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

Anonymous Author(s)
Affiliation
Address
email

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, 34]. 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 [41, 3, 4]. By contrast, long-term memory changes slowly [42, 40], 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 [23, 7].

Deep Learning has seen a variety of architectures for processing sequential data [35, 58, 16]. Sequence models which only rely on a short fixed context window may be seen as a form of fast-processing mechanisms, because there is no dependence on the longer-term history. Early research in deep learning found that robustness is substantially improved by using a recurrent neural network to represent the entire history of the sequence, with direct evidence found in the domain of handwriting synthesis [26]. These recurrent neural networks which force the entire hidden state to be compressed into a single hidden state became highly successful across speech recognition, natural language processing, and other sequence domains [27, 28, 29, 59]. Recurrent neural networks also saw significant advances in architectural inductive biases, such as a bias towards making small and

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

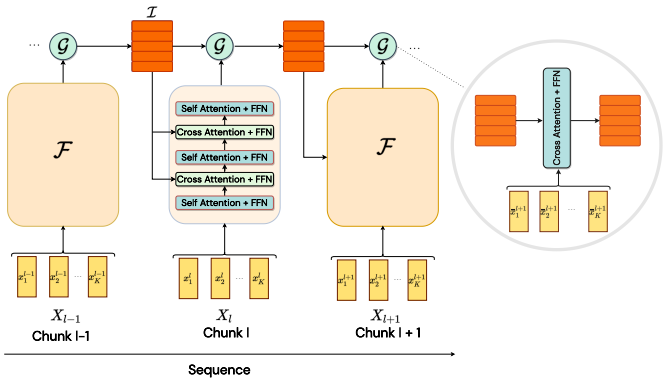

Figure 1: **Perceptual module + Temporal Latent Bottleneck Model**. $\mathcal{F}$ denotes the perceptual module or the fast stream which is a Transformer. $\mathcal{I}$ represents the temporal latent bottleneck state (consisting of a set of vectors) that are updated using a recurrent function denoted 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.

selective changes to the hidden state on each step which helped improve its generalization performance [45, 31, 25].

Despite their wide applicability, recurrent neural networks have at the same time struggled in modeling long-sequences, especially in the presence of long-range dependencies. To help address this, attention was added to recurrent neural networks to allow for information processing which can selectively bypass the recurrent state [8]. In later works the recurrent state was removed entirely in favor of the Transformer architecture, which shares information between positions using attention [61]. Research on scaling laws has shown that the expressiveness advantage of Transformers over recurrent neural networks grows with the length of the sequences being modeled [20, 55, 11].

Transformers have become the dominant architecture across a wide range of domains including vision [21], natural language [20, 55, 11, 67, 17, 56], and reinforcement learning [13, 39]. Despite their success, it is well known that Transformers are extremely data hungry and work well mainly at scale [21, 46, 49]. This may be attributed to the inductive bias in Transformers to model all possible pairwise interactions between the inputs which also results in a very high computational complexity that scales quadratically with the input size. The lack of selectivity in interactions in the Transformer makes them extremely memory inefficient and may bias them towards modeling unnecessary, irrelevant, or noisy interactions.

This motivates the use of a *Temporal Latent Bottleneck*, with the goal of improving Transformers while using lesser data and memory. The proposed model aims at introducing selectivity in the interactions by dividing computation into two streams - (1) A high-level slow stream consisting of a set of vectors updated in a recurrent manner (also referred to as the temporal latent bottleneck), and, (2) A low-level attention-based fast stream that processes the input. The fast stream processes locally neighbouring information within chunks, while the slow stream contains information about distant tokens across chunks. The slow stream and the fast stream interact using a multi-head attention mechanism to achieve selectivity in how local and distant information is mixed. We show that the resulting model substantially outperforms Transformers and demonstrates improved generalization, especially when the test data includes novel challenges for the model that were not encountered 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 [16, 35]. The fast stream operates on raw inputs and is instantiated using a Transformer. We denote this Transformer by $\mathcal{F}$. The slow stream operates at a higher level and updates at a lower frequency than the fast stream. Any recurrent function can be used to instantiate the slow stream.

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. We present a detailed dia-

gram of our model in Figure 1. Given an input sequence, it is first divided into chunks of size K. Each chunk is processed by a Transformer (denoted as $\mathcal{F}$) which is also known as the perceptual module (since it processes the raw input) or the fast stream. While processing each chunk, $\mathcal{F}$ is also conditioned on information from the slow stream $\mathcal{G}$ which is also called the temporal latent bottleneck module. The slow stream is a recurrent stream which has its own state consisting of a set of N vectors 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. One advantage of this is that the computational complexity of the attention mechanism in the proposed model is $\mathcal{O}(K^2 + KN)$ 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 $K^2 + KN < T^2$. 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.

One advantage of having a bottlenecked view 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.

Introducing specialized fast and slow processing mechanisms in machine learning is not completely new. A number of works [66, 51, 63, 65, 62, 64] build on vision Transformers [21] by introducing convolution-like hierarchies in Transformers. The main idea behind these models is to progressively reduce the size of the feature maps similar to convolutions so that some parts of the network can specialize at a coarser granularity. Our way of specializing fast and slow processing is different. We do not progressively reduce the size of the feature maps in deeper layers, rather information flows both ways between the fast processing stream and the slow processing stream. This allows the higher level (i.e. the slow stream) to convey information about the long-term context to the lower levels (i.e. the fast stream) as top-down conditioning information . However, previous works on introducing hierarchies in vision Transformer do not allow the higher levels to convey any information to the lower levels. Past works have shown the usefulness of top-down information sharing between specialized processing streams. [53] explored an architecture in which multiple recurrent streams are active at every time-step, but encouraged one stream to be more active by providing it direct access to the input. This selective sharing of information between the streams using attention led to better generalization. [22] showed that top-down sequential feedback in Transformers, although computationally expensive, improves the learning of long term dependencies.

Through our experiments we show that the proposed model substantially outperforms Transformers on tasks where generalization to variations unseen during training is critical. We verify this improvement in imitation learning (behavior cloning), where the training and evaluation distributions can diverge as a result of the agent visiting states which are not present in the dataset. Next, we describe the detailed implementation of our model.

2.2 Computational Steps

We first describe our notation. We denote the input $\boldsymbol{X}$ as a sequence of $\boldsymbol{T}$ tokens - $\boldsymbol{X} = [\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_t]$. We chunk this input into chunks of size $\boldsymbol{K}$ resulting in $\lfloor \boldsymbol{T}/\boldsymbol{K} \rfloor$ 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 $\boldsymbol{M}$ 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 $\bar{X}_l$ -

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

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

Next, we go into details of the implementation of these modules.

Preliminaries. The central component of our model is the key value attention mechanism [8, 61]. The attention mechanism allows to dynamically select information from a set of $\mathcal{N}$ *read* vectors $\mathcal{R}$ to update a set of $\mathcal{M}$ *write* vectors $\mathcal{W}$. The write vectors are first projected to a set of queries $Q = W^q\mathcal{W}$, where $Q \in \mathbb{R}^{\mathcal{M}\times D}$. The read vectors are projected into keys $K = W^k\mathcal{R}$ and values $V = W^v\mathcal{R}$, where $K \in \mathbb{R}^{\mathcal{N}\times D}$ and $V \in \mathbb{R}^{\mathcal{N}\times D}$. The updated write vectors $\bar{\mathcal{W}}$ are obtained as -

$$\bar{\mathcal{W}} = \textsc{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{D}})V \tag{3}$$

The first term $\frac{QK^T}{D}$ is a dot product that decides the *attention scores* given to each query-key pair. The softmax normalizes the scores across the key dimension. Pairs with high similarity will be assigned a high score. The final update to the write vector is obtained by taking the convex combination of values weighted by the attention scores. The convex combination ensures that the dynamic selection of information is differentiable and can be trained by backpropagation. The updated vectors $\bar{\mathcal{W}}$ are utilized by residually adding them to the write vectors ($\mathcal{W} = \mathcal{W} + \bar{\mathcal{W}}$). If the read and write vectors refer to the same set of vectors, the attention mechanism is called a *self-attention* mechanism [61]. If the read and write vectors refer to seperate set of vectors, the attention mechanism is called a *cross attention* mechanism [25, 24, 38]. In this paper, we use both kinds of attention mechanisms.

The Transformer architecture is another central component of our model which stacks a series of self attention and FFN layers. Each FFN layer is a 2 layered MLP that projects the input to a larger dimension and projects it back to the input dimension. In this case, we use the gelu activation [32] for the FFN.

$$\text{FFN}(x) = W_2.\text{GELU}(W_1x + B_1) + B_2 \tag{4}$$

Given the attention and FFN layers, a single Transformer layer can be written as -

$$\begin{aligned} x &= \textsc{Attention}(\text{LN}(x), \text{LN}(x), \text{LN}(x)) + x \\ x &= \text{FFN}(\text{LN}(x)) + x \end{aligned} \tag{5}$$

where LN refers to layer normalization layers [6]. In general, multiple such Transformer layers are stacked to solve any task.

We augment the Transformer, which has only self attentions, to include cross attentions too so that the temporal latent bottleneck module and perceptual module can share information with each other. We now describe the specifics of the perceptual module and the temporal latent bottleneck module.

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 $\boldsymbol{X}_l = [\boldsymbol{x}_{l\times K}, \boldsymbol{x}_{l\times K+1}, \ldots, \boldsymbol{x}_{l\times K+K}]$. 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 as described in Equation 5 and the CROSS ATTENTION + FFN layers integrate top-down information from the temporal latent bottleneck state $\mathcal{I}$ as follows -

$$\begin{aligned} \boldsymbol{X}_l &= \textsc{Attention}(\text{LN}(\boldsymbol{X}_l), \text{LN}(\mathcal{I}), \text{LN}(\mathcal{I})) + \boldsymbol{X}_l \\ \boldsymbol{X}_l &= \text{FFN}(\text{LN}(\boldsymbol{X}_l)) + \boldsymbol{X}_l \end{aligned} \tag{6}$$

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 $\lfloor \boldsymbol{T}/\boldsymbol{K} \rfloor$ 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 $\bar{\boldsymbol{X}}_l = [\bar{\boldsymbol{x}}_{l\times K}, \ldots, \bar{\boldsymbol{x}}_{l\times K+K}]$. The update operation is implemented as follows:

$$\begin{aligned} \bar{\mathcal{I}} &= \textsc{Attention}(\text{LN}(\mathcal{I}_l), \text{LN}(\bar{\boldsymbol{X}}_l), \text{LN}(\bar{\boldsymbol{X}}_l)) + \mathcal{I}_l \\ \mathcal{I}_{l+1} &= \text{FFN}(\text{LN}(\bar{\mathcal{I}})) + \bar{\mathcal{I}} \end{aligned} \tag{7}$$

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 $\boldsymbol{X} = [\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_t]$. We chunk this input into chunks of size $\boldsymbol{K}$ resulting in $\lfloor \boldsymbol{T}/\boldsymbol{K} \rfloor$ 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 the Appendix (Algorithm 1).

3 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 [18, 33, 54, 57, 43]. 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 [19] 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 [66, 51, 63, 65, 62, 64] 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 [21]. 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. [53] and [22] 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 [53] and [22].

The proposed model is similar to a parallel work called block recurrent Transformers [36]. 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. They also use special tricks to deal with some instabilities in their case, while we find no such instabilities in our model. Also, while their focus is mainly on natural language tasks, we focus on a broader variety of tasks.

4 Experiments

In this section, we outline the tasks in which we applied the temporal latent bottleneck and direct the reader to the appendix for more details on the 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 [21] 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 [68] which also processes long-term and short-term information using different attention streams. Furthermore, we also compare against Transformer-XL, which also divides the input into segments similar to the proposed model. 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	64×64	128×128	64×64	128×128
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

4.1 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 without top-down conditioning. We can see that the model suffers a significant drop in performance without top-down conditioning thus showing the importance of top-down conditioning. Results averaged across 3 seeds.

	CIFAR100	
Model	64×64	128×128
ViT + TLB	79.17	59.19
w/o Top-Down Condn.	76.04	49.22

We test our hypothesis on the CIFAR10 and CIFAR100 [44] 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 [21] 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 8.1.

One essential component of our model is top-down conditioning. Top down information helps integrate information from the past into the the perceptual module. We hypothesize that top-down conditioning enables the perceptual module to pay attention to the most important information in the input. We conjecture that without this, the perceptual module would have no prior knowledge of which input patches to pay more attention to, hence degrading the performance of the model. To test this, we run an experiment where we omit top-down conditioning. We show the results for this ablation in Table 2. We can see that the performance drops significantly without top-down conditioning thus showing the importance of top-down conditioning.

Self Supervised Learning. Many recent works have used vision Transformers for self-supervised learning [9, 2, 30, 12, 48, 47]. 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 use the CIFAR10 dataset [44] for our experiments. We pretrain the model for 400 epochs and evaluate the pretrained model at different epochs using linear probing. We use the CIFAR10 dataset for linear probing as well. We present the results for this experiment in Figure 2. We can see that the proposed approach outperforms SiT in all cases thus showing the effectiveness of the proposed architecture for self-supervised learning. For additional details regarding the setup, we refer the reader to Appendix section 8.2.

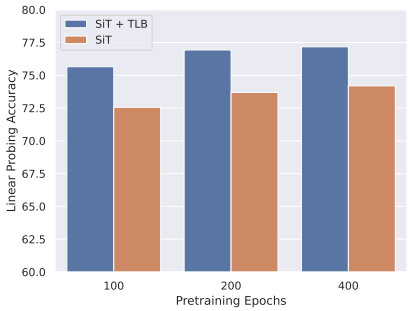

Figure 2: **Self Supervised Learning**. Here we show the result of linear probing on the CIFAR 10 dataset for different pre-training epochs. We can see that the proposed SiT + TLB approach outperforms SiT.

4.2 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, 37]. 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 [10] 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 8.4.

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 mea-

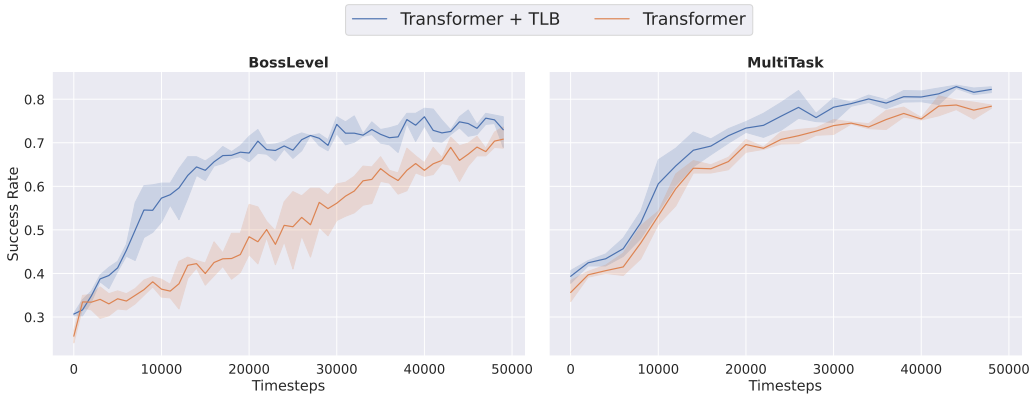

Figure 3: **Single Task BabyAI**. (Left) Here we compare the performance of Transformer and Transformer + TLB on the BossLevel task from BabyAI. We can see that while both the models converge to a similar success rate, Transformer + TLB converges faster than Transformer. **Multi Task BabyAI**. (Right) 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.

sures 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.

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 3 (left) and present the results for the other tasks in Appendix Figure 4. *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 3 (right). We train the model on 8 environment - 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 $\boldsymbol{C}$. During training, $\boldsymbol{C}$ 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 $\boldsymbol{d}$. The state is processed using a convolutional encoder into an embedding of size $\boldsymbol{d}$. 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.

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 8.5.

4.3 Temporal Latent Bottleneck for Language Modeling

We also apply the Transformer + Temporal Latent Bottleneck Model to language modelling on the enwiki8 dataset [52]. We use Transformer XL [19] as our baseline. We find that Transformer XL achieves a performance of 0.99 bit-per-byte while the proposed model achieves 0.97 bit-per-byte (lower is better). Thus showing that the proposed model is useful in language modeling as well.

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 nature of the proposed model will be extremely useful in modelling such hierarchical information. The temporal latent bottleneck which operates at a slow scale can behave as a *composer* that composes low-level information in relevant way to solve the desired task.

Table 4: **Long Range Dependencies**. Here we compare the performance of the proposed model against the recently proposed long-short Transformer model [68] 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	$38.2_{\pm 0.0001}$	$67.8_{\pm 0.0001}$

For this experiment, we use the same setup as [68]. 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 [68], 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 8.3.

5 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

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] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[7] 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.

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

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

[10] 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.

[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] 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.

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

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

[26] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[27] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. *CoRR*, abs/1303.5778, 2013.

[28] Alex Graves and Jürgen Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2008.

[29] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610, 2005. IJCNN 2005.

[30] 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.

[31] Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. Tracking the world state with recurrent entity networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[32] Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016.

[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] 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.

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

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

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

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

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

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

[41] 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.

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

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

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

[45] David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron C. Courville, and Christopher J. Pal. Zoneout: Regularizing rnns by randomly preserving hidden activations. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[46] Seung Hoon Lee, Seunghyun Lee, and Byung Cheol Song. Vision transformer for small-size datasets. *CoRR*, abs/2112.13492, 2021.

[47] 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.

[48] 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.

[49] Yahui Liu, Enver Sangineto, Wei Bi, Nicu Sebe, Bruno Lepri, and Marco De Nadai. Efficient training of visual transformers with small datasets. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 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] Matt Mahoney. Large text compression benchmark. 2011.

[53] 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.

[54] 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.

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

[56] 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 amp; insights from training gopher, 2021.

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

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

[59] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[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] 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.

[63] 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.

[64] 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.

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

[66] 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.

[67] 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.

[68] 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 8
   (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]