Appendix

6 Related Work

Hierarchical or Multiscale Recurrent neural networks. This work takes inspiration from a wide array of work on introducing multiple scales of processing into recurrent neural networks [21, 33, 53, 56, 46]. These works divide the processing into multiple streams each operating at a different temporal granularity. While these works mainly focus on recurrent neural networks and their application is mainly on natural language tasks, we focus on introducing multiple streams of processing and a hierarchical structure into Transformers while also focusing on a broader range of domains beyond natural language.

Transformers. Some of the components we describe in the proposed model have been used previously in various Transformer models. Transformer XL [23] also divides the input into segments. Each segment considers the tokens from the current segment and the previous segment for attention without passing gradients into the previous segments. A number of previous works [68, 51, 64, 66, 63, 65] have worked on introducing a hierarchical structure in Transformers mainly in the domain of vision. The main goal of these works has been to introduce convolution-like hierarchies into Vision Transformers [25]. While these works progressively reduce the spatial resolution of the inputs in order to introduce hierarchies, we introduce hierarchies by adding another slow stream of information processing and without reducing the spatial resolution of the inputs. We also provision for the higher level of the hierarchy (i.e. the slow stream) to provide information to the lower levels as top-down conditioning which is not possible in any of the previous works.

Top-Down Conditioning. Top-down information is information propagated from higher to lower levels of the network. It represents the models beliefs of the world and provides context for interpreting perceptual information. [52] and [26] have shown the advantages of top-down conditioning in recurrent neural networks and Transformers respectively. These works focus on different streams of processing operating at the same temporal granularity and the top-down conditioning is provided by higher level streams to the lower level streams. In our case, the top-down conditioning for the perceptual module is provided by the high-level slow stream which operates at a slower temporal granularity. This allows the perceptual model to be affected by much more long term high level information as compared to just short-term high level information as in the case of [52] and [26].

7 Additional Experimental Details

In this section, we cover the experimental details including the hyperparameter details and the the detailed task setups. In Section 7.1, we cover details for the Image Classification experiment also presenting ablations for the CIFAR10 dataset. In Section 7.2, we describe details of the self-supervised learning experiment also presenting results on the STL10 dataset and showing that the learned representations from the SiT + TLB model transfer better to CIFAR 10 than the baseline SiT model. In Section 7.3, we describe the experimental details of our experiments on the ListOps and Text Classification tasks from the Long Range Arena benchmark [60]. In Section 7.4, we describe details of our BabyAI experiments and also present results for various BabyAI environments. We also show an ablation where we vary the chunk size and examine its effect on model performance. In Section 7.5, we present details of our experiments on the 4 atari games.

7.1 Image Classification

We use the CIFAR10 and CIFAR100 datasets [47] for this task. We use a 9-layered model for the ViT baseline and a 12-layered model for the Swin Transformer Baseline. For the proposed model, we use a 6 layered model with R set to 2 i.e. we apply one CROSS ATTENTION + FFN per 2 SELF ATTENTION + FFN. The proposed model uses 9,649,546 parameters while ViT uses 10,253,578 and Swin Transformer uses 10,438,264. We use the AdamW optimzer for training with learning rate 0.001 and weight decay of 5e-2. We use a batch size of 128. We train all models for 100 epochs. The input to the model is an image of size 64×64 which is divided into a sequence of patches each of size 4×4. For ViT + TB, the sequence is further divided into chunks of 16 patches. For the case with 128×128 sized images, we still divide the image into patches of 4×4 and interpolate the positional embeddings to adapt to the resulting longer sequence length as done in [25] and [50]. For the proposed model, in 128×128 case we use a chunk size of 64. We use 1 v100 to train the model.

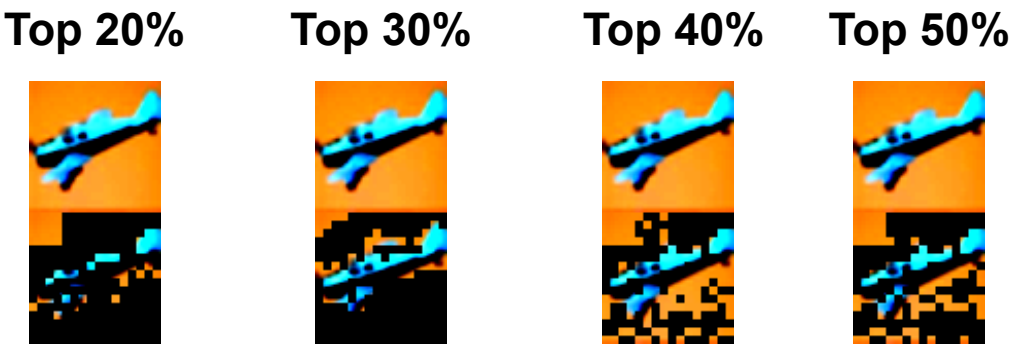
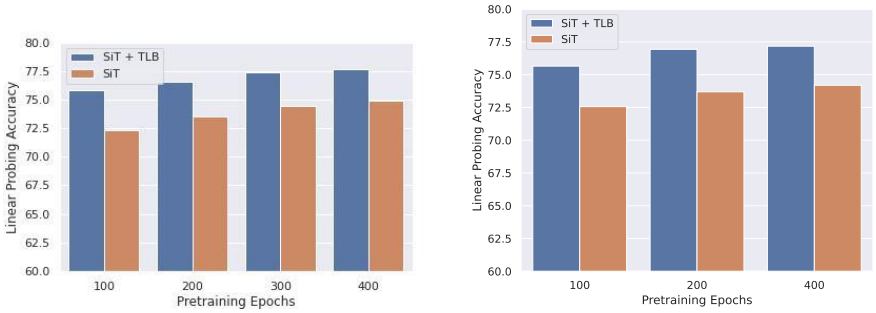


Figure 7: Here we visualize the patches that are in the top-k% of the attention scores paid by the TLB. We can see that as we increase k, the TLB pays attention to a larger area of the image.



(a) (a) STL10 Dataset

(b) (b) CIFAR10 Dataset

Figure 8: **Self Supervised Learning**. Here we show the result of training and linear probing on the STL10 and CIFAR10 dataset for different pre-training epochs. We can see that the proposed SiT + TLB approach outperforms SiT.

The training time for the proposed is 24 hours. We use 5 temporal latent bottleneck state vectors for the proposed model.

We make a similar visualization to Figure 2 in Figure 7 but in this case we vary the attention threshold of the TLB. We mask out all the patches that are not in the top-k% of the attention scores paid by the TLB. We can see that as we increase k, TLB pays attention to more patches which is expected.

7.2 Self Supervised Learning

For self supervised learning, we use the same setup as [2] and build on their codebasehttps://github.com/Sara-Ahmed/SiT. We use images of size 224×224. We use the same augmentation policy as [2]. For the baseline, we use a 12 layered Vision Transformer with 12 heads and embedding dimension 768. We use an FFN dimension of 3072. For the proposed model, we use a 12-layered model with $R = 2$. We use 5 temporal latent bottleneck state vectors. We use a patch size of 16 for both the models. For SiT + TLB, we use a chunk length of 20. Similar to [2], we use the Adam optimizer with batch size 72, momentum 0.9, weight decay 0.05 and learning rate of 0.0005. We train the models for 5 days on 1 RTX8000 GPU completing 400 pretraining epochs. For models pretrained on the CIFAR10 dataset, we perform linear probing training for 500 epochs. For models pretrained on the STL10 dataset, we perform linear probing training for 300 epochs.

We present additional results for the self-supervised learning experiments in Figure 8. In Figure 8(a), we pretrain the model on the STL10 dataset and perform linear probing on the same dataset. In Figure 8(b), we perform pretraining on the CIFAR10 dataset and perform linear probing also on CIFAR10. We can see that in both cases, the proposed SiT + TLB model outperforms SiT.

7.3 Long Range Dependencies

For the experiments on the long range arena benchmark [60], we build on the codebase from [70] https://github.com/NVIDIA/transformer-ls. We describe our setups for the ListOps and text classification tasks below.

ListOps We follow the same hyperparameters as [70]. In this task the model outputs the final number which is the result of a list operations. The number can be any number between 0-9, hence the model outputs a probability distribution over 10 possible numbers. For all the models, we use a transformer with embedding dimension 64 , FFN dimension 128 and 2 layers. For the Transformer + TLB model, we set $R = 1$. We use a chunk size of 20 and set the number of temporal latent bottleneck state vectors to 20. For training, we use Adam optimizer with a learning rate of 0.0001. We train the model for 5000 steps. We warmup the learning rate for 1000 steps. We use a batch size of 32.

In table 7, we run ablations on the chunk size of the proposed model. We can see that TLB shows maximum performance at chunk size = 100, and the performance gradually drops as chunk size reduces or increases. ListOps needs information from all of the input tokens and there is little redundancy in the data. Lower chunk sizes can potentially lead to a lot of information to integrate across chunks which might make the TLB forget important information more quickly and higher chunk sizes can lead to too much information to write in one chunk which can also lead to unwanted forgetting.

Table 7: **ListOps - Chunk Size Ablation**. Here, we vary the chunk size in the listops task. We can see that there is optimal chunk size below and above which the performance drops. Results averaged across 5 seeds.

Chunk Size	2	20	100	500	1000
Acc	$36.08_{\pm 0.26}$	$36.49_{\pm 0.16}$	$38.18_{\pm 0.17}$	$36.97_{\pm 0.27}$	$36.82_{\pm 0.14}$

In table 8, we vary the number of state vectors and analyse its effect on the performance. We can see that there is an optimal number of state vectors (20) above or below which the performance drops. Less number of state vectors can lead to very low capacity in the Temporal Latent Bottleneck leading to loss of important information. Similarly, a high number of state vectors can lead to a high capacity in the Temporal Latent Bottleneck leading to it capturing a lot of unnecessary of noisy information.

The Temporal Latent Bottleneck is not only important since it provides information from the past but also since it provides high-level information through top-down feedback. To confirm this hypothesis we design an ablation study with two baselines. For the first baseline (baseline 1), we remove the Temporal Latent Bottleneck and only let the representations from the current chunk attend to the representations of the past chunk at the same layer (i.e. not high level to low level communication). For the second baseline (baseline 2), we introduce a temporal latent bottleneck at each layer. Each layer writes to its own TLB and reads from its TLB. Each layer-specific TLB provides summarized information from the past to the future chunks of that layer. This baseline is like the proposed model but without top-down communication i.e. the TLBs do not communicate any information to the lower levels. We find that baseline 1 achieves a performance of $32.10_{\pm 0.019}\%$ while baseline 2 achieves a performance of $37.57_{\pm 0.003}\%$. The proposed model outperforms both the baselines achieving $38.2_{\pm 0.0001}\%$. This shows that the top-down information provided by the Temporal Latent Bottleneck is an important component of the model and the TLB is not only a medium for providing information about the past.

Text Classification Here we follow the hyperparameters as used in [60]. For all the models, we use a transformer with embedding dimension 256 , FFN dimension 1024. The baseline transformer has 4 layers with 4 transformer heads. For the Transformer + TLB model, we use 2 self-attention layers followed by 1 cross-attention for the fast step and 1 cross-attention layer for the slow step. We use a chunk size of 10 and set the number of temporal latent bottleneck state vectors to 10 unless otherwise specified. For training, we use the same learning rate schedule as used in [60] but lower the learning rate by half to 0.025. We train the model for 20000 steps. We warmup the learning rate for 8000 steps. We use a batch size of 32. For text classification, we also add positional embeddings to the temporal latent bottleneck state vectors and local positional embeddings to each input chunk. We find that this is crucial for achieving good performance.

Model	ListOps	Text	Retrieval	Image	PathFinder	Avg
Linear Transformer [43]	16.13	65.90	53.09	42.34	75.30	50.55
Reformer [44]	37.27	56.10	53.40	38.07	68.50	50.67
Sparse Transformer [17]	17.07	63.58	59.59	44.24	71.71	51.24
Sinkhorn Transformer [59]	33.67	61.20	53.83	41.23	67.45	51.29
Linformer [62]	35.70	53.94	52.27	38.56	76.34	51.36
Performer [19]	18.01	65.40	53.82	42.77	77.05	51.41
Synthesizer [58]	36.99	61.68	54.67	41.61	69.45	52.88
Longformer [10]	35.63	62.85	56.89	42.22	69.71	53.46
BigBird [67]	36.05	64.02	59.29	40.83	74.87	55.01
Transformer + TLB	37.05	81.88	76.91	57.51	79.06	66.48

Table 9: In this table, we compare the performance of the proposed Transformer + TLB model to various transformer-based baselines that implement attention in an efficient manner. We can see that the proposed Transformer + TLB has the best overall performance by a significant amount. Results averaged across 5 seeds.

To further analyze the effect of chunking we perform an ablation where critical information is divided into two chunks. We introduce spaces in the input text such that each word is divided across two chunks. Therefore, for each chunk the perceptual module sees two half-words - the second half of the previous word and the first half of the next word. Note that the introduced spaces ensure that the chunk size is still 10. We find that the accuracy drops slightly from 82.08% to 81.29%. The very slight drop in accuracy shows that even when critical information is divided across chunks the model can still perform well.

Table 8: **ListOps - Number of State Vectors Ablation**. Here, we vary the number of temporal latent bottleneck state vectors in the ListOps task. We can see that there is an optimal number of state vectors above or below which the performance drops. Results averaged across 5 seeds.

STATE TOKENS	1	10	20	200
ACC	$36.42_{\pm 0.32}$	$37.16_{\pm 0.45}$	$38.18_{\pm 0.17}$	$37.25_{\pm 0.46}$

Comparison to Efficient Transformer Baselines One of the claims of this work is that TLB has much lower computational complexity of attention than the vanilla transformer [61]. There has been a lot of work on reducing the computational complexity of the attention mechanism, especially for tasks involving long sequences such as the long range arena benchmark. We compare the proposed model against many efficient attention baselines on the tasks from the long-range-arena benchmark. We first give a brief description of all the tasks and then present the results in Table 9.

- **ListOps** - As mentioned previously, this task includes performing list operations on numbers such as Max and Min. This task contains sequences upto length 1024.
- **Text Classification** - As mentioned previously, this is a byte level text classification containing sequences of length upto 4k.
- **Retrieval** - This is also a byte-level text task. Here the model is tasked with outputting whether two documents are similar or not. The documents may be of lengths upto 4k.
- **Image Classification** - This is a pixel-level image classification task on the CIFAR10 dataset. Each image, when unrolled in a sequence, is of length 1024.
- **PathFinder** - This is also a pixel-level task containing images of size 32x32. When unrolled, the sequence length is 1024. The model is tasked with predicting whether two circles are reachable through a path containing dashed lines.

In Table 9, we can see that the proposed Transformer + TLB outperforms all efficient transformer baselines to achieve the best overall performance on the long range arena benchmark.

7.4 Baby AI

The BabyAI benchmark [16] offers a number of gridworld environments in which the agent has to carry out a given instruction. Each environment has 9 rooms arranged in a 3×3 matrix. Each room has a size of 6×6. Each environment in BabyAI is partially observable with the agent only having a 7×7 view of its locality. The total size of the

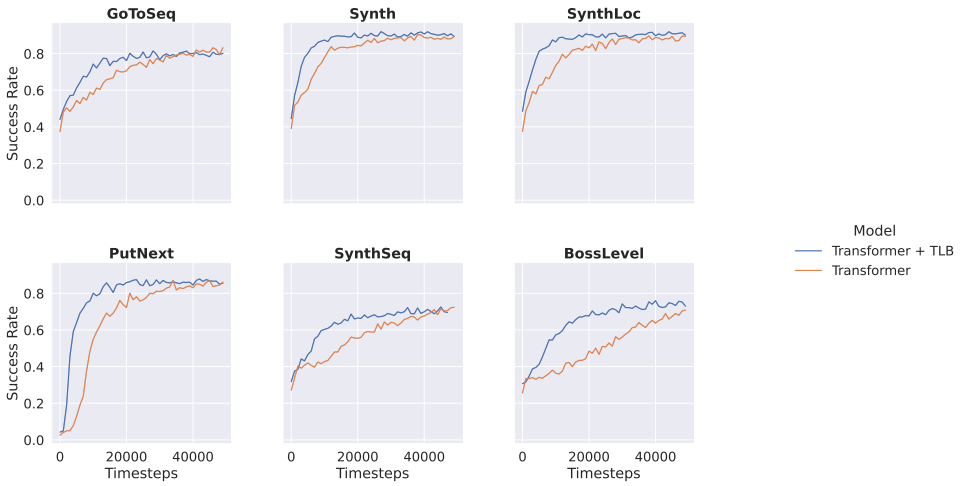


Figure 9: **Single Task Baby AI**. Here we compare the performance of Transformer and Transformer + TLB on individual tasks from the BabyAI benchmark. We can see that Transformer + TB converges much faster. Results averaged across 3 seeds.

maze is 18×18. We present a demonstration of some mazes from the BossLevel BabyAI environment in Figure 11. For each BabyAI environment, a new maze is generated for each episode. Each environment in BabyAI tests a different set of competencies of the model. We consider the most difficult environments in the BabyAI benchmark listed below -

- **GoToSeq**. A single GoTo instruction tasks the agent to go to a particular location on the grid. GoToSeq consists of a sequence of such GoTo commands.
- **Synth**. This includes a combination of instructions that ask the agent to put one object next to another, go to an object, open a door, or pick up an object.
- **SynthLoc**. Similar to Synth but objects are described using their location rather than appearance.
- **PutNext**. The instructions include tasks to put one object next to another.
- **SynthSeq**. Each instruction is a sequence of commands from the *Synth* environment.
- **BossLevel**. This environment includes instructions that are a combination of all competencies in all other environments of the BabyAI benchmark. Hence, this is the most difficult environment of BabyAI.

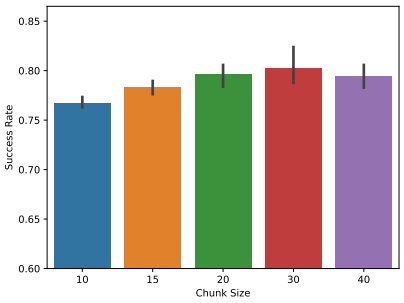


Figure 10: Here we show the effect of chunk size on the performance of the model for the multi-task BabyAI setting. We can see that similar to Table 7 the model performance hits optimal performance at chunk size 30 above or below which the performance drops.

We train all our models using behavior cloning from an expert policy. We collect 100k expert trajectories from an oracle for each environment. We feed the states for each episode into the model sequentially and task the model to predict the actions at each step. For both Transformer and Transformer + TLB, we use a transformer with 6 layers, embedding dimension set to 512 with 4 heads, FFN dimension set to 1024. For Transformer + TLB, we use 5 temporal latent bottleneck state vectors and chunk size of 30. We perform 1 CROSS ATTENTION + FFN per SELF ATTENTION + FFN. We train our models for 50000 steps. We evaluate the models by directly deploying them in the environments. We report the success rate across 512 evaluation episodes. An episode is successfully if the agent correctly carries out the given instruction. We train the model using Adam optimizer with a learning rate of 1e-4. Each model is trained on 1 RTX8000 GPU for 24 hours.

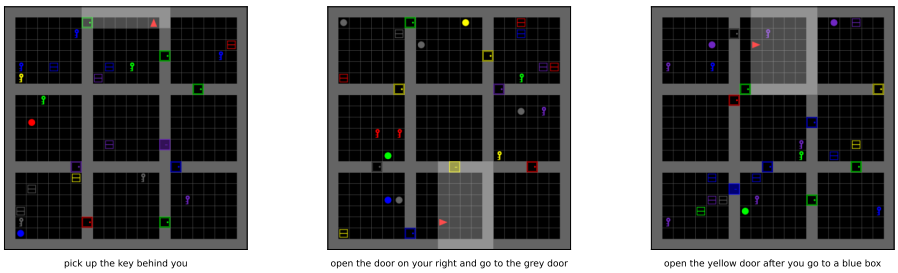


Figure 11: **BabyAI Demo**. Here we show some examples from the BossLevel environment of BabyAI. The agent is indicated by a red arrow. The bright region in front of the agent shows the partial view of the agent.

We report the performance of the Transformer baseline and the proposed Transformer + TLB on single tasks in Figure 9. We can see that in each case, the proposed model converges faster and also outperforms the baseline in some cases.

We also probe the effect of chunk size on the performance of the model. We show the result of this ablation in Figure 10. We can see that there is a sweet spot at chunk size 30, above or below which the performance drops. This indicates that a very high chunk size may be too much information for the temporal latent bottleneck state vectors to aggregate while a too low chunk size might lead to large recurrent sequence length which might be difficult to optimize.

7.5 Atari

For the experiments on Atari, we build on the codebase from [13] and extend it by introducing a temporal latent bottleneck. We test it on the same four games (Breakout, Pong, Seaquest, Qbert) as [13]. The model is trained on 1% of the Atari DQN-replay dataset [1] (500K transitions for each game).

The models are trained to predict the actions in the offline RL dataset. and are evaluated by directly deploying them in the environments. The results are reported across 10 seeds. The models are trained using a cross-entropy loss for 10 epochs. We use the same hyperparameters as [13]. For all the models, we use a transformer with an embedding dimension of 128, 6 layers, 8 heads, and FFN dimension set to 512. The model is trained using AdamW optimizer with a learning rate of 6e-4 with weight decay 0.1.

We train the models on 1 V100 GPU with 32 GB memory for 12 hours.

For the proposed model that incorporates a temporal latent bottleneck the following hyper parameters are used for the temporal latent bottleneck:

- **Pong**: We use chunk size of 18, we set $R = 1$, and use 6 temporal latent bottleneck state vectors.
- **Seaquest**: We use chunk size of 12, we set $R = 2$, and use 6 temporal latent bottleneck state vectors.
- **Qbert**: We use chunk size of 12, we set $R = 2$, and use 12 temporal latent bottleneck state vectors.
- **Breakout**: We use chunk size of 12, we set $R = 2$, and use 6 temporal latent bottleneck state vectors.

7.6 Copying Task

Here we give a detailed description of the copying task and the hyperparameter details of the used models. For a copying task of sequence length 100, the model first receives a sequence of 10 digits between 1 and 8 followed by 100 zeros. The model then receives an indicator input which indicates that the model should start outputting the original sequence it received. The indicator in our case is the digit 9. After receiving the indicator, the model receives 10 more zeros and then it outputs the original sequence again.

For both the Transformer + TLB and the Feedback Transformer model, we use 4 layers and 256 hidden dimension. We use an FFN dimension of 512. For the Transformer + TLB model, we set R to 1. We use a chunk size of 10. We use adam optimizer with learning rate 1e-4. We use a batch size of 100.