AN INCREMENTAL TENSOR TRAIN DECOMPOSITION ALGORITHM

DORUK AKSOY*, DAVID J. GORSICH†, SHRAVAN VEERAPANENI‡, AND ALEX A. GORODETSKY*

Abstract. We present a new algorithm for incrementally updating the tensor train decomposition of a stream of tensor data. This new algorithm, called the *tensor train incremental core expansion* (TT-ICE) improves upon the current state-of-the-art algorithms for compressing in tensor train format by developing a new adaptive approach that incurs significantly slower rank growth and guarantees compression accuracy. This capability is achieved by limiting the number of new vectors appended to the TT-cores of an existing accumulation tensor after each data increment. These vectors represent directions orthogonal to the span of existing cores and are limited to those needed to represent a newly arrived tensor to a target accuracy. We provide two versions of the algorithm: TT-ICE and TT-ICE accelerated with heuristics (TT-ICE*). We provide a proof of correctness for TT-ICE and empirically demonstrate the performance of the algorithms in compressing large-scale video and scientific simulation datasets. Compared to existing approaches that also use rank adaptation, TT-ICE* achieves 57× higher compression and up to 95% reduction in computational time.

Key words. Tensor decompositions, data compression, streaming data, low-rank factorizations

AMS subject classifications. 15A23, 65-04, 15A69, 65F55

1. Introduction.

Tensors provide a representation for multivariate or high-dimensional data in many problems, including RGB images [14], social networks [24, 21], multisensory experiments [3], neuroscience [17, 19], and finance [11]. As the size of the tensor increases, computational processes become harder, if not impossible, due to the curse of dimensionality — both storage and computational processing requirements can grow exponentially with the number of dimensions. Tensor decompositions that rely on low-rank approximations provide a solution to mitigate the curse of dimensionality to help reduce computational demands.

In many applications, data becomes available incrementally, e.g., from a sequence of experiments [3], a video (which can be seen as a sequence of RGB images) [14], in IoT applications [6], or from simulations of physical systems [2]. As a result, batch tensor compression approaches that wait for all data to be collected may not be affordable because the data storage may exceed capacity. Furthermore, the total number of data points might not be known *a priori*. In this setting, it would be inefficient to reconstruct and decompose the tensor every time a new data point arrives. Incremental algorithms that update the decomposition without reconstructing the compressed tensor are needed.

Existing works include incremental tensor decomposition algorithms in Canonical Polyadic (CP) format [3, 8, 30, 25], Tucker format [10, 26, 4], and tensor train (TT) [22] format [27, 15, 31]. In this work, we focus on the TT-format. Unlike the CP-format, the TT-format avoids the NP-hard problem of computing the canonical rank [12] and offers controlled compression algorithms. Also, the storage of a tensor in TT-format scales linearly in the number of dimensions d, whereas in Tucker format, it scales exponentially in d. Furthermore, the TT-format offers an efficient way to execute basic linear algebra operations without reconstructing the full tensor, and therefore it is beneficial for computing with large-scale data.

In this work, we consider a *d-way tensor* $\mathcal{Y} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ to be a multidimensional array. A *tensor stream* (or alternatively, *stream of tensors*) is a sequence of d-way tensors $\mathcal{Y}^1, \mathcal{Y}^2, \ldots$, where each element in the sequence $\mathcal{Y}^k \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ is a d-dimensional tensor[1]. A finite stream of tensors is called an *accumulation* and can be viewed as a $(d+1)$-way tensor $\mathcal{X}^k \in \mathbb{R}^{n_1 \times \cdots \times n_d \times n_{d+1}^k}$ by concatenating the tensors in the stream along the last dimension. The problem we consider can then be summarized as follows.

PROBLEM 1. *Construct a scheme to update the approximation $\hat{\mathcal{X}}^k$ of the accumulation tensor $\mathcal{X}^k$ after every increment k in TT-format. The constructed scheme should maintain the guaranteed bounds on the error $\|\mathcal{X}^k - \hat{\mathcal{X}}^k\|_F$ for all k. Furthermore, this approximate accumulation should represent all tensor increments $\mathcal{Y}^\ell$ with error $\|\mathcal{Y}^\ell - \hat{\mathcal{Y}}^\ell\|_F \leq \varepsilon_{des}\|\mathcal{Y}^\ell\|_F$ for any $\ell \leq k$, where $\hat{\mathcal{Y}}^\ell$ can be extracted from $\hat{\mathcal{X}}^k$.*

*Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI, USA (doruk@umich.edu, goroda@umich.edu)

†Ground Vehicle Systems Center, U.S. Army, Warren, MI, USA (david.j.gorsich.civ@army.mil)

‡Department of Mathematics, University of Michigan, Ann Arbor, MI, USA (shravan@umich.edu)

DISTRIBUTION A. Approved for public release; distribution unlimited. OPSEC #7057

[1]Despite the conservative definition here, we generalize the notion of tensor stream to support batches of tensors in Section 3.

There are a few recent works for computing an incremental TT in this setting [27, 15]; however, they suffer from drawbacks that limit their scalability. For example, the TT-FOA [27] algorithm considers a fixed-rank approximation and uses an optimization procedure that cannot guarantee bounded errors on *all* tensors seen in the stream. The ITTD algorithm [15] (also see [7]), on the other hand, can guarantee a prescribed error tolerance and enables rank adaptation. However, this algorithm has computational challenges, which we show are due to inefficient preprocessing of new data and overly conservative rank growth when the new data is incorporated into the existing representation. Moreover, it does not maintain an orthonormal set of TT-cores and therefore cannot be used to project arbitrary new data to obtain its latent representation in TT-format.

Our contribution is a new incremental algorithm, the TT incremental core expansion (TT-ICE), for computing and updating the tensor train representation of an approximate accumulation $\hat{\mathcal{X}}^k$. Moreover, this approach is suitable for the online setting and never requires full storage or reconstruction of all data while providing a solution to Problem 1. Specifically, the new aspects of our approach include:

- efficiently updating the rank to accurately represent each new data point;
- controlling error growth and maintaining provably guaranteed bounds on the compression error of each tensor in the stream for all times, and
- a set of heuristic modifications that further significantly increase computational efficiency – this algorithm is called TT-ICE*.

These contributions are achieved by limiting the number of new vectors appended to the TT-cores of an existing accumulation tensor after each data increment. These vectors represent directions orthogonal to the span of the existing cores and are limited to those needed to represent a newly arrived tensor to a target error. Moreover, our theoretical results are empirically justified on video compression applications arising from video game data and from solutions to numerical PDEs. Our results indicate order-of-magnitude benefits in compression by TT-ICE over ITTD. We also demonstrate that TT-ICE works in cases where ITTD runs out of memory. Finally, we show that the heuristic version, TT-ICE*, can achieve $2.6 - 7.3\times$ speedup over TT-ICE, with only a negligible sacrifice in compression accuracy.

The rest of this paper is structured as follows. In Section 2, we present the foundational concepts behind the TT-format and discuss the existing literature on incremental TT-decompositions, in detail. In Section 3, we present the TT-ICE and TT-ICE* algorithms and prove the correctness of TT-ICE. In Section 4, we provide preliminary experiments using our proposed approach on physical and cyber-physical data.

2. Background.

In this section, we provide the necessary background for the TT-decomposition and review two existing approaches for incremental approximation in TT-format.

2.1. tensor train decomposition.

This section reviews the relevant background on tensors and the tensor train decomposition.

The mode-i *matricization* (or *unfolding*) of a d-way tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ reshapes the tensor into a matrix $A_{(i)}$ with size $n_1 \dots n_i \times n_{i+1} \dots n_d$. In this unfolding, all the modes up to the i-th mode are mapped into the rows of the matrix and all other modes are mapped to the columns.

The *contraction* between two tensors $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ and $\mathcal{B} \in \mathbb{R}^{n_d \times n_{d+1} \times \cdots \times n_D}$ along the d-th dimension of $\mathcal{A}$ and the first dimension of $\mathcal{B}$ is a binary operation represented as

$$\mathcal{C} = \mathcal{A} \ {}_d\times_1 \ \mathcal{B}, \quad \text{where} \quad \mathcal{C}(i_1, i_2, \dots, i_{d-1}, i_{d+1}, \dots, i_D) = \sum_{j=1}^{n_d} \mathcal{A}(i_1, \dots, i_{d-1}, j)\, \mathcal{B}(j, i_{d+1}, \dots, i_D) \tag{1}$$

and the output $\mathcal{C} \in \mathbb{R}^{n_1 \times \cdots \times n_{d-1} \times n_{d+1} \times \cdots \times n_D}$ becomes a $(D-2)$-way tensor. The subscripts on either side of the $\times$ sign indicate the contraction axes of the tensors on their respective sides.

A d-way tensor $\mathcal{Y} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ is said to be in the *TT-format* when it is represented by a sequence of contractions of 3-way tensors $\mathcal{G}_i \in \mathbb{R}^{r_{i-1} \times n_i \times r_i}$, for $i = 1, \dots, d$, according to

$$\mathcal{Y} = \mathcal{G}_1 \ {}_3\times_1 \ \mathcal{G}_2 \ {}_3\times_1 \ \cdots \ {}_3\times_1 \ \mathcal{G}_d. \tag{2}$$

The tensors $\mathcal{G}_i$ are called *TT-cores* and the r_i, $i = 0, \dots, d$ are called *TT-ranks* with $r_0 = r_d = 1$. The TT-ranks are equal to the ranks of a sequential set of unfoldings [22]. Specifically, the i-th TT-rank r_i is

equal to the rank of the mode-i unfolding matrix $Y_{(i)}$. In practice, the unfolding matrices are rarely exactly low-rank, and instead, an *approximate* TT-representation is computed.

In this case, a rank-r_i truncated SVD of the i-th unfolding satisfies

$$Y_{(i)} = U_i \Sigma_i V_i + E_i, \tag{3}$$

where $U_i \in \mathbb{R}^{m \times r_i}$, $\Sigma_i \in \mathbb{R}^{r_i \times r_i}$, $V_i \in \mathbb{R}^{r_i \times \ell}$ are the factors of a rank-r_i truncated SVD, and $E_i \in \mathbb{R}^{m \times \ell}$ is the residual. The dimensions of the unfolding require $m = \prod_{j=1}^{i} n_j$, and $\ell = \prod_{j=i+1}^{d} n_j$. The Eckart-Young-Mirsky Theorem guarantees that the truncated SVD provides the best rank-r_i approximation of the matrix in the Frobenius norm, and the reconstruction error can be bounded by the remaining singular values according to $\|Y_{(i)} - U_i \Sigma_i V_i\|_F = \|E_i\|_F = \sqrt{\sum_{j=r_i+1}^{\min(m,n)} (\sigma_j^{(i)})^2}$, where $\sigma_j^{(i)}$ is the j-th singular value of the mode-i unfolding [9, 18].

The proof for the TT-ranks [22, Th. 2.1], is constructive and provides an algorithm, called TT-SVD [22, Alg. 1], to compress a tensor to a target accuracy. For a d-way tensor $\mathcal{Y} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, TT-SVD begins with a truncated SVD of the mode-1 unfolding $Y_{(1)} \in \mathbb{R}^{n_1 \times n_2 \ldots n_d}$

$$Y_{(1)} = U_1 \Sigma_1 V_1 + E_1, \tag{4}$$

with orthonormal left singular vectors $U_1 \in \mathbb{R}^{n_1 \times r_1}$ and orthogonal $\Sigma_1 V_1 \in \mathbb{R}^{r_1 \times n_2 \ldots n_d}$. Then, $\Sigma_1 V_1$ needs to be compressed. This process uses the fact that orthonormality of the left singular vectors guarantees $U_1^T E_1 = 0$, so that multiplying $Y_{(1)}$ on the left by U_1 leads to

$$U_1^T Y_{(1)} = \Sigma_1 V_1. \tag{5}$$

Now, the matrix $\Sigma_1 V_1$ can be reshaped to form $\mathcal{Z} \in \mathbb{R}^{r_1 n_2 \times n_3 \times \cdots \times n_d}$, and a truncated SVD is computed for its mode-1 unfolding

$$Z_{(1)} = U_2 \Sigma_2 V_2 + E_2, \tag{6}$$

with $U_2 \in \mathbb{R}^{r_1 n_2 \times r_2}$ and $\Sigma_2 V_2 \in \mathbb{R}^{r_2 \times n_3 \ldots n_d}$. The process then repeats analogous to Equations (5) and (6) until all dimensions are considered. The left singular vectors $U_i \in \mathbb{R}^{r_{i-1} n_i \times r_i}$ of each decomposition become the first $d-1$ TT-cores through reshaping

$$\mathcal{G}_i = \texttt{reshape}\left(U_i, [r_{i-1}, n_i, r_i]\right) \in \mathbb{R}^{r_{i-1} \times n_i \times r_i}. \tag{7}$$

The final core consists of the right singular vectors scaled with their respective singular values from the final truncated SVD.

The truncation errors for each unfolding described above need to be carefully chosen to ensure a guaranteed relative error bound $\varepsilon \in [0, 1]$ with respect to the Frobenius norm. Specifically, choosing a dimension-dependent truncation error according to $\delta = \frac{\varepsilon}{\sqrt{d-1}} \|\mathcal{Y}\|_F$ ensures that the computed TT-approximation $\hat{\mathcal{Y}}$ has a relative error less than ε, i.e. $\|\mathcal{Y} - \hat{\mathcal{Y}}\|_F \leq \varepsilon \|\mathcal{Y}\|_F$ [22, Thm. 2.2].

Once a TT is computed, basic algebraic operations, such as addition and multiplication, can be computed in closed form. For example, two d-way tensors in TT-format with cores $\{\mathcal{G}_i\}_{i=1}^d$ and $\{\mathcal{H}_i\}_{i=1}^d$ can be added if both of the tensors have the size $n_1 \times \cdots \times n_d$. The resulting tensor $\mathcal{D} = \mathcal{G} + \mathcal{H}$ is calculated using the following rule

$$\mathcal{D}_m(j_m) = \begin{cases} \begin{bmatrix} \mathcal{G}_1(j_1) & \mathcal{H}_1(j_1) \end{bmatrix} & , \ m = 1; \\ \begin{bmatrix} \mathcal{G}_m(j_m) & 0 \\ 0 & \mathcal{H}_m(j_m) \end{bmatrix} & , \ m = 2, \ldots d-1; \\ \begin{bmatrix} \mathcal{G}_d(j_d) \\ \mathcal{H}_d(j_d) \end{bmatrix} & , \ m = d, \end{cases} \tag{8}$$

where $\mathcal{G}_m(j_m) \in \mathbb{R}^{r_{m-1} \times r_m}$ denotes the j_m-th slice of the m-th TT-core $\mathcal{G}_m$. If $\mathcal{G}$ has TT-ranks r_i and $\mathcal{H}$ has TT-ranks $\bar{r}_i$, the sum $\mathcal{D}$ will have the TT-ranks $\hat{r}_i = r_i + \bar{r}_i$ with the exception of $\hat{r}_0 = \hat{r}_d = 1$.

2.2. Existing incremental tensor decompositions. This section discusses two existing approaches for solving Problem 1: the First-Order Adaptive Tensor Train Decomposition (TT-FOA) [27] and the Incremental Tensor Train Decomposition (ITTD) [15][2].

There are two main steps of incremental procedures: (1) preprocessing the newly arrived tensor $\mathcal{Y}^{k+1} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ and (2) updating the TT-cores of the previous accumulation tensor $\mathcal{X}^k$ to obtain new TT-cores for $\mathcal{X}^{k+1}$ with this information. Below we describe how TT-FOA and ITTD handle these two steps.

Preprocessing $\mathcal{Y}^{k+1}$. At time $k+1$, TT-FOA poses the problem of preprocessing the new tensor $\mathcal{Y}^{k+1}$ using the TT-cores of the approximate accumulation $\hat{\mathcal{X}}^k$ as the following regularized optimization problem

$$g^{k+1} = \underset{g \in \mathbb{R}^{r_d \times 1}}{\operatorname{argmin}} \|\mathcal{Y}^{k+1} - \hat{\mathcal{X}}^k \ {}_{(d+1)}\times_1 g\|_F^2 + \frac{\rho}{2}\|g\|_2^2, \tag{9}$$

where g^{k+1} is the representation of $\mathcal{Y}^{k+1}$ as the $(d+1)$-th TT-core of the accumulation $\mathcal{X}^k$, ρ is a small regularization parameter, and the contraction of first d TT-cores $\hat{\mathcal{X}}^k \in \mathbb{R}^{n_1 \times \cdots \times n_d \times r_d}$ is computed as

$$\hat{\mathcal{X}}^k = \mathcal{G}_1 \ {}_3\times_1 \ \cdots \ {}_3\times_1 \ \mathcal{G}_d. \tag{10}$$

Note that these first d cores are common for all compressed data, and only the final core is unique to a particular datum. In other words, one can view the first d cores as the basis of approximation that is common to all the data and the final core as the coefficients of this basis that are specific to each datum. Without the regularization term, the optimal solution g^{k+1} would be the projection of $\mathcal{Y}^{k+1}$ onto the basis defined by $\hat{\mathcal{X}}^k$. Thus, any component of the new data not represented by the existing accumulation would be discarded. The regularization biases the solution to be closer to zero.

In [27], the optimization problem is solved using the randomized sketching technique [16]. The closed-form solution is

$$g^{k+1} = \left(\mathcal{L}(\hat{X}^k_{(d)})^T \mathcal{L}(\hat{X}^k_{(d)}) + \rho I_{r_d}\right)^{-1} \mathcal{L}(\hat{X}^k_{(d)})^T \mathcal{L}(y^{k+1}), \tag{11}$$

with $\hat{X}^k_{(d)} \in \mathbb{R}^{n_1 \cdots n_d \times r_d}$ as the mode-d unfolding of $\hat{\mathcal{X}}^k$, $y^{k+1} \in \mathbb{R}^{n_1 \cdots n_d}$ as the reshaping of $\mathcal{Y}^{k+1}$ into a vector, and $\mathcal{L}(\cdot)$ as the sketching map that samples the same set of rows from $\hat{X}^k_{(d)}$ and y^{k+1}.

ITTD follows another approach. It reshapes the streamed d-way tensor $\mathcal{Y}^{k+1}$ to be a $d+1$-way tensor with size $n_1 \times \cdots \times n_d \times 1$ and computes the TT-decomposition of the reshaped tensor using the TT-SVD algorithm, without leveraging any information from the accumulation tensor. As a result, it requires the computation of a new sequence of singular value decompositions to discover an entirely new basis — a computation step not needed by TT-FOA or our proposed approach.

Updating the TT-approximation of $\mathcal{X}^k$ to that of $\mathcal{X}^{k+1}$. TT-FOA assumes fixed TT-ranks and adopts a gradient descent procedure to update the TT-cores of $\mathcal{X}^k$. With this fixed-rank assumption, TT-FOA cannot guarantee a predefined upper bound for representation error except in the limit. TT-FOA uses the following optimization objective to update the TT-cores of $\mathcal{X}^k$

$$\mathcal{G}_i = \underset{\mathcal{G} \in \mathbb{R}^{r_{i-1} \times n_i \times r_i}}{\operatorname{argmin}} \sum_{j=1}^{k+1} \lambda^{k+1-j} \|\mathcal{Y}^j - \mathcal{A}_i \ {}_{(i+1)}\times_1 \ \mathcal{G} \ {}_{(i+2)}\times_1 \ \mathcal{B}_i^j\|_F^2, \tag{12}$$

where

$$\begin{aligned} \mathcal{A}_i &= \mathcal{G}_1 \ {}_3\times_1 \ \cdots \ {}_3\times_1 \ \mathcal{G}_{i-1}, \\ \mathcal{B}_i^j &= \mathcal{G}_{i+1} \ {}_3\times_1 \ \cdots \ {}_3\times_1 \ \mathcal{G}_d \ {}_3\times_1 \ g^j, \end{aligned} \tag{13}$$

[2]There are two other methods in the literature that add additional functionality to the TT-FOA framework. ATT [29] extends TT-FOA by providing support for tensor streams with incomplete data, and ROBOT [28] extends ATT by providing robustness to the outliers in the streamed tensors. However, we only consider TT-FOA in our analysis since both ATT and ROBOT build onto TT-FOA algorithm, and we do not consider missing data in Problem 1.

and $\lambda \in (0, 1]$ is the forgetting factor discounting the effect of previous tensors. Note that only the superscript of the forgetting factor λ is for exponentiation. g^j denotes the coefficient vector representing $\mathcal{Y}^j$ in the $(d+1)$-th TT-core. Note that the objective function updates the i-th core $\mathcal{G}_i$ independent from the other cores and allows updating the cores in parallel. The TT-FOA algorithm provides a recursive approach to efficiently update this core, and we leave further details to [27].

Next, we describe how ITTD updates the accumulated tensor. Recall that at ITTD, this update is preceded by reshaping the d-way tensor $\mathcal{Y}^{k+1}$ into a $d+1$-way tensor with the $(d+1)$-th dimension as 1 and calculating an independent TT-decomposition for $\mathcal{Y}^{k+1}$. ITTD then merges the newly compressed tensor with the existing accumulation through a specific addition operation. To this end, the TT-core of $\mathcal{Y}^{k+1}$ corresponding to the growing dimension is padded with zeros so that the growing dimension becomes $(k+1)$. The same padding procedure is repeated for the TT-core of $\mathcal{X}^k$ so that the growing dimension becomes $(k+1)$. This zero-padding process is done to ensure the dimensional consistency between the two TT-representations. While padding zeros to the TT-cores, the ITTD algorithm places the padding tensors so that both tensors can be added without interfering with each other. ITTD algorithm then combines the TT-cores of the streamed tensor $\mathcal{Y}^{k+1}$ and the existing TT-cores of $\mathcal{X}^k$ by adding them in TT-format using the rule shown in (8). After the addition of these two tensors in TT-format, the TT-ranks are the sum of the component ranks. Therefore, an optional final step of the ITTD algorithm, the TT-rounding procedure of [22, Alg 2] is executed to reorthogonalize and recompress the TT-cores after addition.

2.3. Limitations of existing approaches. This section discusses the shortcomings of the existing methods in the literature.

The main limitation of TT-FOA is that the TT-representation error cannot be guaranteed for two reasons. First, due to the random initialization and the recursive update scheme, the TT-FOA algorithm starts with a high representation error. As time progresses and new observations become available, this error converges to a steady-state value. However, the lower bound of this steady-state value depends on the second limitation – that the TT-ranks of the accumulation are fixed as an input to the TT-FOA algorithm. If the selected TT-ranks are insufficient to represent the data accurately, there is no choice but to restart the entire TT-FOA algorithm with higher TT-ranks. Furthermore, the forgetting factor influences the TT-cores of the accumulation, so that representing the recently streamed tensors is more important than the earlier parts of the accumulation. This results in a loss of representation accuracy if the information in the tensor stream varies over time.

On the other hand, the approach we propose does not require time to converge to a steady-state error for an accumulation since truncated SVD computes the best-low-rank approximation. Furthermore, when the current TT-ranks of the decomposed accumulation are not sufficient to represent the streamed tensor $\mathcal{Y}^{k+1}$ up to a prescribed precision, our approach increases the TT-ranks accordingly to meet the precision criterion. Most importantly, our approach guarantees that the whole accumulation is represented within their prescribed precision bounds for all times.

The limitation of the ITTD algorithm stems from the inefficiencies of using the existing TT-approximation of accumulation $\mathcal{X}^k$ to aid the incorporation of the new data tensor. If the streamed data is highly structured, the TT-SVD step in ITTD is likely to return TT-cores that are redundant with the TT-cores of the accumulation. Therefore, the addition in TT-format likely results in linearly dependent TT-cores and superfluous TT-ranks that make TT-rounding an obligatory step for adequate compression.

In the case of an unbounded tensor stream, such a rounding procedure is required; otherwise, the TT-cores would grow without bound. However, the rounding procedure has a complexity that scales with the cube of the TT-ranks. This leads to a vicious cycle between rank growth and speed for ITTD. Putting off TT-rounding allows faster execution time, but the rank growth causes a cubic increase in time for future rounding attempts. On the other hand, avoiding rounding results in unbounded rank growth, which makes the rounding procedure impossible due to memory limitations.

We provide a more controlled rank growth by using the TT-cores of $\mathcal{X}^k$ as a foundation and expanding them only with complementing information obtained through $\mathcal{Y}^{k+1}$. Furthermore, for the edge case where the current TT-cores of the accumulation can represent the streamed data sufficiently accurate, our method can skip updating the TT-cores of the accumulation.

In the next section, we present our incremental TT algorithm, which provides a solution to all the

shortcomings of existing incremental TT-decomposition algorithms in detail.

3. Methodology: the TT-ICE algorithm. In this section, we present a new incremental TT-decomposition algorithm. Similar to the existing work, we tackle the problem of computing the incremental TT-decomposition of tensor streams in two steps: preprocessing and update. However, a distinguishing feature of our approach is that it processes and updates each dimension sequentially in a single pass. Furthermore, the subsequent algorithm considers a slightly generalized setting in which n_{d+1}^{k+1} tensors become available during the $k+1$-th increment. Equivalently, this can view the new data at increment $k+1$ as a tensor with an additional dimension $\mathcal{Y}^{k+1} \in \mathbb{R}^{n_1 \times \cdots \times n_d \times n_{d+1}^{k+1}}$.

3.1. Overview. This section walks through the steps of TT-ICE and presents a pseudocode for TT-ICE.

The proposed algorithm updates the TT-cores $\{\mathcal{G}_i^k\}_{i=1}^{d+1}$ of the approximate accumulation tensor $\hat{\mathcal{X}}^k$ to obtain updated cores $\{\mathcal{G}_i^{k+1}\}_{i=1}^{d+1}$ of the approximate accumulation tensor $\hat{\mathcal{X}}^{k+1}$. Let $\{U_i^k\}_{i=1}^{d+1}$ denote the reshapings of the cores at increment k according to Equation (7), these will then be updated to obtain $\{U_i^{k+1}\}_{i=1}^{d+1}$. At each increment, these matrices will be orthonormal, and the update will ensure they remain orthonormal. Similarly, we will update the ranks $\{r_i^k\}_{i=0}^{d+1}$ to $\{r_i^{k+1}\}_{i=0}^{d+1}$. The approach is provided in Algorithm 3.1, and we describe each step next.

Algorithm 3.1 `TT-ICE`: Incremental update of a tensor train decomposition

Input
- $\{U_i^k\}_{i=1}^{d+1}$ reshaped cores of the TT-decomposition of the accumulation $\mathcal{X}^k$
- $\mathcal{Y}^{k+1} \in \mathbb{R}^{n_1 \times \cdots \times n_d \times n_{d+1}^{k+1}}$ new tensor
- $\{\epsilon_i\}_{i=1}^{d}$ SVD truncation tolerances

Output
- $\{U_i^{k+1}\}_{i=1}^{d+1}$ updated cores for the accumulation $\mathcal{X}^{k+1}$

Preprocessing the first dimension

$Y_1 = \texttt{reshape}\left(\mathcal{Y}^{k+1}, [n_1, n_2 \dots n_{d+1}^{k+1}]\right)$

$R_1^k = \left(I - U_1^k {U_1^k}^T\right) Y_1$

$U_1^{k,pad} \leftarrow U_1^k$ ▷ First core has no padding

for $i = 1$ to $d-1$ **do**

  Updating i-th core $\mathcal{G}_i^k \to \mathcal{G}_i^{k+1}$

  $U_{R_i}^k, r_{R_i}, V_{R_i}^k \leftarrow \texttt{SVD}\left(R_i^k, \epsilon_i\right)$ ▷ $U_{R_i} \in \mathbb{R}^{r_{i-1}^k n_i \times r_{R_i}}$

  $U_i^{k+1} \leftarrow \begin{bmatrix} U_i^{k,pad} & U_{R_i}^k \end{bmatrix}$

  $U_{i+1}^{k,pad} \leftarrow \texttt{reshape}\left(\begin{bmatrix} \texttt{reshape}\left(U_{i+1}^k, [r_i^k, n_{i+1} r_{i+1}^k]\right) \\ \mathbf{0}_{r_{R_i} \times n_{i+1} r_{i+1}^k} \end{bmatrix}, [r_i^{k+1} n_{i+1}, r_{i+1}^k]\right)$ ▷ Pad for rank compat.

  Preprocessing the subsequent dimensions of $\mathcal{Y}^{k+1}$

  $Y_{i+1} \leftarrow \texttt{reshape}\left({U_i^{k+1}}^T Y_i, [r_i^{k+1} n_{i+1}, n_{i+2} \cdots n_{d+1}^{k+1}]\right)$

  $R_{i+1}^k \leftarrow \left(I - U_{i+1}^{k,pad} {U_{i+1}^{k,pad}}^T\right) Y_{i+1}$

end for

Updating the d-th core

$U_{R_d}^k, r_{R_d}, V_{R_d}^k \leftarrow \texttt{SVD}\left(R_d^k, \epsilon_d\right)$

$U_d^{k+1} \leftarrow \begin{bmatrix} U_d^{k,pad} & U_{R_d}^k \end{bmatrix}$

Updating the last core

$Y_{d+1} \leftarrow {U_d^{k+1}}^T Y_d$ ▷ No need to reshape for Y_{d+1} since ${U_d^{k+1}}^T Y_d \in \mathbb{R}^{r_d^{k+1} \times n_{d+1}^{k+1}}$

$U_{d+1}^{k+1} \leftarrow \begin{bmatrix} U_{d+1}^{k,pad} & Y_{d+1} \end{bmatrix}$

Preprocessing the first dimension of $\mathcal{Y}^{k+1}$. The first dimension is processed by projecting the reshaped tensor $Y_1 \equiv Y_{(1)}^{k+1}$ onto the orthogonal complement of the space spanned by the first dimension U_1^k through

$$R_1^k = Y_{(1)}^{k+1} - U_1^k {U_1^k}^T Y_{(1)}^{k+1} = \left(I - U_1^k {U_1^k}^T\right) Y_1, \quad R_1^k \in \mathbb{R}^{n_1 \times n_2 \cdots n_{d+1}^{k+1}}, \tag{14}$$

as provided in Line 9. The column-space of this residual is orthogonal to U_1^k, and the corresponding basis is extracted in the updated phase for each dimension via the truncated SVD.

Updating the i-th core $\mathcal{G}_i^k \to \mathcal{G}_i^{k+1}$. The update step now uses both the residual R_i^k and an intermediate, padded version of the i-th core $U_i^{k,pad}$ to compute the updated core U_i^{k+1}. First the residual is decomposed via the truncated SVD

$$R_i^k = U_{R_i}^k \Sigma_{R_i}^k V_{R_i}^k + E_i^k = U_{R_i}^k V_{R_i}^* + E_i^k, \quad \|E_i^k\|_F = \epsilon_i, \tag{15}$$

where the second equality multiplies the singular value matrix and the right singular vectors. The left singular vectors are orthogonal to $U_i^{k,pad}$, so the updated basis can be obtained by appending these new directions as in Line 14

$$U_i^{k+1} = \begin{bmatrix} U_i^{k,pad} & U_{R_i}^k \end{bmatrix} \in \mathbb{R}^{r_{i-1}^{k+1} n_i \times (r_i^{k+1})}, \tag{16}$$

where the new TT-rank becomes $r_i^{k+1} = r_i^k + r_{R_i}$.

The updated TT-core becomes $\mathcal{G}_i^{k+1} = \texttt{reshape}\left(U_i^{k+1}, \left[r_{i-1}^{k+1}, n_i, r_i^{k+1}\right]\right)$. Note that this update increases the rank between dimensions i and $i+1$ from r_i^k to r_i^{k+1}. This new dimension causes a discrepancy that prohibits the contraction between the i-th and $i+1$-th cores. Therefore, the $i+1$-th core must be padded with zeros

$$U_{i+1}^{k,pad} \leftarrow \texttt{reshape}\left(\begin{bmatrix} \texttt{reshape}\left(U_{i+1}^k, [r_i^k, n_{i+1} r_{i+1}^k]\right) \\ \mathbf{0}_{r_{R_i} \times n_{i+1} r_{i+1}^k} \end{bmatrix}, [r_i^{k+1} n_{i+1}, r_{i+1}^k]\right). \tag{17}$$

Note that this padding leaves $U_{i+1}^{k,pad}$ as orthonormal and does not change the span defined by the components of the first r_i^k rows of the reshaping.

After this next core, we have a partially updated representation of the accumulation for $\hat{\mathcal{X}}^{k+1}$

$$\hat{\mathcal{X}}^{k+1} = \mathcal{G}_1^{k+1} \,{}_3\times_1 \cdots {}_3\times_1 \mathcal{G}_i^{k+1} \,{}_3\times_1 \mathcal{G}_{i+1}^{k,\text{pad}} \,{}_3\times_1 \mathcal{G}_{i+2}^k \,{}_3\times_1 \cdots {}_3\times_1 \mathcal{G}_{d+1}^k, \tag{18}$$

where $\mathcal{G}_{i+1}^{k,\text{pad}}$ refers to the padded core formed by reshaping $U_{i+1}^{k,pad}$ following the update in (17). After this step, we can proceed to pre-process in preparation for updating the next dimension.

Preprocessing the subsequent dimensions of $\mathcal{Y}^{k+1}$. Preprocessing preparation for the update of the next dimension begins by updating the projection of the new data onto the updated core and reshaping

$$Y_{i+1} = \texttt{reshape}\left({U_i^{k+1}}^T Y_i, [r_i^{k+1} n_{i+1}, n_{i+2} \cdots n_{d+1}^{k+1}]\right). \tag{19}$$

This projection uses the *updated* tensor cores U_i^{k+1} and therefore has dimensions corresponding to the updated rank r_i^{k+1}. Then, the residual with respect to the span of the next core is obtained via a projection onto the orthogonal complement to the existing space

$$R_i^k = \left(I - U_i^{k,pad} {U_i^{k,pad}}^T\right) Y_{i+1}. \tag{20}$$

Updating the last core. The procedure in (19)-(17) is repeated sequentially for the first d dimensions. Then after updating $U_d^{k,pad}$ to U_d^{k+1} and padding U_{d+1}^k with zeros as in (17), projecting Y_d onto U_d^{k+1} as in Line 24 yields Y_{d+1}, with which we can update $U_{d+1}^{k,pad}$ as

$$U_{d+1}^{k+1} \leftarrow \begin{bmatrix} U_{d+1}^{k,pad} & Y_{d+1} \end{bmatrix}, \tag{21}$$

and obtain the representation for $\hat{\mathcal{X}}^{k+1}$ in TT-format.

3.2. Analysis. This section provides proof that TT-ICE can achieve and maintain a target accuracy throughout the compression process.

We first show that Algorithm 3.1 provides guaranteed reconstruction error of a new tensor given appropriate truncation of the truncated SVDs. Following the completion of TT-ICE, the approximation of the most recently compressed data point can be obtained by

$$\hat{\mathcal{Y}}^{k+1} = \mathcal{G}_1^{k+1} \,{}_3\times_1 \cdots {}_3\times_1 \mathcal{G}_d^{k+1} \,{}_3\times_1 \mathcal{G}_{d+1}^{k+1}[-n_{d+1}^{k+1}:], \tag{22}$$

where the index $[-n_{d+1}^{k+1}:]$ denotes the last n_{d+1}^{k+1} columns of the core, as seen in Equation (21). Recall that $\mathcal{G}_{d+1}^{k+1}$ is a matrix and that the columns $\mathcal{G}_{d+1}^{k+1}[-n_{d+1}^{k+1}:]$ are the only elements of the updated accumulation that are unique to the $(k+1)$-th data point. We now show that the algorithm enables a well-controlled approximation error of this latest tensor.

THEOREM 2. *Let $\{U_i^k\}_{i=1}^{d+1}$ denote the unfolded TT-cores of the approximate accumulation $\hat{\mathcal{X}}^k$, $\mathcal{Y}^{k+1}$ be a new streamed tensor and $\{\epsilon_i\}_{i=1}^{d}$ be the SVD truncation tolerances, then the TT-ICE algorithm computes a TT-approximation $\hat{\mathcal{Y}}^{k+1}$ (22) satisfying*

$$\|\mathcal{Y}^{k+1} - \hat{\mathcal{Y}}^{k+1}\|_F \leq \sqrt{\sum_{i=1}^{d} \epsilon_i^2}. \tag{23}$$

Proof. The proof is similar to that of [22, Thm 2.2] and is by induction. For $d = 1$, the statement follows from the properties of the truncated SVD.

Now we consider an arbitrary $d > 1$. The incremental updates start with the first dimension. We start with processing the first core, which works on the first unfolding of the new data. First, we show that computing the SVD of the residual leads to a low-rank approximation of $Y_{(1)}^{k+1}$ with an approximation error bounded by a controlled truncation tolerance. The definitions of the residual and projection given by Equations (14) lead to

$$Y_{(1)}^{k+1} = U_1^k U_1^{k^T} Y_{(1)}^{k+1} + R_1^k = U_1^k U_1^{k^T} Y_1 + U_{R_1}^k V_{R_1}^k + E_1^k = U_1^{k+1} \underbrace{\begin{bmatrix} U_1^{k^T} Y_1 \\ V_{R_1}^k \end{bmatrix}}_{B_1} + E_1^k = U_1^{k+1} B_1 + E_1^k, \tag{24}$$

where the third equality appends the existing basis U_1^k and the new directions obtained from the SVD of the residual $U_{R_1}^k$ to form the updated core U_1^{k+1}. Thus, after the first dimension is processed, the first core of $\mathcal{G}_1^{k+1}$ is obtained, as a reshaping of U_1^{k+1}. Equation (24) demonstrates that this core is multiplied by a tensor $\mathcal{B}_1 \in \mathbb{R}^{r_1^{k+1} n_2 \times n_3 \cdots n_{d+1}^{k+1}}$ that has reshaping $B_1 \in \mathbb{R}^{r_1^{k+1} \times n_2 \cdots n_{d+1}^{k+1}}$, and that the truncated SVD implies the approximation error ϵ_1^2:

$$\|Y_{(1)}^{k+1} - U_1^{k+1} B_1\|_F^2 \leq \epsilon_1^2. \tag{25}$$

The algorithm then proceeds to decompose the still high-dimensional B_1 into an approximation. Let $\hat{B}_1$ denote this approximation of the remaining dimensions so that we now seek to bound the error

$$\|\mathcal{Y}^{k+1} - \hat{\mathcal{Y}}^{k+1}\|_F^2 = \|Y_{(1)}^{k+1} - U_1^{k+1} \hat{B}_1\|_F^2. \tag{26}$$

Next, this error can be rewritten in terms of the known error (25) and the subsequent approximation error incurred by $\hat{B}_1$. To this end, add and subtract the exact B_1 to obtain

$$\|\mathcal{Y}^{k+1} - \hat{\mathcal{Y}}^{k+1}\|_F^2 = \|Y_{(1)}^{k+1} - U_1^{k+1} \hat{B}_1\|_F^2 = \|Y_{(1)}^{k+1} - U_1^{k+1} \left(\hat{B}_1 + B_1 - B_1\right)\|_F^2, \tag{27}$$

$$= \|Y_{(1)}^{k+1} - U_1^{k+1} B_1\|_F^2 + \|U_1^{k+1} \left(B_1 - \hat{B}_1\right)\|_F^2, \tag{28}$$

$$\leq \epsilon_1^2 + \|B_1 - \hat{B}_1\|_F^2, \tag{29}$$

where U_1^{k+1} has orthonormal columns and the inequality in Eq. (29) arises from Eq. (25).

At this stage of the algorithm, we have updated U_1^{k+1} and padded U_2^k with $r_{R_1} \times n_2 r_2^k$ zeros according to (17). The rest of the algorithm proceeds by now performing the same approximation for B_1, beginning with a projection onto the zero padded second core, $U_2^{k,pad}$. First we note that the orthonormality of the left singular vectors implies $B_1 = U_1^{(k+1)^T} Y_{(1)}^{k+1} = U_1^{(k+1)^T} Y_1$ from (24). Combined with Equation (19), these facts imply that $Y_2 \equiv B_1$ up to a reshaping. Then the next residual becomes

$$\left(I - U_2^{k,pad} U_2^{k,pad^T}\right) B_1 \equiv \left(I - U_2^{k,pad} U_2^{k,pad^T}\right) Y_2 = R_2^k. \tag{30}$$

Similar to (24), we can expand B_1 into the components parallel to the space spanned by the padded by $U_2^{k,pad}$ and orthogonal to this space so that

(31)

$$B_1 = U_2^{k,pad} U_2^{k,pad^T} B_1 + R_2^k = U_2^{k,pad} U_2^{k,pad^T} Y_2 + U_{R_2}^k V_{R_2}^k + E_2^k = U_2^{k+1} \underbrace{\begin{bmatrix} U_2^{k,pad^T} Y_2 \\ V_{R_2}^k \end{bmatrix}}_{B_2} + E_2^{k+1} = U_2^{k+1} B_2 + E_2^k.$$

Similar to (25), the truncated SVD for this dimension ensured $\|E_2^k\|_F^2 \leq \epsilon_2^2$. Then, parallel to (27)-(29), this time we expand the error in B_1 according to

$$\begin{aligned} \|B_1 - \hat{B}_1\|_F^2 =& \|B_1 - U_2^{k+1} \hat{B}_2\|_F^2, \\ =& \|B_1 - U_2^{k+1} \left(\hat{B}_2 + B_2 - B_2\right)\|_F^2, \\ =& \|B_1 - U_2^{k+1} B_2\|_F^2 + \|U_2^{k+1} \left(B_2 - \hat{B}_2\right)\|_F^2, \\ \leq& \epsilon_2^2 + \|B_2 - \hat{B}_2\|_F^2. \end{aligned} \tag{32}$$

If we rewrite Equation (31) for B_{d-1}, we see that

$$B_{d-1} = U_d^{k,pad} U_d^{k,pad^T} B_{d-1} + R_d^k \equiv U_d^{k,pad} U_d^{k,pad^T} Y_d + U_{R_d}^k V_{R_d}^k + E_d^k = U_d^{k+1} \underbrace{\begin{bmatrix} U_d^{k,pad^T} Y_d \\ V_{R_d}^k \end{bmatrix}}_{B_d} + E_d^{k+1} = U_d^{k+1} B_d + E_d^k \tag{33}$$

with $B_d \in \mathbb{R}^{r_d^{k+1} \times n_{d+1}^{k+1}}$, which is equal to Y_{d+1}. We directly append Y_{d+1} to the $d+1$-th core and conclude our update procedure for $\mathcal{Y}^{k+1}$. Therefore, for a $d+1$ dimensional tensor, repeating the update for the first d dimensions successively yields the approximation error

$$\|\mathcal{Y}^{k+1} - \hat{\mathcal{Y}}^{k+1}\|_F^2 \leq \sum_{i=1}^{d} \epsilon_i^2. \tag{34}$$

Computing the square root of this term yields (23) and thus concludes our proof. ∎

A straightforward corollary shows that TT-ICE can then ensure that the compression of the new data point can be achieved to any desired tolerance.

COROLLARY 3. *Let $\{U_i^k\}_{i=1}^{d+1}$ denote the unfolded TT-cores of the approximate accumulation $\hat{\mathcal{X}}^k$, $\mathcal{Y}^{k+1}$ be a new streamed tensor. If $\epsilon_i = \epsilon = \varepsilon_{des}\|\mathcal{Y}^{k+1}\|_F/\sqrt{d}$, then*

$$\frac{\|\mathcal{Y}^{k+1} - \hat{\mathcal{Y}}^{k+1}\|_F}{\|\mathcal{Y}^{k+1}\|_F} \leq \varepsilon_{des} \tag{35}$$

for any $\varepsilon_{des} > 0$.

Proof. The proof is a direct result of Theorem 2. If we assume $\epsilon_i = \epsilon$ for all i and simply set

$$\sqrt{\sum_{i=1}^{d} \epsilon_i^2} = \epsilon\sqrt{d} = \varepsilon_{des}\|\mathcal{Y}^{k+1}\|_F, \tag{36}$$

we can assure the desired relative error upper bound ε_{des} by truncating the SVD on Line 21 of TT-ICE at $\epsilon = \varepsilon_{des}\|\mathcal{Y}^{k+1}\|_F/\sqrt{d}$. ∎

The last step is to ensure that the update of $\mathcal{X}^k$ does not reduce the approximation error of the previously compressed elements $\mathcal{Y}^i$ for $i \leq k$.

THEOREM 4. *Let $\{U_i^k\}_{i=1}^{d+1}$ denote the unfolded TT-cores of the approximate accumulation $\hat{\mathcal{X}}^k$ such that the approximation of any existing tensor $\mathcal{Y}^\ell$ satisfies $\|\mathcal{Y}^\ell - \hat{\mathcal{Y}}^\ell\|_F \leq \epsilon\|\mathcal{Y}^\ell\|_F$ for $\ell = 1, \ldots, k$, and $\epsilon > 0$. Let $\mathcal{Y}^{k+1}$ be a new streamed tensor, and $\{\epsilon_i\}_{i=1}^{d}$ be the SVD truncation tolerances; then the updated TT-cores $\{U_i^{k+1}\}_{i=1}^{d+1}$ computed by TT-ICE represent an approximate accumulation $\hat{\mathcal{X}}^{k+1}$ that still satisfies $\|\mathcal{Y}^\ell - \hat{\mathcal{Y}}^\ell\|_F \leq \epsilon\|\mathcal{Y}^\ell\|_F$. for $\ell = 1, \ldots, k$.*

Proof. The proof shows simply that the core modifications performed by TT-ICE have no impact on the representation of earlier tensors. First recall from (22) that the approximation of the ℓ-th tensor is given by

$$\hat{\mathcal{Y}}^\ell = \mathcal{G}_1^k \,{}_3\times_1 \cdots {}_3\times_1 \mathcal{G}_d^k \,{}_3\times_1 \mathcal{G}_{d+1}^k\left(\mathcal{Y}^\ell\right), \quad \ell = 1, \ldots, k, \tag{37}$$

where we slightly abuse the notation by letting $\mathcal{G}_{d+1}^k\left(\mathcal{Y}^\ell\right)$ denote the columns of $\mathcal{G}_{d+1}^k$ corresponding to $\mathcal{Y}^\ell$.

First, we note that TT-ICE only appends columns to the last core, so that the columns corresponding to the ℓ-th tensor are unchanged when $\mathcal{G}_{d+1}^k$ is updated to $\mathcal{G}_{d+1}^{k+1}$. Thus it remains to show that the portions of $\mathcal{G}_i^k$ that multiply together and finally multiply the corresponding columns of $\mathcal{G}_{d+1}^k$ remain unchanged as the U_i^k are updated to U_i^{k+1}.

Let $G_i^k[i_1] \in \mathbb{R}^{r_{i-1}\times r_i}$ denote the slice of each TT-core for $i = 1 \ldots n_i$. Then (37) can be written for each element of $\mathcal{Y}^\ell$ according to

$$\hat{\mathcal{Y}}^\ell(i_1, \ldots, i_d, : \mathcal{Y}^\ell) = \mathcal{G}_1^k[i_1]\mathcal{G}_2^k[i_2]\cdots\mathcal{G}_d^k[i_d]\mathcal{G}_{d+1}^k[: \mathcal{Y}^\ell], \tag{38}$$

where $[: \mathcal{Y}^\ell]$ is used to denote the columns of the last core pertaining to $\hat{Y}^\ell$ so that $\mathcal{G}_{d+1}^k[: \mathcal{Y}^\ell] \equiv \mathcal{G}_{d+1}^k\left(\mathcal{Y}^\ell\right)$. Next, the zero padding and column appending of each U_i^k imply the following relationship between G_i^k and G_i^{k+1}

$$\mathcal{G}_j^{k+1}[i_j] = \begin{cases} \begin{bmatrix} \mathcal{G}_1^k[i_1] & L_1 \end{bmatrix} & , \ j = 1; \\ \begin{bmatrix} \mathcal{G}_j^k[i_j] & L_{j,1} \\ 0 & L_{j,2} \end{bmatrix} & , \ j = 2, \ldots d; \\ \begin{bmatrix} \mathcal{G}_{d+1}^k[i_{d+1}] \\ 0 \end{bmatrix} & , \ j = d+1, \end{cases} \tag{39}$$

where the zeros in the lower left block come from the padding, and the $L_{j,l}$ matrices arise from appending the new directions. Now the representation of the ℓ-th tensor in the stream with the new cores becomes

$$\hat{\mathcal{Y}}^\ell = \begin{bmatrix} \mathcal{G}_1^k[i_1] & L_1 \end{bmatrix} \begin{bmatrix} \mathcal{G}_2^k[i_2] & L_{2,1} \\ 0 & L_{2,2} \end{bmatrix} \cdots \begin{bmatrix} \mathcal{G}_d^k[i_d] & L_{d,1} \\ 0 & L_{d,2} \end{bmatrix} \begin{bmatrix} \mathcal{G}_d^k[i_{d+1}] \\ 0 \end{bmatrix} = \mathcal{G}_1^k[i_1]\mathcal{G}_2^k[i_2]\cdots\mathcal{G}_d^k[i_{d+1}] = \hat{\mathcal{Y}}^\ell, \tag{40}$$

where the locations of the zeros enable the new terms to appropriately cancel all the new terms. Thus the representation of previously compressed tensors does not change, and therefore their reconstruction error remains the same after the core update. ∎

3.3. Heuristics to improve performance. This section discusses three heuristic modifications to TT-ICE to reduce the computational load and memory requirements of our approach.

These approaches aim to reduce the number of core updates between $\mathcal{X}^k$ and $\mathcal{X}^{k+1}$. First, we provide a metric based on TT-ranks to decide whether an attempt at updating a TT-core should even be made. Second, when a new batch of n_{d+1}^{k+1} tensors is presented at increment $k+1$, we describe an approach to subselect from this batch to only perform an update with a smaller number of tensors. This update is based on each of the approximation errors of individual tensors in the streamed batch. Third, we propose a mechanism to determine when it is feasible to skip the core updating process.

Core occupancy for reducing the number of cores to be updated. The TT-ICE Algorithm 3.1 proposes a scheme for updating all the TT-cores $\{\mathcal{G}_i^k\}_{i=1}^{d+1}$ at each increment. However, the new information provided by increment may not be uniform throughout all these TT-cores. If we can detect the core where the most information loss occurs, focusing update efforts on that core could increase the efficiency of Algorithm 3.1.

We propose a heuristic named *core occupancy* to measure how much information is already represented by a core to determine if it should be updated. Recall that the i-th TT-core is constructed by reshaping the left singular vectors $U_i^k \in \mathbb{R}^{r_{i-1}^k n_i \times r_i^k}$. As a result, there are r_i^k orthogonal vectors from the possible $r_{i-1}^k n_i$ basis vectors. For a given TT-core, core occupancy represents the ratio of the truncation rank r_i^k over the maximum rank possible (the number of rows $r_{i-1}^k n_i$). It can also be seen as a ratio of the column to row-ranks:

$$\texttt{occupancy}\left(\mathcal{G}_i^k\right) = \frac{r_i^k}{r_{i-1}^k n_i}. \tag{41}$$

If a core has a high occupancy ratio, then we can skip its update.

Subselecting the observations of $\mathcal{Y}^{k+1}$ used to update $\{\mathcal{G}_i^k\}_{i=1}^{d+1}$. If the new data comes in a batch (i.e. $n_{d+1}^{k+1} > 1$), another way of reducing the computational cost of updating the TT-cores is to use a reduced number of observations from the new tensor $\mathcal{Y}^{k+1}$ to update the first d cores of the accumulation tensor. To subselect which elements of the batch to use for the update, we first compute the projection of $\mathcal{Y}^{k+1}$ onto TT-cores $\{\mathcal{G}_i^k\}_{i=1}^{d+1}$ and use the relative projection error of the individual observations as a heuristic to select a subset of $\mathcal{Y}^{k+1}$. With a slight abuse of notation, we denote the i-th observation in $\mathcal{Y}^{k+1}$ as $\mathcal{Y}^{k+1}(i)$, where $i = 1, \ldots, n_{d+1}^{k+1}$. Let $\tilde{\mathcal{Y}}^{k+1}$ be the approximation of $\mathcal{Y}^{k+1}$ using the TT-cores $\{\mathcal{G}_i^k\}_{i=1}^{d+1}$. Similarly, let $\tilde{\mathcal{Y}}^{k+1}(i)$ denote the i-th observation in $\tilde{\mathcal{Y}}^{k+1}$. Then, the vector $\varepsilon_{\mathcal{Y}^{k+1}} \in \mathbb{R}^{n_{d+1}^{k+1}}$ is vector of relative errors of individual observations, where

$$\varepsilon_{\mathcal{Y}^{k+1}}(i) = \frac{\|\mathcal{Y}^{k+1}(i) - \tilde{\mathcal{Y}}^{k+1}(i)\|_F}{\|\mathcal{Y}^{k+1}(i)\|_F}, \tag{42}$$

for $i = 1, \ldots, n_{d+1}^{k+1}$.

Recall that we can use Algorithm 3.1 with truncated SVD and update the TT-cores of $\hat{\mathcal{X}}^k$ to $\hat{\mathcal{X}}^{k+1}$ so that $\mathcal{Y}^{k+1}$ is represented within a truncation error threshold. Let ε_{des} denote the determined relative error threshold as in Corollary 3. Before attempting an update, we calculate the approximation error of each new data point $\varepsilon_{\mathcal{Y}^{k+1}}$ with the existing TT-cores using (42) and then compute its average, $\texttt{mean}\left(\varepsilon_{\mathcal{Y}^{k+1}}\right)$. If the average approximation error is greater than the desired error tolerance ($\texttt{mean}\left(\varepsilon_{\mathcal{Y}^{k+1}}\right) > \varepsilon_{des}$), then we will proceed to update at least some of the cores. In particular, we only update the TT-cores of the accumulation using the data points for which the approximation error exceeds ε_{des}. Let $\mathcal{D}^{k+1}$ be that constructed new $(d+1)$-way tensor with a reduced number of observations such that

$$\mathcal{D}^{k+1} = \left\{\mathcal{Y}^{k+1}(i) :\ \varepsilon_{\mathcal{Y}^{k+1}}(i) > \varepsilon_{des}\right\}_{i=1}^{n_{d+1}^{k+1}}. \tag{43}$$

Since the algorithm operates on a subset of observations, the truncation parameter needs to be adjusted accordingly. Using the same ε_{des} as $\mathcal{Y}^{k+1}$ for $\mathcal{D}^{k+1}$ enforces a tighter relative error upper bound on $\mathcal{Y}^{k+1}$. This tighter error bound results in increased TT-ranks that impair the compression performance. To avoid this, we need to relax the truncation parameter ε_{des} using the approximation error of the discarded observations. Let $\mathcal{D}_C^{k+1}$ be the tensor constructed with those discarded observations such that

$\mathcal{D}_C^{k+1} = \left\{\mathcal{Y}^{k+1}(i) : \ \varepsilon_{\mathcal{Y}^{k+1}}(i) \leq \varepsilon_{des}\right\}_{i=1}^{n_{d+1}^{k+1}}$. Then, we compute the relaxed relative error tolerance ε_{upd} for the tensor of selected observations $\mathcal{D}^{k+1}$ as

$$\varepsilon_{upd} = \sqrt{\frac{\left(\varepsilon_{des}\|\mathcal{Y}^{k+1}\|_F\right)^2 - \|\mathcal{D}_C^{k+1} - \tilde{\mathcal{D}}_C^{k+1}\|_F^2}{\|\mathcal{D}^{k+1}\|_F^2}}, \tag{44}$$

where $\tilde{\mathcal{D}}_C^{k+1}$ is the approximation of $\mathcal{D}_C^{k+1}$ using $\{U_i^k\}_{i=1}^{d+1}$. The first term in the numerator is the maximum amount of error that TT-ICE can allow for $\mathcal{Y}^{k+1}$ in the Frobenius norm, the second term is the approximation error of the discarded observations in the Frobenius norm, and the denominator is the Frobenius norm of the tensor of selected observations. Since the discarded observations have a relative error $\leq \varepsilon_{des}$, subtracting the approximation error gives how much error TT-ICE can allow if it only uses $\mathcal{D}^{k+1}$ to update the TT-cores. The denominator functions as a normalizing factor, converting the error in the numerator to a relative error for incremental updates. Through ε_{upd}, the low approximation error of the discarded observations will be balanced by tolerating slightly more error for $\mathcal{D}^{k+1}$ and the overall approximation $\hat{\mathcal{Y}}^{k+1}$ will have a relative error closer to ε_{des}. Note that for the edge case where all $\varepsilon_{\mathcal{Y}^{k+1}}(i) > \varepsilon_{des}$, we have $\mathcal{D}^{k+1} = \mathcal{Y}^{k+1}$ and $\mathcal{D}_C^{k+1} = \emptyset$. This results in $\varepsilon_{upd} = \varepsilon_{des}$, therefore providing a consistent method to update ε_{des}.

If the observations in the same batch have similar norms, we can approximate Eq. (44) by replacing the norm operators with the count of observations in each tensor $|\cdot|$ and get

$$\varepsilon_{upd} \approx \frac{\varepsilon_{des} n_{d+1}^{k+1} - \varepsilon_{\mathcal{D}_C^{k+1}}|\mathcal{D}_C^{k+1}|}{|\mathcal{D}^{k+1}|}, \tag{45}$$

where $\varepsilon_{\mathcal{D}_C^{k+1}}$ is the mean relative error of the observations in $\mathcal{D}_C^{k+1}$ computed analogously to (42).

Through these modifications, we prevent a superfluous increase in TT-ranks. After determining ε_{upd}, we simply compute the truncation parameter for the SVD using ε_{upd} as $\epsilon = \frac{\varepsilon_{upd}}{\sqrt{d}}\|\mathcal{D}^{k+1}\|_F$.

Skip updating the first d TT-cores. Our final heuristic is to skip updating the cores if the average error of the tensors in a batch is less than the desired threshold: $\texttt{mean}\left(\varepsilon_{\mathcal{Y}^{k+1}}\right) \leq \varepsilon_{des}$. We justify this heuristic via the following argument.

Let ε_{k+1} represent the relative error of approximating the new tensor via the existing cores

$$\varepsilon_{k+1} = \frac{\|\mathcal{Y}^{k+1} - \tilde{\mathcal{Y}}^{k+1}\|_F}{\|\mathcal{Y}^{k+1}\|_F} = \sqrt{\frac{\sum_{i=1}^{n_{d+1}^{k+1}} \|\mathcal{Y}^{k+1}(i) - \tilde{\mathcal{Y}}^{k+1}(i)\|_F^2}{\sum_{i=1}^{n_{d+1}^{k+1}} \|\mathcal{Y}^{k+1}(i)\|_F^2}}, \tag{46}$$

where $\tilde{\mathcal{Y}}^{k+1}$ is the approximation of $\mathcal{Y}^{k+1}$ with the existing TT-cores. If $\varepsilon_{k+1} \leq \varepsilon_{des}$, then the first d TT-cores of $\mathcal{X}_k$ can represent $\mathcal{Y}^{k+1}$ sufficiently accurately and do not need updates to meet the desired accuracy. As a result, the error-truncated SVD in Line 13 of Algorithm 3.1 returns empty matrices and TT-ICE will return the first d TT-cores without an update. In order to save invaluable computation time, we can calculate ε_{k+1} using (46) and complete the core update by appending the TT-representation of $\mathcal{Y}^{k+1}$ after projecting onto the TT-cores of $\mathcal{X}^k$.

Let $\hat{Y}^{k+1}$ be the projection of $\mathcal{Y}^{k+1}$ onto the first d TT-cores $\{\mathcal{G}_i^k\}_{i=1}^d$. Then, updating $\mathcal{G}_{d+1}^k$ is simply done by appending $\hat{Y}^{k+1}$ to $\mathcal{G}_{d+1}^k$ as

$$\mathcal{G}_{d+1}^{k+1} \leftarrow \begin{bmatrix}\mathcal{G}_{d+1}^k & \hat{Y}^{k+1}\end{bmatrix}. \tag{47}$$

However, an explicit computation of (46) can become computationally expensive if the batch consists of a high number of observations. In that case, we investigate the use of the surrogate of $\varepsilon_{\mathcal{Y}^{k+1}}$ from (42) as an approximation to ε_{k+1}. In future works, this strategy might be adapted to sampling tensors in the batch to have a stochastic approximation of the average. Meanwhile, the numerical experiments in section 4 use $\texttt{mean}(\varepsilon_{\mathcal{Y}^{k+1}})$ to determine if TT-ICE* should skip updating the first d TT-cores. As an example, Figure 3 provides empirical proof that using the mean as an approximation does not result in a violation of the relative error upper bound. However, note that there might be other, more conservative heuristic measures available to approximate ε_{k+1}.

The pseudocode of the modified algorithm is provided in Algorithm 3.2. The `project` function on Line 36 projects $\mathcal{Y}^{k+1}$ sequentially onto $\{U_i^{k+1}\}_{i=1}^d$ and reshapes into proper dimensions in a way similar to Line 33.

Algorithm 3.2 `TT-ICE*`: Incremental update of a tensor train decomposition with heuristic performance upgrades

Input
- $\left\{U_i^k\right\}_{i=1}^{d+1}$ reshaped cores of the TT-decomposition of the accumulation $\mathcal{X}^k$
- $\mathcal{Y} \in \mathbb{R}^{n_1 \times \cdots \times n_d \times n_{d+1}^{k+1}}$ new tensor
- τ occupancy threshold (suggested=0.8)
- ε_{des} relative error upper bound

Output
- $\left\{U_i^{k+1}\right\}_{i=1}^{d+1}$ updated cores for the accumulation $\mathcal{X}^{k+1}$

Check representation accuracy with existing TT-cores and *Skip* updating if sufficient

$\varepsilon_{\mathcal{Y}}(i) \leftarrow \frac{\|\mathcal{Y}(i)-\tilde{\mathcal{Y}}(i)\|_F}{\|\mathcal{Y}(i)\|_F}\Big|_{i=1}^{n_{d+1}^{k+1}}$ ▷ $\tilde{\mathcal{Y}}$ is the approximation of $\mathcal{Y}$ using U^k, and $\varepsilon_{\mathcal{Y}} \in \mathbb{R}^{n_{d+1}^{k+1}}$ as defined in (42)

if $\texttt{mean}(\varepsilon_{\mathcal{Y}}) \leq \varepsilon_{des}$ **then**
- $\{U_i^{k+1}\}_{i=1}^d \leftarrow \{U_i^k\}_{i=1}^d$

else
- ***Subselect*** **observations**
- $\mathcal{D} \leftarrow \{\mathcal{Y}(j) : \varepsilon_{\mathcal{Y}}(j) > \varepsilon_{des}\}_{j=1}^{n_{d+1}^{k+1}}$ ▷ $\mathcal{D} \subset \mathcal{Y}$ will be used at update
- $\varepsilon_{upd} \leftarrow \sqrt{\frac{(\varepsilon_{des}\|\mathcal{Y}\|_F)^2 - \left(\|\mathcal{D}_C - \tilde{\mathcal{D}}_C\|_F\right)^2}{\|\mathcal{D}\|_F^2}}$ ▷ $\mathcal{D} \cup \mathcal{D}_C = \mathcal{Y}$, $\tilde{\mathcal{D}}_C$ is approximation of $\mathcal{D}_C$ using U^k
- $\epsilon = \frac{\varepsilon_{upd}}{\sqrt{d}}\|\mathcal{D}\|_F$ ▷ ϵ is the truncation parameter for SVD
- **Perform incremental updates with the selected observations**
- $D_1 = \texttt{reshape}(\mathcal{D}, [n_1, n_2 \dots n_d n_{d+1}^{\mathcal{D}}])$ ▷ $n_{d+1}^{\mathcal{D}}$ is the number of selected observations
- $U_1^{k,pad} \leftarrow U_1^k$ ▷ First core has no padding
- **for** $i = 1$ to d **do**
  - **Check core *Occupancy***
  - **if** $\texttt{occupancy}(U_i^{k,pad}) \geq \tau$ **then**
    - $U_i^{k+1} \leftarrow U_i^{k,pad}$
    - $r_i^{k+1} \leftarrow r_i^k$
    - $U_{i+1}^{k,pad} \leftarrow U_{i+1}^k$
  - **else**
    - **Perform incremental update**
    - $R_i^k = \left(I - U_i^{k,pad} U_i^{k,pad^T}\right) D_i$
    - $U_{R_i}^k \leftarrow \texttt{SVD}\left(R_i^k, \epsilon\right)$
    - $U_i^{k+1} \leftarrow \begin{bmatrix} U_i^{k,pad} & U_{R_i}^k \end{bmatrix}$
    - $U_{i+1}^{k,pad} \leftarrow \texttt{reshape}\left(\begin{bmatrix} \texttt{reshape}\left(U_{i+1}^k, \left[r_i^k, n_{i+1} r_{i+1}^k\right]\right) \\ \mathbf{0}_{r_{R_i} \times n_{i+1} r_{i+1}^k} \end{bmatrix}, [r_i^{k+1} n_{i+1}, r_{i+1}^k]\right)$
  - **end if**
  - $D_{i+1} \leftarrow \texttt{reshape}(U_i^{k+1^T} D_i, [r_i^{k+1} n_{i+1}, n_{i+2} \dots n_{d+1}^{\mathcal{D}}])$
- **end for**

end if

$\hat{Y}^{k+1} \leftarrow \texttt{project}(\mathcal{Y}^{k+1}, \{U_i^{k+1}\}_{i=1}^d)$ ▷ $\hat{Y}$ is obtained by sequentially projecting $\mathcal{Y}^{k+1}$ onto $\{U_i^{k+1}\}_{i=1}^d$

$U_{d+1}^{k+1} \leftarrow \begin{bmatrix} U_{d+1}^k & \hat{Y}^{k+1} \end{bmatrix}$ ▷ $\hat{Y}^{k+1} \in \mathbb{R}^{r_d \times n_{d+1}^{k+1}}$ are the columns

4. Experiments. In this section, we compare the performance of TT-ICE and TT-ICE* with existing approaches in compressing large-scale data including videos from gameplay sequences and grid data from physics-based simulations. In all the experiments, the TT-SVD algorithm [22, Alg 1] is applied to obtain an initial set of TT-cores and subsequently, the cores are updated using each of the incremental algorithms. In all of the results provided, ITTDk corresponds to updating the TT-cores of the accumulation using ITTD and performing TT-rounding [22, Alg 2] after every k-th update step. For comparisons between TT-FOA and our approach, we refer to Appendix B. We do not include it here because TT-FOA does not support rank adaptation and does not offer an upper bound on approximation error. As a result, it either performed poorly or was unable to handle the datasets considered.

4.1. Evaluation criteria. This section contains performance metrics we use to compare performance of algorithms.

The performance of the incremental algorithms is evaluated using three main criteria. First, the *compression ratio* (CR) of the accumulation $\mathcal{X}^k$ with core sizes $n_1 \times n_2 \times \cdots \times n_{d+1}$ is defined as

$$CR = \frac{\text{number of elements of full tensor}}{\text{number of parameters in compressed representation}} = \frac{\prod_{i=1}^{d+1} n_i}{\sum_{i=1}^{d+1} r_{i-1} n_i r_i}, \tag{48}$$

where r_i is the i-th TT-rank. Second, the *relative reconstruction error* (RRE) of the same tensor is given by

$$RRE = \frac{\|\mathcal{X}^k - \hat{\mathcal{X}}^k\|_F}{\|\mathcal{X}^k\|_F}. \tag{49}$$

Similarly, we can measure the error in the representation of *unseen* data: since the last dimension of $\mathcal{X}^k$ provides a latent representation for individual observations stored in the stream, as shown in Eq. (22), we can use the first d cores to estimate a latent representation for unseen data by projecting onto the first d dimensions. We shall call the error in the estimation the *relative prediction error* (RPE), which can again be computed using the Frobenius error similar to RRE. An RPE lower than the target tolerance ε_{des} indicates that the existing basis of the accumulation is expressive enough to represent this unseen data. Our final evaluation metric is the *execution time*, which measures the CPU time spent executing the steps of the corresponding incremental algorithms and does not include the time for data to load into memory.

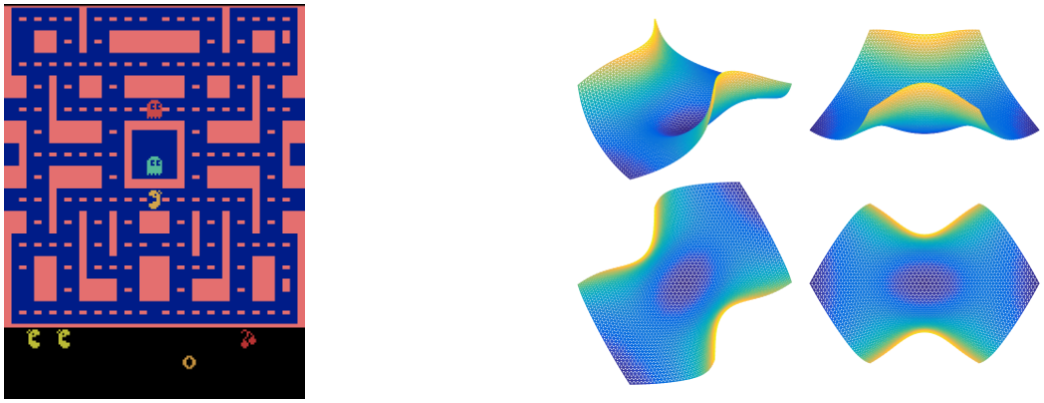
4.2. Datasets and experiments. This section includes tests on two different types of datasets (snapshots in Figure 1): (i) Raw pixel data from an ATARI game-playing reinforcement learning (RL) agent [5] and (ii) Grid data from a numerical simulation of self-oscillating gels [2].

4.2.1. ATARI gameplay sequences. This section includes tests of TT-ICE's performance to compress raw pixel data.

The first dataset involves a sequential set of video game screenshots. It is often useful to reduce the dimension of this type of video data to enable learning in the latent space. This dataset consists of a collection of gameplay screen captures from various ATARI games. The dataset is collected using a video game-playing RL agent trained by a Deep Q-Network model [20] from the RL-Baselines Zoo package [23] and further trained using the Stable Baselines platform [13]. The ATARI games we used are Ms.Pac-Man, Enduro, Seaquest, Q*bert, Breakout, Pong, and Beamrider. We present the results of experiments for Ms.Pac-Man game captures in this section and the rest of the games in the Appendix.

Each game has multiple individual gameplay sessions, which we refer to as *runs*. Each run may have different durations and therefore a different number of frames (see Figure 1a) with dimensions $210 \times 160 \times 3$ ($Width \times Height \times RGB$). Each individual run is treated as an incremental unit and reshaped into a 5-way tensor of dimension $30 \times 28 \times 40 \times 3 \times n_5^k$, where n_5^k is the number of frames in the k-th run. Depending on the duration of each gameplay sequence, each run consists of a varying number of frames. Therefore, the 5-way tensors are stacked along the fifth dimension.

We compared TT-ICE, TT-ICE*, and ITTD5 on all of the different game datasets, and the results are summarized in Table A.1. A more detailed study of the efficiency of each algorithm was done for Ms.Pac-Man. For the comprehensive experiments with Ms.Pac-Man gameplay sequences, we used a training set of 60 runs for incremental updates and a validation set of 160 runs to measure the prediction error on unseen



(a) (b)

Fig. 1: *Example visualizations from the datasets used in the compression experiments. (a) A frame from a Ms.Pac-Man gameplay session. (b) Snapshots from self-oscillating gel simulation using the simulation developed in [2].*

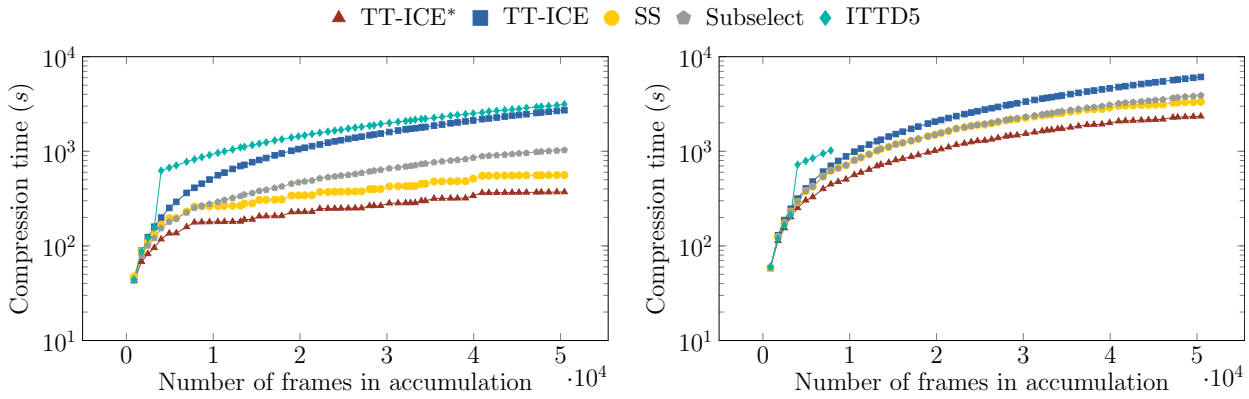
runs. This prediction tests the performance of the TT-format to find a suitable latent space for describing Ms.Pac-Man frames. We present the results of detailed experiments in Figures 2 to 4. In those figures, *Subselect* means updating TT-cores using only the observations from $\mathcal{Y}^{k+1}$ with relative error higher than ε_{des} and *SS* means skipping updates for the first d TT-cores when $\texttt{mean}(\varepsilon_{\mathcal{Y}^{k+1}}) \leq \varepsilon_{des}$ along with *Subselect* heuristic. Please note that the *Occupancy* heuristic is only used when the complete TT-ICE* algorithm is used.

For all other games in Table A.1, we used a training set of 60 runs and a validation set of 100 runs. Enduro and Pong were the only exceptions to that setup since they had a much higher number of frames in each run. This resulted in much higher memory requirements than all other games even just for storing the runs. Again, due to the high number of frames per run, the validation tests required a significant amount of time. Therefore we used a training set of 40 runs and a validation set of 40 runs. During the repetitions for different algorithms, the streaming order was the same for all methods subject to investigation. All experiments were repeated for two ε_{des} settings, $\varepsilon_{des} = 0.1$ and $\varepsilon_{des} = 0.01$. Finally, an occupancy threshold of 0.8 is set for this class of experiments.

Compression results. This section includes the results of experiments on Ms.Pac-Man frames, and then comment on the compression experiments with other ATARI games.

Figures 2a and 2b show the influence of each variation of TT-ICE on the overall compression time and compare these results with ITTD5. In Figure 2a, ITTD5 performs worse than all of the TT-ICE variations at $\varepsilon_{des} = 0.1$. Specifically, Figure 2a depicts that implementing heuristic improvements provides at least 62% reduction in compression time. This improvement in compression time reaches its peak when all of the heuristic upgrades are implemented (i.e. when the complete TT-ICE* algorithm is used). Note that when the truncation tolerance is tightened to $\varepsilon_{des} = 0.01$, the problem with ITTD intensifies. This time, ITTD5 fails to compress the full duration of the stream due to insufficient memory and fails at the 10^{th} increment while attempting TT-rounding. On the other hand, TT-ICE and variations of TT-ICE* display performances in parallel to the experiments with $\varepsilon_{des} = 0.1$.

Figures 3a and 3b investigate a difference in the representation error of the TT-cores trained with different algorithms. For each case, we present the mean RRE over the compressed portion of the training set and the mean RPE over the entire validation set after each increment. Figure 3a shows that all methods can successfully represent the streamed data within the desired relative error upper bound ε_{des}. Furthermore, Figure 3a depicts that the mean RPE of TT-ICE and TT-ICE* converge asymptotically to the mean RRE. Since the mean RPE represents the representation quality of the validation set, these results indicate that a suitable basis is found prior to observing the validation dataset itself. Once the basis to represent observations within the desired accuracy is complete, TT-ICE will not be able to find any unique orthogonal directions



(a) $\varepsilon_{des} = 0.1$ (b) $\varepsilon_{des} = 0.01$

Fig. 2: *Comparison of execution time vs total number of observations in the tensor train for different versions of TT-ICE, TT-ICE* and ITTD[15] with two different ε settings. TT-ICE*: full TT-ICE* algorithm with all heuristics, TT-ICE: TT-ICE, Subselect: subselecting frames from runs, SS: subselecting frames from runs and skip updating the first d TT-cores, ITTD 5: ITTD with rounding at every fifth increment step. ITTD has worst compression time while TT-ICE* with all heuristics performs best. ITTD Fails to compress the entire stream for* $\varepsilon_{des} = 0.01$.

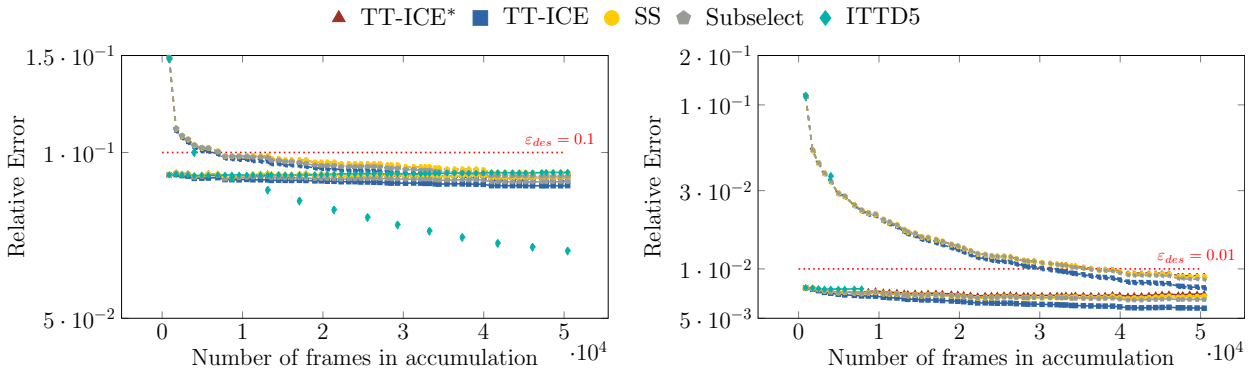
to expand the bases of TT-cores for a stream.

Figures 3a and 3b present the mean RPE for ITTD5 only at the steps where rounding is performed and do not connect the data points with dashed line. This is caused by the fact that without reorthogonalization, TT-cores obtained through ITTD are not suitable for assimilating data (projection and prediction). This is an artifact caused by implementing addition in TT-format and the uncontrolled rank inflation and lack of core orthogonality. However, an interesting point from Figure 3a is that the mean RPE falls well below the mean RRE for ITTD5 after reorthogonalization. The results for TT-ICE and TT-ICE* are similar for $\varepsilon_{des} = 0.01$ in Figure 3b. Unfortunately, ITTD5 cannot display the same pattern in this case, since it terminates prematurely due to insufficient memory.

Finally, Figures 4a and 4b indicate differences in compression ratio between algorithms. Figure 4a shows that the TT-ICE variations have comparable performance to each other. Furthermore, Figure 4a shows selecting any TT-ICE variation results in almost two orders of magnitude higher compression over ITTD5. This significantly lower compression ratio of ITTD also explains its reduction in mean RPE after TT-rounding. The superfluous increase in TT-ranks allows the TT-cores trained with ITTD to cover a much larger portion of the multidimensional basis. Right after reorthogonalization, this larger basis provides an increase in the generalization capability of the TT-cores but also results in a much lower compression ratio. The rounding step also provides a positive jump in the compression ratio, but when the TT-cores are incremented further with ITTD, this improvement decays quickly. When the relative error tolerance changes to $\varepsilon_{des} = 0.01$, the low compression ratio problem for ITTD5 is exacerbated where the compression ratio falls below the critical value of 1 for ITTD5. Having a compression ratio lower than 1 means that TT-cores have more entries than the original accumulation. Despite having a much lower compression ratio than the case with $\varepsilon_{des} = 0.1$, all of the TT-ICE variations again perform comparably with a compression ratio around $3.5\times$.

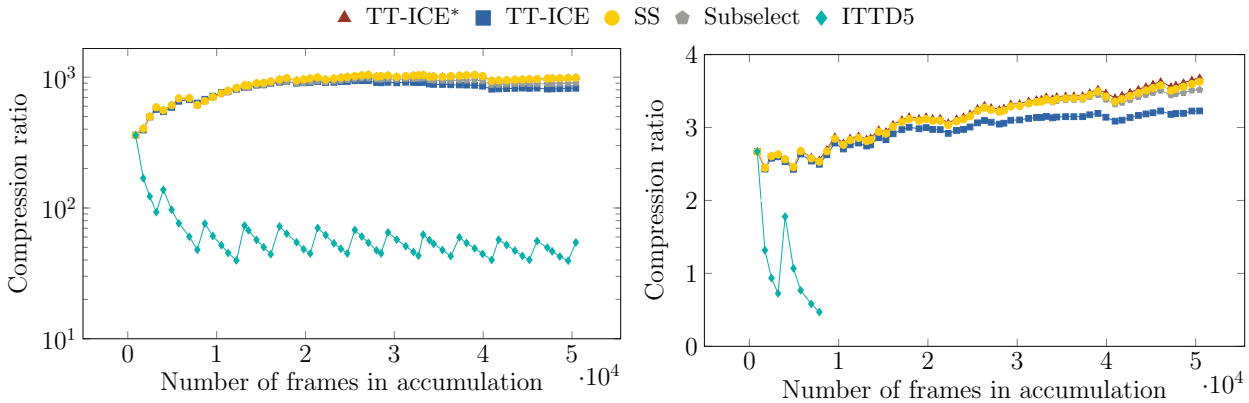
To summarize, the investigations yielded a decrease in compression time, an increase in compression ratio, and improved execution stability when TT-ICE is preferred over ITTD. Furthermore, investigations show an additional decrease in compression time and an increase in compression ratio when TT-ICE* is preferred over TT-ICE.

4.2.2. Self-oscillating gel simulations. This section includes tests of TT-ICE's performance to compress simulation outputs of high-dimensional PDEs.



(a) $\varepsilon_{des} = 0.1$ (Dotted line) (b) $\varepsilon_{des} = 0.01$ (Dotted line)

Fig. 3: *Comparison of mean RRE and mean RPE vs total number of observations in the tensor train for different versions of TT-ICE, TT-ICE* and ITTD[15] with two different ε settings. We present the mean RRE over compressed runs in the accumulation and mean RPE over the validation set after each increment step. Data points connected with a solid line represent the mean RRE and data points connected with a dashed line represent the mean RPE. Please refer to Figure 2 for a description of the legend. For $\varepsilon_{des} = 0.1$ TT-cores returned from ITTD have much lower RPE than desired. RPE of TT-cores trained with TT-ICE and TT-ICE* converge to RRE as accumulation size increases. ITTD Fails to compress the entire stream for $\varepsilon_{des} = 0.01$.*



(a) $\varepsilon_{des} = 0.1$ (b) $\varepsilon_{des} = 0.01$

Fig. 4: *Comparison of compression ratio vs total number of observations in the tensor train for different versions of TT-ICE, TT-ICE*, and ITTD[15] with two different ε settings. Please refer to Figure 2 for a description of the legend. ITTD has the worst compression ratio while TT-ICE* with all heuristics performs the best. ITTD Fails to compress the entire stream for $\varepsilon_{des} = 0.01$.*

When high-dimensional systems are of interest, the size of the simulation outputs can become prohibitive. In the extreme, even the storage of the data may not be feasible. Streaming compression algorithms can become essential when the outputs of a dynamical simulation become sequentially available. Moreover, tensor decompositions can extract more information than incremental matrix decomposition methods to be used to learn low-dimensional representations, for example, for inverse design [1].

The second dataset arises from solutions of a parametric PDE that simulates the motion of a hexagonal sheet of self-oscillating gels. The motion of the gel is governed by the following time-dependent parametric

PDE

$$\mu \frac{\partial r}{\partial t} = f_s\left(r, \mathbf{K_s}, \eta\right) + f_B\left(r\right), \qquad \eta\left(x, y, t, \mathbf{A}, \mathbf{k}\right) = 1 + \mathbf{A}\sin\left(2\pi\left(\mathbf{k}\sqrt{x^2+y^2} - t\right)\right), \tag{50}$$

where the bold terms indicate the input parameters to the forward model that define the characteristics of the excitation as well as the mechanical properties of the gel. More specifically, $\mathbf{K_s}$ denotes the stretching stiffness of the sheet, $\mathbf{k}$ determines the wavenumber of the sinusoidal excitation, and $\mathbf{A}$ determines the amplitude of the wave traveling on the sheet. Other terms governing the overdamped sheet dynamics are: internal damping coefficient μ, material coordinates $r = (x, y, z)$, stretching force f_s, bending force f_b, rest strain η, and time t.

We uniformly sample from this 3-dimensional space of input parameters to obtain 6400 unique parameter combinations and then simulate each of those parameter combinations using the approach in [2]. The simulations are chaotic, but we use 10 sequential timesteps from each simulation as our data. Specifically, we seek to compress the x, y, and z coordinates of 3367 mesh nodes on a hexagonal gel sheet for each time snapshot as shown in Figure 1b.

To summarize, the data consists of $3367 \times 3 \times 10$ tensors for *each* parameter combination that contain the coordinate information of the mesh. We refer to those output tensors as *simulations* for brevity and treat them as individual incremental units. For compression experiments, each simulation is reshaped into tensors of size $7 \times 13 \times 37 \times 3 \times 10 \times 1$ and stacked along the sixth dimension to obtain the accumulation. We conducted all the experiments on M3 machine that has a Xeon Silver 4110 processor and 16GB memory. Figures 5 to 7 show the results of those experiments. The occupancy threshold of TT-ICE* is set to 1 for this dataset. This prevents update attempts when the basis for one dimension is fully explored (i.e. the TT-core has full rank). Finally, we compute the mean of RRE over compressed simulations in the accumulation.

Despite the suitability of TT-FOA to this tensor stream, we can't provide any results for TT-FOA. Even when we assume that TT-FOA is initialized with the final ranks of TT-ICE* with $\varepsilon_{des} = 0.1$, the memory required by the auxiliary matrices, matrix inversions, and Kronecker products in the algorithm exceeds the memory of M3 machine and fails at the first step[3].

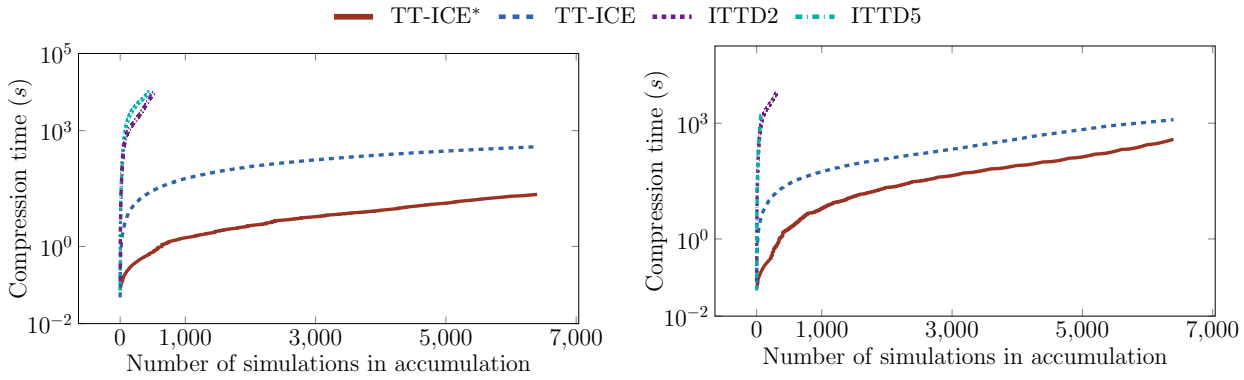
Compression results. This section includes the results of experiments on self-oscillating gel simulation snapshots.

Unlike the previous set of experiments involving ATARI gameplay frames, ITTD methods fail to compress the entire stream in both ε_{des} settings due to insufficient memory while attempting TT-rounding. In all scenarios, TT-ICE* outperforms TT-ICE, but both algorithms are able to compress the entire stream successfully.

Figures 5a and 5b depict the differences in compression speed between algorithms. Figure 5a shows that it takes more time for both ITTD2 and ITTD5 to compress 1/20 of the entire stream than for TT-ICE and TT-ICE* to compress the entire stream. Figure 5a also clearly illustrates the benefits of implementing heuristic upgrades, since TT-ICE* compresses the entire stream an order of magnitude faster than TT-ICE. This difference in compression time becomes less in Figure 5b with $\varepsilon_{des} = 0.01$ but TT-ICE* still provides a significant reduction in time in comparison to TT-ICE. At $\varepsilon_{des} = 0.01$, ITTD methods fail even earlier than before and compress at most 1/50 of the entire stream.

Figures 6a and 6b present the differences in the compression ratio between algorithms. Figure 7a shows that both TT-ICE and TT-ICE* achieve excessive compression close to $10^4\times$ in the early stages of the stream and then exhibit a decay in compression ratio. This behavior is related to the streaming order of the simulations and is expected. Towards the beginning, the stream consists of simulations with similar parameter combinations. This leads to simulations exhibiting similar motion patterns and allows the accumulation to be represented with a small basis. Then, as the stream progresses, the accumulation consists of simulations from a greater variety of parameter combinations and calls for an expansion in the bases. The same pattern repeats itself in Figure 6b for $\varepsilon_{des} = 0.01$, but this time the decay is greater than it was for $\varepsilon_{des} = 0.1$. Since both ITTD methods fail to compress the entire stream, we can not make meaningful comments on their compression performance. However, Figures 6a and 6b indicate that the

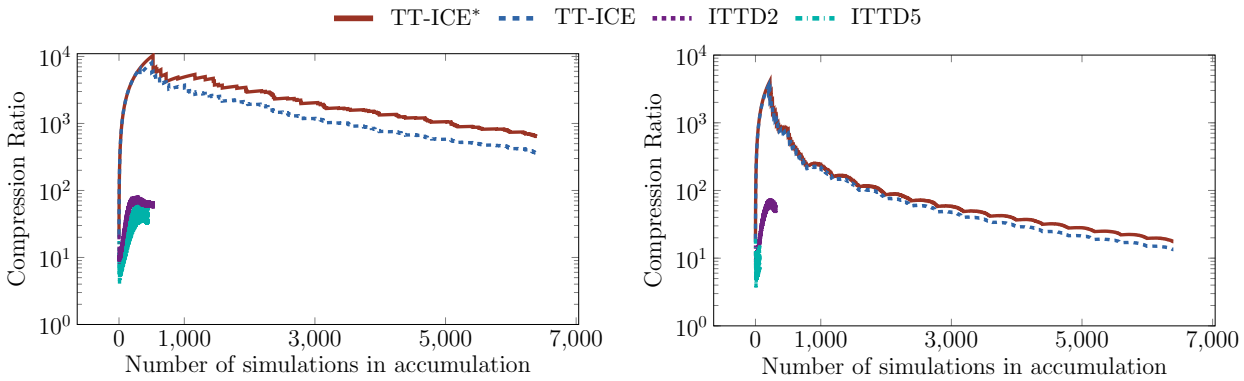
[3]When we repeat the same experiment with M1 machine, which has the same processor with M3 and has higher memory, computing one step of the stream takes more than $1500s$.



(a) $\varepsilon_{des} = 0.1$ (b) $\varepsilon_{des} = 0.01$

Fig. 5: *Comparison of execution time vs total number of simulations in the tensor train for Algorithm 3.1, Algorithm 3.2 and ITTD[15] with two different ε settings. TT-ICE* outperforms TT-ICE by one order of magnitude. ITTD Fails to compress the entire stream for both cases. TT-ICE*: full TT-ICE* with all heuristics defined in Section 3.3, TT-ICE: TT-ICE algorithm (Algorithm 3.1), ITTD5: ITTD with rounding at every fifth increment step, ITTD2: ITTD with rounding at every second increment step.*

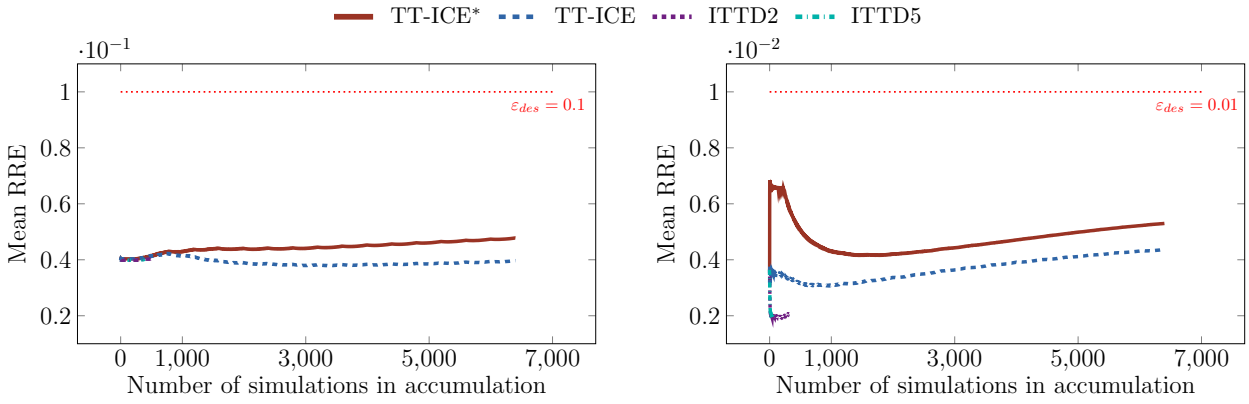
peak compression achieved is consistently two orders of magnitude less than that achieved by the TT-ICE methods.



(a) $\varepsilon_{des} = 0.1$ (b) $\varepsilon_{des} = 0.01$

Fig. 6: *Comparison of compression ratio vs total number of simulations in the tensor train for Algorithm 3.1, Algorithm 3.2 and ITTD[15] with two different ε settings. Please refer to Figure 5 for a description of the legend. TT-ICE* outperforms TT-ICE* but both methods show comparable compression performances. ITTD Fails to compress the entire stream for both cases.*

Finally, Figures 7a and 7b indicate differences in reconstruction error between algorithms. Figure 7a shows that TT-ICE* has a mean RRE slightly closer to ε_{des} than TT-ICE. On the other hand, Figure 7b shows that TT-ICE* has nearly twice the error of TT-ICE, but this difference in mean RRE diminishes to almost a constant offset as the stream progresses. In both Figure 7a and Figure 7b, ITTD methods start with lower mean RRE values. This can be explained by the significantly lower compression performance of these methods, where the higher coverage in the bases results in both lower compression and lower mean RRE. However, we cannot draw meaningful conclusions for ITTD since both ITTD2 and ITTD5 fail prematurely.



(a) $\varepsilon = 0.1$ (b) $\varepsilon_{des} = 0.01$

Fig. 7: *Comparison of mean RRE vs total number of simulations in the tensor train for TT-ICE, TT-ICE* and ITTD[15] with two different ε settings. Please refer to Figure 5 for a description of the legend. TT-ICE* provides mean RRE closer to ε_{des} in both cases. ITTD Fails to compress the entire stream for both cases.*

Mean RRE tests conclude the experiments with self-oscillating gel simulations. Similar to experiments with ATARI data, TT-ICE and TT-ICE* algorithms yield reduced compression time, increased compression ratio, and improved execution stability over ITTD.

5. Conclusion. In this work, we proposed a new algorithm to incrementally update a TT-decomposition to compress a stream of data. Our algorithm TT-ICE improves on the existing state-of-the-art because (1) it maintains a desired error tolerance *for all data increments*, (2) it updates its ranks without excessive growth, and (3) it maintains orthogonality of the TT-cores to enable efficient projection and prediction for uncompressed data. We provide proof that the TT-ICE algorithm maintains its accuracy throughout the compression process. We then provide three heuristics to improve on this algorithm and show empirical evidence that they improve performance with little sacrifice in accuracy. This enhanced version of TT-ICE is also guaranteed to maintain the accuracy of the already compressed portion of the stream. However, no such guarantee can be provided for the portion of the stream compressed with TT-ICE* since heuristics are used in the first place. Experimental results on two different types of data demonstrate the superior performance of TT-ICE over ITTD and provide empirical proof of the additional benefits obtained from the heuristic upgrades we implemented in TT-ICE*. For simulation data, TT-ICE* achieves twice the compression ratio of TT-ICE in half the time. For image data, TT-ICE* achieves a comparable, if not higher, compression ratio with up to 80% reduction in the TT-ICE time. Moreover, at resource-limited hardware, TT-ICE and TT-ICE* have proven themselves to be reliable and performant compression methods.

Extensions of this work will attempt to theoretically justify the heuristics used in TT-ICE*. It will also deploy the proposed methods to enable scalable machine learning in the latent space identified by the compression, e.g., for inverse design [1] and behavioral cloning [5].

Acknowledgments. DA, AG, and SV acknowledge partial support from the Automotive Research Center at the University of Michigan (UM) in accordance with Cooperative Agreement W56HZV-19-2-0001 with U.S. Army DEVCOM Ground Vehicle Systems Center. DA and AG also acknowledge partial support from the Department of Energy Office of Scientific Research, ASCR, under grant DE-SC0020364. We thank Brian Chen for preparing the ATARI game dataset. The code for TT-ICE and TT-ICE* is publicly available on github.com/dorukaks/TT-ICE.

A note by DA: *This paper is dedicated to the loving memory of my grandmother Ayla Yaşar (1942-2022). I will miss making you Turkish coffee.*

REFERENCES

[1] D. Aksoy, S. Alben, R. D. Deegan, and A. A. Gorodetsky, *Inverse design of self-oscillatory gels through deep learning*, Neural Computing and Applications, 34 (2022), pp. 6879–6905, https://doi.org/10.1007/s00521-021-06788-9, https://doi.org/10.1007/s00521-021-06788-9.

[2] S. Alben, A. A. Gorodetsky, D. Kim, and R. D. Deegan, *Semi-implicit methods for the dynamics of elastic sheets*, Journal of Computational Physics, 399 (2019), p. 108952, https://doi.org/10.1016/j.jcp.2019.108952, https://doi.org/10.1016/j.jcp.2019.108952, https://arxiv.org/abs/1904.09198.

[3] A. Anaissi, B. Suleiman, and S. M. Zandavi, *NeCPD: An Online Tensor Decomposition with Optimal Stochastic Gradient Descent*, (2020), http://arxiv.org/abs/2003.08844, https://arxiv.org/abs/2003.08844.

[4] C. G. Baker, K. A. Gallivan, and P. Van Dooren, *Low-rank incremental methods for computing dominant singular subspaces*, Linear Algebra and Its Applications, 436 (2012), pp. 2866–2888, https://doi.org/10.1016/j.laa.2011.07.018, http://dx.doi.org/10.1016/j.laa.2011.07.018.

[5] B. Chen, S. Tandon, D. Gorsich, A. Gorodetsky, and S. Veerapaneni, *Behavioral cloning in atari games using a combined variational autoencoder and predictor model*, in 2021 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2021, pp. 2077–2084.

[6] A. A. Cook, G. Misirli, and Z. Fan, *Anomaly detection for iot time-series data: A survey*, IEEE Internet of Things Journal, 7 (2019), pp. 6481–6494.

[7] S. De, E. Corona, P. Jayakumar, and S. Veerapaneni, *Tensor-train compression of discrete element method simulation data*, arXiv preprint arXiv:2210.08399, (2022).

[8] Y. Du, Y. Zheng, K. C. Lee, and S. Zhe, *Probabilistic Streaming Tensor Decomposition*, Proceedings - IEEE International Conference on Data Mining, ICDM, 2018-November (2018), pp. 99–108, https://doi.org/10.1109/ICDM.2018.00025.

[9] C. Eckart and G. Young, *The approximation of one matrix by another of lower rank*, Psychometrika, 1 (1936), pp. 211–218, https://doi.org/10.1007/BF02288367, http://link.springer.com/10.1007/BF02288367.

[10] M. H. Engeli, *Bits and Spaces*, Architecture and and Computing for Physical, Virtual and Hybrid Realms, 37 (2001), p. 207, https://doi.org/10.5555/ICML.3045145, http://bitsandspaces.ethz.ch/acknowledgments.

[11] P. Giudici, B. Huang, and A. Spelta, *Trade networks and economic fluctuations in Asian countries*, Economic Systems, 43 (2019), p. 100695, https://doi.org/10.1016/j.ecosys.2019.100695, https://doi.org/10.1016/j.ecosys.2019.100695.

[12] J. Håstad, *Tensor rank is NP-complete*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 372 LNCS (1989), pp. 451–460, https://doi.org/10.1007/BFb0035776.

[13] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, *Stable baselines*. https://github.com/hill-a/stable-baselines, 2018.

[14] C. Jia, Y. Kong, Z. Ding, and Y. Fu, *Latent tensor transfer learning for RGB-D action recognition*, in MM 2014 - Proceedings of the 2014 ACM Conference on Multimedia, New York, NY, USA, nov 2014, ACM, pp. 87–96, https://doi.org/10.1145/2647868.2654928, http://link.springer.com/10.1007/978-1-4419-7142-5_3https://dl.acm.org/doi/10.1145/2647868.2654928.

[15] H. Liu, L. T. Yang, Y. Guo, X. Xie, and J. Ma, *An Incremental Tensor-Train Decomposition for Cyber-Physical-Social Big Data*, IEEE Transactions on Big Data, 7 (2018), pp. 341–354, https://doi.org/10.1109/tbdata.2018.2867485.

[16] M. W. Mahoney, *Randomized algorithms for matrices and data*, Foundations and Trends in Machine Learning, 3 (2010), pp. 123–224, https://doi.org/10.1561/2200000035, http://arxiv.org/abs/1104.5557http://www.nowpublishers.com/article/Details/MAL-035, https://arxiv.org/abs/1104.5557.

[17] E. Martínez-Montes, P. A. Valdés-Sosa, F. Miwakeichi, R. I. Goldman, and M. S. Cohen, *Concurrent EEG/fMRI analysis by multiway Partial Least Squares*, NeuroImage, 22 (2004), pp. 1023–1034, https://doi.org/10.1016/j.neuroimage.2004.03.038.

[18] L. Mirsky, *Symmetric gauge functions and unitarily invariant norms*, Quarterly Journal of Mathematics, 11 (1960), pp. 50–59, https://doi.org/10.1093/qmath/11.1.50, https://academic.oup.com/qjmath/article-lookup/doi/10.1093/qmath/11.1.50.

[19] F. Miwakeichi, E. Martínez-Montes, P. A. Valdés-Sosa, N. Nishiyama, H. Mizuhara, and Y. Yamaguchi, *Decomposing EEG data into space-time-frequency components using Parallel Factor Analysis*, NeuroImage, 22 (2004), pp. 1035–1045, https://doi.org/10.1016/j.neuroimage.2004.03.039.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, *Human-level control through deep reinforcement learning*, Nature, 518 (2015), pp. 529–533, https://doi.org/10.1038/nature14236, http://dx.doi.org/10.1038/nature14236.

[21] M. Nakatsuji, Q. Zhang, X. Lu, B. Makni, and J. A. Hendler, *Semantic Social Network Analysis by Cross-Domain Tensor Factorization*, IEEE Transactions on Computational Social Systems, 4 (2017), pp. 207–217, https://doi.org/10.1109/TCSS.2017.2732685.

[22] I. V. Oseledets, *Tensor-train decomposition*, SIAM Journal on Scientific Computing, 33 (2011), pp. 2295–2317, https://doi.org/10.1137/090752286, http://epubs.siam.org/doi/10.1137/090752286.

[23] A. Raffin, *Rl baselines zoo*. https://github.com/araffin/rl-baselines-zoo, 2018.

[24] S. Sizov, S. Staab, and T. Franz, *Analysis of Social Networks by Tensor Decomposition*, in Handbook of Social Network Technologies and Applications, B. Furht, ed., Springer US, Boston, MA, 2010, pp. 45–58, https://doi.org/10.1007/978-1-4419-7142-5_3, http://link.springer.com/10.1007/978-1-4419-7142-5http://link.springer.com/

10.1007/978-1-4419-7142-5_3.
[25] S. Smith, K. Huang, N. D. Sidiropoulos, and G. Karypis, *Streaming tensor factorization for infinite data sources*, in SIAM International Conference on Data Mining, SDM 2018, Philadelphia, PA, may 2018, Society for Industrial and Applied Mathematics, pp. 81–89, https://doi.org/10.1137/1.9781611975321.10, https://epubs.siam.org/doi/10.1137/1.9781611975321.10.
[26] A. Sobral, C. G. Baker, T. Bouwmans, and E. H. Zahzah, *Incremental and multi-feature tensor subspace learning applied for background modeling and subtraction*, in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 8814, 2014, pp. 94–103, https://doi.org/10.1007/978-3-319-11758-4_11, http://link.springer.com/10.1007/978-3-319-11758-4_11.
[27] L. T. Thanh, K. Abed-Meraim, N. L. Trung, and R. Boyer, *Adaptive algorithms for tracking tensor-train decomposition of streaming tensors*, European Signal Processing Conference, 2021-January (2021), pp. 995–999, https://doi.org/10.23919/Eusipco47968.2020.9287780.
[28] L. T. Thanh, K. Abed-Meraim, N. L. Trung, and A. Hafiane, *Robust tensor tracking with missing data under tensor-train format*, in 2022 30th European Signal Processing Conference (EUSIPCO), 2022, pp. 832–836, https://doi.org/10.23919/EUSIPCO55093.2022.9909702.
[29] L. T. Thanh, N. Linh-Trung, et al., *Streaming tensor-train decomposition with missing data*, (2022).
[30] M. Vandecappelle, N. Vervliet, and L. De Lathauwer, *Nonlinear least squares updating of the canonical polyadic decomposition*, 25th European Signal Processing Conference, EUSIPCO 2017, 2017-January (2017), pp. 663–667, https://doi.org/10.23919/EUSIPCO.2017.8081290.
[31] X. Wang, L. T. Yang, Y. Wang, L. Ren, and M. J. Deen, *ADTT: A Highly Efficient Distributed Tensor-Train Decomposition Method for IIoT Big Data*, IEEE Transactions on Industrial Informatics, 17 (2021), pp. 1573–1582, https://doi.org/10.1109/TII.2020.2967768.

Appendix A. Other Atari Games. This section includes the results of the experiments with the other ATARI games and their discussion.

TT-ICE and TT-ICE* proved their performance in the first experiments with Ms.Pac-Man frames. Next, we investigated the performance of algorithms across different hardware combinations.

For these experiments, we focus our attention on TT-ICE, TT-ICE*, and ITTD and to understand performance across different hardware combinations, we tested the algorithms on several machines. For each experiment, we detail the machine on which it was conducted. The machines are *Dell Precision 7820* workstation with Xeon Silver 4110 processor and 32 GB memory, *Dell XPS-15 9570* laptop with i7-8750H processor and 16 GB memory, and *Dell Precision 7820* workstation with Xeon Silver 4110 processor and 16 GB memory. The machines are referred to in Table A.1 as M1, M2, and M3, respectively. Table A.1 indicates improvements between 44.5× to 56.7× in compression and between 8.2× to 23.5× in time over ITTD5.

In Table A.1, TT-ICE and TT-ICE* complete the compression of streams faster than ITTD5. Furthermore, our proposed algorithms achieve at least an order of magnitude higher compression ratio than ITTD5. In Table A.1, ITTD5 fails without an exception to compress the entire stream due to insufficient memory when $\varepsilon_{des} = 0.01$. This is caused by the high computational requirements of the rounding step in ITTD5 and becomes prohibitive even with $\varepsilon_{des} = 0.1$ for games Beamrider and Enduro. The only case where all three methods fail to compress the entire stream is Enduro with $\varepsilon_{des} = 0.01$ due to the limited memory of the machine. However, even in that case, we see that TT-ICE can compress at least 3 times more frames than ITTD5.

An impressive point from Table A.1 is the difference in number of frames used between TT-ICE and TT-ICE*. By subselecting frames and skipping the update of the first d TT-cores, TT-ICE* reduces the number of frames used in updates by up to 97% (Pong with $\varepsilon_{des} = 0.1$) without any loss in mean RRE. The reduction in time is also parallel with the reduction in the frames used. TT-ICE* achieves at least 48% reduction (Seaquest with $\varepsilon_{des} = 0.01$) in execution time over TT-ICE.

In Table A.1, Beamrider with $\varepsilon = 0.01$ is another insightful example, which shows that implementing heuristics can be influential on the success of TT-ICE. In that case, TT-ICE* successfully completes the decomposition of the stream, whereas TT-ICE fails to complete the same task.

The additional experiments using ATARI data show that TT-ICE and TT-ICE* can successfully compress streams across different hardware combinations. Furthermore, they provide proof that the decrease in compression time over ITTD is not hardware dependent.

To provide a qualitative perspective on what each of the ε_{des} levels corresponds, we provide reconstructions of the frames compressed both with $\varepsilon_{des} = 0.1$ (Figure A.8) and $\varepsilon_{des} = 0.01$ (Figure A.9). In Figure A.8 we include ITTD5 reconstructions in addition to TT-ICE reconstructions. Figure A.8 shows that there are visual distortions on each reconstructed frame regardless of the algorithm used. These artifacts caused at

the error tolerance of $\varepsilon_{des} = 0.1$ can be as unobtrusive as discolored parts/objects (Breakout, Ms.Pac-Man) but also can be completely disruptive as invisible agents (Pong, Q*bert).

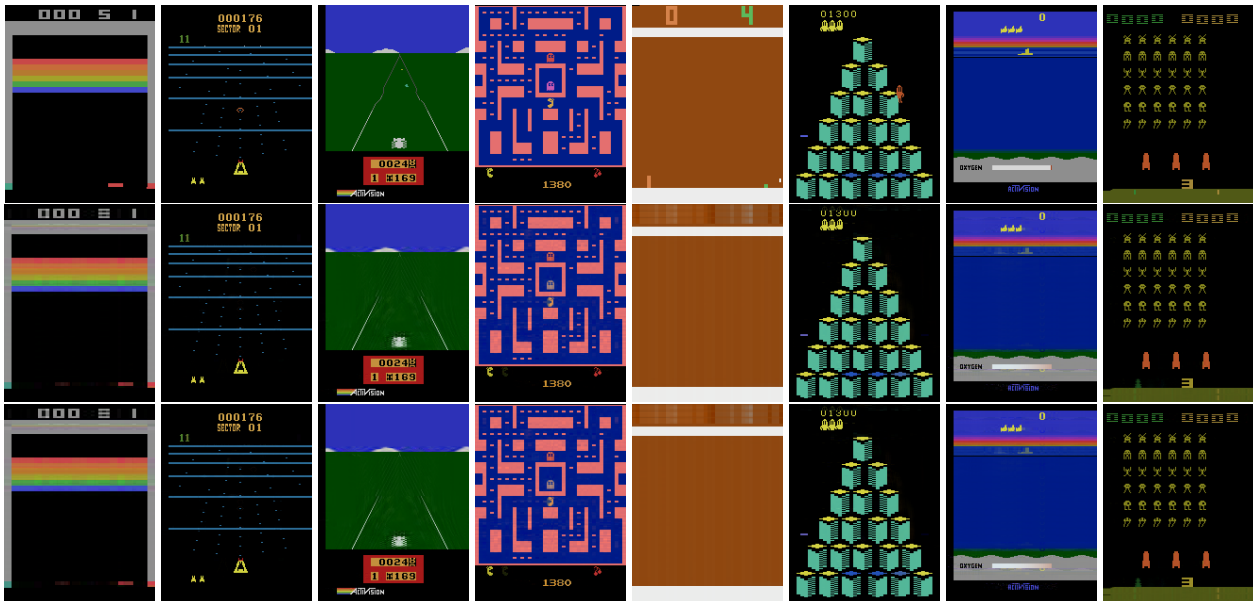


Fig. A.8: Original frames from the Atari dataset (top) along with their respective reconstructions from the TT-cores compressed using TT-ICE (middle) and ITTD5 (bottom) with $\varepsilon_{des} = 0.1$. There are visible visual artifacts on some of the frames. There is no visible difference between TT-ICE and ITTD5 reconstructions. Left-to-right: Breakout, Beam Rider, Enduro, Ms.Pac-Man, Pong, Q*bert, Seaquest, Space Invaders

As the error tolerance is reduced to $\varepsilon_{des} = 0.01$, the visual artifacts in the reconstruction disappear. In Figure A.9, we see that for $\varepsilon_{des} = 0.01$ the frames can be reconstructed with no visible disruptions. This also provides evidence that TT-cores trained with TT-ICE and TT-ICE* can be used as a method of storage. Unfortunately at this error tolerance level, we can not provide reconstructions of ITTD, as ITTD5 fails due to insufficient memory for all games.

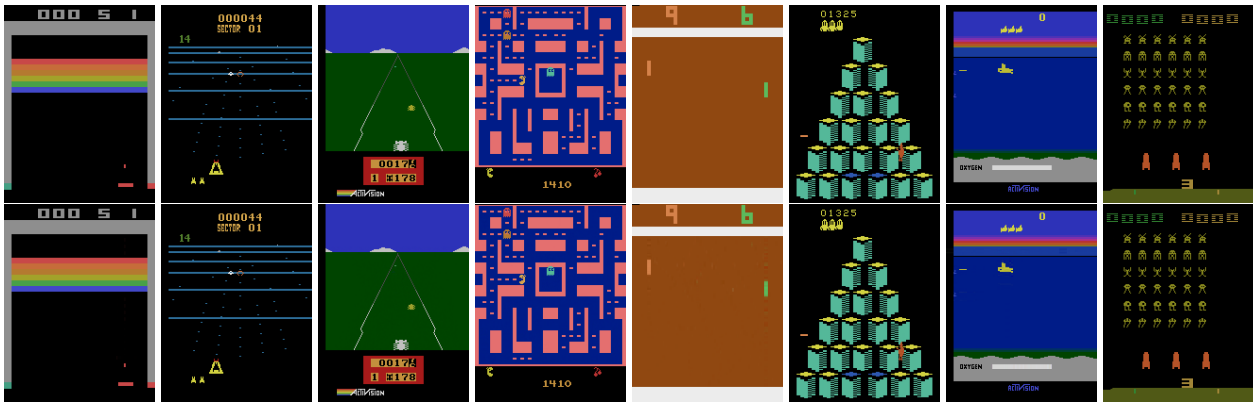


Fig. A.9: Original frames from the Atari dataset (top) along with their respective reconstructions (bottom) from the TT-cores compressed using TT-ICE with $\varepsilon_{des} = 0.01$. There are no visible artifacts on the frames. ITTD5 reconstructions are not included here since ITTD5 fails at all games for $\varepsilon_{des} = 0.01$. Left-to-right: Breakout, Beam Rider, Enduro, Ms.Pac-Man, Pong, Q*bert, Seaquest, Space Invaders

Table A.1: *Summary statistics for the other ATARI games. Game: name of the game, Machine: codename of the computer that conducted the experiments, Method: decomposition algorithm used (TT-ICE*: Algorithm 3.2, TT-ICE: Algorithm 3.1, ITTD: [15, Alg 3]), ε: the desired relative error bound for the given decomposition method, Frames: total number of the frames compressed in the TT-cores, Used: number of frames used to update the TT-cores, Time: total execution time of the algorithm, mean RRE: final mean RRE at the end of the experiment, CR: compression ratio. For each experiment, the best performance is shown in bold for Time, mean RRE, and CR. All failures are due to insufficient memory.*

Game	Machine	Method	ε	Frames	Used	Time	mean RRE	CR
Breakout	M2	TT-ICE	0.1	9295	9295	212.2	0.078	2748.1
		TT-ICE*	0.1	9295	2376	**58.9**	**0.084**	**3525.9**
		ITTD 5	0.1	9295	9295	801.2	0.083	66.55
Breakout	M2	TT-ICE	0.01	9295	9295	572.7	0.0075	9.7
		TT-ICE*	0.01	9295	4997	**198.5**	**0.0078**	**10.1**
		ITTD 5	0.01	436	436	Fails	Fails	Fails
Beamrider	M2	TT-ICE	0.1	42473	42473	2689.1	**0.089**	**130.7**
		TT-ICE*	0.1	42473	4017	**193.0**	0.087	122.3
		ITTD 5	0.1	22606	22606	Fails	Fails	Fails
Beamrider	M2	TT-ICE	0.01	26257	26257	Fails	Fails	Fails
		TT-ICE*	0.01	42473	22693	**2334.3**	**0.008**	**4.3**
		ITTD 5	0.01	4937	4937	Fails	Fails	Fails
Enduro	M1	TT-ICE	0.1	132880	132880	11158.8	**0.096**	**246.6**
		TT-ICE*	0.1	132880	10043	**940.1**	0.095	225.6
		ITTD 5	0.1	96329	96329	Fails	Fails	Fails
Enduro	M1	TT-ICE	0.01	49819	49819	Fails	Fails	Fails
		TT-ICE*	0.01	76396	40482	Fails	Fails	Fails
		ITTD 5	0.01	13289	13289	Fails	Fails	Fails
Q*bert	M3	TT-ICE	0.1	18742	18742	1063.6	0.089	817.7
		TT-ICE*	0.1	18742	3993	**180.7**	**0.094**	**968.4**
		ITTD 5	0.1	18742	18742	1477.5	**0.094**	18.0
Q*bert	M3	TT-ICE	0.01	18742	18742	1740.4	0.006	13.6
		TT-ICE*	0.01	18742	5272	**470.0**	**0.008**	**15.8**
		ITTD 5	0.01	2286	2286	Fails	Fails	Fails
Seaquest	M1	TT-ICE	0.1	36805	36805	1793.8	0.091	1176.8
		TT-ICE*	0.1	36805	3183	**173.1**	**0.096**	**1703.7**
		ITTD 5	0.1	36805	36805	2435.0	0.096	30.4
Seaquest	M3	TT-ICE	0.01	36805	36805	3907.0	0.007	3.1
		TT-ICE*	0.01	36805	21194	**2024.6**	**0.008**	**3.4**
		ITTD 5	0.01	6667	6667	Fails	Fails	Fails
Pong	M1	TT-ICE	0.1	96302	96302	1566.0	**0.066**	**99835.9**
		TT-ICE*	0.1	96302	2738	**70.0**	**0.066**	**99835.9**
		ITTD 5	0.1	96302	96302	1644.6	0.065	2234.6
Pong	M1	TT-ICE	0.01	96302	96302	12895.3	**0.009**	**48.3**
		TT-ICE*	0.01	96302	17219	**1283.2**	**0.009**	47.2
		ITTD 5	0.01	46015	46015	Fails	Fails	Fails

Appendix B. Comparison with TT-FOA. This section includes the comparison study between TT-ICE* and TT-FOA along with its discussion. In regimes where we found TT-FOA effective to run. We were not able to deploy TT-FOA on our problems of interest because it required significantly greater storage and computational requirements than TT-FOA or ITTD due to large-scale auxiliary matrices. For example,

the TT-ranks used to approximate the tensors in [27] are very low in comparison to the final TT-ranks that we obtain in our numerical experiments. As a result, we performed a comparison using a similar testbed to that in the TT-FOA paper. To be able to provide a fair comparison between TT-ICE* and TT-FOA, we implemented the Matlab script provided by authors of [27] in python[4] and benchmarked against TT-ICE* using a synthetic 4-dimensional tensor with size $10 \times 15 \times 20 \times 500$ with TT-ranks $[1, 2, 3, 5, 1]$, which is same as the study conducted in [27].

We considered 3 scenarios for TT-FOA: 1) TT-FOA initialized with the true TT-ranks, 2) TT-FOA initialized with underestimated TT-ranks, and 3) TT-FOA initialized with overestimated TT-ranks. On the other hand, for TT-ICE* we set $\varepsilon_{des} = 10^{-8}$ since the synthetic tensor is exactly low rank. We then constructed 20 random 4-dimensional tensor streams having increments of size $10 \times 15 \times 20$.

Figure B.10 shows the relative error of the approximation at each increment step averaged over 20 repetitions of the experiment. A comparison of the compression ratio of methods is not provided here since the TT-ranks are assumed known from the beginning for TT-FOA.

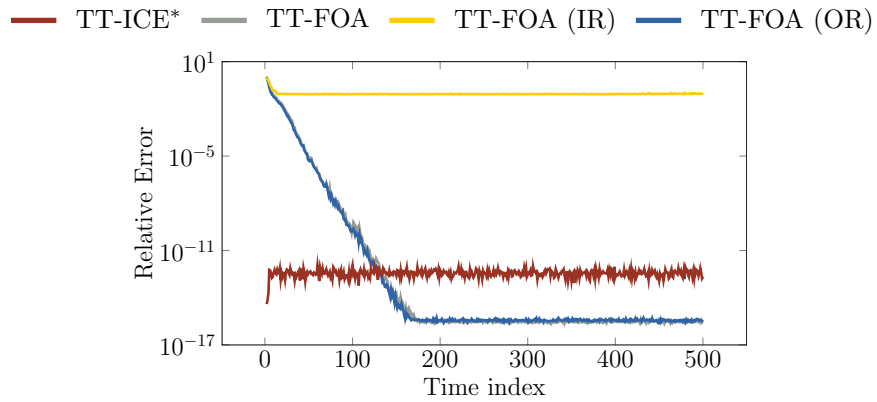


Fig. B.10: Comparison of relative errors of TT-ICE* and TT-FOA[27] averaged over 20 repetitions. TT-ICE*: TT-ICE* with $\varepsilon_{des} = 10^{-8}$, TT-FOA: TT-FOA initialized with correct TT-ranks, TT-FOA (IR): TT-FOA initialized with insufficient TT-ranks, TT-FOA (OR): TT-FOA initialized with overestimated TT-ranks. Unlike TT-FOA, TT-ICE* achieves a low relative error without the need for a convergence period and discovers the correct underlying TT-ranks. In case of underestimated TT-ranks, TT-FOA fails to achieve low relative error.

Figure B.10 shows that TT-FOA requires a convergence period before reaching a stable relative error even when it is initialized with correct TT-ranks. In the case of underestimated TT-ranks, TT-FOA converges to a higher relative error depending on the difference in the estimated and actual TT-ranks. TT-FOA converges to the same level of relative error as the correctly estimated case when the TT-ranks are overestimated. This behavior is expected for this set of experiments since we have an exactly low-rank tensor stream. On the other hand, TT-ICE* neither requires estimation of the TT-ranks beforehand, nor needs a convergence period to achieve the desired level of relative error.

The difference between both methods becomes evident when the time to compress the streams is considered. TT-FOA performs Kronecker products and matrix inversions, and the size of the matrices involved in those operations plays a decisive role in the speed of the algorithm. Therefore, there is a notable difference in computation time between different TT-rank estimations. When the correct TT-ranks are estimated ([1,2,3,5,1]), compression of the streams takes on average $1.8070s$. This time becomes $1.5543s$ when TT-ranks are underestimated as [1,1,2,3,1] and increases to $4.1290s$ when TT-ranks are overestimated as [1,3,5,10,1]. TT-ICE* compresses the entire stream in $0.5806s$ on average.

[4]The accuracy of the python implementation was verified by comparing the results of our python implementation and the published Matlab script using the same 4-dimensional tensor streams. To time TT-FOA fairly in the following experiments, we removed the relative error computation that was included in the Matlab implementation of the algorithm and computed the relative error separately.

When the same experiment is repeated with a tensor of the same size but with TT-ranks [1,4,10,30,1], TT-FOA requires $314.53s$ on average with correct TT-ranks. This time can get as low as $14.965s$ when TT-FOA is initialized with underestimated TT-ranks [1,3,5,20,1] and can get as high as $628.70s$ with overestimated TT-ranks [1,6,15,40,1]. On the other hand, TT-ICE* scales very well in this scenario and compresses the entire stream in $0.6656s$ on average.