Block Access Pattern Discovery via Compressed Full Tensor Transformer

Anonymous Author(s)

ABSTRACT

The discovery and prediction of block access patterns in hybrid storage systems is of crucial importance for effective tier management. We remark that access is aggregated temporally and spatially, so the runtime requirements are relatively decreased, making complex models applicable for the deployment in the tier management systems. Existing methods are usually based on heuristics and unable to handle complex patterns. This work newly introduces *transformer* to block access pattern prediction. Moreover, enormous and rarely accessed blocks in storage systems would result in millions of redundant parameters in traditional transformers and make them impractical to be deployed. We incorporate Tensor-Train Decomposition (TTD) with transformer and propose the Compressed Full Tenor Transformer **(CFTT)**, in which all linear layers in the vanilla transformer are replaced with tensor-train layers. Weights of input and output layers are shared to further reduce parameters and reuse knowledge implicitly. CFTT can significantly reduce the model size and computation cost, which is critical to save storage space and inference time. Extensive experiments are conducted on one synthetic and three real-world datasets. The results demonstrate that the transformer-based methods achieve stably with up to 21.92% improvements of top-k hit rates than conventional methods. Moreover, the proposed CFTT compresses transformer 16 to 461 times and speeds up inference 5 times without sacrificing performance on the whole, which facilitates its applications in tier management in hybrid storage systems.

CCS CONCEPTS

• **Information systems → Hierarchical storage management**; **Data stream mining**; • **Applied computing → Forecasting**.

KEYWORDS

Storage System; Tensor Decomposition; Time Series Prediction

ACM Reference Format:
Anonymous Author(s). 2021. Block Access Pattern Discovery via Compressed Full Tensor Transformer. In *CIKM '21: 30th ACM International Conference on Information and Knowledge Management,November 1-5, 2021, Queensland, Australia.* 10 pages. https://doi.org/XX.XX/XX.XX

1 INTRODUCTION

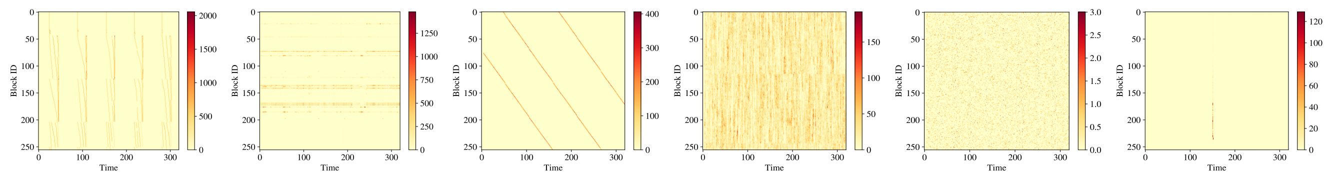
The behavior of applications to storage systems can be captured by a stream of Input/Output (I/O) requests. An I/O request is a read or write operation which points a Logic Block Address (LBA) where to read or write data with a size to represent how big the data are. The discovery of access patterns plays a key role in storage systems for cache prefetch [3, 6, 19], admission [4], and replacement [17], and data placement [7]. Traditional heuristic methods cannot learn from data and are incapable of modeling recency, frequency, spatial access locality, and future trends simultaneously [4, 9, 23, 26]. More flexible and capable neural networks like Recurrent Neural Networks (RNNs) are also utilized for predicting access patterns in the LBA request level for CPUs and cloud server cache prefetching [3, 6, 19]. However, multiple applications usually run in parallel and each application may access multiple regions simultaneously [1]. It would result in multi-stream switch and randomness that commonly exist in modern storage devices, making predicting each request difficult. Besides, real-time CPU cache prefetch poses strict requirements for model deployment, so neural networks are usually only tested in simulation environments [3, 6].

In hybrid storage systems, low latency and capacity Solid-State Drives (SSDs) serve as the cache or upper tier for high latency and capacity Hard-Disk Drives (HDDs) to reduce the cost but provide decent volume and access latency [7]. Specifically, frequently accessed data should be located in SSDs to reduce the access latency; rarely accessed data should reside in HDD to free SSD space. When a new request read data from SSDs, it is called as a hit in SSDs. Otherwise, a miss happens in the low-latency SSDs and it has to be read from lower tier. Due to the latency gap, reading data from SSDs rather than HDDs is of great significance to reduce the latency.

The granularity of data placement between SSD and HDD is blocks (4MB in our case) rather than sectors (512B) or pages (4KB). Additionally, data are moved between SSD and HDD intermittently in batches of requests rather than each LBA request. The spatial and temporal aggregation not only reduces the space of address and access randomness significantly, but also alleviates real-time requirements and makes complex model deployment feasible.

This work focuses on predicting the access frequency of blocks, which is modeled as a multivariate time series prediction problem. In this case, it is more practical to predict the high-level access pattern distribution with powerful neural networks. Six different real-world access patterns in aggregated multivariate time series are illustrated in Fig. 1. Seasonal (Fig. 1a), Re-accessed (Fig. 1b), and Sequential (Fig. 1c) access patterns are more predictable than Random (Fig. 1d and 1e) and Instantaneous ones (Fig. 1f).

Recently, transformer models show their superiority for natural language processing [2, 14] and time series prediction [16, 34]. Transformer models are more scalable and powerful than RNNs, and the multi-head self-attention mechanism provides global perceptive fields and short connection paths [30]. We thus introduce



(a) Seasonal (b) Re-accessed (c) Sequential (d) Random (e) Random (f) Instantaneous

Figure 1: Classes of block-level stream access patterns.

transformer to capture the spatial and temporal relationships of the more stable and multivariate block frequency series.

Even after request aggregation, there are still enormous and rarely accessed blocks in storage systems. The high memory or computation usage of neural networks, which is linear to the product of the dimensions of data and model, is a critical issue for deployment [2]. In addition, sparse data may lead to numerical issues and increase the difficulty of model training. Tensor decomposition especially Tensor-Train Decomposition (TTD) [22] is widely used for compression of input and hidden layers in classification neural networks [11, 29, 31]. Nevertheless, they are inapplicable to be directly deployed in limited-resource storage systems, as the uncompressed output layer contributes to almost 50% memory or computational usage for high-dimensional regression tasks.

To tackle these issues, we take advantage of TTD and propose the novel Compressed Full Tensor Transformer (CFTT) for access pattern prediction, to reduce redundant parameters and computational cost. Specifically, CFTT utilizes TTD to decompose all linear layers in the input embedding, internal transformer, and final output prediction layers with Tensor-Train Layers (TTLs). Weights sharing between input and output layers is enabled for implicit feature extraction and memory consumption reduction. The memory and computational complexity of CFTT is now linear to the data dimension rather than the product of the data and model dimensions in transformer and above tensor networks, which makes CFTT efficient and light-weight.

The contributions of this work are summarized as follows:

- We model the block access pattern mining as multivariate access frequency series predictions of blocks in time windows. To the best of our knowledge, we are the first to employ transformer and several tensor networks to predict block access patterns in storage systems.
- We propose a novel compressed Full Tensor Transformer to reduce parameters and computational cost of transformer, and share the compact weights among the input and output TTLs to obtain better compression and prediction results.
- We validate the memory and computational efficiency with theoretical analysis and extensive experiments. We also conduct ablation studies for CFTT to analyse the effect of compressing input, internal transformer, and output TTLs, and tensor-train settings.
- CFTT has been verified in commercial storage products and is in the experimental phase for deployment in future series.

2 RELATED WORK

In this section, we briefly review works including univariate and multivariate approaches for cache, storage systems and general time series prediction, and efficient and tensor networks.

2.1 Univariate Approaches

Practical solutions for access pattern discovery are normally simple but efficient due to the constraints of storage and computational resources. For example, cache usually utilizes heuristic approaches such as stride prefetch, Least Recently Used (LRU) [26], and Least Frequently Used (LFU) [4]. LRU only utilizes the recency of access, while Window-based Direct Address Counting (WDAC) [23], Countering Bloom Filter (CBF) [9], and Multiple Bloom Filter (MBF) [23] can capture recency and frequency simultaneously. Another advantage of Bloom filters is storage efficiency as them only store the hashed index of addresses which may range among 2^{64}.

These methods are usually suboptimal and unable to learn from dynamic patterns. The data placement granularity between SSD and HDD alleviates requirements for model inference time cost, creating opportunities for complex models. RNNs have natural advantages of modeling sequence, as they can process variable length sequence globally by reusing weights in each step. RNNs are thus widely used to model the LBA level access patterns and show competitive performance for cache prefetch [3, 6] and eviction [17], compared with traditional approaches. For example, the large LBA address space is first reduced by computing the difference of two adjacent LBA requests as LBA deltas and only feeding the most common deltas to RNNs. RNNs then predict LBAs that are most likely to be accesses, which will be prefetched cache in advance [3, 6].

However, these models take fine-grain access streams as input and can only memorize information in relatively short streams. Therefore, they are sensitive to randomness in access streams such as stream mixture and random streams. In which cases, cache is polluted by random I/Os with large reuse distances or prefetch is not triggered. Therefore, the fine-grain access stream should be aggregated temporally and spatially to get multivariate access frequency series which contain robust and high-level patterns. This way of access preprocessing is commonly used for tier management in hybrid storage systems. Traditional time series model like Auto Regressive Integrated Moving Average (ARIMA) can also model high-level workload of CPU and memory [8]. Although univariate WDAC, CBF and ARIMA can also model frequency series, they cannot capture the spatial locality of access.

2.2 Multivariate Approaches

Neural network models can be easily extended to model complex patterns among multivariate series [14, 16]. For example, RNN-based seq2seq models generate frequency predictions of multivariate objects in content delivery networks [19]. However, RNNs may be incapable of modeling long and complex frequency sequences. Long Short Term Network (LSTNet) incorporates convolutional, recurrent, highway-autoregressive (AR), and recurrent-skip layers to model seasonal time series and increases model robustness to scale changing [14]. Transformer is known for the excellent sequence modeling capability with sufficient training data [2, 16, 34]. It is already applied to model application-level features in storage systems [33]. The internal linear layers can learn local and spatial features, then the self-attention mechanism captures the global and temporal patterns. The self-attention mechanism does not introduce new parameters but builds short connection paths between input and output elements. In addition, transformer can be easily scaled to utilize the power of Graphical Processing Units (GPUs).

2.3 Efficient and Tensor Networks

Nevertheless, the high-dimensional and sparse data lead to millions even billions of redundant parameters of neural models especially transformer, which may take thousands GPU days for training [2]. Pruning, quantization, and knowledge distillation are common ways to compress learned models for mobile or low-capacity devices [10, 15, 28]. However, transformer and other neural models are still hard to be trained and deployed in those data, which hinders them to be introduced to block access pattern discovery.

The problems caused by high-dimensional and sparse data can be partly solved with tensor decomposition methods, which are already introduced to learn compact weights in ARIMA [27], input embedding layers [11, 12, 25], Convolutional Neural Networks (CNNs) [24] and RNNs [12, 29, 31, 32]. For example, Tucker decomposition compresses the high-dimensional and short series into low-rank core tensors and learns intrinsic correlations of input series. The generalized ARIMA then explicitly predicts future core tensors, followed by inverse transformation that shares weights from the input decomposition [27]. Tensor-Train Recurrent Neural Network (TT-RNN) utilizes a Tensor-Train Layer (TTL) to replace the input-to-hidden weight in RNN models and significantly improves video classification performance without any commonly used deep CNN layers [31]. In addition, the work applies TTD to compress the input word embedding layer of RNN and transformer models [11], but complexity is limitedly reduced due to the untouched large internal transformer and output layers.

3 PROBLEM DEFINITION

To alleviate the randomness of fine-grain access requests, the requirements for real-time model inference, and reduce the address space, the fine-grain requests in hybrid storage systems are aggregate temporally and spatially as multivariate access pattern series. Specifically, we first split the trace into windows by a certain number of requests, then count the number of requests to a block as the access frequency to that block. A block that is not accessed in a time window has frequency as 0. We can thus model the access pattern mining as access frequency prediction of blocks in time windows. Assuming the historical access frequency series of all blocks $\mathbf{X}_{1:T} \in \mathbb{R}^{T\times d}$ at steps $1, \cdots, T$ are known, where T is the sequence length and d is the series dimension or number of blocks, we want to predict the future frequency $\mathbf{X}_{T+1} \in \mathbb{R}^{d}$ in step $T+1$.

An issue of modeling block level access patterns is how to process high dimensional and sparse data, whose dimension is 256K under 4MB blocks in a 1TB disk and most blocks are not accessed in a time window. A common way to compress high dimensional input

Table 1: Model size comparison of vanilla transformer for series prediction (or language modeling).

d	d_e	L	Input layer	Transformer	Output layer	Total
256	512	1	128K (7.1%)	1.5M (85.8%)	128K (7.1%)	1.75M
256	2048	1	512K (4.0%)	24M (92.0%)	512K (4.0%)	25M
256K	2048	1	512M (48.9%)	24M (2.2%)	512M (48.9%)	1.02B
256K	2048	12	512M (39.0%)	288M (22.0%)	512M (39.0%)	1.28B

Table 2: Space and computational complexity of transformer (Assuming $d \gg d_e, L$)

Modules	Space	Computational
Input layer	$d_e d + d_e$	$O(d_e d)$
Transformer	$6L(d_e^2 + d_e)$	$O(Ld_e^2)$
Output layer	$d_e d + d$	$O(d_e d)$
Total	$O(d_e d)$	$O(d_e d)$

$\mathbf{X}$ as a low dimensional hidden state $\tilde{\mathbf{X}} \in \mathbb{R}^{T\times d_e}$ is the position-wise feed forward layer:

$$\tilde{\mathbf{X}} = \mathbf{X} \cdot \mathbf{W} + \mathbf{b}, \tag{1}$$

where $\mathbf{W} \in \mathbb{R}^{d\times d_e}$ is the embedding matrix and $\mathbf{b} \in \mathbb{R}^{d_e}$ is the bias vector. Although this approach is simple and intuitive, $\mathbf{W}$ will become significantly large with the high input (d) and model (d_e) dimensions. From Tab. 2, we can conclude that the input and output layers are the dominant parts for memory and time usage if d is large, and the transformer layers with large model sizes are also noticeable. Last but not least, the high sparsity of data indicates the redundancy of model parameters and increases the difficulty of robust training and inference.

4 COMPRESSED FULL TENSOR TRANSFORMER

In this section, we first introduce general definitions of Tensor-Train Decomposition (TTD), Tensor-Train Layers (TTLs). Inspired by [11, 27, 29, 31], we approximate all linear input, transformer and output layers with TTLs to fully tensorize the vanilla transformer. The proposed CFTT as visualized in Fig. 3 can thus improve memory and computation usage for series prediction. CFTT can also be utilized together with other efficient transformer approaches [10, 16, 28].

Specifically, we first tensorize the high-dimensional historical input series into high-order and low-rank tensors, then use the input TTL to extract local and inter-variate features as a tensor, whose dimension is much smaller than the original input dimension (number of blocks). From which, the compressed transformer layers could model the global temporal patterns with the self-attention mechanism. Finally, the output TTL generates prediction in the tensor form from the high-level features extracted by transformer TTLs. Besides, the high-order weights of input and output tensor train layers are shared. That not only further compresses the model parameters, but also shares the prior information learned from input data, leading to better hit rates. Instead of compressing a well-trained transformer with linear layers, the weight tensors of CFTT are initialized randomly and learned from scratch to remove the need of storing and training giant vanilla transformer models. The memory and computational complexity of CFTT is now $O(d)$

instead of $O(d_e d)$ of transformer, assuming d is much greater than d_e and other model parameters. More details are in Sec. 4.5.

4.1 Tensor-Train Layer

A high-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ can be represented in the Tensor-Train Decomposition (TTD) [22]:

$$\mathcal{X}(i_1, i_2, \ldots, i_N) \overset{\text{TTD}}{=} \sum_{R_0,\ldots,R_N} \mathcal{G}_1(R_0, i_1, R_1)\mathcal{G}_2(R_1, i_2, R_2)\ldots\mathcal{G}_N(R_{N-1}, i_N, R_N), \tag{2}$$

where smaller Tensor-Train (TT) core tensors $\mathcal{G}_n \in \mathbb{R}^{R_{n-1} \times I_n \times R_n}$ for $n \in \{1, \ldots, N\}$. N is the total number of TT core tensors. Since $R_0 = R_N = 1$, this decomposition can also be represented graphically by a linear tensor network [22]. From which, we can accurately approximate and reconstruct $\mathcal{X}$ with less memory and computational cost if TT ranks $\{R_n\}_{n=1}^{N-1}$ are correctly set.

In order to compress the giant weight matrix of linear layers in the input, transformer, and output layers in Equation 1, the general input $\mathbf{X} \in \mathbb{R}^{T \times D}$ of linear layers can be tensorized along the spatial dimension as a high order tensor $\mathcal{X} \in \mathbb{R}^{T \times I_1 \times \cdots \times I_N}$. The strong spatial similarity due to spatial locality can thus be modeled implicitly. However, the input is not folded along the temporal dimension as temporal correlation does not always exist and T is relatively small in our case. The general output $\tilde{\mathbf{X}} \in \mathbb{R}^{T \times D_e}$ is also folded to a more compact tensor $\tilde{\mathcal{X}} \in \mathbb{R}^{T \times J_1 \times \cdots \times J_N}$ with a TTL with weights $\mathcal{W} \in \mathbb{R}^{I_1 \times \cdots \times I_N \times J_1 \times \cdots \times J_N}$ and a bias $\mathcal{B} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$ [11, 21, 31], i.e.,

$$\begin{aligned}\tilde{\mathcal{X}}(:, j_1, j_2, \ldots, j_N) &= \sum_{i_1=1}^{I_1}\sum_{i_2=1}^{I_2}\cdots\sum_{i_N=1}^{I_N} \mathcal{W}\big((i_1, j_1), (i_2, j_2), \ldots, \\ &(i_N, j_N)\big) \cdot \mathcal{X}(:, i_1, i_2, \ldots, i_N) + \mathcal{B}(j_1, j_2, \ldots, j_N).\end{aligned} \tag{3}$$

Compared with Equation 1, we can see that all input, output, and weights in TTL are high-order tensors instead of matrices in original linear layers. The full weight tensor $\mathcal{W}$ of size $\prod_{n=1}^{N} I_n J_n = D_e D$ are then decomposed by TTD with the double index trick [21]:

$$\begin{aligned}&\hat{\mathcal{W}}((i_1, j_1), (i_2, j_2), \ldots, (i_N, j_N)) \\ &\overset{\text{TTD}}{=} \mathcal{G}_1(i_1, j_1)\mathcal{G}_2(i_2, j_2)\ldots\mathcal{G}_N(i_N, j_N),\end{aligned} \tag{4}$$

where 4D core tensors $\mathcal{G}_n \in \mathbb{R}^{I_n \times J_n \times R_{n-1} \times R_n}$ for $n \in \{1, \ldots, N\}$ will be learned jointly with downstream components. In which, I_n, J_n, and R_n are the input shape, output shape, and rank of the n-th tensor-train core $\mathcal{G}_n$, respectively. $R_0 = R_N = 1$ by default. Therefore, the vanilla linear layer is approximated with a TTL as shown in Fig. 2:

$$\tilde{\mathcal{X}} = \text{TTL}(\mathcal{G}_{1:N}, \mathcal{X}, \mathcal{B}), \tag{5}$$

where $\mathcal{G}_{1:N}$ is an ensemble of $\mathcal{G}_n, n \in \{1, \cdots, N\}$. Now the number of parameters is $\sum_{n=1}^{N}(I_n J_n R_{n-1} R_n)$, which is much smaller than $D_e D$. Besides, the computational complexity is of $O(N\tilde{I}\tilde{R}^2\tilde{J}^N)$, where $\tilde{I} = \max_{n\in[1,N]} I_n$, $\tilde{J} = \max_{n\in[1,N]} J_n$, $\tilde{R} = \max_{n\in[1,N]} R_n$, $O(\sum_{i=1}^{N}(I_i J_i R_{i-1} R_i))$ instead of $O(D_e D)$ [31]. Therefore, the problems caused by the high dimensional input data and hidden state is solved with TTLs.

4.2 TTL for Input Embedding

We borrow the idea from the tensor-train embedding layer [11] to approximate the input embedding layer for local series feature extraction with a TTL. Besides, We found that directly initializing lower-rank core tensors and training them from scratch leads to similar results as training then decomposing a full size weight $\mathbf{W}$ to warm start core tensors. It is time and memory consuming to initialize or train the large weight $\mathbf{W}$ when d is significant. We thus directly initialize a TTL with predefined input shapes $I_{1:N}$, output shapes $J_{1:N}$, and ranks $R_{0:N}$. Therefore, the TTL in Equation 5 is modified for the input layer

$$\tilde{\mathcal{X}} = \text{TTL}(\mathcal{G}^I_{1:N}, \mathcal{X}, \mathcal{B}^I), \tag{6}$$

where the tensorized input $\mathcal{X} \in \mathbb{R}^{T \times I_1 \times \cdots \times I_N}$, the tensorized output $\tilde{\mathcal{X}} \in \mathbb{R}^{T \times J_1 \times \cdots \times J_N}$, and 4D weights $\mathcal{G}^I_{1:N}$ are initialized randomly and learned together with downstream modules. Therefore, $\mathbf{W} \in \mathbb{R}^{d \times d_e}$ in Equation 1 are approximated with core tensors $\mathcal{G}^I_n \in \mathbb{R}^{I_n \times J_n \times R_{n-1} \times R_n}$ in the input TTL with Equation 5. $\prod_{n=1}^{N} I_n = d$ and $\prod_{n=1}^{N} J_n = d_e$. In addition, the bias vector $\mathbf{b}$ is folded into $\mathcal{B}^I \in \mathbb{R}^{J_1 \times \cdots \times J_N}$.

4.3 TTL for Tensor Transformer Layer

There are usually L transformer layers in transformer models. In the implementation of the original transformer model, there are 6 linear layers (query, key, value, and output projection layers in the self-attention layer, and two additional linear layers in the following feed forward networks) in each transformer layer.

4.3.1 *Self-Attention Layer.*

The linear query, key, and value projection layers at the attention layer l can be represented as:

$$\mathbf{Q}^l = \mathbf{H}^{l-1} \cdot \mathbf{W}^{l,q} + \mathbf{b}^{l,q}, \tag{7}$$

$$\mathbf{K}^l = \mathbf{H}^{l-1} \cdot \mathbf{W}^{l,k} + \mathbf{b}^{l,k}, \tag{8}$$

$$\mathbf{V}^l = \mathbf{H}^{l-1} \cdot \mathbf{W}^{l,v} + \mathbf{b}^{l,v}, \tag{9}$$

where $\mathbf{H}^{l-1}$ represents the hidden state input of the l-th layer and the initial value $\mathbf{H}^0$ is the output of the input layer $\tilde{\mathbf{X}}$. $\mathbf{Q}^l$, $\mathbf{K}^l$, and $\mathbf{V}^l$ denote query, key and value variables respectively. 2D weight matrices $\mathbf{W}^{l,q}$, $\mathbf{W}^{l,k}$ and $\mathbf{W}^{l,v} \in \mathbb{R}^{d_h \times d_{ff}}$ and bias vectors $\mathbf{b}^{l,q}$, $\mathbf{b}^{l,k}$, and $\mathbf{b}^{l,v} \in \mathbb{R}^{d_{ff}}$. d_h is the input and output dimension, and $d_h = d_e$ in our case. d_{ff} is the hidden size of feed forward layer. These projection layers map the same input state to the different hidden states representing query, key and value variables, respectively, for the self-attention computation [30], which is defined as follows:

$$\tilde{\mathbf{V}^l} = \text{softmax}(\frac{\mathbf{Q}^l \mathbf{K}^{l\top}}{\sqrt{d_h}})\mathbf{V}^l. \tag{10}$$

Afterwards, an additional linear output projection layer is also used to process the weighted output $\tilde{V}$ with

$$\mathbf{O}^l = \tilde{\mathbf{V}^l} \cdot \mathbf{W}^{l,o} + \mathbf{b}^{l,o}, \tag{11}$$

where the weight $\mathbf{W}^{l,o} \in \mathbb{R}^{d_{ff} \times d_h}$ and bias $\mathbf{b}^{l,o} \in \mathbb{R}^{d_h}$.

4.3.2 *Feedforward Layer.*

Then $\mathbf{O}^l$ is fed to the position-wise two-layer feed forward layer

$$\mathbf{H}^l = \text{RELU}(\mathbf{O}^l \cdot \mathbf{W}^{l,f_1} + \mathbf{b}^{l,f_1}) \cdot \mathbf{W}^{l,f_2} + \mathbf{b}^{l,f_2}, \tag{12}$$

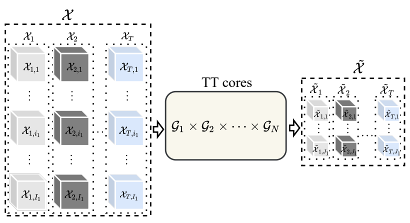


Figure 2: Tensor-Train Layer.

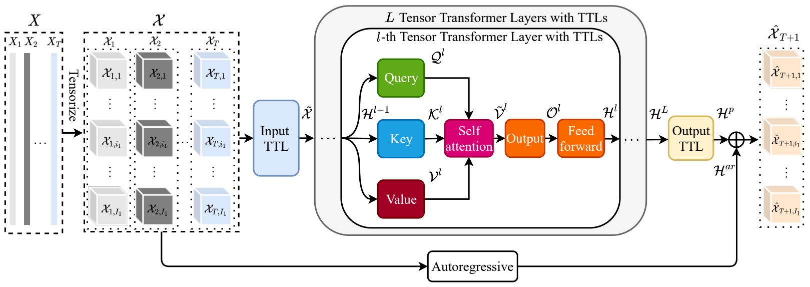


Figure 3: Compressed Full Tensor Transformer.

where weight $\mathbf{W}^{l,f_1}$ and bias $\mathbf{b}^{l,f_1}$ are at the first feed forward layer, and weight $\mathbf{W}^{l,f_2}$ and bias $\mathbf{b}^{l,f_2}$ are at the second feed forward layer, RELU is the REctified Linear Unit function [30].

4.3.3 Internal Linear Tensor-Train Layers. According to Tab. 2, the internal transformer layers will also have millions of parameters when the model size and the number of layers are large. Self-attention computation does not introduce any learnable parameters, while parameters of six dense projection layers are of $O(d_h \times d_{ff})$. Usually, $d_{ff} = d_h$ or $d_{ff} = 4 \cdot d_h$. Similar to the input embedding layer, the six internal linear layers can also be approximated with TTLs in Equation 5 to compress the internal transformer model and learn more compact representations. For simplicity, we use the superscript $*$ to denote all six projection layers, including q, k, v, o, f_1, f_2. In this way, weight matrices $\mathbf{W}^{l,*}$ of the six linear projection layers from Equation 7 to 12 are decomposed with $\mathcal{G}^{l,*}$, and bias vectors $b^{l,*}$ are folded as $\mathcal{B}^{l,*}$. For example, if $d_{ff} = d_h = d_e$ and $d_e = J_1 \times \cdots \times J_N$, we can also set $d_{ff} = J_1 \times \cdots \times J_N$. With them as input and output TT shapes, the square matrix $\mathbf{W}^{l,*}$ can be first folded as $\mathcal{W}^{l,*} \in \mathbb{R}^{J_1 \times \cdots \times J_N \times J_1 \times \cdots \times J_N}$, which is then decomposed into $\mathcal{G}^{l,*}_{1:N}$ with TTD in Equation 4:

$$\begin{aligned}&\hat{\mathcal{W}}^{l,*}((j_1', j_1), (j_2', j_2), \ldots, (j_N', j_N))\\ &\overset{\text{TTD}}{=} \mathcal{G}_1^{l,*}(j_1', j_1)\mathcal{G}_2^{l,*}(j_2', j_2)\ldots\mathcal{G}_N^{l,*}(j_N', j_N),\end{aligned} \tag{13}$$

where $\mathcal{G}^{l,*}_{1:N}$ is an ensemble of N core tensors $\mathcal{G}^{l,*}_n \in \mathbb{R}^{J_n \times J_n \times R'_{n-1} \times R'_n}$ with the same input and output shapes. R'_n for $n \in [0, N]$ are the TT ranks of projection layers. As a result, the query, key, and value projection layers are approximated with the corresponding TTLs:

$$\mathcal{Q}^l = \text{TTL}(\mathcal{G}^{l,q}, \mathcal{H}^{l-1}, \mathcal{B}^{l,q}), \tag{14}$$

$$\mathcal{K}^l = \text{TTL}(\mathcal{G}^{l,k}, \mathcal{H}^{l-1}, \mathcal{B}^{l,k}), \tag{15}$$

$$\mathcal{V}^l = \text{TTL}(\mathcal{G}^{l,v}, \mathcal{H}^{l-1}, \mathcal{B}^{l,v}). \tag{16}$$

$\mathcal{Q}^l, \mathcal{K}^l$, and $\mathcal{V}^l$ are unfolded into matrices to forward the softmax computation in Equation 10 and layer normalization, then the softly weighted $\tilde{\mathbf{V}}^l$ is folded to high-order tensor $\tilde{\mathcal{V}}^l$. The outptu and two linear feedforward layers are also compressed with TTLs:

$$\mathcal{O}^l = \text{TTL}(\mathcal{G}^{l,o}, \tilde{\mathcal{V}}^l, \mathcal{B}^{l,o}), \tag{17}$$

$$\mathcal{H}^l = \text{TTL}\big(\mathcal{G}^{l,f_2}, \text{RELU}\big(\text{TTL}(\mathcal{G}^{l,f_1}, \mathcal{O}^l, \mathcal{B}^{l,f_1})\big), \mathcal{B}^{l,f_2}\big). \tag{18}$$

From Equation 14 to 18, the right hand high order tensors $\mathcal{Q}^l, \ldots \mathcal{H}^l \in \mathbb{R}^{T \times J_1 \times \ldots, J_N}$, and $\mathcal{B}^{l,*} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$. The initial value of high-order $\mathcal{H}^0$ in the first tensor transformer layer of CFTT is set as the tensor output $\tilde{\mathcal{X}}$ of the input TTL. The last hidden state $\mathcal{H}^L$ in the last transformer layer is fed to the output layer as extracted global features to make predictions.

4.4 TTL for Output Layer with Highway Autoregressive

The output layer works as a predictor and maps the low dimensional hidden states to the high dimensional future block access frequency series. A simple linear layer can generate accurate prediction if the output of transformer layers already represents informative features [2], so we employ a linear layer for prediction

$$\mathbf{H}^p = \mathbf{H}^L \cdot \mathbf{W}^p + \mathbf{b}^p, \tag{19}$$

where $\mathbf{W}^p \in \mathbb{R}^{d_h \times d}$, and $\mathbf{b}^p \in \mathbb{R}^d$. According to Tab. 2, it is also necessary to approximate the original linear output layer in Equation 19 with TTL to reduce the space complexity:

$$\mathcal{H}^p = \text{TTL}(\mathcal{G}^p, \mathcal{H}^L, \mathcal{B}^p). \tag{20}$$

Following the settings in above subsections, we define the number of core TT tensors as N, and the input and output TT shapes of $\mathcal{G}^p_n$ are J_n and I_n, respectively, so TT cores $\mathcal{G}^p_n \in \mathbb{R}^{J_n \times I_n \times R_{n-1} \times R_n}$ and the bias tensor $\mathcal{B}^p \in \mathbb{R}^{I_1 \times \cdots \times I_N}$.

If $d_h = d_e$ and $\tilde{X}$ is in the same space of $\mathbf{H}^L$, then the transpose of W can be reused as $\mathbf{W}^p$ to decrease parameters and build a connection of input and output. Extending the relationship to tensor forms, the transpose of $\mathcal{G}^I$ is used in the output TTL as prior information learned from input data. Apart from further reducing model parameters, weight sharing is also helpful for increasing top-k IO hit rates. In our case, the positions of top-k elements are critical, compressing the output layer may cause a small decrease of prediction errors but a considerable drop of the top-k IO accuracy or hit rate. In addition, in order to make models robust to scale changing and unstationary patterns [14], we introduce a highway autoregressive path connecting input in Equation 6 and output:

$$\mathcal{H}^{ar} = w^{ar} \cdot \mathcal{X}, \tag{21}$$

where $w^{ar} \in \mathbb{R}^T$. Finally, the sum of $\mathcal{H}^p$ and $\mathcal{H}^{ar}$ is the final prediction in the next step:

$$\hat{\mathcal{X}}_{T+1} = \mathcal{H}^p + \mathcal{H}^{ar}. \tag{22}$$

4.5 Space and Computational Complexity Analysis of CFTT

Based on the analysis in Sec. 4.1, the space complexity of input TTL weights is $\sum_{n=1}^{N}(I_n J_n R_{n-1} R_n)$, and the bias tensor has d_e elements.

Table 3: Space and computational complexity of CFTT

Modules	Space	Computational
Input layer	$\sum_{n=1}^{N}(I_n J_n R_{n-1} R_n) + d_e$	$O(N\tilde{I}\tilde{R}^2 d_e)$
Transformer	$6L\big(\sum_{n=1}^{N}(J_n J_n R'_{n-1} R'_n) + d_e\big)$	$O\big(LN\tilde{J}(\tilde{R}')^2 d_e\big)$
Output layer	$\sum_{n=1}^{N}(I_n J_n R_{n-1} R_n) + d$	$O(N\tilde{J}\tilde{R}^2 d)$
Total	$O(d)$	$O(N\tilde{J}\tilde{R}^2 d)$

Therefore, input TTL would take $\sum_{n=1}^{N}(I_n J_n R_{n-1} R_n) + d_e$ memory space. The computational complexity is $O(N\tilde{I}\tilde{R}^2\tilde{J}^N) = O(N\tilde{I}\tilde{R}^2 d_e)$, where $\tilde{I} = \max_{n\in[1,N]} I_n$, $\tilde{J} = \max_{n\in[1,N]} J_n$, $\tilde{R} = \max_{n\in[1,N]} R_n$ [21, 31]. Assuming $d_{ff} = d_h = d_e$, then the number of parameters of a internal transformer projection layer is $\sum_{n=1}^{N}(J_n J_n R'_{n-1} R'_n)+d_e$, which is much less than the original d_e^2. In addition, the computational complexity is now $O(N\tilde{J}(\tilde{R}')^2\tilde{J}^N) = O(N\tilde{J}(\tilde{R}')^2 d_e)$, where $\tilde{R}' = \max_{n\in[1,N]} R'_n$. The memory usage of the output TTL weight tensors is reduced to $\sum_{n=1}^{N}(I_n J_n R_{n-1} R_n)$, and the huge bias tensor now dominates and has d parameters because of the the large output dimension. In total, There are thus $\sum_{n=1}^{N}(I_n J_n R_{n-1} R_n) + d$ parameters, which results in the linear dependency of CFTT space complexity to the data dimension. The computational complexity is $O(N\tilde{J}\tilde{R}^2\tilde{I}^N) = O(N\tilde{J}\tilde{R}^2 d)$.

The space and computational complexity of each parts of CFTT is summarized in Tab. 3, assuming $d \gg d_e, I_n, J_n, R_n, R'_n$. From which, we can summarize that the space complexity of CFTT is $O(\max\{d, d_e\}) = O(d)$, because of the huge bias tensor. Without bias tensors, the space complexity of CFTT would dramatically decrease to $O\big(\max\{\tilde{I}\tilde{J}\tilde{R}^2, \tilde{J}^2(\tilde{R}')^2\}\big)$ and only depend on tensor-train shape and rank settings. In addition, The computational complexity of CFTT is $O(N\tilde{J}\tilde{R}^2 d)$, and $O(N\tilde{J}\tilde{R}^2 d) \ll O(d_e d)$, if $d_e \gg N, \tilde{J}, \tilde{R}$. In contrast, the space and computational complexity of transformer are both $O(d_e d)$ from Tab. 2. Therefore, we can conclude that it is necessary to decompose all input, internal transformer and output layers to efficiently compress transformer for series prediction.

5 EXPERIMENTAL STUDIES

We evaluate the proposed models on one synthetic and three real-world trace datasets to validate the advantages of the proposed CFTT and transformer compared with conventional and state-of-the-art neural models.

5.1 Experimental Settings

5.1.1 Datasets. One synthetic trace dataset, one trace collected from a commercial hybrid storage device, and two public trace datasets from Microsoft with distinct properties are utilized:

- SPC-1 is synthesized by the synthetic benchmark SPC-1, which is widely used for performance measurement in storage systems. There are 40% read requests and 60% write requests in SPC-1. Besides, there are 40% chance for sequential read/write, and 60% chance for random read/write with temporal locality [5]. SPC-1 thus has a small part of sequential access with high frequency in Fig. 1c and a large part of random access with lower frequency in Fig. 1d and 1e.
- Hybrid-1 is collected from a real-world commercial hybrid storage device, whose access patterns are plotted in Fig. 1b.
- Microsoft-1 is an one week trace of the source control function in enterprise servers from Microsoft Research Cambridge [20] in 2008 and reconstructed by [13] in 2017 on flash-based devices. There are mainly seasonal and sequential access patterns in this trace as visualized in Fig. 1a.
- Microsoft-2 is from another volume similar to Microsoft-1.

Table 4: Data summary of selected trace datasets.

Datasets	Len	d	$\hat{d}$	Sparsity	Access patterns
SPC-1	424	30,720	3,796	8.1	Re-accessed + Random
Hybrid-1	647	102,400	2,339	43.8	Sequential + Random
Microsoft-1	360	7,680	341	22.5	Seasonal + Sequential
Microsoft-2	448	8,192	408	20.1	Seasonal + Sequential

5.1.2 Data Processing. The single (512B) LBA level requests are summarized along the temporal and spatial dimensions as the multivariate (4MB in our case) block level access frequency series. Specifically, the long LBA level streams are first split into Len stream chunks, and each stream chunk is also called a window and has a fixed number of LBA requests. The number of LBA requests in a window is set as 100,000 in our cases to balance model inference time against data placement response time. It is a adjustable setting depending on the actual storage device Input/Output operations Per Second (IOPS) and latency. Then the LBA level stream chunks are aggregated as d-dimensional vectors, which save access frequencies to all possible blocks. Concatenating Len vectors, we get a 2D block access frequency series $\mathbf{X} \in \mathbb{R}^{Len\times d}$. The first 80% and last 20% of samples are used for training and testing, respectively. Tab. 4 summarizes the details of these datasets. Len is the number of split request windows and the length of the whole multivariate sequence. d is the total number of accessed blocks in all windows and also the sequence dimension. $\hat{d}$ means the averaged number of accessed blocks in a window. Thee sparsity is $\frac{d}{\hat{d}}$ and indicates the redundancy of model parameters.

5.1.3 Methods for Comparison. We compare **seven** competing methods in three categories:

- *Two traditional univariate methods*: WDAC and CBF with shared weights for multivariate series.
- *Two multivariate neural networks*: LSTNet and TF-AR which is introduced by incorporating Transformer with an additional highway autoregressive module of LSTNet.
- *Three tensor networks*: TT-RNN, TT, and CFTT.

All models take the aggregated multivariate frequency series as input. CFTT with specific settings is defined as CFTT(d_e,L)([I_1,..., I_N],[J_1,..., J_N],[R_1,..., R_{N-1}]). The ranks R'_n of internal transformer TTLs are set as the ranks R_n of input TTL. Model hyperparameters of all these models are searched in grid for the best tier-related performance. Specifically, the best CFTT models are

- SPC-1: CFTT(2048,1)([8,12,20,16],[4,8,8,8],[8,8,8]),
- Hybrid-1: CFTT(1024,1)([256,400],[32,32], [16]),
- Microsoft-1: CFTT(1024,1)([120,64],[16,64],[16]),
- Microsoft-2: CFTT(1024,1)([64,128],[16,64],[64]).

5.1.4 Loss Function and Evaluation Criteria. The loss function is defined as Root Mean Squared Error (RMSE). Besides, we evaluate models with Mean Average Error (MAE) and two classification-like and tier-related criteria: top-k accuracy, and top-k hit rate. The top-k accuracy (ACC) is the ratio between the number of correctly predicted top-k blocks $\hat{k}$ and predefined k: $ACC = \frac{\hat{k}}{k}$ which treats all blocks equally like MAE. However, the access frequencies and top-k orders also matter in practice. If a block is labeled as the top-k data, it will be moved to upper tiers like SSDs to reduce access latency. It can be approximated with the top-k hit rate (HR):

$$HR = \frac{\sum_{i=1}^{k} \mathbf{X}_{T+1,i}}{\sum_{j=1}^{d} \mathbf{X}_{T+1,j}}, \tag{23}$$

where X_{T+1} is the true read frequency. k is the predefined value to simulate the corresponding upper tier volume k, and d is the number of available blocks. Therefore, $\sum_{i=1}^{k} \mathbf{X}_{T+1,i}$ is the sum of read frequency of the predicted top-k blocks, and $\sum_{j=1}^{d} \mathbf{X}_{T+1,j}$ is the sum of all read operations in the $T+1$-th window. After step T, k hot blocks with the top predicted future access frequency are prefetched into the upper tier whose size is k blocks (4MB).

5.1.5 Optimizer and Hardware. The AdamW optimizer [18] is used to train neural models, and parameters are set by default. We train and evaluate neural models in one machine in 2 NVIDIA V100 GPUs. Most neural models converge in 100 to 200 epochs and one epoch of training costs at most 1 to 2 seconds.

5.2 Experimental Results

Fig. 4 and 5 visualize the top-k accuracy and hit rate results in various cache size k settings. Tab. 5 summarizes the numbers of model parameters (#Params), compression ratios (CR), relative differences of regression performance (ΔMAE and ΔRMSE), averaged relative differences of the top-k accuracy ΔACC and hit rates ΔHR, and the max top-k hit rates (max HR). Units of ΔMAE, ΔRMSE, ΔACC, ΔHR, and max HR are %.

5.2.1 Comparison of Prediction Performance. As reported in Tab. 5, the compressed models CFTT, TT, and TT-RNN outperform LSTNet and TF-AR on the whole, while TF-AR and LSTNet show similar prediction results. Specifically, CFTT not only achieves significant compression ratios (up to 461 times than TF-AR), but also obtains better prediction performance on the high-dimensional Hybrid-1 and SPC-1 datasets with random patterns. For example, CFTT shows great advantages (20.68%) than TF-AR on MAE on the synthetic SPC-1 trace. On the Hybrid-1 dataset with more complex patterns, CFTT still outperforms TF-AR by 2.45% in MAE and 9.17% in RMSE.

5.2.2 Comparison of Top-k Accuracy. From Fig. 4 and Tab. 5, we can conclude that the neural prediction models, especially CFTT models, achieve better (**10.90% to 42.05%** on average) top-k accuracy than traditional univariate models (WDAC and CBF). Compressed models including TT-RNN, TT, and CFTT can further improve the top-k accuracy than full-parameter LSTNet and TF-AR.

5.2.3 Comparison of Top-k Hit Rates. Regarding to top-k hit rates, LSTNet, TF-AR, TT-RNN, TT, and CFTT can predict the most frequently accessed blocks with higher accuracy than traditional WDAC and CBF in general, as shown in Fig. 5. In detail, our introduced TF-AR shows stable advantages than LSTNet, TT-RNN, and TT. Furthermore, the proposed CFTT can dramatically reduce model parameters of TF-AR up to 461 times, achieving the even 2.05% higher top-k hit rate as stated in Tab. 5 and 7. These results are analysed in details as below.

Neural Models vs. Conventional Methods. Most blocks are accessed periodically, and there are mainly long sequential streams in the Microsoft-1 and Microsoft-2 datasets as in Fig. 1a. The simply (linearly or exponentially) decaying weights of WDAC and CBF are incapable of modeling those seasonal and shifting patterns, because decaying weights are suboptimal and these univariate approaches lack information of neighbouring blocks. In comparison, the multivariate neural models not only capture recent patterns, but also predict the adjacent blocks that will be accessed in the next window. The neural networks like our CFTT models thus achieve 16.09% and 21.55% higher top-k hit rates on average than LRU in Microsoft datasets, respectively.

The aggregation of LBA requests along spatial and temporal dimensions to high-level multivariate series makes WDAC, CBF, and neural models robust to the randomness. Despite that, these aggregated series are still noisy as visualized in Fig. 5b for the Hybrid-1 trace, Fig. 1d and 1e for the SPC-1 trace. For such trace datasets with random access patterns, the upper tier hit performance depends on the upper tier size. When the upper tier size k is sufficient, these random patterns can also be captured by keeping all blocks that are likely to be accessed in upper tier. In this way, the neural networks models only slightly outperform WDAC and CBF. In contrast, when k is limited, the neural models work much better than conventional models. For example, the top-500 hit rate of CFTT is 10.98% higher than CBF in the SPC-1 dataset.

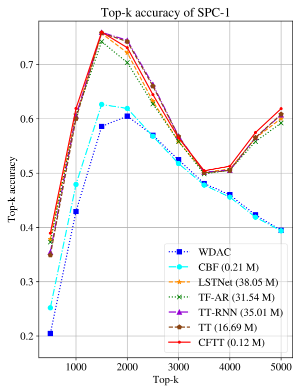
CFTT vs. Neural and Tensor Networks. CFTT and TF-AR always rank in the top and are more stable than LSTNet, TT-RNN, and TT. For instance, in the SPC-1 dataset, CFTT outperforms TF-AR, LSTNet, and TT by 2.05%, 4.69%, and 11.14%, with much less model paramters respectively in case of $k = 5000$. Similarly, TF-AR consistently achieves the most accurate top-k predictions for all k values in the Hybrid-1 dataset. We remark that TF-AR is our introduced combination of the general transformer model and autoregressive module. Here, TF-AR, TT-RNN, and TT slightly outperform our CFTT because TF-AR does not suffer from the inaccurate output address information caused by the output layer compression. However, they have hundreds of millions of parameters, making them inapplicable to be deployed in limited-resource storage systems. In comparison, CFTT can compress TF-AR up to 461 times, while obtaining similar or 2.05% better top-k hit rates.

5.2.4 Results of Model Compression. From Tab. 5 and 7, CFTT can usually compress TF-AR from at least 16 times in Microsoft traces, and even up to 220 and 461 times in Hybrid-1 and SPC-1 traces, respectively. With such effective compression results, CFTT does not sacrifice its performance and even obtains a slight improvement in top-k hit rates, as summarized in Tab. 5 and 7. Besides, the compression ratios of TT-RNN and TT are bounded at around 2, because these two methods only compress the input layers without considering the internal layers and output layer. While CFTT applies TTLs to compress all the layers of TF-AR, resulting in significantly less parameters than TT-RNN and TT. The gaps are up to 244 and 113 times in the SPC-1 and Hybrid-1 datasets, respectively.

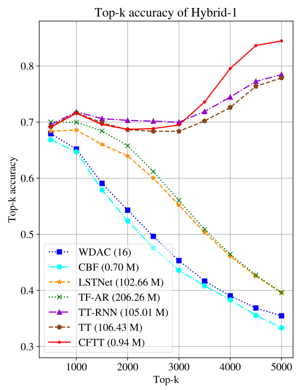
Table 5: Model comparison in term of model sizes, prediction errors, and top-k accuracy and hit rates. (We highlight the best results in bold font and underline the second best results.)

Models	SPC-1 (d = 30720)							Hybrid-1 (d = 102400)						
	#Params	CR	ΔMAE	ΔRMSE	ΔACC	ΔHR	MaxHR	#Params	CR	ΔMAE	ΔRMSE	ΔACC	ΔHR	max HR
WDAC	16	> 1K	/	/	-10.87	-5.88	83.01	16	> 1K	/	/	-7.66	-2.90	93.26
CBF	220,840	149.74	/	/	-9.56	-3.92	83.01	736,133	293.81	/	/	-9.01	-1.94	94.19
LSTNet	39,893,521	0.83	-19.67	-1.13	0.64	-0.57	83.79	107,648,060	2.01	0.00	7.13	-1.04	-1.00	95.31
TF-AR	33,067,537	1	0	0	0	0	86.43	216,280,141	1	0	0	0	**0**	**95.65**
TT-RNN	36,706,304	0.90	-20.43	1.45	1.11	-8.67	77.49	110,112,768	1.96	0.97	-5.16	15.34	-1.24	95.46
TT	17,500,672	1.89	-16.39	1.53	0.92	-8.88	77.34	111,600,640	1.94	1.31	**-12.28**	14.21	-1.88	94.63
CFTT	128,017	258.31	**-21.90**	**-1.29**	**1.55**	**1.74**	**88.48**	984,064	219.78	**-2.45**	-9.17	**16.74**	-1.89	94.82

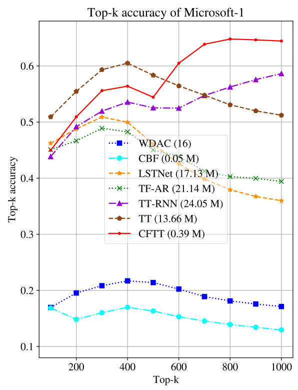
Models	Microsoft-1 (d = 7680)							Microsoft-2 (d = 8192)						
	#Params	CR	ΔMAE	ΔRMSE	ΔACC	ΔHR	max HR	#Params	CR	ΔMAE	ΔRMSE	ΔACC	ΔHR	max HR
WDAC	16	> 1K	/	/	-24.35	-34.56	59.23	16	> 1K	/	/	-37.36	-71.90	22.83
CBF	55,210	401.41	/	/	-28.75	-34.59	59.42	58,891	392.88	/	/	-40.29	-66.57	29.91
LSTNet	17,958,971	1.23	**-0.25**	**-11.41**	-0.37	-0.04	96.83	25,238,039	0.92	8.38	5.15	-4.60	-3.97	96.90
TF-AR	22,161,979	1	0	0	0	0	96.20	23,137,303	1	0	0	0	0	96.44
TT-RNN	25,220,608	0.88	36.55	10.07	9.20	-4.24	95.46	12,894,208	1.79	**-7.37**	**-6.70**	3.51	**0.69**	**97.31**
TT	14,323,200	1.55	14.91	7.75	11.33	**0.46**	**96.97**	14,996,480	1.54	-3.73	-3.20	**3.64**	0.37	96.09
CFTT	410,112	54.04	50.76	20.35	**14.18**	-0.66	96.00	1,397,760	16.55	3.73	4.56	1.76	0.32	96.53



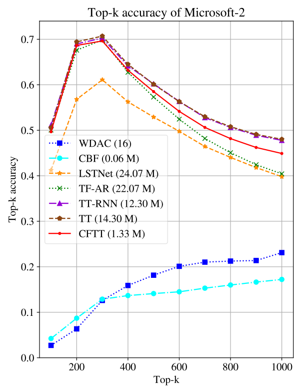
(a) **SPC-1** (d = 30720)



(b) **Hybrid-1** (d = 102400)

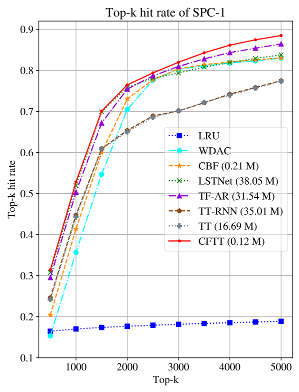


(c) **Microsoft-1** (d = 7680)

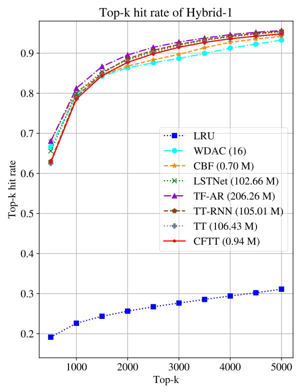


(d) **Microsoft-2** (d = 8192)

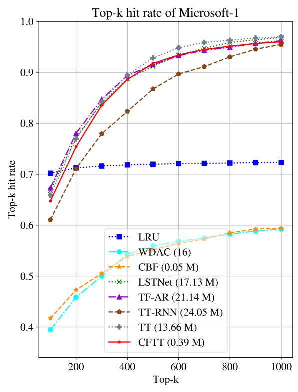
Figure 4: Top-k accuracy of best performed models on the four traces (Best view on screens with colors).



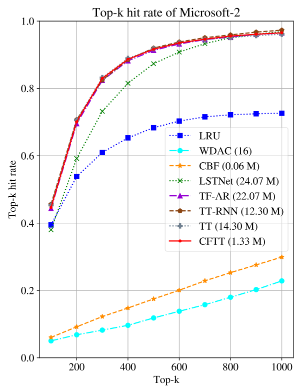
(a) **SPC-1** (d = 30720)



(b) **Hybrid-1** (d = 102400)



(c) **Microsoft-1** (d = 7680)



(d) **Microsoft-2** (d = 8192)

Figure 5: Top-k hit rates of best performed models on the four traces (Best view on screens with colors).

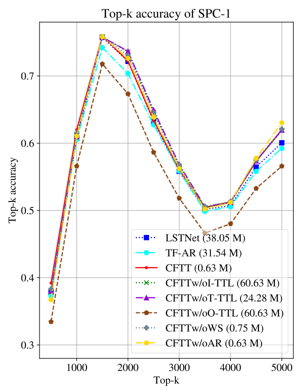
Table 6: Runtime comparison of TF-AR, TT, and CFTT

Models	#Params	CR	#Threads	Runtime(s)	Speedup
TF-AR	216,280,141	1	1	0.5207	1
			16	0.0883	1
TT	111,600,640	1.94	1	0.1978	2.63
			16	0.0381	2.32
CFTT	984,064	219.78	1	0.1845	2.82
			16	0.0508	1.74
CFTT-R8	381,952	566.24	1	0.0960	5.42
			16	0.0416	2.12

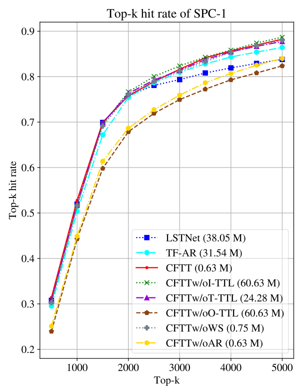
5.2.5 Comparison of runtime. In the most high-dimensional ($d = 102,400$) Hybrid-1 dataset, we compare the inference time cost of TF-AR, TT, and CFTT as stated in Tab. 6. The computation hardware is set as CPUs as most storage devices are not equipped with GPUs and computation resources are limited. From which, CFTT speeds up the inference 2.82 times than TF-AR with one CPU thread while improving memory usage 219.78 times. By reducing the rank of TTD to 8, we can even speed up 5.42 times with single thread. In comparison, TT only compresses the model 1.94 times, although it shows competitive inference performance. The runtime of CFTT is less than 0.2 second with one thread and about 0.05 second with 16 threads, which already meets the requirements for data placement of most hybrid storage systems whose read IOPS is less than 2 million [8]. Besides, for high IOPS and low latency systems, we can adjust the series aggregation settings or extend CFTT to make multi-step ahead predictions to improve timeliness [3].

5.3 Ablation Study

5.3.1 Impacts of model components. We first compare different model settings of CFTT in the SPC-1 dataset and evaluate the contribution of each component to the top-k accuracy and hit rates. The SPC-1 dataset is selected for ablation studies because it has more complex access patterns and results in gaps between CFTT and other models. Specifically, we remove the input TTL, transformer TTLs, output TTL, weight sharing, highway AR in CFTT to get CFTT model variants include CFTT, CFTTw/oI-TTL, CFTTw/oT-TTL, CFTTw/oO-TTL, CFTTw/oWS, and CFTTw/oAR, respectively.



(a) Top-k Accuracy



(b) Top-k hit rates

Figure 6: CFTT model comparison in the SPC-1 trace.

As visualized in Fig. 6a, removing the highway AR component as CFTTw/oAR leads to significant hit rate drops for all ks, which may be caused by the inaccuracy of modeling blocks with high-frequency and sequential access patterns. CFTTw/oO-TTL has even more parameters than the vanilla TF-AR with a smaller hidden size. The enormous parameters are hard to train with limited data, so CFTTw/oO-TTL has the lowest top-k accuracy and hit rates. Apart from these two models, other variants outperform the vanilla TF-AR. However, CFTTw/oI-TTL and CFTTw/oT-TTL have significantly more complexity than CFTT. Besides, weight sharing removes the need of storing and training the weights in the output TTL and further reduces model complexity by 14.1% with even higher hit rates, comparing CFTT and CFTTw/oWS.

Table 7: CFTT models with various TT shapes and ranks.

Models	#	CR	ΔMAE	ΔRMSE	ΔACC	ΔHR	max HR
TF-AR	33,067,537	1	0	0	0	0	86.43
CFTT-1	802,833	41.19	-20.84	-1.21	1.51	1.37	88.04
CFTT-2	240,657	137.41	-21.09	-1.29	1.27	1.65	88.33
CFTT-3	128,017	258.31	**-21.90**	-1.29	**1.55**	1.74	**88.48**
CFTT-4	71,697	461.21	-20.99	-1.29	1.46	**1.79**	88.33

5.3.2 Impacts of TT shapes and ranks. We also study the effects of TT shapes and ranks in CFTT in the SPC-1 dataset. The TT setting details of TF-AR and CFTT models are as follows

- TF-AR: TF-AR(512,1),
- CFTT-1: CFTT(2048,1)([240,128],[32,64],16),
- CFTT-2: CFTT(2048,1)([8,12,20,16],[8,8,4,8],[16,16,16]),
- CFTT-3: CFTT(2048,1)([8,12,20,16],[4,8,8,8],[8,8,8]),
- CFTT-4: CFTT(2048,1)([8,12,20,16],[8,8,4,8],[4,4,4]).

As stated in Tab. 7, we can achieve the 2.05% higher top-5000 hit rate with CFTT-3 while reducing parameters 258 times. By decreasing TT ranks R_n to 4 as CFTT-4, we further compress TF-AR by 461 times with only $71,697$ parameters and still obtain the 1.90% higher top-5000 hit rate and 1.79% higher averaged top-k hit rates than TF-AR. Comparing the last 4 CFTTs, we can conclude that CFTT models are robust to TT ranks. The larger TT ranks usually lead to better performance but more model complexity.

6 CONCLUSION

In this work, we are the first to apply transformer to model access patterns in the block and time window level. To reduce space and computational usage of transformer caused by high-dimensional and sparse data, we propose an efficient compression and prediction model CFTT by incorporating the advantages of tensor-train decomposition and transformer for time series prediction. Extensive experimental studies on both synthetic and real-world trace datasets demonstrate that 1) transformer-based methods show stable and excellent performance (up to 21.92%) than traditional methods; 2) CFTT saves space up to 461 times than the original transformer, while achieving 2.05% higher top-k hit rates than transformer especially for extremely high dimensional data; 3) Sharing weights among input and output TTLs can further compress CFTT by 14.1% with slight improvements of prediction performance; 4) A remarkable speedup is derived from CFTT, up to 5.42× with one CPU thread. These are of great significance for deployment in storage systems, where available space and computational resources are limited. In future work, we would further improve the performance of CFTT by auto-tuning TTL ranks and increase its scalability.

REFERENCES

[1] Janki Bhimani, Ningfang Mi, Zhengyu Yang, Jingpei Yang, Rajinikanth Pandurangan, Changho Choi, and Vijay Balakrishnan. 2018. FIOS: feature based I/O stream identification for improving endurance of multi-stream SSDs. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 17–24.

[2] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[3] Chandranil Chakraborttii and Heiner Litz. 2020. Learning I/O Access Patterns to Improve Prefetching in SSDs. *ECML-PKDD* (2020).

[4] Gil Einziger, Roy Friedman, and Ben Manes. 2017. Tinylfu: A highly efficient cache admission policy. *ACM Transactions on Storage (ToS)* 13, 4 (2017), 1–31.

[5] Binny S. Gill and Dharmendra S. Modha. 2005. SARC: Sequential Prefetching in Adaptive Replacement Cache. In *Proceedings of the 2005 USENIX Annual Technical Conference, April 10-15, 2005, Anaheim, CA, USA*. USENIX, 293–308.

[6] Milad Hashemi, Kevin Swersky, Jamie Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. 2018. Learning memory access patterns. In *ICML*. PMLR, 1919–1928.

[7] Shuibing He, Zheng Li, Jiang Zhou, Yanlong Yin, Xiaohua Xu, Yong Chen, and Xian-He Sun. 2020. A Holistic Heterogeneity-Aware Data Placement Scheme for Hybrid Parallel I/O Systems. *IEEE Trans. Parallel Distributed Syst.* 31, 4 (2020), 830–842.

[8] Antony S Higginson, Mihaela Dediu, Octavian Arsene, Norman W Paton, and Suzanne M Embury. 2020. Database Workload Capacity Planning using Time Series Analysis and Machine Learning. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 769–783.

[9] Jen-Wei Hsieh, Tei-Wei Kuo, and Li-Pin Chang. 2006. Efficient identification of hot data for flash memory storage systems. *ACM Transactions on Storage (TOS)* 2, 1 (2006), 22–40.

[10] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. In *ICML*. PMLR, 5156–5165.

[11] Valentin Khrulkov, Oleksii Hrinchuk, Leyla Mirvakhabova, and Ivan Oseledets. 2019. Tensorized embedding layers for efficient model compression. *arXiv preprint arXiv:1901.10787* (2019).

[12] Valentin Khrulkov, Oleksii Hrinchuk, and Ivan Oseledets. 2019. Generalized Tensor Models for Recurrent Neural Networks. In *ICLR*.

[13] Miryeong Kwon, Jie Zhang, Gyuyoung Park, Wonil Choi, David Donofrio, John Shalf, Mahmut Kandemir, and Myoungsoo Jung. 2017. TraceTracker: Hardware/software co-evaluation for large-scale I/O workload reconstruction. In *2017 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 87–96.

[14] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. 2018. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 95–104.

[15] Naihan Li, Shujie Liu, Yanqing Liu, Sheng Zhao, and Ming Liu. 2019. Neural speech synthesis with transformer network. In *AAAI*, Vol. 33. 6706–6713.

[16] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. 2019. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In *NIPS*. 5243–5253.

[17] Evan Liu, Milad Hashemi, Kevin Swersky, Parthasarathy Ranganathan, and Junwhan Ahn. 2020. An imitation learning approach for cache replacement. In *ICML*. PMLR, 6237–6247.

[18] Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).

[19] Arvind Narayanan, Saurabh Verma, Eman Ramadan, Pariya Babaie, and Zhi-Li Zhang. 2018. Deepcache: A deep learning based framework for content caching. In *Proceedings of the 2018 Workshop on Network Meets AI & ML*. 48–53.

[20] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. 2008. Write off-loading: Practical power management for enterprise storage. *ACM Transactions on Storage (TOS)* 4, 3 (2008), 1–23.

[21] A Novikov, D Podoprikhin, D Vetrov, and A Osokin. 2015. Tensorizing neural networks. In *NIPS*. 442–450.

[22] Ivan V Oseledets. 2011. Tensor-train decomposition. *SIAM Journal on Scientific Computing* 33, 5 (2011), 2295–2317.

[23] Dongchul Park and David HC Du. 2011. Hot data identification for flash-based storage systems using multiple bloom filters. In *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 1–11.

[24] Anh-Huy Phan, Konstantin Sobolev, Konstantin Sozykin, Dmitry Ermilov, Julia Gusak, Petr Tichavskỳ, Valeriy Glukhov, Ivan Oseledets, and Andrzej Cichocki. 2020. Stable low-rank tensor decomposition for compression of convolutional neural network. In *ECCV*. Springer, 522–539.

[25] Anna Shalova and Ivan Oseledets. 2020. Tensorized Transformer for Dynamical Systems Modeling. *arXiv preprint arXiv:2006.03445* (2020).

[26] Biaobiao Shen, Yongkun Li, Yinlong Xu, and Yubiao Pan. 2015. A light-weight hot data identification scheme via grouping-based LRU lists. In *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 88–103.

[27] Qiquan Shi, Jiaming Yin, Jiajun Cai, Andrzej Cichocki, Tatsuya Yokota, Lei Chen, Mingxuan Yuan, and Jia Zeng. 2020. Block Hankel tensor ARIMA for multiple short time series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 5758–5766.

[28] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732* (2020).

[29] Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. 2018. Tensor decomposition for compressing recurrent neural network. In *2018 IJCNN*. IEEE, 1–8.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *NIPS* 30 (2017), 5998–6008.

[31] Yinchong Yang, Denis Krompass, and Volker Tresp. 2017. Tensor-Train Recurrent Neural Networks for Video Classification. In *ICML*. 3891–3900.

[32] Rose Yu, Stephan Zheng, Anima Anandkumar, and Yisong Yue. 2017. Long-term forecasting using tensor-train rnns. *Arxiv* (2017).

[33] Giulio Zhou and Martin Maas. 2019. Multi-task learning for storage systems. In *Proc. ML Syst. Workshop*.

[34] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2020. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. *arXiv preprint arXiv:2012.07436* (2020).