Hierarchical Token Merging: A New Architecture to Learn Semantically Meaningful Representations

Student Name: A. Z. Durgheu
Supervisor Name: Dr R. Lieck
Submitted as part of the degree of MEng Computer Science to the
Board of Examiners in the Department of Computer Sciences, Durham University

Abstract—Complex data can often be recursively decomposed into simpler structures; yet state-of-the-art models do not leverage this hierarchical design, potentially overlooking a valuable avenue for enhanced performance. This paper introduces a hierarchical autoencoder-based architecture designed to learn compressed representations that preserve data semantics, facilitating downstream applications in reconstruction and generation. The proposed encoder iteratively merges tokenized input data within an embedding space, clustering semantically similar concepts to produce meaningful latent root-node embeddings. It is trained end-to-end alongside a symmetric decoder, which reconstructs the original data by recursively unmerging these embeddings, and supports novel data generation via sampling from a learned latent space distribution. Additionally, by framing the encoder's decisions within a reinforcement learning paradigm, the model gains greater generalisability as well as interpretability - which helps combat the black-box problem frequently exhibited by other deep learning approaches. The experimental results show promising performance on synthetic datasets and reveal important challenges that shape the trajectory of future research.

Index Terms—Hierarchical image representation, Image generation, Machine learning, Neural nets

1 Introduction

REAL-WORLD data frequently exhibit rich structural patterns that naturally lend themselves to hierarchical organization. For example, an image of a person may contain their hands or face, each of which can be further segmented into constituent parts, such as their fingers or eyes, respectively. Similar hierarchical structures are observed across other domains as well - for instance, in language, where sentences can be broken down into clauses, phrases, and individual words; or in music, where compositions consist of phrases, motifs, bars, and individual notes. Therefore, it is expected that accounting for these structured hierarchies could play a crucial role in effectively representing complex data. However, most state-of-the-art models are built on frameworks that lack the inherent architectural priors conducive to learning and exploiting these patterns [1], [2].

The proposed Hierarchical Token Merging (HTM) network adopts an autoencoder-inspired architecture, in which the encoder recursively merges pairs of embedded tokens, following a process that mirrors the structure of a binary tree computational graph. Note that these embedded tokens initially correspond to semantically meaningful segments obtained during the earlier stage of tokenization. The hierarchical merging continues until a single root-node embedding remains, serving as a compact bottleneck that encapsulates the essential information of the original input. Token pair selection for the encoder merging can also be guided by a learned policy, sampling pairs based on their conceptual similarity and their capacity to be effectively compressed into lower-dimensional representations. In the decoding phase that follows, this merging process is reversed. Starting from the root embedding, the decoder iteratively *unmerges* tokens to regenerate a token population equivalent in size and informational content to the original input.

Although merging decisions can be guided using a simple, greedy reconstruction-based heuristic, a more principled and potentially more effective alternative is to learn a policy inspired by reinforcement learning (RL). Such an approach allows the model to weigh not only immediate token similarities, but also prospective structural advantages across the merging tree. As such, the model may choose to postpone the merging of seemingly obvious token pairs, if it anticipates that a more semantically rich representation could be achieved by merging with future token nodes at deeper hierarchical levels instead. Whilst the model's capacity to predict long-term rewards and cumulative returns is inherently constrained (section 3.6) - and further hindered by its myopic design (section 5.2) - these limitations present opportunites for future enhancements.

Nonetheless, training the encoder-decoder framework in an end-to-end manner enables the learning of expressive, non-linear transformations. These go beyond simplistic operations, such as weighted averaging or other linear interpolations, which may inadequately capture the semantic complexities of the given task. In the context of this paper, the primary focus lies within the image domain, specifically on tasks involving image reconstruction and generation. This is explored using synthetically generated datasets that possess an artificial yet explicitly structured hierarchical organisation. It is expected that the iterative merging process not only yields a compact bottleneck - by reducing data redundancy whilst preserving critical information - but also facilitates the promotion of a more structured and semantically meaningful embedding space, wherein conceptually similar elements are effectively clustered.

Furthermore, HTM naturally incorporates depth-wise recurrence, with identical encoder and decoder modules repeatedly applied at each stage of the hierarchical process. Such a design is advantageous, as it enables the model to identify and represent features across multiple levels of granularity and abstraction throughout the recursive procedure. Consider, for instance, the representation of a human hand. In one scenario, the hand may constitute the primary subject of the input image, while in another, it may appear as a smaller component within a more complex scene - such as a crowded gathering of people. In the former case, the representation of the hand would persist deeper into the merging process, likely completing in the final stages; in the latter, it may be subsumed earlier, as part of broader compositional structures (e.g. arms, bodies, etc.). This capacity to abstract and encode elements at varying degrees of prominence is crucial for developing an expressive and flexible model.

In this regard, the proposed HTM network is more closely aligned with the hierarchical structure inherently present in the input data, in contrast to prior methods that attempt to exploit such patterns. Existing models typically employ processing modules across a fixed number of hierarchical levels, with each module operating uniformly at a predetermined level of granularity [3], [4]. By comparison, whilst the encoder and decoder components of HTM maintain a fixed architectural depth (i.e. a set number of layers), the number of times these modules are applied is not static, but instead, dynamically determined by the number of merges required to compress a given input. Moreover, they exhibit the flexibility to operate across varying granularities due to the aforementioned depth-wise recurrence. This ultimately allows the network to adapt its depth and processing to the specific structural complexity of each unique data instance.

In the context of existing literature, several works have sought to exploit the hierarchical patterns found in natural data. Context-free grammars (CFGs) have previously been used in natural language processing to model these structures [5], [6]. They operate by recursively expanding non-terminal symbols into subcomponents according to a set of production rules, which are typically learned from the input data using probabilistic or neural methods. This naturally gives rise to hierarchical structures, where higher-level constructs are composed of increasingly simpler constituents - conceptually identical to the merged token representations seen in HTM. The recursive design also forms tree-like structures - referred to as parse trees - allowing CFGs to capture the nested, compositional semantics of language; however, the search space for possible parse trees grows combinatorially, necessitating the use of dynamic programming techniques - such as the frequently implemented CYK algorithm [7].

The concept of token merging has also been explored within conventional deep learning frameworks, where meticulously designed algorithmic strategies are often developed to combine tokens, whilst preserving the integrity of their original feature distributions [8]. Prior research has demonstrated that such techniques can lead to enhanced performance, particularly in classification tasks involving state-of-the-art architectures - such as Vision Transformers (ViTs), [9], [10]. Notably, some approaches have sought to mitigate the quadratic computational complexity associated with the self-attention mechanism (arising from the pairwise interactions) by merging similar and spatially adjacent tokens. However, these methods typically rely on predefined similarity metrics, such as cosine similarity, to guide the merging process in a computationally efficient manner [11]. In a similar vein, other approaches integrate the information bottleneck principle with a focus on discarding redundant tokens during the merging process instead [12]. These resulting representations can then be combined with attention-derived features and passed through a multilayer perceptron (MLP) to improve the model's inference speed without harming its efficacy.

Regarding similar methods that seek to structure embedding spaces in a more interpretable and organized manner, Conceptual Variational Autoencoders (Conceptual VAEs) [13] introduce the idea of concept subspaces, which represent higher-level abstract features of data - e.g. colour, shape, lighting, etc. Conceptual VAEs typically aim to first disentangle the latent dimensions into human-understandable concepts, and then explicitly associate these concepts with specific latent variables. This facilitates more intuitive manipulations of these variables, enabling greater control over the generation process, and often resulting in an enhanced quality and diversity of the generated outputs. Section 2 offers a more comprehensive exploration of the aforementioned topics related to HTM, including a discussion of policy gradient techniques and their role in learning optimal merging decisions within the proposed framework.

Whilst the HTM architecture is theoretically extensible and holds potential as a foundational framework for a variety of other downstream applications - such as image classification, segmentation, etc. - this paper focuses on reconstruction and generation using simple, synthetic datasets as a proof-of-concept demonstration. The empirical results obtained from these experiments are encouraging, indicating the viability of the approach; however, they also expose a number of challenges. Notably, the model encounters some difficulties in learning an effective merging policy and in consistently producing visually plausible generations. Detailed experimental analyses and discussions of these observations are presented in sections 4 and 5.

2 Related Work

This section presents a detailed overview of research areas related to HTM, situating the proposed framework at the intersection of these fields. It is broadly organized into five subsections: hierarchical models, token merging, autoencoders, grammars, and policy gradient methods, each of which contributes conceptually or methodologically to the development and understanding of HTM.

2.1 Hierarchical Models

Several related works share HTM's emphasis on hierarchical design as a means to support more robust downstream inference. However, current state-of-the-art approaches in this domain predominantly build on transformer architectures [2]. Whilst the HTM architecture consists of a simple autoencoder at this stage, future extensions would benefit from integrating self-attention to mitigate the pairwise merging myopia, enabling token merges to be contextualized with respect to the entire token population as a result (section 5.2).

Many studies leverage hierarchical architectures to improve performance on tasks involving large input data [3], [14], [4], [15], [16], [17]. For instance, [3] processes long documents by first segmenting them into smaller units - analogous to tokens in HTM - and encoding each of these using BERT [18]. These segment-level representations are then aggregated at the document level using either RNN or Transformer-based models [19], [2], enabling the capture of both local and global contextual embeddings for improved document classification. Besides differing from HTM in both data domain and task - focusing on classification rather than reconstruction or generation - this study, like many others, employs a comparatively shallow and fixed hierarchical structure. In this case, a document-level layer stacked atop a conventional segment-level layer, which nonetheless results in an improved representation of large text inputs.

Another example of this is presented in [4], where the authors propose a hierarchical transformer architecture for abstractive multi-document summarization. In this setting, the input consists of long sequences of text drawn from multiple documents, and the model is tasked with synthesizing the information into a coherent summary. Y. Liu and M. Lapata argue that flat transformer architectures struggle to capture higher-level inter-document relationships - such as redundancy, contradiction, complementarity, etc. - when processing large text corpora. Consequently, a hierarchical design is adopted to effectively manage the scalability issues posed by integrating information across these extensive, multi-source inputs. Following an initial preprocessing step, in which paragraphs are ranked by relevance, the model applies a two-level hierarchical encoding procedure. Local transformer layers first encode the intra-paragraph contents alongside their positional information, after which a global transformer layer captures the inter-paragraph dependencies via cross-paragraph attention. Ultimately, the local paragraph-level embeddings are enriched by the global document-level contextual embeddings, before being passed through a final feed-forward layer for the summarisation task.

These examples highlight the effectiveness of hierarchical design in capturing relationships across varying levels of granularity, despite their static implementation of both: the modules applied, and the expected granularity of their inputs. Further contrasting with HTM, these approaches also lack the necessary modifications to align their architecture to the specific data structure of each input instance.

With a similar objective of improving performance on long sequences, some approaches focus on improving the efficiency of transformer architectures by ameliorating the quadratic complexity of self-attention [20], [21], [22], [23], [24], [25], [11]. For example, to address this challenge, [20] proposes the Hourglass Transformer, a hierarchical architecture composed of three main stages. The first stage, downsampling, compresses the long input sequences into shorter representations, using techniques such as standard pooling, and a novel method referred to as linear pooling - wherein the input is reshaped and linearly projected into fewer dimensions. The second stage involves bottleneck processing, where a transformer module operates on the compressed sequences to capture the global context and long-range dependencies more effectively. Finally, in the upsampling stage, the reduced sequences are restored to their original dimensionality, further enriched by the global contextual embeddings obtained at the bottleneck. This can be achieved by using repetition-based techniques [26], or an inverse operation they denote as linear upsampling. Whilst learnable attention layers were also explored during the downsampling and upsampling phases, the authors observed that the simpler, more computationally efficient operations yielded competitive performance. However, such lightweight operations may fall short in capturing the intricate semantic relationships required by HTM - particularly those involving higher-level, abstract tokens that emerge in the deeper stages of the merging hierarchy.

2.2 Token Merging

In alignment with the previously discussed objectives, the authors of [11] propose a Dynamic Hierarchical Token Merging (DHTM) strategy to improve the trade-off between ViT's attention complexity and accuracy on large-scale benchmarks - such as ImageNet [27]. Drawing inspiration from Hierarchical Agglomerative Clustering (HAC) [28], their method performs layer-wise token merging, incrementally building a hierarchy of increasingly abstract token representations - conceptually akin to the higher-level, near-root tokens found in HTM merging trees. Rather than applying token merging globally in a single step, a fixed number of merges are performed at each layer of the transformer, progressively reducing the token population and accelerating inference as a result. To further enhance computational efficiency, the merging process is spatially constrained: only a small set of neighbouring tokens (typically the four or eight nearest, based on grid adjacency) are considered for each merge. This operates under the assumption that spatially proximal tokens are likely to share redundant or complementary information. Among these local candidates, the most similar token pairs - measured using cosine similarity - are merged via average pooling to update the token population. By repeating this process across multiple layers, the model establishes a hierarchical design that effectively improves inference speed without incurring a significant degradation in performance.

However, within the HTM framework, the reliance on cosine similarity as a merging criterion is likely insufficient for capturing the more abstract semantic relationships between the merged tokens in the embedding space. A more appropriate strategy involves using a reconstruction-based heuristic, which evaluates the merging decisions according to how well the token pairs can later be unmerged, thereby better aligning with the goals of hierarchical representation learning. Moreover, the structure of the HTM embedding space may not be amenable to effective merging decisions based solely on cosine similarity, even under L2 normalization (i.e., a unit hypersphere). Such normalization not only risks reducing the flexibility and expressivity of the HTM model, but may still yield fragmented clusters in embedding spaces with otherwise sufficient dimensions to support meaningful compression. Additionally, as previously discussed, average pooling also presents a suboptimal merging strategy for the more abstract tokens in HTM.

Alternative token merging strategies exist [8], [10], [29], with some approaches also incorporating token pruning to further enhance their performance [30], [31]. Token pruning involves discarding redundant tokens to accelerate inference, rather than merging them together as is done in HTM. In addition to being primarily built upon transformer architectures, these state-of-the-art methods also lack a recursive framework and do not dynamically adapt to the structural characteristics of individual input instances, as HTM is designed to do. Furthermore, these methods are not situated within a reinforcement learning (RL) paradigm, where a learned policy guides the selection of tokens to merge at each iteration.

2.3 Autoencoders

While most hierarchical modelling efforts have focused on transformers, some studies have also explored hierarchical designs in autoencoders [32], [33], [34], [35], [36]. Additionally, conceptual VAEs - though not hierarchical by design - offer valuable insights into HTM's goal of enhancing the semantic structuring of its embedding space [13], [37], [38], [39].

The Nouveau VAE (NVAE) [32] exemplifies recent efforts to incorporate hierarchical structures into VAEs. Traditional VAEs typically employ a flat layer of latent variables for sampling and generation, which can prove inadequate when modelling complex, high-dimensional data such as images - the primary focus of this work. In contrast, NVAE introduces a hierarchy of latent variables distributed across multiple layers, each corresponding to different spatial resolutions. These latent variables are grouped such that the higher-level groups capture coarse, global structures (e.g., shapes and colours), whilst the lower-level groups focus on finer-grained details (e.g., shading and textures). The hierarchical aspect arises from the conditioning of each group on the representations of the groups above it. In a similar vein, [33] argues that significantly deeper hierarchical VAEs can achieve performance comparable to that of autoregressive models. Their design builds upon the framework proposed in [34], demonstrating that an N-layer VAE - where N represents the number of stochastic (Gaussian) layers - can theoretically approximate autoregressive models when N matches the dimensionality of the input data (in this case, the product of the height and width dimensions of the images). Nonetheless, these approaches rely on predefined spatial heuristics that separate global and local features in a top-down manner, whereby the top layers represent abstract semantic features, and the bottom layers learn pixel level details. In contrast, HTM does not impose such structural assumptions, instead aiming to learn these relationships dynamically from the data itself.

In addition to this, some works aim to address the well-documented limitation of hierarchical VAEs, posterior collapse [40], [41], [42]. This is where the higher-level latent variables become unused as the network's depth increases, thereby constraining the model's capacity to produce high quality representations. To mitigate this issue, the authors of [40] propose the Bidirectional-Inference Variational Autoencoder (BIVA), a hierarchical VAE architecture in which each layer comprises two groups of latent variables: one associated with the bottom-up inference path, and the other with the top-down. These groups are connected via conditional dependencies and are responsible for capturing distinct levels of abstraction from the input - thereby forming a structured latent hierarchy. Crucially, the top-down path integrates information from *all* of the higher levels, functioning analogously to skip connections in ResNet [43], and preventing posterior collapse by helping to preserve the flow of information. The bidirectional inference mechanism emerges from this dual-path structure: the bottom-up path constructs increasingly abstract latent representations from the input data, whilst the top-down path refines these by conditioning on both higher-level top-down variables and lower-level bottom-up variables. Thus, this architecture enables more expressive posterior distributions, and improves the model's capacity to represent complex data. Whilst it is a noteworthy effort to incorporate hierarchical structure into classical VAEs, the approach lacks the flexibility to dynamically adapt to the varying structural characteristics of individual input instances. Moreover, its primary objective is to closely approximate the true posterior distribution, whereas HTM places greater emphasis on learning an optimal underlying structuring of the embedding space itself. After learning this structure, generative models - such as Gaussian Mixture Models (GMMs) - can subsequently be employed to estimate the probability density function instead.

The field of research regarding conceptual VAEs observes that standard VAEs typically make use of a single bottleneck latent space distribution, from which all data points are sampled in an undifferentiated manner. The variables within this distribution are generally abstract and uninterpretable by humans, as they do not necessarily correspond directly to semantic concepts. Conceptual VAEs aim to address this limitation by enhancing the conceptual structures captured by these latent variables. In particular, [13] introduces a conceptual VAE in which the standard Gaussian prior is replaced by one conditioned on varying concept labels, each potentially corresponding to different semantic understandings (e.g., green, triangle, small). In this framework, concepts form separated interpretable regions, resulting in a more semantically organised latent space, where concept labels are reflected within their unique subspaces. Specifically, each conceptual domain (e.g., colour, shape, size) is assigned its own latent subspace, and a given concept label is modelled by a Gaussian within that subspace. This design encourages disentangled representations, and enables compositional reasoning and generation across different concepts - for example, naturally combining "green" and "triangle" to sample green triangles.

A precursor to this idea of disentangling the latent space to achieve more interpretable latent variables - and thus, a more structured embedding space - is the β-VAE [37]. This approach modifies the standard Evidence Lower Bound (ELBO) maximisation objective used in variational modelling by introducing a β hyperparameter, formulated as so:

$$\mathcal{L}_{\beta\text{-VAE}}(x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \beta \cdot \mathrm{KL}\left(q_\phi(z|x) \,\|\, p(z)\right)$$

As seen in classical settings, the first term encourages the encoder, $q_\phi(z|x)$, and decoder, $p_\theta(x|z)$, to reconstruct the original data, x, effectively from the learned latent variables, z, by maximising the expected log-likelihood. The second term is a KL divergence term which aims to minimise the distance between the learned posterior (the encoder) and the prior, $p(z)$. Note that the true posterior is typically intractable, and so the encoder is used to approximate it. This KL divergence term acts as a regularisation penalty to prevent the model from overfitting to the training data, as well as to ensure that the prior can be used for effective sampling and generation. By carefully tuning β to be greater than 1, greater emphasis is placed on the KL regularization term, which tightens the constraint on how much information can pass through the latent space. As a result, this forces the model to compress information more efficiently. When combined with an isotropic prior, this increased pressure encourages the model to disentangle the dominant factors of variation into statistically independent latent variables, indirectly promoting the alignment of certain dimensions with distinct, interpretable concepts (e.g. rotation, lighting, shape, etc.). Although these methods also seek to structure the latent space and impose an efficient information bottleneck through regularisation techniques, HTM achieves this via the recursive application of encoder–decoder modules, effectively leveraging a hierarchical design to do so optimally.

2.4 Context-free Grammars

Recent research on CFGs explores neural and probabilistic methods that learn or approximate parse tree-like structures from raw data - typically without relying on autoencoder architectures [44], [5], [6]. However, the Deep Inside-Outside Recursive Autoencoder (DIORA) introduced in [45] shares key conceptual similarities with HTM - most notably, the use of a recursively applied autoencoder to extract latent representations from the input data. DIORA operates in two key stages: the inside pass and the outside pass.

The inside pass applies the encoder module recursively to pairs of child-node representations from all of the possible binary parse trees. A learned composition function is used in tandem to compute the latent "inside" representations (vectors) of the corresponding parent nodes - conceptually similar to the merged representations found in HTM. As also seen in HTM, this recursive encoding process continues up the trees until root-node bottleneck representations of the complete inputs are obtained.

The outside pass follows after this inside pass, and initiates recursive decoding with a generic root bias vector. In this stage, the decoder is applied iteratively to generate "outside" vectors and reconstruct the original input tokens, whereby a reconstruction loss is used in end-to-end training. A key distinction from HTM lies in this stage. HTM feeds the encoder's learned bottleneck representations directly into the decoder, whereas DIORA uses generic bias vectors and conditions the decoding on the many "inside" vectors of the subtrees instead. More specifically, the "outside" vector of a node is computed as a function of its parent's "outside" vector and sibling's "inside" vector. DIORA uses a learned compatibility function during the outside pass as well, which is later used during inference parsing. Note that these functions are simple bilinear operations using a learned weights matrix, and are therefore less expressive than the deeper modules typically required by HTM. At test time, the CYK dynamic programming algorithm [7] parses the inputs by selecting trees based on these learned compatibility scores. However, unlike DIORA, exhaustively evaluating all possible binary merge trees for high-dimensional image data leads to intractable space and time complexity - given the theoretically large token populations encountered by HTM. To address this issue, HTM instead aims to learn optimal merging policies using policy gradient methods, allowing for the discovery of data structure without enumerating over all of the possible trees.

2.5 Policy Gradient Methods

In HTM, the task of learning optimal merging decisions involves a discrete action space, owing to the finite number of tokens available at each iteration. Due to this discreteness, the action selection process is non-differentiable, and cannot be trained directly with the (un)merging modules in an end-to-end manner. To address this issue, this paper employs the REINFORCE algorithm [46] to estimate gradients for these components. However, these gradient estimates are inherently noisy, due to the high variance introduced by Monte Carlo sampling. Regarding HTM, the stochastic and sparse nature of the reward signals (arising from the input data variability and the rarity of advantageous actions) both contribute significantly to the overall noise. Furthermore, consistent with the credit assignment problem [47], REINFORCE attributes the overall episodic (merge tree) reward to all of the actions taken. This dilutes the precision of the gradient signal and impairs the model's ability to accurately distinguish between beneficial and suboptimal actions. Although alternative gradient estimation methods exist [48], [49], they reduce variance at the cost of introducing bias in a less interpretable manner, potentially obscuring the validity of the experimental results. Consequently, this work reduces variance by using a simpler baselining technique instead, as well as by incorporating a curriculum learning schedule (section 3.6 and 3.7).

To summarize, the key distinctions that differentiate HTM from the previously discussed approaches are: the learned depth-wise recurrent (un)merging modules, their dynamic and recursive application to each input according to learnable decisions (actions), and their flexibility in handling multiple levels of granularity at each step of the merging process. Together, these elements also enable HTM to more effectively align its architectural structure with the inherent data structure of each individual input instance.

3 Methodology

3.1 Architecture

The HTM architecture consists of six neural networks, the: lifter, merger, unmerger, unlifter, classifier and policy networks.

3.1.1 Lifters

Lifting and unlifting modules facilitate transitions between the real token space and the latent embedding space in which HTM operates. Specifically, the lifter network is a simple linear transformation that expands real tokens from their original dimensionality (T) into that of the embedding space - both of which must be specified as hyperparameters during HTM initialisation. Conversely, the unlifter applies an inverse linear transformation, mapping embedded tokens back to their original token dimensionality. For the sake of simplicity, these networks do not incorporate any activation functions.

3.1.2 Mergers

The merger and unmerger networks are the depth-wise recurrent modules responsible for structuring the embedding space. The merger (encoder) is made up of three linear layers, with ReLU activations after the first two, ensuring the learning of expressive non-linear transformations. The first linear layer receives flattened vectors twice the dimensionality of the embedding space, and expands them into much larger vectors, as specified by the hidden dimension hyperparameter of these networks. The second layer performs the complex untangling/clustering transformation in the higher dimensional space, before the third layer then projects it back into one embedded token, representing the merger of the two. The unmerger (decoder) operates in a symmetric manner, except it receives as input one embedded token, and the final projection from the hidden space is to two embedded tokens, aiming to instead reconstruct the child-node representations from the merged parent node.

3.1.3 Classifier

The classifier network also adopts this architecture; however, after processing a single embedded token as the input, it outputs two logit values representing its confidence in the token's class membership. The first logit corresponds to whether or not the token is a leaf in the embedding space unmerging tree, whereas the second is for internal nodes - i.e., a merged token. Section 3.8 elaborates on its role regarding data generation.

3.1.4 Policy

Used in the learned policy setting - as opposed to the heuristic approach (section 3.2) - the policy network adopts the same three-layer linear architecture as the above modules. After processing two embedded token inputs, it produces a single logit value that reflects the module's confidence in the optimality of the proposed merge. However, if these unbounded logits become excessively large in magnitude, the resulting shifts in the logits adjacency matrix may bias the decision space towards invalid actions. Consequently, the network can begin to select impossible merges, such as merging a token with itself or with an inactive token that has already been merged. Earlier versions of this network included a hyperbolic tangent activation function - potentially scaled by an arbitrary constant to expand its output range - as a means to constrain logit values. However, this introduced complications when integrating the entropy loss (section 3.3.4). Specifically, the network would saturate the activation function to reach the maximum exploration entropy. This saturation led to vanishing gradients, significantly slowing the policy's ability to adapt and follow the entropy schedule. The issue persisted even with lower target exploration entropies (e.g., 0.7-1) and with alternative approaches, such as logit clipping. To mitigate these problems, L2 regularisation was applied to the policy network's parameters instead. Additionally, Glorot weights initialization [50] proved effective in reducing the likelihood of exploding logit values due to random initializations.

3.2 Forward Pass

3.2.1 Memory

HTM's forward pass begins by reinitialising the necessary memory for a given input, x. The input x is typically a tensor of shape $B \times P \times T$, where B denotes the (micro) batch size, P is the token population size, and T represents the dimensionality of each token prior to embedding. Since both B and P can vary across inputs, their values must be extracted at each pass. A logits adjacency matrix is then initialised across the batch dimension, storing logits that quantify the quality of a merge between any two tokens, in *both* possible permutations. Additional required memory includes a tensor to track the selected actions, and another to monitor which tokens remain available for merging at each iteration - both referencing tokens by their index in the merging tree. This tree is initialised as the input tensor x, and is appended with newly merged tokens as they are created. Other required memory includes containers for the one-step reconstruction losses (section 3.3.1), entropy values (section 3.4), classifier predictions, log-probabilities of actions and their associated rewards.

3.2.2 Merging

The forward pass begins following memory initialisation, at which point the input x is lifted and subjected to additive noise (section 3.5.1). Iterative merging then proceeds for the specified number of steps - typically continuing up to the root node. Within each iteration, the process starts by selecting an action from the current token population. This action is used to index into the merging tree, retrieving and concatenating the candidate embedded token pairs for merging via the merger module (in a permutation-dependent manner). The merger network outputs a newly merged token, which is also perturbed with noise before being concatenated to the merging tree. Additionally, the container that tracks active tokens is updated to reflect that this new token is now available for future merges. The next section details the action selection procedure that occurs within each merging iteration. Once the iterative merging concludes, the unmerging stage begins (section 3.2.4).

3.2.3 Selecting Actions

If this is the first merging iteration of the forward pass, the procedure populates the entire logits matrix using the input token population, x. Otherwise, it is more efficient to update only the necessary entries in the matrix based on the newly created token from the previous iteration. In both cases, both permutations of the token pairs can be computed in a single pass, without significantly impacting control over the (macro) batch dimension (at worst, by a factor of 2; section 3.7.1). Although the actions taken across the batch dimension may differ, this optimisation is reliable because the active token population sizes remain the same, allowing for more efficient vectorisation (section 4.1.1).

In terms of how the logits matrix is populated, the learned policy setting uses the outputs of the policy network. However, it is also possible to use the one-step reconstruction loss heuristic instead (section 3.3.1), thereby guiding HTM to selecting actions that yield better immediate reconstructions. Note that this heuristic loss must be multiplied by negative one, so that better reconstructions indeed correspond to higher values (i.e., less negative) when used in the softmax sampling.

After this, the logits matrix is used to compute the entropy of the decision space, resulting in a temperature-adjusted probability matrix (section 3.4.2). This matrix is then passed through a softmax function and sampled to select an action, based on the token indices corresponding to higher-valued logits. The resulting action tensor, of shape $B \times 2$, is returned for use in the iterative merging procedure described earlier. Before proceeding, however, several final steps are carried out. Specifically, the logits matrix is masked by assigning the mask value to the rows and columns corresponding to the varying actions taken across B. Together with the masking of the active tokens container, this ensures that tokens that are already merged into a new token are excluded from future sampling. Additionally, the one-step reconstructions are recomputed along with their associated one-step encoder-decoder gradients, and the log-probabilities of the selected actions are calculated and stored - along with their respective rewards.

3.2.4 Unmerging

The unmerging procedure begins by creating an empty container of the same dimensions as the merging tree, and copying over the nodes belonging to the active token population at the end of the merging stage (typically only the root). Iterative unmerging then reverses the process by operating on the most recently merged tokens in reverse order. The action container is used as a stack to determine where the resulting unmerged tokens should be reinserted into the unmerging tree. Before moving on to the next unmerging step, the merged tokens - unmerged during the current iteration - are passed to the classifier to generate class predictions for the internal nodes. Whilst it is theoretically possible to classify the unmerged token pairs at each step - based on their location within the unmerging tree - this approach may result in some overlooked leaf nodes - more specifically, the active ones transferred from the merging tree. To ensure full coverage, the algorithm explicitly iterates over the leaves after the iterative unmerging of internal nodes, thus separating the classification task. The final stages of the forward pass are to average the accumulated step losses and entropies across the iterations (episode), and, if applicable, computing the REINFORCE objective using the collected log-probabilities and rewards. The final return tuple includes the: unlifted unmerging-side leaves, full merging (and unmerging) trees, averaged step losses and entropies, classifier predictions (and labels), and the REINFORCE objective for policy optimisation.

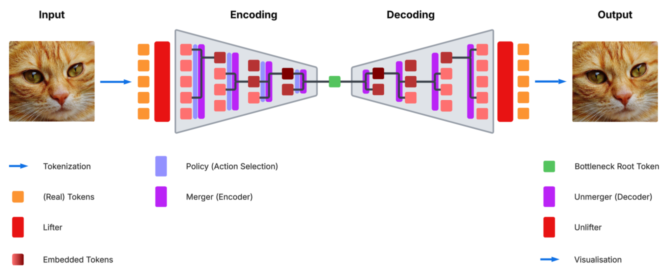


Fig. 1: Illustration of the HTM architecture and forward pass. The symmetry of the iterative merging (encoding) and unmerging (decoding) procedures is highlighted, with matching colours used to indicate the mirrored components. Additionally, the embedded tokens are shown becoming progressively darker as they are merged throughout the hierarchy, representing the compression of information. The classifier has been omitted for clarity, in order to focus on the end-to-end reconstruction objective on which HTM is trained.

3.3 Losses

Lifters and mergers are trained end-to-end using the loss function below, referred to as the HTM Loss.

$$\mathcal{L}_{\text{HTM}}(x) = f_b \cdot [\mathcal{L}_r(x) + \lambda_s \mathcal{L}_s(x) + \lambda_t \mathcal{L}_t(x) + \lambda_l \mathcal{R}_l(x) + \lambda_e \mathcal{R}_e(x)]$$

where:

$$\mathcal{L}_r(x) = \mathbb{E}[\,(x - \hat{x})^2\,]$$
$$\mathcal{L}_s(x) = \mathbb{E}[\,(\mathcal{S}[x_s] - \hat{x_s})^2\,]$$
$$\mathcal{L}_t(x) = \mathbb{E}[\,((1-\tau)\cdot x_d \;+\; (\tau - 1)\cdot \mathcal{S}[x_u])^2\,]$$

$$\mathcal{R}_l(x) = \mathbb{E}[\,x_{u[\,:l\,]}{}^2\,]$$
$$\mathcal{R}_e(x) = \mathbb{E}[\,x_{u[\,l:\,]}{}^2\,]$$

$$f_b = \left[min\left(1,\ \frac{P-1}{d_c}\right)\right]^{l_p}$$

The most important component of $\mathcal{L}_{\text{HTM}}$ is $\mathcal{L}_r$, the end-to-end reconstruction loss of the input image tokens, x. This is a simple mean squared error (MSE) loss between the ground truth x and the unlifted unmerged tokens, $\hat{x}$. The remaining two loss terms, $\mathcal{L}_s$ and $\mathcal{L}_t$, are the (one) step losses and transport losses respectively - discussed in subsequent sections. The last two terms are the regularisation penalty terms, R_l and R_e, for the lifter and the encoder. The subscript slicing represents the fact that they use the merging "up" tree leaves, $x_{u[\,:l\,]}$, and internal nodes, $x_{u[\,l:\,]}$, respectively (section 3.5.2). Each of the auxilliary loss terms have their own λ determining their weight in shaping the gradients from the loss function. Furthermore, the $\mathcal{S}$ function is used to denote the "stop-gradients" method (e.g., *detach()* in PyTorch), and the f_b biasing factor's role is explained in section 3.3.3.

3.3.1 One-Step Loss

The step losses are the expected immediate reconstruction losses of the token pairs selected for merging at each iteration. x_s denotes the concatenated pairs selected by the policy - which, in the heuristic setting, is indeed guided by these immediate reconstructions to begin with. $\hat{x_s}$ represents the predicted reconstructions obtained by applying the encoder and decoder to x_s once (in that order), thus preserving shallow one-step gradient computational graphs for auxiliary optimisation.

Alongside the transport losses, the auxiliary step losses were found to be crucial for promoting the expected degree of merging-unmerging symmetry within HTM. This challenge arguably posed the greatest difficulty to overcome in this work. Even in early, simplified experiments - where a heuristic policy was employed in an ϵ-greedy manner on a small dataset (e.g., 3 images with 6 tokens each) - the basic end-to-end reconstruction objective ($\mathcal{L}_r$) alone was insufficient to guide the network towards optimal policies and reconstructions beyond a single merge. It was hypothesised that, at these early curricula, the decoder and unlifter possessed enough expressivity to somewhat reconstruct the original tokens from a largely random policy. Prior approaches, such as training the lifters and mergers separately using distinct optimisers, or initially guiding the model with a hardcoded optimal policy before phasing it out, were explored before the introduction of step-losses. However, neither approach resolved the issue as effectively as these auxiliary loss components.

3.3.2 Transport Loss

The transport losses also serve as an auxiliary embedding-space reconstruction objective, operating between the full merging "up-trees" (x_u) and the unmerging "down-trees" (x_d). This term encourages symmetry between the up-trees and down-trees, whereas the step losses ensure that the encoder–decoder modules can accurately reconstruct merged tokens immediately. However, this objective introduces value-level teacher forcing onto the decoding components of HTM, compelling them to follow a reconstruction path that closely mirrors the encoder-side up-tree. Whilst this form of imitation learning is effective when the decoder consistently receives the expected tokens for unmerging, it prevents the decoder from learning how to recover from slight deviations - something that is likely to occur during the stochastic sampling involved in the generative setting. This is where the τ blending parameter comes in. By setting τ to values greater than zero, the decoder is permitted to generate more diverse/random tokens for the same loss, as they are linearly interpolated to resemble the encoder-side tokens regardless. Naturally, τ can be decayed gradually to zero over the course of training to progressively reduce reliance on these encoder-side tokens, thereby allowing the decoder to operate independently - reflecting the conditions seen in the generation pipeline, where the encoder outputs are not available (section 3.8). Although these auxiliary loss components significantly improved HTM's performance, they may introduce a risk of mode collapse (section 5.4).

3.3.3 Bias Factor

The intuition behind the biasing factor f_b is that, as curriculum learning progresses (section 3.7), it may be beneficial to focus the learning - and consequently the gradients from the HTM loss function - on the more complex and relevant token population sizes, P, associated with the newer curricula. The denominator term d_c initially starts at 1 (resulting in no bias), and is gradually increased to the number of merges (i.e., the complexity) characterising each curriculum. As a result, the HTM loss computed for a given input x of size P is weighted by the fraction of the maximum number of merges possible with that P (i.e., $P-1$), over the current curriculum's merges (d_c). Eventually, d_c reaches the maximum number of merges, which can be determined after the first epoch from the distribution of P. Additionally, the magnitude of f_b is controlled by the training hyperparameter l_p (the loss power), where values near 0 reduce the effect of the bias, whereas larger values amplify it. Whilst it is generally advantageous for the model to focus on mastering the more challenging, newer tasks (assuming satisfactory performance on prior curricula), it is uncertain whether this bias adversely affects performance on smaller P that require fewer merges than the current curriculum. Consequently, in practice, a small value of $l_p = 0.01$ was used.

3.3.4 Other Losses

Another loss component is that of the policy network:

$$\mathcal{L}_{\mathrm{p}}(x, \pi_\theta) = \mathop{\mathbb{E}}_{T\sim\pi_\theta}\left[-\sum_{t=0}^{T}\log \pi_\theta(a_t \mid x_{u_t}) \cdot R(a_t)\right] + \lambda_{entrp} \cdot \mathbb{E}\left[\,|e - \hat{e}|\,\right]$$

where: π_θ is the learnable policy network, T are the merging trajectories, $R(a_t)$ are the advantages (return) for the actions at iteration t, and $\hat{e}$ and e are the average measured and target entropy respectively. The first term corresponds to the expected sum from the REINFORCE objective, which is computed during the forward pass using the actions taken and their associated rewards. Note that this term is made negative because PyTorch's autograd minimizes objectives, whereas REINFORCE requires maximization. Whilst this term is absent in the heuristic setting, the second term - related to entropy - is present in both cases; the main distinction lies in the parameters that are updated during training (section 3.4.3). Naturally, λ_{entrp} is also not used in the heuristic setting. Interestingly, it was observed that the L1 Loss served as a better criterion for the entropy optimisation. This was speculated to result from the narrow range $[0, 1]$ within which the normalized entropies lie, and the fact that the MSE loss reduces gradients more significantly at these smaller ranges compared to the L1 loss. Finally, the classifier uses a cross entropy loss function, and is computed from the predicted logits and generated labels returned by the forward pass.

3.4 Entropy

3.4.1 Motivation

Using a softmax function for sampling from the logits matrix, rather than employing an ϵ-greedy strategy, enables more stable and focused exploration within HTM's action space. This is because the stochasticity of ϵ-greedy sampling can repeatedly favour suboptimal actions associated with low rewards, thereby dispersing the exploration excessively. Before applying the softmax function to convert the logits matrix into a probability distribution (matrix), it is essential to divide the logits by a temperature parameter, which governs the exploration–exploitation trade-off. Whilst curriculum learning may begin with an arbitrarily high temperature - decayed linearly or exponentially over time - such schedules risk prematurely transitioning the network from exploration to exploitation, thereby limiting its capacity to sufficiently explore the action space and discover an optimal merging policy. Moreover, identifying an appropriate temperature throughout training is challenging, as it is inherently dependent on the continuously adjusting distribution of logits. A more generalisable alternative is to instead monitor the normalised entropy of the resulting probability matrix, and adjust the temperature dynamically to maintain this at a desired level. This allows exploration–exploitation schedules to be defined in terms of target entropy rather than temperature. Although other adaptive algorithms exist to modulate entropy directly [51], [52], HTM instead dynamically adjusts the temperature parameter via gradient descent alongside its other modules, with the objective of keeping the observed entropy aligned with the scheduled target.

3.4.2 *Implementation*

Entropy computation begins by first creating a copy of the logits matrix, a step found necessary due to the in-place nature of the operations used for updating and masking it. Although it may be possible to avoid this memory overhead, doing so would likely introduce additional computational complexity in tracking inactive and newly created tokens (potentially requiring more sophisticated data structures), which could in turn hinder HTM's performance. After applying softmax to the temperature-adjusted clone, a small ϵ (e.g., 1×10^{-20}) is added - particularly for the dimensions with zero values - enabling the gradient descent optimisation. This adjustment is critical, as the gradient of the function $x \log x$ (used in calculating entropy) is undefined at zero, resulting in NaNs that would propagate through the gradient computation graph and disrupt optimisation. In this context, initialising mask values with negative infinity also proved impractical; in practice, these are instead assigned a large negative constant (e.g., -9×10^{20}) to avoid selection. Following the re-normalisation of the ϵ-adjusted probability matrix, three quantities are computed: (1) the probability mass assigned to unavailable actions, which had zero probability under the softmax, and can be expressed as:

$$\frac{\epsilon}{1 + p_s \cdot \epsilon}$$

where p_s is the number of entries in the probability matrix; (2) the number of valid actions, calculated using the combinations formula based on the active token population size; and (3), the number of invalid actions (i.e., those originally masked), equal to p_s minus the second quantity (2). The valid action count (2) is used to normalise the entropy by dividing by its logarithm. Interestingly, although this normalisation of the entropy does not alter the relative ordering of preferred actions based on their probabilities, it does attenuate the determinism associated with the absolute differences between them. This may lead to under-representing the most favourable actions' values in the measured entropy, potentially blunting the exploitation response to highly optimal actions. Nonetheless, to counteract the bias introduced by the ϵ adjustment and ensure more accurate entropy measurements, terms (1) and (3) are used in subtraction within the summation component of the entropy formula. It is also important to note that the temperature parameter utilised during optimisation is treated as an exponent, with the division being performed using the output of the exponential function applied onto it. This ensures that the resulting temperature remains within an appropriate range - specifically, the interval $(0, \infty]$ - across the entire real domain.

3.4.3 *Tuning*

The target entropy schedule typically begins at one for a sustained period, permitting the random selection of all possible actions, and providing the HTM agent with some initial experience at the task. It then gradually decays linearly to the working entropy, which is typically maintained for the majority of the curriculum learning. Eventually, the target linearly decays to zero during exploitation, and is held at zero for a short period to ensure full optimisation across all modules for the determined optimal policy. These critical scheduling points are user-defined training hyperparameters and will depend heavily on the curriculum setup.

Whilst the temperature parameter is indeed employed and optimised by its own dedicated optimiser in the heuristic policy setting, this is not the case for the learned policy setting. This version utilises the policy network's gradients in the logits matrix to directly adjust them towards the target entropy. It should be noted that the temperature parameter could be used in this setting; however, employing two separate optimisers - one for the temperature and one for the policy - would lead to conflicting updates and result in unstable behaviour. Due to the REINFORCE objective in the learned policy loss function, entropy optimisation is hindered and typically does not adhere as closely to the target schedule (Fig. 2). The extent of this discrepancy can be mitigated by increasing λ_{entrp}, although large values are generally unnecessary, as the behaviour is typically only strongly exploitative when a clearly optimal merging policy exists and is likely more worthwhile to follow. Additionally, it is important not to set λ_{entrp} too high (empirical evidence suggests ideally an upper bound of 10), as otherwise the gradients from the REINFORCE term will lack sufficient bias, thereby prolonging convergence to an optimal policy.

In the heuristic setting, issues were encountered with temperature optimisation, where the optimiser (Adam [53]) would remain stuck at an entropy of one after increasing the temperature sufficiently to induce maximal exploration. Initially, it was hypothesised that adjusting the β parameters - responsible for the exponential smoothing of the first and second moments - could result in a more reactive optimiser, ideally reducing this unsustainable propagation delay; however, this approach proved less effective than performing a full reset on the optimiser's memory. This is likely because Adam is not well-suited to handling prolonged periods of constant gradients (zero, in this case) followed by a sudden shift in the optimisation objective - which typically remains semantically consistent. To further reduce propagation delays, a higher learning rate (0.05) was found to be optimal,

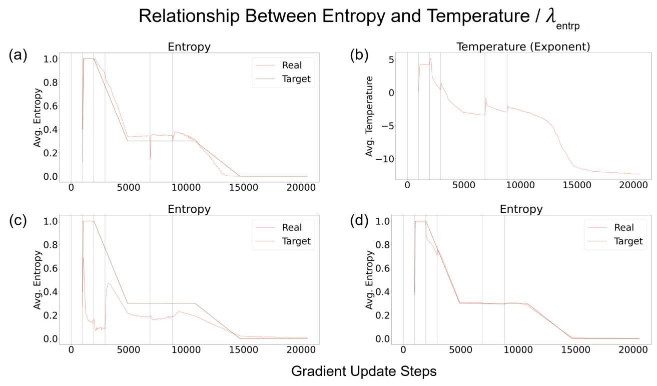


Fig. 2: Example illustrating the alignment between the measured (real) entropy and the scheduled (target) entropy throughout the curriculum learning. Sub-figures (a, b) show the entropy fluctuations under the heuristic policy setting, along with the corresponding variations in the temperature parameter, respectively. Sub-figures (c, d) show the entropy fluctuations under the learned policy setting, with λ_{entrp} set to 1 and 10, respectively. Grey vertical lines indicate curriculum transitions (section 3.7), and results are plotted every 50 gradient update steps.

despite theoretically compromising the optimiser's ability to track the target entropy precisely. Whilst smaller learning rates may, in principle, result in more accurate tracking of the target entropy, this was not observed in practice, likely because HTM's logit distributions typically fluctuate too rapidly for the comparatively slower response of the optimiser. Additionally, similar to the scenario involving a maximally explorative policy, during maximal exploitation, the temperature parameter continued to decrease to exponentiated values approaching zero, eventually reaching infinitesimal magnitudes that destabilised the implementation. To mitigate this, once the measured entropy fell below a defined threshold (e.g., ≤ 0.0005), optimiser updates were suspended until the entropy values rose above the threshold.

3.5 Noise & Regularisation

3.5.1 Noise

HTM adds a small amount of non-trainable Gaussian noise to the up-trees, x_u, after lifting and merging, which can be controlled by a standard deviation hyperparameter (typically set to 0.01). Since the lifters and encoders are deterministic operations, once the entropy is reduced near zero, they possess the capacity to compress an unbounded amount of information into a finite region of the embedding space. This instability also affects the generative decoding process, as it reduces robustness to stochastic variation from sampling outside of that region. Injecting noise into embedded tokens upon creation constrains the amount of information that can be encoded into a given volume, thereby encouraging more structured clustering in the embedding space. Furthermore, it provides a smoothing effect during root space sampling (section 3.8), which facilitates better fitting of the continuous GMMs and enables the decoder to effectively denoise slightly perturbed tokens - thereby enhancing the consistency of HTM's generations. However, it is crucial to ensure that the noise magnitude remains small relative to the input signal, as excessive noise may obscure the underlying information, and impinge on HTM's ability to reconstruct x.

3.5.2 Regularisation

When analysing the embedding space as an information channel, its volume remains unbounded unless regularisation penalties are applied. There are generally two types of approaches to address this. One involves harder constraints, such as normalising all embedded token vectors onto a unit hypersphere (section 2.2). On the other hand, an example of the softer approaches - adopted by this work - involves applying an L2 regularisation term, similar to that used in Ridge regression, but instead applied directly to the embedded tokens rather than to the network's parameters. As such, the HTM loss function includes these mean-reduced L2 penalty terms: one applied to the lifted tokens from x_u for the lifter, and the other to the merged tokens from x_u, predominantly penalising the encoder. Each penalty is governed by its own λ weighting hyperparameter. Regularising in this manner can help organise the embedding space more effectively - potentially encouraging the grouping of similar concepts, such as related root spaces, into compact regions - resulting in a more tractable distribution for sampling. It is important to note that performing this without any added noise to the embedded tokens would exacerbate the previously mentioned instability. Moreover, careful tuning of these λ_l, λ_e hyperparameters is essential. Excessive compression of the embedding space can make the reconstruction task practically infeasible, as some volume is required to accommodate the injected noise in each signal.

3.6 Reinforcement Learning

In this work, structural action-level teacher forcing is applied to the decoding-side modules - the generative components - such that the up and down-trees are isomorphic. Whilst this simplifies the problem by removing the need for two learnable policies - one for the encoder and one for the decoder - and instead enforcing that they operate together in the expected symmetric manner, it leads to long-term dependency issues in the rewards. More specifically, whilst the transitions themselves are Markovian, the rewards exhibit long-term non-Markovian behaviour.

The reward of an action can be defined as the (negative) reconstruction loss between the encoder's original input tokens, and the decoder's output tokens when that merged token is later unmerged. With the teacher forcing, the actions taken by the encoder are replicated in reverse order by the decoder; thus, the first encoding action is the final decoding action. Consequently, it is influenced by the entire future merging and unmerging hierarchy, with each stage potentially altering the input received by the decoder for that action, and thereby impacting the quality of the reconstruction - i.e., the reward. This observation persists throughout the hierarchy, though it is more pronounced for earlier merges.

Although curriculum learning helps in alleviating this issue (section 3.7), learning effective policies remains challenging in such a noisy environment. To address this, the immediate one-step reconstruction losses (section 3.3.1) can be used as rewards across the merging hierarchy instead. This heuristic is supported by the observation that prior merging actions typically do not influence the reward of the current action as much as the future actions do, especially since they occur afterwards in the decoding process - assuming the structural teacher forcing is applied. Furthermore, this hierarchy of step losses can serve as a reasonable approximation of the cumulative reward (return) for each action, based on the intuition that poor immediate reconstructions are strongly correlated with inferior rewards in that action's unmerging subtree - and vice versa. For example, if the subsequent unmerging actions do not belong to the current action's unmerging subtree, they remain unaffected by this outcome. However, if they do, they are likely to be reconstructed with reduced accuracy, as the decoder received a merged parent node that deviates greatly from what it was expecting in order to unmerge effectively.

The hierarchical application of this heuristic can eventually represent the returns for all actions reliably, allowing the (negative) step-losses to be used as the advantages for the REINFORCE objective. Additionally, to further reduce the variance-derived noise in the REINFORCE gradients, HTM uses baselining by subtracting normalised running advantages. These running advantages are stored in a large buffer and updated after each episode - i.e., each application of HTM to an input instance. Note that a standard deviation ϵ is required during normalisation to prevent division-by-zero errors.

3.7 Curriculum Learning

The approach to training HTM draws inspiration from the curriculum learning paradigm in RL, whereby training is structured into a sequence of curricula that begin with simpler tasks and gradually increase in complexity; thus, guiding the model progressively through a challenging environment. In HTM, task complexity is characterised by the number of merges imposed onto the input token populations. The curricula are therefore typically assigned an ascending number of merges, along with a specified number of training epochs each. For example, starting with zero merges focuses training solely on the lifter networks, whereas progressing to a single merge engages the full pipeline - albeit with comparatively shallow gradient computational graphs relative to the later curricula involving deeper merging hierarchies. Furthermore, tuning the learning rates and incorporating learning rate decay within each curriculum - treating them as distinct tasks - was found to enhance the overall training process. This remained effective even though the embedding space may theoretically need substantial restructuring as tasks become less trivial. Learning rates are also bounded by a minimum threshold, beyond which training typically becomes ineffective. Additionally, the λs used for the auxiliary loss terms can be independently adjusted for each curriculum, allowing for dynamic adaptation to task complexity. It is also important to note that scheduling variables that evolve across the curricula (e.g., learning rates, target entropies, τ, etc.) are updated at each gradient step rather than at the epoch level. This finer-grained update strategy ensured smoother transitions between training environments, and was particularly beneficial for managing the entropy and τ schedules in practice - especially in settings with fewer training epochs.

3.7.1 Macro & Micro Batch Dimensions

Furthermore, during curriculum learning, the micro-batch dimensions, B, passed into the HTM architecture are captured to construct a micro-batch distribution. This provides insight into the effective batch sizes received by the GPU for parallelisation, and is strongly influenced by the distribution of token population sizes, P - which can be fully determined after the first epoch. Fig. 3 demonstrates this relationship. Since different images can yield varying P, special consideration must be given when storing the data as tensors, as discussed in section 4.1.1.

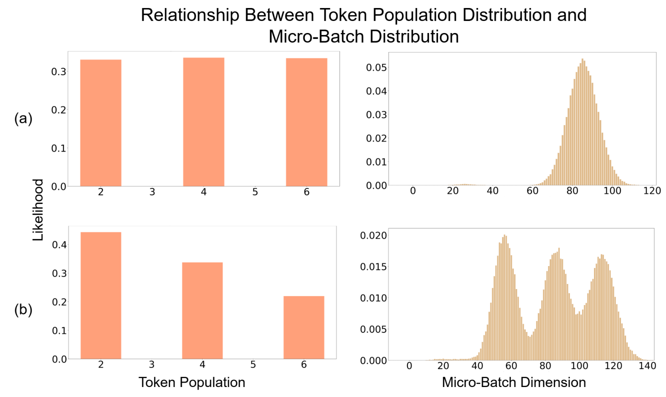


Fig. 3: Demonstration of the relationship between P and B distributions across two different training sets. Row (a) follows a uniform P distribution, resulting in a unimodal, Gaussian-like B distribution. Row (b) illustrates how slight differences in the frequencies of some P can cause skewed/multimodal B distributions, reflecting the imbalances in batch compositions for GPU parallelisation.

Although the macro-batch dimension specified during HTM training influences B in a directly proportional manner, the resulting B sizes can often be significantly smaller than intended. This discrepancy justifies the average increase in batch dimensions observed during the forward pass, making it more comparable to the user-defined macro-batch training hyperparameter. During HTM training, when iterating over macro-batch-sized input data - grouped by the varying number of P - an additional gradient normalisation divisor is applied, together with the standard gradient accumulation divisor. This divisor is the number of distinct P present in the macro-batch. Dividing the loss terms by this number is generally well-behaved; however, in cases of significant B variance, it can introduce a strong bias towards the gradients from instances of P with smaller B, potentially reducing training efficiency. One may opt to favour the less frequently occurring values of P to promote greater robustness of HTM across the full range of P, rather than risk degraded performance on these rarer cases by weighting according to their B proportion within the macro-batch. Nonetheless, for the uniformly synthesized datasets used in this study, where B distributions are largely unimodal, the difference is theoretically expected to be negligible.

3.8 Generation

3.8.1 Gaussian Mixture Model

Once HTM training concludes, the structured embedding space is sampled for root tokens across various random input instances (the number of which can be controlled by the user). A GMM is then fit to this root data using expectation maximisation, completing the generative components of HTM. By sampling the latent GMM and applying the decoding-side modules accordingly, HTM is able to synthesis data that theoretically resides on the original input data manifold. Additionally, the GMM's training parameters may be tuned independently from the learning of HTM's embedding space - guided by evaluation metrics such as BIC or AIC - which is advantageous for determining the optimal number of Gaussian components to represent the latent root structures.

3.8.2 Sampling

After confirming that a GMM has been trained for the HTM object, the sampling procedure begins by initiating the generation down-tree with a root sample drawn from this GMM. Additionally, a list is initialised to store the indices at which leaf tokens are identified during generation, facilitating the extraction of the final tokens to be unlifted. The main generation loop then proceeds, terminating once all tokens in the growing down-tree have been processed, or upon reaching a maximum number of generation steps (arbitrarily set to avoid indefinite looping). Starting with the root, each iteration begins by extracting the current token to be analysed from the down-tree. The classifier is then applied to it to obtain logit predictions, which indicate whether the token is likely to be a leaf or an internal (merged) node. A softmax function and decision threshold (typically 0.5) are applied to yield the final classification output. If the token is classified as a leaf, its index in the down-tree is appended to the leaf index list. If, however, it is identified as a merged node, the decoder is applied to generate two new tokens, which are then appended to the down-tree for future analysis. In this manner, the down-tree effectively operates as a queue data structure, with the sampling iterations performing a level-order recursive traversal of this tree. Once the termination condition is met, the leaf tokens are retrieved (using the index tracking list), unlifted, and returned in tokenised form - rendered later via the dataset-dependent visualisation procedure (section 4.1.2). Note that this sampling approach presents challenges for vectorisation, primarily due to the variable number of output tokens across generated instances.

3.8.3 Classifier

As previously noted, the classifier is employed during sampling to determine whether a generated token should continue to be unmerged. Owing to the curriculum learning setup, the classifier is initially exposed to leaf tokens at a disproportionately high frequency; however, the classification task becomes more balanced in the later curricula, as the number of internal nodes ($\approx P - 1$) approaches that of the leaves (P) - resulting in an approximately even split. Whilst it is possible to delay classifier training until the final curriculum (i.e., when merging to the root), it is likely advantageous to expose the classifier to a broader range of merged tokens earlier in the training. This is particularly beneficial when applying the τ schedule, which introduces diversity within the merged token subspaces before HTM converges.

4 Experiments & Results

The experiments conducted on HTM - including, but not limited to, those presented in this section - were carried out on training sets comprising $50{,}000$ instances, and evaluated on testing sets of $20{,}000$ instances. For clarity in policy analysis (section 4.2), token ordering was kept consistent within each dataset.

4.1 Dataset Implementation

4.1.1 Collate Function

During experimentation with HTM, dataset objects are expected to implement an overloaded collate function, utilised by the training data-loader to assemble batches in the format required for HTM training. Given the varying P across different input images, the collate function returns a dictionary, wherein the keys correspond to instances of P, and the associated values are tensors of shape $B \times P \times T$. Recall: B denotes the micro-batch size, P the token population size, and T the dimensionality of the input tokens prior to embedding. Note that the sum of all B across the dictionary's entries is equal to the macro-batch size specified by the user as a training hyperparameter. Additionally, this collate function is also responsible for ensuring the tokenization of the input images.

Although padding-based approaches could be employed to handle this variability in P, they introduce additional complexity during HTM's merging, classification, and sampling stages (e.g., tracking padding tokens across the varying active P of B at each iteration), and may not yield any substantial computational efficiency gains. Nonetheless, if the dataset exhibits high variance in P, the resulting B distribution shifts closer to one - even with a large macro-batch size or training set size - thereby diminishing the effectiveness of the dictionary batching strategy. Ideally, data should be pre-arranged in this manner and stored locally according to P, allowing the training pipeline to load one group of P at a time; however, reading in separated data in this fashion can also pose significant efficiency challenges, and may still result in too small a B for effective GPU parallelisation.

4.1.2 Visualisation Function

Each dataset used in HTM experimentation should also provide a visualisation function, which reverses the tokenization process by mapping from the token domain back to the image domain. This function must be invariant to both the size and ordering of the token populations, as the generative procedure may produce an unexpected number of tokens, likely in a different permutation than those resulting from tokenization. Addressing this with exact correspondence would require a more sophisticated generation pipeline - for example, where the decoder or classifier are capable of learning transition dynamics. This design aspect highlights the importance of encoding relative positional information directly within the tokens themselves (e.g., the x and y coordinates of pixels in an image).

4.2 Policy Diagrams

A HTM model's learned policy on a given dataset can be analysed by repeatedly executing the forward pass, and extracting the actions taken during the hierarchical merging process. The policy diagrams (Figs. 6, 10 and 12) illustrate these trajectories of merging actions - ordered from top to bottom - where each action is represented as a pair of indices corresponding to unique tokens within the population. The red numbers denote the unique index assigned to each newly created token from the associated merging action in the trajectory, thereby facilitating interpretability. For instance, in Fig. 6 (a), the initial merge between tokens 1 and 0 (in that order) produces a new token indexed as 6, which is later merged with token 7 to yield token 8, and so on. In general, visualising or computing scoring functions for such policies is challenging due to the complexity of hierarchical merging, particularly when accounting for the ordering dependencies within and between actions in "correct" trajectories.

4.3 Crosses Dataset & Results

The synthesised crosses dataset is composed of cross-shaped structures formed by two line segments intersecting at their midpoints. Each token in this dataset is a four-dimensional representation of a line segment, expressed as (x_1, y_1, x_2, y_2) using its two endpoints. Fig. 4 (a) shows potential parametrisations of this dataset, whereas Fig. 4 (b, c) presents examples from the training set used, where line segments are of fixed length and cross angles vary within the range $[20°, 160°]$. Furthermore, the training set is uniformly distributed across P values of two, four, and six, corresponding to images containing between one and three crosses.

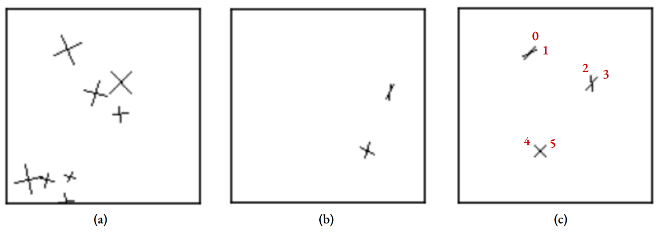


Fig. 4: (a) Other possible parametrisations of the crosses dataset (e.g., fixed $90°$ cross angles, varying line lengths). (b) Example from the training set used, $P = 4$ (two crosses). (c) Example from the training set used, $P = 6$ (three crosses). The token indices are labelled, highlighting the adjacency pattern of line segments which form a cross.

Fig. 5 illustrates the reconstruction losses of HTM during curriculum learning on the crosses training set - distinguishing between $\mathcal{L}_r$, $\mathcal{L}_s$ and $\mathcal{L}_t$ (section 3.3). Training was conducted over a total of 210 epochs, using REINFORCE with 18 embedding dimensions and a working entropy of 0.3. The λ coefficients for the entropy, lifter, encoder, step, and transport loss components were set to 3, 0.01, 0.001, 0.2, and 0.2, respectively. Furthermore, learning rates were appropriately decayed throughout training to promote faster, more stable convergence, and the τ parameter was linearly annealed from 0.5 to 0 over the initial four-fifths of training. Fig. 6 demonstrates the resulting learned policy through a policy diagram, constructed by sampling to-root trajectories on example test instances. Following the core HTM training, the GMM can be tuned (Fig. 7) to determine the optimal number of components required to model the structure of the learned embedding space. Sampling from this latent distribution enables the generation of synthetic data, examples of which can be seen in Fig. 8.

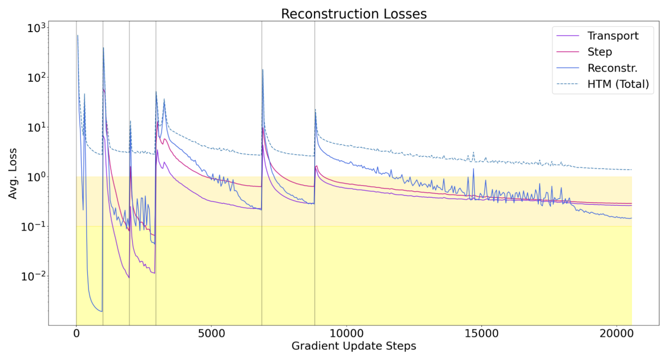


Fig. 5: HTM training losses on the crosses dataset. Grey vertical lines separate the curricula - corresponding to zero through to five merges - and average losses are plotted every 50 gradient update steps. The highlighted yellow regions depict perceptually satisfactory reconstruction losses.

```
6  ¦[[[1, 0], ¦[[[2, 3], ¦[[[2, 3], ¦        4 ¦[[[1, 0], ¦[[[2, 3],
7  ¦ [4, 5],  ¦ [4, 5],  ¦ [1, 0],  ¦        5 ¦ [3, 2],  ¦ [0, 1],
8  ¦ [7, 6],  ¦ [7, 6],  ¦ [6, 7],  ¦        6 ¦ [5, 4]]] ¦ [4, 5]]]
9  ¦ [2, 3],  ¦ [0, 1],  ¦ [5, 4],  ¦          ¦          ¦
10 ¦ [8, 9]]] ¦ [8, 9]]] ¦ [8, 9]]] ¦          ¦          ¦
        (a)        (b)        (c)                 (d)        (e)
```

Fig. 6: Policy diagram using example instances from the crosses testing set. (a, b, c) correspond to images containing three crosses, and (d, e) to ones containing two. See Fig. 4 (c) to help interpret the up-tree leaf indexing.

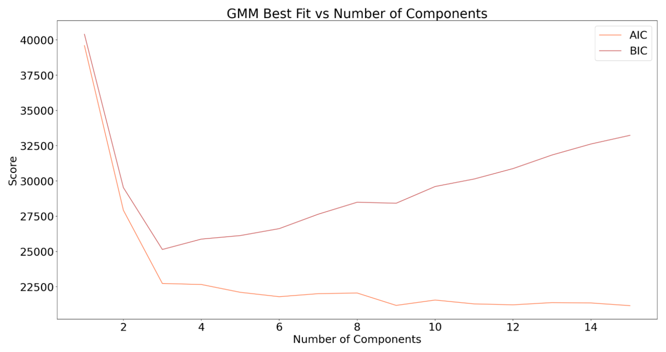


Fig. 7: Hyperparameter tuning the number of Gaussian components in the crosses GMM, evaluated using both the Bayesian Information Criterion (BIC) and Akaike Information Criterion (AIC).

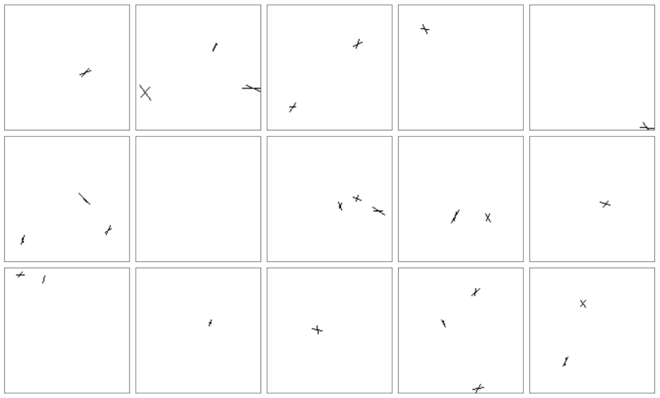


Fig. 8: Fifteen samples generated by the final HTM model trained on the crosses dataset (not cherry-picked).

4.4 Trees Dataset & Results

This dataset consists of images depicting complete binary trees, represented in tokenized form as a set of coordinates, (x_1, y_1), each corresponding to the position of a node in the tree. Several degrees of freedom are intentionally constrained: the separation angles, $L\hat{P}R$ - where L and R are the left and right child nodes of the parent node P - are fixed across the hierarchy. Additionally, the scale factor, s_f, reducing the distance between parent and child nodes at each tree level remains constant. This design ensures that child nodes appear progressively closer to their parents, allowing the myopic pairwise policy procedure to infer the token relationships (section 5.2). With this in mind, each image contains only one tree, with a fixed tree height of three ($P = 7$). Whilst it is possible to use a fixed angular deviation to rotate the child subtrees at each level, and thereby relax other constraints, the final training set avoids this and instead enforces s_f to be less than 0.5. However, not only does this design necessitate that $L\hat{P}R$ must be less than $60°$, but the cosine rule can be used to show that s_f must satisfy:

$$s_f < \sqrt{2 \cdot (1 - \cos(L\hat{P}R))}$$

Fig. 9 (c) illustrates examples from the final trees training set used, although wider tree structures - also incorporating angular subtree deviations - were initially explored (b). In particular, the first set of experiments aimed to assess whether HTM could reliably merge nearest neighbours. To this end, a simple model with a two-dimensional embedding space was trained for only one merge on datasets (b) and (c). These results - shown in Figs. 10 and 15 (b, c) - motivated the subsequent experiments on narrower trees, where HTM was trained to the root bottleneck (Fig. 11) using an embedding dimensionality of 7. This final model underwent training for 861 epochs, also in the learnable policy environment. On the other hand, smaller λ values were used for the lifter and encoder loss terms - 0.0002 and 0.00002, respectively - and an extended exploitation phase was employed in the entropy schedule. The other hyperparameters remained consistent with those used in the crosses experiments. This final iteration's learned policy is exemplified in Fig. 12, and the GMM tuning with example image generations can be seen in Figs. 13 and 14, respectively.

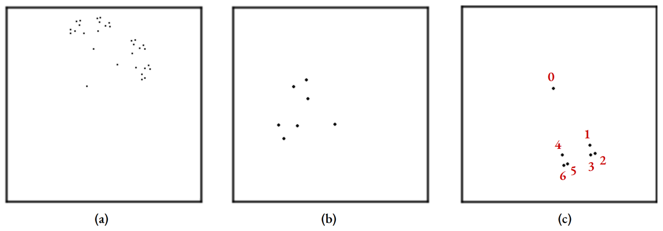
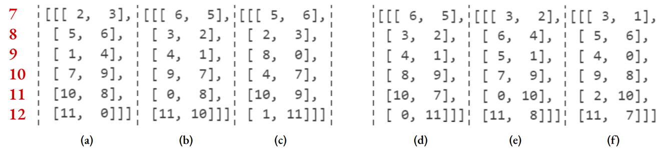


Fig. 9: Example parametrisations of the trees dataset. (a) Wide tree structures, similar to (b), but with a tree height of 5. (b) Wide tree structures with parameters: $L\hat{P}R = 45°$, $s_f = 0.5$, and $20°$ deviation angles. (c) Narrow tree structures with parameters: $L\hat{P}R = 25°$, $s_f = 0.15$, and no deviation angles (set to $-\frac{1}{2} \cdot L\hat{P}R$ due to the algorithm's recursive design). The token indices highlight the nested structure of the tokenized image: $[P] + [L_{tree}] + [R_{tree}]$, utilising the parent node and child subtrees recursively.



```
7   [[[ 2,  3], [[[ 6,  5], [[[ 5,  6],    [[[ 6,  5], [[[ 3,  2], [[[ 3,  1],
8     [ 5,  6],   [ 3,  2],   [ 2,  3],      [ 3,  2],   [ 6,  4],   [ 5,  6],
9     [ 1,  4],   [ 4,  1],   [ 8,  0],      [ 4,  1],   [ 5,  1],   [ 4,  0],
10    [ 7,  9],   [ 9,  7],   [ 4,  7],      [ 8,  9],   [ 7,  9],   [ 9,  8],
11    [10,  8],   [ 0,  8],   [10,  9],      [10,  7],   [ 0, 10],   [ 2, 10],
12    [11,  0]]]  [11, 10]]]  [ 1, 11]]]     [ 0, 11]]]  [11,  8]]]  [11,  7]]]
         (a)         (b)         (c)            (d)         (e)         (f)
```

Fig. 10: Policy diagram of the initial HTM models trained for a single merge with two embedding dimensions. The policies of two models are exemplified: one trained on the narrow dataset (a, b, c) and the other on the wide dataset (d, e, f). See Fig. 9 (c) to help interpret the up-tree leaf indexing.

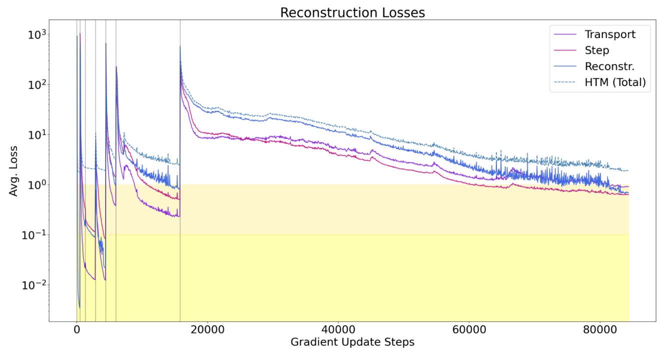
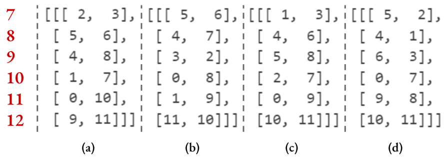


Fig. 11: HTM training losses on the narrow trees dataset. As in Fig. 5, the grey vertical lines separate the curricula, which in this case range from zero through to six merges (i.e., the final model trained to-root).



```
7   [[[ 2,  3], [[[ 5,  6], [[[ 1,  3], [[[ 5,  2],
8     [ 5,  6],   [ 4,  7],   [ 4,  6],   [ 4,  1],
9     [ 4,  8],   [ 3,  2],   [ 5,  8],   [ 6,  3],
10    [ 1,  7],   [ 0,  8],   [ 2,  7],   [ 0,  7],
11    [ 0, 10],   [ 1,  9],   [ 0,  9],   [ 9,  8],
12    [ 9, 11]]]  [11, 10]]]  [10, 11]]]  [10, 11]]]
         (a)         (b)         (c)         (d)
```

Fig. 12: Policy diagram of the final HTM model trained to-root on the narrow trees training set.

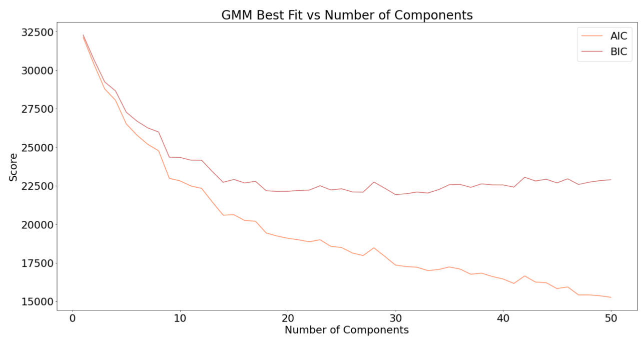


Fig. 13: Hyperparameter tuning the number of Gaussian components in the final trees GMM (similar to Fig. 7).

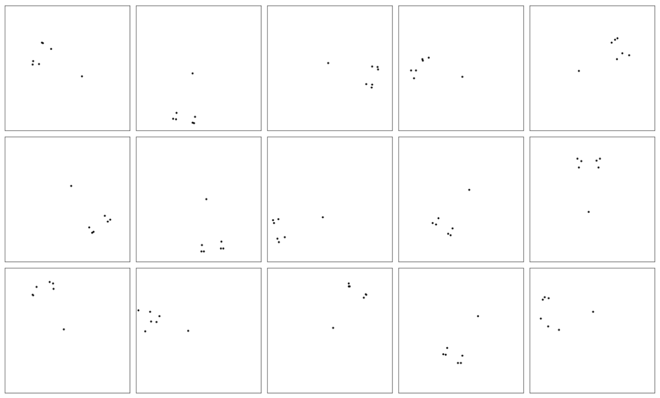


Fig. 14: Fifteen samples generated by the final HTM model trained on the narrow trees dataset (not cherry-picked).

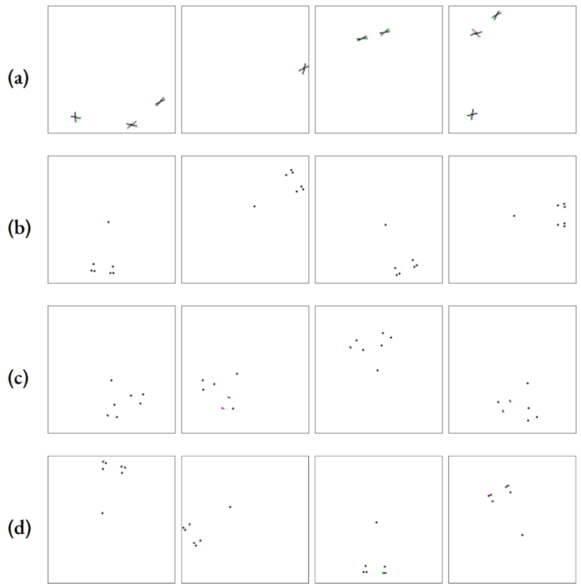


Fig. 15: Rows (a, b, c, d) present reconstruction examples from the aforementioned HTM models trained on the: crosses (to-root), narrow trees (one merge), wide trees (one merge), and final narrow trees (to-root) datasets, respectively - using their corresponding test sets. Ground truth test inputs are shown in green, reconstructed outputs in magenta, and their intersection in black - enabling clearer visual assessment of reconstruction accuracy.

4.5 Evaluation

Fig. 16 presents a violin plot illustrating the distributions of the end-to-end reconstruction error, $\mathcal{L}_r$, for the three previously discussed model types. Training and testing sets are shown side-by-side to facilitate the identification of potential overfitting. This figure primarily evaluates the lifting and merging components of HTM with respect to the image reconstruction tasks they were trained on. Conversely, Table 1 and Fig. 17 assess the classifier component for the models trained to-root: the former provides relevant performance metrics, whilst the latter visualises misclassifications using ROC and PR curves. Additionally, the generative performance is evaluated via the FID scores reported in Table 1.

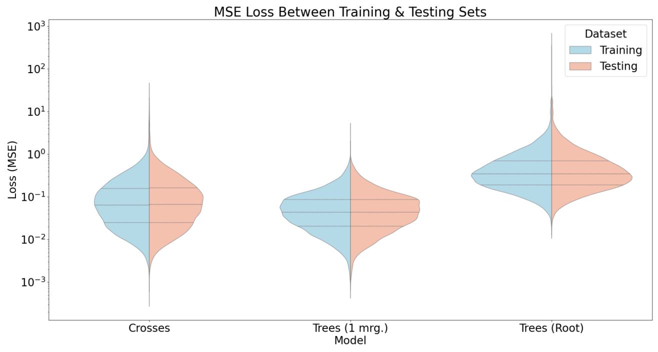


Fig. 16: Violin plot of the mean squared image reconstruction error between different HTM models and their training/testing sets. The quartiles for each are also shown.

TABLE 1: Performance metrics for the models trained to the root (i.e., incorporating learned classifiers for generation). All scores are computed on the corresponding testing sets, using macro-averaging where appropriate. Note that within HTM, leaf tokens are treated as the negative class, and merged internal nodes as the positive.

Metric	**Crosses Model**	**Trees Model**
Accuracy	0.9998	1.0000
Precision	0.9998	1.0000
Recall	0.9998	1.0000
F1 Score	0.9998	1.0000
# of False Positives	9	0
Total # of Positives	59,844	120,000
# of False Negatives	17	0
Total # of Negatives	79,844	140,000
FID Score	22.8251	3.2515

5 Discussion

5.1 Crosses

The crosses training set used in this work consists of images containing up to six tokens (i.e., three crosses), with each token represented by a 4D vector. Therefore, one could in principle employ an embedding space of size 24, allowing HTM to essentially concatenate all input tokens in any arbitrary order. However, not only does this approach not

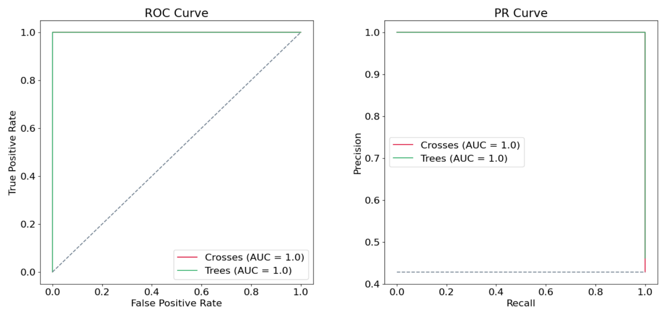


Fig. 17: Receiver Operating Characteristic (ROC) and Precision-Recall (PR) curves for the HTM classifiers, illustrating the misclassification behaviour as the decision threshold varies.

create a compressed bottleneck representation, but it would also allow the decoder to trivially reconstruct the original inputs, rendering the learned random policy redundant. Instead, the guiding intuition is that at cross structures, the information from the two 4D tokens (totalling 8 dimensions) can be more compactly represented using only four dimensions: $(x_m, y_m, \theta_1, \theta_2)$. Here, (x_m, y_m) denotes the midpoint of the line segments where they intersect, whilst θ_1 and θ_2 represent the angular directions of each line with respect to some axis. This compression is feasible when imposing fixed line lengths in the training set, as this can be identified as an inherent characteristic of the input data during training.

However, the current architecture of the mergers struggles to learn such compact representations. Specifically, extracting the directional angles θ from raw coordinates implicitly requires the modelling of periodic trigonometric functions, which standard dense layers with ReLU activations handle poorly. This is because they have to warp repeatedly across the unconstrained domain and ranges of these functions and their inverses, respectively. Preliminary experiments with 12 embedding dimensions confirm this. Nonetheless, a possible remedy could be to incorporate a periodic inductive bias, such as through the use of sinusoidal activation functions [54]. Instead, to mitigate this issue without modifying the architecture, the directions of the line segments can be encoded as 2D vectors. This allows each cross to be represented in 6 dimensions rather than 8, capturing both the intersection point and the two direction vectors, whilst still ensuring compression. As no further data structure exists to exploit, subsequent merges in the HTM model can simply concatenate these 6D cross representations, leading to the final 18-dimensional embedding space used during training.

Fig. 6 illustrates that the HTM model learns the optimum merging policy on the crosses dataset. Notably, even in input instances with fewer than three crosses - where simple concatenation would suffice - the model continues to follow this policy. Interestingly, after the first two merges yielding two cross representations, the network then prefers to concatenate these two first before merging other lines into crosses. This behavior is inline with the belief that direct reconstruction from concatenated representations is easier than decoding from a cross representation - and could perhaps scale up the hierarchy accordingly in data instances with larger P.

5.2 Trees

Similar to the concatenative setting seen in the crosses dataset, an embedding dimensionality of 14 (7 tokens, each being a 2D vector) enables accurate reconstruction, yet fails to promote the learning of a meaningful compression policy. However, with an embedding dimensionality of 2, the model is expected to merge the nearest tokens - essentially learning their midpoint - as this minimises the reconstruction error during unmerging, given the lack of directional information. Fig. 10 demonstrates the resulting policies, revealing the differences across the two datasets. In the narrow dataset, merges predominantly occur between sibling nodes first, before merging with their corresponding parents (a, b). Whilst this is seen in the wide dataset too (d), merges here typically occur between children nodes and their parents separately, shown in (e, f). Both models exhibit anomalous trajectories (c, f) - e.g., the premature merging of higher-level nodes - but the narrow configuration aligns more consistently with the expected policy. This discrepancy likely stems from the insufficient variance in distances between children and parent nodes in the wide dataset, reducing the impact a suboptimal merge has on the reconstruction loss.

It is important to note that such distance variations are necessary, as the policy network can only assess pairs of tokens at a time. In fact, when multiple trees are present in an image, the network may assign a high merging probability to nodes from different trees if they are spatially proximate. This behaviour can also occur within a single tree if node distances do not reflect the expected structure. Additional issues arise when increasing the tree height, as the factor s_f compresses the distances between children and their parents to the extent that, not only can the injected noise influence decisions, but the mean squared objective also becomes more lenient of the smaller reconstruction errors. This motivates future implementations to incorporate an attention-based mechanism, capable of leveraging information from the entire token population to improve the action selection decisions - though this may be challenging due to the varying cardinality of the discrete action space.

Although merged children were expected to be merged with their lifted parents first, the model's behaviour indicates a possible exploitation of spatial biases inherent in the dataset. Indeed, the surprisingly accurate reconstructions (Fig. 15) suggest it may be inferring directionality - despite none being explicitly provided - likely due to the data synthesis technique. In this work, trees are randomly generated to extend outwards from a central root node - never from a root positioned outside this central region - in order to ensure that the full structures remain largely within the image boundaries. Further investigations should aim to verify this behaviour, especially as even a simple greedy clustering approach successfully recovers the optimal policy with both datasets. Furthermore, the top-down recursive synthesis of tree nodes leads to varying leaf distances across different tree heights. Combined with the hypothesised behaviour regarding directional cues, this may account for the dataset's limitation to a single P value. Preliminary experiments with

P values of 3, 7, and 15 also produced surprisingly accurate reconstructions; however, the policy network frequently selected distant nodes in larger trees, likely mirroring the shorter distances observed in smaller ones. Note that the policy network is not conditioned on auxiliary information such as tree height. Additionally, training these models for more than 1 merge did not yield significant improvements, even without any exploration beyond the first curriculum.

Regarding the model trained to-root, it was initially hypothesised that 4D embedding vectors would suffice, by representing each tree using the coordinates of the parent node, and the 2D direction vector in which it extends. However, regardless of the direction provided, the network would still require trigonometric operations to generate both child nodes at each stage. As a result, two 2D direction vectors - one for each child - were allocated instead, effectively forming a 6D vector comprising three 2D positions: one for the parent and one for each child. To further simplify the generation of subsequent child nodes, the angles of deviation were excluded. This allows the decoder to reuse the same children direction vectors, recursively shrinking them by the factor s_f at each level. Nevertheless, without an additional seventh embedding dimension, the decoder may still struggle to determine when to unmerge a token into a plain lifted parent and child concept, versus when to unmerge it into two child subtrees - where the tokens are still representing their (new) parent nodes. Essentially, the seventh dimension serves to simplify the partitioning of the embedding space by acting as a signalling flag - conceptually similar to representing the height of the token in the tree.

Fig. 12 illustrates the learned policy, which generally follows a plausible trajectory as shown in (a). As with the single-merge scenario, the model still behaves unexpectedly in several instances (b, c, d), particularly in (d), where children from different subtrees are incorrectly merged with each other. More broadly, the policy tends to fail during subtree merging at the root level. This suggests that the task may still benefit from an inductive bias that captures periodicity, due to the iterative notion of merging subtrees into larger subtrees across the levels. It is also worth noting that increasing the embedding dimensionality beyond 7 yields only marginal improvements in reconstruction quality, at the cost of less consistent policies.

5.3 Generation

The GMMs for the crosses and trees models were fitted using 3 and 18 Gaussian components, respectively, based on the BIC elbow points identified in Figs. 7 and 13. Initially, a greater number of components were selected following the consistently decreasing AIC scores - a metric that prioritises the minimisation of information loss, which is generally desirable for generative tasks. However, the image quality did not improve beyond the smaller number of components suggested by BIC, which favours simpler models via a stronger penalty. Interestingly, in the crosses dataset, the chosen number of components coincided with the number of distinct conceptual root spaces - three, corresponding to the three types of images with varying P values. Additional experiments explored whether improved generations could be achieved by conditioning the GMM on root data from a single P, but this approach did not yield better results.

As shown in Figs. 8 and 14, some image generations are realistic, but a large proportion of them are inadequate. Whilst the FID scores reported in Table 1 appear favourable, they may be somewhat misleading. This is primarily due to the nature of the datasets, in which the majority of each image consists of a white background, with only a small proportion occupied by a cross or tree structure - particularly in the tree dataset, where the nodes are represented by discrete points. FID relies on a pretrained Inception-v3 network to extract feature representations, and compare the distributions of the real and generated images. However, the synthetic datasets used in this work differ substantially from the natural images on which Inception-v3 was trained (i.e., ImageNet), potentially limiting the relevance and discriminative power of the extracted features for this specific domain.

5.4 Reconstructions & Other Observations

The solid blue curves in Figs. 5 and 11, which represent the end-to-end reconstruction mean squared errors ($\mathcal{L}_r$), indicate that both final models perform well at the reconstruction task, despite the relatively long training durations - particularly in the case of the trees dataset. Interestingly, the number of epochs allocated to each curriculum does not always follow a monotonic exponential function (e.g., Fig. 5), suggesting that the curriculum learning design is highly data-dependent. Moreover, Fig. 16 indicates that none of the evaluated models exhibit significant overfitting to the training data, as evidenced by the minimal discrepancy in reconstruction errors between the training and testing sets. This is further supported by the close alignment of the quartile values within each distribution. The classifiers within HTM also demonstrate excellent performance at distinguishing between merged and leaf tokens, as shown by the near-perfect ROC and PR curves in Fig. 17. These results suggest that, if improvements are to be made in HTM to enhance generative performance, the primary focus should likely be on the lifters, mergers and policy components within the HTM pipeline. In fact, it is plausible that the models may instead be underfitting, potentially requiring more extensive training as a result.

Intermediate experiments were conducted to explore this hypothesis by increasing model capacity and training resources. Specifically, the hidden dimensions in the lifters, mergers, and policy networks were increased from the reported 256 to 1024, and the training set size, macro-batch size, and number of training epochs were all scaled up accordingly. As expected, these modifications led to lower training reconstruction losses for both models. Notably, the larger model trained on the crosses dataset also produced qualitatively better generated images, demonstrating that improving reconstruction fidelity is an effective strategy for enhancing HTM's generative capabilities. However, similar improvements were not as evident in the trees dataset. The learned policy in this case appeared to undergo minimal change, likely causing the limited gains in reconstruction performance, and thus generative quality.

Additionally, these experiments suggested that beyond architectural enhancements - such as wider or deeper autoencoder layers - one of the most impactful factors may have been the increased availability of training data. A larger dataset provides a more comprehensive representation of the underlying data manifold, which in turn can facilitate more accurate modelling. This insight motivates a potential avenue for improved results: introducing a curriculum-based data augmentation scheme. Specifically, token values could initially be confined to a smaller spatial domain (e.g., smaller than the image sizes of 128 or 256 used in this work), and translated randomly throughout training by progressively larger amounts. Although this technique's feasibility is highly data-dependent, it would allow the model to encounter a broader variety of configurations, effectively simulating a larger and more diverse input space without requiring manual data collection/synthesis. Nonetheless, such a scheme would likely necessitate careful, gradual tuning of λ_l and λ_e to accommodate the corresponding increasing embedding space required - potentially adding considerable complexity to the training procedure.

Further experiments were conducted to assess the impact of varying τ schedules, with the initial τ values also ranging from zero to one. However, only marginal and statistically insignificant improvements were observed in the end-to-end reconstruction losses and generations. Ideally, if the transport losses remained similar to those observed when τ was unused, this would indicate that the decoder is producing representations that meaningfully deviate from the up-tree, as intended. However, the observed continued reduction in the transport losses suggests that the interpolation mechanism may not be encouraging decoder divergence as expected. Instead, the decrease can be explained as a consequence of larger τ values enforcing greater similarity between the up and down-trees. Nevertheless, it remains possible that the advantages of τ scheduling become more evident in settings with deeper merging hierarchies.

It is also important to consider that, when auxiliary loss components are used, the lifter may converge towards mapping inputs to zero, thereby minimizing the step and transport losses, as well as any associated penalties. This behaviour can reduce the influence of the reconstruction term within the overall HTM loss function, resulting in degraded performance over the course of training. Although such mode collapse was not observed in the simple experiments conducted, it remains a risk. Theoretically, this issue can be mitigated by carefully tuning the relevant λ coefficients throughout the curricula, especially when improvements in auxiliary losses are not accompanied by reconstruction gains - or even occur at its expense.

6 Conclusion

This paper presented an autoencoder-based architecture for learning compressed representations through an iterative token merging process. A learned policy governs this procedure, enabling the model to construct more compact, high-quality information bottlenecks by exploiting redundancies in the input data. The recursive application of depth-wise recurrent modules allows HTM to more closely reflect the hierarchical structures commonly found in natural data, thereby fostering a more semantically organised embedding space aligned with the input distribution. Empirical results on simple synthesised datasets provide supporting evidence for the effectiveness of this approach in both image reconstruction and generation - via the sampling from a latent distribution over these resulting embedding clusters.

Among the key findings of this work, it was observed that the expressive power of even a relatively simple network design can hinder policy learning and reconstruction as the merging hierarchies deepen. This finding motivated the incorporation of auxiliary loss functions alongside the end-to-end reconstruction objective - such as the step or transport losses discussed. In addition, the trajectories shown on the crosses and trees datasets are encouraging and demonstrate the capabilities of HTM, although further experimentation is needed to assess whether interpretable policies are also learned on real-world data. Furthermore, the implemented image generation pipeline yielded promising results, particularly considering the simplicity of the network architecture used. The curriculum learning strategy likely contributed to this outcome, supporting robust reconstructions on both the training and testing sets.

Beyond the directions already discussed, future work should prioritise the parallelisation of the entire HTM training and inference pipeline, particularly the forward pass. Even relatively simple real-world datasets, such as MNIST [55], present token population sizes (P) between 50 and 200 - significantly larger than those seen in the synthetic datasets. Preliminary attempts to train HTM on a thresholded, positionally-tokenized MNIST dataset revealed that even one merging step is computationally intensive to train. Further investigation is needed to identify performance bottlenecks, particularly whether they stem from the quadratic memory complexity of the logits adjacency matrix. If so, it may be advantageous to develop a more memory-efficient graphical data structure to track active tokens and logit values across the merging hierarchy. This could also mitigate the need for cloning operations during the forward pass, which arise from the in-place operations during token creation and masking.

Once scalability is addressed, dimensionality reduction techniques (e.g., UMAP or PCA) should be employed to visualise the embedding space and root-level clusters, facilitating the interpretation of how semantic structures are embedded - especially in ambiguous cases, where an object may represent part-of or an entire image. Additionally, future work could explore the use of off-policy reinforcement learning, potentially reducing policy training time through the use of memory buffers and importance sampling. However, this may present challenges due to the variable cardinality of both the action space and active token population across the transition tuples.

Overall, the proposed HTM architecture demonstrates promise as a proof-of-concept, and warrants further investigation - with its efficacy to be eventually tested against state-of-the-art approaches.

7 Acknowledgements

The author of this paper would like to thank Dr Robert Lieck for his support and guidance throughout this project.

References

[1] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 6840–6851. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[3] R. Pappagari, P. Zelasko, J. Villalba, Y. Carmiel, and N. Dehak, "Hierarchical transformers for long document classification," in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2019, pp. 838–844. [Online]. Available: https://ieeexplore.ieee.org/document/9003958

[4] Y. Liu and M. Lapata, "Hierarchical transformers for multi-document summarization," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, A. Korhonen, D. Traum, and L. Màrquez, Eds. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 5070–5081. [Online]. Available: https://aclanthology.org/P19-1500/

[5] S. Yang, Y. Zhao, and K. Tu, "PCFGs can do better: Inducing probabilistic context-free grammars with many symbols," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tur, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, and Y. Zhou, Eds. Online: Association for Computational Linguistics, Jun. 2021, pp. 1487–1498. [Online]. Available: https://aclanthology.org/2021.naacl-main.117/

[6] Y. Kim, C. Dyer, and A. Rush, "Compound probabilistic context-free grammars for grammar induction," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, A. Korhonen, D. Traum, and L. Màrquez, Eds. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 2369–2385. [Online]. Available: https://aclanthology.org/P19-1228/

[7] D. H. Younger, "Recognition and parsing of context-free languages in time n3," *Information and Control*, vol. 10, no. 2, pp. 189–208, 1967. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S001999586780007X

[8] D. Bolya and J. Hoffman, " Token Merging for Fast Stable Diffusion ," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2023, pp. 4599–4603. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPRW59228.2023.00484

[9] 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," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy

[10] D. Bolya, C.-Y. Fu, X. Dai, P. Zhang, C. Feichtenhofer, and J. Hoffman, "Token merging: Your vit but faster," 2023. [Online]. Available: https://arxiv.org/abs/2210.09461

[11] K. Haroun, T. Allenet, K. Ben Chehida, and J. Martinet, "Dynamic Hierarchical Token Merging for Vision Transformers," in *VISAPP-2025 - 20th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, Porto, Portugal, Feb. 2025. [Online]. Available: https://hal.science/hal-04885469

[12] Y. Rao, W. Zhao, B. Liu, J. Lu, J. Zhou, and C.-J. Hsieh, "Dynamicvit: Efficient vision transformers with dynamic token sparsification," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 13 937–13 949. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/747d3443e319a22747fbb873e8b2f9f2-Paper.pdf

[13] R. A. Shaikh, S. S. Zemljic, S. Tull, and S. Clark, "The conceptual vae," 2022. [Online]. Available: https://arxiv.org/abs/2203.11216

[14] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, K. Knight, A. Nenkova, and O. Rambow, Eds. San Diego, California: Association for Computational Linguistics, Jun. 2016, pp. 1480–1489. [Online]. Available: https://aclanthology.org/N16-1174/

[15] K. Kowsari, D. E. Brown, M. Heidarysafa, K. Jafari Meimandi, M. S. Gerber, and L. E. Barnes, "Hdltex: Hierarchical deep learning for text classification," in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2017, pp. 364–371. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8260658

[16] A. Katole, K. Yellapragada, A. Bedi, S. Kalra, and S. C. Mynepalli, "Hierarchical deep learning architecture for 10k objects classification," *Computer Science Information Technology*, vol. 5, 09 2015. [Online]. Available: https://www.researchgate.net/publication/281607765_Hierarchical_Deep_Learning_Architecture_For_10K_Objects_Classification

[17] W. Song, S. Oh, S. Mo, J. Kim, S. Yun, J.-W. Ha, and J. Shin, "Hierarchical context merging: Better long context understanding for pre-trained llms," 2024. [Online]. Available: https://arxiv.org/abs/2404.10308

[18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423/

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6795963

[20] P. Nawrot, S. Tworkowski, M. Tyrolski, L. Kaiser, Y. Wu, C. Szegedy, and H. Michalewski, "Hierarchical transformers are more efficient language models," in *Findings of the Association for Computational Linguistics: NAACL 2022*, M. Carpuat, M.-C. de Marneffe, and I. V. Meza Ruiz, Eds. Seattle, United States: Association for Computational Linguistics, Jul. 2022, pp. 1559–1571. [Online]. Available: https://aclanthology.org/2022.findings-naacl.117/

[21] Z. Pan, B. Zhuang, J. Liu, H. He, and J. Cai, "Scalable vision transformers with hierarchical pooling," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 377–386. [Online]. Available: https://openaccess.thecvf.com/content/ICCV2021/html/Pan_Scalable_Vision_Transformers_With_Hierarchical_Pooling_ICCV_2021_paper.html

[22] M. Ding, W. Zheng, W. Hong, and J. Tang, "Cogview2: Faster and better text-to-image generation via hierarchical transformers," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 16 890–16 902. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/6baec7c4ba0a8734ccbd528a8090cb1f-Paper-Conference.pdf

[23] Y. Liu, Y.-H. Wu, G. Sun, L. Zhang, A. Chhatkuli, and L. V. Gool, "Vision transformers with hierarchical attention," 2024. [Online]. Available: https://arxiv.org/abs/2106.03180

[24] 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," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 10 012–10 022. [Online]. Available: https://openaccess.thecvf.com/content/ICCV2021/html/Liu_Swin_Transformer_Hierarchical_Vision_Transformer_Using_Shifted_Windows_ICCV_2021_paper

[25] Z. Zhang, H. Zhang, L. Zhao, T. Chen, S. Arik, and T. Pfister, "Nested hierarchical transformer: Towards accurate, data-efficient and interpretable visual understanding," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 3, pp. 3417–3425, Jun. 2022. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/20252

[26] Z. Dai, G. Lai, Y. Yang, and Q. Le, "Funnel-transformer: Filtering out sequential redundancy for efficient language processing," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 4271–4282.

[Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/2cd2915e69546904e4e5d4a2ac9e1652-Paper.pdf

[27] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. [Online]. Available: https://ieeexplore.ieee.org/document/5206848

[28] F. Murtagh and P. Legendre, "Ward's hierarchical agglomerative clustering method: Which algorithms implement ward's criterion?" *Journal of Classification*, vol. 31, no. 3, p. 274–295, Oct. 2014. [Online]. Available: http://dx.doi.org/10.1007/s00357-014-9161-z

[29] Y. Wang and Y. Yang, "Efficient visual transformer by learnable token merging," 2024. [Online]. Available: https://arxiv.org/abs/2407.15219

[30] M. Kim, S. Gao, Y.-C. Hsu, Y. Shen, and H. Jin, "Token fusion: Bridging the gap between token pruning and token merging," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, January 2024, pp. 1383–1392. [Online]. Available: https://openaccess.thecvf.com/content/WACV2024/html/Kim_Token_Fusion_Bridging_the_Gap_Between_Token_Pruning_and_Token_WACV_2024_paper.html

[31] Q. Cao, B. Paranjape, and H. Hajishirzi, "PuMer: Pruning and merging tokens for efficient vision language models," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 12 890–12 903. [Online]. Available: https://aclanthology.org/2023.acl-long.721/

[32] A. Vahdat and J. Kautz, "Nvae: A deep hierarchical variational autoencoder," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 19 667–19 679. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/e3b21256183cf7c2c7a66be163579d37-Paper.pdf

[33] R. Child, "Very deep vaes generalize autoregressive models and can outperform them on images," 2021. [Online]. Available: https://arxiv.org/abs/2011.10650

[34] C. K. Sø nderby, T. Raiko, L. Maalø e, S. r. K. Sø nderby, and O. Winther, "Ladder variational autoencoders," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/6ae07dcb33ec3b7c814df797cbda0f87-Paper.pdf

[35] R. Ranganath, D. Tran, and D. Blei, "Hierarchical variational models," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 324–333. [Online]. Available: https://proceedings.mlr.press/v48/ranganath16.html

[36] E. Mathieu, C. Le Lan, C. J. Maddison, R. Tomioka, and Y. W. Teh, "Continuous hierarchical representations with poincaré variational auto-encoders," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/0ec04cb3912c4f08874dd03716f80df1-Paper.pdf

[37] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-VAE: Learning basic visual concepts with a constrained variational framework," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=Sy2fzU9gl

[38] R. Togo, N. Nakagawa, T. Ogawa, and M. Haseyama, "Concvae: Conceptual representation learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 4, pp. 7529–7541, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10584324

[39] Z. Liu, Y. Liu, Z. Yu, Z. Yang, Q. Fu, Y. Guo, Q. Liu, and G. Wang, "Pt-vae: Variational autoencoder with prior concept transformation," *Neurocomputing*, vol. 638, p. 130129, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092523122500801X

[40] L. Maalø e, M. Fraccaro, V. Liévin, and O. Winther, "Biva: A very deep hierarchy of latent variables for generative modeling," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/9bdb8b1faffa4b3d41779bb495d79fb9-Paper.pdf

[41] M. Willetts, X. Miscouridou, S. Roberts, and C. Holmes, "Relaxed-responsibility hierarchical discrete vaes," 2021. [Online]. Available: https://arxiv.org/abs/2007.07307

[42] J. D. Havtorn, J. Frellsen, S. Hauberg, and L. Maaløe, "Hierarchical vaes know what they don't know," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 4117–4128. [Online]. Available: https://proceedings.mlr.press/v139/havtorn21a.html

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html

[44] S. Yang, Y. Zhao, and K. Tu, "Neural bi-lexicalized PCFG induction," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, C. Zong, F. Xia, W. Li, and R. Navigli, Eds. Online: Association for Computational Linguistics, Aug. 2021, pp. 2688–2699. [Online]. Available: https://aclanthology.org/2021.acl-long.209/

[45] A. Drozdov, P. Verga, M. Yadav, M. Iyyer, and A. McCallum, "Unsupervised latent tree induction with deep inside-outside recursive auto-encoders," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 1129–1141. [Online]. Available: https://aclanthology.org/N19-1116/

[46] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, no. 3–4, p. 229–256, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992696

[47] E. Pignatelli, J. Ferret, M. Geist, T. Mesnard, H. van Hasselt, O. Pietquin, and L. Toni, "A survey of temporal credit assignment in deep reinforcement learning," 2024. [Online]. Available: https://arxiv.org/abs/2312.01072

[48] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=rkE3y85ee

[49] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," 2017. [Online]. Available: https://arxiv.org/abs/1611.00712

[50] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a.html

[51] A. R. Carmo, J.-A. Delamer, Y. Watanabe, R. Ventura, and C. Ponzoni Carvalho Chanel, "Entropy-based adaptive exploit-explore coefficient for Monte-Carlo path planning," in *Proceedings of the 10th International Conference on Prestigious Applications of Intelligent Systems (PAIS 2020), a subconference of the 24th European Conference on Artificial Intelligence (ECAI 2020)*, Online, Spain, Aug. 2020, pp. 1–8. [Online]. Available: https://hal.science/hal-03125159

[52] S.-h. Liu, M. Mernik, and B. Bryant, "Entropy-driven exploration and exploitation in evolutionary algorithms," 01 2006. [Online]. Available: https://www.researchgate.net/publication/228689187_Entropy-driven_exploration_and_exploitation_in_evolutionary_algorithms

[53] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:6628106

[54] L. Ziyin, T. Hartwig, and M. Ueda, "Neural networks fail to learn periodic functions and how to fix it," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds.,

vol. 33. Curran Associates, Inc., 2020, pp. 1583–1594. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/1160453108d3e537255e9f7b931f4e90-Paper.pdf

[55] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998. [Online]. Available: https://hal.science/hal-03926082