Holographic Global Convolutional Networks for Long-Range Prediction Tasks in Malware Detection

Mohammad Mahmudul Alam[1] **Edward Raff**[1,2] **Stella Biderman**[2]

Tim Oates[1] **James Holt**[3]

[1] University of Maryland, Baltimore County [2] Booz Allen Hamilton [3] Laboratory for Physical Sciences

Abstract

Malware detection is an interesting and valuable domain to work in because it has significant real-world impact and unique machine-learning challenges. We investigate existing long-range techniques and benchmarks and find that they're not very suitable in this problem area. In this paper, we introduce Holographic Global Convolutional Networks (HGConv) that utilize the properties of Holographic Reduced Representations (HRR) to encode and decode features from sequence elements. Unlike other global convolutional methods, our method does not require any intricate kernel computation or crafted kernel design. HGConv kernels are defined as simple parameters learned through backpropagation. The proposed method has achieved new SOTA results on Microsoft Malware Classification Challenge, Drebin, and EMBER malware benchmarks. With log-linear complexity in sequence length, the empirical results demonstrate substantially faster run-time by HGConv compared to other methods achieving far more efficient scaling even with sequence length $\geq 100,000$.

1 Introduction

Ever since the transformer (Vaswani et al., 2017) revolutionized natural language processing research (Brown et al., 2020; Devlin et al., 2018; Raffel et al., 2020), significant attention has been paid to the quadratic cost of increasing sequence length. While traditional academic benchmarks tend to not require sequence lengths beyond 4096, many real-world applications such as multi-round chat (Team, 2023; Yao et al., 2023), biological sequence modeling (Ahdritz et al., 2022; Avsec et al., 2021; Dalla-Torre et al., 2023; Jumper et al., 2020; Lin et al., 2022), and analyzing computer programs (Alam et al., 2023a; Muennighoff et al., 2023; Rozière et al., 2023) do. The unique challenges, data, and sequence dynamics that occur within each application can have a significant effect on what techniques work well, which is not well elucidated within the current Transformer literature.

In this paper we are concerned with malware classification using byte-level representations of executables (Raff and Nicholas, 2017b), a task that can require sequence lengths of up to 200 million in common real-world scenarios. Though we are not able to process this extreme length in its entirety, we focus on it as an important research direction to test and develop algorithms for long-sequence task modeling. In particular, we find that some popular benchmarks from natural language processing are not well correlated with improvement in malware detection tasks. Thus, we find it necessary to develop new architectures, which we do by incorporating aspects of classical neuro-symbolic methods like the Holographic Reduced Representation (HRR) (Plate, 1995).

1.1 Malware Detection

Two predominant types of malware detection tasks exist: distinguishing malicious programs from benign and distinguishing a known malicious file into unique families of malware. Both of these tasks are relevant to real-world cyber security and are complicated by the long-range interactions, spatial and non-spatial locality, exhibited within binary sequences (Raff and Nicholas, 2020). Because ML algorithms can not usually handle more than a few thousand tokens of sequence length, the field has relied heavily on man-

Proceedings of the 27th International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238. Copyright 2024 by the author(s).

ually designed hash functions (Botacin et al., 2021; Breitinger et al., 2013; Lillis et al., 2017; Oliver et al., 2013; Raff and Nicholas, 2018b; Roussev, 2009; Winter et al., 2013). In this work we will push deep learning-based sequence modeling to over 100,000 tokens, and longer sequences will be truncated down. Though this does not yet reach the full possible sequence length, it serves as a real-world task to determine the efficacy of our methods.

1.2 Efficient Transformer-Based Models

The quadratic cost of attention has motivated substantial research into more efficient architectures that maintain the performance of transformers. For smaller-scale models, there are a wide variety of such architectures (Choromanski et al., 2020; Katharopoulos et al., 2020; Ma et al., 2021; Wang et al., 2020; Zaheer et al., 2020), however they are limited by their inability to be scaled and match the performance of traditional transformers.

Another approach to the quadratic run-time of attention that's gained popularity lately has been to simply pay it. Newer kernels for attention are reasonably fast in practice (Dao, 2023; Dao et al., 2022) and new techniques for extending context length during post-training (Chen et al., 2023; Peng et al., 2023b; Rozière et al., 2023). However the expense of such models is impractical for many applications, as while they substantially decrease the costs associated with training long-context models they do not substantially decrease the memory overhead at inference time. This is essential because for most applications the primary bottleneck is GPU VRAM and not raw computing power.

1.3 Non-Transformer Models for Sequences

Recent research has also raised the prospect of alternatives to the transformer architecture for sequence-based tasks. Foremost among these are *state-space models*, S4 (Gu et al., 2021) and its variants (Gu et al., 2020; Li et al., 2022; Poli et al., 2023) which have achieved impressive performance on language and vision tasks. Previous work in malware detection has independently developed larger-width convolutions on the order of 128-256 wide kernels, followed by temporal pooling (Raff et al., 2021; Raff and Nicholas, 2017b)

Simultaneously with this work, non-transformer architectures with more efficient inference-time context length scaling have begun to match the performance of transformers on natural language tasks(Gu and Dao, 2023; Peng et al., 2023a) and pose an interesting area of exploration for future work in malware detection and other long-sequence problems.

1.4 Our Contributions

Our primary contributions are as follows:

1. We introduce HGConv, a novel fusion of previous architectures (Li et al., 2022; Plate, 1995) that achieves state-of-the-art performance on three standard malware classification benchmarks, and furthermore achieves its excellent performance with lower inter-run variance.

2. We introduce novel algorithmic optimizations that enable HGConv to run substantially faster and with lower memory overhead than other global convolutional models.

3. We show that the widely used Long Range Arena (LRA) (Tay et al., 2020) benchmark is a poor proxy for performance at malware classification, despite the fact that it is a task that requires reasoning about long contexts. This underlines the need for using domain-specific benchmarks whose construct validity has been validated in the real world instead of "general performance" benchmarks.

2 Methodology

In convolution, inputs are convolved with kernels or filters. Recent works have demonstrated the potential of global convolution in sequence modeling yet intricate kernel computation requires custom CUDA extensions to run (Gu et al., 2021) or crafted kernel design trying to make an approximation of the S4 kernel for each task (Li et al., 2022). In this paper, we focus on building a neuro-symbolic mechanism where kernels are defined as parameters and learned through auto-differentiation eliminating the necessity of intricate and detailed computations and task-specific kernel design. Before going over the details of the proposed HGConv, first we will give a brief overview of the HRR, and its properties, then the proposed method will be elaborated and finally, the algorithmic complexity will be delineated. Our implementation can be found at `https://github.com/FutureComputing4AI/HGConv`.

2.1 Holographic Reduced Representations

Holographic Reduced Representations (HRR) is a type of vector symbolic architecture (VSA) that represents compositional structure using circular convolution in distributed representations (Plate, 1995). In HRR, vector representations of properties and values can be combined together using circular convolution, and has been successfully used in recent literature (Alam et al., 2023b, 2022; Menet et al., 2023).

For instance, the color and shape of a red circle can be stored in a compressed representation using *binding* operation ($\circledhash$) and additive properties of HRR by simply $\mathbf{b} = \textit{color} \circledhash \textit{red} + \textit{shape} \circledhash \textit{circle}$. Here the abstract concepts "color", "red", "shape", and "circle" are arbitrarily assigned to a d dimensional vector. The method of retrieving knowledge from this compressed representation is known as *unbinding* which is similar to binding operation with the inverse of a vector representation. Given vectors x_i, y_i of dimension d, the binding operation is defined in Equation 1.

$$\mathbb{B} = x_i \circledhash y_i = \mathcal{F}^{-1}(\mathcal{F}(x_i) \odot \mathcal{F}(y_i)) \tag{1}$$

Here, $\mathcal{F}(\cdot)$ and $\mathcal{F}^{-1}(\cdot)$ refer to Fast Fourier Transform (FFT) and its inverse, respectively. To retrieve x_i component from bound representation $\mathbb{B}$, the same binding operation is performed with the inverse of the y_i vector component defined in Equation 2.

$$y_i^{\dagger} = \mathcal{F}^{-1}(\frac{1}{\mathcal{F}(y_i)}) \tag{2}$$

To extract the shape of the object in our example from $\mathbf{b}$, the unbinding operation is performed as $\mathbf{b} \circledhash \textit{shape}^{\dagger} \approx \textit{circle}$. Similarly, the same concept can be utilized to encode features by binding and decode by unbinding.

2.2 Holographic Global Convolutional Networks

We will learn both sequence-wise and depth-wise by integrating binding, global circular convolution, and unbinding operations subsequently. The filters for all the operations will be defined as parameters. First, the binding will be applied along the features which will encode kernel features with input features. Next, a global convolution will be applied along the elements of the sequence which will inter-mix the features of each sequence element. Finally, unbinding will decode the necessary useful features for learning. Since the binding step encodes the filter features to the input features, it will be denoted as the Encoder (E). For conciseness, circular convolution will be referred to as Conv or Convolution (C) unless otherwise specified, and likewise, the unbinding step will be deemed the Decoder (D).

Given an input sequence of i-th layer $\mathbf{X}_i \in \mathbb{R}^{T \times H}$ has T tokens each having H dimensional features, we will define three filter weights $\mathbf{W}_i^E \in \mathbb{R}^H, \mathbf{W}_i^C \in \mathbb{R}^{K \times H}, \mathbf{W}_i^D \in \mathbb{R}^H$ for encoder, convolution, and decoder, respectively where K is the kernel dimension and $K \leq T$. $\mathbf{W}_i^C$ will be padded by zero up to maximum sequence length T to perform global convolution.

The input features are encoded with encoder filter $\mathbf{W}_i^E$ using binding given in Equation 3 and Equation 4 [1]. Here, each m-th element of feature vector y_n of $\mathbf{Y}_i$ is a linear combination of features where $\forall\ n, m \in \mathbb{N} : 0 \leq n \leq T-1, 0 \leq m \leq H-1$. The encoder step does not mix or alter the sequence elements. The sole attention is put on feature learning.

$$\mathbf{Y}_i = \mathbf{X}_i \circledhash \mathbf{W}_i^E \in \mathbb{R}^{T \times H} \tag{3}$$

$$y_n[m] = \sum_{j=0}^{H-1} x_n[j]\ w_n^e[((m-j))_H] \tag{4}$$

After learning the features, the encoded features of each element are mixed with weighted input features, i.e., kernel $\mathbf{W}_i^C$ using convolution given in Equation 5. Each feature vector $h[n]$ of the convolution layer is a linear weighted combination of encoded features of the tokens expressed in Equation 6 and Equation 7. To include a bias term, a weight $\mathbf{W}_i^B \in \mathbb{R}^H$ is defined which is elementwise multiplied with $\mathbf{Y}_i$ is added, and consecutively a *gelu* (Hendrycks and Gimpel, 2016) is applied.

$$\mathbf{H}_i = \mathbf{Y}_i \circledast \mathbf{W}_i^C + \mathbf{Y}_i \otimes \mathbf{W}_i^B \in \mathbb{R}^{T \times H} \tag{5}$$

$$\mathbf{Y}_i \circledast \mathbf{W}_i^C \mathbf{:}\ h[n] = \sum_{j=0}^{T-1} y[j]\ w^c[((n-j))_T] \tag{6}$$

$$\begin{gathered} h[n] = y_0 w_n^c + y_1 w_{n-1}^c + \cdots + y_n w_0^c + \\ y_{n+1} w_{T-1}^c + y_{n+2} w_{T-2}^c + \cdots + y_{T-1} w_{n+1}^c \end{gathered} \tag{7}$$

Since unbinding can extract information from the added feature vectors, it will be utilized to decode useful features from the convolutional step. Given that features are mixed regardless of their significance, by learning appropriate kernels, the most important features can be extracted using unbinding. Specifically, the unbinding step is expected to learn to get rid of overmixed or unnecessarily mixed element features.

$$\mathbf{Z}_i = \mathbf{H}_i \circledhash \mathbf{W}_i^{D^{\dagger}} \in \mathbb{R}^{T \times H} \tag{8}$$

$$z_n[m] = \sum_{j=0}^{H-1} h_n[j]\ w_n^d[((m-j))_H] \tag{9}$$

The extracted features are processed by a gated linear unit (GLU) (Dauphin et al., 2017) given in Equation 10 and subsequently a dropout layer is used. $\mathbf{W}_i^{\alpha}$ and $\mathbf{W}_i^{\beta}$ are the weights are GLU unit and σ is the sigmoid activation.

$$\mathbf{G}_i = \mathbf{W}_i^{\alpha}\ \mathbf{Z}_i\ \odot\ \sigma(\mathbf{W}_i^{\beta}\ \mathbf{Z}_i) \tag{10}$$

[1] `Notations:`
$\circledhash$ $\rightarrow$ `binding ops`
$\circledast$ $\rightarrow$ `circular convolution`
$\odot$ $\rightarrow$ `elementwise multiplication`

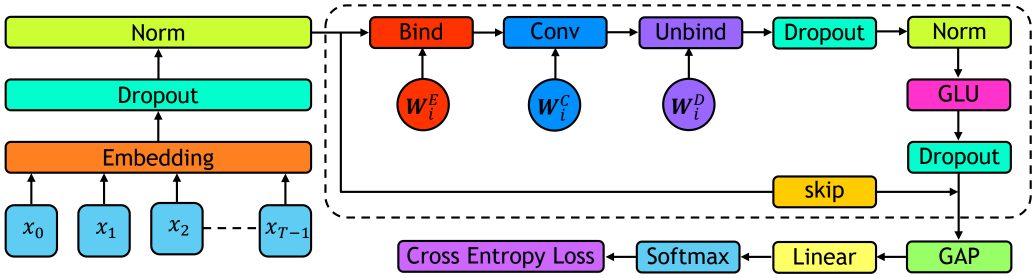


Figure 1: The block diagram of the proposed method. The dotted region shows a single layer of the proposed network which is repeated N times. In the figure, *prenorm* is applied. In the case of *postnorm*, normalization is applied after the GLU layer before the skip connection.

Finally, a skip connection is used by adding the unperturbed input $\mathbf{X}_i$ to the processed feature from GLU unit $\mathbf{G}_i$. The output of the i-th layer $\mathbf{X}_{i+1}$ can be fed to the next layer the process can be repeated N times to extract the deeper features by the combinations of `bind` → `conv` → `unbind` → `glu` units in each layer to improve the performance of the network.

$$\mathbf{X}_{i+1} = \mathbf{G}_i + \mathbf{X}_i \tag{11}$$

A generic block diagram of the proposed method is presented in Figure 1. In the embedding layer, both word and position embeddings are used and added together. For normalized floating point inputs, a linear layer is used in place of word embedding. In the norm layer, either layer normalization (Ba et al., 2016) or batch normalization (Ioffe and Szegedy, 2015) can be employed. The global average pooling (GAP) is applied to the output of the N-th layer which is subsequently fed to a linear layer with a feature size the same as the number of classes. The loss is calculated using the softmax cross-entropy loss function which is optimized using the Adam optimizer where a cosine decay learning rate scheduler with warmup is employed.

2.3 Algorithmic Complexity

The time complexity of the main three layers, i.e., binding, convolution, and unbinding are $\mathcal{O}(T \cdot H \log H)$, $\mathcal{O}(T \log T \cdot H)$, and $\mathcal{O}(T \cdot H \log H)$, respectively. Therefore, the overall time complexity is $\mathcal{O}(T \log T)$ log-linear with respect to the sequence length T. Since in all the layers, the shape of the tensors is $T \times H$, the space complexity is $\mathcal{O}(T)$ linear. Feature dimension H is assumed to be constant. A step-by-step breakdown of the time and space complexity is given in Equation 12 and Equation 13.

$$\begin{aligned}
&\textsc{Time Complexity}\\
&= \mathcal{O}(T \cdot H \log H + T \log T \cdot H + T \cdot H \log H)\\
&= \mathcal{O}(2 \times T \cdot H \log H + T \log T \cdot H)\\
&= \mathcal{O}(T + T \log T) \ \texttt{[H is constant]}\\
&= \mathcal{O}(T \cdot \{1 + \log T\})\\
&= \mathcal{O}(T \log T) \ \texttt{log-linear}
\end{aligned} \tag{12}$$

$$\begin{aligned}
&\textsc{Space Complexity}\\
&= \mathcal{O}(T \cdot H)\\
&= \mathcal{O}(T) \ \texttt{linear [H is constant]}
\end{aligned} \tag{13}$$

3 Experiments and Results

In this paper, we are proposing a neuro-symbolic method of sequence processing that encode feature, convolve along all the sequence elements, and finally decode necessary features compensating for overmixing. To validate the proposed method, experiments are performed focusing on practical applications where long sequences are a common phenomenon such as malware classification where sequence length can reach up to ≈ 200M. In our experiments, we will adopt well-known malware classification benchmarks such as the Microsoft Windows Malware benchmark that comes from the 2015 Kaggle competition (Panconesi et al., 2015), Android application packages (APK) Malware benchmark from Drebin dataset (Arp et al., 2014), and EMBER malware classification benchmark (Anderson and Roth, 2018). As will be seen in the results, in most cases existing hash-based algorithms that have no learning phase outperform existing Transformer and similar long-sequence learning algorithms.

Kaggle Microsoft Malware Classification Challenge (BIG 2015) hosted on Kaggle (Panconesi et al., 2015) is a benchmark of 9 Windows malware families. The dataset contains $10,868$ samples total uncompressed size of 184 GB which is split into train and test set by

80−20 ratio per class by random sampling. Each of the data samples comes in two different forms, in one form it is the raw binary of the original executables referred to as Kaggle Raw of size 47 GB, and in another form, it is the human-readable assembly referred to as Kaggle Asm of size 137 GB. Asm files are generated by IDA Pro which contains additional features that seem to make it easier to learn. However, it is also ≈ 3× larger with longer sequence lengths than Raw files, thus, balancing the difficulty of the dataset.

Drebin Android APK namely Drebin (Arp et al., 2014) is a benchmark of 178 malware families containing $5,560$ samples total uncompressed size of 16 GB. Nevertheless, 70% of the families contains less than 10 samples and 88.8% of the families contains less than 40 samples. Therefore, to be able to learn from enough data, in our experiments we have utilized top 20 malware families containing $4,664$ samples of size 14 GB which is split into train and test set by $80-20$ ratio per class. The original data of the dataset is in APK format which is referred to as Drebin Apk of size 6 GB. Like Kaggle, another version of the dataset is built by converting the APK files to uncompressed TAR files which have a size of 8 GB and are referred to as Drebin Tar. Since the difference between the samples is the amount of compression, it will be useful to understand how compression is handled by each algorithm.

EMBER EMBER is binary malware classification benchmark (Anderson and Roth, 2018) containing $800K$ samples of Windows executable files of total 1.02 TB of size. Among them, the training split contains $300K$ benign and $300K$ malicious files of a total size of 826 GB. On the other hand, the test split contains $100K$ benign and $100K$ malicious files of a total size of 220 GB. Although the sequence length of the files in the EMBER dataset can be over $100M$ long which is not practical to process by any sequence model, we start our experiments with a relatively shorter length of 256 (2^8) which is exponentially incremented until $131,072$ (2^{17}). Since most of the important features are encoded at the beginning of the sequence, we could not see any benefit of using a much longer sequence length than 2^{17}.

3.1 Training

The sequences of inputs are padded or truncated up to the maximum sequence length to train the proposed HGConv network. To suppress the embedding for the padded tokens binary mask is produced and multiplied by the embedding matrix. In the convolutional step, the kernel dimension K can be smaller than the actual sequence length which is also padded with zeros up to the maximum sequence length T to perform FFT convolution. Since all the tasks are essentially classification, to train the network, the softmax cross-entropy loss function is employed which is optimized using the Adam optimizer with cosine scheduler learning rate. Moreover, label smoothing is applied with a smoothing factor $\alpha = 0.1$. The hyperparameter used in each of the tasks is fine-tuned and optimized. The list of the hyperparameters used in each task is presented in Appendix A. The training is performed on a single node 16 NVIDIA TESLA PH402 32GB GPU machine where the mean of the gradient from each machine is used to update the parameters.

3.2 Evaluations

To evaluate the performance, the proposed HGConv is compared with other state-of-the-art (SOTA) sequence models. For Kaggle and Drebin datasets, the proposed method is compared with non-attention-based processors such as Lempel-Ziv Jaccard Distance (LZJD) (Edward Raff et al., 2019; Raff and Nicholas, 2018a, 2017a), Stochastic Hashed Weighted Lempel-Ziv (SHWeL) (Raff and Nicholas, 2017b), attention-based processors such as Transformer (Vaswani et al., 2017), Performer (Choromanski et al., 2020), Hrrformer (Alam et al., 2023a), and state space model based processors S4 (Gu et al., 2021) and SGConv (Li et al., 2022). Other compression-based methods like Burrows-Wheeler Markov Distance (BWMD) (Raff et al., 2020) and Lempel-Ziv Networks (Saul et al., 2023) were not considered due to lower accuracy compared to the selected baselines, and their other benefits are not a focus of this work. Table 1 shows the mean accuracy with standard deviation for 10-fold cross-validation for each of the methods. Among all the methods, the proposed HGConv achieved the best results for all the datasets with the smallest standard deviation. It is also the only method to out-perform the existing hash-based approaches, showing how existing methods did not adequately learn from long sequence problems. In terms of fluctuation among the models, the variation in the results of Drebin Apk is the most noticeable. Figure 3 shows the UMAP 3D representation (McInnes et al., 2018; Nolet et al., 2021) of the output of the penultimate layer of all the models which reveals the clustering patterns. HGConv has visibly better clusters which makes the final layer classifier predict correctly. Moreover, qualitative inspection shows that models that perform better generally show clearer and better separated clusters, with HGConv in particular showing the best clustering behavior.

EMBER is a benchmark with very long sequences. In our experiments, we started with a moderate sequence

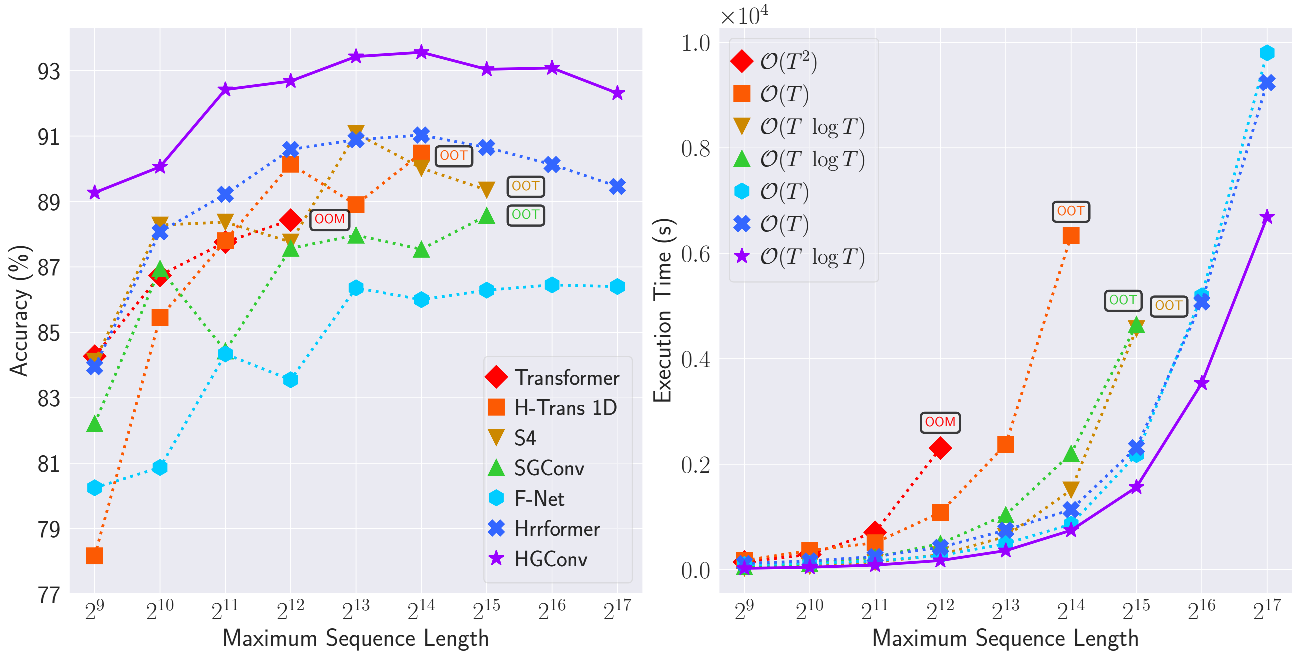


Figure 2: Ember long sequence malware classification results. In the figure, OOT and OOM stand for out-of-time and memory shown for models that face such issues after a particular sequence length. The figure shows a shorter comparison. A broader comparison with additional models Linformer (Wang et al., 2020), Performer (Choromanski et al., 2020), and F-Net (Lee-Thorp et al., 2021) and numeric results are presented in Appendix C.

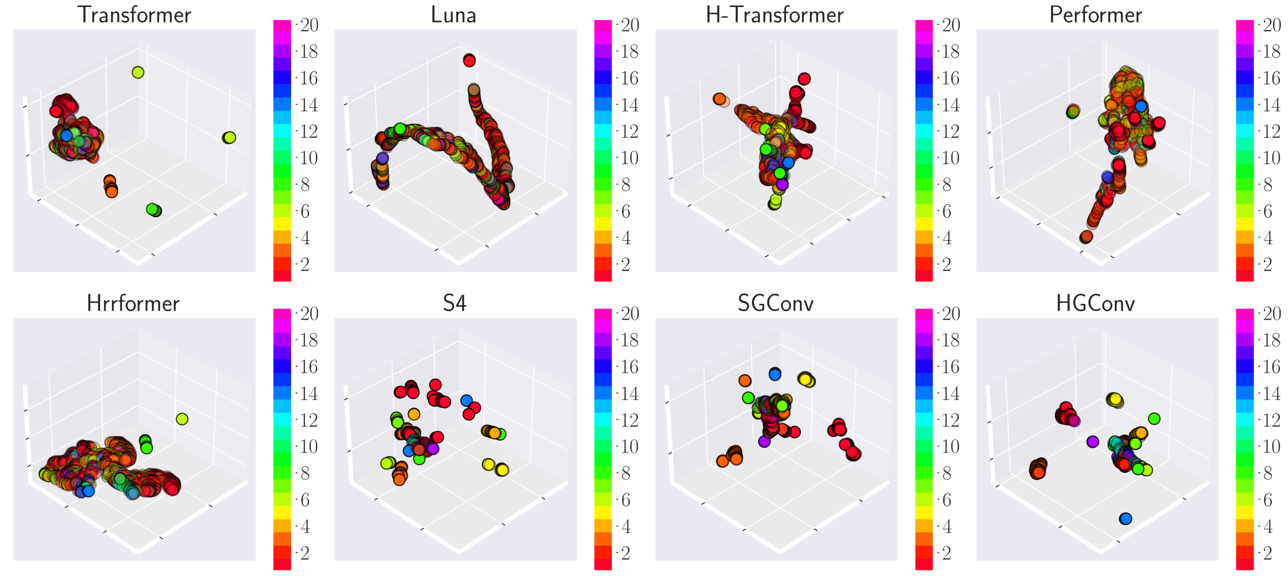


Figure 3: Drebin Apk dataset in the benchmark has the most variation in the results across the models. The figure shows the UMAP 3D representation of the output from the penultimate layer of all the models for Drebin Apk. The better the clusters the higher the accuracy.

Table 1: Results of 10-fold cross-validation on Kaggle Microsoft Malware Classification Challenge and Drebin Android Malware classification. Values inside the parenthesis are standard deviations. Also, for both of the datasets, the training time per epoch is provided in seconds.

Model	Kaggle Raw	Kaggle Asm	Kaggle Time	Drebin Apk	Drebin Tar	Drebin Time
LZJD (Raff and Nicholas, 2017a)	97.6 (1.50)	97.1 (6.10)	–	80.8 (2.60)	81.0 (6.50)	–
1NN-SHWeL (Raff and Nicholas, 2017b)	97.6 (1.38)	97.3 (1.93)	–	83.6 (1.94)	87.9 (1.84)	–
LR-SHWeL (Raff and Nicholas, 2017b)	96.7 (2.07)	96.9 (2.08)	–	78.4 (2.26)	89.1 (2.29)	–
Transformer (Vaswani et al., 2017)	72.68 (3.77)	95.60 (1.52)	31.55	40.13 (6.11)	69.50 (2.67)	15.90
F-Net (Lee-Thorp et al., 2021)	93.17 (1.08)	95.74 (1.03)	6.54	69.41 (1.81)	80.98 (1.22)	4.73
Luna-256 (Ma et al., 2021)	89.50 (0.89)	93.47 (0.96)	26.19	24.30 (2.36)	56.42 (7.90)	16.49
H-Transformer (Zhu and Soricut, 2021)	92.78 (0.49)	98.07 (0.29)	117.53	71.85 (0.84)	87.40 (0.70)	99.64
Performer (Choromanski et al., 2020)	94.63 (0.79)	97.66 (0.48)	37.08	70.44 (3.65)	82.38 (1.12)	18.31
Hrrformer (Alam et al., 2023a)	94.41 (0.57)	98.52 (0.23)	7.35	57.28 (3.80)	84.07 (1.03)	5.42
S4 (Gu et al., 2021)	96.44 (0.41)	98.66 (0.32)	17.51	88.38 (1.69)	87.94 (1.05)	14.97
SGConv (Li et al., 2022)	95.13 (0.91)	98.12 (0.56)	24.37	76.23 (3.14)	80.04 (4.33)	24.37
HGConv	**98.86 (0.12)**	**99.63 (0.14)**	**5.86**	**90.15 (0.47)**	**91.86 (0.35)**	**3.63**

length of 256 (2^8) and incremented up to $131,072$ (2^{17}) and computed the accuracy and execution time for each sequence length which is presented in Figure 2. For practical reasons, we have set a maximum limit of $10,000$ seconds per epoch. If a method takes more than that is marked as out-of-time (OOT). If the model and data can not be put onto the memory for a particular sequence length that is marked as out-of-memory (OOM) in the figure. HGConv not only achieves the best accuracy but also takes the least amount of time among all the compared methods. The full comparison and all the numerical results are presented in Appendix C. We also find that HGConv runs substantially faster than all other methods, achieving far more efficient scaling despite the increased theoretical complexity compared to Hrrformer and F-Net.

4 Long Range Arena Does Not Predict EMBER Reliably

Recent work on benchmarking large language models (Gao et al., 2021; Raji et al., 2021) has questioned the construct validity of the widespread practice of assuming that "diverse" acontextual benchmarks are indicative of performance on tasks of interest. For long-context models, this is exemplified by the widespread use of the Long Range Arena (Tay et al., 2020), which contains tasks that evaluate parsing long expressions, classifying movie reviews, assessing text similarity, classifying flattened CIFAR-10 images, and identifying if two points are connected by a long path. Despite the lack of relevance of these tasks to their application domains, LRA scores have been used to motivate architectural design choices in work in genomics (Nguyen et al., 2023; Romero and Zeghidour, 2023), analyzing ECGs (Zama and Schwenker, 2023), speech enhancement (Du et al., 2023), and reinforcement learning (Lu et al., 2023).

Long Range Arena (LRA) is a benchmark of 6 tasks covering diverse problem areas with different modalities. The ListOps task deals with the hierarchically structured data of mathematical operations with delimiters with $96K$ training and $2K$ test data. In Text task, IMDB movie review (Maas et al., 2011) text dataset is employed. Classification is performed character level to include additional complexity. The task has a balanced train-test split of size $25K$. The Retrieval task models the textual similarity of two documents for which the ACL Anthology Network (AAN) (Radev et al., 2013) dataset is utilized with $147K$ training and $17K$ test samples. Image task comprises of grayscale sequential CIFAR-10 image classification that puts the hurdle of 2D spatial relations into a 1D sequence of pixels. Finally, the Pathfinder and Path-X are the binary classification tasks containing grayscale images of dotted lines and circles connected or disconnected introduced in (Linsley et al., 2018). The difference between them is the sequence length from $1K$ to $16K$ both containing $160K$ training and $20K$ test samples.

We investigate the Long Range Arena and find that average performance is uncorrelated with performance on any of our malware tasks. While performance between LRA tasks is highly correlated with one another, they all correlate far worse with performance on malware task benchmarks shown in Figure 4.

In terms of performance, in text classification, HGConv achieved the second-best score of 88.15% and overall third-based average accuracy of 81.13% in the

Table 2: LRA benchmark scores. HGConv is from this work, while Hrrformer, S4, and SGConv scores are from their respective papers. All other scores are from (Tay et al., 2020). (Tay et al., 2020) and (Alam et al., 2023a) report that models "do not learn anything" on Path-X, shown here with a ✗. We observe this happening with HGConv as well.

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Average
Random	10.00	50.00	50.00	10.00	50.00	50.00	36.67
Transformer (Vaswani et al., 2017)	36.37	64.27	57.46	42.44	71.40	✗	54.39
Linformer (Wang et al., 2020)	35.70	53.94	52.27	38.56	76.34	✗	51.36
Performer (Choromanski et al., 2020)	18.01	65.40	53.82	42.77	77.05	✗	51.41
F-Net (Lee-Thorp et al., 2021)	35.33	65.11	59.61	38.67	77.80	✗	55.30
Luna-256 (Ma et al., 2021)	37.25	64.57	79.29	47.38	77.72	✗	61.24
H-Transformer (Zhu and Soricut, 2021)	49.53	78.69	63.99	46.05	68.78	✗	61.41
Hrrformer (Alam et al., 2023a)	39.98	65.38	76.15	50.45	72.17	✗	60.83
S4 (Gu et al., 2021)	59.60	86.82	90.90	**88.65**	94.20	96.35	86.09
SGConv (Li et al., 2022)	**61.45**	**89.20**	**91.11**	87.97	**95.46**	**97.83**	**87.17**
HGConv	49.75	88.15	90.62	85.08	92.04	✗	81.13

Table 3: Rank order performance of models on each benchmark. For Ember we use sequences of length up to 2^{14} as that's the maximum size found in the LRA Benchmark.

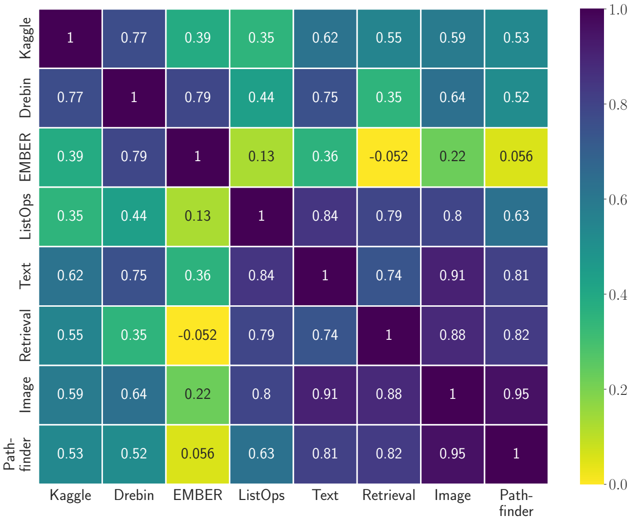
Model	K. Raw	K. Asm	D. Apk	D. Tar	Ember (2^{14})	LRA
Transformer (Vaswani et al., 2017)	9	8	8	8	9	8
Performer (Choromanski et al., 2020)	3	7	4	5	7	9
F-Net (Lee-Thorp et al., 2021)	3	6	6	7	5	7
Luna-256 (Ma et al., 2021)	8	9	9	9	8	5
H-Transformer (Zhu and Soricut, 2021)	7	4	4	2	3	4
Hrrformer (Alam et al., 2023a)	6	2	7	4	2	6
S4 (Gu et al., 2021)	2	2	2	2	4	2
SGConv (Li et al., 2022)	3	4	3	6	5	**1**
HGConv (ours)	**1**	**1**	**1**	**1**	**1**	3

LRA benchmark presented in Table 2. When comparing the rank order of model performance on LRA to our malware tasks, we see that LRA scores are not very predictive of performance on the malware benchmarks as shown in Table 3. LRA rates S4 and SGConv well ahead of the other models, while their performance is far less outstanding on malware benchmarks. SGConv in particular has a median ranking of fourth on our malware benchmarks, despite being the clear best model on LRA. The actual best malware model, HGConv, only beats S4 or SGConv on one of the six LRA tasks.

5 Conclusion

In this paper, we have introduced a long-range global convolutional network named HGConv by utilizing the properties of vector symbolic architecture called HRR. The proposed network facilitates learning by encoding and decoding features. Our network does not require a custom CUDA extension to run or have any intricate kernel design, unlike other existing global convolutional networks. Rather kernel is defined as parameters and learnt through auto-differentiation. In this work, we particularly focused on a real-world application of long-range models in malware detection. On Kaggle and Drebin benchmarks, the proposed method scored a new SOTA of 99.3% and 91.0%, respectively.

Results on EMBER demonstrate both superior accuracy and execution time where experiments are performed for various sequence lengths. HGConv has not only achieved a new highest accuracy of 93.56% for sequence length $16,384$ but also consistently consumed the least amount of time to execute. We have also investigated the performance and rank order of LRA and found that they all correlate far worse with malware performance and do not predict EMBER reliably. In



	Kaggle	Drebin	EMBER	ListOps	Text	Retrieval	Image	Path-finder
Kaggle	1	0.77	0.39	0.35	0.62	0.55	0.59	0.53
Drebin	0.77	1	0.79	0.44	0.75	0.35	0.64	0.52
EMBER	0.39	0.79	1	0.13	0.36	-0.052	0.22	0.056
ListOps	0.35	0.44	0.13	1	0.84	0.79	0.8	0.63
Text	0.62	0.75	0.36	0.84	1	0.74	0.91	0.81
Retrieval	0.55	0.35	-0.052	0.79	0.74	1	0.88	0.82
Image	0.59	0.64	0.22	0.8	0.91	0.88	1	0.95
Path-finder	0.53	0.52	0.056	0.63	0.81	0.82	0.95	1

Figure 4: The correlation between Malware and other LRA tasks accuracies. While performance between LRA tasks is highly correlated with one another, they all correlate far worse with the malware benchmarks.

conclusion, the experimental results, and comparison demonstrate the fidelity of the proposed method in a practical application like malware detection.

Acknowledgement

We thank Maya Fuchs for her feedback and copy-editing on this paper.

References

Ahdritz, G., Bouatta, N., Kadyan, S., Xia, Q., Gerecke, W., O'Donnell, T. J., Berenberg, D., Fisk, I., Zanichelli, N., Zhang, B., et al. (2022). Openfold: Retraining alphafold2 yields new insights into its learning mechanisms and capacity for generalization. *bioRxiv*, pages 2022–11.

Alam, M. M., Raff, E., Biderman, S., Oates, T., and Holt, J. (2023a). Recasting self-attention with holographic reduced representations. *arXiv preprint arXiv:2305.19534*.

Alam, M. M., Raff, E., and Oates, T. (2023b). Towards generalization in subitizing with neuro-symbolic loss using holographic reduced representations.

Alam, M. M., Raff, E., Oates, T., and Holt, J. (2022). Deploying convolutional networks on untrusted platforms using 2D holographic reduced representations. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S., editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 367–393. PMLR.

Anderson, H. S. and Roth, P. (2018). Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*.

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., and Siemens, C. (2014). Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26.

Avsec, Ž., Agarwal, V., Visentin, D., Ledsam, J. R., Grabska-Barwinska, A., Taylor, K. R., Assael, Y., Jumper, J., Kohli, P., and Kelley, D. R. (2021). Effective gene expression prediction from sequence by integrating long-range interactions. *Nature methods*, 18(10):1196–1203.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Botacin, M., Galhardo Moia, V. H., Ceschin, F., Amaral Henriques, M. A., and Grégio, A. (2021). Understanding uses and misuses of similarity hashing functions for malware detection and family clustering in actual scenarios. *Forensic Science International: Digital Investigation*, 38:301220.

Breitinger, F., Astebol, K. P., Baier, H., and Busch, C. (2013). mvhash-b - a new approach for similarity preserving hashing. In *Proceedings of the 2013 Seventh International Conference on IT Security Incident Management and IT Forensics*, IMF '13, page 33–44, Washington, DC, USA. IEEE Computer Society.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Chen, S., Wong, S., Chen, L., and Tian, Y. (2023). Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., et al. (2020). Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*.

Dalla-Torre, H., Gonzalez, L., Mendoza-Revilla, J., Carranza, N. L., Grzywaczewski, A. H., Oteri, F., Dallago, C., Trop, E., de Almeida, B. P., Sirelkhatim, H., et al. (2023). The nucleotide transformer: Building and evaluating robust foundation models for human genomics. *bioRxiv*, pages 2023–01.

Dao, T. (2023). Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. (2022). Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359.

Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. (2017). Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Du, Y., Liu, X., and Chua, Y. (2023). Spiking structured state space model for monaural speech enhancement. *arXiv preprint arXiv:2309.03641*.

Edward Raff, Joe Aurelio, and Charles Nicholas (2019). PyLZJD: An Easy to Use Tool for Machine Learning. In Chris Calloway, David Lippa, Dillon Niederhut, and David Shupe, editors, *Proceedings of the 18th Python in Science Conference*, pages 101 – 106.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, Chris Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. (2021). A framework for few-shot language model evaluation.

Gu, A. and Dao, T. (2023). Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*.

Gu, A., Dao, T., Ermon, S., Rudra, A., and Ré, C. (2020). Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33:1474–1487.

Gu, A., Goel, K., and Ré, C. (2021). Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*.

Hendrycks, D. and Gimpel, K. (2016). Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Tunyasuvunakool, K., Ronneberger, O., Bates, R., Žídek, A., Bridgland, A., et al. (2020). Alphafold 2. *Fourteenth Critical Assessment of Techniques for Protein Structure Prediction; DeepMind: London, UK*.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. (2020). Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR.

Lee-Thorp, J., Ainslie, J., Eckstein, I., and Ontanon, S. (2021). Fnet: Mixing tokens with fourier transforms. *arXiv preprint arXiv:2105.03824*.

Li, Y., Cai, T., Zhang, Y., Chen, D., and Dey, D. (2022). What makes convolutional models great on long sequence modeling? *arXiv preprint arXiv:2210.09298*.

Lillis, D., Breitinger, F., and Scanlon, M. (2017). Expediting mrsh-v2 approximate matching with hierarchical bloom filter trees. In *9th EAI International Conference on Digital Forensics and Cyber Crime (ICDF2C 2017)*, Prague, Czechia. Springer.

Lin, Z., Akin, H., Rao, R., Hie, B., Zhu, Z., Lu, W., dos Santos Costa, A., Fazel-Zarandi, M., Sercu, T., Candido, S., et al. (2022). Language models of protein sequences at the scale of evolution enable accurate structure prediction. *BioRxiv*, 2022:500902.

Linsley, D., Kim, J., Veerabadran, V., Windolf, C., and Serre, T. (2018). Learning long-range spatial dependencies with horizontal gated recurrent units. *Advances in neural information processing systems*, 31.

Lu, C., Schroecker, Y., Gu, A., Parisotto, E., Foerster, J., Singh, S., and Behbahani, F. (2023). Structured state space models for in-context reinforcement learning. *arXiv preprint arXiv:2303.03982*.

Ma, X., Kong, X., Wang, S., Zhou, C., May, J., Ma, H., and Zettlemoyer, L. (2021). Luna: Linear unified nested attention. *Advances in Neural Information Processing Systems*, 34:2441–2453.

Maas, A., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150.

McInnes, L., Healy, J., and Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

Menet, N., Hersche, M., Karunaratne, G., Benini, L., Sebastian, A., and Rahimi, A. (2023). Mimonets: Multiple-input-multiple-output neural networks exploiting computation in superposition. *Advances in Neural Information Processing Systems (NeurIPS)*, 36.

Muennighoff, N., Wang, T., Sutawika, L., Roberts, A., Biderman, S., Le Scao, T., Bari, M. S., Shen, S., Yong, Z. X., Schoelkopf, H., Tang, X., Radev, D., Aji, A. F., Almubarak, K., Albanie, S., Alyafeai, Z., Webson, A., Raff, E., and Raffel, C. (2023). Crosslingual generalization through multitask finetuning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Nguyen, E., Poli, M., Faizi, M., Thomas, A., Birch-Sykes, C., Wornow, M., Patel, A., Rabideau, C., Massaroli, S., Bengio, Y., et al. (2023). Hyenadna: Long-range genomic sequence modeling at single nucleotide resolution. *arXiv preprint arXiv:2306.15794*.

Nolet, C. J., Lafargue, V., Raff, E., Nanditale, T., Oates, T., Zedlewski, J., and Patterson, J. (2021). Bringing umap closer to the speed of light with gpu acceleration. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(1):418–426.

Oliver, J., Cheng, C., and Chen, Y. (2013). Tlsh – a locality sensitive hash. In *2013 Fourth Cybercrime and Trustworthy Computing Workshop*, page 7–13. IEEE.

Panconesi, A., Marian, Cukierski, W., and Committee, W. B. C. (2015). Microsoft malware classification challenge (big 2015).

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Cao, H., Cheng, X., Chung, M., Grella, M., GV, K. K., et al. (2023a). Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*.

Peng, B., Quesnelle, J., Fan, H., and Shippole, E. (2023b). Yarn: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*.

Plate, T. A. (1995). Holographic reduced representations. *IEEE Transactions on Neural networks*, 6(3):623–641.

Poli, M., Massaroli, S., Nguyen, E., Fu, D. Y., Dao, T., Baccus, S., Bengio, Y., Ermon, S., and Ré, C. (2023). Hyena hierarchy: Towards larger convolutional language models. *arXiv preprint arXiv:2302.10866*.

Radev, D. R., Muthukrishnan, P., Qazvinian, V., and Abu-Jbara, A. (2013). The acl anthology network corpus. *Language Resources and Evaluation*, 47(4):919–944.

Raff, E., Fleshman, W., Zak, R., Anderson, H. S., Filar, B., and McLean, M. (2021). Classifying sequences of extreme length with constant memory applied to malware detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(11):9386–9394.

Raff, E. and Nicholas, C. (2017a). An alternative to ncd for large sequences, lempel-ziv jaccard distance. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 1007–1015, New York, NY, USA. Association for Computing Machinery.

Raff, E. and Nicholas, C. (2017b). Malware classification and class imbalance via stochastic hashed lzjd. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 111–120.

Raff, E. and Nicholas, C. (2018a). Lempel-ziv jaccard distance, an effective alternative to ssdeep and sdhash. *Digital Investigation*, 24:34–49.

Raff, E. and Nicholas, C. (2020). A survey of machine learning methods and challenges for windows malware classification. In *NeurIPS 2020 Workshop: ML Retrospectives, Surveys & Meta-Analyses (ML-RSA)*.

Raff, E., Nicholas, C., and McLean, M. (2020). A New Burrows Wheeler Transform Markov Distance. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Raff, E. and Nicholas, C. K. (2018b). Lempel-ziv jaccard distance, an effective alternative to ssdeep and sdhash. *Digital Investigation*.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Raji, I. D., Denton, E., Bender, E. M., Hanna, A., and Paullada, A. (2021). Ai and the everything in the whole wide world benchmark. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Romero, D. W. and Zeghidour, N. (2023). Dnarch: Learning convolutional neural architectures by backpropagation. *arXiv preprint arXiv:2302.05400*.

Roussev, V. (2009). Building a better similarity trap with statistically improbable features. In *Proceedings of the 42Nd Hawaii International Conference on System Sciences*, HICSS '09, page 1–10, Washington, DC, USA. IEEE Computer Society.

Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Remez, T., Rapin, J., et al. (2023). Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Saul, R., Alam, M. M., Hurwitz, J., Raff, E., Oates, T., and Holt, J. (2023). Lempel-ziv networks. In Antorán, J., Blaas, A., Feng, F., Ghalebikesabi, S., Mason, I., Pradier, M. F., Rohde, D., Ruiz, F. J. R., and Schein, A., editors, *Proceedings on "I Can't Believe It's Not Better! - Understanding Deep Learning Through Empirical Falsification" at NeurIPS 2022 Workshops*, volume 187 of *Proceedings of Machine Learning Research*, pages 1–11. PMLR.

Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., and Metzler, D. (2020). Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*.

Team, M. N. (2023). Introducing mpt-30b: Raising the bar for open-source foundation models. Accessed: 2023-06-22.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. (2020). Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

Winter, C., Schneider, M., and Yannikos, Y. (2013). F2s2: Fast forensic similarity search through indexing piecewise hash signatures. *Digital Investigation*, 10(4):361–371.

Yao, Z., Wu, X., Li, C., Zhang, M., Qi, H., Ruwase, O., Awan, A. A., Rajbhandari, S., and He, Y. (2023). Deepspeed-visualchat: Multi-round multi-image interleave chat via multi-modal causal attention. *arXiv preprint arXiv:2309.14327*.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. (2020). Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 33:17283–17297.

Zama, M. H. and Schwenker, F. (2023). Ecg synthesis via diffusion-based state space augmented transformer. *Sensors*, 23(19):8328.

Zhu, Z. and Soricut, R. (2021). H-transformer-1d: Fast one-dimensional hierarchical attention for sequences. *arXiv preprint arXiv:2107.11906*.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes/No/Not Applicable]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Yes]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Yes]
   (d) Information about consent from data providers/curators. [Yes]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

A Hyperparameters

The hyperparameters used in the experiments are presented in Table 4. For Kaggle, Drebin, and EMBER no weight decay is used. For all the LRA tasks, weight decay of 0.05 is used except for Pathfinder where weight decay rate is 0.03.

Table 4: The hyperparameters used in each of the experiments. LN and BN refer to Layer Norm and Batch Norm. When the pre-norm is True, no post-normalization is used, and vice versa. For EMBER, experiments are performed on a variable sequence length T and the batch size is chosen according to the given expression to fit the data on memory.

	Norm	Pre Norm	Batch Size	Vocab Size	Sequence Length	Kernel Dim	Features	Dropout	Layers	Learning Rate	Epochs
Kaggle	LN	True	64	257	4096	32	256	0.1	1	0.01	10
Drebin	LN	True	32	257	4096	32	256	0.1	1	0.01	10
EMBER	LN	True	$\max(2^{16-\log_2 T}, 1)$	257	T	32	256	0.1	1	0.01	10
ListOps	BN	False	100	17	2000	64	128	0.0	6	0.01	40
Text	BN	True	50	257	4096	7	512	0.0	2	0.01	32
Retrieval	LN	True	128	128	4000	4	128	0.0	6	0.01	30
Image	LN	True	50	256	1024	128	512	0.2	6	0.01	200
Pathfinder	BN	False	64	256	1024	128	512	0.0	4	0.004	200

B UMAP 3D Representations

The UMAP 3D representations of the output from the penultimate layer of different models for Kaggle Raw, Kaggle Asm, Drebin Apk, and Drebin Tar are presented in Figure 5, Figure 6, Figure 7, and Figure 8, respectively. For Kaggle Raw and Asm, S4, SGConv, Hrrformer, HGConv has quite close accuracy, on the other hand for Drebin Apk and Tar, high variance in accuracy can be noted. Thus, noticeable changes can be observed for Drebin Apk and Tar in Figure 7 and Figure 8 where HGConv has visibly better clusters which makes the final layer classifier predict correctly.

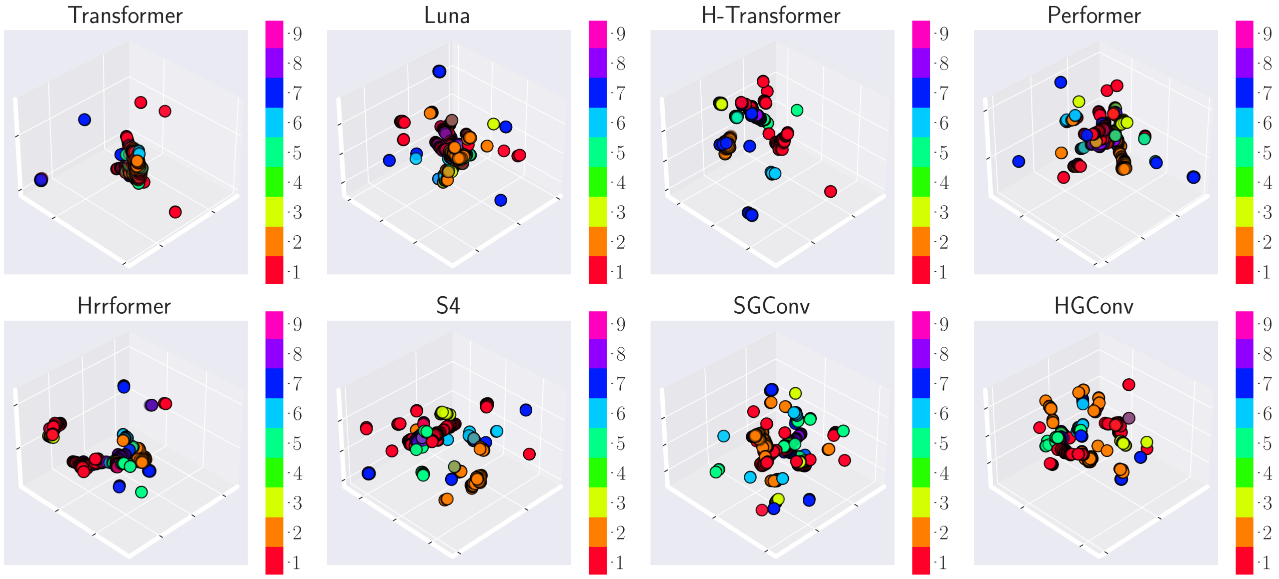

Transformer
Luna
H-Transformer
Performer
Hrrformer
S4
SGConv
HGConv

Figure 5: Kaggle Raw

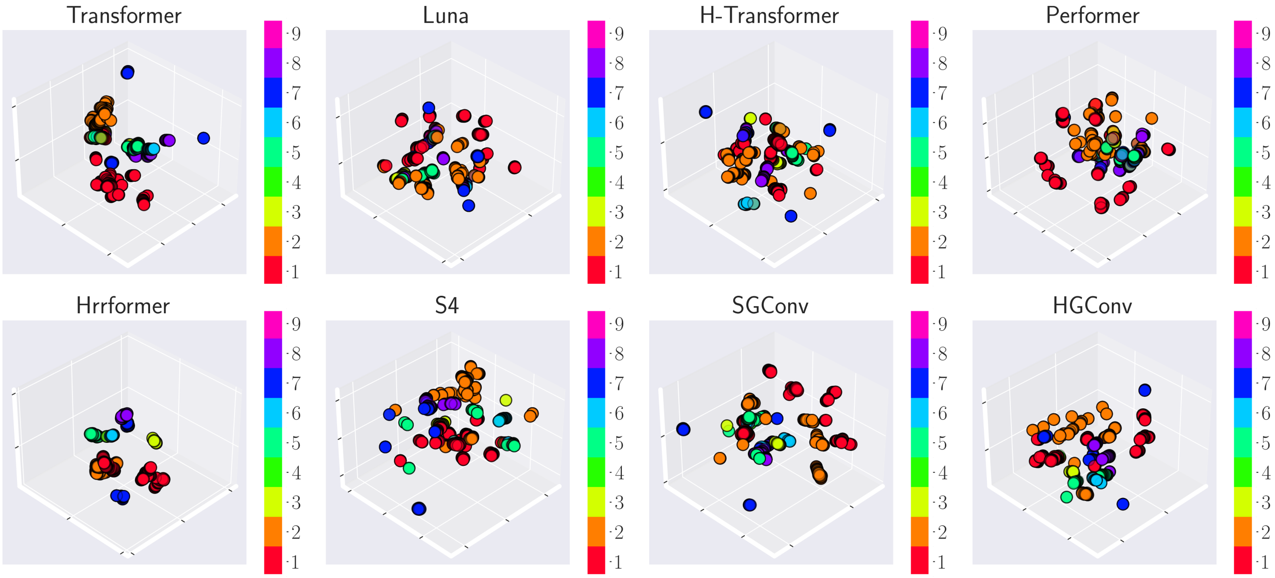


Figure 6: Kaggle Asm

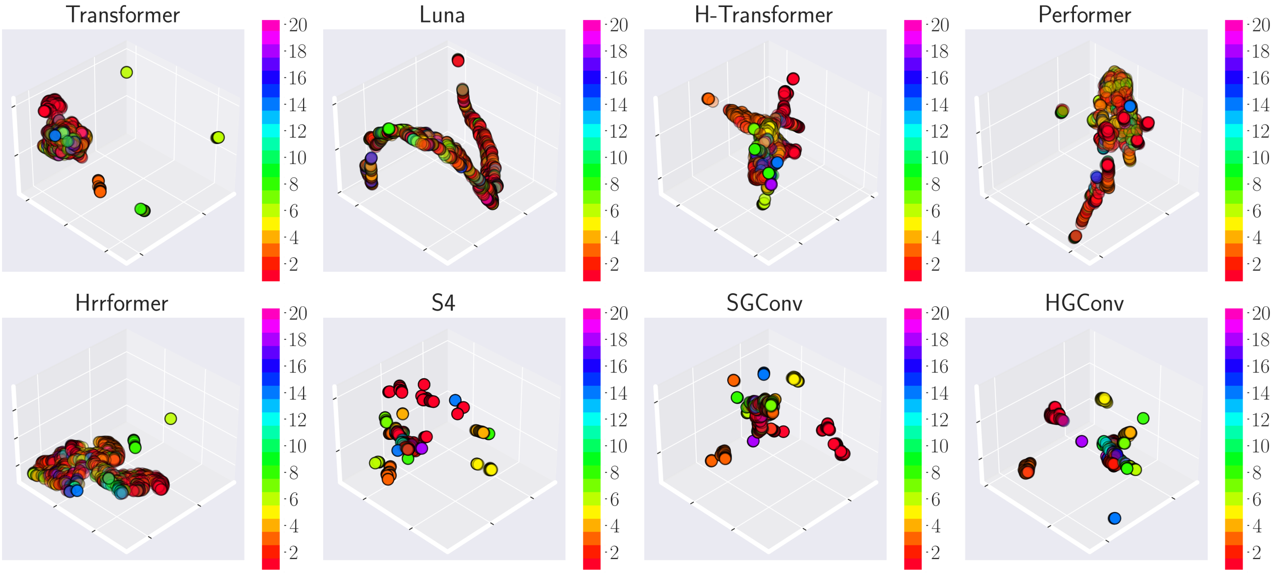


Figure 7: Drebin Apk

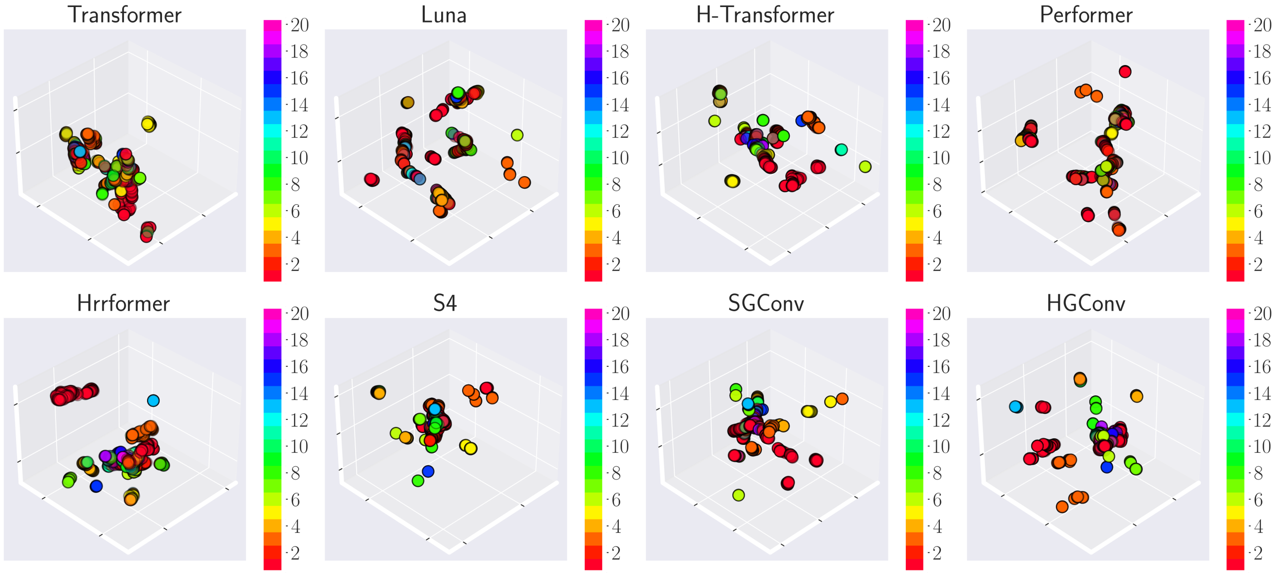


Figure 8: Drebin Tar

C EMBER Comprehensive Results

A broader comparison on EMBER classification results is shown in Figure 9 with additional models Linformer (Wang et al., 2020), Performer (Choromanski et al., 2020), and F-Net (Lee-Thorp et al., 2021). The numeric results of each method for different sequence lengths are presented in Table 5 where columns are kept empty if it faces OOM or OOT issue.

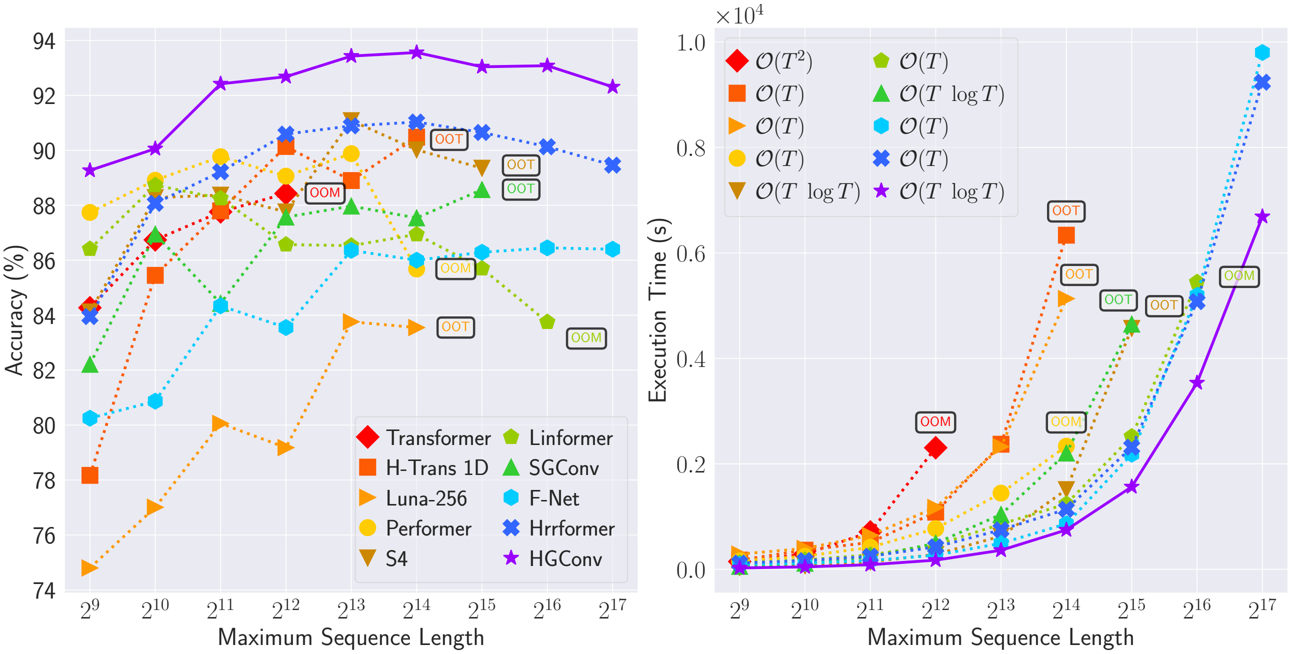


Figure 9: Ember long sequence malware classification. In the figure, OOT and OOM stand for out-of-time (OOT) and memory (OOM) shown for models that face such issues after a particular sequence length.

Table 5: Ember malware classification benchmark. The maximum sequence length ranges from 2^8 to 2^{17}. The best results are **boldfaced** and the second-best results are underlined.

Model	Metrics	Maximum Sequence Length									
		256	512	1,024	2,048	4,096	8,192	16,384	32,768	65,536	131,072
Transformer (Vaswani et al., 2017)	Acc	74.87	84.27	86.74	87.76	88.43	OOM				
	Time	101.59	146.96	286.98	708.7	2305.28	OOM				
Performer (Choromanski et al., 2020)	Acc	78.0	87.74	88.91	89.77	89.06	89.88	85.68	OOM		
	Time	115.77	159.59	247.02	418.1	770.75	1444.38	2334.94	OOM		
F-Net (Lee-Thorp et al., 2021)	Acc	76.42	80.25	80.87	84.34	83.55	86.36	86.00	86.29	86.45	86.40
	Time	84.84	95.58	113.2	165.77	267.21	492.44	861.48	2182.30	5191.26	9800.97
Luna-256 (Ma et al., 2021)	Acc	70.21	74.8	77.01	80.06	79.18	83.76	83.55	OOT		
	Time	243.04	287.5	395.87	643.81	1172.35	2326.15	5132.95	OOT		
H-Transformer (Zhu and Soricut, 2021)	Acc	59.59	78.17	85.45	87.8	90.14	88.9	90.48	OOT		
	Time	116.6	175.04	362.41	509.63	1082.67	2371.96	6336.37	OOT		
Hrrformer (Alam et al., 2023a)	Acc	78.06	83.95	88.07	89.22	90.59	90.89	91.03	90.65	90.13	89.46
	Time	91.35	117.96	165.18	247.32	423.55	748.48	1138.75	2315.62	5076.65	9237.78
S4 (Gu et al., 2021)	Acc	75.21	84.01	89.98	89.54	88.03	91.39	91.05	90.32	OOT	
	Time	25.19	34.48	70.4	133.58	276.98	615.95	1484.02	4486.83	OOT	
SGConv (Li et al., 2022)	Acc	74.45	82.21	86.95	84.43	87.57	87.97	87.54	88.57	OOT	
	Time	28.92	54.73	113.67	235.87	493.43	1039.98	2203.36	4648.69	OOT	
HGConv	Acc	**80.66**	**89.27**	**91.19**	**92.42**	**92.68**	**93.43**	**93.56**	**93.04**	**93.08**	**92.31**
	Time	**16.01**	**24.90**	**45.16**	**86.62**	**173.42**	**360.81**	**746.44**	**1564.69**	**3536.16**	**6689.41**