# Attention for Compositional Modularity

**Oleksiy Ostapenko**[123] **Pau Rodríguez**[3] **Alexandre Lacoste**[3] **Laurent Charlin**[145]
[1]Mila - Quebec AI Institute, [2]Université de Montréal, [3]ServiceNow, [4]HEC Montréal,
[5]Canada CIFAR AI Chair

## Abstract

Modularity and compositionality are promising inductive biases for addressing longstanding problems in machine learning such as better systematic generalization, as well as better transfer and lower forgetting in the context of continual learning. Here we study how attention-based module selection can help achieve compositonal modularity – i.e. decomposition of tasks into meaningful sub-tasks which are tackled by independent architectural entities that we call modules. These sub-tasks must be reusable and the system should be able to learn them without additional supervision. We design a simple experimental setup in which the model is trained to solve mathematical equations with multiple math operations applied sequentially. We study different attention-based module selection strategies, inspired by the principles introduced in the recent literature. We evaluate the method's ability to learn modules that can recover the underling sub-tasks (operation) used for data generation, as well as the ability to generalize compositionally. We find that meaningful module selection (i.e. routing) is the key to compositional generalization. Further, without access to the privileged information about which part of the input should be used for module selection, the routing component performs poorly for samples that are compositionally out of training distribution. We find that the the main reason for this lies in the routing component, since many of the tested methods perform well OOD if we report the performance of the best performing path at test time. Additionally, we study the role of the number of primitives, the number of training points and bottlenecks for modular specialization.

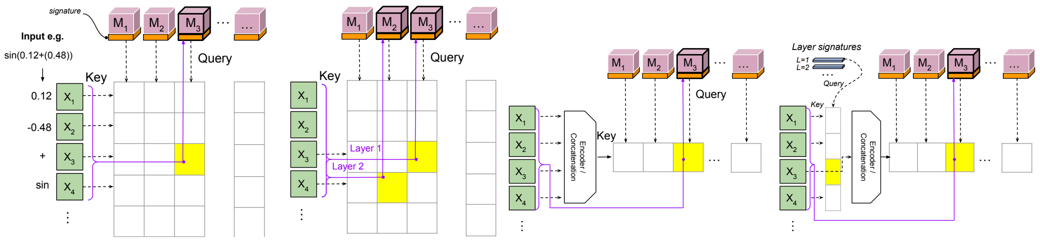
## 1 Introduction

Modularity and compositionality are appealing inductive biases for addressing several long-standing problems of machine learning such as generalization under distribution shift, a.k.a out of distribution (OOD) generalization [19, 7] and continual learning (CL) [6, 20, 17]. Given a set of modules the goal is to find a decomposition of knowledge into these modules s.t. a new task can be solved efficiently through recombination of modules (compositionality), addition of new, or/and fast adaptation of existing modules.

Achieving compositionality presents a number of challenges. These include (a) decomposing tasks into reusable modules. While related to sub-task discovery [14, 23], this has a promise of improving sample complexity through positive transfer and improved OOD generalization [19]. (b) Routing samples through a set of modules. The latter is especially challenging when the routing procedure can be subject to forgetting as is the case continual learning. Additional challenges not addressed in this work include e.g. pruning and addition of new modules to ensure enough capacity when new tasks must be learned.

Different forms of attention have been proven useful for achieving meaningful modularity in the mixture-of-experts (MoE) setup [21, 7, 10, 9, 22]. Here we are interested in *compositional modularity* — the training tasks should be decomposed into meaningful and reusable sub-tasks that can be executed in parallel or sequentially. In our instantiation each such subtask is assigned to an architectural entity

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



(a) Cross Attention (b) Structural Oracle (c) Attention with encoder (d) Attn. + encoder + Sbtl.

Figure 1: Different attention-based module selection mechanisms. (a) Cross attention where each input is encoded separately into a key which is matched to module signatures encoded into queries. (b) Structural oracle that is given the privileged information about which variable is relevant for module selection at each layer. (c) Module selection mechanism where inputs are first pre-processed using a learnable encoder, or a concatenation function. (d) A cross-attention like selection bottleneck is added to select one variable that will be used for matching with modules.

— a module, and the network is dynamically constructed for each input sample through a sequence of module selection & application steps. In contrast to the MoE, where the supervision is provided after each module's application, in compositional modularity the supervision is provided after the final selection step — i.e. after the desired depth of the modular network is achieved. While MoE is only limited to parallel sub-task execution, the main advantage of compositional modularity is that it can decompose a task into sub-tasks, where the sub-tasks are executed both sequentially or in parallel. For example, given a task of solving additions and subtractions (e.g. $4 + 5 - 3 = 6$), a MoE would learn a single module (i.e. expert) to execute the whole expression, while a compositional system should be able to learn a module specialized on "+" and a module specialized on "-" executing both sub-tasks sequentially. This can allow such system to achieve better compositional OOD [4, 22] generalization and efficient adaptation to new tasks in CL [25].

The main question we study here is how compositional modularity can be achieved. To this end we first design an experimental setup based on math equations. This naturally compositional domain provides the possibility to generate simple compositional problems for fast experimentation. We study a wide range of key-value attention [3] based module selection mechanisms in the compositional modularity setting. We find that only the structural oracle baseline — a baseline which is given the privileged information about which part of the input should be used for module selection, can reliably achieve modular specialization and reduce the compositional OOD loss. Further, we find that the main reason for the poor OOD performance lies in the pure quality of the routing component, which we demonstrate through the excellent OOD performance of modular baselines in the test-time oracle setting — a setting where we report the OOD loss of the best module selection path after checking all possible paths. Finally, we discuss the role of task diversity, number of samples per primitive operation, as well as the role of bottlenecks for preventing modular collapse.

## 2 Models

Here we describe the modular systems used in this work. Similarly to neural production systems (NPS) [9], we aim to learn a set of modules, where each module learns a simple function that can be applied to the input variables. In contrast to NPS and similarly to [22], we want the modules to form a chain (modules are applied in superposition to each other), where the supervision is only applied at the end of the chain.

**Input data.** Similarly to [9], we assume that the input variables represent some meaningful entities of the real world. In general terms, these entities can correspond to, e.g., objects in vision, words in NLP, or even to other high level variables. The task of extracting such variables from high-dimensional input data is considered external to the system that we study here and can be accomplished by any method built for extracting object-centric representations [8, 15, 16]. We assume that input data is given as a set of $N$ variables $\{X_1, X_2, \dots X_N\}$, hence a sample $\mathbf{x} = \{x_1, x_2, \dots, x_N\}$ consists of values of each of the $N$ underlying variables, s.t. each $x_n \in \mathbb{R}^d$ is a $d$-dimensional vector.

**Module selection.** We test several approaches for module selection. These approaches share the following module selection principle, that differ in detail. We consider a set of $M$ modules, each module is equipped with a signature vector $s_m \in \mathbb{R}^r$. Additionally, we consider a modular system with $L$ layers (we use $l$ to index these layers). At each layer the system must select a module to apply to the input of the current layer $l$ conditioned on this input. The pool of possible modules is shared across layers. Module selection is performed in the following 2 steps using key-value attention mechanism:

$$k_m = W_l^{(k)} s_m, \quad q = W_l^{(q)} E_{\eta_l}(\mathbf{x}) \tag{1}$$

$$p_l(m|\mathbf{x}) \propto e^{k_m^T q}, \tag{2}$$

where $W_l^{(k)} \in \mathbb{R}^{p \times r}$, $W_l^{(q)} \in \mathbb{R}^{p \times d'}$ are learnable parameters and $p_l(m|\cdot)$ is the probability of the $m^{th}$ module at $l^{th}$ layer. $E$ is an encoder that is optionally applied to the input $x$. As detailed next, encoder can be a learnable function such as neural net, in which case it is parameterized with $\eta_l$. Encoder can also represent a simple concatenation operation. The encoder projects $\mathbf{x}$ into $d'$ dimensional space. We use $\theta_m^{(st)} := (W^{(k)}, W^{(q)}, \eta_m)$ to refer to structural parameters. Next, we describe four variations of the above mechanism used in this paper.

(1) *Cross-attention:* in this case, the projector $E$ is an identity function, i.e $\mathbf{x} = E(\mathbf{x})$. Additionally, we encode each $x_n$ separately with a shared set of weights $W_l^{(q)}$, which gives a separate query $q_n$ per input variable. As visualized in Fig. 1a), we use dot-product to calculate the attention score matrix $R \in \mathbb{R}^{N \times M}$, s.t. $R_{n,m} = k_m^T q_n$.

(2) *Concatenation:* the projector $E$ performs a concatenation operation producing a single vector $\mathbf{x}' = E(\mathbf{x}) = concat(\mathbf{x}) \in \mathbb{R}^{d'}$ ($d' = dn$ here). Queries are then produced using Eq. 1. The score matrix $R \in \mathbb{R}^{1 \times M}$, s.t. $R_{1,m} = k_m^T q$ (Fig. 1c).

(3) *Transf. Encoder:* in this case $E$ is an encoder neural net parameterized with $\eta_l$, whose goal is to summarize information present in all variables in $\mathbf{x}$ into a single vector $x' = E(\mathbf{x}; \eta_m) \in \mathbb{R}^{1xd'}$, where $d' < dn$ (Fig. 1c). In our experiments, $E$ is instantiated with a transformer encoder network [24]. The score matrix $R \in \mathbb{R}^{1 \times M}$, s.t. $R_{1,m} = k_m^T q$.

(4) *Structural Oracle:* the model has privileged information about which of the $N$ input variables is relevant for module selection at each layer. E.g. for the math expression $(0.4 + 0.01) * 0.4$, instead of having to learn which input variable to use for module selection at each layer (one of the real numbers or one of the two operations), this baseline will use operation "$+$" for selecting module at layer $l = 0$ and "$*$" at layer $l = 1$ (Fig. 1b).

We use the straight-through Gumbel softmax trick [13] to perform differential hard-module selection over the score matrix $R$. In our experiments we always perform hard selection at test time. At train time, we additionaly consider a baseline ("+*all path*") that, instead of a hard selection, checks every possible path in the modular system and weights the training loss of each path with it's probability. A path probability can be calculated as an exponential of the sum of the log-probabilities of each module in a given path, where module's activation probability is calculated as in Eq. 1.

**Module application and functional bottleneck.** The selected module $m^*$ receives all $N$ input variables. Similarly to NPS [9] we instantiate another bottleneck for the number of variables that can be processed by the functional component of each module – the *functional bottleneck*. More specifically, we only allow two variables to be processed by the module's functional parameters. This design decision is motivated by the math equations domain as detailed in Sec. 3. This bottleneck can be learned using another attention mechanism (query coming from each variable and key coming from the selected module's signature). As we demonstrate in Sec. 4.2, the functional bottleneck is essential to prevent module collapse.

**Compositional modular systems vs. MoE.** In our experiments we consider compositional modular systems. Its main difference to MoE is that the number of module applications $L$ is greater than 1, whereas in MoE $L = 1$. Here, we refer to a single application of modules as a modular layers, where at each layer a module is selected from the same pool of modules. Vanilla MoE system has the disadvantage of not being able to compositionally generalize to unseen combinations of attributes.

**Parameter sharing.** Unless stated otherwise, modules at each layer share structural parameters $W_l^{(q)}$ as well as $\eta_l$ (if applicable). Motivated by the CL use case, where less parameter sharing can reduce forgetting, we ablate the role of sharing of $W_l^{(k)}$ in Sec. 4. Note, not sharing $W_l^{(k)}$ between modules is essentially equivalent to instantiating $s_m \in \mathbb{R}^p$ and learning it directly. Similarly to NPS, the inputs are pre-processed by a simple MLP encoder (before application of each of the modular layers), which is also shared across all modules and layers.

**Training.** We derive our training objective by treating the optimal module selection path as a hidden variable denoted as $z$. We consider a single path through the modular system as a sequence of $L$ consecutive module selection and application steps. The number of possible paths is $M^L$. We first derive a lower bound for the log-likelihood as (cf. App. A for details):

$$\log p(y|x) \geq \mathbb{E}_q \Big( \log \frac{p(y|x, y)}{q(z)} \Big) = \mathbb{E}_q \log p(y|x, z) + \mathbb{H}_q, \tag{3}$$

where $\mathbb{H}_q$ denotes the entropy of the distribution over paths $q(z)$. Our training objective is to find parameters $\theta^* = \{\theta^{(st)}, \theta^{(f)}\}^*$ that maximize the right-hand side in the Eq. 4, that is:

$$\theta^* = \arg\max_\theta \frac{1}{|D_{tr}|} \sum_{x,y \in D_{tr}} \Big( \sum_z q(z; \theta^{(st)}) \log p(y|x, z; \theta^{(f)}, \theta^{(st)}) - \sum_z q(z; \theta^{(st)}) \log q(z; \theta^{(st)}) \Big), \tag{4}$$

where $D_{tr}$ is the training dataset. By default, in our experiments we ignore the entropy term and approximate the expected log-likelihood with a single sample of $z$ obtained by sampling modules at each layer from the Gumbel-softmax distribution parameterized with score matrix $\mathbb{R}$. We also consider baselines where loss is computed exactly by checking every possible path for a given pool of modules. This is not a scalable solution in a general case, as the number of paths grows as $O(M^L)$.

## 3 Data generation

**Math equation tasks.** Here we consider the input data to be comprised of a set of $N$ attributes $X_{1...N_x}$ paired with labels $Y$. Both are sampled from a joint distribution $p(Y, X_{1...N}) = p(Y|X_{1...N_x})p_t(X_{1...N_x})$. We instantiate this setting in the math equations domain similar to [9, 19] with $N = 5$. We select this domain due to it's simplicity, which enables fast experimentation, as well as the compositional nature of this domain.

Specifically, $X_1, X_2 \in \mathbb{R}^{[-1,1]}$ follow a uniform distribution $U([-1, 1])$ over the domain of real numbers between $-1$ and 1. Variables $X_{3,4}$, represent the primitive math operations to be performed on the first two variables, $X_5$ is a constant. Hence, $p_{X_{3,4}} = U(\mathbb{S})$, where $\mathbb{S}$ is a set of all possible operators (e.g."+", "-", "sin" etc., the list of operations is in App. B). The dependant variable $y$ is generated by applying sampled operations $x_3$ and $x_4$ to the inputs $x_1$ and $x_2$ sequentially, i.e. the result of one operation becomes the input to the next: in the reverse polish notation "$x_1$ $x_2$ $x_3$ $x_1$ $x_4$" (e.g. `".1 .04 + .1 *"` evaluates to .014).

In this setup, given $L = 2$, at each step the modular system has to learn to select modules given all the samples of all five input variables $X_{1...N}$. The information about which module to select is encoded in $X_3$ for the first layer and in $X_4$ for the second. The privileged information about which variable to use for selection ($X_3$ or $X_4$) is only provided to the structural oracle baseline. The goal is to minimize the MSE loss by learning the mechanism $p(Y|X_{1...N_x})$.

**Data generation.** The training data $D_{tr}$ is generated by considering all possible combinations of length $L = 2$ of $|\mathbb{S}|$ math operations, i.e. $|\mathbb{S}|^L$ combinations. In the follwoing, we refer to one such combination as a task, and an underlying math operations as primitive, sub-task or mechanisms. For each sample we first randomly select 1 task out of $|\mathbb{S}|^L$ tasks that can be composed with $L = 2$ operations encoded in input variables $X_3$ and $X_4$. To generate $y$, we first sample $x_1, x_2 \sim U([-1, 1])$ and $x_3, x_4 \sim U(\mathbb{S})$. We then apply the operations $x_3$ and $x_4$ as described above. We sample the total of $(|\mathbb{S}|^L * C)/(|\mathbb{S}| * L))$ points, where $C$ defines the expected number of training points to be seen by the algorithm per operation[1]. Out of $|\mathbb{S}|^L$ tasks, we select $|\mathbb{S}|$ tasks for the compositional OOD test set (i.e. one task per primitive operation), which are excluded from the training data. In our experiments we use $C \in [150, 500, 1000, 5000, 10000, 20000]$ for the training dataset. We use $C = 200$ for the in-distribution (ID) test set and a larger $C = 2000$ for the OOD test set.

[1] In this process we see an expected number of $(|\mathbb{S}| * L)/|\mathbb{S}|^L$ operations per task, resulting in the expected number of samples per operation indeed to be $C = [(|\mathbb{S}| * L)/|\mathbb{S}|^L] * [(|\mathbb{S}|^L * C)/(|\mathbb{S}| * L))]$.

## 4 Experiments

In this section we investigate how different module selection approaches can achieve compositionality. We first present the baselines and metrics we use.

**Baselines.** We consider the following baselines. **Cross-Attention** *(X-Attn.)* baseline separately encodes each input variable and performs cross-attention operation to obtain an $N \times M$ matrix of scores, it selects the module with the highest score (Fig. 1a). **Structural Oracle** (Str.Oracle): similar to the *X-Attn.* baselines, but it is given the information about which variable to use for module selection at each step (i.e. $x_3$ at layer 1 and $x_4$ at layer 2). **Transformer Encoder** (Trans.): processes the input variables with a transformer encoder, producing one encoding vector that is then matched with the modules' signatures using key-value attention. This encoding incorporates information about all input variables but can become selective throughout the learning process.[2] **Concatenation + Attention** (C-Attn) baseline simply concatenates all the input variables and performs key-value matching with the resulted concatenated vector. **Functional Oracle**: only learns the structural parameters of the modules and uses a set of modules with a hard-coded ability to perform perfectly one of the mechanisms (math operations) used for data generation. For all modular architectures we use modules with functional component being an MLP of width 300, depth 2, and a ReLU [2] activation. Unless stated otherwise, we fix the number of modules to 30 in all cases independently from the number of primitives $|\mathbb{S}|$. We use two modular layers, i.e. $L = 2$, which is equal to the number of primitive operations per sample in our experiments. **Monolithic** baseline represents an MLP with 2 layers and the width of 9000 (30 times the width of each module in modular case). This baselines receives as input a concatenation of all input variables (i.e. no functional bottleneck).

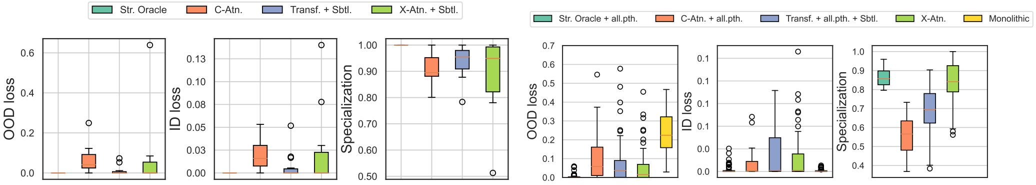
We ablate the importance of sharing $W_L^{(k)}$ between modules at a layer in Figs. 8a and 8d and come to the conclusion that sharing these weights does not result in performance improvement, and in many cases it actually results in performance decrease. For this reason, in our experiments modular baselines do not share $W_l^{(k)}$ between modules in a layer unless stated otherwise.

By default, modular baselines described above only apply the most likely module at each layer. Hence, at test time the most likely module corresponds to the module with the highest score in $R$; at training time, the module index is samples from the Gumbel-softmax distribution parameterized with the score matrix $R$. Additionally, we consider a version of the modular baselines that optimizes the objective in Eq. 4 exactly. Here, the final loss (MSE) is calculated by weighting the losses obtained with each possible module combination (i.e. path) with the probability of this particular path. The probability of each path is calculated as an exponential of the sum of the log probabilities of each module selected in a given path. These baselines are marked with *+all.pth.*.

We optionally add another hard attention layer to select which input variable to use as input to the module selection mechanism (as illustrated in Fig. 1d). This additional selection bottleneck is signaled with *+Sbtl.*. Here, instead of passing all variables to the selection mechanism at each layer, we only pass a single variable selected through matching it with layer signatures using cross-attention. We include this baseline as it results in slight performance improvements in some cases. The intuition is that in this way the model can easier learn the structure of the underlying math expressions – i.e. $x_3$ always encodes operations applied first and $x_4$ always encodes the operations applied thereafter.

**Metrics.** We use the following metrics in this section: (1) *ID loss* – in-distribution test loss; (2) Compositional OOD loss (*OOD loss*) — loss on the compositional OOD test set containing novel combinations of operations not seen during training; (3) *Specialization score* – consider an activation matrix $A^{R \times M}$, s.t. $A_{i,j}$ is proportional to the probability that module $j$ activates if the input contains operation $i$. We define specialization score as inverse average normalized entropy of each the rows of $A$, i.e. $1 - \frac{1}{R}\sum_{i=1}^{R}\frac{1}{\log(M)}\mathbb{H}(A_{i,:})$. A specialization score of $1$ signifies that the module selection mechanism could assign each primitive operation in the data generative process to a single module. This module, however, can be potentially the same module for all primitives. To control for this, we consider the collapse score. (4) *Collapse score*: consider $\vec{a} = \sum_{i=1}^{R} A_{i,:}$ — a vector representing unnormalized module activation distribution. We define collapse as the normalized entropy of $\vec{a}$, i.e. $\frac{1}{\log(M)}\mathbb{H}(\vec{a})$. Lower collapse score means that many operations activate the same modules. An ideal

[2] Only this selection mechanism worked in MoE context, cf. Fig. 6, where an expert must be selected based on a combination of $x_3$ and $x_4$, selecting experts based on only of the two would result in higher training error.



(a) Functional Oracle

(b) End-2-end training

Figure 2: (a) Functional Oracle, (b) End-2-end training. For both we show compositional OOD test loss (↓), ID test loss (↓) and specialization (↑) for selected module selection strategies. We use $M = 30$. Variation in (a) is due to 5 seeds, 4 different $|\mathbb{S}| \in [5, 7, 10, 15]$, $C = 10000$, in (b) – 5 seeds, $|\mathbb{S}| \in [5, 7, 10, 15]$, $C \in [1000, 5000, 10000, 20000]$. Detailed view in Fig. 8

system should have specialization and collapse scores equal to 1. We note that, as shown in Fig. 8(c), none of the tested baseline severely suffers from module collapse (c.f. discussion in sec. 4.2).

### 4.1 Results

**Functional Oracle setting.** We first assess different module selection strategies in a functional oracle setup. We present consolidated view of selection strategies in Fig. 2a. We observe that even in the functional oracle setting only the structural oracle baseline can achieve perfect specialization, which is also reflected in low comp. OOD loss. Second best is the "*Trans.* + *Sbtl.*" baseline which can achieve reasonably low OOD loss on average as well as good specialization. This result demonstrates that even if the knowledge about the sub-tasks is already perfectly encoded in the modules, learning how to combine these sub-task modules to solve the global task is a difficult problem on its own.

**Only the structural oracle minimizes compositional OOD loss.** Results are shown for the end-2-end learning setting in Fig. 2b. This is inline with [18], who use one-hot task descriptors identifying which module to use for each task (our oracle only get information about which variable to use for module selection, the latter is still learned, and not told directly which module to use). The good performance of Str. Oracle can be attributed ton the perfect specialization as shown in Fig. 2b (right). The second best selection strategy is *X-Atn.*, which also achieves second best OOD loss. All modular methods achieved OOD loss lower than that of the *Monolithic* baseline, which proofs the utility of modular solutions. We consider the loss (↓) of the monolithic baselines as an upper bound for good performance.

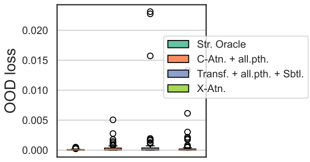


Figure 3: End-2-end training of test-time oracle.

**Poor OOD performance in end-2-end setting is due to sub-optimal module selection.** To find why modular methods fail to generalize compositionally OOD in the end-2-end regime we measure the compositional OOD loss in the so-called test-time oracle regime. In this regime we check every possible path in the modular network during the test time and report the minimal loss. This testing strategy assumes the ground truth knowledge for the test set. In Fig. 3 we see that for all modular solutions we can achieve very small loss in this regime. This suggests that functional components in these models have learned meaningful information and that the reason for bad performance in the non-oracle settings lies in the structural components responsible for routing. Given that good OOD loss can be achieved purely through recombination of existing modules, this approach can be useful for meta-learning and continual meta-learning settings like OSAKA [5].

**Better performance of *+all.pth.*** As expected, we observe in Fig. 8(a) that performing training in the + *all.pth.* regime, where each possible path is checked during each training step, results in better ID for all methods (with exception of the Str. Oracle).

**Bad specialization of *Transf.* baselines.** We observe in Fig. 8(c), that good OOD performance generally goes along with high specialization score, c.f. *X-Atn.* and *Str. Oracle* baselines. Nevertheless, there is a clear exception: *Transf.* + *all.pth.* baseline reaches OOD loss comparable to that of the *X-Atn.* + *all.pth.* baseline, while the later one reaches specialization score double as high. In fact, we observe that both *Transf.* + *all.pth.* and *Transf.* baselines tend to have very bad specialization while

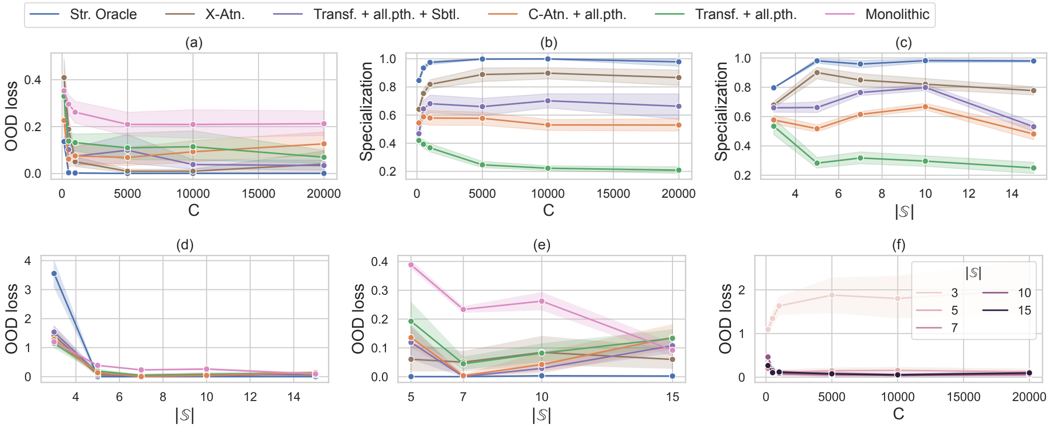


Figure 4: The role of number of points per operation $C$ and number of primitive operations $|\mathbb{S}|$.For ease of exposition, we reduced noise in the plots by using Grubbs' [11] test to remove outlier runs per application mode, $C$ and $|\mathbb{S}|$.

maintaining a high collapse score. After analysing the module activation patterns (c.f. examples Fig. 9) of these methods, we observe that these two methods activate modules much less sparsely than other methods making decision about which module to activate not only based on operations (vars $x_3$ and $x_4$) but also based on the digits ($x_1$ and $x_2$). This can be attributed to the fact that module selection is strongly over-parameterized for these baselines and misses any bottleneck. Hence, simply adding a selection bottleneck (*+Sbtl.*) improves specialization considerably.

**The role of the number of primitives $|\mathbb{S}|$ and training points per operation $C$.** We now turn our attention to the role of the number of primitives $|\mathbb{S}|$ and the number of training points per primitive $C$. We start with two hypotheses: (1) Increasing the number of primitives results in better compositional generalization ability of modular methods due to the increase in the number of tasks (i.e. number of combinations in which each primitive is seen); (2) Increasing number of training points per task can have a harmful effect on the compositional generalization ability. Intuitively, for large number of samples per operation, the bias towards solutions with specialized and reusable modules is weak. This is because enough training samples are given for achieving low training loss even when a separate module per primitive per task (as opposed to only per primitive) is learned. On the other hand, when the number of samples per primitive is limited, the number of training points may be not enough for achieving low training loss with solution without sharing, and hence, sharing modules across tasks can result in lower training loss and more bias towards learning reusable and sharing sub-tasks.

Regarding our first hypothesis, indeed in Fig. 4(d) we observe a reduction in the OOD loss as we increase the number of primitives $|\mathbb{S}|$ from 3 to 5 consistently for all methods emphasizing the importance of diversity of tasks for specialization and compositional generalization. Interestingly, for $|\mathbb{S}| > 5$ the effect of $|\mathbb{S}|$ is insignificant, and as we show in Fig. 4(e), OOD loss of some methods actually rises as more primitives are introduced, with only *Str. Oracle* reliably performing well. This can be attributed to the fact that increasing $|\mathbb{S}|$ also leads to more OOD tasks (unique combinations of primitves) in the OOD test set, which makes the OOD problem harder. Furthermore, in Fig. 4(c) we observe that increasing $|\mathbb{S}|$ can result in decrease in specialization score, a trend especially pronounced for $|\mathbb{S}|$ between 10 and 15 (except *Str. Oracle*). Hence, disentangling sub-tasks into specialized modules becomes harder when more sub-tasks are present in the underlying data.

We now turn our attention to the second hypothesis regarding the role of the number of training samples per operation $C$. We make the following observations in Fig. 4. (a) First, even for the monolithic architecture, that has no inductive bias favouring specialization and OOD generalization, increasing $C$ results in decreased OOD loss. This is inline with observations made in the context of large scale training [12, 1], where good amount of generalization and transfer is achieved through simply increasing the amount of training data[3]. (b) For *Str. Oracle* and *X-Atn.* we observe in Fig. 4(b)

[3] In many of these works however, it is not clear whether generalization is really compositional/systematic due to possibility of an overlap between the training and test set

that modular specialization improves as we increase $C$ (this is against our hypothesis), which is reflected in the tendency of decreasing OOD loss for larger $C$ (Fig. 4(a)). For the *C-Atn.*, *Transf. + all.pth.* and *Transf. + all.path. +Sbtl.* baselines, that generally do not result in good specialization and OOD loss, we observe an opposite trend: i.e. larger $C$ leads to worse specialization, and only for *C-Atn.* we observe the tendency to larger OOD loss with increased $C$ (Fig. 4(a)). (c) In Fig. 4(f) only for small $|\mathbb{S}|$ we observe the overall trend of growing IID loss with increased $C$ (in line with out hypothesis). This again emphasizes the point that if the data is not diverse enough (is the case for $|\mathbb{S}| = 3$), increasing $C$ worsens the performance. Overall, from observation (b) and (c) we can conclude that keeping $C$ low does not result into enough bias towards specialized solution, hence the main source of such bias must originate from the architectural/algorithm choice. On the other hand, if the algorithm under consideration does not provide enough inductive biases (e.g. in form of bottlenecks) towards specialized solution, increasing $C$ indeed worsens the performance.

### 4.2 Ablation: the role of functional bottleneck

Central to our modular methods to work is the functional module bottleneck: each module can only take a subset of variables as input which prevents collapse as we show next. In the experiments above, for the sake of simplicity we manually selected only $x_1$ and $x_2$ (i.e. the input digits) as inputs to the functional parts of the modules. Here we show that, similarly to NPS [9], this bottleneck can also be learned. In Fig. 5 we plot the collapse score (↑) for three application options (Str. Oracle, Transf., X-Attn). We test three types of functional bottlenecks: the "oracle" knowledge that only $x_1$ and $x_2$ should be selected as input to the functional component, a learned attention bottleneck that selects which two variable to take as input, and without the functional bottleneck at all. We find that across all three application modes, both oracle and attention bottlenecks perform similarly in terms of collapse. This shows that the functional bottleneck can be instantiated with a learnable key-value attention. Removing this bottleneck results in increasing collapse in all application modes.

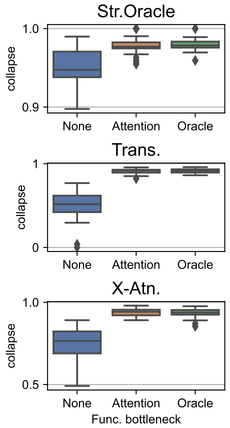


Figure 5: Ablation: functional bottleneck.

## 5 Conclusion

In this work we studied different attention-based module selection approaches for compositional modularity. We have shown that information bottlenecks play an important role for module specialization (functional bottleneck) and module selection (selection bottleneck). The main take-away is that in our experiments only the structural oracle baseline could reliably achieve good specialization, resulting in good compositional OOD performance. We also showed that poor performance of non-oracle solutions can be attributed to attention-based routing failing to correctly route samples at test time. These findings motivate application of these methods in settings where a small number of samples are available at test time to adapt the routing mechanism, or using self-supervised routing mechanisms that would facilitate test-time adaptation. Another important research direction is identifying and implementing information bottlenecks necessary for specialization in modular networks.

## References

[1] Samira Abnar, Mostafa Dehghani, Behnam Neyshabur, and Hanie Sedghi. Exploring the limits of large scale pre-training. *arXiv preprint arXiv:2110.02095*, 2021.

[2] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron Courville. Systematic generalization: What is required and can it be learned? In *International Conference on Learning Representations*, 2018.

[5] Massimo Caccia, Pau Rodriguez, Oleksiy Ostapenko, Fabrice Normandin, Min Lin, Lucas Caccia, Issam Laradji, Irina Rish, Alexande Lacoste, David Vazquez, et al. Online fast adaptation and knowledge accumulation: a new approach to continual learning. *arXiv preprint arXiv:2003.05856*, 2020.

[6] Robert M French. Pseudo-recurrent connectionist networks: An approach to the'sensitivity-stability'dilemma. *Connection Science*, 9(4):353–380, 1997.

[7] Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*, 2019.

[8] Anirudh Goyal, Alex Lamb, Phanideep Gampa, Philippe Beaudoin, Sergey Levine, Charles Blundell, Yoshua Bengio, and Michael Mozer. Object files and schemata: Factorizing declarative and procedural knowledge in dynamical systems. *arXiv preprint arXiv:2006.16225*, 2020.

[9] Anirudh Goyal, Aniket Didolkar, Nan Rosemary Ke, Charles Blundell, Philippe Beaudoin, Nicolas Heess, Michael Mozer, and Yoshua Bengio. Neural production systems. *CoRR*, abs/2103.01937, 2021. URL https://arxiv.org/abs/2103.01937.

[10] Anirudh Goyal, Aniket Didolkar, Alex Lamb, Kartikeya Badola, Nan Rosemary Ke, Nasim Rahaman, Jonathan Binas, Charles Blundell, Michael Mozer, and Yoshua Bengio. Coordination among neural modules through a shared global workspace. *arXiv preprint arXiv:2103.01197*, 2021.

[11] Frank E Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, 1969.

[12] Danny Hernandez, Jared Kaplan, Tom Henighan, and Sam McCandlish. Scaling laws for transfer. *arXiv preprint arXiv:2102.01293*, 2021.

[13] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[14] Cam Linke, Nadia M Ady, Martha White, Thomas Degris, and Adam White. Adapting behavior via intrinsic reward: A survey and empirical study. *Journal of Artificial Intelligence Research*, 69:1287–1332, 2020.

[15] Francesco Locatello, Michael Tschannen, Stefan Bauer, Gunnar Rätsch, Bernhard Schölkopf, and Olivier Bachem. Disentangling factors of variation using few labels. *arXiv preprint arXiv:1905.01258*, 2019.

[16] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. *Advances in Neural Information Processing Systems*, 33:11525–11538, 2020.

[17] Jorge A Mendez and Eric Eaton. Lifelong learning of compositional structures. *arXiv preprint arXiv:2007.07732*, 2020.

[18] Jorge A Mendez, Marcel Hussing, Meghna Gummadi, and Eric Eaton. Composuite: A compositional reinforcement learning benchmark. *arXiv preprint arXiv:2207.04136*, 2022.

[19] Sarthak Mittal, Yoshua Bengio, and Guillaume Lajoie. Is a modular architecture enough? *arXiv preprint arXiv:2206.02713*, 2022.

[20] Oleksiy Ostapenko, Pau Rodriguez, Massimo Caccia, and Laurent Charlin. Continual learning via local module composition. *Advances in Neural Information Processing Systems*, 34, 2021.

[21] Giambattista Parascandolo, Niki Kilbertus, Mateo Rojas-Carulla, and Bernhard Schölkopf. Learning independent causal mechanisms. In *International Conference on Machine Learning*, pages 4036–4044. PMLR, 2018.

[22] Nasim Rahaman, Muhammad Waleed Gondal, Shruti Joshi, Peter Gehler, Yoshua Bengio, Francesco Locatello, and Bernhard Schölkopf. Dynamic inference with neural interpreters. *Advances in Neural Information Processing Systems*, 34, 2021.

[23] Richard S Sutton, Marlos C Machado, G Zacharias Holland, David Szepesvari Finbarr Timbers, Brian Tanner, and Adam White. Reward-respecting subtasks for model-based reinforcement learning. *arXiv preprint arXiv:2202.03466*, 2022.

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

[25] Tom Veniat, Ludovic Denoyer, and Marc'Aurelio Ranzato. Efficient continual learning with modular networks and task-driven priors. *arXiv preprint arXiv:2012.12631*, 2020.

# A Modular network training

$$
\begin{aligned}
\log p(y|x) = \log \sum_Z p(y|x,z)p(z) = \sum_Z \log p(y|x,y) + \sum_Z \log p(z) = \\
= \sum_Z \log p(y|x,y)\frac{q(z)}{q(z)} + \sum_Z \log p(z) = \sum_Z \log p(y|x,y)\frac{q(z)}{q(z)} + \sum_Z \log p(z) = \\
= \log \mathbb{E}_q\Big(\frac{p(y|x,y)}{q(z)}\Big) + \sum_Z \log p(z) \geq \mathbb{E}_q\Big(\log \frac{p(y|x,y)}{q(z)}\Big) = \\
= \sum_Z q(z)\log p(y|x) - \mathbb{E}_q \log q(Z) = \mathbb{E}_q \log p(y|x) + \mathbb{H}\big(q(z)\big)
\end{aligned}
\tag{5}
$$

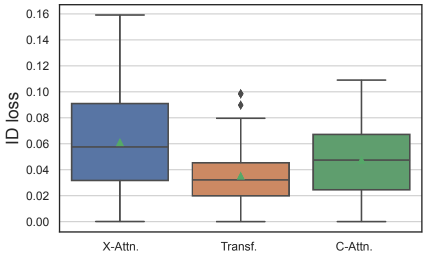
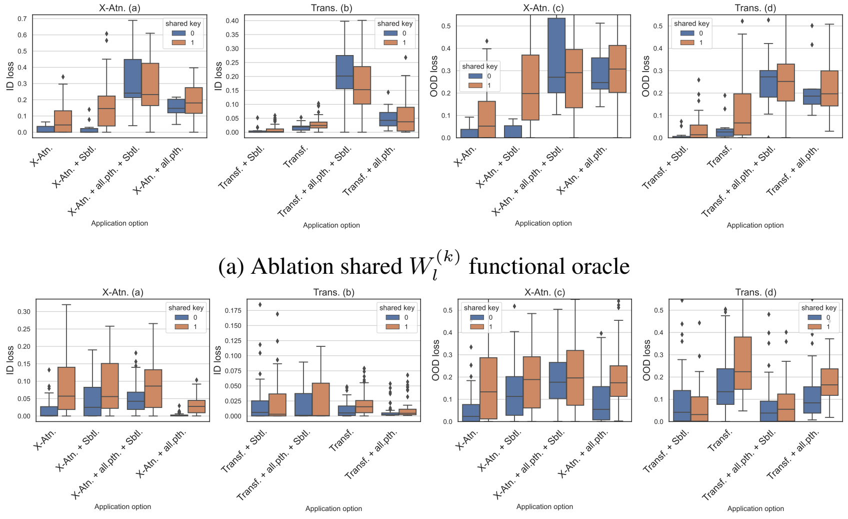


Figure 6: Ablation MOE on complex task selection: we apply MoE model with attention-based selection mechanism to tasks, where the information relevant for module selection is encoded in 2 inputs ($x_3$ and $x_4$). As expected, Trans. Encoder baseline outperforms the rest, and X-Attn. under-performs. This is because X-Attn. baseline encodes each variable separately for selection, while correct selection decision can only be made based on two vars.

# B Basic operations

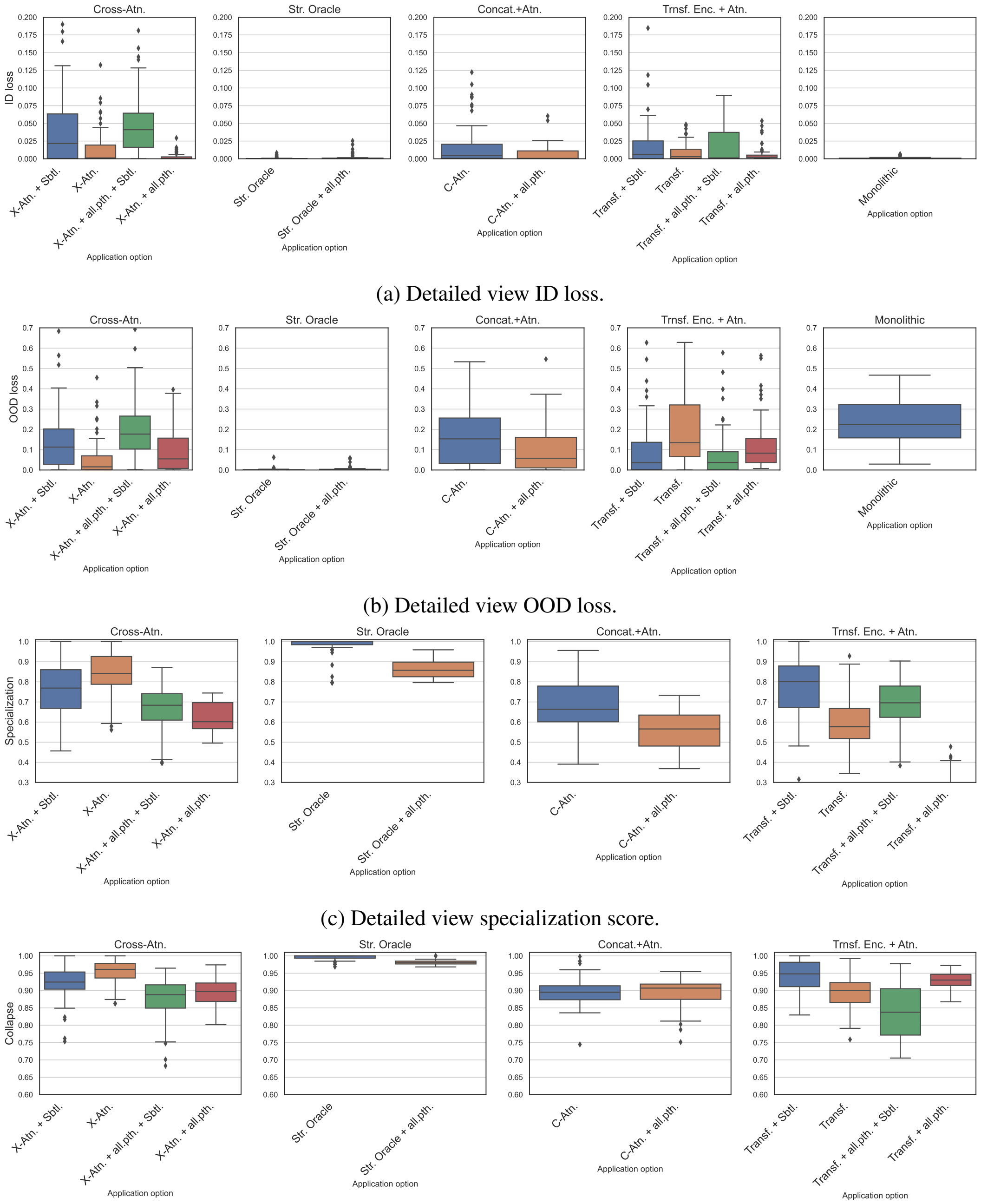
We sue the following list of basic expressions for training and testing data generation. For some input x and y: `x+y`, `x-y`, `x*y`, `x*x`, `y*y`, `(x+y)*x`, `min(x,y)`, `max(x,y)`, `(x+y)-x`, `(x+(2*y))`, `sin(x)`,`sin(y)`, `cos(x)`, `cos(y)`,`cos(x+y)`.



(a) Ablation shared $W_l^{(k)}$ functional oracle

(b) Ablation shared $W_l^{(k)}$ end2end

Figure 7: Ablation sharing $W_l^{(k)}$ in functional oracle (a) and end2end training settings (b). No sharing tends to result in lower loss. $C \in [1000, 5000, 10000, 20000]$, $|\mathbb{S}| \in [5, 7, 10, 15]$, 5 seeds.



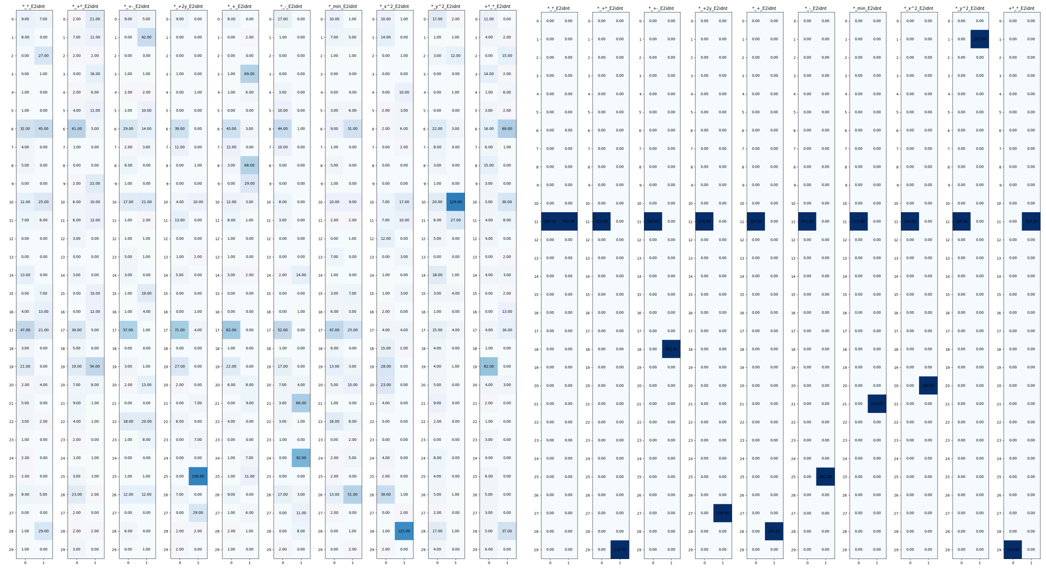
(a) Detailed view ID loss.

(b) Detailed view OOD loss.

(c) Detailed view specialization score.

(d) Detailed view collapse score.

Figure 8: Detailed results for ID (a) and OOD (b) loss for the end2end regime. $C \in [1000, 5000, 10000, 20000]$, $|\mathbb{S}| \in [5, 7, 10, 15]$, 5 seeds.



(a) Trans. + all.pth. (end2end)

(b) X-Atn. (end2end)

Figure 9: We plot test-time module activation patterns for selected runs and a subset of 10 tasks for *Trans. + all.pth.* (a) and *X-Atn*(b) methods. This runs use $|\mathbb{S}| = 10$. Both runs result in rather high collapse score (↑), $\sim .91$ in (a) and $\sim .98$ in (b), meaning that the system does not collapse to using the same modules for all operations. On the other hand, (a) suffers from low specialization score, $\sim .25$ in (a) and $\sim .92$ in (b).