Part I

Appendix

A PSEUDO CODES

Alg. 1 shows the integration of shared workspace with RIMs (Goyal et al., 2019). We replace the direct module to module interaction via attention in RIMs, with shared workspace. Specialists compete to write in the shared workspace, and the contents of the workspace are broadcasted to all the specialists.

Alg. 2 shows the integration of the shared workspace with TIMs (Lamb et al., 2021). Again we replace the direct module to module communication in TIMs, with a shared workspace.

Algorithm 1: Shared Workspace integration with RIMs

Input: Current sequence element, $\boldsymbol{x}_t$ and previous state of the specialist, $\{\boldsymbol{h}_{t-1,k}\}$, for $k \in \{1, \ldots, n_s\}$ and structure of memory as a matrix M with row wise compartmentalized memories, where m_i refers to the state of slot i (total number of slots is n_m).

Step 1: Process image by position p with fully convolutional net

- $\boldsymbol{c}_p = [\mathrm{CNN}(\boldsymbol{x}_t)]_p$
- $\boldsymbol{z}_t = [\boldsymbol{c}_p \; \boldsymbol{e}_p]$ *(concatenate encoding of position to CNN output)*

Step 2: Specialists compete to be selected to update the workspace based on current input

- $\boldsymbol{q}_k = \boldsymbol{h}_{t-1,k}\boldsymbol{W}^q$
- $s_k = \mathrm{softmax}\big(\frac{\boldsymbol{q}_k \boldsymbol{\kappa}}{\sqrt{d_e}}\big)$, where $\boldsymbol{\kappa} = (\boldsymbol{z}_t \boldsymbol{W}^e)^{\mathrm{T}}$
- Construct a set $\mathcal{F}_t$ which contains the indices of the n_{sel} specialists that have the largest s_k
- $\bar{\boldsymbol{h}}_{t,k} = \begin{cases} g_k\left(s_k \boldsymbol{z}_t \boldsymbol{W}^v, \boldsymbol{h}_{t-1,k}\right) & k \in \mathcal{F}_t\,, \\ \boldsymbol{h}_{t-1,k} & k \notin \mathcal{F}_t\,, \end{cases}$
- $\boldsymbol{a}_k = s_k \boldsymbol{z}_t \boldsymbol{W}^v \; \forall \; k \in \mathcal{F}_t$ (Scaled Dot Product Attention)

Step 3: Activated specialists write in a shared workspace

- $\widetilde{\boldsymbol{Q}} = \boldsymbol{M}\widetilde{\boldsymbol{W}}^q$
- $\boldsymbol{R} = [\boldsymbol{M}; \boldsymbol{A}]$ where $\boldsymbol{A}$ is the matrix whose rows are the $\boldsymbol{a}_k \forall k \in \mathcal{F}_t$
- $\boldsymbol{M} \leftarrow \mathrm{softmax}\left(\frac{\widetilde{\boldsymbol{Q}}(\boldsymbol{R}\widetilde{\boldsymbol{W}}^e)^{\mathrm{T}}}{\sqrt{d_e}}\right)\boldsymbol{R}\widetilde{\boldsymbol{W}}^v$

Step 4: Broadcast of information from the shared workspace

- $\widehat{\boldsymbol{q}}_k = \bar{\boldsymbol{h}}_{t,k}\widehat{\boldsymbol{W}}^q \quad \forall k \in \{1, \ldots, n_s\}$
- $s_{k,j} = \mathrm{softmax}\left(\frac{\widehat{\boldsymbol{q}}_k \widehat{\boldsymbol{\kappa}}_j}{\sqrt{d_e}}\right)$ where $\widehat{\boldsymbol{\kappa}}_j = (\boldsymbol{m}_j \widehat{\boldsymbol{W}}^e)^{\mathrm{T}} \quad \forall k \in \{1, \ldots, n_s\},\; j \in \{1, \ldots, n_m\}$
- $\boldsymbol{h}_{t,k} = \bar{\boldsymbol{h}}_{t,k} + \sum_j s_{k,j}\widehat{\boldsymbol{v}}_j$ where $\widehat{\boldsymbol{v}}_j = \boldsymbol{m}_j \widehat{\boldsymbol{W}}^v \quad \forall k \in \{1, \ldots, n_s\}$

B HYPERPARAMETERS

Table 3 lists the different hyper-parameters.

Parameters in RIMs+SW:

RIMs with shared workspace has three set of parameters:

- **Parameters corresponding to Input attention** Parameters for the attention for the k-th specialist $\theta_k = (W_k^q, W^e, W^v)$ corresponding to query, keys, and values respectively. Each specialist has different query parameters but share the same keys and values (which are function of the input). In the table it corresponds to the inp keys, inp values, inp heads respectively.
- **Writing in a shared workspace:** Parameters corresponding to the writing in the memory. Here, we follow the similar mechanisms as in RMC(Santoro et al., 2018), where shared

Algorithm 2: Shared Workspace integration with TIMs

Notation: Consider $\boldsymbol{h}_l$ as the output of the l^{th} transformer layer. Let sequence length of original input be T and embedding dimension of transformer be D. Let the transformer be composed of n_b mechanisms and memory be denoted as a matrix M with row wise compartmentalized memories, where m_i refers to the state of slot i (total number of slots is n_m). Consider $\boldsymbol{h}_l^k = \boldsymbol{h}_l[:, (k-1)D/n_b : kD/n_b]$ to be the hidden state of mechanism indexed k at layer l.

Initialization: Convert the raw input $X \in \mathbb{R}^{T \times vocab\_size}$ to
$\boldsymbol{h}_0 = positional\_encoding + Embedding(X)$ where $\boldsymbol{h}_0 \in \mathbb{R}^{T \times D}$. Initialize memory matrix M which remains common for all layers in the transformer.

Input to the layer l: $\boldsymbol{h}_{l-1}$ having shape $\mathbb{R}^{T \times D}$

Step 1: Mechanisms compete to be selected to update the workspace based on the input they receive from the previous layer

- $W^c \in \mathbb{R}^{D/n_b \times 1}$
- $c_k = \boldsymbol{h}_{l-1}^k W_k^c \quad \forall k \in \{1, \ldots, n_b\}$
- $c = softmax(concat(c_1, .., c_{n_b}))$, $c \in \mathbb{R}^{T \times n_b}$
- For each time step t in the original sequence of length T, we use the soft score c to select the top n_{sel} mechanisms which would self-attend and write to the memory. Hence generating set $\mathcal{F}_t$ which stores the indices of n_{sel} mechanisms for position $t \in \{1, 2, ..., T\}$. Also construct $c_k^* \in \mathbb{R}^{T \times D/n_b}$ where

$$c_k^*[t, :] = \begin{cases} c[t][k] & k \in \mathcal{F}_t \,, \\ 0 & k \notin \mathcal{F}_t \,, \end{cases}$$

Step 2: Selected mechanisms self-attend and update their hidden state

- $residual_k = \boldsymbol{h}_{l-1}^k$
- $\bar{\boldsymbol{h}}_l^k = c_k^* \odot SelfAttention(\boldsymbol{h}_{l-1}^k) + residual_k \quad \forall k \in \{1, \ldots, n_b\}$

Step 3: Selected mechanisms write on the shared workspace

- Memory matrix M was last modified by mechanisms of layer $l-1$
- Let $\boldsymbol{a}_k = c_k^* \odot \bar{\boldsymbol{h}}_l^k$ and $\boldsymbol{a} = concat(\boldsymbol{a}_1, .., \boldsymbol{a}_{n_b})$. Absorb the first dimension (corresponding to position in the sequence) in the batch dimension by reshaping $\boldsymbol{a}$. Perform the same steps as in algorithm 1.
- $\widetilde{\boldsymbol{Q}} = \boldsymbol{M}\widetilde{\boldsymbol{W}}^q$
- $\boldsymbol{R} = [\boldsymbol{M}; \boldsymbol{A}]$ where $\boldsymbol{A} = \boldsymbol{a}\boldsymbol{W}^v$
- $\boldsymbol{M} \leftarrow \text{softmax}\left(\frac{\widetilde{\boldsymbol{Q}}(\boldsymbol{R}\widetilde{\boldsymbol{W}}^e)^{\mathrm{T}}}{\sqrt{d_e}}\right)\boldsymbol{R}\widetilde{\boldsymbol{W}}^v$

Step 4: Broadcast of information from the shared workspace

- Reshape the new memory to bring back the sequence dimension. Perform the same steps as in algorithm 1.
- $\widehat{\boldsymbol{q}}_k = \bar{\boldsymbol{h}}_l^k \widehat{\boldsymbol{W}}^q \quad \forall k \in \{1, \ldots, n_b\}$
- $s_{k,j} = \text{softmax}\left(\frac{\widehat{\boldsymbol{q}}_k \widehat{\boldsymbol{\kappa}}_j}{\sqrt{d_e}}\right)$ where $\widehat{\boldsymbol{\kappa}}_j = (\boldsymbol{m}_j \widehat{\boldsymbol{W}}^e)^{\mathrm{T}} \quad \forall k \in \{1, \ldots, n_b\},\ j \in \{1, \ldots, n_m\}$
- $\boldsymbol{h}_l^k = \bar{\boldsymbol{h}}_l^k + \sum_j s_{k,j} \widehat{\boldsymbol{v}}_j$ where $\widehat{\boldsymbol{v}}_j = \boldsymbol{m}_j \widehat{\boldsymbol{W}}^v \quad \forall k \in \{1, \ldots, n_b\}$

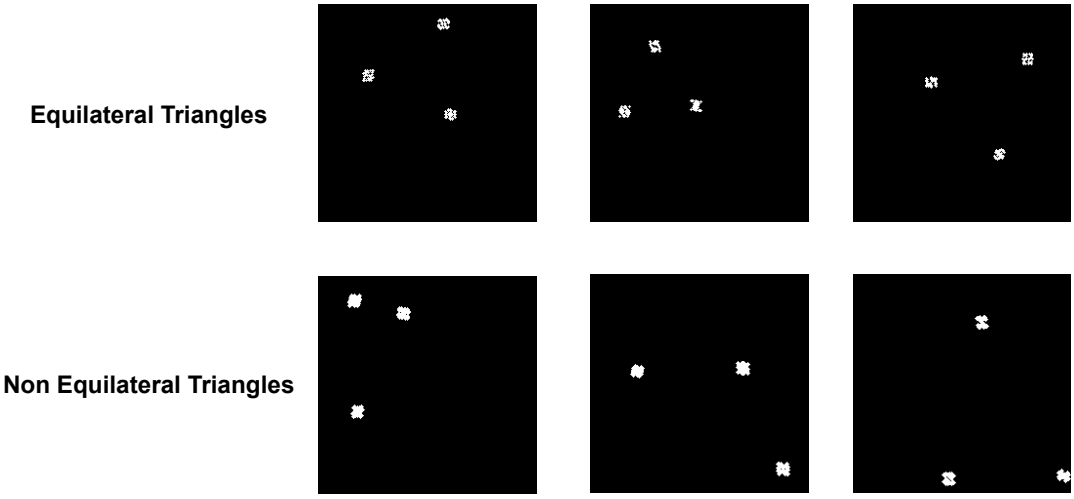

Figure 6: A demonstration of the detecting equilateral triangles task.

Parameter	Value
Number of specialists (n_s)	6
Size of each specialist	85
Number of memory slots (n_m)	
Optimizer	Adam(Kingma and Ba, 2014)
learning rate	$1 \cdot 10^{-4}$
batch size	64
Inp keys	64
Inp Values	85
Inp Heads	4
Inp Dropout	0.1
Number of memory slots	4
Number of memory heads	1
Size of attention head	32
Key size	32
Number of MLP layers in Attention	3
Gate Style	'unit'
Memory Attention Heads	4
Memory Attention keys	32
Memory Attention Values	32

Table 3: Generic Hyperparameters for the proposed model (for RIMs)

workspace is seen as a Matrix with row wise compartmentalized memories (i.e slots) i.e $\widetilde{\boldsymbol{W}}^q$, $\widetilde{\boldsymbol{W}}^e$, $\widetilde{\boldsymbol{W}}^v$. In the table it corresponds to number of memory slots, number of memory heads, size of attention head, key size and number of mlp layers in attention. These are the same hyper-paramter as in RMC (Santoro et al., 2018). We tried two different set of hyper-parameters (a) where we only have a single slot and (b) where we have 4 slots.

- **Broadcast of Information from the shared workspace**: In this process, the information in the workspace gets broadcasted to all the specialists such that each specialist produces a query, and the keys and values are a function of the memory state. Each specialist gets information from the memory according to its query, and this information is used to update the state of each specialist in a residual fashion. This corresponds to the parameters of $\widehat{\boldsymbol{W}}^v$, $\widehat{\boldsymbol{W}}^q$, $\widehat{\boldsymbol{W}}^e$ in the table i.e memory attention heads, memory attention keys, and memory attention values. We did not do any hyper-parameter search for these hyper-parameters.

Resources Used:

- For vision tasks like Sort-of-clever, Equilateral triangle, CIFAR classification, it takes about 6 hours to run 200 epochs on V100 (32G) GPU.
- It takes about 2 days to train the proposed model on bouncing ball task for 100 epochs on V100 (32G) GPU. We did not do any hyper-parameter search specific to a particular dataset (i.e 4Balls or 678Balls or Curtain Task). We ran the proposed model for different number of memory slots (i.e 2/4/8) for all the different datasets.
- For Starcraft task, it takes about 5 days to train on V100 (16G) GPU with batch size of 4.

C Implementation Details

Writing Information in the shared workspace. While writing information to the shared workspace, we update the workspace using a gating mechanism as proposed in Santoro et al. (2018). The gating mechanism consists of *input* and *forget* gates. Let $\boldsymbol{M}^{t-1}$ and $\boldsymbol{M}^{t}$ be the previous and updated memory matrix respectively. Let $\boldsymbol{M}$ be the result of the attention mechanism as described in step 2 of section 2.1. Let $\boldsymbol{X}_{1\ldots n_s}$ be the input to n_s specialists. The gating mechanism can be formulated as follows.

$$
\begin{aligned}
\bar{\boldsymbol{X}} &= \frac{1}{n_s}\sum_{i=1}^{n_s} \text{relu}(\boldsymbol{X}_i \times \boldsymbol{W}^1) \\
\boldsymbol{K} &= \bar{\boldsymbol{X}} + \tanh(\boldsymbol{M}^{t-1}) \\
\boldsymbol{I} &= \text{sigmoid}(\boldsymbol{K}\boldsymbol{W}^I) \\
\boldsymbol{F} &= \text{sigmoid}(\boldsymbol{K}\boldsymbol{W}^F) \\
\boldsymbol{M}^t &= \boldsymbol{I} \times \tanh(\boldsymbol{M}) + \boldsymbol{F} \times \boldsymbol{M}^{t-1}
\end{aligned}
$$

Here, $\boldsymbol{I}$ and $\boldsymbol{F}$ indicate the input and forget gates respectively. Note that $\boldsymbol{W}^1$ is shared across all n_s specialists.

D Properties of Shared Workspace

In section 2, we claim that higher-order interaction terms and effects due to persistence of memory are key contributors to Shared Workspace performance. We support those claims here:

Shared Workspace vs repeated self attention Higher-order interaction can be simulated by repeating the self-attention step multiple times at the same layer/time-step. However, due to the absence of a global communication channel, there is no constraint that the messages passed among the neural modules should lie in the same representation space. We modify a standard transformer where we repeat the self-attention step two times in every layer. We expect that $2\times$Self Attention will perform worse than SW. We also run a model where both self-attention as well as shared workspace is used by the transformer to update its state.

Persistence of Memory To check whether persistence is crucial for our model to perform well, we run a model where we re-initialize the shared workspace at every layer. Again we expect that removing memory persistence should result in a drop in performance and speed of convergence.

We run these models on sort-of-clevr dataset and present the results in figure 7

We note that removing persistence of memory results in significantly slower convergence. Replacing SW with $2\times$SA results in a significant drop in performance.

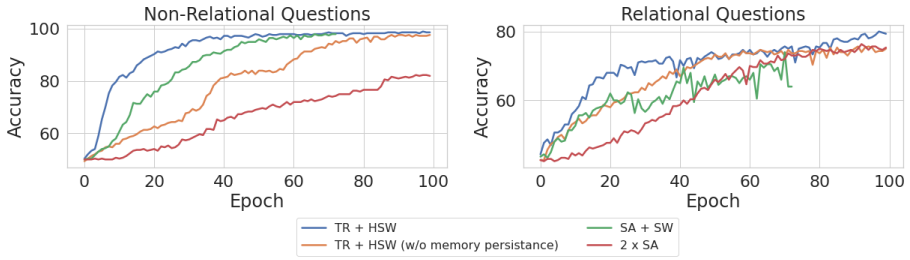

Figure 7: **Comparison on Sort-of-CLEVR relational reasoning**. Speed of convergence for relational and non-relational questions in the sort-of-clevr dataset. We can see that the Shared Workspace model converges faster and generalizes better as compared to all the other models. Here SW refers to shared workspace, 2×SA refers to applying self-attention twice in the same layer, SW+SA refers using both Shared Workspace and Self Attention in each transformer layer.

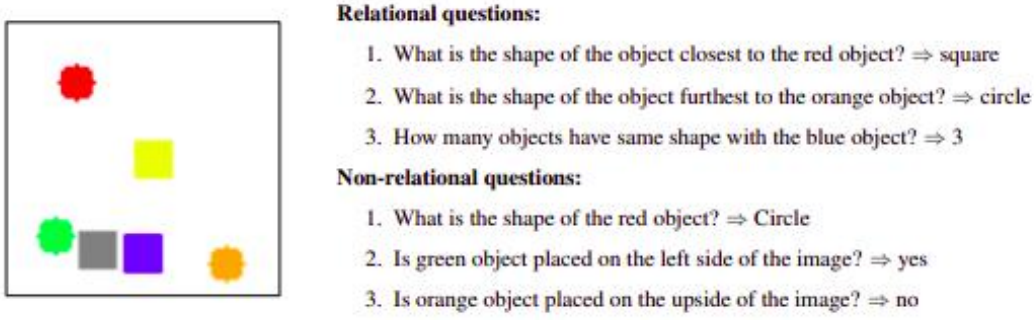

Figure 8: A sample from the sort-of-clevr dataset.

E TRANSFORMER TASKS

E.1 DETECTING EQUILATERAL TRIANGLES

A demonstration of this task can be found in figure 6. We use images of size 64×64 for this task. Our training dataset consists of 50000 examples and we evaluate on 10000 examples. We follow the same setup as vision transformers Dosovitskiy et al. (2020) for this task. We divide the image into patches of size 4×4, this sequence of patches is fed as input to a 4-layered transformer along with the CLS token which is used for classification. We set hidden dim to 256 and ffn dim to 512. For the proposed model (TR+SSW, TR+HSW), We use a query and key size of 32, and value size of 64. We use 4 heads during reading from and writing into the shared workspace which consist of 8 memory slots. For the baseline models (TR, TR + HC, STR), we use query, key and value size of 64 and 4 heads. For training, we use a batch size of 64. We train the model for 200 epochs using Adam optimizer with a learning rate of 0.0001. We anneal the learning rate using cosine annealing.

E.2 SORT-OF-CLEVR

Figure 8 shows a sample from this dataset. The images in this dataset are of size 75×75. Each question is encoded into 11 bits. The first 6 bits indicate color, the next 2 bits indicate question type (relational or non-relational), and the remaining 3 bits indicate question subtype (according to figure 8). We use a 4-layered transformer for this task with hidden dim set to 256 and ffn dim set to 512. For the proposed model (TR+SSW, TR+HSW), We use a query and key size of 32, and value size of 64. We use 4 heads during reading from and writing into the shared workspace which consists of 8 memory slots. For the baseline models (TR, TR + HC, STR), we use query, key and value size of 64 and 4 heads. We encode the 11 bit question into a 256 dimensional vector representation and concatenate it with the sequence of 15×15 sized patched obtained from the image.

We use the representation corresponding to the CLS token for classification. We train the model using cross-entropy loss. We use a batch size of 64 and train the model for 100 epochs. We use Adam optimizer with a learning rate of 0.0001 for training.

E.3 CATER: OBJECT TRACKING

Each CATER video consists of about 300 frames of size 224×224. We first sample frames at a sampling rate of 6 which results in 50 frames. From these 50 frames, we stack 5 consecutive frames together and pass each stack through a 18 layered resnet. The corresponding sequence of 10 frames is passed as input to the transformer. This task is setup as a classification task where we have to predict which cell in the 6×6 grid contains the snitch in the final frame. We use a 6-layered transformer with hidden dim set to 512 and ffn dim set to 2048. For the proposed model (TR+SSW, TR+HSW), We use a query and key size of 32, and value size of 64. We use 8 heads during reading from and writing into the shared workspace which consists of 8 memory slots. For the baseline models (TR, TR + HC, STR), we use query, key and value size of 64 and 8 heads.

F RIMS TASKS

F.1 BOUNCING BALL

The dataset consists of 50,000 training examples and 10,000 test examples showing $\sim$50 frames of either 4 solid balls bouncing in a confined square geometry (*4Balls*), 6-8 balls bouncing in a confined geometry (*678Balls*), 3 balls bouncing in a confined geometry with an occluded region (*Curtain*), or balls of different colors (*Colored 4Balls*) and (*Colored 678Balls*). We train baselines as well as the proposed shared workspace extension (e.g., RIMs + SW). As shown in Fig. 9, we study the performance of the proposed model compared with LSTM, RIMs and RMC. The first 10 frames of ground truth are fed in and then the system is rolled out for the next 35 time steps. During the rollout phase, the proposed method performs better than the baselines in accurately predicting the dynamics of the balls as reflected by cross entropy (CE).

We trained baselines as well as proposed model for about 100 epochs. We use the same architecture for encoder as well as decoder as in (Van Steenkiste et al., 2018). Hyper-parameters specific to the proposed architecture are listed in Tab. 3.

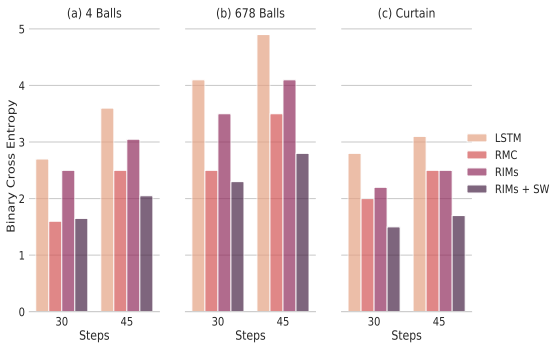

Figure 9: **Bouncing ball motion:** Prediction error comparison of the proposed method, LSTM, RIMs and RMC baseline. Given 10 frames of ground truth, the model predicts the rollout over the next 35 steps. Here, we present the BCE for the 30th frame and 45th frame. The *proposed SW extension performs better than other baselines in accurately predicting the dynamics, with an increasing advantage as the number of unrolled steps (30 vs 45) and balls ((a) vs (b)) increases*. Results are an average over 5 random seeds.

G INTEGRATING SW WITH MORE ARCHITECTURES

G.1 TIMS

TIMs was proposed by Lamb et al. (2021). A transformer network is divided into 'independent mechanisms' which update their state via sharing information between positions and sharing information between mechanisms. The information sharing step between mechanisms can be replaced by SW to create TIMs+SW.

G.1.1 MULTIMNIST GENERATION

In this task, we train an Image Transformer Parmar et al. (2018) (pixel-by-pixel, raster-order generative model) for next pixel prediction task on the "MultiMNIST dataset"

Parameter	Value
Common Parameters	
Optimizer	Adam(Kingma and Ba, 2014)
Learning rate	$1 \cdot 10^{-3}$
Batch size	12
Number of attention heads	8
TR	
Size of transformer layer	256
TIMs	
Number of mechanisms	4
Size of mechanism	48
TIMs+SW	
Number of mechanisms	4
Size of mechanism	40
Number of memory slots	2
Size of memory slots	160
Memory Attention Heads	8
Gate Style	'unit'
Number of MLP layers in Attention	2

Table 4: Hyperparameters for MultiMNIST Task

Each 32×32 image in this dataset is made up of four randomly selected (and augmented) MNIST digits (resized to 32×8) placed side-by-side as shown in figure 10. The digits themselves are selected independently of one-another.

The main aim of creating such a task is to observe the working of independent mechanisms in architectures such as TIMs (Lamb et al., 2021). Each image in the MultiMNIST dataset can be broken down into different sets of independent spatial components. Since the digits which make up the image are independently selected, the joint distribution of pixel intensities in any one of the four sections of the image is statistically independent of the pixel intensities in any other section of the image. Moreover each section of the image can be further broken down into independent spatial components: one that pertains to the background and one that pertains to the foreground.

It is expected that a monolithic architecture (having a single computational unit) would have to devote a significant portion of its training to learn the statistical independence between the different constituents of the image. On the other hand, architectures made up of sparsely interacting independent mechanisms have a natural way of capturing such statistical independence. A division of labour where each mechanism is focused on the generation of a distinct independent constituent of the image should allow for better generalization on the test set. Once the generation of a constituent is completed, the task can be handed over to some other mechanism based on current position in the image.

For this experiment we train a standard transformer with shared parameters across all layers (denoted by TR), TIMs (Lamb et al., 2021) with 4 mechanisms, and a modified version of TIMs with 4 mechanisms where the pair-wise communication between the mechanisms is replaced by communication via a shared workspace (denoted by TIMs+SW).

Training. We follow the minGPT Image Transformer setup Karpathy (2020) for our experiments. All three of the configurations have 8 layers, 8 heads for multi-headed attention and use the exact same parameter initialization and base architecture. We train all three of the models for 20 epochs.

In the TR model, all of the 8 monolithic layers share the same set of parameters. In TIMs and TIMs+SW, the first two layers are the standard monolithic layers having shared parameters. The middle four layers in both of these architectures are modular layers with four mechanisms. These four

Model	Loss
TR	0.000058
TIMs (4 mechanisms)	0.000050
TIMs+SW (4 mechanisms)	0.000042

Table 5: **MultiMNIST Generation Task:** We report cross-entropy loss between the generated pixel values and the true pixel values on the test set of MultiMNIST Generation Task (smaller numbers are better)

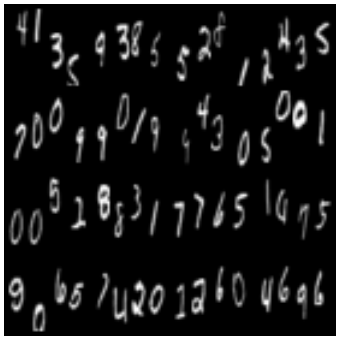

Figure 10: A randomly selected batch of 16 images from the MultiMNIST generation dataset (4 rows and 4 columns)

layers share the same set of parameters. In the case of TIMs+SW, the four mechanisms in these layers communicate via a shared workspace (having 2 memory slots). This shared workspace is common for all four middles layers and is absent in TIMs where the mechanisms communicate via pair-wise competition as proposed in the original paper. TIMs and TIMs+SW architectures are concluded by two more monolithic layers which again share the same parameters.

For all three models to have comparable number of parameters, we chose the transformer embedding dimension to be 256 for TR model, 192 for TIMs model and 160 for TIMs+SW model. In TIMs and TIMs+SW, the embedding dimension is divided equally among the four specialists. Each memory slot in the shared workspace of the TIMs+SW model has a 160 dimensional embedding and the model uses four heads to perform read and write operations on the shared workspace. Total number of parameters for all three architectures lie between 1M and 1.8M.

Results. We observe the best cross-entropy loss in 20 epochs on the test set of the MultiMNIST dataset for the next pixel prediction task in the table 5. We further plot the sixth layer "mechanism activation score" of TIMs and TIMs+SW while generating the first four images of the test set in the best epoch (shown in figure 5).

G.1.2 Using Workspace for language Modelling

We train our models on the WikiText-103 dataset by posing a language modeling problem. The dataset is divided into train, test and validation sets which are composed out of 28,475, 60 and 60 articles respectively. The total number of tokens in the train set is more than 103 million, hence the name of the dataset. This dataset retains numbers, punctuation and case.

Training. We train our models for 15 epochs for the next word prediction task on the WikiText-103 dataset and report the perplexity on the validation set. We show the results using TIMs (Lamb et al., 2021) with 4 mechanisms and TIMs+SW with 4 mechanisms (where we replace the pairwise communication in TIMs with communication via a shared workspace like in the MultiMNIST experiment). We modify the FAIRSEQ Ott et al. (2019) transformer language model class for all of our experiments.

For TIMs+SW, we train and test two different variants: TIMs+SSW uses soft attention to generate the activation scores of competing independent mechanisms whereas TIMs+HSW uses top-k attention with k=2.

Since in this test, our aim is to compare the performance of the two models for the language modeling task, the architectures are only made up of a transformer decoder. In both of the models, there are 8 transformer decoder layers divided into 3 sets. The first 2 layers are standard monolithic decoder layers which share the same parameters. The next 4 layers are modular layers (TIMs layers or TIMs+SW layers depending on the model choice). These layers also share the same parameters among themselves. The last 2 layers are again standard monolothic decoder layers, both sharing the same parameters.

The inputs to the network are 1024 dimensional word embeddings, input to a transformer layer of dimension 1024 and feed forward dimension of 2048.

Both of the networks have 8 attention heads with head dimension of 128. The total transformer layer size of $8 \times 128 = 1024$ is equally divided among the four mechanisms. In the case of TIMs, these mechanisms (in layers 3,4,5) interact via pair-wise communication, whereas in TIMs+SSW and TIMs+HSW, these mechanisms interact via a shared workspace. The shared workspace has 2 memory slots, each 1024 dimensional, having 4 attention heads for reading and writing.

Parameter	Value
Common Parameters	
Optimizer	Adam(Kingma and Ba, 2014)
Learning rate	$5 \cdot 10^{-4}$
Adam betas	$0.99, 0.98$
Weight decay	0.01
lr scheduler	'inverse square root'
Max tokens per gpu	3078
Batch size multiple	8
Number of attention heads	8
Transformer layer size	1024
Number of Mechanisms	4
Update frequency	4
Number of warmup updates	4000
Starting Warmup lr	$1 \cdot 10^{-7}$
TIMs+SSW	
Number of memory slots	2
Size of memory slots	1024
Memory Attention Heads	4
Gate Style	'unit'
Number of MLP layers in Attention	3
top-k competition	False
TIMs+HSW	
Number of memory slots	2
Size of memory slots	1024
Memory Attention Heads	4
Gate Style	'unit'
Number of MLP layers in Attention	3
top-k competition	True, k=2

Table 6: Hyperparameters for WikiText-103 Language Modeling Task

Results. We plot the perplexity (per epoch) on the validation set. All models have comparable number of parameters (within a 10% difference). We note that TIMs performs poorly on this dataset but adding shared workspace improves the performance consistently. We also note that sparsity indeed helps as TIMs+HSW performed the best.

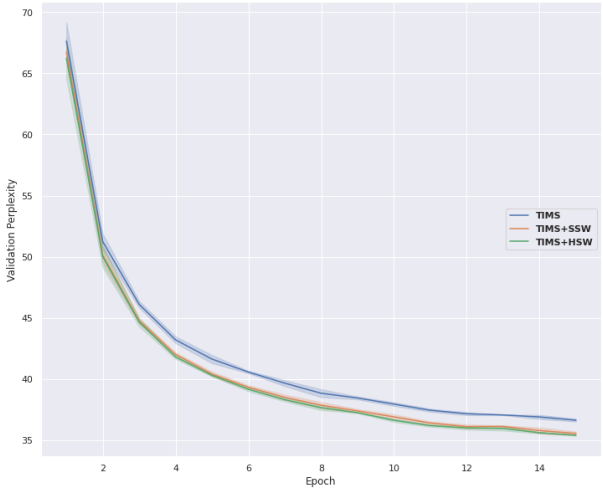

Figure 11: Per epoch validation perplexity for TIMs, TIMs+SSW, TIMs+HSW for wikitext-103 language modeling task