Temporal Latent Bottleneck: Synthesis of Fast and Slow Processing Mechanisms in Sequence Learning

Aniket Didolkar [1], Kshitij Gupta [1], Anirudh Goyal [1], Nitesh B. Gundavarapu [5]
Alex Lamb [2], Nan Rosemary Ke [3], Yoshua Bengio [1,4]

Abstract

Recurrent neural networks have a strong inductive bias towards learning temporally compressed representations, as the entire history of a sequence is represented by a single vector. By contrast, Transformers have little inductive bias towards learning temporally compressed representations, as they allow for attention over all previously computed elements in a sequence. Having a more compressed representation of a sequence may be beneficial for generalization, as a high-level representation may be more easily re-used and re-purposed and will contain fewer irrelevant details. At the same time, excessive compression of representations comes at the cost of expressiveness. We propose a solution which divides computation into two streams. A slow stream that is recurrent in nature aims to learn a specialized and compressed representation, by forcing chunks of K time steps into a single representation which is divided into multiple vectors. At the same time, a fast stream is parameterized as a Transformer to process chunks consisting of K time-steps conditioned on the information in the slow-stream. In the proposed approach we hope to gain the expressiveness of the Transformer, while encouraging better compression and structuring of representations in the slow stream. We show the benefits of the proposed method in terms of improved sample efficiency and generalization performance as compared to various competitive baselines for visual perception and sequential decision making tasks.

1 Introduction

The interplay between fast and slow mechanisms for information processing and perception has been studied in both cognitive science and machine learning [5, 35]. In the brain, short-term and long-term memory have developed in a specialized way. Short-term memory is allowed to change very quickly to react to immediate sensory inputs and perception. It also tends towards high capacity storage of all pieces of information which may be relevant for future reasoning [42, 3, 4]. By contrast, long-term memory changes slowly [45, 41], is highly selective and involves repeated consolidation. It contains a set of memories that summarize the entire past, only storing details about observations which are most relevant [28, 6].

Deep Learning has seen a variety of architectures for processing sequential data [36, 57, 18]. For example. recurrent neural networks compress information about a sequence into a single hidden state. Transformers get rid of the recurrent state by dynamically capturing information between positions using multi-head dot product attention [61]. Transformers have become the dominant architecture across a wide range of domains including vision [25], natural language [24, 54, 11, 69, 20, 55], and reinforcement learning [13, 40]. They have eclipsed recurrent neural networks [36, 57, 18] in almost all sequence processing domains due to their high representational capacity and scalability. Despite their wide applicability, it is well known that Transformers are very data

[01] Mila, University of Montreal, [2] Microsoft Research, New York, NY, [3] Google Deepmind, [4] CIFAR Fellow, [5] Google Research, Corresponding authors: adidolkar123@gmail.com

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

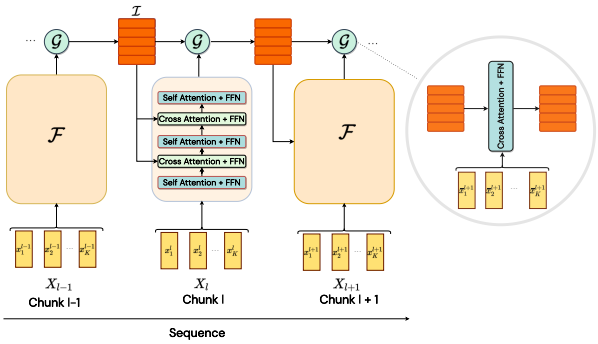


Figure 1: **Perceptual module + Temporal Latent Bottleneck Model**. $\mathcal{F}$ denotes the perceptual module or the fast stream which is a Transformer. $\mathcal{I}$ represents the temporal latent bottleneck state (consisting of a set of vectors) that are updated using a recurrent function denoted by $\mathcal{G}$. The given sequence is first divided into chunks of size K and each chunk X_l is processed by $\mathcal{F}$ which consists of interleaved SELF ATTENTION + FFN (denoted in blue) and CROSS ATTENTION + FFN (denoted in green) layers. The CROSS ATTENTION + FFN layers allow the representation of $\mathcal{F}$ to be conditioned on top-down information from $\mathcal{I}$. The representations of the temporal latent bottleneck state is updated using the outputs of $\mathcal{F}$ by a recurrent function $\mathcal{G}$, which consists of a CROSS ATTENTION + FFN layer as shown in the circle.

hungry and work well mainly at scale. This can be attributed to their inductive bias towards modeling all possible pairwise interactions in the sequence which results in no consolidation of information. This lack of selectivity in the attention mechanism also leads to a high computational complexity which scales quadratically with input size. Additionally, modeling all possible pairwise interactions maybe extremely wasteful and may result in capturing unnecessary information not useful for the downstream task [29, 39]. The goal of this work is to design an architecture for autoregressive modeling that has an inductive bias towards learning temporally compressed representation that retains the benefits of Transformers while preserving long-range interactions.

For learning temporally compressed representations, we start by dividing the computation of the Transformer into two streams of processing - a fast stream and a slow stream. Inspired by the idea of long-term and short-term memory, we want the fast stream to have a short-term memory with a high capacity that reacts quickly to sensory input. We refer to this fast stream as the perceptual module and implement it using a Transformer since they are known to have high representational capacity. On the other hand, we want the slow stream to have a long-term memory which updates at a slower rate and summarizes the most important information in the input sequence. We refer to this slow stream as the **Temporal Latent Bottleneck**.

Implementation-wise, we divide the input into fixed size chunks (Figure 1). The fast stream operates within each chunk while the slow stream consolidates and aggregates information across chunks updating itself once per chunk. This leads to *information asymmetry* between fast and slow stream as the fast stream contains fine-grained local information while the slow stream contains coarse-grained distant information. Such kind of information asymmetry has shown to improve generalization and adaptation performance of learned policies in the context of RL [30, 27]. The fast and slow streams interact with each other though bottleneck of attention. The division of computation into a fast and slow stream eliminates the need for capturing all possible pairwise interactions and thus introducing selectivity in the attention mechanism resulting in a much lower computational complexity which is not quadratic in the input size. We show that the limited capacity of the slow stream and consolidation of information by a recurrent neural network prevents the model from capturing unnecessary information not useful for the downstream task. We evaluate the proposed model in a number of domains showing that it consistently outperforms competent baselines showing improved generalization to scenarios not seen during training.

2 Methodology

We now present the proposed approach in detail. Our model jointly leverages the strengths of Transformers [61] and recurrent neural networks [18, 36].

2.1 Desiderata for Fast and Slow Streams of Processing

We give the detailed description of the proposed model in the next section. Here, we give an overview of our architecture and discuss some of its key properties. Given an input sequence, it

Algorithm 1: PyTorch-style pseudocode for proposed model

```
# 𝒞(query, key, value):  Cross Attention + FFN layer
# 𝒮(query, key, value):  Self Attention + FFN layer
# L: Num.  Layers
# R: Num.  𝒞 per 𝒮
# X: Input sequence of length T. shape:  [B x T x D]
# ℐ:  The Temporal Bottleneck
# K: Chunk Size

X = torch.chunk(X, K, dim = 1) # List of length ⌊T/K⌋ with each element
 of size [B x K x D]

for X_c in X:
   for l in range(L):
      X_c = 𝒮^l(X_c,   X_c,   X_c)
      if l % R == 0:
         X_c = 𝒞^⌊L/l⌋(X_c,   ℐ,   ℐ)
   ℐ = 𝒞(ℐ,   X_c,   X_c)
```

is first divided into chunks of size K. Each chunk is processed by perceptual module represented by a Transformer (denoted as $\mathcal{F}$). While processing each chunk, $\mathcal{F}$ is also conditioned on information from the Temporal Latent Bottleneck module $\mathcal{G}$. The slow stream is a recurrent stream which has its own state consisting of a set of N vectors (or slots) also called temporal latent bottleneck state denoted as $\mathcal{I}$ in Figure 1. In the following sections, we use the term *temporal latent bottleneck* to refer to the temporal latent bottleneck state $\mathcal{I}$. This state is updated once per chunk using information from the perceptual module through a cross attention mechanism.

The perceptual module operates within each chunk while the temporal latent bottleneck operates across chunks slowly updating itself after each chunk has been processed by the perceptual module. Thus, the only way the perceptual module gets information about inputs beyond its own chunk is through the temporal latent bottleneck. An added advantage of this is that the computational complexity of the attention mechanism in the proposed model is $\mathcal{O}(\frac{T}{K}(K^2 + KN))$ while that of a Transformer is $\mathcal{O}(T^2)$, where T is the length of the sequence, K is the chunk size, and N is the number of temporal latent bottleneck state vectors. Since $K << T$ and $N << T$, we can see that $\frac{T}{K}(K^2 + KN) < T^2$. Therefore the proposed model has a much lower computational complexity compared to a Transformer. Furthermore, the capacity of the temporal latent bottleneck is limited and much smaller than that of the perceptual module. This encourages the temporal latent bottleneck to represent the most salient information about the past while the perceptual module represents only local information. This creates an *information asymmetry* between the two streams. This information asymmetry leads to the perceptual module having a fine grained view of the nearby inputs but a very coarse grained view of the distant past. This is very different from the usual self-attention which attends to all tokens in the sequence at the same level of granularity.

An advantage of having a compressed representation of the past is that it allows the model to forget irrelevant information. For example, if an agent is navigating in a large maze, it does not need to have fine grained knowledge of its actions from the distant past. In the case of a Transformer, it would attend to every step from the past (including steps from the distant past) which may be irrelevant in the present context thus wasting its capacity in modeling irrelevant details. Another important component of the proposed model is top-down attention which conveys contextual information from the high-level Temporal Latent Bottleneck module to the processing of low-level perceptual module. Past works [52, 26, 34, 23] have shown that top-down attention improves generalization and adaptation performance of the learned model. One difference between these works and the proposed model is that in their case the multiple streams operate at the same temporal granularity while in our case the streams operate at a different time scales (because of information asymmetry). Through our experiments, we show the advantage of the proposed architecture over these works. Next, we describe the detailed implementation of the proposed model.

2.2 Computational Steps

We denote the input X as a sequence of T tokens - $X = [x_0, x_1, x_2, \ldots, x_t]$. We chunk this input into chunks of size K resulting in $\lfloor T/K \rfloor$ chunks. We refer to l^{th} chunk as X_l. We represent the

state of the temporal latent bottleneck $\mathcal{I}$ (i.e. the slow stream) as a set of M d-dimensional vectors. As mentioned previously, we denote the temporal latent bottleneck module as $\mathcal{G}$ and the perceptual module as $\mathcal{F}$. $\mathcal{G}$ updates the temporal latent bottleneck state while $\mathcal{F}$ processes chunks X_l to form the latent representation $\bar{X}_l$ -

$$\textbf{Perceptual Module} \quad \bar{X}_l = \mathcal{F}(X_l, \mathcal{I}_l) \tag{1}$$

$$\textbf{Temporal Latent Bottleneck Module} \quad \mathcal{I}_{l+1} = \mathcal{G}(\mathcal{I}_l, \bar{X}_l) \tag{2}$$

Preliminaries. The central components of our model are the key value attention mechanism [7, 61] and the FFN module [61]. We use two forms of the attention mechanism -(1) Self Attention [61]: In this the query and key vectors refer to the same set of vectors; (2) Cross Attention [29, 39, 31]: In this the query and key vectors refer to seperate sets of vectors.

Perceptual Module $\mathcal{F}$. As mentioned previously, the perceptual module refers to the fast stream that acts directly on the input. The perceptual module operates on each chunk separately. Therefore, at any time the input to the perceptual module are the tokens corresponding to a particular chunk $X_l = [x_{l\times K}, x_{l\times K+1}, \ldots, x_{l\times K+K}]$. The perceptual module is a Transformer with self attention layers, cross attention layers, and FFNs. It has 2 kinds of layers - (1) SELF ATTENTION + FFN; (2) CROSS ATTENTION + FFN. The SELF ATTENTION + FFN layers process the input tokens and the CROSS ATTENTION + FFN layers integrate top-down information from the temporal latent bottleneck state $\mathcal{I}$ as follows -

$$\begin{aligned} X_l &= \text{ATTENTION}(\text{LN}(X_l), \text{LN}(\mathcal{I}), \text{LN}(\mathcal{I})) + X_l \\ X_l &= \text{FFN}(\text{LN}(X_l)) + X_l \end{aligned} \tag{3}$$

We include one CROSS ATTENTION + FFN layer per R SELF ATTENTION + FFN layers. The diagramatic representation of the perceptual module is presented in Figure 1 (in the processing of chunk X_l). In the figure, we set $R = 1$.

Temporal Latent Bottleneck Module $\mathcal{G}$. The temporal latent bottleneck (TLB) module represents the slow stream that operates on the temporal latent bottleneck state $\mathcal{I}$. $\mathcal{I}$ is updated using information from a particular chunk processed by the perceptual module. This update happens once for each chunk of the perceptual module resulting in $\lfloor T/K \rfloor$ updates for $\mathcal{I}$. Since the temporal latent bottleneck state $\mathcal{I}$ updates at a lower frequency than the perceptual module, it is expected to capture more stable and slowly changing features while the perceptual module captures faster changing features resulting in multiple scales of information representation. An update to the temporal latent bottleneck state $\mathcal{I}$ consists of a cross attention operation where the queries come from $\mathcal{I}$ and the keys and values come from the output of the perceptual module. This cross attention operation is followed by an FFN update to $\mathcal{I}$. Consider the perceptual module outputs for a chunk l to be $\bar{X}_l = [\bar{x}_{l\times K}, \ldots, \bar{x}_{l\times K+K}]$. The update operation is implemented as follows:

$$\begin{aligned} \bar{\mathcal{I}} &= \text{ATTENTION}(\text{LN}(\mathcal{I}_l), \text{LN}(\bar{X}_l), \text{LN}(\bar{X}_l)) + \mathcal{I}_l \\ \mathcal{I}_{l+1} &= \text{FFN}(\text{LN}(\bar{\mathcal{I}})) + \bar{\mathcal{I}} \end{aligned} \tag{4}$$

The temporal latent bottleneck module introduces the notion of recurrence in our model. We show the details of this module in Figure 1 (inside the circle).

Perceptual Module + Temporal Latent Bottleneck Model. We now present our complete architecture integrating both the perceptual module and the temporal latent bottleneck together. Given a sequence of tokens $X = [x_0, x_1, x_2, \ldots, x_t]$. We chunk this input into chunks of size K resulting in $\lfloor T/K \rfloor$ chunks. The chunks are processed sequentially one after the other. For a chunk k, it is first processed using the perceptual module conditioned on information from the temporal latent bottleneck state. The outputs of the chunk are used to update the temporal latent bottleneck state $\mathcal{I}$. The resultant temporal latent bottleneck state is then used to process the next chunk. The full model is presented in Figure 1. We use a Transformer as the perceptual module in our experiments. Thus our main contribution is introducing a temporal latent bottleneck into Transformers and showing its advantages through a variety of experiments. We also present the detailed algorithm for the proposed approach in Algorithm 1.

The proposed model is similar to a parallel work called Block Recurrent Transformers [37]. There are few differences between our work and theirs. First, they use a sliding window attention, while

we divide the input into chunks. In their paper, they perform cross attention and self attention in parallel while we find that doing them sequentially and performing cross attention once per R self attention steps yields better results. We defer the rest of the discussion on related works to Appendix Section 6

3 Experiments

Our goal is to show the wide applicability and benefits offered by the *temporal latent bottleneck*, which we refer to as TLB. We demonstrate that the proposed model outperforms competitive baselines across many domains including vision, reinforcement learning, and natural language. Our main goal is to show that the proposed approach has high expressive power like Transformers while also being sample efficient unlike Transformers. Thus our main baselines are based on the original Transformer architecture. For example, we compare against ViT [25] in image classification, Decision Transformer [13] in Reinforcement Learning, and Vanilla Transformer in rest of the tasks. We also compare against some representative baseline that offer some of the key properties that our model offers. For example, we compare against state-of-the art Swin Transformer [50] which is a strong baseline for image classification and is also hierarchical similar to the proposed model. We also compare against Transformer LS [70] which also processes long-term and short-term information using different attention streams. Furthermore, we also compare against Feedback Transformer [26], which also introduces top-down communication into Transformers. Another key point of the proposed model is that any position cannot attend to any information from the future beyond its chunk since the temporal latent bottleneck only *summarizes the past, not the future*. Meanwhile, **all the baselines we consider have bidirectional context** i.e. they can attend to all of the past and the future. We observe that despite this limitation, the proposed model outperforms all the considered baselines.

Table 1: **Image Classification**. Here we compare the performance of the proposed ViT + TLB model against ViT and SwinV2 on CIFAR10 and CIFAR100 datasets for 64×64 images and 128×128 images. Note that the model is trained only on the 64×64 sized images and then transferred to 128×128 sized images. Results averaged across 3 seeds.

	CIFAR10		CIFAR100	
Model	$\mathbf{64 \times 64}$	$\mathbf{128 \times 128}$	$\mathbf{64 \times 64}$	$\mathbf{128 \times 128}$
ViT	93.75	73.18	69.53	47.4
Swin V2	97.66	84.9	79.95	58.59
ViT + TLB	94.79	84.38	79.17	59.19

3.1 Temporal Latent Bottleneck For Perception

Image Classification. Recently, Transformers have been widely applied for visual perception and have shown strong performance improvements over CNNs in tasks such as image classification, semantic segmentation, instance segmentation, etc. In this work we focus on image classification using Transformers. For a model to do well on image classification, it should learn to only focus on the relevant information and ignore other details (eg. background information). Self attention does not inherently have this inductive bias of ignoring irrelevant information since it models all pairwise interactions between the inputs. We posit that adding a limited bandwidth temporal latent bottleneck into the Transformer will allow the model to focus only on the most important information in the image which should enable the model to perform well.

Table 2: Here we show the performance of the proposed ViT + TLB model against two baselines - One with no access to the past and One with no top-down information (i.e. high level to low level communication). We can see that the model suffers a drop in performance for both the baseline thus showing the importance of past information and top-down communication. Results averaged across 3 seeds.

			CIFAR10	
Model	Past Info	Top Down	$\mathbf{64 \times 64}$	$\mathbf{128 \times 128}$
ViT + TLB	✓	✓	94.79	84.38
No Past Info.	×	×	91.30	72.92
No Top-Down Condn	✓	×	93.75	83.59

Results. We test our hypothesis on the CIFAR10 and CIFAR100 [47] image classification datasets. We also test the generalization abilities of the models by comparing their performance on images of higher resolution (128×128) than seen during training (64×64). We use ViT [25] and Swin Transformer V2 (denoted as Swin V2) [50] as our baselines. Swin Transformer V2 has a key strength of generalizing to higher resolution images than those seen during training, making it a strong baseline. The input image is split into patches of size 4×4 and fed in rastor order to all the models. For the proposed model we use ViT as the perceptual module and add a temporal latent bottleneck module

to it. We call this model ViT + TLB. To predict the classification scores, we take the mean across the final temporal latent bottleneck state vectors and pass the resulting representation through an MLP. We present the results for this experiment in table 1. *We can see that* ViT + TLB *outperforms* ViT *for all cases and performs competitively to Swin Transformer V2*. For further hyperparameter details, we refer the reader to Appendix section 7.1.

Quantitative Analysis. One essential component of our model is top-down conditioning. Top down information helps in integrating information from the past as well as high-level information into the perceptual module. We hypothesize that both these kinds of information are important for the model to perform well. To test this, we design two baselines - (1) ViT + TLB (No Past Info): In this baseline, we do not allow the TLB to communicate to the perceptual module, therefore the perceptual module has no information about the past; (2) ViT + TLB (No Top-Down Condn): In this baseline, we have a separate temporal latent bottleneck module at every layer, therefore the perceptual module has access to past information but does not have access to any high-level information through top-down feedback. We show the results for this ablation in Table 2. We can see that the performance of both the baselines is worse than the proposed ViT + TLB model. This shows that both high-level information through top-down feedback and information from the past is important for the model to perform well.

Figure 2: We want to analyze the information which is being used to influence the processing of Temporal Latent Bottleneck. To calculate this, we calculate the attention scores for different patches and we mask out all the patches that are not in the top 30% of the attention scores. We can see that for both the images it recovers the foreground almost perfectly which shows it learns to focus on the most important information required to solve the downstream task.

Qualitative Analysis. To get a better understanding of what the temporal latent bottleneck is doing, we visualize the parts of the image where the temporal latent bottleneck pays most attention while it is being updated by the perceptual module. We present this visualization in Figure 2. We can see that the temporal latent bottleneck learns to pay the most attention to the foreground in each case. This further confirms our hypothesis that the limited capacity bottleneck focuses on the most important information required to solve the downstream task.

Self Supervised Learning. Many recent works have used Vision Transformers for self-supervised learning [8, 2, 32, 12, 49, 48]. Here we show a proof-of-concept that introducing a temporal latent bottleneck in Vision Transformers results in better self-supervised representations. We consider the SiT model from [2] for this experiment. They use 3 objectives to pretrain their model - (1) The Reconstruction Objective - Reconstructs the input image, (2) The Rotation Prediction Objective - Predicts the rotation angle from [0°, 90°, 180°, 270°], and (3) The Constrastive Objective (similar to SimCLR [14]). For the proposed approach, we introduce a temporal latent bottleneck into SiT resulting in the SiT + TLB model. SiT also uses additional trainable contrastive and rotation tokens as input for calculating the contrastive and rotation objectives respectively. For SiT + TLB, we take the mean across the temporal latent bottleneck state vectors and use the resulting representation for computing the rotation and contrastive objectives. We use a chunk length of 20 for the SiT + TLB model. We pretrain the model for 400 epochs and evaluate the pretrained model at different epochs using linear probing.

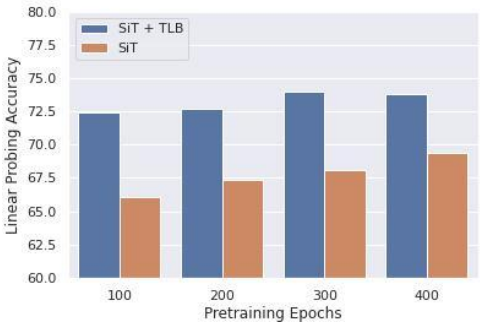


Figure 3: **Self Supervised Learning** Results of linear probing on the CIFAR 10 dataset for models pretrained on the STL 10 dataset. We can see that the proposed SiT + TLB approach outperforms SiT.

Results. To evaluate the model, we pretrain the model on the STL10 dataset [22] and evaluate the learned representation using linear probing on the CIFAR10 dataset [47]. We present the re-

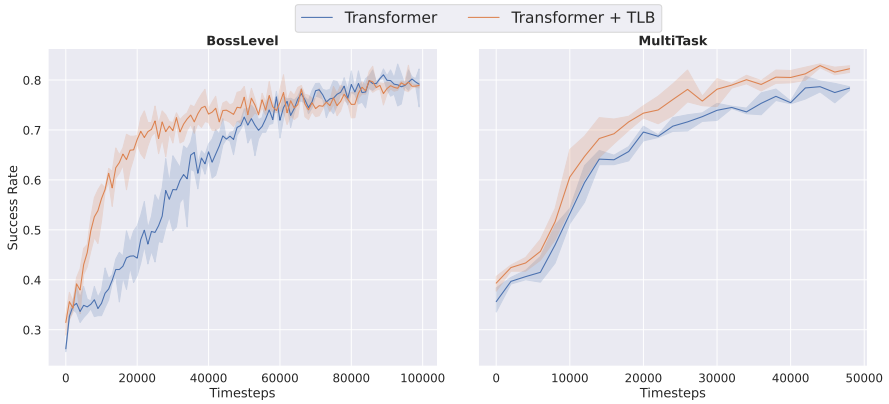


Figure 4: **Single Task BabyAI**. (Left) Here we compare the performance of Transformer and Transformer + TLB on the BossLevel task from BabyAI. We can see that while both the models converge to a similar success rate, Transformer + TLB converges faster than Transformer. **Multi Task BabyAI**. (Right) Here we compare the performance of Transformer and Transformer + TLB on 8 tasks from the BabyAI suite of environments. A single model is trained for all the 8 tasks. We can see that Transformer + TLB converges faster and achieves a better performance than Transformer.

sults for this experiment in Figure 3. We can see that the proposed approach outperforms SiT thus showing the effectiveness of the proposed architecture for self-supervised learning. For additional experimental results and details, we refer the reader to Appendix section 7.2.

3.2 Temporal Latent Bottleneck for Sequential Decision Making

Transformers have recently been used for sequential decision making in reinforcement learning tasks such as Atari and BabyAI [13, 38]. These works deploy Transformers in the offline RL setting where a large number of trajectories are available either through another trained agent or an expert agent. The Transformer is trained as an autoregressive generative model that predicts actions conditioned on the past context. We incorporate the temporal latent bottleneck module into the Transformer and explore its benefits in the RL setting. We test the proposed model in the BabyAI [15] and Atari [9] benchmarks. We describe our setups in detail below.

Instruction Based Decision Making: BabyAI. BabyAI [15] provides a suite of environments where the agent has to carry out a given instruction in a partially-observable maze. These instructions include competencies such as going to an object in the maze, placing an object beside another object in the maze, opening a door with a key, etc. Some environments in the benchmark contain instructions that combine multiple competencies sequentially. For example, *pick up a red ball and open the door in front of you after you pick up the grey ball on your left and pick up a red box*. Each environment in Baby AI benchmark has a different type of instruction that tests a different competency. The *BossLevel* is the most complicated environment that contains instructions from all competencies. For more details regarding the various environments from the BabyAI benchmark, we refer the reader to Appendix section 7.4.

We train our models with behavior cloning using expert trajectories from an oracle. For evaluation, we test the model by directly deploying it in the environment. We report the *success rate* which measures whether the agent successfully carried out the given instruction or not. We use a Transformer [61] as the baseline in these experiments. For the proposed model, we introduce a temporal latent bottleneck into the Transformer-based perceptual module. For the baseline Transformer model, we append the language instruction to the sequence of states allowing the model to attend to the language instruction at each layer. For the proposed model, the language instruction is appended to each chunk, allowing each chunk to attend to it.

Results. We consider two settings - **Single task** and **Multi task**. In the single task setting, we evaluate the proposed approach on individual environments from the BabyAI benchmark while in the multi-task setting we train a single model on 8 different environments.

Single Task. We present the results for BossLevel in Figure 4 (left) and present the results for the other tasks in Appendix Figure 9. *We can see that while Transformer and Transformer + TLB achieve almost similar performance at convergence. However, Transformer + TLB is much more sample efficient, converging much faster*. We posit that the temporal latent bottleneck module prohibits the model from paying attention to unnecessary information which allows it to converge faster.

Multi Task. We present the results for the multi task setting in Figure 4 (right). We train the model on 8 environments - PutNext, Unlock, Synth, GoToSeq, SynthLoc, GoToImpUnlock, BossLevel. We evaluate the model on the same 8 environments. We report the average success rate across 8 games. *We can see that the Transformer + TLB model converges faster and also outperforms the Transformer*. We refer the reader to the appendix for more details regarding the model and training.

Atari. [13] recently introduced the Decision Transformer (DT) which learns to play various games in the Atari benchmark from suboptimal trajectories of a learned agent. Decision Transformer models the offline RL problem as a conditional sequence modelling task. The model uses a causal mask and supervised training to match the actions in the offline dataset conditioned on the future expected returns and the past history. This is done by feeding into the model the states, actions, and the return-to-go $\hat{R}_c = \sum_{c'=c}^{C} r_c$, where c denotes the timesteps. This results in the following trajectory representation: $\tau = \left(\hat{R}_1, s_1, a_1, \hat{R}_2, s_2, a_2, \hat{R}_3, s_3, a_3, \ldots\right)$, where a_c denotes the actions and s_c denotes the states. At test time, the start state s_1 and desired return $\hat{R}_1$ is fed into the model and it autoregressively generates the rest of the trajectory. Experimental results show that DT can leverage the strong generalization capabilities of Transformers and achieve the desired returns in a wide variety of tasks in Atari and OpenAI Gym, outperforming previous approaches in offline RL.

Table 3: **Atari**. Here we show that adding a temporal latent bottleneck into decision Transformer improves performance across various atari games. Results are averaged across 10 seeds.

GAME	DT	DT + TLB
BREAKOUT	$71.51_{\pm 20.58}$	$87.63_{\pm 16.24}$
PONG	$13.68_{\pm 2.00}$	$14.71_{\pm 1.78}$
QBERT	$3268_{\pm 1773.07}$	$5019.75_{\pm 1647.13}$
SEAQUEST	$1039.11_{\pm 122.90}$	$1248.22_{\pm 86.62}$

We use the same setup as used in [13] for our experiments. We set the context length to a fixed number C. During training, C timesteps from an episode are sampled and fed into the model resulting in a trajectory of length $3\boldsymbol{C}$ (considering 3 modalities - returns-to-go, states, and actions). Each modality is processed into an embedding of size d. The state is processed using a convolutional encoder into an embedding of size d. The resulting trajectory is fed into the decision Transformer. The outputs corresponding to the states s_c are fed into a linear layer to predict the action a_c to be taken at timestep c. For the proposed model, we incorporate a temporal latent bottleneck module into the Decision Transformer.

Table 4: **Long Range Dependencies**. Here we compare the performance of the proposed model against the recently proposed long-short Transformer model [70] and the vanilla Transformer model [61]. We can see that the proposed model outperforms both the baselines thus showing the superiority of the proposed model in modelling long-range and hierarchical dependencies. Results averaged across 3 seeds.

MODEL	LISTOPS	TEXT CLASSIFICATION
TRANSFORMER	$37.64_{\pm 0.0001}$	$64.0_{\pm 0.0001}$
TRANSFORMER LS	$37.5_{\pm 0.0002}$	$65.5_{\pm 0.0003}$
TRANSFORMER + TLB	$38.2_{\pm 0.0001}$	$82.08_{\pm 0.44}$

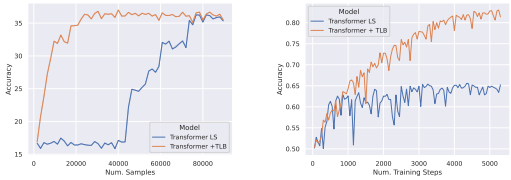
Results. We present our results in Table 3. The model is trained on 1% of the Atari DQN-replay dataset [1] (500K transitions for each game). We use the same 4 games used in [13]: Pong, Seaquest, Qbert, and Breakout. *We can see that the proposed model outperforms Decision Transformer in all the considered games thus showing the effectiveness of the proposed model*. More details regarding the model and training can be found in the appendix section 7.5.

3.3 Temporal Latent Bottleneck for Long Range Dependencies

Here, we test the effectiveness of the proposed model in modelling long range dependencies. We apply the proposed model on the ListOps and text classification tasks from the Long Range Arena (LRA) benchmark [60]. Both these tasks have very long sequences ranging from 1K to 4K tokens. Thus, for a model to do well, it has to learn to capture dependencies across very long time scales. Additionally, all these tasks have an inherent hierarchical structure. For example, Listops consists of nested list operations which makes it hierarchical. For text classification, the inputs consist of text in the form of bytes. Therefore, the model has to learn to compose bytes into characters and characters into words. We hypothesize that the multi-scale hierarchical nature of the proposed model will be extremely useful in modelling such hierarchical information.

Results. For this experiment, we use the same setup as[70]. For the proposed model, we use a Transformer as the perceptual model and implement the temporal latent bottleneck as described in Section 2.2. We take the mean across the temporal latent bottleneck state vectors and use the resulting representation for classification. We compare the model against the long-short Transformer (LS) model [70], which is a recently proposed model for the long range arena benchmark, and the vanilla Transformer model [61]. We present the results in Table 4. *We can see that the proposed model outperforms both the baselines in both the tasks thus showing its usefulness in modeling long range dependencies*. For further details, we refer the reader to Appendix section 7.3.

In Fig. 5, we plot the convergence curves for ListOps and Text Classification. For ListOps (Figure 5(a)), we plot the convergence curves against the number of samples i.e. we do only one pass over the dataset hence the model does not see any example more than once. We can see that the proposed Transformer + TLB model is much more sample efficient than the baseline Transformer LS model. For Text Classification (Figure 5(b)), we plot the convergence curves against the number of training steps. We find that doing only one pass over the dataset does not work well for both the baseline and the proposed model hence we use number of training steps on the x-axis. We can see that while initially both models converge at a similar pace, the proposed model achieves a much higher performance.



(a) ListOps (b) Text Classification

Figure 5: **(a)** Here we show the performance on ListOps as a function of the number of samples in the dataset. We do only one pass over the entire data and find that Transformer +TLB takes much fewer samples to converge as compared to the baseline Transformer LS. **(b)** Here we show the convergence curves of both the Transformer + TLB model and the Transformer LS model on the text classification task. In this case, we do not perform only one pass over the dataset since we observe that both models do not reach convergence in a single pass. Therefore, we report the number of training steps on the x-axis. We can see that the proposed model achieves much higher score than the baseline.

We measure the wall-clock time and memory required for text classification task as we vary the chunk size in Table 5. All TLB models have an increased memory efficiency and supports faster inference speeds with respect to the baseline transformer model. The training speeds also get better with increased chunking. The only exception is very small chunk sizes, where the training is slower than the baseline because of increased temporal unrolling. However, as shown in Figure 5, such models are very sample efficient resulting in lesser training steps overall.

Table 5: **Text Classification - Performance Ablation** Here, we compare the wall-clock time and memory during the training and inference phase of the text classification task w.r.t baseline transformer model.

Chunk Size	1000	100	40	20	10
Inference Speed	3.5x	3.6x	3.3x	2.2x	1.2x
Inference Memory	0.09x	0.08x	0.12x	0.08x	0.1x
Training Speed	4.4x	4.4x	2.2x	1.4x	0.7x
Training Memory	0.14x	0.08x	0.49x	0.40x	0.42x

Analysis. Here we perform an ablation to show that the Temporal Latent Bottleneck does not only contain short-term information but also summarizes information from long term past. To test this hypothesis we design a baseline in which the current chunk attends to the previous few chunks instead of attending to the temporal latent bottleneck. We find that this baseline achieves a performance of $\mathbf{32.10}_{\pm 0.019}$**%** compared to the proposed models $38.2_{\pm 0.0001}$% on the ListOps task. This shows that the Temporal Latent Bottleneck contains information about the long-term past. Additionally, here also we perform an experiment to probe the importance of top-down communication (i.e. high level to low level feedback). To do this we use the same Transformer + TLB (No Top-Down Condn) baseline used in Table 2. We find that this baseline achieves a performance of $\mathbf{37.57}_{\pm 0.003}$**%** which is lower than the performance of the proposed Transformer + TLB model which

Figure 6: **Copying Task**. Here we compare the performance of the proposed Transformer + TLB model to the Feedback Transformer model on the copying task. We can see that the Transformer + TLB achieves perfect accuracy for all the studied sequence length while the the performance of Feedback Transformer starts dropping after sequence length 400.

Sequence Length	Feedback Transformer	Transformer + TLB
100	11800	6200
200	16600	9100
300	35100	12700
400	NA	14600
500	NA	13600
600	NA	19300

Table 6: **Copying Sample Efficiency Ablation**. Here we present the number of unique samples required for the models to reach to perfect accuracy on the copying task. NA indicates that the model does not reach perfect accuracy. We can see that in all cases the Transformer + TLB model is more sample effecient than the Feedback Transformer model.

achieves $\mathbf{38.2}_{\pm 0.0001}$ **%** which further shows that top-down information from high-level to low-level is important for the model to perform well.

We perform additional experiments to give us more insight into the behavior of the proposed model. We present these experiments in Appendix Section 7.3. We also compare the model to additional efficient transformer baselines for all LRA tasks in Appendix Table 9.

Temporal Latent Bottleneck for Copying Task. Here, we study the copying task used in [36]. In the copying task, the model receives a sequence of 10 digits followed by blank inputs for a large number of steps, and then the model is asked to output the sequence of digits it received initially. Therefore, the model has to remember the original sequence of digits across long time scales. We can control the sequence length of this task by controlling the length of the blank input.

The main motive behind studying this task is comparing the model to the Feedback Transformer model introduced in [26] which also has top-down attention similar to the proposed model but does not represent information at multiple scales. We compare both the models on the copying task for sequence lengths 100, 200, 300, 400, 500, and 600. We present the results for this task in Figure 6. We can see that while both Transformer + TLB and Feedback Transform perform well for low sequence lengths, the performance of Feedback Transformer drops for longer sequence lengths above 400 while the proposed Transformer + TLB model still achieves perfect accuracy at long sequence lengths. We also compare the sample efficiencies to achieve perfect accuracy for both the models. We present this result in Table 6. We can see that the proposed Transformer + TLB is more sample effecient than the baseline Feedback Transformer achieving perfect accuracy in much lesser number of samples in each case. For further details we refer the reader to Appendix Section 7.6.

4 Conclusion

We have developed an approach aimed at introducing selectivity in the interactions across time-steps in a transformer by splitting processing into two streams: (a) a slow stream that is updated in a recurrent manner and (b) a fast stream that processes the visual input. The two streams are parameterized independently and interact with each other via attentional bottleneck. The information processed by the fast stream is used to change the state of the slow stream, and the information in the slow stream is used by the fast stream as contextual information to process the input. Through our experiments we show that the proposed approach works well across wide range of domains and problems. One limitation of the proposed model is that the chunk size is fixed and treated as a hyperparameter which requires some domain knowledge. Future work should explore methods for dynamic chunking.

5 Acknowledgement

The authors would like to thank Compute Canada for providing the computational resources used in this project. The authors also gratefully acknowledge the funding from Samsung, IBM and CIFAR.

References

[1] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. Striving for simplicity in off-policy deep reinforcement learning. *CoRR*, abs/1907.04543, 2019.

[2] Sara Atito Ali Ahmed, Muhammad Awais, and Josef Kittler. Sit: Self-supervised vision transformer. *CoRR*, abs/2104.03602, 2021.

[3] Richard C Atkinson and Richard M Shiffrin. The control of short-term memory. *Scientific american*, 225(2):82–91, 1971.

[4] Emanuel Averbach and Abner S Coriell. Short-term memory in vision. *The Bell System Technical Journal*, 40(1):309–328, 1961.

[5] Jimmy Ba, Geoffrey E. Hinton, Volodymyr Mnih, Joel Z. Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4331–4339, 2016.

[6] Alan Baddeley, Vivien Lewis, Margery Eldridge, and Neil Thomson. Attention and retrieval from long-term memory. *Journal of Experimental Psychology: General*, 113(4):518, 1984.

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2015.

[8] Hangbo Bao, Li Dong, and Furu Wei. Beit: BERT pre-training of image transformers. *CoRR*, abs/2106.08254, 2021.

[9] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *CoRR*, abs/1207.4708, 2012.

[10] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *CoRR*, abs/2004.05150, 2020.

[11] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

[12] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. *CoRR*, abs/2104.14294, 2021.

[13] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *CoRR*, abs/2106.01345, 2021.

[14] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. *CoRR*, abs/2002.05709, 2020.

[15] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: First steps towards grounded language learning with a human in the loop. *CoRR*, abs/1810.08272, 2018.

[16] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: First steps towards grounded language learning with a human in the loop. *CoRR*, abs/1810.08272, 2018.

[17] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *CoRR*, abs/1904.10509, 2019.

[18] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[19] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy J. Colwell, and Adrian Weller. Rethinking attention with performers. *CoRR*, abs/2009.14794, 2020.

[20] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways, 2022.

[21] Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *CoRR*, abs/1609.01704, 2016.

[22] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 215–223, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[23] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860, 2019.

[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[25] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[26] Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. Addressing some limitations of transformers with feedback memory, 2021.

[27] Alexandre Galashov, Siddhant M Jayakumar, Leonard Hasenclever, Dhruva Tirumala, Jonathan Schwarz, Guillaume Desjardins, Wojciech M Czarnecki, Yee Whye Teh, Razvan Pascanu, and Nicolas Heess. Information asymmetry in kl-regularized rl. *arXiv preprint arXiv:1905.01240*, 2019.

[28] Philip Goelet, Vincent F Castellucci, Samuel Schacher, and Eric R Kandel. The long and the short of long–term memory—a molecular framework. *Nature*, 322(6078):419–422, 1986.

[29] 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. *CoRR*, abs/2103.01197, 2021.

[30] Anirudh Goyal, Riashat Islam, Daniel Strouse, Zafarali Ahmed, Matthew Botvinick, Hugo Larochelle, Yoshua Bengio, and Sergey Levine. Infobot: Transfer and exploration via the information bottleneck. *arXiv preprint arXiv:1901.10902*, 2019.

[31] Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. *CoRR*, abs/1909.10893, 2019.

[32] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross B. Girshick. Masked autoencoders are scalable vision learners. *CoRR*, abs/2111.06377, 2021.

[33] Salah Hihi and Yoshua Bengio. Hierarchical recurrent neural networks for long-term dependencies. In D. Touretzky, M. C. Mozer, and M. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8. MIT Press, 1995.

[34] Felix Hill, Olivier Tieleman, Tamara Von Glehn, Nathaniel Wong, Hamza Merzic, and Stephen Clark. Grounded language learning fast and slow. *arXiv preprint arXiv:2009.01719*, 2020.

[35] Geoffrey E. Hinton and David C. Plaut. Using fast weights to deblur old memories. In *IN PROCEEDINGS OF THE 9TH ANNUAL CONFERENCE OF THE COGNITIVE SCIENCE SOCIETY*, pages 177–186. Erlbaum, 1987.

[36] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[37] DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. Block-recurrent transformers, 2022.

[38] Donald Joseph Hejna III, Pieter Abbeel, and Lerrel Pinto. Improving long-horizon imitation through language prediction, 2022.

[39] Andrew Jaegle, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver: General perception with iterative attention, 2021.

[40] Michael Janner, Qiyang Li, and Sergey Levine. Reinforcement learning as one big sequence modeling problem. *CoRR*, abs/2106.02039, 2021.

[41] Annette Jeneson and Larry R Squire. Working memory, long-term memory, and medial temporal lobe function. *Learning & memory*, 19(1):15–25, 2012.

[42] John Jonides, Richard L Lewis, Derek Evan Nee, Cindy A Lustig, Marc G Berman, and Katherine Sledge Moore. The mind and brain of short-term memory. *Annu. Rev. Psychol.*, 59:193–224, 2008.

[43] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. *CoRR*, abs/2006.16236, 2020.

[44] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *CoRR*, abs/2001.04451, 2020.

[45] Janet L Kolodner. Maintaining organization in a dynamic long-term memory. *Cognitive science*, 7(4):243–280, 1983.

[46] Jan Koutník, Klaus Greff, Faustino J. Gomez, and Jürgen Schmidhuber. A clockwork RNN. *CoRR*, abs/1402.3511, 2014.

[47] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[48] Chunyuan Li, Jianwei Yang, Pengchuan Zhang, Mei Gao, Bin Xiao, Xiyang Dai, Lu Yuan, and Jianfeng Gao. Efficient self-supervised vision transformers for representation learning. *CoRR*, abs/2106.09785, 2021.

[49] Zhaowen Li, Zhiyang Chen, Fan Yang, Wei Li, Yousong Zhu, Chaoyang Zhao, Rui Deng, Liwei Wu, Rui Zhao, Ming Tang, and Jinqiao Wang. MST: masked self-supervised transformer for visual representation. *CoRR*, abs/2106.05656, 2021.

[50] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, and Baining Guo. Swin transformer V2: scaling up capacity and resolution. *CoRR*, abs/2111.09883, 2021.

[51] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *CoRR*, abs/2103.14030, 2021.

[52] Sarthak Mittal, Alex Lamb, Anirudh Goyal, Vikram Voleti, Murray Shanahan, Guillaume Lajoie, Michael Mozer, and Yoshua Bengio. Learning to combine top-down and bottom-up signals in recurrent neural networks with attention over modules. *CoRR*, abs/2006.16981, 2020.

[53] Michael C Mozer. Induction of multiscale temporal structure. In J. Moody, S. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4. Morgan-Kaufmann, 1991.

[54] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[55] Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsen, Siddhant Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, Laurent Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsimpoukelli, Nikolai Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Toby Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew Johnson, Blake Hechtman, Laura Weidinger, Iason Gabriel, William Isaac, Ed Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub, Jeff Stanway, Lorrayne Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. Scaling language models: Methods, analysis; insights from training gopher, 2021.

[56] Jürgen Schmidhuber. Neural sequence chunkers. Technical report, 1991.

[57] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[58] Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. Synthesizer: Rethinking self-attention in transformer models. *CoRR*, abs/2005.00743, 2020.

[59] Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. Sparse sinkhorn attention. *CoRR*, abs/2002.11296, 2020.

[60] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. *CoRR*, abs/2011.04006, 2020.

[61] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[62] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *CoRR*, abs/2006.04768, 2020.

[63] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *CoRR*, abs/2102.12122, 2021.

[64] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. *CoRR*, abs/2103.15808, 2021.

[65] Jianwei Yang, Chunyuan Li, Pengchuan Zhang, Xiyang Dai, Bin Xiao, Lu Yuan, and Jianfeng Gao. Focal self-attention for local-global interactions in vision transformers. *CoRR*, abs/2107.00641, 2021.

[66] Kun Yuan, Shaopeng Guo, Ziwei Liu, Aojun Zhou, Fengwei Yu, and Wei Wu. Incorporating convolution designs into visual transformers. *CoRR*, abs/2103.11816, 2021.

[67] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. *CoRR*, abs/2007.14062, 2020.

[68] Pengchuan Zhang, Xiyang Dai, Jianwei Yang, Bin Xiao, Lu Yuan, Lei Zhang, and Jianfeng Gao. Multi-scale vision longformer: A new vision transformer for high-resolution image encoding. *CoRR*, abs/2103.15358, 2021.

[69] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022.

[70] Chen Zhu, Wei Ping, Chaowei Xiao, Mohammad Shoeybi, Tom Goldstein, Anima Anandkumar, and Bryan Catanzaro. Long-short transformer: Efficient transformers for language and vision. *CoRR*, abs/2107.02192, 2021.

Checklist

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Appendix Section 7
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No] All the datasets we use are openly available
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] The datasets we use do not contain any sensitive information.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

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

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

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

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. In Section **??**, we present details of our language modelling experiments.

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

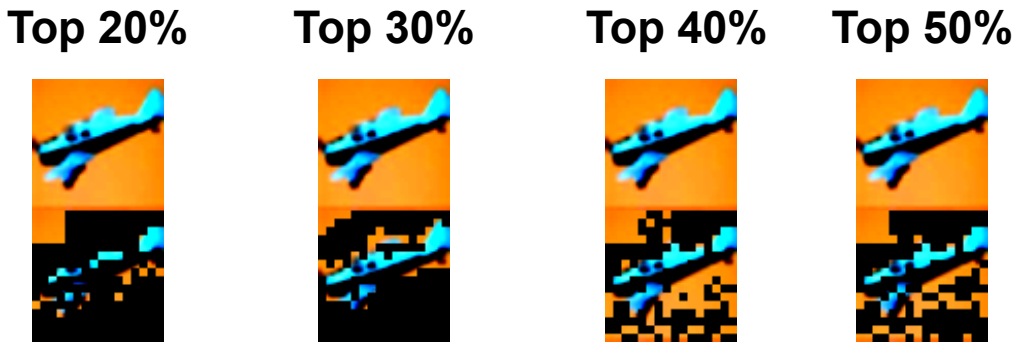
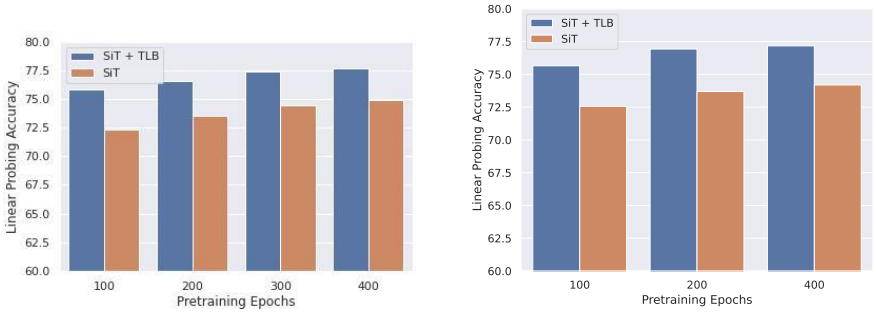


Figure 7: Here we visualize the patches that are in the top-k% of the attention scores paid by the TLB. We can see that as we increase k, the TLB pays attention to a larger area of the image.



(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 proposed model, in 128×128 case we use a chunk size of 64. We use 1 v100 to train the model. 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 baseline 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 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

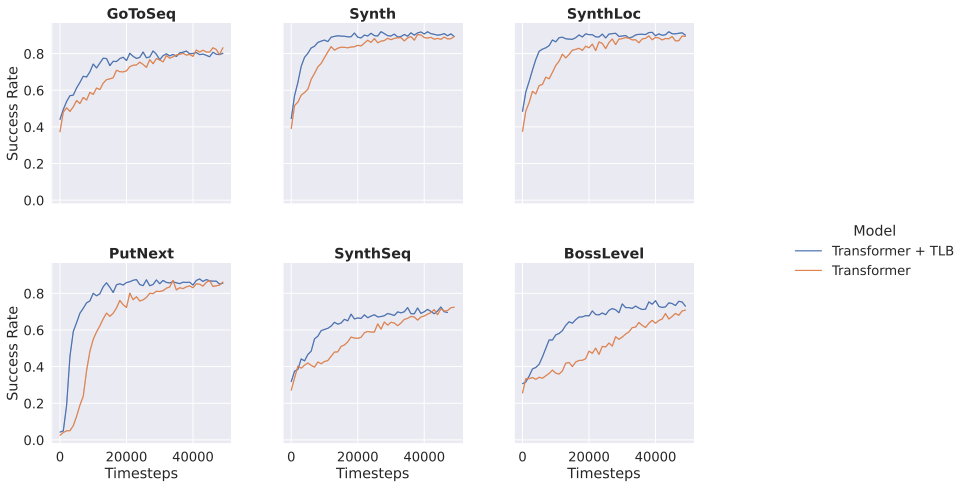


Figure 9: **Single Task Baby AI**. Here we compare the performance of Transformer and Transformer + TLB on individual tasks from the BabyAI benchmark. We can see that Transformer + TB converges much faster. Results averaged across 3 seeds.

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.

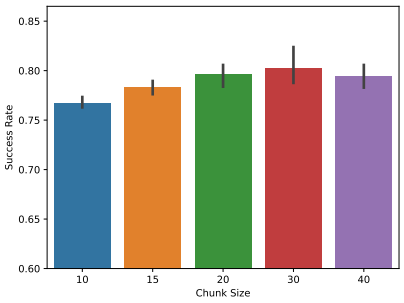


Figure 10: Here we show the effect of chunk size on the performance of the model for the multi-task BabyAI setting. We can see that similar to Table 7 the model performance hits optimal performance at chunk size 30 above or below which the performance drops.

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.

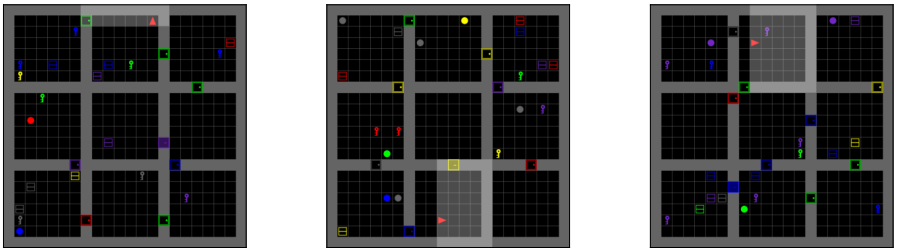


Figure 11: **BabyAI Demo**. Here we show some examples from the BossLevel environment of BabyAI. The agent is indicated by a red arrow. The bright region in front of the agent shows the partial view of the agent.

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.