1 SUPPLEMENTARY MATERIAL

1.1 CODE

We provide an executable python script that runs a trial of the algorithmic experiment (section 5.3.), which is the only experiment that uses normal training methods. We also provide pytorch modules for the Multiplexer, the FNNR, the MFNNR, and the SMFR.

The code we provide is not optimized for performance because we favored readability over speed. The modules use lists of tensor-blocks that we loop over instead of combining all blocks into a single tensor and using a single more complicated matrix operation. The code could be sped up significantly by refactoring the lists of block-tensors into single larger tensors. Our modules are written the way they are because we applied debugging and analysis code that is easier to interpret in this format than for runtime-optimized code.

1.2 EXPERIMENT DETAILS, GENERAL

For all of our experiments, we used an Adam optimizer with a learning rate of 3e-4 and gradient clipping at 0.1. We used LeakyRelu with a negative slope of 0.01 as the activation function. The FNNs contained inside SMFRs had a width of 100 and variable depth.

We used single-digit numbers in the addition/multiplication experiment and the double-addition experiment. We ran some exploratory tests with two digits as well and got similar results. We focused our analysis on single digit experiments because those converged faster, so we could run more trials.

1.2.1 EXPERIMENT DETAILS: ADDITION/MULTIPLICATION EXPERIMENTS

We performed grid search over the following parameters:

- The threshold value before switching training regimes: 0.7, 0.8, 0.9, 0.95, 1.0
- For FNNs
    - Layer widths: 100, 200, 300
    - Layer depths: 2, 3, 4
    - Random seeds: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- For SMFRs
    - Stack widths: 1, 2, 3, 4, 5, 6, 7
    - Stack depths: 1, 2, 3
    - Contained FNN layer depths: 1, 2, 3
    - Attention type: Softmax, Straight Through Gumbel-Softmax

This resulted in model sizes in the range of roughly 13k to 280k parameters for both FNNs and SMFRs.

Table 1 shows the effect of model size on the accuracy after catastrophic interference. HIGH refers to all models with at least 200,000 parameters, LOW refers to models with less than 50,000 parameters, and MID refers to the rest. The table shows that FNNs perform better at higher model sizes. It should be noted that the model sizes of FNNs and SMFRs were varied in different ways. The table indicates that growing SMFRs by stacking more MFNNR modules together is not as effective as increasing the FNNs inside the SMFRs.

Table 2 shows that SMFRs with more complex FNNs inside them performed better, especially at low thresholds. However this also had diminishing returns at 3 layers.

Architecture Optimization. Note that we varied the model sizes of FNNs and SMFRs in different ways, and this may explain some of our findings. Our tests only included small variations on this because we focused on varying the number of blocks and MFNNR modules instead of the complexity of each FNN contained within them. Additionally, we only altered the depths of the contained FNNs, not their width as well, as we did for the baseline FNNs that weren't part of an MFNNR. For FNNs, altering the width made a large difference. Increasing the width of the FNNs inside the SMFRs as

Table 1: Effects of model size on performance

Threshold	Architecture	Softmax/Gumbel	$HIGH$	MID	LOW
0.7	FNN		0.054	0.024	0.035
0.7	SMFR	Softmax	**0.204**	0.146	0.107
0.7	SMFR	Gumbel	0.146	0.094	0.049
0.8	FNN		0.089	0.053	0.052
0.8	SMFR	Softmax	0.186	**0.205**	0.116
0.8	SMFR	Gumbel	0.118	0.138	0.063
0.9	FNN		0.179	0.134	0.098
0.9	SMFR	Softmax	**0.300**	0.274	0.198
0.9	SMFR	Gumbel	0.266	0.232	0.109
0.95	FNN		0.325	0.223	0.150
0.95	SMFR	Softmax	0.346	**0.349**	0.245
0.95	SMFR	Gumbel	0.274	0.333	0.172
1.0	FNN		**0.468**	0.413	0.259
1.0	SMFR	Softmax	0.450	0.448	0.382
1.0	SMFR	Gumbel	0.408	0.415	0.327

Table 2: Complexity of FNNs within MFNNRs

Threshold	1 hidden layer	2 hidden layers	3 hidden layers
0.7	0.099	0.134	**0.192**
0.8	0.125	**0.216**	0.210
0.9	0.194	**0.299**	0.284
0.95	0.275	**0.365**	0.338
1.0	0.365	**0.489**	0.449

well would have led to much larger SMFR models, which is why we did not test this. Therefore, the reason that Pure FNNs performed slightly better than SMFRs at high model sizes and thresholds may be because we grew the complexity of SMFRs in a suboptimal way. It remains future work to test if tuning the FNNs inside the SMFRs gives even better results than varying the number of blocks and MFNNR modules.

1.2.2 EXPERIMENT DETAILS: DOUBLE-ADDITION EXPERIMENTS

We performed grid search over the following parameters:

- Type of split for the training data: high vs. low, even vs. odd
- For FNNs
  - Layer widths: 100, 200, 300
  - Layer depths: 2, 3, 4
  - Random seeds: 0, 1, 2, 3, 4
- For SMFRs
  - Stack widths: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
  - Stack depths: 0, 1, 2
  - Contained FNN layer depths: 1, 2
  - Attention type: Softmax, Straight Through Gumbel-Softmax

Module size between FNN and SMFR is largely irrelevant here because we noticed early on that all FNNs fail.

Table 3 shows all 25 architectures with an OOD of 1.0, because a detailed breakdown over several dimensions is much more confusing to look at. This shows that model size correlates with performance.

Table 3: Best architectures for the double-addition task

alternate split	Attention type	depth	width	FNN layers	model size
True	soft	1	9	2	95364
True	soft	1	5	1	36096
False	soft	1	9	1	54964
False	soft	1	8	1	50247
False	soft	1	10	1	59681
False	soft	1	7	1	45530
False	gumbel	1	8	1	50249
False	soft	1	6	1	40813
True	soft	1	10	1	59681
True	soft	1	9	1	54964
True	soft	1	8	1	50247
True	soft	1	7	1	45530
True	soft	1	6	1	40813
False	gumbel	1	10	1	59683
True	soft	2	10	1	111091
True	gumbel	1	6	1	40815
False	gumbel	1	7	1	45532
False	gumbel	1	6	1	40815
True	soft	1	6	2	81213
True	soft	1	8	2	90647
True	gumbel	1	9	1	54966
True	soft	1	10	2	100081
True	soft	2	9	2	160944
True	soft	2	5	2	119976
False	gumbel	1	9	1	54966

However, bigger models are not always better because the number of FNN layers was often 1 instead of 2, even though this parameter has a large impact on the model size.

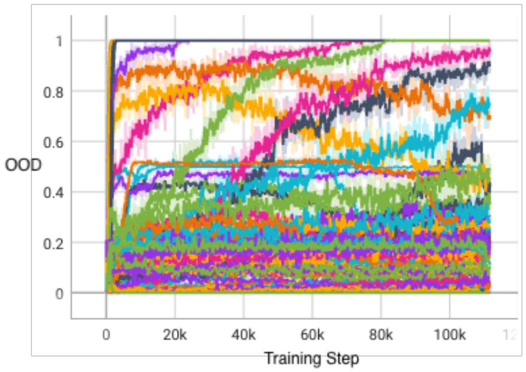

Figure 1: Each line shows the OOD accuracy of one SMFR experiment as training progresses.

Figure 1 shows examples of the development of the OOD accuracy over time for different SMFR architectures. Each line represents one training run. The colors are meaningless and are just there to make it possible to tell the lines apart. Most experiments with a perfect OOD accuracy converged to 1.0 almost immediately. It happened more often that OOD accuracy increased than that it dropped. Perhaps most importantly, **the OOD accuracy never dropped after reaching 1.0**. In other words, the SMFR usually came to reuse modules more rather than less as the training progressed and remained stable once converged. This suggests that module reuse will also occur if SMFRs are used as components inside larger architectures that train for longer.

1.2.3 EXPERIMENT DETAILS: ALGORITHMIC EXPERIMENTS

We performed grid search over the following parameters:

- Increment the value after applying the formula of the task: True, False
- For FNNs
  - Layer widths: 100, 200, 300
  - Layer depths: 2, 3, 4, 5, 6
  - Random seeds: 0, 1
- For SMFRs
  - Stack widths: 6, 8, 10
  - Stack depths: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
  - Contained FNN layer depths: 1, 2, 3

This resulted in model sizes in the range of roughly 20k to 500k parameters for both FNNs and SMFRs.

Table 4: Effect of model size on OOD performance

Architecture	model size category	OOD accuracy on iteration 1
FNN	high	0.229
FNN	mid	0.211
FNN	low	0.157
SMFR	high	0.889
SMFR	mid	0.975
SMFR	low	0.414

Table 3 shows that FNNs again benefitted more from larger model sizes than SMFRs, and SMFRs reached their best performance at lower model sizes. However, even the largest FNNs we tested performed much worse than the SMFRs on OOD iterations.

1.3 REGULARIZATION LOSS

As mentioned in the Limitations, our model used to suffer from a stability issue that we have since fixed. The problem was that softmax and sigmoid weights sometimes kept growing even after becoming extreme enough that they had 0 gradient. Our inspection showed that this was caused by the fact that the softmax and sigmoid values are based on computations that are also used for generating data in the MFNNRs. This means that they are subject to destructive interference. They effectively receive gradients from two sources: The softmax/sigmoid they are supposed to work on and incidental gradients from other parts of the model, but the former irrecoverably become zero.

We fixed this by adding a regularization loss: Whenever the absolute of the weight used for a softmax or sigmoid exceeds a threshold value, we apply an MSE-loss to that weight, pushing it back to the threshold. The target value of the MSE-loss is simply the tensor itself with each neuron clamped to a threshold value. We found that a threshold of 20 works well empirically.

It should be noted that we introduced the regularization loss while working on a larger, more complex experiment that we will report in a subsequent paper. It is possible that this loss is not needed for many of the experiments we report in this paper. We have noticed that using it slows convergence but increases reliability. We have opted to always use it because reliability is more important to us than convergence speed while developing new architectures. As mentioned above, there is a lot of room for improvement in speeding up our code that we haven't done, yet.

After submitting this paper but while creating reproducible code as supplementary material, we noticed that not using the regularization loss appeared to give better results in the algorithm task from Experiments section 4.3, although we did not run enough trials yet to be certain. We will investigate this more deeply in future experiments to determine under what circumstances the regularization loss is needed and under what circumstances it is detrimental.

1.4 INTERPRETABILITY

To inspect an SMFR model, it is often enough to look at the weights that determine routing decisions: The softmax values in the Multiplexer, and the gating weight in the FNNR. It can be helpful to have a set of example inputs for which you expect different routing behavior and to compare the routing weights for these.

On the double-addition task, our inspection showed the following: **It is possible to measure the degree of modularity in our SMFR directly by investigating the softmax weights on example inputs**. We manually inspected the values of the weights that the multiplexers and FNNR modules use in a trial that achieved good OOD performance and had a simple architecture, with depth 1 and width 2. The architecture had only 12 Multiplexer weights and 3 FNNR residual weights, which made it very easy to investigate how data is routed in the network for any given input. Our inspection showed that the irrelevant inputs did correctly receive a weight of 0, while the ones that are important received a weight of 0.5. As it turns out, the network learned to use the correct pair of numbers and used interpolation in a single block to take advantage of symmetry. Instead of always moving block $A1$ to $B1$ and $A2$ to $B2$ as a human would intuitively do it, it learned to interpolate $B1 = 0.5 * A1 + 0.5 * A2$, which is actually more efficient. We also found that the gating weights of several FNNR modules approached zero as training progressed. This implied that the network came to rely more and more on copy operations and less on FNN operations. It was quite easy to inspect the network and learn this information, since we only needed to inspect the gating weights. In larger networks used for practical tasks, the number of network weights may be too large for a manual analysis. However there should still be few enough of them for an automated analysis to yield useful results, especially since the gating weights have inherent meaning (does a block of data get used or not?), unlike normal neurons in a fully-connected neural network, which can mean anything.

On the algorithmic task, our inspection confirmed that the SMFRs can learn to pass blocks through without altering them where doing so is useful, and to alter only the ones that need to be altered. We were able to confirm this desired behavior by only looking at the routing weights. In an FNN, confirming such behavior would have required a much more complex correlation analysis of the neurons.