
Hierarchical Transformers

Shanai Nair Robert Lieck

Department of Computer Science
Durham University

Abstract

Natural data is inherently hierarchical in structure. Yet, current state-of-the-art models do not reflect this structure within their architectures. Even hierarchical models feature static, unlearned structures that only loosely approximate the actual data hierarchy. We propose that aligning model structure more closely with data structure could enhance both generalizability and interpretability. This paper introduces an encoder-decoder architecture based on the Transformer, which incorporates inductive biases that guide the model to learn a structure that more closely resembles the true structure of the data which it is tasked to model. Concretely, the architecture obtains an information bottleneck through the iterative merging, and subsequent unmerging, of embeddings. The discrete decisions of which embeddings to merge and unmerge are learned, as are the merging and unmerging operations themselves. Together with the use of depth-wise recurrence, these mechanisms are designed to bias the model toward leveraging the hierarchical structure within its training data, as a means of ensuring quality compression. We assess the extent to which these mechanisms work as intended by training with a simple reconstruction objective. We find that the network converges without learning a sophisticated merging policy, which impedes performance at the highest levels of compression. Following a careful investigation, we attribute this issue to a form of the credit assignment problem, encountered during the training of the merging policy. Our findings provide valuable insights into the nature of the model and suggest a clear direction for future research.

1 Introduction

Natural data such as images, language, decision making, and music exhibit an abundance of structure that can be expressed in terms of hierarchy [1–6]. Complex phenomena are, by their very nature, recursive constructions of increasingly simpler structures. Hence, it is logical to suggest that consideration of this hierarchical structure is valuable when asked to represent or model such data.

Despite this, current state-of-the-art models – most of which are based on the Transformer architecture [7] – do not contain any inductive biases toward learning this structure [8–10]. The structure of the models does not reflect the structure of the data.

Variants which do have hierarchical structures, have fixed heuristic architectures which must be manually designed depending on the domain. Further, they only coarsely approximate the true hierarchy within the data, given that they operate at uniform and unlearned degrees of granularity at each hierarchical level [11–13].

We suggest that guiding the model into learning a structure that more closely resembles the structure of the data, through the use of carefully designed inductive biases, can provide the models with valuable benefits such as increased generalizability and interpretability.

We propose a domain-general transformer-based encoder-decoder architecture which obtains an information bottleneck through the implicit prediction of the part-whole hierarchy of the input data. Concretely, the architecture operates in the embedding space, and obtains a compressed representation through the iterative merging of tokens¹. This compressed representation is then passed to the decoder, which gradually decomposes tokens back down, to ultimately obtain a token population of the same cardinality as the original.

Three key design choices were made which allow the model to learn a structure that better resembles the inherent hierarchy within instances of its training data:

1. The architecture is depth-wise recurrent, which is to say that the same layer is applied repeatedly in each of the encoding and decoding phases, giving the network the potential to recognize structures across all levels of granularity and abstraction.
2. The choice of which tokens to merge and unmerge² is learned. That is, the network learns policies from which these decisions are sampled.
3. The merging and unmerging operations are also learned. Rather than naive approaches to merging and unmerging, such as averaging or duplication, the network learns non linear transformations for these processes.

In the context of an information bottleneck, we expect these design choices to bias the model toward learning to leverage the inherent structure in the data by merging tokens that either share a significant amount of information – whether structural or semantic – or are correlated, and do so in a manner that leverages this redundancy.

This formulation satisfies the original ambition to make the model structure more reflective of the data structure. Drawing an analogy to existing hierarchical models, which have a fixed number of hierarchical levels where each level processes the representation at a uniform level of granularity, our approach also has a fixed number of levels (number of encoder layers). However, each level is able to process data at different, instance-dependent, and learned levels of granularity – depending on the merging decisions that have been taken up to that point.

In this paper, the model is evaluated against a classical reconstruction objective. However, the underlying architecture is more broadly applicable. The architecture can be readily adapted for tasks such as masked language modeling or classification, and is also suitable as a substitute for U-Net style architectures [14]. Theoretically, it can also be used for generative modelling. It is important to note that the efficacy of the model across these various applications has not been fully tested. As such, details of these applications are precluded from the main paper but are provided in Appendix A.1.

Given this reconstruction objective, our findings indicate that the performance of the model is constrained by the sophistication, specifically a lack thereof, of the merging policy being learned. This inability of the network to learn an effective policy stems from a manifestation of the credit assignment problem, as encountered in reinforcement learning settings [15, 16]. Evidence and justification for this assessment, as well as a brief discussion of future work to resolve this issue, are provided in Sections 4 and 5.

2 Related Work

To reiterate, our depth-wise recurrent encoder-decoder architecture is domain-general and obtains an information bottleneck by merging tokens using a learned policy and learned operations, before attempting to reconstruct the original representation using a symmetric decoding process. Together, these design decisions work to differentiate the architecture from existing contributions.

¹In this work, we occasionally employ the term ‘token’ in instances where ‘embedding’ would be more accurate, essentially using the terms interchangeably. This choice is made to maintain consistency with existing literature that discusses ‘token merging,’ and because the term ‘token’ is (linguistically) more natural within certain contexts.

²Similarly to our usage of ‘token,’ we employ the colloquial term ‘unmerge’ and its derivatives, including ‘unmerging,’ to describe the process of deriving two tokens from one. We find these terms are more intuitive than other, more formal, descriptors.

Information Bottleneck

Our architecture can be used for autoencoding, or it can be used in place of U-Net style architectures. Classical forms of these architectures obtain an information bottleneck through the use of convolutions [17, 18, 14, 19]. We suggest that although convolutions are natural for certain domains like vision, they are unnatural for others, like language and music [12, 20]. Therefore, in the spirit of remaining domain independent, our architecture strives to achieve an information bottleneck without the use of convolutions.

Transformers

Our architecture is based on the Transformer [7], utilizing a Transformer encoder at various points in the network, along with a self-attention inspired mechanism as part of the process for deciding which tokens to merge.

In each of the encoder and the decoder of our network, we share weights between layers. This is similar to Universal Transformers [21]. Indeed, we use an identical method for injecting ‘timestep information’ into the network, to allow it to differentiate between layers at different depths.

We use the patching technique of Vision Transformers [8] to translate images from the pixel space to the embedding/token space. We note that this method, which can be viewed as a convolution operation, is applied only for vision tasks, for efficiency reasons. It is not strictly necessary and can be dispensed with, at the cost of adding two extra layers to the encoder.

The primary difference between our work and these contributions is our aforementioned use of an information bottleneck to guide the model into considering the structure and hierarchy of the data more explicitly.

The Perceiver IO and its predecessor, the Perceiver, also introduce an information bottleneck [22, 23]. This is achieved by distilling input data into a reduced latent representation using cross-attention. Aimed at domain generality and scalability, they typically perform this reduction in a single step. In contrast, given our focus on understanding the structure and hierarchy of data, we opt for a gradual reduction of the latent space size through our previously mentioned token merging process. Nevertheless, we draw significant inspiration from the Perceiver IO, particularly in adopting their method of employing cross-attention to map from the embedding space back to pixel space during decoding. Unlike their approach, which decodes a maximally compressed latent representation, we query a decompressed embedding space, obtained through unmerging tokens in the decoder.

Token Merging

A number of approaches that use token merging exist; all of them focus on improving the efficiency of transformers [24–27, 11, 28]. Our approach aims to learn both the decision of which (pairs of) tokens to merge, as well as the merging operation itself – via a fully connected feedforward network.

Due to their unified focus on improving efficiency, most of the existing approaches merge subsets of tokens via average pooling [24, 25, 11, 27], whilst the remainder learn a weighted sum across all tokens [26, 28]. These methods of combination are not suitable for our purposes since the architecture is required to not only merge tokens which are similar, but also tokens that are distinctly different but closely correlated.

Then, regarding the task of selecting which tokens to merge, some perform a soft merging of all tokens [26, 28], whilst most others use convolutions to merge tokens which are adjacent [25, 11]. One approach uses attention key similarity to decide which pairs of tokens to merge [24]. Whilst elegant, this method does not allow for the merging of different but correlated tokens, as above; an essential part of being able to learn hierarchy. Finally, there exists another approach that aims to learn which tokens to merge, where they use a simple scoring network to segment a sequence into sections [27]. This approach does not extend naturally to domains other than language, nor does it align with our aims of learning about structure and hierarchy. Therefore, we require a novel mechanism. Our proposition for such is described in Section 3.

Hierarchical Models

There are numerous ways of designing hierarchical models. The common differentiator between our architecture and most of the existing approaches is that our architecture seeks to learn the hierarchy whilst the others use heuristics to approximate it [12, 13].

One approach also seeks to learn the hierarchy, however this approach is more complicated, has not been made to work in practice, and is evidently different from our own approach [6].

Graph Neural Networks

Certain contributions in the field of graph neural networks (GNNs) are aimed at, or can be utilized for, obtaining hierarchical representations of graphs, for tasks such as classification [29–32]. The work most similar to ours learns which nodes in a graph to pool in order to obtain the next level in the hierarchy [29]. The key conceptual difference between our work and theirs is that our work computes a hard merging of pairs whilst they produce a soft clustering (weighted sum) of all nodes. This approach is intuitive for their purposes of classification; however, any tasks beyond this become challenging as there is no obvious way to reverse this process and decode the compressed representation. Their approach also prevents the use of weight sharing, a key requirement of our architecture, to facilitate efficient generalisation.

Specifically, depth-wise weight sharing requires that layers remain invariant to the number of input tokens. Clearly, this is not possible for the soft methods described previously, in both contexts of GNNs and token merging in transformers.

Policy Gradient Methods

Of course, introducing any hard decision prevents the direct flow of gradients to the network responsible for making this decision. We estimate these gradients using a formulation of the score function estimator (REINFORCE [33]). We do not use variance reduction techniques such as subtracting a baseline, since these are neither optimal, nor do they offer a guaranteed reduction in variance [34]. Whilst alternatives to REINFORCE exist, such as the Straight-Through Gumbel-Softmax estimator [35, 36], we do not use these as they are more complicated, yet biased estimators of the gradients. Our approach is simpler, while remaining unbiased. Further details of our approach are specified in subsection 3.1.5.

3 Methods

We present two distinct classes of methods: foundational and auxiliary. The ‘Foundational Methods’ subsection details the key components of our final architecture, discussing the encoder, the decoder, and model training techniques such as the ‘stochastic merging schedule.’ The ‘Auxiliary Methods’ subsection covers approaches that were explored during development but are not included in the final design. These methods are included to provide context for further discussions in Section 5.

3.1 Foundational Methods

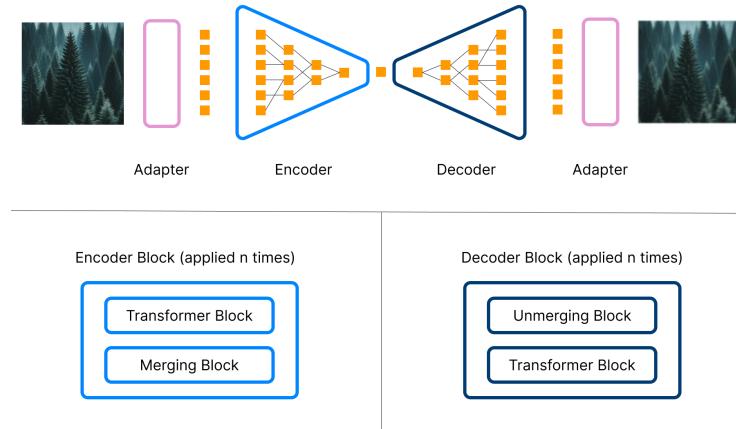


Figure 1: **Top:** A diagram that depicts the encoder-decoder nature of the architecture, where an information bottleneck is obtained by merging tokens. Adapter layers convert between pixel space and the embedding/token space. **Bottom:** The encoder comprises a transformer block followed by a merging block. The decoder comprises an unmerging block followed a transformer block.

3.1.1 Adapter Layers

As mentioned previously, the encoder and decoder of the architecture operate on the embedding space. As seen in Figure 1, we map to and from this space using adapter layers. These adapter layers are the only part of the network which are domain dependent.

For discrete applications, such as masked language modelling, we map to and from the embedding space in the same way as existing transformer based language models [9, 37]. That is, embeddings are obtained via an embedding layer, and token predictions are obtained using a classification head (a learned linear transformation, followed by a softmax over the vocabulary).

For vision applications, where other continuous domains follow an analogous process, we obtain embeddings by splitting the input into $n \times n$ ‘patches’ and linearly embedding each patch [8]. For CIFAR10 we have $n = 2$, resulting in an initial token population of 256 [38]. Before embedding the raw RGB values of the pixels in each patch, we concatenate Fourier positional encodings along the channel dimension. Concrete details of this process can be found in Appendix A.2.

Then, to map back from the embedding space to the pixel space, we directly employ a technique from [23], wherein the embedding space is queried for the RGB values of each pixel. This is achieved by utilizing the Fourier positions of each point as attention queries, and where the keys and values are linear projections of the set of tokens produced by the final layer of the decoder.

3.1.2 Encoder

The encoder comprises a stack of n identical layers, whose weights are tied. To enable the network to distinguish between layers at different depths, we generate timestep embeddings corresponding to each layer and add them to the latent representations entering these layers. This technique was adopted from [21]; further details on this process are provided in Appendix A.2.

Each encoder layer consists of two sub-blocks, as depicted in the bottom left portion of Figure 1.

The first of these sub-blocks is a standard transformer encoder [7], which fundamentally consists of a multi-head self-attention layer, followed by a token-wise fully connected feed-forward layer. For our experiments, we use the modern, but not state-of-the-art, pre-norm formulation – with GELU activation functions [39, 8]. We note that these standalone transformer blocks in the encoder and decoder are optional, but provide a moderate boost in performance (Section 4).

The second of these blocks is our token merging block. During the forward pass of the network, each merging block takes as input a set of tokens $x \in \mathbb{R}^{B \times T \times C}$ (where B is the batch dimension, T is the number of tokens, and C is the embedding dimension), as well as an argument r , which denotes the number of pairs of tokens to merge. r is typically different for each layer, and is defined by a hyperparameter of the network known as the merging schedule. Hence, the output of each merging block is $x' \in \mathbb{R}^{B \times (T-r) \times C}$.

To decide which pairs of tokens to merge, the network uses a self-attention inspired mechanism. First, key (K_p) and query (Q_p) projections of x are obtained – each with dimension $B \times T \times h \times d_p$, where h is the number of attention heads and $d_p = \frac{C}{h}$. Let l_h be defined as:

$$l_h = \frac{Q_p K_p^T}{\sqrt{d_p}}. \quad (1)$$

Logits ($B \times T \times T$) are obtained from this l_h ($B \times T \times T \times h$) by combining heads using a learned weighted sum³.

Two matrices are obtained from these logits, one via a softmax, and the other via a log-softmax. We refer to the first of these matrices as the merging policy, and the second as the log-probabilities.

We detach the merging policy from the computation graph and repeatedly sample from it to obtain r pairs of tokens to merge. Concretely, each index (i, j) in each $(T \times T)$ policy, represents the probability of token i being merged into token j – where the asymmetry of the attention operation is

³Standard concatenation of heads using a linear projection is infeasible in a shared-weight context when T is a variable.

preserved, to permit this notion of directional merging. We use masking to enforce the constraints that a token cannot be merged with itself, nor can it be involved in more than a single merge operation.

Unlike the merging policy, the log-probabilities remain part of the computation graph. They are used to provide an unbiased estimate of the gradients for the discrete merging decisions sampled from said policy. Details of this, REINFORCE-style, process are described in Subsection 3.1.5. These gradient estimates are used to train the policy networks – those networks that generate the K_p and Q_p projections of x , and thus, determine the merging policy.

Given the decision of which tokens to merge, we perform the actual merging operation by concatenating the tokens and passing them through a small multi-layer perceptron (MLP). This MLP, referred to as the merging operation, reduces the dimensions of the concatenated tokens to that of a single token. Therefore, replacing the two original tokens with this single merged representation reduces the token population by 1. Repeating this for r pairs is how we obtain a reduced token population of $T - r$, at this stage in the network.

However, these aren’t the final representations that are returned by the merging block. In particular, the reduced set of $T - r$ tokens themselves generate a set of attention queries (Q_x). These are used in a cross-attention operation with a set of keys (K_x) and values (V_x) derived from the original (unmerged) set of T tokens. This technique, introduced in a similar context by [25], allows the network to augment the compressed representations with information from the uncompressed representations, before they are passed to another encoder block or to the decoder. Whilst valuable, we note that this step is not essential by any means (Section 4). An alternate formulation of the merging block involves using the same set of keys for both the derivation of the merging policy, and the cross-attention operation. This is in contrast to the current $K_x \neq K_p$ formulation, where distinct sets are generated by separate networks. The intuition behind, and details of, this formulation are discussed in Appendix A.3, but it is otherwise disregarded due to inferior performance.

To summarize, the token merging block reduces the initial token population x by a factor of r , by merging r distinct pairs of tokens. The decisions of which tokens to merge are sampled from a policy which is obtained, in the $h = 1$ case, by computing the inner product between two distinct linear transformations of the token population. The networks that generate these linear transformations are trained using gradient approximations derived through a REINFORCE-style approach that is detailed in Subsection 3.1.5. The merging operation itself is performed by a MLP that transforms two tokens into one. Finally, the reduced set of tokens x' are augmented via a cross-attention operation, where the keys and values are derived from the original set x .

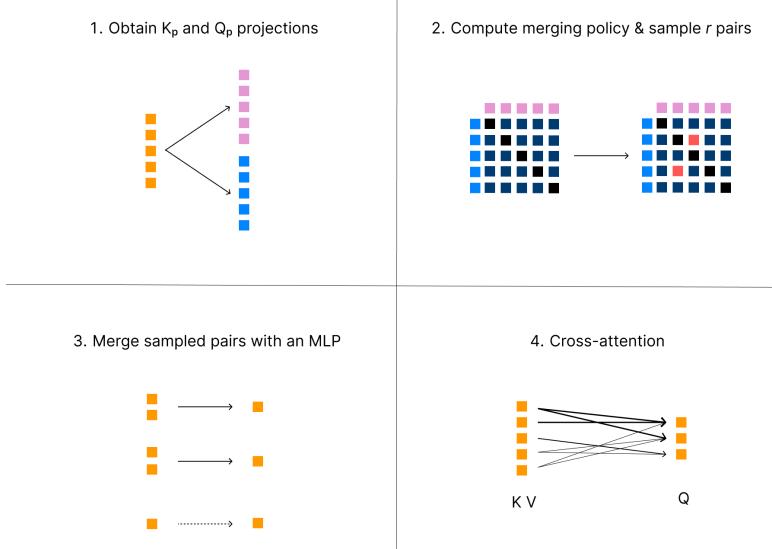


Figure 2: Summary of the token merging block stages: K_p and Q_p projections of x are first obtained, followed by computation and normalization of logits via softmax to generate a policy. This policy is sampled to merge selected token pairs, while other tokens remain unchanged. A final cross attention mechanism refines the compressed representations with information from the original representations.

3.1.3 Decoder

The decoder is, by design, almost an exact reflection of the encoder, where the main break in symmetry relates to how the unmerging policy is obtained. Concretely, the decoder swaps out the merging block of the encoder for an unmerging block, and places this before the transformer block, rather than after (Figure 1). Then, given a set of compressed representations $x' \in \mathbb{R}^{B \times (T-r) \times C}$, the unmerging block unmerges r tokens (as defined by the merging schedule) to derive a reconstructed representation $x \in \mathbb{R}^{B \times T \times C}$.

Rather than using attention, a simple scoring function (MLP) is used to decide which tokens to unmerge at a particular level of the network – a choice reflective of the inherent asymmetry of the merging and unmerging problems. This function yields a score for each token, and a softmax is used to obtain a policy from which the unmerging decisions are sampled. Like in the encoder, a matrix containing the log-probabilities is obtained. This matrix is used to obtain gradient estimates when training the unmerging policy.

The unmerging operation itself, is learned by another MLP – used to obtain two tokens from one.

Finally, cross-attention in the unmerging block generates keys and values from the compressed representations, and queries from the newly decompressed representations – in contrast to the encoder where the process is reversed.

Decoder blocks are stacked with shared weights until a token population of the same cardinality as the original (pre-encoding) is obtained. These representations are then passed to one of the aforementioned adapter layers to obtain a final reconstruction.

3.1.4 Stochastic Merging Schedule

For the highest levels of compression the network struggles to converge due to a lack of signal. Recall that during the forward pass, both the encoder and decoder receive a merging schedule, as well as the tokens themselves. Where, for example, if the input to the encoder contained 256 tokens, the schedule [128, 64, 32, 16, 8, 4, 2, 1] would result in 8 layers (applications of the encoder block), and a final latent representation consisting of only a single token. We resolve the convergence issue by gradually increasing the level of compression as training progresses. We refer to this as having a stochastic merging schedule. Experimental results reveal that this method expedites the convergence of all models, even those who converge successfully on their own.

Specifically, we define a variable $0 \leq \rho \leq 1$ that linearly increases in this range during training. ρ defines the probability of progressing from any given layer l to layer $l + 1$. The depth d is sampled with a probability proportional to ρ^d , that is from a Geometric distribution, constrained to $0 < d \leq \text{num_layers}$. Naturally, the decoder is always supplied with the same merging schedule as the encoder.

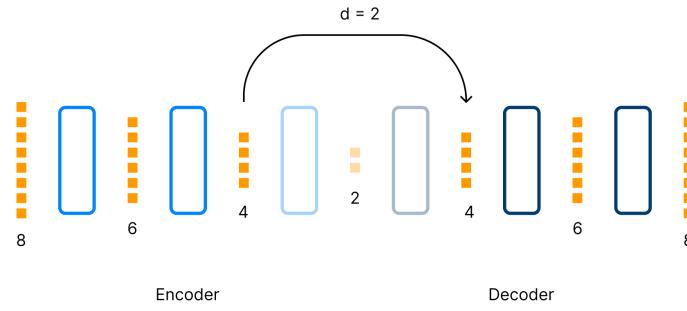


Figure 3: Illustration of a stochastic merging schedule of $[2, 2, 2]$ where the value at each index of the list represents the number of tokens merged at the corresponding level of the encoder. In this case, $0 < d \leq 3$ was sampled to be 2, which means that the encoder and decoder are only applied twice respectively. Therefore, the stochastic merging schedule can be interpreted as a form of skipped connection, where the information is allowed to flow directly from a particular depth in the encoder to the corresponding depth in the decoder.

3.1.5 Gradient Approximation Through Discrete Decisions

Token merging and unmerging decisions are discrete, determined by sampling from the network-generated policies. This discrete sampling process is non-differentiable, precluding gradient propagation to the sub-networks responsible for policy generation. To facilitate the learning of these policies via gradient-based optimization, gradient estimates with respect to the weights of these sub-networks are required.

Formally, given our scalar-valued and differentiable loss function $g(Y, \theta)$, we want to optimize the expected loss:

$$E(\theta) := \mathbb{E}_{Y \sim p_\theta}[g(Y, \theta)]. \quad (2)$$

We can rewrite the gradient of this expectation as:

$$\nabla E(\theta) = \mathbb{E}_{Y \sim p_\theta}[g(Y, \theta)\nabla_\theta \log p_\theta(Y)] + \mathbb{E}_{Y \sim p_\theta}[\nabla_\theta g(Y, \theta)] \quad (3)$$

$$\approx \nabla_\theta \sum_{Y \sim p_\theta} [\text{sg}[g(Y, \theta)] \log p_\theta(Y) + g(Y, \theta)], \quad (4)$$

where $\text{sg}[\cdot]$ is the stop-gradient operator (PyTorch `detach()`) and in the second line, the expectation is approximated by sampling from p_θ [40, 41]. The first term corresponds to the standard REINFORCE score function estimator, while the second term accounts for the fact that g is also learnable and not fixed, as assumed in REINFORCE [33].

As discussed in previous sections, we obtain a matrix containing the log-probabilities directly from the logits. This allows the policy from which the decisions are sampled to be detached from the computation graph, so that any masking that takes place during the sampling of these decisions does not effect the probabilities themselves – which is to say that we optimise the original unconstrained distribution.

The log-probabilities can be accumulated into a single variable, then scaled and backpropagated through once the loss is calculated (after completing a full forward pass through the network). However, for simplicity and memory efficiency, we opt to directly accumulate the gradients by invoking the backward operation on these sampled log-probabilities after each merging and unmerging step. After the loss is calculated following a full forward pass, we re-scale the gradients before applying the optimizer. This approach is feasible without any complications because the sub-networks whose gradients are being estimated do not participate in any other computations.

3.2 Auxiliary Methods

Early experimental results revealed that although the architecture was able to reach convergence at high levels of compression, the reconstructions themselves remained blurry. Investigations into this revealed that the merging network was not learning a particularly useful policy (Section 4). One class of theory for why this was the case is that the model lacked sufficient ‘incentive’ to learn a valuable policy. That is, despite the ability of the model to learn a valuable policy in theory, there was not sufficient incentive for it to do so in practice. The methods discussed in this subsection aimed to address this theory by making the task of learning a useful policy more pronounced. Despite none of these methods proving successful, we discuss them here for the sake of completeness and to lay the groundwork for subsequent discussions (Section 5).

3.2.1 Reducing the Capacity of Tokens

The observation that the network was able to converge and produce realistic, albeit very blurry, reconstructions without learning a particularly useful policy – or even using any attention mechanisms (Section 4) – suggests that the merging network possessed enough expressivity to learn a general operation for merging tokens, irrespective of their content. Consequently, in the later stages of training, when further improvement becomes elusive, learning a sophisticated token merging policy is no longer natural. This is because the merging network persistently executes a generic merging operation, one that is agnostic to the tokens’ content. Consequently, diverging from this entrenched position within the optimization landscape proves challenging.

We theorized that limiting the amount of information that can be contained within a single token would force the model to make more judicious decisions regarding token merging. This strategy, referred to as reducing the tokens’ capacity, was implemented by:

First, augmenting the token vectors \mathbf{x} at each layer’s entry point with unit Gaussian noise:

$$\mathbf{x}' = \mathbf{x} + \mathbf{n}, \quad \text{where } \mathbf{n} \sim \mathcal{N}(0, I). \quad (5)$$

And second, applying a regularization on the noised vectors to penalize their squared L_2 norm:

$$\lambda \cdot \|\mathbf{x}'\|_2^2, \quad (6)$$

where λ represents the regularization strength – which is equivalent to that of Variational Autoencoders with a fixed variance of 1 [18].

We selected this technique over a simple reduction in the embedding dimension of each token, since it would allow us to re-introduce capacity to the tokens as training progressed, once a policy was being learned and the merging network was performing accordingly. Therefore, the result would be a network with full capacity, but one which had also learned useful merging and unmerging policies. As mentioned, in practice, this method was ineffective in training a more useful policy.

3.2.2 Reproducing Intermediate Token Populations

The intuition that learning sophisticated merging and unmerging policies should be straightforward, arises from viewing the decoder as a mechanism designed to reverse the operations of the encoder. From this perspective, each layer of the decoder is seen as a direct counterpart to a corresponding layer in the encoder. However, the mere symmetry between encoder and decoder layers may not inherently drive the architecture towards adopting this intended behavior.

To address this, we introduced two architectural variants specifically designed to promote the model’s ability to accurately reproduce intermediate token populations. By doing so, we aim to make the learning of suitable merging and unmerging policies a more natural outcome of the architecture’s learning process.

The first of these variants attempted to achieve this by using an auxiliary loss function (Sinkhorn Distance) borrowed from Optimal Transport Theory, and applied between corresponding token populations in the encoder and the decoder [42]. The gradients from this term would encourage the model to minimize the transportation cost between the encoder and decoder token populations, effectively aligning the distributions of tokens at corresponding stages of processing [43].

The second variant was slightly more direct, implementing a pre-training stage that used teacher forcing. That is, for each r in the merging schedule, the merging block would select and merge r pairs of tokens. Instead of passing these to another encoder layer to be compressed further, these are passed straight to the decoder, which is tasked with unmerging the r tokens that were merged. A mean squared error (MSE) loss is computed between the original tokens and their corresponding reconstructions. Gradients from this MSE loss are used to directly train the merging and unmerging operations. The merging decision is trained using REINFORCE as usual, but where the log-probabilities are re-scaled using the sum of these token-wise MSE reconstruction losses. Further, since the unmerging policy does not control which tokens are unmerged, due to teacher forcing, it is trained using a cross-entropy loss on the inverse merging decisions. That is, it learns to classify which tokens should unmerged, rather than controlling the decision itself during the pre-training stage.

As stated earlier, both of these variants failed in improving the performance of the learned policy, which is why they are discussed only briefly. Moreover, whilst being the most relevant to future discussions, the architecture variants discussed in this subsection were not the only ones that were tested. Please refer to Appendix A.4 for a discussion of two more architecture variants, which were inspired by Sparsely-Gated Mixture of Experts techniques [44].

4 Experiments

In this subsection we evaluate our ‘original’ model, as presented in Subsection 3.1, by testing its performance with classification and reconstruction objectives. Then, we attempt to assess the role of the policy and the role of attention within the model by comparing it with appropriately crafted variants. All model variants use the same hyperparameters as the original, details of these can be found in Appendix A.5, along with a compact summary of all model variants. All models operate on an initial token population of 256, where each token also has a dimension equal to 256.

4.1 Classification Performance

As alluded to in Section 1, the architecture is applicable for a wide range of objectives. One such objective includes using an encoder-only setup for classification. A brief assessment of the efficacy of such an approach is presented in this subsection.

We use a merging schedule of [128, 64, 32, 16, 8, 4, 2, 1] to reduce the input representation to that of a single token. This token is then passed through a classification head (a learned linear transformation, followed by a softmax over the classes). Our model achieved a classification accuracy of 71 ± 0.20 , while a standard vision transformer [8] reached 76 ± 0.15 . These statistics were derived from five separate runs for each model. Both models were trained under identical conditions using the same hyperparameters, except for the embedding dimensionality of the vision transformer, which was adjusted to ensure both models had an equivalent number of parameters ($2.8M$). Due to computational constraints, we could not perform experiments at the native scale of the vision transformer.

We recognize that vision transformers perform optimally at significantly larger scales – in terms of parameter count, training data volume, and training times. But with both architectures being transformer-based, we hypothesise that they have similar scaling behaviour. Moreover, we did not experiment with data augmentation techniques or engage in hyperparameter optimization for either model at this scale. Nevertheless, these results substantiate our claim that the architecture can be successfully trained on various objectives and provides a performance that is competitive to the state-of-the-art in a reduced setting.

4.2 Reconstruction Performance

In Table 1, we evaluate the reconstruction quality of the original model at various levels of compression. The first column represents the percentage of tokens merged per layer. For consistency, each merging schedule is defined to have a length of 8; hence, this percentage directly determines the merging schedule that is passed to the model. For example, merging 100% of tokens at each level corresponds to a merging schedule of [128, 64, 32, 16, 8, 4, 2, 1], which results in a single bottleneck token, and is equivalent to $\frac{1}{256} \approx 0.39\%$ of the initial token population.

% Merged per Layer	Bottleneck Tokens	MSE Loss ($\times 10^{-3}$)	Image Reconstructions
0	256 = 100%	0	
25	91 ≈ 36%	2	
50	27 ≈ 10%	9	
100	1 ≈ 0.39%	40	

Table 1: Model performance at various levels of compression. The first row illustrates the benchmark of perfect reconstructions. We observe that at the highest level of compression (with respect to the number of tokens), the reconstruction quality significantly deteriorates.

4.3 The Role of the Policy

The following experiments were performed using a [128, 64, 32, 16, 8, 4, 2, 1] merging schedule i.e. maximal compression with respect to the number of tokens. Observe that under this merging schedule, the unmerging policy becomes redundant, since every token is involved in an unmerging operation at each level of the decoder. Therefore, in the experiments that follow, the focus is on what the merging policy learns.

To evaluate the role of a learned merging policy, we compare of MSE loss of our model with two baseline versions. The first of these baselines used a uniform-random policy, that is, every token has an equal probability of being merged with every other token. The second of these baselines used the weights of the transformer block in the encoder as the merging policy itself. This ‘heuristic variant’ would therefore merge tokens which attended highly to each other. The results of these experiments are presented in the left side of Figure 4, where the boxes for each variant were derived from a sample of 5 runs for each. The right side of Figure 4 features two sets of 4×4 subplots. Within each set, the 16 individual plots visualize the merging policy for a specific token (denoted in pink) during the very first merging operation. The two distinct sets of subplots are derived from two distinct instances of the test set.

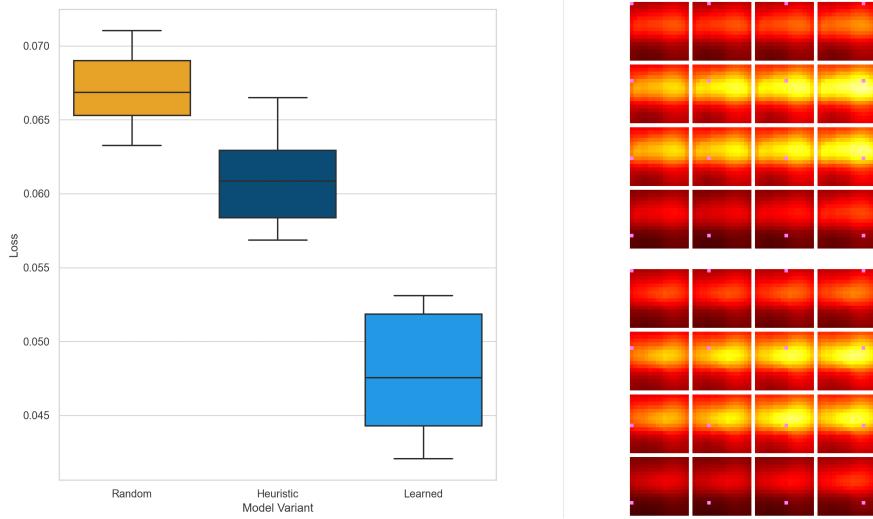


Figure 4: **Left:** Box plot comparing the reconstruction losses of learned, heuristic, and uniform-random merging policies. The plot reveals that a learned policy is more effective than both heuristic and random policies. **Right:** Visualizations of the token merging probabilities for various tokens in two distinct test instances; brighter colors indicate higher probabilities. The fact that the top and bottom collections of plots are similar indicates that the network is learning to perform a generic merging policy that is independent of the input data.

The box plot reveals the expected trend that the more ‘informed’ the policy, the lower the loss. Specifically, the losses from the various runs of the original model are consistently lower than those of the heuristic and uniform-random variants. However, the plot also reveals that this difference is small (1.9×10^{-2} shift in the mean loss of the learned and uniform-random variants). This suggests that only a rudimentary policy is being learned, a notion which is corroborated by the visualisations of the policies themselves.

Concretely, the visualizations of the merging policies for various tokens for two distinct instances of the test set are reflective of an overall trend: that the merging policy for different images is, effectively, the same. In particular, we observe that regardless of a particular token’s position in the image, and the content of the image itself, the network will always choose to merge it with tokens in the upper-middle portion of the image – although it does demonstrate some competence by activating more highly for those (pink) tokens which are actually in this region.

4.4 The Role of Attention

Observing the primitive nature of the learned merging policy, we hypothesised that the ability of the network to transmit information globally between tokens, via its highly expressive attention mechanisms, might be muting the seemingly intuitive nature of learning an effective policy. Concretely, that the architecture was leveraging its highly expressive, global, attention mechanisms to efficiently compress (and decompress) the entire latent space from one layer to another, rather than learning to perform more token-specific reconstructions as initially intended. Of course, operating in this way shifts the burden of compression unevenly onto the merging operation and makes the learning of any policy, effectively, redundant.

To test this, we ran variants of the original model, and the one with a uniform-random policy, which contained no transformer blocks or cross-attention mechanisms. Therefore, any sharing of information between tokens would be solely controlled by the merging and unmerging operations. Figure 5 presents the performance of these ablated models alongside the original variants.

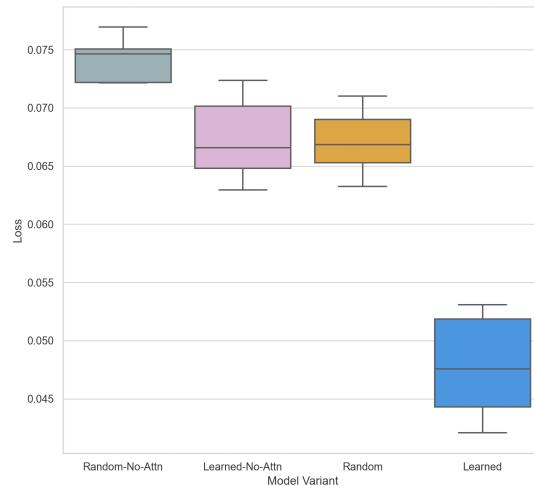


Figure 5: Box plot comparing the performance of learned and uniform-random policies in the context of the full architecture, versus an ablation with all attention mechanisms removed. The plot reveals that performance increases both in the presence of a learned policy and with attention. Crucially it also reveals that the presence of attention does not hinder the learning of a merging policy – the gap between the two leftmost boxes is no greater than the gap between the two rightmost boxes.

Figure 5 reveals that, as expected, the removal of the attention mechanisms results in a decrease in performance. However, this difference in performance is relatively small, very similar in magnitude to the effect of the policy which indicates that the model is able to perform comparably with entirely clean information flow (no global communication). The key observation though, is that the gap between the learned and unlearned variants without attention, is no greater than the gap between the variants with attention – which refutes the hypothesis that attention mechanisms prevent the learning of a useful policy due to their global expressivity.

5 Discussion

In this discussion, we attempt to refine our perspective on why the network struggles to learn an effective merging policy – a key limiting factor on the reconstruction ability of the network as it struggles to minimize the information loss incurred from being forced to merge tokens that are completely unrelated. Theories for this fall into two categories: a ‘lack of ability’ and a ‘lack of incentive.’ Lack of ability refers to the model’s inability to learn an effective policy, which may stem from insufficient signal to the networks that determine the merging policy, or from the limited expressivity of said networks. Meanwhile, a lack of incentive refers to the architectural or training conditions around the decision networks, that may make the learning of an effective policy redundant – despite the ability of the networks themselves to learn a policy, in theory.

An initial theory within the lack of incentive class suggested that the transformer blocks and cross-attention operations made learning a merging policy unintuitive. Consequently, when convergence stagnated later in training, the model was already too entrenched in a specific compression methodology, which deemed the learning of an effective policy irrelevant. Experimental results refuted this particular theory (Subsection 4.4). Consequently, a number of other methods were designed to ‘motivate’ the model into learning an effective policy, these methods were discussed in Subsection 3.2, but also failed to have a significant impact on the quality of the learned policy.

These observations lead us to suggest that any lack of incentive present in the model is secondary to a more fundamental lack of ability. Regarding this, we propose that any lack of ability is caused by the networks which determine the merging policy either lacking enough signal to train effectively, or lacking sufficient expressivity. Expressivity can trivially be increased by increasing the depth of the layers which determine the merging policy, or by increasing the number of heads of the policy networks (typically $h = 1$). However, experiments deemed these changes to be ineffective.

Given these insights, we converge on the viewpoint that the merging networks suffer from a lack of signal. Specifically, the unbiased gradient approximations obtained through the REINFORCE-style approach do not provide sufficient information to train the merging networks to produce sophisticated policies. We surmise that this is caused by the network making a large number of merging decisions in between each computation of the reconstruction loss. Consequently, within a single training example, it becomes challenging for the model to discern which actions were beneficial and which were detrimental, as the reconstruction loss evaluates the collective quality of all decisions. Despite this, the model is expected to gradually develop a functional policy over the course of training by detecting statistical regularities among various sets of merging decisions. However, our experimental findings clearly suggest that the complexity of the policy that can be learned is fundamentally constrained by these issues. Indeed, this perspective provides an explanation for the observation that the network’s merging policy is indifferent to the content being compressed, as discussed in Subsection 4.3.

The problem described above is a formulation of the credit assignment problem from reinforcement learning, where the actions are the merging decisions, and the reward is the inverse of the reconstruction loss [15, 16]. There exist many solutions from reinforcement learning that aim to solve this problem, such as actor-critic methods. Despite details being unclear, we expect insights from these to be valuable when designing future iterations of the model which do learn a sophisticated policy and therefore potentially perform much better than the current iteration [45, 46, 16].

6 Conclusion

This paper proposed a transformer-based architecture designed to compress and reconstruct representations through the iterative merging and unmerging of tokens, using both learned policies and learned operations. In the context of an information bottleneck, these mechanisms are included to bias the model into learning to leverage the redundancies in the data as a means of ensuring high quality compression. Through the lens of hierarchical models, this corresponds to making the model structure more representative of the data structure by having it be learned rather than imposed. It was postulated that such a change would result in a more interpretable model which is better able to generalise – due to an increased ability to model structure by abstracting detail using hierarchy.

Initial experiments deemed the reconstruction quality of the model at the highest levels of compression to be poor. Ultimately this was attributed to the basic nature of the merging policy that was being learned. Theories explaining the ineffectiveness of the policy fell into two classes, a lack of incentive and a lack of ability. Ablation studies and experimentation with numerous architecture variants challenged the most plausible theories regarding a lack of incentive. Hence, it was determined that a lack of ability, in particular a lack of sufficient training signal to the merging networks, was the primary cause of the overly basic merging policy.

To close, we deem this work successful to an extent. We were able to conceive, design, implement, and test an architecture that fulfills the initial requirements; however, we were not able to get it to work to the level that we had hoped it would, which meant that it fell short of initial expectations. Nevertheless, we conclude this investigation with concrete ideas of why the model doesn’t work as anticipated, and have some ideas about how it can potentially be made to work in the future.

References

- [1] P. Rebuschat, M. Rohrmeier, J. A. Hawkins, and I. Cross, *Language and Music as Cognitive Systems*. OUP Oxford, 2011.
- [2] M. A. Arbib, *Language, Music, and the Brain: A Mysterious Relationship*. mit Press, 2013, vol. 10.
- [3] D. Jurafsky and J. H. Martin, *Speech & Language Processing*. Prentice Hall Press, 2000.
- [4] A. G. Barto and S. Mahadevan, “Recent Advances in Hierarchical Reinforcement Learning,” *Discrete Event Dynamic Systems*, vol. 13, no. 4, pp. 341–379, 2003.
- [5] M. M. Botvinick, “Hierarchical reinforcement learning and decision making,” *Current Opinion in Neurobiology*, vol. 22, no. 6, pp. 956–962, Dec. 2012.
- [6] G. Hinton, “How to represent part-whole hierarchies in a neural network,” 2021.
- [7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2023.
- [8] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An image is worth 16x16 words: Transformers for image recognition at scale,” 2021.
- [9] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, “Improving language understanding by generative pre-training,” 2018.
- [10] W. Peebles and S. Xie, “Scalable diffusion models with transformers,” 2023.
- [11] P. Nawrot, S. Tworkowski, M. Tyrolski, Łukasz Kaiser, Y. Wu, C. Szegedy, and H. Michalewski, “Hierarchical transformers are more efficient language models,” 2022.
- [12] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, “Jukebox: A generative model for music,” 2020.
- [13] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, “Swin transformer: Hierarchical vision transformer using shifted windows,” 2021.
- [14] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” 2015.
- [15] M. Minsky, “Steps toward artificial intelligence,” MIT Course Materials, 1974. [Online]. Available: <https://courses.csail.mit.edu/6.803/pdf/steps.pdf>
- [16] E. Pignatelli, J. Ferret, M. Geist, T. Mesnard, H. van Hasselt, and L. Toni, “A survey of temporal credit assignment in deep reinforcement learning,” 2023.
- [17] M. A. Kramer, “Nonlinear principal component analysis using autoassociative neural networks,” *AIChe journal*, vol. 37, no. 2, pp. 233–243, 1991.
- [18] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” 2022.
- [19] A. van den Oord, O. Vinyals, and K. Kavukcuoglu, “Neural discrete representation learning,” 2018.
- [20] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, “Wavenet: A generative model for raw audio,” 2016.
- [21] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and Łukasz Kaiser, “Universal transformers,” 2019.
- [22] A. Jaegle, F. Gimeno, A. Brock, A. Zisserman, O. Vinyals, and J. Carreira, “Perceiver: General perception with iterative attention,” 2021.

- [23] A. Jaegle, S. Borgeaud, J.-B. Alayrac, C. Doersch, C. Ionescu, D. Ding, S. Koppula, D. Zoran, A. Brock, E. Shelhamer, O. Hénaff, M. M. Botvinick, A. Zisserman, O. Vinyals, and J. Carreira, “Perceiver io: A general architecture for structured inputs & outputs,” 2022.
- [24] D. Bolya, C.-Y. Fu, X. Dai, P. Zhang, C. Feichtenhofer, and J. Hoffman, “Token merging: Your vit but faster,” 2023.
- [25] Z. Dai, G. Lai, Y. Yang, and Q. V. Le, “Funnel-transformer: Filtering out sequential redundancy for efficient language processing,” 2020.
- [26] D. Marin, J.-H. R. Chang, A. Ranjan, A. Prabhu, M. Rastegari, and O. Tuzel, “Token pooling in vision transformers,” 2021.
- [27] P. Nawrot, J. Chorowski, A. Lancucki, and E. M. Ponti, “Efficient transformers with dynamic token pooling,” in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2023. [Online]. Available: <http://dx.doi.org/10.18653/v1/2023.acl-long.353>
- [28] C. Renggli, A. S. Pinto, N. Houlsby, B. Mustafa, J. Puigcerver, and C. Riquelme, “Learning to merge tokens in vision transformers,” 2022.
- [29] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, “Hierarchical graph representation learning with differentiable pooling,” 2019.
- [30] J. Lee, I. Lee, and J. Kang, “Self-attention graph pooling,” 2019.
- [31] H. Gao and S. Ji, “Graph u-nets,” 2019.
- [32] T. N. Kipf, “Deep learning with graph-structured representations,” Ph.D. dissertation, University of Amsterdam, 2016. [Online]. Available: <https://dare.uva.nl/search?identifier=1b63b965-24c4-4bcd-aabb-b849056fa76d>
- [33] R. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine Learning*, vol. 8, pp. 229–256, 1992. [Online]. Available: <https://doi.org/10.1007/BF00992696>
- [34] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih, “Monte carlo gradient estimation in machine learning,” 2020.
- [35] E. Jang, S. Gu, and B. Poole, “Categorical reparameterization with gumbel-softmax,” 2017.
- [36] C. J. Maddison, A. Mnih, and Y. W. Teh, “The concrete distribution: A continuous relaxation of discrete random variables,” 2017.
- [37] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” 2019.
- [38] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” University of Toronto, Toronto, Ontario, Tech. Rep. 0, 2009. [Online]. Available: <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>
- [39] D. Hendrycks and K. Gimpel, “Bridging nonlinearities and stochastic regularizers with gaussian error linear units,” *CoRR*, vol. abs/1606.08415, 2016. [Online]. Available: <http://arxiv.org/abs/1606.08415>
- [40] M. Blondel and V. Roulet, “The elements of differentiable programming,” 2024.
- [41] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” 2019.
- [42] C. Villani, *Optimal Transport: Old and New*, ser. Grundlehren der mathematischen Wissenschaften. Springer Science & Business Media, 2008, vol. 338.

- [43] J. Feydy, “Geometric data analysis,” 2020, accessed: 2024-04-10. [Online]. Available: https://www.jeanfeydy.com/geometric_data_analysis.pdf
- [44] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, “Outrageously large neural networks: The sparsely-gated mixture-of-experts layer,” 2017.
- [45] S. Fujimoto, H. van Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” 2018.
- [46] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, “A survey of actor-critic reinforcement learning: Standard and natural policy gradients,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.
- [47] D. Reynolds, “Gaussian mixture models,” in *Encyclopedia of Biometrics*, S. Z. Li and A. K. Jain, Eds. Boston, MA: Springer, 2009. [Online]. Available: https://doi.org/10.1007/978-0-387-73003-5_196
- [48] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. B. Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L. R. Lavaud, L. Saulnier, M.-A. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, T. L. Scao, T. Gervet, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, “Mixtral of experts,” 2024.
- [49] W. Fedus, J. Dean, and B. Zoph, “A review of sparse expert models in deep learning,” 2022.
- [50] W. Fedus, B. Zoph, and N. Shazeer, “Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity,” 2022.

A Appendix

A.1 Architecture Applications

This subsection explores how the architecture can be altered to perform tasks other than reconstruction. They all assume that the policy learns a sophisticated policy, which is not the case currently.

Masked Language Modelling: The architecture can be used for masked language modelling by replacing the first adapter layer with a standard language tokenization layer, masking a proportion of tokens by replacing them with a special [MASK] token, and then predicting the masked words using a standard classification head on the refined mask representations returned by the decoder.

U-Net: The architecture can be used as a drop in replacement for U-Net style architectures by replacing the U-Net style skipped connections of the stochastic merging schedule with actual skipped connections. Concretely these can be implemented by summing the representations from a particular encoder layer with the representations from the corresponding decoder layer. In theory this formulation would work best after pre-training the policy with a reconstruction objective.

Generative Modelling: A Gaussian Mixture Model can be trained on the single token embeddings of the dataset, obtained using the encoder [47]. Consequently, new latents can be sampled and decoded using the decoder. This method was implemented, but due to the blurry nature of the reconstructions at the lowest levels of compression the quality of the sampled images was of a comparable standard.

A.2 Positional and Timestep Encodings

Positional Encodings

Positional encodings in our model serve the same purpose as in other transformer based models [7, 22, 23]. Thus, they are defined similarly.

Positional encodings for an image are generated by applying Fourier transformations across its spatial dimensions. Specifically, a normalized coordinate grid ranging from -1 to 1 is created for the image's height h and width w . These (x, y) coordinates are stored in the matrices X and Y respectively.

We define the frequency bands linearly spaced from a minimum frequency to a maximum frequency – defined as the Nyquist frequency:

$$\omega_{\min} = \frac{1}{\max(h, w)}, \quad \omega_{\max} = \frac{\min(h, w)}{2}, \quad \text{freqs} = \text{linspace}(\omega_{\min}, \omega_{\max}, \text{bands}).$$

Applying these frequencies to the coordinate grid, we compute the following Fourier features:

$$\begin{aligned} \text{PE}_{\sin, y}(i, j) &= \sin(2\pi \cdot y_{i,j} \cdot \text{freqs}), \\ \text{PE}_{\cos, y}(i, j) &= \cos(2\pi \cdot y_{i,j} \cdot \text{freqs}), \\ \text{PE}_{\sin, x}(i, j) &= \sin(2\pi \cdot x_{i,j} \cdot \text{freqs}), \\ \text{PE}_{\cos, x}(i, j) &= \cos(2\pi \cdot x_{i,j} \cdot \text{freqs}), \end{aligned}$$

where i and j index into the height and width of the image respectively, and the frequency transformations are applied independently to each axis.

These embeddings are concatenated with the original (x, y) spatial coordinates and passed through a dimension reducing linear layer to obtain the initial set of tokens.

For CIFAR10 we have bands = 64, $\omega_{\min} = \frac{1}{32}$, and $\omega_{\max} = \frac{32}{2}$.

Timestep Encodings

In order to enable the model to differentiate between applications of the same layer at different depths, timestep encodings are added to the representations (each token) passing into each layer. These are fixed, as defined in equation 7 – taken from [21, 7]. Symmetric layers of the encoder and decoder are passed the same encodings. This conceptually aligns with the ideal that the decoder learns to reverse the operations of the encoder.

$$TE(d, 2i) = \sin\left(\frac{d}{10000^{2i/n_{\text{embed}}}}\right), \quad TE(d, 2i + 1) = \cos\left(\frac{d}{10000^{2i/n_{\text{embed}}}}\right). \quad (7)$$

Where, d is the layer depth and i is the specific dimension within the n_{embed} dimensional tensor. That is, each dimension of the positional encoding corresponds to a sinusoid whose wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$.

A.3 Alternate Merging Block Formulation

An alternate formulation of the merging block uses the same set of keys in both the derivation of the merging policy and the cross-attention operation.

Concretely, let $K = K_p = K_x$. Then, let l_h be defined as:

$$l_h = \frac{Q_m \cdot \text{sg}[K^T]}{\sqrt{d_m}},$$

where $\text{sg}[\cdot]$ denotes the stop-gradient operation.

Once again, logits are obtained by performing a weighted sum over the head dimension. We observe that a detached version of the keys is utilized in the computation of these logits, which subsequently define the merging policy. Consequently, when training the merging decision with gradient estimates derived from the log-probabilities – which, by design, originate from the same logits – only the query network is trained. The network responsible for generating the keys is solely trained by their involvement in the cross-attention mechanism.

The intuition here is that the keys will be trained, via their involvement in this cross-attention mechanism, to compactly summarize the information contained within their parent tokens. Therefore, the estimated gradients obtained via the log-probabilities flow only to the query network, and its sole purpose is to decide which tokens to merge into which others by controlling the queries generated and, as a result, the merging policy itself.

Despite its inherent intuitiveness, experimental results revealed that this formulation prevented the policy from learning, which is to say that the learned policies were as effective as uniform policies i.e., not effective at all.

A.4 Additional Architecture Variants

Two additional architecture variants were experimented with. The first of which explores a different method for approximating gradients for the merging and unmerging decisions, while the second pertains to increasing the expressivity of the merging and unmerging operations themselves.

Both variants were inspired by the Sparsely-Gated Mixture of Experts (MOE) technique [44], that has been successfully applied in modern transformer-based large language models [48].

This technique allows for an increase in the capacity of neural networks by facilitating an increased number of parameters, without requiring a proportional increase in computation during the network’s forward pass.

The method involves replacing each token-wise feedforward network in a transformer with multiple networks, where tokens are routed to a discrete (top-k) subset of these networks using a scoring function [49]. Thus, the method also requires gradients to flow through a discrete decision. A biased approximation for these gradients is computed by scaling the output of each network by the probability of selection.

The first architecture variant that we tested replaced our REINFORCE-style gradient approximation approach with the one used in [50]. Concretely, instead of backwarding through the log-probabilities of the sampled tokens, the outputs of the merging and unmerging networks are directly scaled by the probabilities with which the input tokens were sampled. This approach resulted in a significant decrease in performance, likely due to the scaling probabilities being very small, due to a large $T \times (T - 1)$ action space.

The second variant that we explored involved replacing the single merging and unmerging networks with an MOE-style multiple network configuration, where a scoring function was used to distribute tokens between the different networks. This change increases the expressivity of the model, whilst still retaining the benefits of weight sharing. An interpretation of this formulation is that it allows the network to learn multiple merging and unmerging operations for different use cases. These networks, due to weight sharing, can be used at different levels of abstraction throughout the network. For example, one pair of networks could learn functions to merge and unmerge correlated tokens, whilst another pair could operate on tokens which are similar. However, this variant also resulted in a decrease in performance, due to overfitting. Nevertheless, for larger datasets this increase in expressivity could be valuable, and could overcome the limitations of weight sharing, whilst retaining the same benefits.

A.5 Experimental Details

Variant	Description	Subsection
Original	The original model (learned policy)	4.2, 4.3, 4.4
Classification	‘Original’ but encoder-only	4.1
Heuristic	‘Original’ but with a heuristic policy	4.3
Random	‘Original’ but with a random policy	4.3, 4.4
Learned-No-Attn	‘Original’ with all transformer blocks and attention mechanisms removed	4.4
Random-No-Attn	‘Random’ but with all transformer blocks and attention mechanisms removed	4.4

Table 2: Summary of model variants used in the various experiments detailed in Section 4

The following details are consistent across all model variants.

Training Hyperparameters	Value
BATCH_SIZE	256
EPOCHS	100
LR	1e-4
RHO_0	0.05
RHO_STEP	0.01
COMPLETE_MERGING_SCHEDULE	e.g. [128, 64, 32, 16, 8, 4, 2, 1]

Architecture Hyperparameters	Value
image_size	(32, 32)
patch_size	(2, 2)
fourier_bands	64
num_layers	8
n_embed	256
num_heads	8
dropout	0.1

Table 3: Model and training hyperparameters