	Standard [33, 50]	Standard [33, 50]
Weight decay	0.05	0.05
Optimizer	AdamW	AdamW
Learning rate scheduler	Cosine	Cosine
Base peak learning rate (for batch size 512)	0.0003	0.0005
Base min learning rate (for batch size 512)	1e-5	1e-5
Warmup epochs	25	25
Warmup from learning rate	1e-6	1e-6
Dimension of state (d_{model})	384	384
Layers (L)	8	8
Layers that do not share weights	8	8
Processor modules (NACs) or latents (PIOs) (U_p)	320	320
Attention heads	6	6
Read-in (NACs) or cross-attention (PIOs) heads	1	1
Activation function	GEGLU	GEGLU
FFN hidden units	1536	1536
Output modules (U_o)	64	N/A
Signature dimension (d_{sig})	64	N/A
Code dimension (d_{code})	384	N/A
Sampling temperature (τ)	0.5	N/A
Kernel bandwidth (ϵ)	1.0	N/A
Modulation α at initialization	0.1	N/A



(a) CUB 1-Shot

(b) CIFAR 1-Shot

Figure 9: We evaluate NACs and the baseline PIO on CUB (left) and CIFAR (right) while varying the number of classes between 1 and 64 (x-axis) and holding the number shots constant at 8. On the Y-axis we show the lift, which is the model accuracy divided by the random accuracy for the task. We see that the NAC remains competitive across the board with respect to 1-shot performance.

E Additional Fewshot Experiments

Figure 9 shows the result of varying the number of ways in the few-shot task. This experiment demonstrates that NACs maintain a consistent improvement over the PIO baseline across a wide range of task difficulties.

F Computational Complexity Comparison

Selecting what modules are dropped. After pre-training on Tiny-ImageNet, we construct a measure of importance of a module. To this end, we consider the link probability matrix P (see Equation 2),

Table 5: Hyperparameters for training on ImageNet. Note that we do not train the Perceiver IO, but use a pre-trained model. Nevertheless, we gather available hyperparameters from Jaegle et al. [28] and the official code-release for the reader's convenience.

Hyperparameter	**Neural Attentive Circuit**	**Perciever IO**
Batch size	1024	1024
Number of epochs	110	110
CutMix	✓	✓
MixUp	✓	✓
RandAugment	✓	✓
Augmentation pipeline	Standard [33, 50]	Custom [28]
Weight decay	0.05	0.1
Optimizer	AdamW	LAMB
Learning rate scheduler	Cosine	Custom[28]
Base peak learning rate (for batch size 512)	0.0003	0.001
Base min learning rate (for batch size 512)	1e-5	0
Warmup epochs	25	N/A
Warmup from learning rate	1e-6	N/A
Dimension of state (d_{model})	512	1024
Layers (L)	8	48
Layers that do not share weights	8	6
Processor modules (NACs) or latents (PIOs) (U_p)	960	512
Attention heads	8	8
Read-in (NACs) or cross-attention (PIOs) heads	1	1
Activation function	GEGLU	GELU
FFN hidden units	1024	1024
Output modules (U_o)	64	N/A
Signature dimension (d_{sig})	64	N/A
Code dimension (d_{code})	512	N/A
Sampling temperature (τ)	0.5	N/A
Kernel bandwidth (ϵ)	1.0	N/A
Modulation α at initialization	0.1	N/A

where P_{ij} gives a measure of how strongly module i is connected with module j. To obtain the importance measure of the module at index i, we define the quantity q_i as following:

$$q_i = \sum_j P_{ij} \tag{10}$$

We then sort the modules by their respective q_i, and drop the ones for which q_i is smallest. This ensures that the most strongly connected modules are dropped last.

Eliminating connections between modules. In addition to dropping modules, we also investigate the effect of prohibiting attentive connections between modules (Figure 10). To this end, we sort P_{ij} and successively eliminate the smallest elements by setting them to 0. For instance, if 90% of connections are removed, we have that 90% of elements in the lower and upper triangular portion of the matrix P_{ij} (excluding the diagonal) are set to 0.

From Figure 10, we make two **key observations: (a)** A large fraction ($\approx 90\%$) of connections can be disallowed without incurring a significant loss in performance. In this regime, the SBM and Ring-of-Cliques priors (with 59.46% and 59.91% validation accuracy, respectively) are advantaged over the scale-free prior (with 59.1% validation accuracy), even though the latter obtains the best validation performance given the full model (60.76%). **(b)** As more edges are disallowed, we find that the SBM and Ring-of-Cliques prior abruptly break down. However, the scale-free and Erdös-Renyi priors keep gracefully degrading until eventually almost all connections are prohibited.

Further, we investigate how the graph over module changes as connections are dropped, e.g. around the critical value for SBM and Ring-of-Cliques priors. We discover interesting patterns, as shown in Figures 11, 12, 13 and 14.

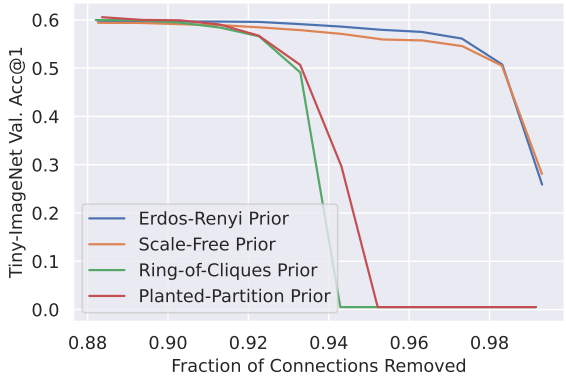


Figure 10: **Effect of eliminating connection between modules.** We find that a relatively large fraction ($\approx 90\%$) of module connections can be disallowed while only marginally affecting performance. Beyond that, we see a stark difference in behaviour: on the one hand, for ring-of-clique and stochastic block model priors, we find that beyond a critical amount of removed connections, the performance degrades drastically. On the other hand, we find that scale-free and Erdös-Renyi priors degrade gracefully as eventually all connections are disallowed.

(a) 50.69% acc. @ 93% edges lost. (b) 29.62% acc. @ 94% edges lost. (c) 05.00% acc. @ 95% edges lost.

Figure 11: **Eliminating attentive connections (edges) in a NAC trained with SBM Prior.** We observe that the accuracy drops drastically as the edges relaying information between clusters of modules are eliminated. Further, we see the emergence of four core clusters over modules, and the performance is essentially random when the attentive connections between these clusters are severed.

Finally, while we leave a thorough benchmarking to future work, we note that this behaviour allows for the use of sparse attention kernels. Even without invoking sparse attention kernels, it is possible to extract an inference-time speed-up by leveraging the block-sparse structure of the attention sparsity mask, for e.g. the SBM prior.

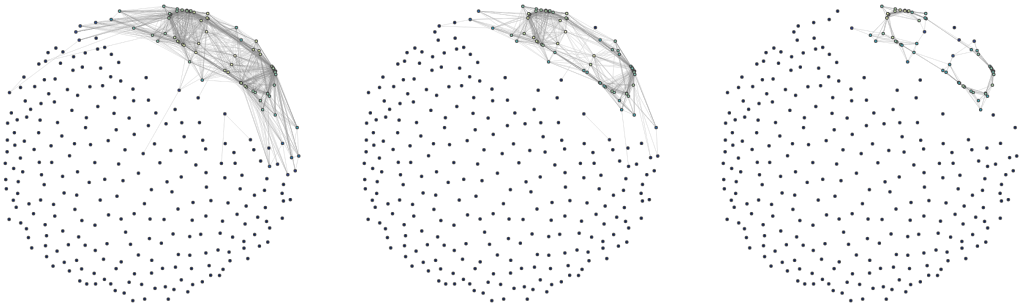
Details around inference speed measurements. For Figure 5, we measured the time per sample for the NAC on an A100 GPU. We used a batch-size of 64 and timed the model under a `torch.inference_mode` context manager.

Clarifying the effect of sparsification. Figure 15 clarifies the effect of sparsification on NACs (with different graph priors) and Perceiver IO. The figure shows GFLOPs / sample on the X-axis and Tiny-ImageNet val accuracy on the Y-axis.

We vary the GFLOPs by dropping modules in NACs or latents in Perceiver IO. We make two observations: for NACs, even when the FLOP budget is reduced by a factor of 5 (to roughly 1.6 GFLOPs), the amount of performance lost is remarkably small, corresponding to e.g., roughly 1% in accuracy difference with respect to the full model with ring-of-cliques prior. In contrast, Perceiver IO loses 5.3% in validation accuracy with respect to the full model when given roughly the same amount of FLOPs (1.7 GFLOPs). Under 1 GFLOP, Perceiver IO degrades slower than NACs. This is because

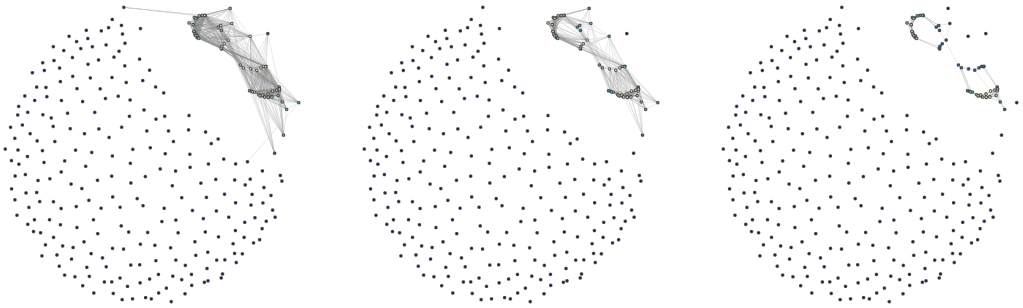
(a) 56.59% acc. @ 92% edges lost. (b) 49.09% acc. @ 93% edges lost. (c) 05.00% acc. @ 94% edges lost.

Figure 12: **Eliminating attentive connections (edges) in a NAC trained with Ring-of-Cliques Prior.** Like in Figure 11, observe the emergence of core clusters, but also important relay modules that connect multiple clusters.



(a) 54.56% acc. @ 97% edges lost. (b) 50.51% acc. @ 98% edges lost. (c) 28.08% acc. @ 99% edges lost.

Figure 13: **Eliminating attentive connections (edges) in a NAC trained with Scale-free Prior.** We observe the emergence of a cluster of well-connected *hub* modules, together with a tail of less connected modules. Severing the connections to the tail modules while just keeping the hub modules connected results in a roughly 10% decline in validation accuracy. But as the connections between the hub modules are severed, the performance significantly drops.



(a) 56.13% acc. @ 97% edges lost. (b) 50.75% acc. @ 98% edges lost. (c) 25.88% acc. @ 99% edges lost.

Figure 14: **Eliminating attentive connections (edges) in a NAC trained with Erdös-Renyi Prior.** We see a pattern similar to the one in Figure 13. Interestingly, we observe that while the Erdös-Renyi prior encourages uniform connectivity between nodes (i.e. it is an uninformative prior), the learned graph resembles a scale-free network, which happens to perform best in the IID regime.

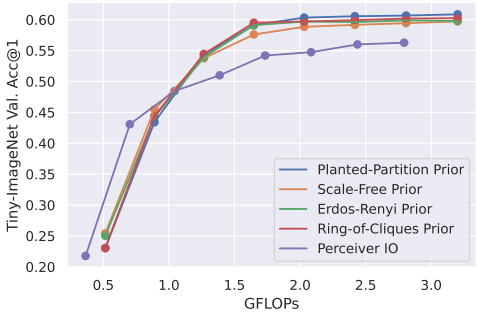


Figure 15: **Effect of Sparsification on NACs and Perceiver IO.** We show the effect of dropping modules (in NAC) and latents (in Perceiver IO) on Tiny-ImageNet validation accuracy and compute.

the cross attention in Perceiver IO is computationally lighter than the read-in attention in NACs. This difference only becomes visible in the very low compute regime (i.e., under 1 GFLOP), where the read-in attention's fixed compute cost is not outweighed by the other attention mechanisms and FFNs. We further note that the read-in attention in NACs can be replaced by the computationally cheaper cross attention like in Perceiver IO, at the cost of a modest amount of performance (corresponding to roughly 1% validation accuracy on Tiny-ImageNet).

G NLVR2 Experiments

We begin by evaluating the zero-shot performance of OpenAI's CLIP model on NLVR2. CLIP takes in a collection of images and captions and produces a similarity score between the two modalities. In order to get CLIP to produce logits, we initially attempted to mimic the evaluation setup of the NLVR2 leaderboard by providing CLIP with a concatenated pair of input images and two sentences. The first sentence was provided in the NLVR2 dataset, while the second was generated by prompt engineering GPT-J-6B to produce a negated sentence. We **achieved a 53% zero-shot accuracy** on this task, only 3% above the random chance baseline. For context, the official NLVR2 leaderboard, would place zero-shot CLIP in fourth place just behind VisualBERT which achieved a 67% accuracy, and above methods such as MaxEnt (54%), FiLM[40] (51%), N2NMN [26] (51%) and MAC Networks [27] (50.8%).

In this set of experiments, we use CLIP as a frozen backbone network and apply the image and text encoder to each modality. We take then take the output encoded tokens and fine-tune a Perceiver IO and a Neural Attentive Circuit on these, following the typical NLVR leaderboard evaluation. With Perceiver IO, we see an improvement in the **validation accuracy to 64%**.

Conditional NAC Architectural Details. In our experiments with conditional NACs, the circuit generator was implemented by a simple cross attention layer followed by a feed forward network (FFN), and a self-attention layer followed by two FFNs run in parallel (more on that below). In the initial cross-attention layer, the keys and values were derived by linearly transforming pre-pooling CLIP text embeddings (which yields one embedding vector per text token). We used CLIP-base (the smallest available pre-trained CLIP) to minimize the compute requirement. The queries in the first cross-attention layer were learned 384-dimensional vectors, of which we had exactly U (i.e. one per module). The output of the self attention layer was therefore another set of U vectors, each of which was passed through two different FFNs in parallel. The first FFN produced the signatures for each module (U 64-dimensional vectors), whereas the second FFN produced the corresponding codes (U 384-dimensional vectors).

H Text Embedding Experiments

Text Dataset Construction. In order to evaluate the text embeddings produced by the NAC circuit generator, we constructed a small dataset of simple statements. These statements involve different semantics including *hard cardinality* which includes a precise number of objects, *soft cardinality*

which includes words like “many” or “few”, *existential* which only indicates that an object is present, and *spatial relationships* which use a prepositional phrase to denote a physical relationship. We constructed a simple template for each reasoning skill,

1. **hard cardinality**: “There are exactly {integer} {object(s)}”.
2. **soft cardinality**: “There are many {object(s)}”.
3. **existential (singular)**: “There is a {object}”.
4. **existential (plural)**: “There are any {objects}”.
5. **existential (adjective)**: “There is a {color} {object}”.
6. **spatial relationship**: “There is a {object} {spatial relationship} {object}”.

and constructed a statement for each combination of our chosen set of integers, colors, objects (we combined animals into the objects category), and spatial relationships. The sets we selected are as follows:

1. **integers**: "two", "three", "four", "five", "six", "2", "3", "4", "5", "6"
2. **animals**: "cat", "dog", "mouse", "cheetah", "stingray", "jellyfish", "guinea pig", "marmot", "chimp", "gorilla"
3. **objects**: "boat", "flute", "door", "laptop", "ball", "machine", "paper towel"
4. **colors**: "blue", "black", "red", "green", "yellow", "purple", "orange", "white", "teal", "beige", "pink", "gray", "pink"
5. **spatial relationships**: "near", "far away from", "next to", "on top of", "under", "behind", "on", "by", "above", "in front of"

CLIP Text Embeddings. We evaluate the CLIP text embeddings using the same text dataset used to assess the conditional NAC’s link probability matrix generation in Figure 6. We see in Figure 16 that CLIP’s latent space is primarily organized around semantic categories such as “ball” or “dog”. This stands in contrast with NAC’s embeddings which are organized by abstract reasoning tasks such as counting (hard cardinality) and spatial relationships. Figure 17 shows the same TSNE embedding but points to the same samples as referenced in the main paper in Figure 6.

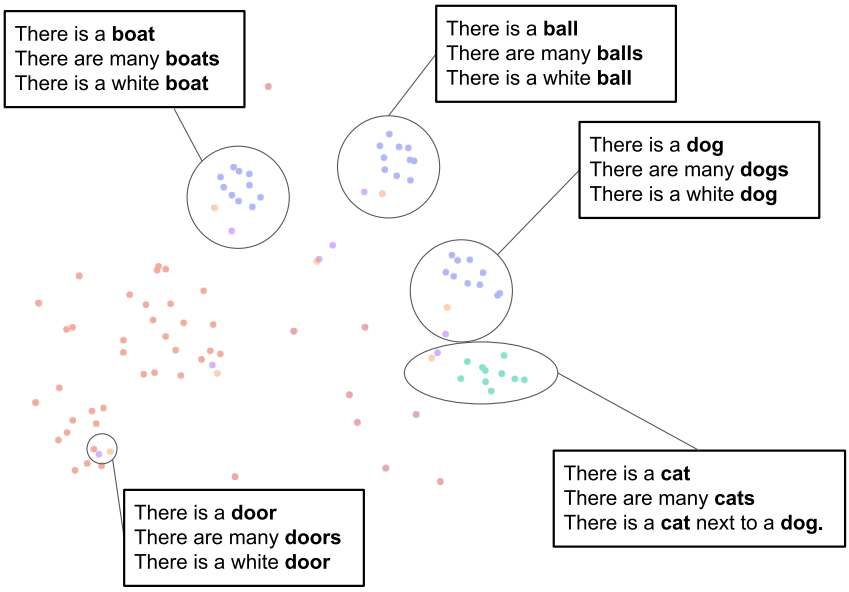


Figure 16: **CLIP Text Embeddings**. This plot shows a TSNE of CLIP’s pooled output for the same sentences. We can see that CLIP’s latent space is primarily organized by semantic category, not cardinality, spatial relationships, or quantification.

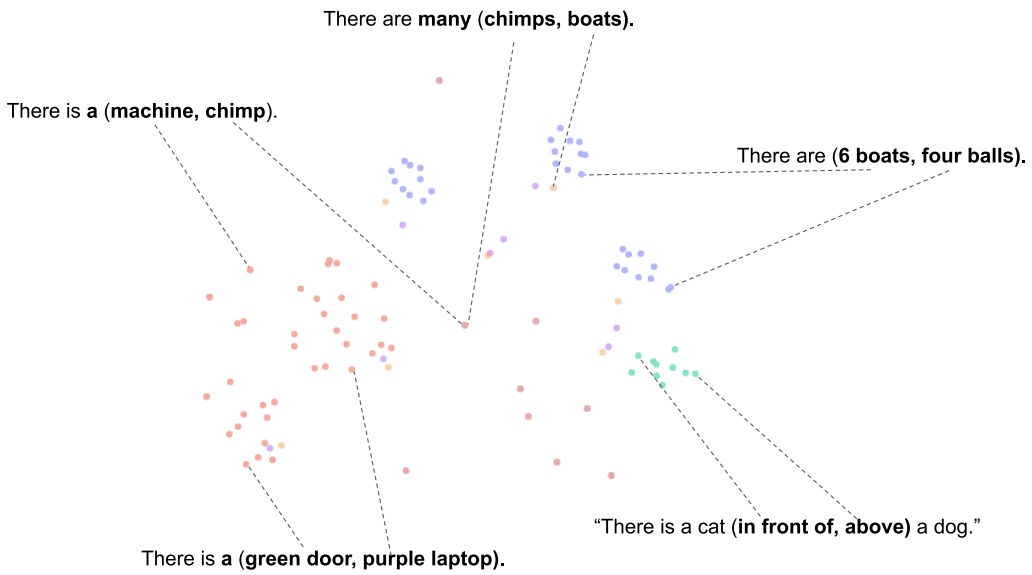


Figure 17: **CLIP Text Embeddings**. This plot shows a TSNE of CLIP's pooled output for the same sentences. We can see that CLIP's latent space is primarily organized by semantic category, not cardinality, spatial relationships, or quantification.

I Anonymized Code

Anonymized research code can be found here[3].

[3] `https://anonymous.4open.science/r/Neural-Attentive-Circuits/README.md`