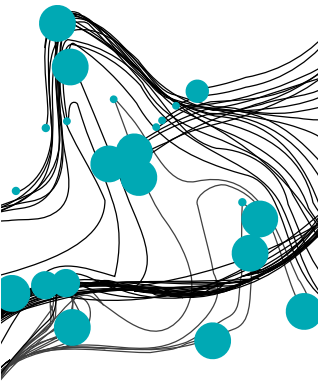
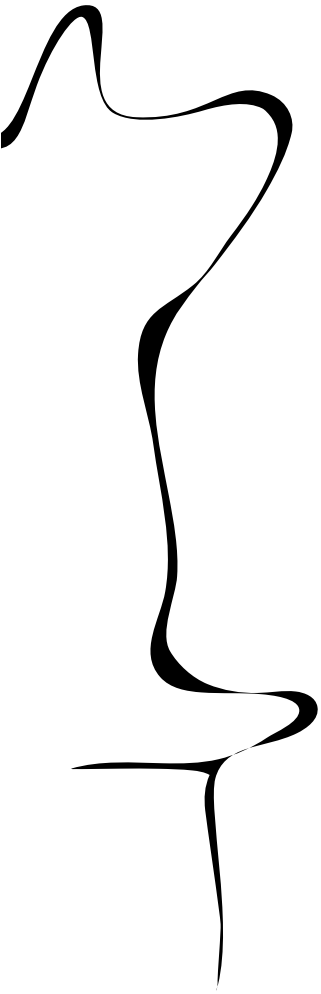


.77507

DMB

DATABASE MANAGEMENT
AND
BIOMETRICS



CAST: CLUSTERING ATTENTION USING SURROGATE TOKENS FOR EFFICIENT TRANSFORMERS

Adjorn van Engelenhoven

MASTER THESIS ASSIGNMENT

Committee:
Dr. Nicola Strisciuglio (Chair)
Dr. Estefania Talavera (Supervisor)
Dr. Mariët Theune (External)

June, 2023

2023DMB0007
Data Management and Biometrics
EEMathCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Abstract

The Transformer architecture has shown to be a powerful tool for a wide range of tasks. It is based on the self-attention mechanism, which is an inherently computationally expensive operation with quadratic computational complexity: memory usage and compute time increase quadratically with the length of the input sequences, thus limiting the application of Transformers. In this work, we propose a novel Clustering self-Attention mechanism using Surrogate Tokens (CAST), to optimize the attention computation and achieve efficient transformers. CAST utilizes learnable surrogate tokens to construct a cluster affinity matrix, used to cluster the input sequence and generate novel cluster summaries. The self-attention from within each cluster is then combined with the cluster summaries of other clusters, enabling information flow across the entire input sequence. CAST improves efficiency by reducing the complexity from $O(N^2)$ to $O(\alpha N)$ where N is the sequence length, and α is constant according to the number of clusters and samples per cluster. We show that CAST performs better than or comparable to the baseline Transformers on long-range sequence modeling tasks, while also achieving state-of-the-art results on time and memory efficiency.

1 Introduction

The Transformer architecture [45] has revolutionized many fields within machine learning such as translation [45], summarization [34], text generation [5], sentiment classification [39], and also tasks like image classification [13], object detection [27], and protein folding [17]. The self-attention mechanism stands at the core of its strengths. It allows the Transformer to directly model long-range dependencies within a sequence without the need for a hidden state like in recurrent neural networks [16]. However, the self-attention mechanism has an inherent large memory cost, since its complexity grows quadratically with the input sequence length. With these memory requirements and the ever-increasing use and size of large language models, such as the GPT series [4, 35] and LLaMA [44], a need for more efficient attention mechanisms has emerged [9].

Current implementations of more efficient self-attention mechanisms can be roughly grouped into the following categories: (1) apply self-attention on subsets of the input sequences (sparsification) [1, 3, 11, 20, 30, 41, 49], (2) approximate the self-attention mechanism with a lower complexity [8, 26, 47], and (3) remove self-attention in favor of a lower complexity similar operation [14, 23, 38, 43].

One way of sparsifying self-attention is to cluster the input sequence, which was done by both the Reformer [20], and SMYRF [11]. However, these self-attention alternatives both cluster in a non-learnable and algorithmic way, which can cause them to get stuck in local minima resulting in a worse performance.

In this work, we introduce a new efficient variant of self-attention, *Clustering Attention using Surrogate Tokens* (CAST). CAST employs clustering to self-attention but introduces two novel ideas: (1) The introduction of learnable surrogate tokens S, which are used to cluster the input sequence, but also directly influence the result. (2) The creation of *cluster summaries*, which allows for information to flow between clustered parts of the input sequence.

CAST learns to cluster tokens that would have a strong connection in the original attention matrix by clustering based on a similarity matrix between the surrogate tokens, queries, and keys. Standard self-attention is applied within clusters, where its result is combined with cluster summaries based on a previously created similarity matrix. This allows for each token to retrieve information from the rest of the sequence, improving stability and performance.

In this work, we aim to significantly improve memory usage and time efficiency of the Transformer architecture with the use of CAST. More specifically, we investigate the following research questions:

- RQ(1) How does the memory and computational complexity of CAST compare to other efficient Transformers?
- RQ(2) How does CAST generalize over both long- and short-range machine learning tasks?
- RQ(3) How does the choice of the configuration of surrogate tokens affect the generalizability of CAST?

Experimentally we show that CAST is significantly faster than other efficient Transformer variants, and match or improve the performance of a standard Transformer on long-sequence modeling tasks.

In Section 2, background information about the Transformer architecture is given. In Section 3, related work on efficient Transformers is discussed. In Section 4, CAST is explained in more detail. In Section 5, the experimental setup is described. In Section 6 the results of the experiments are discussed. In Section 7, the research questions are answered. In Section 8, a conclusion is drawn, and in Section 9, future work on CAST is proposed.

2 Transformer Background

In this section, the Transformer architecture is elaborated using Figure 1. Starting from the input the different blocks will be explained one by one, until the output.

First, we have the input embedding. The Transformer is first and foremost a sequence model. As input, it takes a sequence of **tokens** which each are represented using a vector of numbers. In Natural Language Processing (NLP) these tokens correspond to words, however, words cannot be inherently represented using vectors. To still be able to use words as input, an input embedding is created to map words to vectors. Within NLP, these representational vectors are often created by methods like Word2Vec [33] or SentencePiece 32000 [22], which learn representations of words based on contextual information and word co-occurrence. However, it is also possible to learn the input embedding is learned on the fly while the Transformer is being trained [45]. The input embedding block's sole purpose is to represent otherwise non-numerical data, as a sequence of vectors such that it can be used as input for the Transformer.

After the tokens have been represented using vectors, a Positional Encoding is added. This is done because the Transformer does not inherently model positions in its input. The addition of a positional encoding gives the Transformer the ability to reason about the positions of tokens. In the original paper, the authors decided on a sinusoidal positional encoding:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d}) \tag{1}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d}), \tag{2}$$

where $PE_{pos,2i}$, pos indicates the position in the sequence while $2i$ represents the index for the vector at that position. The intuition is that lower values of i have a larger phase, allowing the Transformer to reason about more global positions, while higher values of i have a smaller phase allowing the Transformer to reason more about relative positions. Although sinusoidal positional encoding is used often in practice, it is not necessarily required. Instead, the positional encoding could also be learned on the fly, while training the Transformer. The purpose of the Positional Encoding block is to allow the Transformer to reason about the position of tokens inside the sequence because it otherwise would not inherently do so.

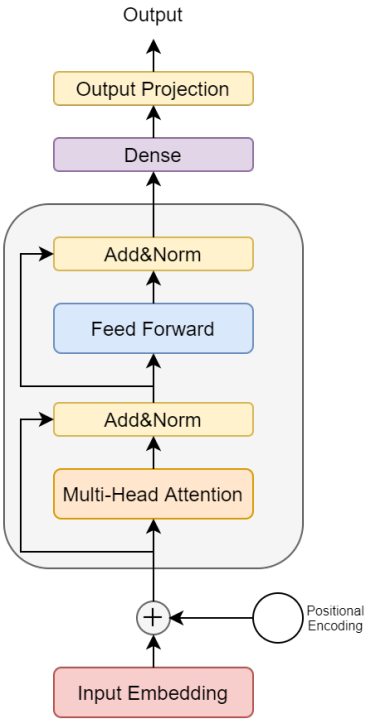


Figure 1. The Transformer Encoder Architecture

After the tokens have received the positional encoding Multi-Head self-attention is applied. Multi-Head self-attention is the core of the Transformer and is also the block with the highest complexity. The first step is to apply three linear projections to the tokens separately resulting in three matrices, the queries (Q), keys (K), and values (V). A simplified way to understand these projections is that: Q represents the type of information a certain token is requesting, K represents the type of information a certain token holds, and V represents the information that a certain token would want to forward.

Now the idea is to create a matrix of the similarity between each token's query and all tokens' key. For any given query the result will mean: "How similar is the type of information every other token has to the type of information that the given query is requesting".

After these projections have been applied the queries, keys, and values are split into h equally sized chunks along the representational dimension. Each of these chunks represents a **head** inside the Multi-Head attention mechanism. Scaled dot product attention is then applied per head as follows:

$$A(Q, K, V) = \text{softmax}\left(\frac{QK^\intercal}{\sqrt{d_k}}\right), \tag{3}$$

where d_k indicates the size of the chunks along the representational dimension. The dot product between a query and a key essentially represents their similarity, meaning that the matrix multiplication between all queries and all keys gives us a matrix of similarity scores for each combination. This resulting matrix is often called **scores** or **attention matrix**. A softmax operation is then also applied on the matrix row-wise along the queries. The attention matrix is

then multiplied with the values, making the attention matrix act as weights in a weighted sum over all values, retrieving the new values for the tokens. After the scaled dot product attention has been applied to each of the chunks separately, the results are concatenated again to match the original input dimensions. Lastly, one more linear projection is applied to the result before it is passed on to the next block. In the next block a residual connection is used to sum the input of the multi-headed attention block with its output and Layer Normalization [2] is applied.

Lastly, there is the feedforward block, for each respective token it applies a parameterized linear function to the values. This can be done on each of the tokens separately, making this function highly parallelizable. The exact role of the feedforward layer in the Transformer architecture has long been unclear, but recently it has been shown that in large language models, factual information is stored in the weights of the Feed Forward layer, acting like an information lookup [32]. Blocks containing Multi-Head Attention block and the Feed Forward block are then sequentially repeated, after which the resulting features go through a projection to retrieve the desired output.

3 Related Work

There has been a significant amount of prior related work regarding the improvement of the efficiency of Transformers. In this section, we specifically focus on the closely related works of chunking attention, clustering attention, and the current state-of-the-art, structured-state-space-based models.

Chunking attention. One obvious way to solve the quadratic complexity of self-attention is to chunk the given sequence into smaller pieces and apply self-attention within those pieces. This is known as Local Attention [28]. However, by chunking the sequence, no information can be passed between chunks, causing a decrease in task performance below that of the standard Transformer. Several works have sparsified the attention matrix by windowing or chunking the sequence [1, 3, 7, 28, 49]. Some opted for applying attention in a sliding window manner, but the use of global attention to special tokens, such as the "CLS", is also common among the original efficient Transformers of the LRA benchmark [1, 3, 49]. These models also use the "CLS" token for the final classification, allowing all parts of the sequence to contribute to the final result. BigBird [49] combined global attention, window attention, and random blocks of attention to achieve state-of-the-art performance on the LRA benchmark. Despite the efficiency gains of the chunking of self-attention, it does not necessarily model long-range dependencies well. Multiple rounds of self-attention can be necessary to create a large enough receptive field to model long-range dependencies. Although chunking has shown its effectiveness, it does not model long-range dependencies well, since no information can flow from distant parts of the input sequence.

Clustering attention. One way to easily model long-range dependencies is the clustering of the input sequence which is only partially dependent on the order of the input. Specifically, the Reformer [20] and its descendant SMYRF [11] both use locality-sensitive hashing (LSH) to apply clustering to the input sequence and then apply a form of attention. The Reformer first uses the constraint of the queries and keys being equal such that the attention matrix is symmetric. Then to create the clusters they define a random matrix $R \in \mathbb{R}^{d_h \times \frac{N_c}{2}}$, which is then matrix multiplied with query-key. The query-keys are then clustered based on the result of $argmax([X_{qk}R \oplus -X_{qk}R])$. The symbol $\oplus$ stands for concatenation, while X_{qk} stands for the shared query-key representation. The resulting clusters are of different sizes, making it impossible to be easily computed on conventional hardware. This drawback reflects in the slower speed of the Reformer in the Long Range Arena Benchmark [42].

SMYRF efficiently computes asymmetric clustering of queries and keys, that is a query is not necessarily clustered with its corresponding key. Unlike the Reformer, SMYRF also creates balanced clusters of constant size and thus achieving better computational efficiency. Although both these clustering Transformers do model long-range dependencies, clustering also creates problems. The random initialization of the network causes queries and keys to be clustered randomly at first. As a result, the weight update that the queries and keys receive is based only on the information that is inside these clusters. Furthermore, gradients could also be unstable when a query or key switches from one cluster to another. Ideally, an efficient Transformer retains the original strength of the Transformer which is the information flow throughout the entire input sequence through the self-attention mechanism. In contrast, our approach keeps the information flow and introduces more stability through the use of cluster summaries, which act as an indicator of the type of information that can be obtained in a given cluster.

Structured State Space Models. More recent work on efficient sequence models includes Structured State Space Models (SSSM), such as S4 [14], S5 [38], and MEGA [30] which are currently the state-of-the-art on long-range sequence modeling benchmarks. Structured State Space Models do not use the self-attention mechanism, and rely on a learnable state space to capture relevant dependencies in a sequence, which is defined as followed:

$$\begin{aligned} x'(t) &= \mathbf{A}x(t) + \mathbf{B}u(t) \\ y(t) &= \mathbf{C}x(t) + \mathbf{D}u(t), \end{aligned} \tag{4}$$

where $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$ are parameterized matrices, which map the input signal $u(t)$ to an output $y(t)$ through a latent space $x(t)$. However, we do not investigate these types of architectures further, as they are no longer related to the Transformer architecture and thus the self-attention mechanism.

4 Method: Clustering Attention using Surrogate Tokens

In this section, the proposed method is described in detail. First, we describe the intuition behind our method in Section 4.1. We then describe our method in a Single-Headed Attention scenario in Section 4.2, after which the clustering mechanism is expanded in more detail in Section 4.2. We further explain CAST in a Multi-Headed scenario in Section 4.3. Lastly, the complexity of CAST is discussed in Section 4.4.

4.1 Intuition

A query (Q) and key (K) which are in the same direction with a large enough magnitude will end up with a large score in the self-attention matrix. This relationship can be exploited for clustering by defining some static clustering directions, determining the similarity of all queries and all keys with these clustering directions, and then clustering based on this similarity. However, this approach has two problems: (1) when clustering directions are randomly initialized, their configuration might not be optimal for the task that is trained on, and (2) when training is started, queries and keys are clustered randomly. Consequentially, the gradient of the queries and keys is only based on the self-attention within their cluster, making it impossible for queries and keys from different clusters to align themselves according to the loss. To alleviate this problem, we design CAST to ensure that the clustering directions are learnable and that each token receives information from all clusters. In CAST, surrogate tokens represent the learnable clustering directions and are used as a surrogate for finding similar queries and keys. The weight of each cluster's result is based on the similarity of its query and the clustering direction. For the cluster that the token is a part of, we simply apply self-attention. For the rest of the clusters, cluster summaries are created, based on the similarity of a token's key with the direction of the cluster it belongs to.

4.2 Single-Head Clustering Attention using Surrogate Tokens

CAST is an extension of the self-attention mechanism in the Transformer architecture [45]. We create query-key-value combinations from the input sequence $\mathbf{X} \in \mathbb{R}^{N \times d}$, where N is the input sequence length, and d the feature embedding dimension:

$$\mathbf{Q} = \mathbf{X}W_q, \quad \mathbf{K} = \mathbf{X}W_k, \quad \mathbf{V} = \mathbf{X}W_v, \qquad \in \mathbb{R}^{N \times d}, \tag{5}$$

where $W_q, W_k, W_v \in \mathbb{R}^{d \times d}$ are learnable parameters for the queries ($\mathbf{Q}$), keys ($\mathbf{K}$), and values ($\mathbf{V}$) respectively. To create clusters and lower the computational complexity, we define learnable surrogate tokens $\mathbf{S} \in \mathbb{R}^{N_c \times d}$, where N_c indicates the number of clusters. The surrogate tokens represent the learnable clustering directions and are used as a surrogate for finding similar queries and keys. Then we compute the similarity matrices with the surrogate tokens for the queries ($\mathbf{A}_q$) and the keys ($\mathbf{A}_k$). We combine these similarity matrices using a ratio $\sigma(\varphi) : 1-\sigma(\varphi)$ based on a linear transformation of X, where σ indicates the sigmoid function.

$$\begin{aligned} \mathbf{A}_q = \mathbf{Q}\mathbf{S}^T, \mathbf{A}_k = \mathbf{K}\mathbf{S}^T & \qquad \in \mathbb{R}^{N \times N_c} \\ \boldsymbol{\varphi} = \mathbf{X}W_\varphi + b_\varphi & \qquad \in \mathbb{R}^{N \times 1} \\ \mathbf{A}_g = \sigma(\boldsymbol{\varphi}) \odot f_2(\mathbf{A}_q) + (1 - \sigma(\boldsymbol{\varphi})) \odot f_2(\mathbf{A}_k) & \qquad \in \mathbb{R}^{N \times N_c}, \end{aligned} \tag{6}$$

where $\sigma(\varphi)$ is a sigmoid function applied to a linear transformation of $\mathbf{X}$, which is represented as $\mathbf{X}W_\varphi + b_\varphi$. Where $W_\varphi \in \mathbb{R}^{d \times 1}$ and $b_\varphi \in \mathbb{R}^1$ are learnable parameters. The function $f(\cdot)$ indicates an attention function, which in this work includes the classical softmax and the Laplace function from MEGA [30]. In the case of softmax, $f_i(\cdot)$ indicates that the softmax is applied over the dimension i of the matrix. Here the softmax is applied to the dimensions holding corresponding to the different clusters. The symbol $\odot$ represents element-wise multiplication. Subsequently, the calculated similarities are used to cluster the input sequence using a clustering mechanism G, computed as $G : \mathbb{R}^{N \times N_c}, \mathbb{R}^{N \times *} \rightarrow \mathbb{R}^{N_c \times \kappa \times *}$, where $*$ indicates any given shape, and κ indicates the size of a cluster. Furthermore, let G^{-1} indicate the reverse of the function G, such that $G^{-1} : \mathbb{R}^{N \times N_c}, \mathbb{R}^{N_c \times \kappa \times *} \rightarrow \mathbb{R}^{N \times *}$, where in the event of a token being contained in two clusters the sum is calculated. Then, standard self-attention is applied within each cluster as follows:

$$\mathbf{R}_{intra} = f\left(\frac{\mathbf{Q}_g \mathbf{K}_g^T}{\tau}\right)\mathbf{V}_g \qquad \in \mathbb{R}^{N_c \times \kappa \times d}, \tag{7}$$

where $\mathbf{Q}_g = G(\mathbf{A}_g, \mathbf{Q})$, $\mathbf{K}_g = G(\mathbf{A}_g, \mathbf{K})$, $\mathbf{V}_g = G(\mathbf{A}_g, \mathbf{V})$, and τ is a scalar depending on the used attention function. Here $\mathbf{R}_{intra}$ indicates the result of attention within the clusters. To create a gradient between tokens from different clusters we apply attention between clusters as well. To do this, we define value summaries V_{sum}, which is a weighted sum of all values within each cluster where the weights $\mathbf{A}_{value}$ are based on $\mathbf{A}_k$ and φ as follows:

$$\begin{aligned} \mathbf{A}_{value} = G\left(\mathbf{A}_g, \frac{\mathbf{A}_k \odot \phi(-\boldsymbol{\varphi})}{\tau_k}\right)\mathbf{I}'_{N_c} & \qquad \in \mathbb{R}^{N_c \times \kappa \times 1} \\ \mathbf{R}_{inter} = f_2\left(\mathbf{A}_{value}\right)\mathbf{V}_g^T & \qquad \in \mathbb{R}^{N_c \times 1 \times d}, \end{aligned} \tag{8}$$

where I'_{N_c} indicates the expanded identity matrix I_{N_c} such that $I'_{N_c} \in \mathbb{B}^{N_c \times N_c \times 1}$, the function $\phi(x) = \text{Softplus}(x) + 1$ [50], and τ_k is a scaling factor. After this, we use A_Q and r

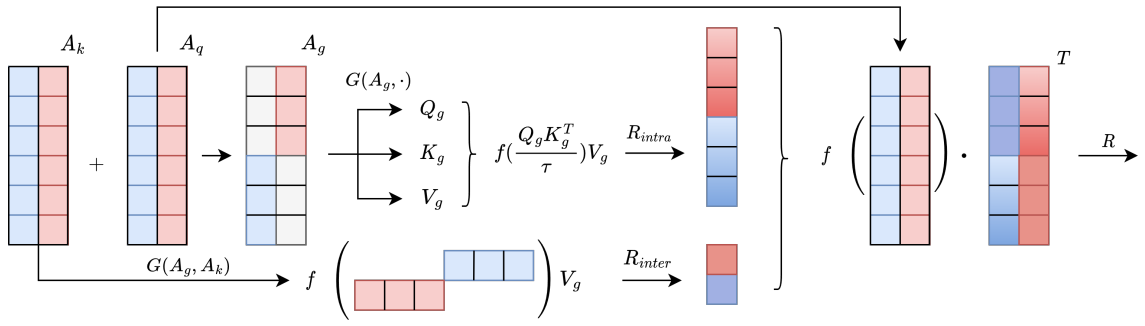


Figure 2. Sketch of the proposed method. With the queries (Q), keys (K), and values (V), we create the surrogate token similarities A_q, and A_k. Here, A_q represents the similarity between the queries and surrogate tokens, and A_k the similarity between the keys and surrogate tokens. They are combined to create a final similarity A_g for each token to each cluster. We then use this clustering of tokens and create the clustered queries (Q_g), keys (K_g), and values (V_g). Within each cluster, self-attention is applied resulting in R_{intra}. Furthermore, A_k is also clustered and matrix multiplied with V_g to create a summary per cluster resulting in R_{inter}. The results R_{intra} and R_{inter} are then combined using A_q as the weights for a weighted sum, resulting in R. Another linear projection O is then applied R and passed on to the feedforward layer of the Transformer.

to create an attention matrix used as a weighted sum for the value summaries and attention within clusters:

$$\begin{aligned}
\mathbf{A}_{sum} &= f_3\left(\frac{\mathbf{A}_q \odot \phi(\boldsymbol{\varphi})}{\tau_q}\right) && \in \mathbb{R}^{N\times N_c} \\
\mathbf{A}_{inter} &= (\mathbf{A}_{sum} \odot \hat{M}) && \in \mathbb{R}^{N\times N_c} \\
\mathbf{A}_{intra} &= G(\mathbf{A}_g, \mathbf{A}_{sum} \odot M) && \in \mathbb{R}^{N\times N_c} \\
\mathbf{R} &= G^{-1}(\mathbf{A}_g, \mathbf{A}_{intra}\mathbf{R}_{intra}) + \mathbf{A}_{inter}\mathbf{R}_{inter} && \in \mathbb{R}^{N\times d},
\end{aligned} \tag{9}$$

where $M \in (0,1)^{N\times N_c}$ is a mask where $M_{i,j} = 1$ if $X_i \in G(\mathbf{A}_g, X)_j$ and $M_{i,j} = 0$ if $X_i \notin G(\mathbf{A}_g, X)_j$. As a result of this operation, R is a weighted sum according to A_{sum} of the attention within clusters ($\mathbf{R}_{intra}$) and the summaries of the clusters ($\mathbf{R}_{inter}$). The final output $\mathbf{O}$ is then calculated as $\mathbf{O} = \mathbf{R}W_o \in \mathbb{R}^{N\times d}$, where $W_o \in \mathbb{R}^{d\times d}$ are learnable parameters. $\mathbf{O}$ is then passed on to the rest of the standard Transformer architecture.

Clustering Mechanisms. The clustering mechanism is an integral part of CAST and serves to group inter-important tokens of the input sequence. Here, this inter-importance is measured as the similarity scores in the attention matrix $\mathbf{A}_g$. We define two clustering mechanisms that maximize the similarity score per cluster: the *Top-K* clustering mechanism and the Single Assignment (SA) *Top-K* clustering mechanism. The practical difference can between *Top-K* and *SA Top-K* can be seen in Figure 3.

A) *Top-K Clustering Mechanism*
The *Top-K* clustering mechanism is a naive approach to clustering the input sequence, the indices of the largest K elements in $\mathbf{A}_g$ are taken per cluster and used to index the original sequence. Because *Top-K* simply maximizes the similarity scores per cluster separately, it is possible for any token to be contained in anywhere between 0 and N_c clusters. This attribute of *Top-K* can be useful in case padding is used, by setting the similarity scores of padding to 0, it can be ensured that padding is never taken into consideration when applying attention within clusters. However, in static sequence domains, like images, it can also cause certain parts of the input sequence to never be clustered. A pseudocode of the *Top-K* clustering mechanism can be seen in Appendix A.1.

B) *Single Assignment Top-K Clustering Mechanism*
The Single Assignment *Top-K* clustering mechanism is an approach that has the constraint that every part of the sequence can only be assigned to a single cluster. With this constraint, we can ensure that every token is part of the result of CAST and thus has a gradient. To achieve this constraint, we cluster tokens in descending order according to their maximum score in $\mathbf{A}_g$. When a cluster has reached its desired cluster size, we no longer assign tokens to this cluster. A pseudocode of the *Top-K* clustering mechanism can be seen in Appendix A.2.

4.3 Clustering Attention using Surrogate Tokens Multi-head

To apply CAST in a multi-headed scenario, the surrogate tokens $\mathbf{S}$ are also split into multiple heads such that $S \in \mathbb{R}^{N_c\times h\times d_h}$, where h is the number of heads, and $d_h = \frac{d}{h}$. The

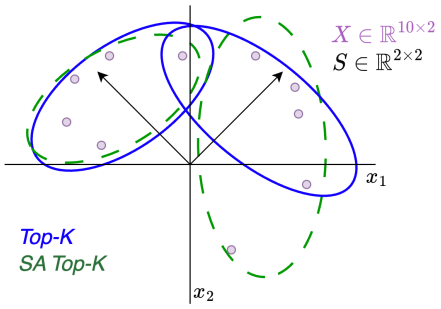


Figure 3. The practical difference between the *Top-K* and *SA Top-K* clustering mechanisms. Here, S indicates the clustering direction of two surrogate tokens. The blue and green dashed circles indicate the clusters that the *Top-K* and SA *Top-K* clustering mechanisms would create, respectively.

score $\mathbf{A}_g$ is then computed as follows:

$$\begin{aligned}
\mathbf{A}_q &= \mathbf{Q}\mathbf{S}^T && \in \mathbb{R}^{N\times h\times N_c} \\
\mathbf{A}_k &= \mathbf{K}\mathbf{S}^T && \in \mathbb{R}^{N\times h\times N_c} \\
\boldsymbol{\varphi} &= \mathbf{X}W_\varphi + b_\varphi && \in \mathbb{R}^{N\times 1} \\
A_q^s &= \sigma(\boldsymbol{\varphi}) \odot f_2(\sum_h \mathbf{A}_{q_{:,h,:}}) && \in \mathbb{R}^{N\times N_c} \\
A_k^s &= (1-\sigma(\boldsymbol{\varphi})) \odot f_2(\sum_h \mathbf{A}_{k_{:,h,:}}) && \in \mathbb{R}^{N\times N_c} \\
\mathbf{A}_g &= A_q^s + A_k^s && \in \mathbb{R}^{N\times N_c}
\end{aligned} \tag{10}$$

In short, we sum the similarity scores $\mathbf{A}_q$ and $\mathbf{A}_k$ over the head dimension to get the similarity of each token to each cluster. After this step CAST works as described in Section 4.2, but with an added constant dimension h. Before the result $\mathbf{O}$ is calculated, the result of the different heads R is concatenated such that $\mathbf{R} \in \mathbb{R}^{N\times d}$.

4.4 Complexity

With the use of CAST, the original quadratic complexity of the self-attention mechanism can be significantly reduced. The complexity of CAST without added constants regarding the number of layers, batch size, and hidden dimensions is $O(\alpha N)$. Here $\alpha = \max(N\kappa, NN_c^2)$, where κ is the number of elements in a cluster, and N_c the number of clusters. Here, the complexity $O(N\kappa)$ is derived from the computation of $\mathbf{R}_{intra}$ being $N_c\kappa^2$, which can be rewritten as $N\kappa$. The complexity $O(NN_c^2)$ is derived from the computation of $\mathbf{R}_{inter}$. Theoretically, the memory usage is lowest with a configuration where $N_c^2 = \kappa$, which will be tested in the ablations.

5 Experimental Setup

To answer the research questions in Section 1, we perform a multitude of experiments. To determine whether CAST can generalize well over long-and short-range machine learning tasks, we determine the performance of CAST on the Long Range Arena Benchmark[42] (LRA). Furthermore, the LRA benchmark is also used to determine the efficiency of CAST compared to that of other efficient Transformer variants. We further perform an ablation study on the surrogate tokens to determine how the amount of surrogate tokens influences the performance, peak memory usage, and the training steps per second of CAST. Lastly, we visually analyze the learned clusters to gain an insight into why CAST works.

5.1 Dataset

As shortly mentioned before we evaluate CAST on the Long Range Arena (LRA) benchmark [42], which consists of six tasks of different modalities and sequence lengths (1K-16K tokens). Together, these tasks test a model's capabilities in dealing with a diverse range of data types and structures such as natural language, images, and mathematics. The LRA benchmark is currently being used as the main benchmark for efficient Transformers and long-range sequence modeling. The evaluation metric for all the tasks in LRA is their classification accuracy. We further describe the six tasks of LRA in more detail in the following sections.

ListOps. The ListOps dataset was originally created for testing the parsing ability of latent tree models, but a larger version is now used in the LRA to test the capability of Transformers to learn the hierarchical structures. The data is a sequence of tokens representing a large mathematical operation on lists of numbers. The numbers 0 to 9 are available as both the input of the operations and the final result. There are four base mathematical operations :

- MAX: The largest value in a given list.
- MIN: The smallest value in a given list.
- MED: The median value in a given list.
- SUM MOD: The sum of the list module 10.

In the LRA the maximal length of the input sequence is set to 2K tokens. This is a ten-way classification task where accuracy is used as the evaluation metric.

Text. The *Text* task takes the IMDb reviews sentiment classification task [31] and the characters as tokens in the input sequence. The maximum length of the input sequences is truncated or padded to 4K tokens. This task is a binary classification task with accuracy as its metric.

Retrieval. For the *Retrieval* task the ACL Anthology Network dataset [37] is used. This dataset holds papers that can either be linked by citation or not, in other words, the final task is a binary classification task. To make the task more challenging, character-level tokens like in the text classification task are used in the setup. A sequence length of 4K tokens is used per document resulting in a total length of 8K tokens.

Image. The *Image* task takes the CIFAR-10 dataset [21] as its base. The images are first greyscaled into a single channel with each pixel having an 8-bit pixel intensity as its representation. This results in a 32×32 image which is unrolled into a 1-D sequence, this sequence is then used as input for a ten-way classification task.

Pathfinder. The *Pathfinder* task [25] consists of images of 32×32 where two dots, represented by circles, are connected by dashed lines. A model is required to make a binary decision of whether the two dots are connected by the dashed lines, however, there are also distraction lines that are not connected to any of the dots. Just like in the image classification task the image is unrolled into a sequence length of 1024 and used as input for this task.

Path-X. The *Path-X* task is a more extreme case of the original Pathfinder, instead of the image being 32×32 it is 128×128 making the sequence length 16 times larger than the original. Apart from the size this task is exactly the same as the original Pathfinder. It should be noted that this task has not yet been achieved with a higher-than-random accuracy with the constraints of the LRA.

5.2 Experiments

In this section, we explain the experimental setup per experiment in more detail. All experiments were implemented and run using PyTorch [36] version 1.13. The experiments were performed on a variety of hardware, and are expanded upon in more detail per experiment where significant. To keep experiments reproducible, we keep the number of layers and features comparable to those used in efficient Transformers in the original LRA paper [42].

Long Range Arena Efficiency. We evaluate the efficiency of CAST by running the CAST on the *Text* task of LRA with a varying sequence length of 1K, 2K, 3K, and 4K. For each of these sequence tasks, we determine the peak memory usage and the number of training steps per second relative to the original Transformer architecture. For comparison with other efficient Transformers we take their relative performance of the original LRA paper [42]. We ensure that CAST and the Transformer use the exact same hyperparameters, such as the number of layers, the number of heads, and the size of the feature dimensions. CAST uses a constant cluster size of 200 throughout all sequence lengths, and all experiments were run on a single A40 GPU.

Long Range Arena Performance. We evaluate the performance of CAST with both the *Top-K* and SA *Top-K* clustering mechanisms on the LRA dataset by performing a small hyperparameter sweep. In total, we ran ten full-length training sessions per task, where the checkpoint with the lowest validation loss was used to evaluate the performance of CAST. Furthermore, we also combine CAST with the current state-of-the-art, MEGA [30], to determine whether the addition of CAST would improve the state-of-the-art. In Appendix B.1, a more detailed description regarding the hyperparameters can be viewed.

Clustering Ablation. We further perform an ablation study on how the number of surrogate tokens, i.e. the number of clusters, affects the performance, the peak memory usage, and the number of training steps per second. Furthermore, we investigate whether there is a difference in using the *Top-K* or Single Assignment *Top-K* clustering mechanisms in the *Image* task. For this ablation, we use the *Text* and *Image* tasks from the LRA dataset to determine whether there is a difference between modalities. For each task, we take the best-performing models from the hyperparameter sweep but vary the cluster size κ such that $\kappa \in \{32, 64, 128, 256, 512\}$.

Visual Analysis on Clusters. Lastly, we perform a visual analysis on the learned clusters in the *Image* task of the LRA dataset. From the ablations, we take a single model with two CAST layers and eight surrogate tokens. We then visualize which tokens are clustered together and have a more in-depth look at the obtained similarity scores $\mathbf{A}_g$.

6 Results

In this Section, we show the results of our experiments. In Section 6.1 we show the memory and time efficiency of CAST compared to other architectures. Then, in Section 6.1 we compare the performance of CAST with other architectures on the Long Range Arena Benchmark. An ablation study is performed on the clustering in Section 6.3, and then a visual analysis is done on the learned clusters in the *Image* task of LRA.

6.1 Long Range Arena Efficiency

In Table 1, we compare the speed and memory efficiency of several notable architectures. We observe that CAST with

Top-K is significantly faster compared to both the original Transformer and other efficient Transformers for all sequence lengths, with it being 6.18 times faster than the original Transformer on a sequence length of 4K. Furthermore, CAST needs slightly less memory than other efficient Transformers, only needing 10% of the memory compared to the original Transformer architecture at a sequence length of 4K. The use of *SA Top-K* lowers the speed of CAST significantly but does seem to affect the memory efficiency. Furthermore, the use of CAST in combination with MEGA [30] is not more efficient nor more performant than simply chunking.

6.2 Long Range Arena Performance

Table 2 reports the performance results of CAST compared to those of the baseline Transformer and its efficient variations, and the current state-of-the-art models. CAST achieves performance between that of the state space models and the other efficient Transformers. Although structured state space models are state-of-the-art, they cannot be directly compared to other efficient Transformers since they apply global convolutions and are not solely relying on attention. CAST has a relatively high score for the *Image* task and a relatively low score for the *Pathfinder* task compared to that of the other efficient Transformers. The low score of the *Pathfinder* task could be explained by the fact that many of the pixels in the Pathfinder image are black, which makes their query-key pairs similar and put in the same cluster. Furthermore, MEGA with CAST does perform significantly better than CAST on its own, but slightly worse than MEGA-Chunk. Specifically, MEGA-CAST performs significantly worse on the image-based tasks of LRA. An extended version of Table 2 can be found in Appendix C.1.

6.3 Clustering Ablation

Here we discuss the multitude of ablation studies done on the clustering that is used in CAST. In Figure 4, a summary of the results of all ablations can be seen.

Clustering Mechanism. Figure 4 shows the difference in performance, memory footprint, and time efficiency between the clustering mechanisms *Top-K* and SA *Top-K* on the *Text* and *Image* tasks of LRA. In Figure 4d, it can be seen that the choice in the clustering mechanism slightly affects the resulting performance on the *Image* task at a cluster size of 128 and 256. It can be observed from Figure 4b and Figure 4e, that the cluster mechanism does not affect the Peak Memory Usage. Furthermore, it can be seen from Figure 4c and Figure 4f that the *Top-K* clustering mechanism is overall significantly faster than the SA *Top-K* clustering mechanism. The SA Top-K clustering mechanism in particular is much slower when using small cluster sizes on large input sequences, like for the *Text* task.

Performance. In Figure 4a and Figure 4d, we show a comparison of the effect of cluster sizes and cluster mechanism on the performance of CAST on *Text* and *Image* task of the LRA dataset. For the *Text* task, the cluster size does not significantly impact the resulting accuracy, although a slight increase in accuracy can be observed at a larger cluster size. However, cluster size does impact the performance of the *Image* task significantly for both *Top-K* and SA *Top-K*. It can be observed that the performance on the *Image* task dips

Table 1. Speed and Memory efficiency of the LRA Benchmark with the average performance (Avg.). The Transformer and CAST were created using the same hyperparameters. A batch size of 25 was used and CAST uses a constant cluster size of 200. Speed and Memory increase/decrease are reported relative to the results of the original Transformer architecture. Models annotated with the † symbol had their relative speed and memory taken from the LRA benchmark [42].

Model	Steps Per Second ↑				Peak Memory Usage ↓				Avg.
	1K	2K	3K	4K	1K	2K	3K	4K	Performance
Transformer [45]	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	57.71
Reformer† [20]	0.5	0.4	0.7	0.8	0.56	0.37	0.28	0.24	50.56
Sinkhorn Trans.† [41]	1.1	1.6	2.9	3.8	0.55	0.31	0.21	0.16	51.23
Performer† [8]	1.2	1.9	3.8	5.7	0.44	0.22	0.15	0.11	51.18
Luna-16 [29]	1.2	1.8	3.7	5.5	0.44	0.23	0.17	**0.10**	59.55
S4 [14]	-	-	-	4.8	-	-	-	0.14	86.09
MEGA [30]	-	-	-	2.9	-	-	-	0.31	88.21
MEGA-Chunk [30]	-	-	-	5.5	-	-	-	0.13	85.66
MEGA-CAST	-	-	-	3.01	-	-	-	0.21	82.11
CAST (Top-K)	**1.76**	**3.25**	**4.48**	**6.18**	**0.33**	**0.18**	**0.13**	**0.10**	59.32
CAST (SA Top-K)	1.47	2.24	2.33	2.62	**0.33**	**0.18**	**0.13**	**0.10**	57.57

Table 2. The performance of different architectures on the Long Range Arena benchmark in classification accuracy. We divide these works in (A) Transformer architectures that do not use Structured State Spaces or any derivation of this, and (B) Architectures using Structured State Spaces. (A-Top) The original Transformer architecture. (A-Middle) Efficient Transformer architectures that came out with the LRA benchmark. (A-Bottom) Notable models that came out after the release of the LRA benchmark. (B) Architectures using Structured State Spaces. here the symbol † indicates that the results came from the original paper from the LRA dataset [42]. Furthermore, the symbol × indicates that the Transformer variant either ran out of memory and – indicates that results were not reported.

Model	**Year**	**ListOps**	**Text**	**Retrieval**	**Image**	**Pathfinder**	**Path-X**	**Avg.**
Random		10.00	50.00	50.00	10.00	50.00	50.00	36.67
		(A) Transformer Based Architectures						
Transformer† [45]	2017	36.37	64.27	57.46	42.44	71.40	×	53,66
Transformer (re-impl [29])	2017	37.11	65.21	79.14	42.94	71.83	×	57.71
Local Att.† [42]	2017	15.82	52.98	53.39	41.46	66.63	×	46.71
Sparse Trans.† [7]	2019	17.07	63.58	59.59	44.24	71.71	×	51.03
Performer† [8]	2020	18.01	65.40	53.82	42.77	77.05	×	51.18
Reformer† [20]	2020	37.27	56.10	53.40	38.07	68.50	×	50.56
Sinkhorn Trans.† [41]	2020	33.67	61.20	53.83	41.23	67.45	×	51.23
BigBird† [49]	2021	36.05	64.02	59.29	40.83	74.87	×	54.18
FNet [23]	2021	35.33	65.11	59.61	38.67	77.80	×	54,42
Luna-16 [29]	2021	37.43	**65.74**	**79.38**	46.39	**78.36**	-	**59.55**
CAST Top-K (**Ours**)	2023	39.90	65.45	78.01	52.37	70.18	×	59.32
CAST SA Top-K (**Ours**)	2023	**40.70**	65.13	74.64	**52.78**	62.22	×	57.57
		(B) Structured State Space Architectures						
S4 [14]	2021	59.60	86.82	90.90	88.65	94.20	96.35	86.09
S5 [38]	2022	62.15	89.31	91.40	88.00	95.33	**98.58**	87.46
MEGA-Chunk [30]	2022	58.76	90.19	90.97	85.80	94.41	93.81	85.66
MEGA [30]	2022	**63.14**	**90.43**	**91.25**	**90.44**	**96.01**	97.98	**88.21**
MEGA-CAST (**Ours**)	2023	56.35	88.51	88.99	77.92	92.79	88.13	82.11

around a cluster size of 64 to 128, but peaks at a cluster size of 32 and 256.

Peak Memory Usage. In Figure 4b and Figure 4e, we show measurements of the influence of the cluster sizes on the peak memory usage of the *Image* and *Text* task. At its lowest, CAST only uses around 1.35 gigabytes of memory for the *Image* and 6.3 gigabytes of memory for the *Text* task. The memory curves represent a quadratic relationship, with an increase in memory when the number of clusters becomes too large, which was expected from Section 4.4. For both tasks, it can be seen that the least amount of memory is used when the number of clusters and the cluster size is close to the relation $N_c^2 = \kappa$. However, knowing that CAST achieves similar performance across different cluster sizes, we can use the cluster size that minimizes the memory footprint without a large decrease in performance.

Time Efficiency. In Figure 4c and Figure 4f, we report measurements of the influence of the cluster size and clustering mechanism on the training steps per second of the *Text* and *Image* task. It can be observed that the number of training steps per second for the SA *Top-K* clustering mechanism (orange) is significantly lower than that of the standard *Top-K* clustering mechanism, especially at smaller cluster sizes. This is due to the constraint of SA *Top-K* must ensure every token is contained only in one cluster. However, knowing that the change of performance between clustering mechanisms and cluster sizes is small, the *Top-K* clustering mechanism can be chosen at a cluster size that maximizes the number of steps per second.

6.4 Visual Analysis on Clusters

The clusters created by CAST do seem to hold visuospatial information on image tasks. More specifically, CAST seems to separate the background from the foreground in images. In Figure 5a, we show an example of an input image from the *Image* task which depicts a horse and its rider. In Figure 5b, the clustered pixels of the two layers of CAST are depicted, where each color corresponds to one of the clusters. In the first layer, we can observe that the clusters are approximately slices of the original image. In the last layer,

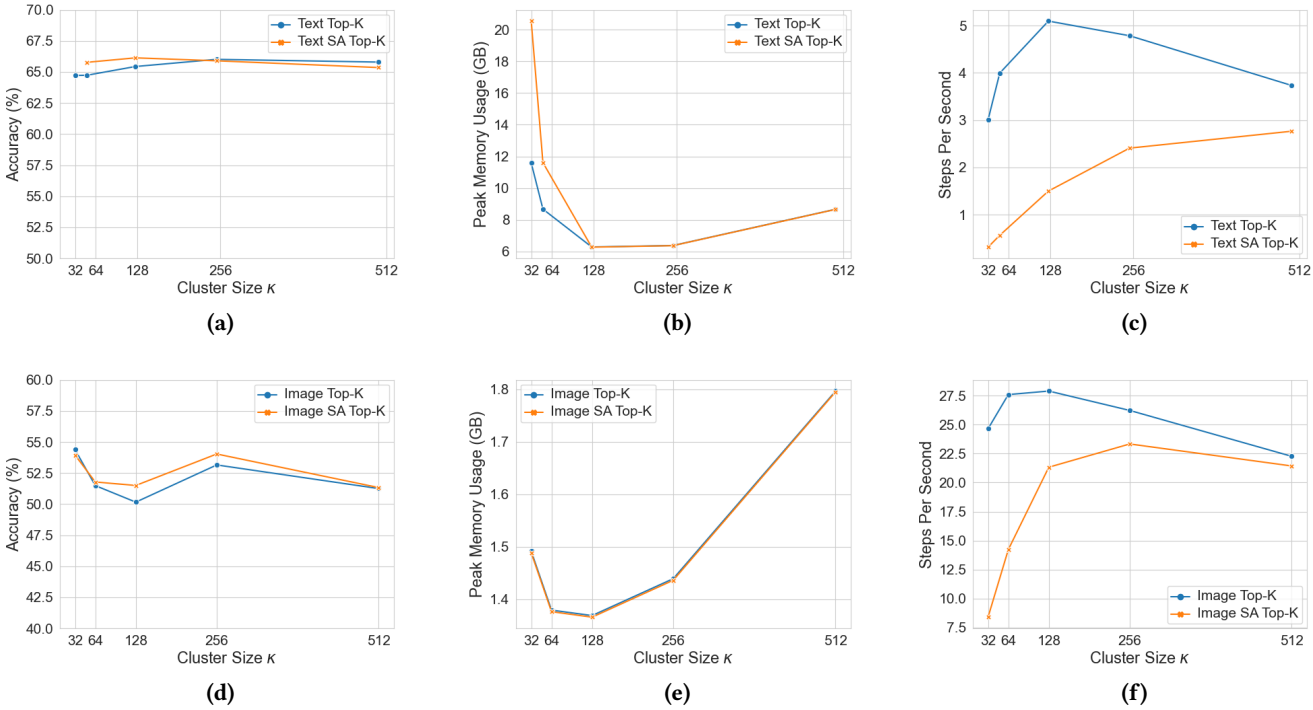


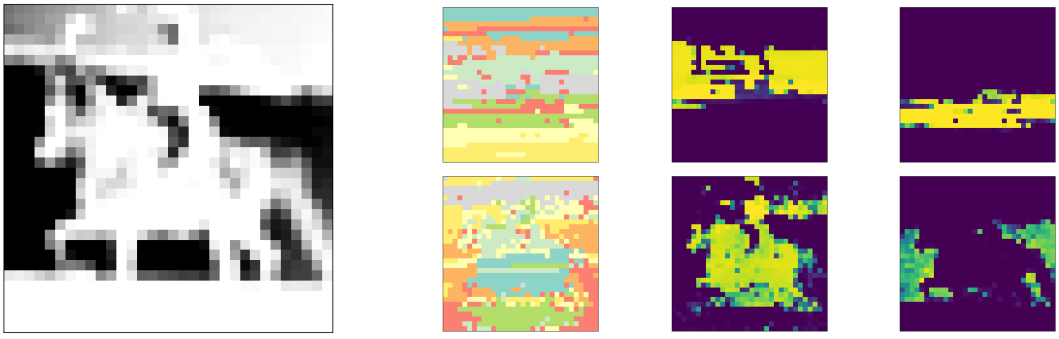
Figure 4. Ablations on the cluster size using CAST with *Top-K* Clustering Mechanism (blue) and Single Assignment *Top-K* Clustering Mechanism (orange) on the *Text* and *Image* tasks of the LRA benchmark against (a & d) the performance, (b & e) the peak memory allocated, and (c & f) the time efficiency, respectively.

it can be observed that the background and foreground of the image are roughly separated in different clusters. This behavior is observed for most of the images in the *Image* task – see Appendix C.2 for more examples. We further analyze the clusters by visualizing the scores of $\mathbf{A}_g$ in Figure 5b, where the separation of foreground and background is more evident, together with the separation per slice of the image.

7 Discussion

Here, we explicitly answer the research questions posed in Section 1, and discuss the results and limitations of CAST.

How does the memory and computational complexity of CAST compare to other efficient Transformers? CAST's memory and computational efficiency are significantly better than other efficient Transformers in the Long



(a) Example Image of the LRA **Image** task.

(b) Visualizations of learned clusters of CAST per layer.

Figure 5. Visualizations of the learned clusters of a CAST model on the LRA **Image** task. The number of clusters N_c is 8. (Left) An example of from the LRA **Image** task. (Middle) Clustered pixels, where each color represents a different cluster. (Right) Example scores for clusters in $\mathbf{A}_g$. (Top) Visualization of the first layer. (Bottom) Visualization for the last layer.

Range Arena efficiency benchmark. CAST uses less memory for all tested sequence lengths and is only slower for a sequence length of 1K. As seen in Table 1, CAST can be up to 6.18 times faster and use 10% of the memory of the original Transformer. We attribute this increase in speed and memory efficiency to the ability of CAST to work well with small cluster sizes.

When CAST is combined with the structured-state-space model MEGA does perform slightly slower than simply chunking MEGA, which is most likely due to the fact that MEGA-Chunk's chunk size is equal to CAST's cluster size, making CAST take longer and use more memory in the form of creating cluster summaries.

How does CAST generalize over both long- and short-range machine learning tasks? CAST performs on par with and often better than the original Transformer architecture on the Long Range Arena benchmark, which tests many different modalities and sequence lengths. Some interesting observations can be made regarding the performance of CAST on the LRA benchmark. Like the other clustering Transformers, CAST performs below the original Transformer on the *Pathfinder* task, we hypothesize that this is due to the fact that many of the pixels in the *Pathfinder* task are the same value. Because of this, their resulting embedding and their clustering affinities in $\mathbf{A}_g$, are the same, which could cause most of the input sequence to be clustered semi-randomly. For this same reason, we hypothesize that the performance of CAST on the *Image* task is relatively high, as the *Image* task contains natural images, which have a large variety in pixel values. The combination of MEGA with CAST does not show a significant improvement of doing chunking with MEGA, we hypothesize this is due to the Structured State Space MEGA uses, which is already modeling the long-range dependencies, making the self-attention that is applied more of a short-range dependency modeler.

How does the choice of the configuration of surrogate tokens affect the generalizability of CAST?. The configuration of surrogate tokens, specifically the number of surrogate tokens, slightly affects CAST's performance on the *Image* task, but minimally on the *Text* task. The use of SA *Top-K* makes CAST slower compared to *Top-K* as a trade-off for being slightly better in the *Image* task. The dip in performance at a cluster size of 128 could be because the cluster size is not large enough to get enough context in the intra-cluster self-attention. At the same time, there are also not enough clusters to get valuable information from the cluster summaries. Both increasing and decreasing the cluster size could fix this since an increase in cluster size gets more context into the intra-cluster self-attention, while a decrease allows for more specificity in cluster affinities. When it comes to the actual generalizability of CAST, the number of surrogate tokens only slightly affects the generalizability, but only for some tasks. We also note that the number of surrogate tokens can be tuned as a hyperparameter, meaning that a trade-off between speed and performance can be made depending on what is more important for any given task.

Visuospatial information of learned clusters. The clusters created by CAST do seem to hold visuospatial information on image tasks, more specifically, CAST seems to separate background from foreground in images. Earlier layers seem to retrieve more local information by clustering slices of the image and applying self-attention within these clusters. While later layers of CAST cluster the image based more on whether the pixels are in the foreground or background. Whether this visuospatial information is the reason for CAST's relatively high score in the *Image* task is uncertain, however, the observation that the clusters do hold some visuospatial information shows that CAST has the ability to create meaningful clusters.

8 Conclusion

We present CAST, a novel method for efficient Transformers, based on clustering self-attention through the use of surrogate tokens. It lowers the complexity of computing the self-attention in Transformers. By utilizing learnable surrogate tokens to represent distinct clustering regions, we obtain a similarity score for clustering regions and create cluster summaries that allow all tokens to access information from each cluster while maintaining a gradient for the surrogate tokens. Our experiments demonstrate that the memory and computational efficiency of CAST is significantly better than other efficient Transformers (e.g. Reformer [20], Performer [8]) in the Long Range Arena efficiency benchmark. CAST uses less memory than existing methods for all tested sequence lengths, being up to about 6× faster than and using 10% of the memory of the original Transformer.

9 Future Work

This work provides an interesting research direction for future papers. CAST can be expanded on by using different clustering mechanisms or applying asymmetric clustering, possibly resulting in an even better performance. Further speed gains could be made by parallelizing the attention within clusters and the creation of cluster summaries over multiple devices. Because CAST shows promise on the *Image* task we could also see if CAST could work as a Vision Transformer [13], this could for instance be done by training CAST on the ImageNet dataset [12]. When it comes to text-related tasks, future work could focus on creating a CAST version of cross-attention to create a Transformer with a full encoder-decoder setup. The full encoder-decoder version of CAST could then be trained for the general language understanding task (GLUE) [46], which could be used as an indicator of whether CAST could be used in large language models.

Acknowledgments

I would like to specifically thank my supervisor Estefania Talavera Martinez for sparring with me about my ideas, being patient, and helping me to push myself to finish this thesis. Furthermore, I would like to thank the following fellow students: Puru (Mei) Vaish, Aleksandra Siderova, Mauk Muller, Julia Kersten, Koen van den Brink, Inigo Artolozaga, Anna Mae van de Peut, Anniek Megens, Tim Yeung, Meng Commissaris, Ronan Oosterveen, and Marius Lupulescu for accompanying me in the EduCafe while working on this thesis, seeing them work alongside me has helped me tremendously in increasing my own productivity. Lastly, I want to specifically thank Puru (Mei) Vaish, Mauk Muller, and Koen van den Brink, for engaging in lively discussions with me about my thesis subject.

References

[1] Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. 2020. ETC: Encoding Long and Structured Inputs in Transformers. arXiv:2004.08483 [cs.LG]

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. https://doi.org/10.48550/ARXIV.1607.06450

[3] Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The Long-Document Transformer. arXiv:2004.05150 [cs.CL]

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

[5] Yen-Chun Chen, Zhe Gan, Yu Cheng, Jingzhou Liu, and Jingjing Liu. 2019. Distilling Knowledge Learned in BERT for Text Generation. https://doi.org/10.48550/ARXIV.1911.03829

[6] Lei Cheng, Ruslan Khalitov, Tong Yu, and Zhirong Yang. 2022. Classification of Long Sequential Data using Circular Dilated Convolutional Neural Networks. arXiv:2201.02143 [cs.LG]

[7] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating Long Sequences with Sparse Transformers. https://doi.org/10.48550/ARXIV.1904.10509

[8] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. 2020. Rethinking Attention with Performers. https://doi.org/10.48550/ARXIV.2009.14794

[9] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. arXiv:2205.14135 [cs.LG]

[10] Tri Dao, Daniel Y. Fu, Khaled K. Saab, Armin W. Thomas, Atri Rudra, and Christopher Ré. 2023. Hungry Hungry Hippos: Towards Language Modeling with State Space Models. arXiv:2212.14052 [cs.LG]

[11] Giannis Daras, Nikita Kitaev, Augustus Odena, and Alexandros G. Dimakis. 2020. SMYRF: Efficient Attention using Asymmetric Clustering. arXiv:2010.05315 [cs.LG]

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2020. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. https://doi.org/10.48550/ARXIV.2010.11929

[14] Albert Gu, Karan Goel, and Christopher Ré. 2022. Efficiently Modeling Long Sequences with Structured State Spaces. arXiv:2111.00396 [cs.LG]

[15] Ramin Hasani, Mathias Lechner, Tsun-Hsuan Wang, Makram Chahine, Alexander Amini, and Daniela Rus. 2022. Liquid Structural State-Space Models. arXiv:2209.12951 [cs.LG]

[16] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[17] John M. Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Zídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A A Kohl, Andy Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David A. Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. 2021. Highly accurate protein structure prediction with AlphaFold. *Nature* 596 (2021), 583 – 589.

[18] Ruslan Khalitov, Tong Yu, Lei Cheng, and Zhirong Yang. 2021. Sparse Factorization of Large Square Matrices. arXiv:2109.08184 [cs.LG]

[19] Ruslan Khalitov, Tong Yu, Lei Cheng, and Zhirong Yang. 2023. ChordMixer: A Scalable Neural Attention Model for Sequences with Different Lengths. arXiv:2206.05852 [cs.LG]

[20] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The Efficient Transformer. https://doi.org/10.48550/ARXIV.2001.04451

[21] Alex Krizhevsky. 2009. Learning Multiple Layers of Features from Tiny Images. (2009), 32–33. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[22] Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. https://doi.org/10.48550/ARXIV.1808.06226

[23] James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontanon. 2021. FNet: Mixing Tokens with Fourier Transforms. *arXiv preprint arXiv:2105.03824* (2021).

[24] Yuhong Li, Tianle Cai, Yi Zhang, Deming Chen, and Debadeepta Dey. 2022. What Makes Convolutional Models Great on Long Sequence Modeling? arXiv:2210.09298 [cs.LG]

[25] Drew Linsley, Junkyung Kim, Vijay Veerabadran, and Thomas Serre. 2018. Learning long-range spatial dependencies with horizontal gated-recurrent units. *CoRR* abs/1805.08315 (2018). arXiv:1805.08315 http://arxiv.org/abs/1805.08315

[26] Liu Liu, Zheng Qu, Zhaodong Chen, Yufei Ding, and Yuan Xie. 2021. Transformer Acceleration with Dynamic Sparse Attention. *CoRR* abs/2110.11299 (2021). arXiv:2110.11299 https://arxiv.org/abs/2110.11299

[27] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. https://doi.org/10.48550/ARXIV.2103.14030

[28] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. https://doi.org/10.48550/ARXIV.1508.04025

[29] Xuezhe Ma, Xiang Kong, Sinong Wang, Chunting Zhou, Jonathan May, Hao Ma, and Luke Zettlemoyer. 2021. Luna: Linear Unified Nested Attention. https://doi.org/10.48550/ARXIV.2106.01540

[30] Xuezhe Ma, Chunting Zhou, Xiang Kong, Junxian He, Liangke Gui, Graham Neubig, Jonathan May, and Luke Zettlemoyer. 2023. Mega:

Moving Average Equipped Gated Attention. arXiv:2209.10655 [cs.LG]
[31] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1* (Portland, Oregon) *(HLT '11)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 142–150. http://dl.acm.org/citation.cfm?id=2002472.2002491
[32] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2023. Locating and Editing Factual Associations in GPT. arXiv:2202.05262 [cs.CL]
[33] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781 [cs.CL]
[34] Derek Miller. 2019. Leveraging BERT for Extractive Text Summarization on Lectures. https://doi.org/10.48550/ARXIV.1906.04165
[35] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]
[36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
[37] Dragomir R. Radev, Pradeep Muthukrishnan, Vahed Qazvinian, and Amjad Abu-Jbara. 2013. The ACL Anthology Network Corpus. *Lang. Resour. Eval.* 47, 4 (dec 2013), 919–944. https://doi.org/10.1007/s10579-012-9211-2
[38] Jimmy T. H. Smith, Andrew Warrington, and Scott W. Linderman. 2023. Simplified State Space Layers for Sequence Modeling. arXiv:2208.04933 [cs.LG]
[39] Chi Sun, Luyao Huang, and Xipeng Qiu. 2019. Utilizing BERT for Aspect-Based Sentiment Analysis via Constructing Auxiliary Sentence. https://doi.org/10.48550/ARXIV.1903.09588
[40] Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. 2020. Synthesizer: Rethinking Self-Attention in Transformer Models. https://doi.org/10.48550/ARXIV.2005.00743
[41] Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. 2020. Sparse Sinkhorn Attention. https://doi.org/10.48550/ARXIV.2002.11296
[42] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. 2020. Long Range Arena: A Benchmark for Efficient Transformers. arXiv:2011.04006 [cs.LG]
[43] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. 2021. MLP-Mixer: An all-MLP Architecture for Vision. arXiv:2105.01601 [cs.CV]
[44] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL]
[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. arXiv:1706.03762 [cs.CL]
[46] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. arXiv:1804.07461 [cs.CL]
[47] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-Attention with Linear Complexity. arXiv:2006.04768 [cs.LG]
[48] Tong Yu, Ruslan Khalitov, Lei Cheng, and Zhirong Yang. 2022. Paramixer: Parameterizing Mixing Links in Sparse Factors Works Better than Dot-Product Self-Attention. arXiv:2204.10670 [cs.LG]
[49] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. 2021. Big Bird: Transformers for Longer Sequences. arXiv:2007.14062 [cs.LG]
[50] Hao Zheng, Zhanlei Yang, Wenju Liu, Jizhong Liang, and Yanpeng Li. 2015. Improving deep neural networks using softplus units. In *2015 International Joint Conference on Neural Networks (IJCNN)*. 1–4. https://doi.org/10.1109/IJCNN.2015.7280459

A Clustering Mechanism Pseudocode

In this section, we describe our proposed *Top-K* clustering mechanism and the SA *Top-K* clustering mechanism.

A.1 *Top-K* Clustering Mechanism

The *Top-K* clustering mechanism groups the indices with the largest similarity scores in $\mathbf{A}_g$, it allows for a token to be clustered into two clusters, but also for a token not to be clustered at all. A formal definition of the *Top-K* clustering mechanism is in Algorithm 1, where $A \in \mathbb{R}^{N \times N_c}$ is the similarity scores for each token to each cluster, and $X \in \mathbb{R}^{N \times *}$ is a matrix of feature vectors that we wish to cluster, where $*$ indicates any shape.

A.2 Single Assignment *Top-K* Clustering Mechanism

The single assignment *Top-K* clustering mechanism has the constraint that each token is assigned to only a single cluster, while also maximizing the total similarity for all clusters combined. The SA *Top-K* clustering mechanism is formally defined in Algorithm 2, where $A \in \mathbb{R}^{N \times N_c}$ represents the similarity scores for each token to each cluster, and $X \in \mathbb{R}^{N \times *}$ represents a matrix of feature vectors that we wish to cluster.

B Experiment Details

B.1 Long Range Arena Hyperparameters

For all tasks, we follow the standards given in the original Long Range Arena paper [42] regarding the data processing and task setup. For our choices in most hyperparameters, we used the current state-of-the-art, MEGA [30], as our baseline regarding the number of weight updates. Furthermore, we use their data splits regarding all tasks. The final hyperparameters used for our reported accuracy are in Table 3. For both the reported performance of *Top-K* and SA *Top-K* the same hyperparameters are used. General hyperparameters include the averaging of the output features over the sequence for the classification features, the use of linear feature embeddings for pixel tasks, the use of sinusoidal positional embeddings

Algorithm 1 Implementation of the proposed *Top-K* clustering mechanism.

Input X, A
Output C
function SA *Top-K*(A, X)
    $C = \{C_1 ... C_{N_c}\}$ ▷ Initialize result
    $I, A_{top} = \text{Top-K}(A)$ ▷ Get the indices of the largest values per cluster
    for $i \leftarrow 1$ to N_c **do**
        for $j \leftarrow 1$ to $\frac{N}{N_c}$ **do**
            $i_{token} = I_{i,j}$
            $C_i.\text{insert}(X_{i_{token}})$
        end for
    end for
end function

Algorithm 2 Implementation of the proposed Single Assignment *Top-K* clustering mechanism.

Input X, A
Output C
function SA *Top-K*(A, X)
    $A^c, I^c = sort_2(A)$ ▷ Sort from highest to lowest cluster
    $A^r, I^r = sort_1(A^c)$ ▷ Sort from highest to lowest token
    $C = \{C_1 ... C_{N_c}\}$ ▷ Initialize result
    $M = \mathbf{0}^N$ ▷ Initialize Assignment Mask
    for $i \leftarrow 1$ to N_c **do**
        for $j \leftarrow 1$ to N **do**
            $j_{token} = I^r_j$
            $i_{cluster} = I^c_{j_{token}}$
            if $M_j = 1$ **or** $\text{length}(C_{i_{cluster}}) = \frac{N}{N_c}$ **then**
                continue for loop
            end if
            $C_{i_{cluster}}.\text{insert}(X_{j_{token}})$
            $M_{j_{token}} = 1$
        end for
    end for
    return C
end function

for all tasks, and an extra normalization layer on the output features when pre-normalization is used.

C Detailed Results

In this section, we go into more depth regarding the results of CAST. We first give an extensive overview of other long-range sequence modeling architectures, and then show more examples of the visual analysis that was done on the *Image* task.

C.1 Long Range Arena Performance

In Table 4, we report a detailed list of results of different efficient Transformer variants, and other long-range sequence models on the Long Range Arena benchmark. We divide these models into the following categories: Transformer Based Architectures, Structured State Space Architectures, and Other Architectures. We can see that among the Transformer Based Architectures (A) Luna [29] and CAST similarly strong performance. When it comes to the Structured State Space Architectures (B), it can be observed that all models perform similarly, with MEGA [30] being slightly better than the rest. As for the other types of architectures, they neither use self-attention nor structured state spaces to "mix" their input sequence. Among them, the recent ChordMixer [19] stands out, ChordMixer was created for handling data with extremely long sequence lengths (in the order of 100K tokens), but has shown impressive results on the LRA benchmark too.

C.2 Further Visual Analysis

Additional visualizations of the clusters created for different samples from the **Image** task of LRA, can be seen in Figure 6

Table 3. Final hyperparameters for the best performing CAST models in our hyperparameter sweep. Here, Depth indicates the number of Transformer blocks, h the number of heads, d the number of features in the self-attention block, d_{ff}, the number of features in the feedforward block, d_{emb} the number of features in the embedding, N_c the number of clusters, Norm the type of normalization being used, BS the batch-size, LR the learning rate, WD the weight decay, and Epochs the number of epochs that were trained for.

Task	Depth	h	d	d_{ff}	d_{emb}	N_c	Norm	Pre-norm	BS	LR	WD	Epochs
ListOps	4	8	64	128	256	10	Layer	False	64	$1e^{-3}$	$1e^{-2}$	60
Text	4	4	64	128	256	20	Scale	False	25	$1e^{-3}$	$1e^{-2}$	25
Retrieval	2	8	256	256	256	20	Layer	False	8	$1e^{-2}$	$1e^{-2}$	5
Image	2	2	128	128	256	16	Batch	True	50	$5e^{-3}$	$1e^{-2}$	200
Pathfinder	2	2	32	32	64	16	Batch	True	128	$1e^{-3}$	$1e^{-2}$	200

(a horse), Figure 7 (a deer), and Figure 8 (an automobile). For each of these figures, subfigure (a) shows the original input image, (b) shows the assignment of clusters for each pixel in the first layer, and (c) shows the assignment of clusters for each pixel in the last layer. Subfigure (d) shows for each cluster the score in $\mathbf{A}_g$ that each token had for the first layer. Subfigure (e) shows for each cluster the score in $\mathbf{A}_g$ that each pixel had for the last layer.

For the mentioned sample images, it can be seen that the in the first layer, i.e. in Figures 6d, 7d, and Figure 8d, each cluster roughly clusters the same pixels. This behavior could occur, because the positional embeddings are most prominent in the first layer, causing the surrogate tokens to cluster based on this positional embedding. Furthermore, it also shows that CAST learns to cluster slices of the image first, similar to convolution. In Figures 6e, 7e and, Figure 8e, the scores in $\mathbf{A}_g$ for the last layer can be seen. This layer (e) shows more image-specific clustering. For instance, from these scores, we can observe the outline and inverse outline of a horse, a deer in a forest, and an automobile, respectively. We interpret this as the separation of background and foreground. In the case of the deer, Figure 7e, we observe a more rough outline, which can be due to the fact that the background and foreground of this image are much more similar.

D Non-performant Methods

CAST has undergone many iterations, in this section we share some ideas that did not necessarily work.

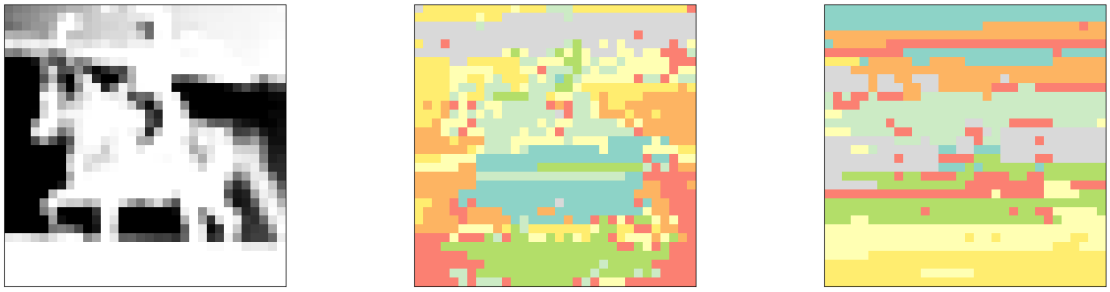
Static Surrogate Tokens without Cluster Summaries. In the first iteration of CAST, no cluster summaries were created, as such the surrogate tokens did not have a gradient and could thus not be learned. The surrogate tokens were initialized as a list of equiangular vectors to ensure that each cluster contained different types of data. However, because the surrogate tokens could not be learned this version of CAST had great instability and often suffered from mode collapse. Furthermore, using more clusters with a smaller cluster size significantly impacted performance.

Learning Surrogate Tokens through an auxiliary loss. After this, we tried to increase the stability of CAST by introducing an auxiliary loss function, which aimed to ensure that the surrogate tokens were actually in the same directions as clusters of queries and keys. The gradient of this loss function was disconnected from the queries and keys such that they were only updated based on the loss of the task. A learnable rotation matrix was used to rotate the equiangular surrogate tokens. Although this did seem to improve stability, the performance of this version was too low.

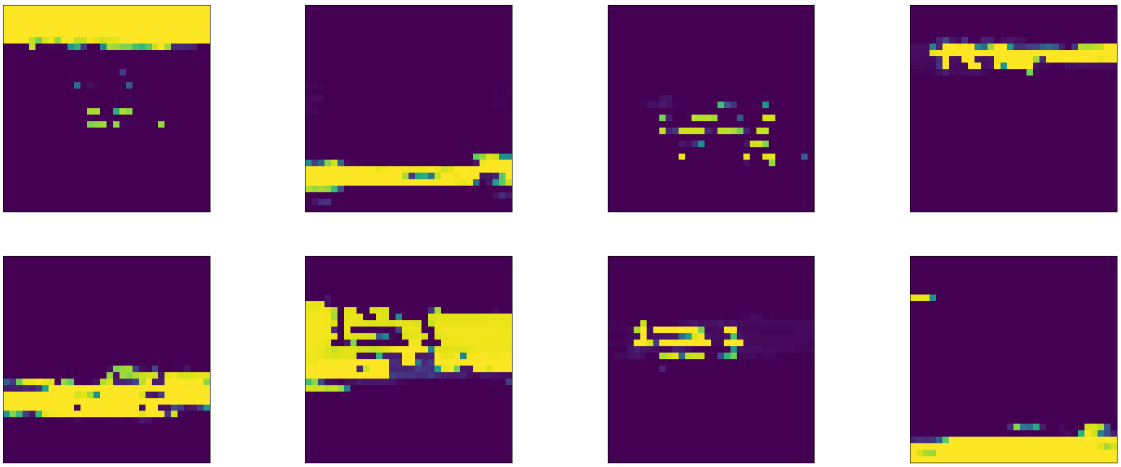
Cluster Summaries without parameterized mixing. To directly connect the surrogate tokens to the loss of the task, cluster summaries were created. The use of cluster summaries seemed to increase the performance significantly, however, mode collapse was still observed often. After this, we introduced a new transformation φ which aimed to let the architecture learn how to mix the affinity of queries and keys.

Table 4. The performance of different architectures on the Long Range Arena benchmark in classification accuracy. We divide these works into (A) Transformer architectures that do not use Structured State Spaces or any derivation of this, (B) Architectures using Structured State Spaces, and (C) Other types of architectures. The (A)-related models are grouped as; (A-Top) The original Transformer architecture, (A-Middle) efficient Transformer architectures that came out with the LRA benchmark, and (A-Bottom) notable models that came out after the release of the LRA benchmark. Here the symbol † indicates that the results came from the original paper from the LRA dataset [42]. Furthermore, the symbol × indicates that the Transformer variant either ran out of memory and – indicates that results were not reported.

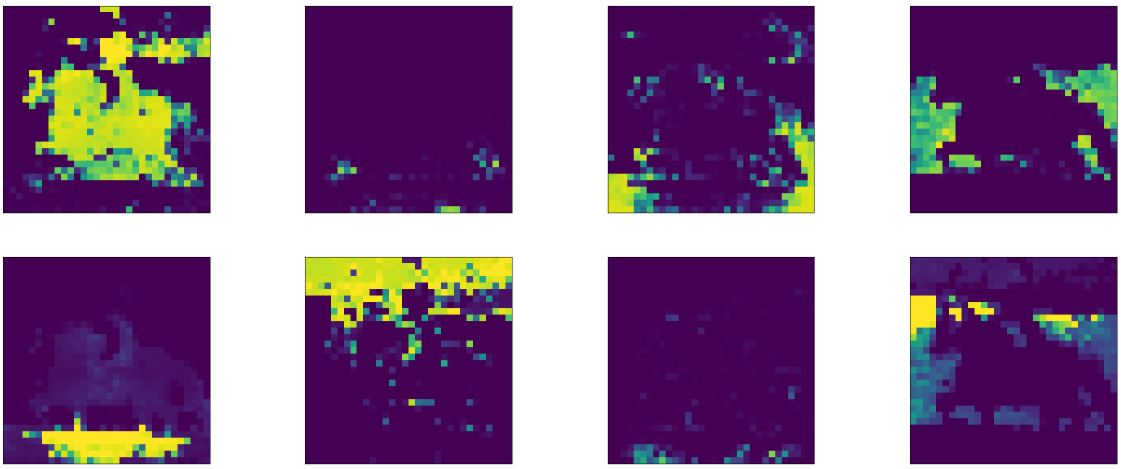
Model	**Year**	**ListOps**	**Text**	**Retrieval**	**Image**	**Pathfinder**	**Path-X**	**Avg.**
Random		10.00	50.00	50.00	10.00	50.00	50.00	36.67
		(A) Transformer Based Architectures						
Transformer† [45]	2017	36.37	64.27	57.46	42.44	71.40	×	53.66
Transformer (re-impl [29])	2017	37.11	65.21	79.14	42.94	71.83	×	57.71
Sparse Trans.† [7]	2019	17.07	63.58	59.59	44.24	71.71	×	51.03
Local Att.† [42]	2020	15.82	52.98	53.39	41.46	66.63	×	46,71
Reformer† [20]	2020	37.27	56.10	53.40	38.07	68.50	×	50.56
Sinkhorn Trans.† [41]	2020	33.67	61.20	53.83	41.23	67.45	×	51.23
Performer† [8]	2020	18.01	65.40	53.82	42.77	77.05	×	51.18
Linformer† [47]	2020	35.70	53.94	52.27	38.56	76.34	×	51.36
Longformer† [3]	2020	35.63	62.85	56.89	42.22	69.71	×	53.46
Synthesizer† [40]	2021	36.99	61.68	54.67	41.61	69.45	×	52.40
BigBird† [49]	2021	36.05	64.02	59.29	40.83	74.87	×	54.18
Luna-16 [29]	2021	37.43	65.74	**79.38**	46.39	78.36	-	**59.55**
Luna-128 [29]	2021	38.01	65.74	79.55	47.47	**78.89**	-	59,94
Luna-256 [29]	2021	37.98	**65.78**	**79.56**	47.86	78.55	-	59,96
PSF [18]	2021	38.85	77.32	-	45.01	80.49	-	56.95
FNet [23]	2021	35.33	65.11	59.61	38.67	77.80	×	54.42
CAST Top-K (**Ours**)	2023	39.90	65.45	78.01	52.37	70.18	×	59.32
CAST SA Top-K (**Ours**)	2023	**40.70**	65.13	74.64	**52.78**	62.22	×	57.57
		(B) Structured State Space Architectures						
S4 [14]	2021	59.60	86.82	90.90	88.65	94.20	96.35	86.09
H3 [10]	2022	57.50	88.20	91.00	87.30	93.00	91.80	84.80
Liquid-S4 [15]	2022	62.75	89.02	91.20	89.50	94.8	96.66	87.32
SGConv [24]	2022	61.45	89.20	91.11	87.97	95.46	97.83	87.17
S5 [38]	2022	62.15	89.31	91.40	88.00	95.33	**98.58**	87.46
MEGA-Chunk [30]	2022	58.76	90.19	90.97	85.80	94.41	93.81	85.66
MEGA [30]	2022	**63.14**	**90.43**	**91.25**	**90.44**	**96.01**	97.98	**88.21**
		(C) Other Architectures						
Paramixer [48]	2022	39.57	83.32	-	46.58	80.49	-	58.33
CDIL [6]	2022	-	87.61	84.27	64.49	91.00	-	64.56
ChordMixer [19]	2023	60.12	**88.82**	**89.98**	**90.17**	**96.69**	**98.63**	**87.40**



(a) Example Image of the LRA **Image** task. **(b)** Clustered Image, $N_c = 8$, First Layer. **(c)** Clustered Image, $N_c = 8$, Last Layer.

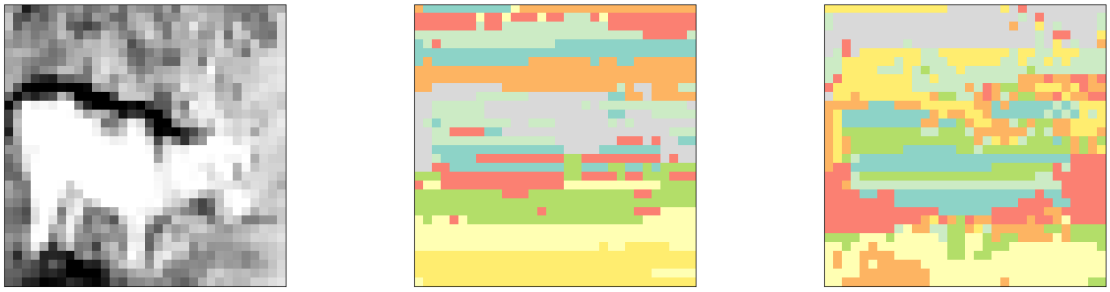


(d) Scores of $\mathbf{A}_g$ per cluster for the first layer of a CAST model trained on the **Image** task of LRA. Here, each image corresponds to a cluster, i.e. a single column of $\mathbf{A}_g$.

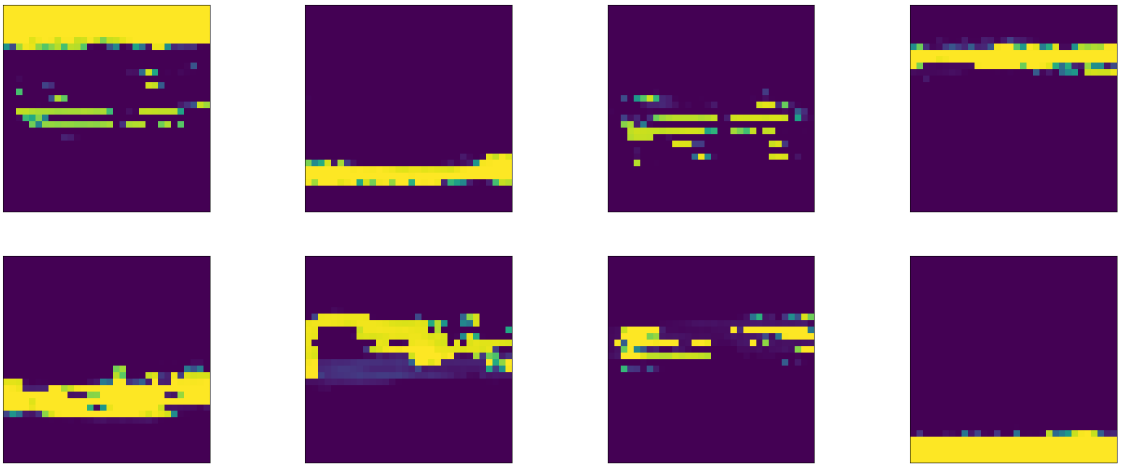


(e) Scores of $\mathbf{A}_g$ per cluster for the last layer of a CAST model trained on the **Image** task of LRA. Here, each image corresponds to a cluster, i.e. a single column of $\mathbf{A}_g$.

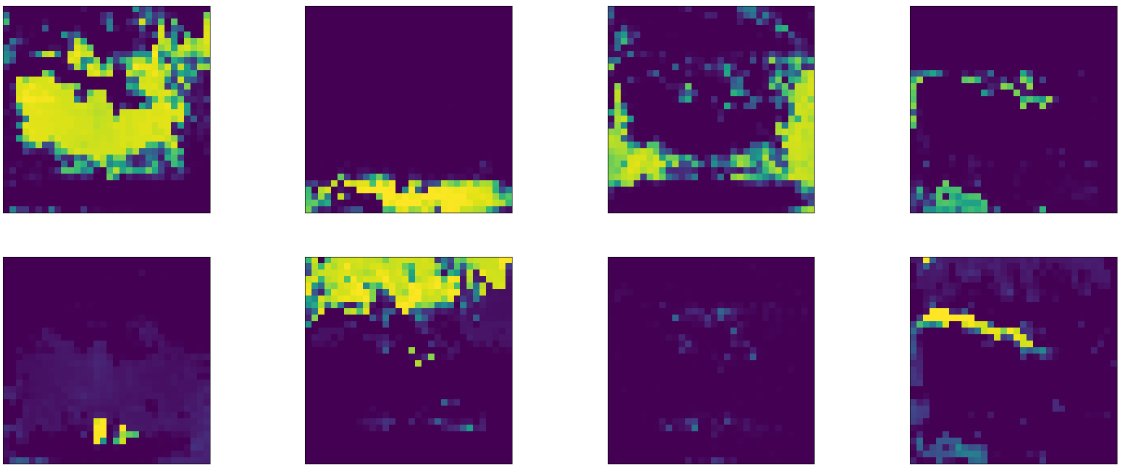
Figure 6. A visualization learned clusters in different layers of CAST. Here, (a) is a sample from the **Image** of LRA, depicting a horse with a rider, (b) is an image representing the clustered pixels in the first layer of CAST, (c) is an image representing the clustered pixels in the last layer of CAST, (d) the scores in $\mathbf{A}_g$ for every pixel in each of the eight clusters in the first layer of CAST, (e) the scores in $\mathbf{A}_g$ for every pixel in each of the eight clusters in the last layer of CAST.



(a) Example Image of the LRA **Image** task. **(b)** Clustered Image, $N_c = 8$, First Layer. **(c)** Clustered Image, $N_c = 8$, Last Layer.

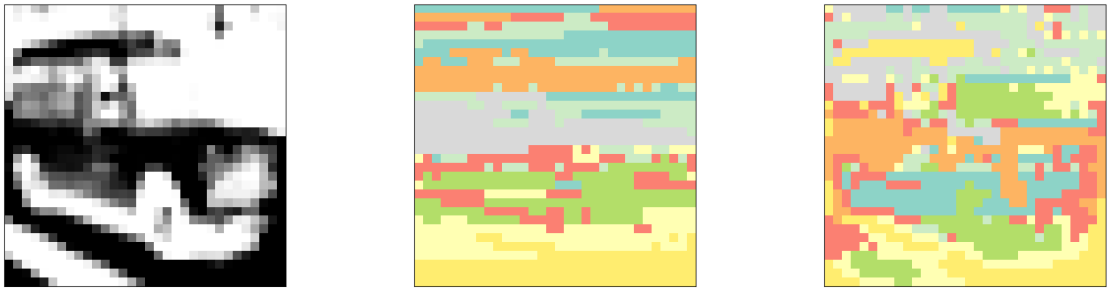


(d) Scores of $\mathbf{A}_g$ per cluster for the first layer of a CAST model trained on the **Image** task of LRA. Here, each image corresponds to a cluster, i.e. a single column of $\mathbf{A}_g$.

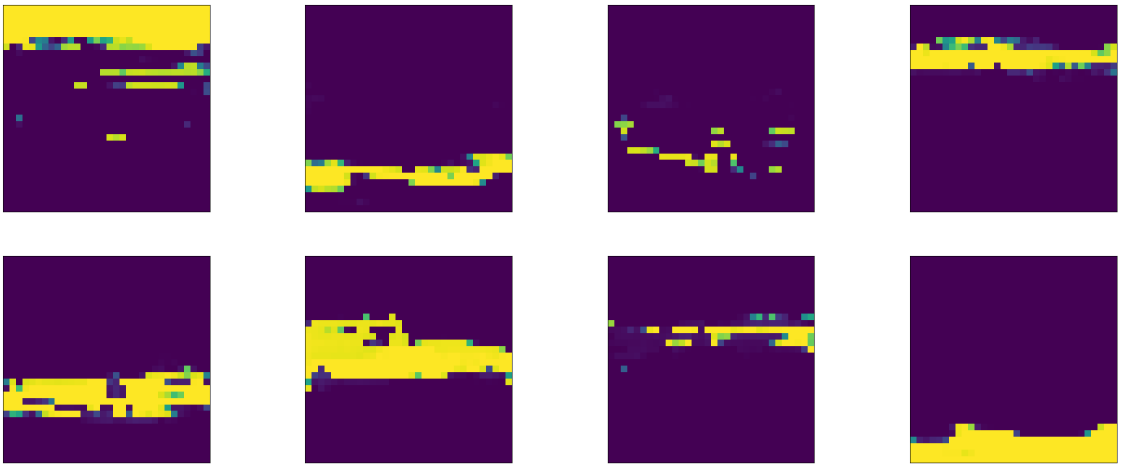


(e) Scores of $\mathbf{A}_g$ per cluster for the last layer of a CAST model trained on the **Image** task of LRA. Here, each image corresponds to a cluster, i.e. a single column of $\mathbf{A}_g$.

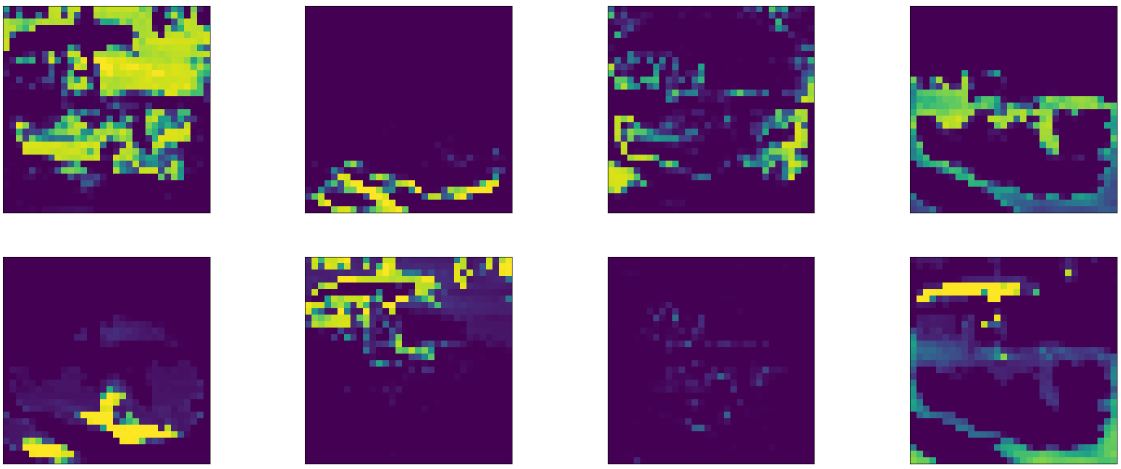
Figure 7. A visualization of the learned clusters in different layers of CAST. Here, (a) is a sample from the **Image** of LRA, depicting a deer in a forest, (b) is an image representing the clustered pixels in the first layer of CAST, (c) is an image representing the clustered pixels in the last layer of CAST, (d) the scores in $\mathbf{A}_g$ for every pixel in each of the eight clusters in the first layer of CAST, (e) the scores in $\mathbf{A}_g$ for every pixel in each of the eight clusters in the last layer of CAST.



(a) Example Image of the LRA **Image** task. **(b)** Clustered Image, $N_c = 8$, First Layer. **(c)** Clustered Image, $N_c = 8$, Last Layer.



(d) Scores of $\mathbf{A}_g$ per cluster for the first layer of a CAST model trained on the **Image** task of LRA. Here, each image corresponds to a cluster, i.e. a single column of $\mathbf{A}_g$.



(e) Scores of $\mathbf{A}_g$ per cluster for the last layer of a CAST model trained on the **Image** task of LRA. Here, each image corresponds to a cluster, i.e. a single column of $\mathbf{A}_g$.

Figure 8. A visualization learned clusters in different layers of CAST. Here, (a) is a sample from the **Image** of LRA, depicting an automobile, (b) is an image representing the clustered pixels in the first layer of CAST, (c) is an image representing the clustered pixels in the last layer of CAST, (d) the scores in $\mathbf{A}_g$ for every pixel in each of the eight clusters in the first layer of CAST, (e) the scores in $\mathbf{A}_g$ for every pixel in each of the eight clusters in the last layer of CAST.