Compressing multivariate functions with tree tensor networks

Joseph Tindall, E. Miles Stoudenmire, and Ryan Levy
Center for Computational Quantum Physics, Flatiron Institute, New York, NY, 10010, USA

(Dated: December 10, 2025)

Tensor networks are a compressed format for multi-dimensional data. One-dimensional tensor networks—often referred to as tensor trains (TT) or matrix product states (MPS)—are increasingly being used as a numerical ansatz for continuum functions by "quantizing" the inputs into discrete binary digits. Here we demonstrate the power of more general tree tensor networks for this purpose. We provide direct constructions of a number of elementary functions as generic tree tensor networks and interpolative constructions for more complicated functions via a generalization of the tensor cross interpolation algorithm. For a range of multi-dimensional functions we show how more structured tree tensor networks offer a significantly more efficient ansatz than the commonly used tensor train. We demonstrate an application of our methods to solving multi-dimensional, non-linear Fredholm equations, providing a rigorous bound on the rank of the solution which, in turn, guarantees exponentially scaling accuracy with the size of the tree tensor network for certain problems.

I. Introduction

High-dimensional data structures are ubiquitous in the modern sciences. They have an inherit exponential scaling with the number of dimensions, making any direct "brute force" approach to their representation quite limited. Tensor networks are a compression of high-order tensors into an interconnected collection of smaller tensors [1–14]. When the data possesses certain low-rank structure this compression can be extremely effective and turn an exponential-scaling problem into a polynomial one. The most common tensor networks take the form of one-dimensional chains of order-three tensors known as tensor trains (TT) or matrix product states (MPS). Their effectiveness has been demonstrated for a number of scientific problems ranging from one-dimensional quantum physics [15–17] to modeling the spread of disease [18, 19].

Tensor trains also offer a somewhat unconventional numerical methodology for problems in continuous space [20–31]. Using an encoding of the relevant continuous variables into binary strings of length L, mathematical functions on a grid with spacing $\mathcal{O}(\exp(-L))$ can be represented with a train of $\mathcal{O}(L)$ order-three tensors. Such an ansatz is commonly referred to as a quantics tensor train (QTT) and has opened up a new field of tensor train-based numerical methods. Whilst tensor trains are known to be highly effective for smooth, one-dimensional functions [32], higher-dimensional functions can pose significant difficulties, typically requiring much larger ranks for the tensors in the train and thus larger computational resources. Other than mathematical and computational simplicity, however, there is no reason to limit these tensor-based numerical methods to trains. Tensor networks of more complex topology—which have proven fundamental in the field of two- and three-dimensional quantum simulation [33–38]—offer a whole new degree of freedom, allowing more structured, complex correlations to be encoded between the underlying variables (which, in this context, are the binary digits). Several works have considered ansatzes beyond the tensor train for representing multivariate functions. Specifically: multiple tensor trains coupled via their leading tensors [39, 40], "functional" tensor trains where the individual tensors in the train constitute matrix-valued functions [41, 42] or heirarchical tucker decompositions where the binary variables are placed solely at the leaves of the tree and correlated with each other via their parents [43, 44]. Very few methods are available in this domain, however, for working with tensor networks of more generic topology and little is understood about how their structure determines their effectiveness at representing a given continuous function.

In this work we rectify this lack of information and methods for working with higher-dimensional tensor networks in the context of representing continuous functions and solving numerical problems. We focus on tree tensor networks (TTNs) as the absence of loops guarantees they can be contracted with computational resources scaling polynomially in the network parameters. First, we introduce direct constructions of several elementary functions, including polynomials, on arbitrary tree tensor networks with tensor ranks bounded independent of the network. We then describe a generalization of the tensor cross interpolation algorithm [29, 30, 45, 46] to *any* tree tensor network — allowing the active learning of general, multi-dimensional target functions $f(\mathbf{x})$ into a TTN format. We benchmark these methods for various functions — showing in the multi-dimensional case how more structured TTNs can be a significantly more effective ansatz than tensor trains. Finally, we introduce a new iterative tree tensor network-based solver for Fredholm integral equations: showing how the size of the tensors in the final output of the solver can be bounded in terms of the size of the tensors in the integral kernel, guaranteeing the effectiveness of the method for kernels which can be represented as a tree tensor network with fixed internal dimensions.

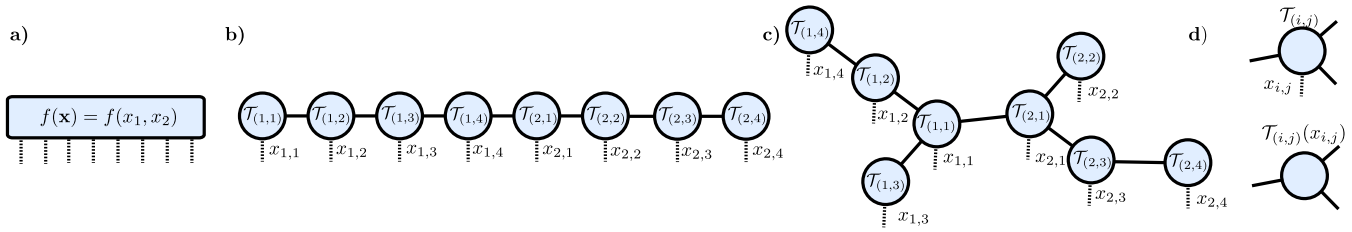


FIG. 1. **a-c)** Representations of the two dimensional function $f(\mathbf{x}) = f(x_1, x_2)$ with the continuous variables $x_1, x_2 \in [0, 1)$ encoded as binary strings $x_1 = 0.x_{1,1}x_{1,2}x_{1,3}x_{1,4}$ and $x_2 = 0.x_{2,1}x_{2,2}x_{2,3}x_{2,4}$ of length four. **a)** The actual values of the function over the domain specified by the binary digits can be encoded as a single order 8 tensor. **b**) Quantics tensor train or Matrix Product State representation of the order 8 tensor with a binary digit ordering commonly referred to as 'sequential'. **c**) An example of a tree tensor network (TTN) representation of the order 8 tensor. **d)** Top: We use the notation $\mathcal{T}_{(i,j)}$ to refer to the local tensor in a tree corresponding to the binary digit $x_{i,j}$ – the jth most significant binary digit in the decomposition of the ith continuous variable x_i. This tensor has $z_{(i,j)}$ 'virtual' indices (black lines) connecting it to its neighbors in the tree and a single external index (dotted line) corresponding to the binary variable $x_{(i,j)} \in \{0, 1\}$. Bottom: We use $\mathcal{T}_{(i,j)}(x_{i,j})$ to refer to the order $z_{(i,j)}$ tensor which is a slice of $\mathcal{T}_{(i,j)}$ for the given value of $x_{i,j}$.

II. Preliminaries

A tensor $\mathcal{T}_v$ is a function which maps discrete variables (which we refer to as its indices) to numbers, i.e. $\mathcal{T}_v(a_1, a_2, \ldots) \in \mathbb{C}$ with $a_i \in [1, 2, \ldots, d_i]$. A tensor network is an interconnected network of tensors: each vertex of the network $v \in V$, with V the set of vertices of the network, hosts a tensor $\mathcal{T}_v$ and the edges of the network dictate which tensors share common 'virtual', 'internal', or 'bond' indices that are implicitly summed over. The maximum dimension of any of the virtual (or bond) indices in the network is referred as the bond dimension or rank χ of the tensor network. Each tensor of a tensor network can also have external indices not common to any other tensors in the network. Defining $\mathbf{I}$ as the set of all indices in the network, $\boldsymbol{\alpha}$ as the set of all indices which appear twice (i.e. are shared by two tensors) and $\mathbf{x} = \mathbf{I} \setminus \boldsymbol{\alpha}$ as the set of all external indices we have

$$\mathcal{T}(\mathbf{x}) = \sum_{\boldsymbol{\alpha}} \prod_{v \in V} \mathcal{T}_v(\mathbf{I_v}), \qquad \mathbf{I_v}, \boldsymbol{\alpha} \subseteq \mathbf{x}, \tag{1}$$

with $\mathbf{I_v}$ the indices possesed by the tensor $\mathcal{T}_v$. The tensor network $\mathcal{T}$ thus corresponds to a factorization of a single tensor with indices $\mathbf{x}$. Some example tensor networks are illustrated in Fig. 1b) and c) along with the tensor they are representing in Fig. 1a).

In this work the external indices of the network represent discrete variables which decompose a series of n continuous variables $\mathbf{x} = (x_1, x_2, ..., x_n) \in [0, 1)^n$ in a binary manner. For a given continuous variable x_i the binary decomposition reads $x_i = \sum_{j=1}^{L} \frac{x_{i,j}}{2^j}$ where the $x_{i,j} \in \{0, 1\}$ are the binary *variables* or *bits* which are each represented by an external index of dimension 2 in the tensor network [47]. The 2^{nL} possible binary strings, or configurations of the bits, realises a uniform discrete grid for $\mathbf{x} \in [0, 1 - \delta]^n$ where $\delta = 2^{-L}$ is the grid spacing in each dimension which is exponentially 'fine' in the number of bits. Unless specified otherwise we will focus on the case where each tensor in the network contains one external index and therefore corresponds to one binary digit $x_{i,j}$ in the decomposition of a continuous variable x_i. We will focus exclusively on tensor networks which are trees, i.e. networks where the virtual indices do not form loops. We should emphasize that this is related, but not the same as the previously introduced hierarchical tucker format [43, 44, 48] where the binary variables live only on the leaves of the tensor network and tensors in the bulk are simply factors that mediate correlations between them and do not contain the external variables. The fact the tensor networks we consider do not contain loops this means that they can be optimised and contracted (for a given configuration of their binary variables) efficiently and we will refer to them as tree tensor networks (TTNs).

The TTNs in this paper have a structure specified by a labelled tree $\mathcal{T}$ where each of the vertices corresponds to a single binary digit $x_{i,j}$ in the decomposition of x_i. We will use the notation $\mathcal{T}_{(i,j)}$ to denote the tensor on a given vertex and $\mathcal{T}_{(i,j)}(x_{i,j})$ to refer to a given 'slice' of that tensor $\mathcal{T}_{(i,j)}$ for a specific value of $x_{i,j} \in \{0, 1\}$, effectively viewing it as a 'function' of the binary variable $x_{i,j}$. Each $\mathcal{T}_{(i,j)}(x_{i,j})$ is just another tensor and has order $z_{i,j}$: the co-ordination number of the given vertex $x_{i,j}$ in the tensor network. This idea is illustrated in Fig. 1d. We refer to the $z_{i,j}$ indices connecting a local tensor to its neighbors as 'internal' or 'virtual' indices. We frequently use the notation $\boldsymbol{\alpha}_{i,j} = (\alpha_1, \alpha_2, \ldots \alpha_{z_{i,j}})$ to denote the virtual indices on the tensor $\mathcal{T}_{(i,j)}$ and index them starting from 0, i.e. $\alpha_k = 0, 1, 2, \ldots, \dim(\alpha_k)$.

The tensor network effectively encodes the values for some function $f(\mathbf{x})$ over the uniform discrete grid $\mathbf{x} \in [0, 1-\delta]^n$ with 2^{nL} grid points. A fixed 'configuration' of its external indices uniquely specifies a value for $\mathbf{x}$. The contraction of

the resulting network yields the scalar $\mathcal{T}(\mathbf{x})$ that approximates $f(\mathbf{x})$. Such a contraction can be done in $\mathcal{O}(nL\chi^z)$ time, where z is the maximum co-ordination number of any of the tensors in the network: i.e. $z = \sup\{z_{1,1}, z_{1,2}, \ldots, z_{n,L}\}$. We illustrate two example tree tensor networks in Fig. 1 as decompositions of a two-dimensional function. In this work we will provide methods for constructing functions as a tensor network with *any* choice of labelled tree $\mathcal{T}$, allowing us to compare the effectiveness of different tree structures.

III. Constructing functions as tree tensor networks

Here we detail how to construct various functions as tree tensor networks of generic topology. We first describe a direct methodology for certain functions via explicit setting of the tensor elements in the network and provide rules for adding and multiplying functions which greatly expands the space of functions which can be *exactly* represented. We then provide an indirect methodology for more generic functions via the tensor cross interpolation algorithm, which variationally minimizes the infinity norm between the tree tensor network and some desired function.

A. Direct construction

Certain elementary functions are factorizable as a product of separate functions for each bit $f(\mathbf{x}) = \prod_{i=1}^{n}\prod_{j=1}^{L} f_{(i,j)}(x_{i,j})$. The corresponding tensor network thus has rank or bond dimension one, with local tensors $\mathcal{T}_{(i,j)}(x_{i,j}) = f_{(i,j)}(x_{i,j})$ with either no virtual indices or virtual indices of dimension one, allowing them to be "trivially" represented on any desired topology. Three such classes of rank-one functions are:

- *Constant functions*: $f(\mathbf{x}) = c = \prod_{i=1}^{n}\prod_{j=1}^{L} c^{\frac{1}{nL}}$.
- *Exponential functions*: $f(\mathbf{x}) = ce^{\mathbf{k}\cdot\mathbf{x}+a} = \prod_{i=1}^{n}\prod_{j=1}^{L} c^{\frac{1}{nL}} e^{k_i \frac{x_{i,j}}{2^j}+a}$ with $a, c \in \mathbb{C}$ and $\mathbf{k} = (k_1, k_2, ..., k_n) \in \mathbb{C}^n$.
- *Dirac delta function*: $f(\mathbf{x}) = \delta(\mathbf{x} - \tilde{\mathbf{x}}) = \prod_{i=1}^{n}\prod_{j=1}^{L} 2\delta_{x_{i,j},\tilde{x}_{i,j}}$ where $\delta_{n,m}$ is the Kronecker delta function and $\tilde{x}_{i,j}$ is the setting of digit (i,j) in the binary decomposition of $\tilde{\mathbf{x}}$.

Polynomials - A more non-trivial case is that of polynomials of degree d. Here we will provide a construction of the one-dimensional degree d polynomial $p(x) = \sum_{k=0}^{d} c_k x^k$ with $c_0, c_1, \ldots c_d \in \mathbb{C}$ on any tree tensor network where the bond dimension will be $\chi = d + 1$, independent of the choice of tree. As we are working in one dimension we will drop the dimension subscript i in our notation for $x_{i,j}$ and the local tensor $\mathcal{T}_{(i,j)}$, i.e. $x_{i,j} \to x_j$ and $\mathcal{T}_{(i,j)} \to \mathcal{T}_j$.

First we pick *any* of the binary digits for the continuous variable x and designate it as x_r. For $j \neq r$ the local tensor is $\mathcal{T}_j$ and we will denote its elements as $\mathcal{T}_j(x_j)_{\alpha_1,\alpha_2,\ldots,\alpha_{z_j-1},\beta}$ where β is the virtual index corresponding to the edge which separates x_j from x_r and the $\alpha_1, \alpha_2, \ldots, \alpha_{z_j-1}$ denote the remaining virtual indices of the tensor. For $j = r$ we define the on-site tensor $\tilde{\mathcal{T}}_r$ and its elements as $\tilde{\mathcal{T}}_r(x_r)_{\alpha_1,\alpha_2,\ldots,\alpha_{z_r}}$.

The elements of the tensors in the network are then

$$\begin{aligned}
&\mathcal{T}_j(x_j)_\beta = \left(\frac{x_j}{2^j}\right)^\beta \qquad j \neq r,\ z_j = 1\\
&\mathcal{T}_j(x_j)_{\alpha_1,\alpha_2,\ldots,\alpha_{z_j-1},\beta} = C_{\alpha_1,\alpha_2,\ldots,\alpha_{z_j-1},\beta}\left(\frac{x_j}{2^j}\right)^{f_{\alpha_1,\alpha_2,\ldots,\alpha_{z_j-1},\beta}} \qquad j \neq r,\ z_{i,j} > 1\\
&\tilde{\mathcal{T}}_r(x_r)_{\alpha_1,\alpha_2,\ldots,\alpha_{z_r}} = \sum_{\beta=0}^{d} c_\beta C_{\alpha_1,\alpha_2,\ldots,\alpha_{z_r},\beta}\left(\frac{x_r}{2^r}\right)^{f_{\alpha_1,\alpha_2,\ldots,\alpha_{z_r},\beta}}
\end{aligned} \tag{2}$$

where we have introduced

$$f_{a_1,a_2,\ldots,a_n,b} = b - \sum_{i=1}^{n} a_i \tag{3}$$

and

$$C_{a_1,a_2,\ldots,a_n,b} = \begin{cases} \binom{b}{a_1} & n = 1 \text{ and } f_{a_1,a_2,\ldots,a_n,b} \geq 0 \\ \binom{b}{a_1}\prod_{i=1}^{n-1}\binom{f_{a_1,a_2,\ldots,a_{n-i},b}}{a_{n-i}} & n > 1 \text{ and } f_{a_1,a_2,\ldots,a_n,b} \geq 0 \\ 0 & \text{Otherwise} \end{cases} \tag{4}$$

for arbitrary integers $a_1, a_2, \ldots a_n$ and b. The c_β are the coefficients of the polynomial. In the supplementary material we prove that this tree tensor network will contract to the one-dimensional polynomial $f(x)$ for any configuration of its external indices. We also describe how to elevate the construction to the multidimensional case $f(\mathbf{x}) = p(x)$ with $x \in \{x_1, x_2, \ldots, x_n\}$ when there are external indices which decompose continuous variables other than x. We emphasize that our construction here is completely general and works on any tree: in the case the tree forms a one-dimensional path our result reduces to the known quantics tensor train construction [24, 32]

Multiplication and Addition

The direct constructions above can be combined with rules for multiplying and adding together tensor networks to vastly expand the space of possible functions which can be realised. We detail these below for a generically structured tensor network, emphasizing that in the case the tree forms a one-dimensional path our rules reduce to the well-established formalism for adding and multiplying QTTs [49, 50].

Addition - Consider two tensor networks which are defined over the same labelled tree $\mathcal{T}$ and encode two functions $t_1(\mathbf{x})$ and $t_2(\mathbf{x})$ and have bond dimensions $\chi^{(1)}$ and $\chi^{(2)}$. We define their local tensors as $\mathcal{T}^{(1)}_{(i,j)}$ and $\mathcal{T}^{(2)}_{(i,j)}$. The external index on a given vertex is common is between the two networks (i.e. it encodes the same binary digit) but virtual indices are not. We define the 'addition' of the two tensor networks as a new tensor network over the same labelled tree $\mathcal{T}$ with on-site tensors $\tilde{\mathcal{T}}_{(i,j)}$ which are defined via $\tilde{\mathcal{T}}_{(i,j)}(x_{i,j}) = \mathcal{T}^{(1)}_{(i,j)}(x_{i,j}) \oplus \mathcal{T}^{(2)}_{(i,j)}(x_{i,j})$ where $\oplus$ represents the tensor direct sum. By the definition of the direct sum, the dimension of the virtual indices of $\tilde{\mathcal{T}}_{(i,j)}$ is the sum of the two corresponding virtual indices in $\mathcal{T}^{(1)}_{(i,j)}$ and $\mathcal{T}^{(2)}_{(i,j)}$. It follows that, for a given configuration of the external indices of the new network, the contraction will yield $\tilde{t}(\mathbf{x}) = t_1(\mathbf{x}) + t_2(\mathbf{x})$ and the bond dimension of the new network is $\chi = \chi^{(1)} + \chi^{(2)}$.

Multiplication - Consider again two tensor networks which are defined over the same labelled tree $\mathcal{T}$ and represent two functions $t_1(\mathbf{x})$ and $t_2(\mathbf{x})$ and have bond dimensions $\chi^{(1)}$ and $\chi^{(2)}$. We define the 'multiplication' of the two networks as a new tensor network over the same labelled tree $\mathcal{T}$ with resulting on-site tensors $\tilde{\mathcal{T}}_{i,j}$ which are defined via $\tilde{\mathcal{T}}_{i,j}(x_{i,j}) = \mathcal{T}^{(1)}_{(i,j)}(x_{i,j}) \otimes \mathcal{T}^{(2)}_{i,j}(x_{i,j})$, where $\otimes$ denotes the tensor outer product. The tensor $\tilde{\mathcal{T}}_{i,j}$ thus has $2z_{i,j}$ virtual indices, and one external index corresponding to the digit $x_{i,j}$. There are two virtual indices for each edge in $\mathcal{T}$ and these pairs of indices can be combined together into a single index to recover a tensor network over $\mathcal{T}$ but with the dimension of the virtual index on a given edge being the product of the dimension of the corresponding indices for that in the original tensor networks. The bond dimension of the new network is thus $\chi = \chi^{(1)}\chi^{(2)}$. It follows that, for a given configuration $\mathbf{x}$ of the external indices the new tensor network contracts to $\tilde{t}(\mathbf{x}) = t_1(\mathbf{x})t_2(\mathbf{x})$.

Example - The hyperbolic function $f(\mathbf{x}) = c\cosh(\mathbf{k}\cdot\mathbf{x} + a)$ with $a, c \in \mathbb{C}$ and $\mathbf{k} = (k_1, k_2, ..., k_n) \in \mathbb{C}^n$ can be built as a tensor network over *any* labelled tree $\mathcal{T}$ with $\chi = 2$ by combining the exponential definition and the rule for addition. The local tensor elements are

$$\mathcal{T}_{(i,j)}(x_{ij})_{\alpha_1,\alpha_2,\ldots\alpha_{z_{i,j}}} = \begin{cases} \left(\frac{c}{2}\right)^{\frac{1}{nL}} \exp(k_i \frac{x_{i,j}}{2^j} + a) & \alpha_1 = \alpha_2 = \ldots = \alpha_{z_{i,j}} = 0 \\ \left(\frac{c}{2}\right)^{\frac{1}{nL}} \exp(-k_i \frac{x_{i,j}}{2^j} - a) & \alpha_1 = \alpha_2 = \ldots = \alpha_{z_{i,j}} = 1 \\ 0 & \text{Otherwise.} \end{cases} \tag{5}$$

B. Interpolative Construction - Tensor Cross Interpolation

The *tensor cross interpolation* (TCI) algorithm, also known as TT-cross, computes or "learns" a tensor network $\mathcal{T}(\mathbf{x})$ which interpolates a "target" function $f(\mathbf{x})$ [45, 51–54]. Assuming that the function can be computed efficiently for arbitrary inputs, the TCI algorithm queries the function at adaptively determined points known as "pivots" to improve tensors in the network, attempting to minimize the infinity norm $\sup_{\mathbf{x}}(|f(\mathbf{x}) - \mathcal{T}(\mathbf{x})|)$ while dynamically adapting the ranks of the network. Though TCI is formally defined for functions of discrete variables $f(i_1, i_2, \ldots, i_n)$, it can be applied to continuum functions by approximating continuous inputs $\mathbf{x}$ as binary index collections as in Sec. II.

TCI is conventionally formulated for tensor trains (tensor networks with a one-dimensional topology), but here we generalize it to arbitrary tree networks. To understand the tree generalization of TCI, it is useful to define a tensor network gauge we call the "interpolative gauge". In this gauge, some specific local tensor $\mathcal{T}_{(i,j)}$, known as the "center" tensor, has the property that each of its elements corresponds exactly to an entry of the "full" tensor represented by the entire network (i.e. that would be formed by contracting all of the tensors in the network together). The other tensors in the network act to interpolate the values of the full tensor not contained in the center tensor. Thus the

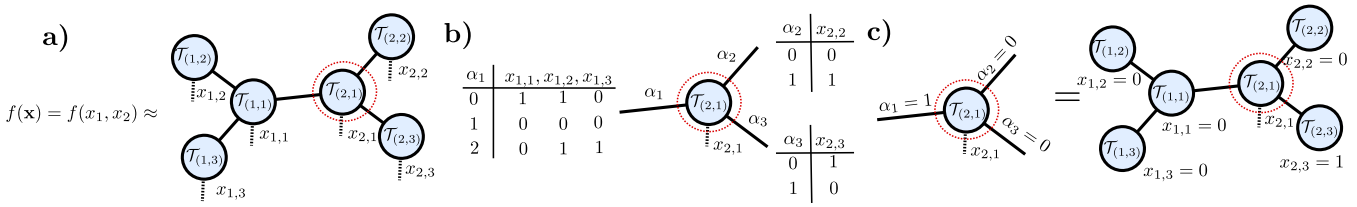


FIG. 2. The interpolative gauge. **a)** Tree tensor network for a two dimensional function $f(\mathbf{x}) = f(x_1, x_2)$ with the external indices decomposing the two continuous variables into binary strings of length three. **b)** The settings on the virtual indices $\alpha_1, \alpha_2, \alpha_3, \ldots$ of a local tensor map to configurations of the binary digits which the given edge connects to that local tensor. An example mapping is given by the tables shown. **c)** In the interpolative gauge, one of the tensors in the TTN has the property that their elements, for a given configuration of their virtual indices and external index, are equivalent to the contraction of the whole network for a specific setting of all the binary digits. This information is encoded via the mapping between virtual indices and external ones. The tensor cross interpolation (TCI) algorithm is an active learning algorithm which utilizes this gauge to move through the network, changing the gauge centre and updating neighboring pairs of local tensors in order minimize the infinity norm between their values $\mathcal{T}_{(i,j)}(x_{i,j})_{\boldsymbol{\alpha}}$ and some set of interpolation points of a target function $f(\mathbf{x})$.

interpolative gauge gives one access to or knowledge of certain values of the (exponentially large) full tensor through smaller, local network tensors.

Going into more detail, the center tensor has elements $\mathcal{T}_{(i,j)}(x_{i,j})_{\boldsymbol{\alpha}_{i,j}}$, where $\boldsymbol{\alpha}_{i,j} = (\alpha_1, \alpha_2, ..., \alpha_{z_{i,j}})$ are special values of the virtual indices of $\mathcal{T}_{(i,j)}$. In the interpolative gauge, additional "pivot" information is stored alongside these values saying how they map onto settings of the external indices. Figure 2a) shows a network whose center tensor is $\mathcal{T}_{(2,1)}$. Pivot entries are shown for the virtual indices in Fig. 2b), for example setting $\alpha_1 = 0$ corresponds to setting $(x_{1,1}, x_{1,2}, x_{1,3}) = (1, 1, 0)$. In particular, as illustrated in Fig. 2(c), the elements $\mathcal{T}_{(2,1)}(x_{2,1})_{\boldsymbol{\alpha}_{1,2}=(1,0,0)}$ of the center tensor correspond exactly to the full tensor values $\mathcal{T}(0, 0, 0, x_{2,1}, 0, 1)$.

On a tree tensor network, the TCI algorithm starts by making an initial guess for the tree tensor network and bringing it into the interpolative gauge. After choosing some tensor to be the root of the tree, one matricizes the leaf tensors and computes an interpolative decomposition of the resulting matrices. The interpolative decomposition (ID) of a matrix M is a factorization $M \approx CZ$ such that the columns of C are specific columns of the matrix M and Z interpolates any remaining columns of M not contained in C. If the matrix M has approximate rank r, then C can be chosen to have slightly more than r columns. (For further discussion of computing ID factorizations with the fewest number of columns, see Refs. [54, 55].) After computing the ID of the leaf tensors, they are replaced by the Z matrices and the C matrices are contracted into the parent tensor up the tree. The algorithm continues by next computing the ID of these parent tensors and multiplying the C tensors toward the root until all of the network consists of interpolating Z tensors.

Once in the interpolative gauge, the interpolation quality can be improved by contracting the center tensor with a neighboring tensor. The resulting combined tensor denoted Π no longer contains exact values of the full tensor, but is only an interpolation. This fact allows one to check point-wise how well the interpolation is matching the target function, and values which deviate by too much can be replaced by exact values obtained by calling the function. More efficient update strategies, which we do not use this work, can be defined such as "rook pivoting" or "block rook pivoting" [52, 54]. After the update, an interpolative decomposition is performed on the Π tensor to restore the interpolative gauge and move the center to the neighboring location. The full algorithm proceeds by contracting the new center with another neighbor, updating, and so on until every bond of the tree is visited once, comprising a single full "sweep" of the algorithm.

IV. Numerical Results for Function Construction

In this section we will compare the effectiveness of different tree tensor network topologies for representing a target function $f(\mathbf{x})$, using both direct methods and the tree tensor cross interpolation algorithm. The structure of the tensor network is specified by a labelled tree $\mathcal{T}$ and we will assess the effectiveness of such a tree by the error measures

$$\epsilon = \frac{1}{|g|} \sum_{\mathbf{x} \in g} |f(\mathbf{x}) - \mathcal{T}(\mathbf{x})|, \qquad \epsilon_\infty = |f(\mathbf{x}) - \mathcal{T}(\mathbf{x})|_\infty = \sup_{\mathbf{x} \in g} |f(\mathbf{x}) - \mathcal{T}(\mathbf{x})| \tag{6}$$

where $\mathcal{T}(\mathbf{x})$ corresponds to the contraction of the given tensor network for a specified configuration $\mathbf{x}$ of the external indices. Here, g is a randomly chosen subset of the full 2^{nL} grid points which is taken to be sufficiently large to avoid

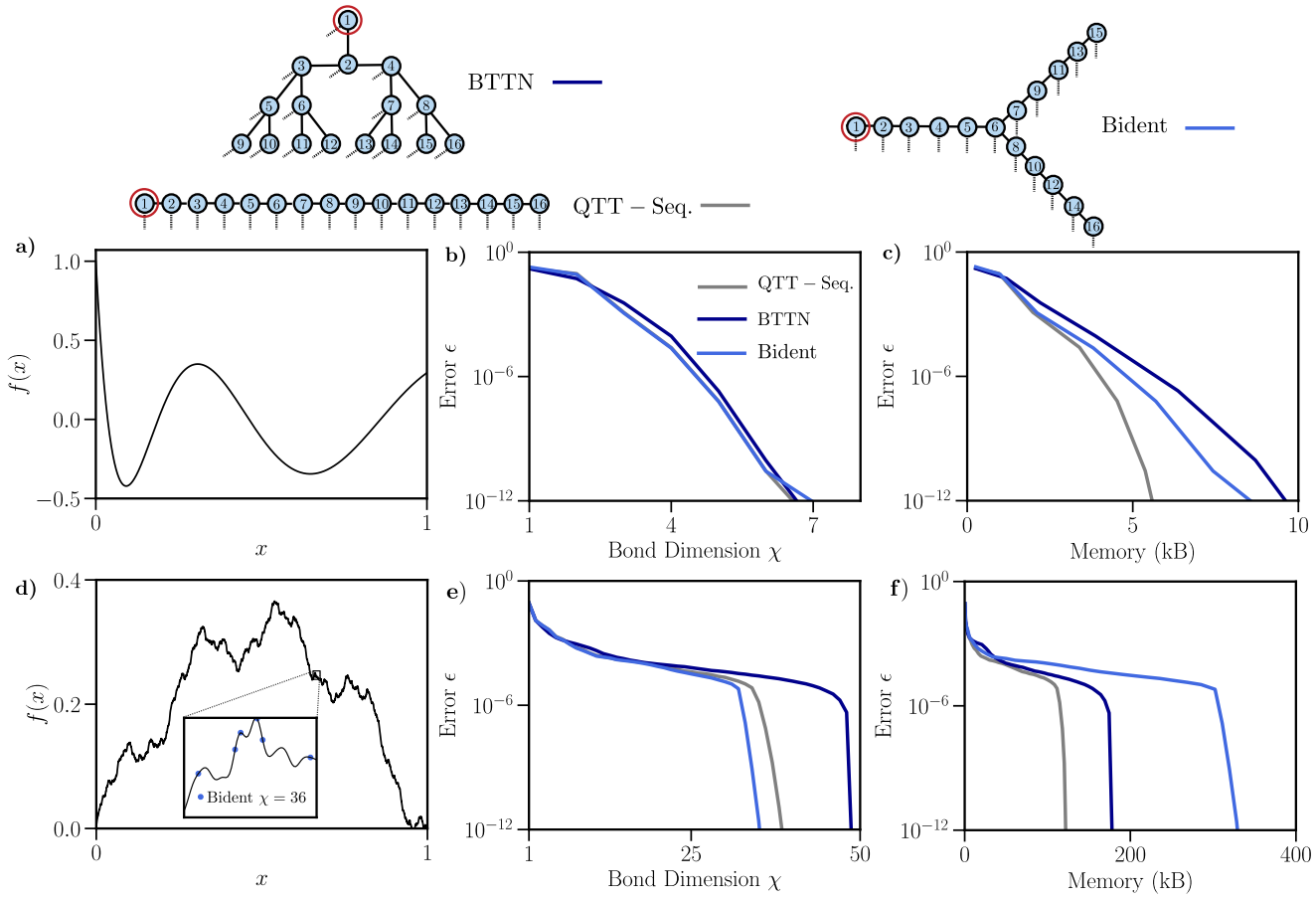


FIG. 3. Comparison of different tree tensor networks, with $L = 16$ vertices, for compressing two different one-dimensional functions $f(x)$ with a binary decomposition $x = 0.x_1x_2...x_{16}$. Error ϵ is calculated via Eq.(6), sampling the function over 10^3 random grid points. The different labelled trees used are shown at the top and the bits are numbered from most significant to least significant: with the most significant digit circled in red. Two functions are considered. Top plots) the Laguerre polynomial $f(x) = L_n(x) = \sum_{k=0}^{n} \binom{n}{k} \frac{(-1)^k}{k!} x^k$ with $n = 40$ and Bottom plots) the Weierstrass function $f(x) = \sum_{k=1}^{n} \frac{\sin(\pi k^a x)}{\pi k^a}$ with $a = 3$ and $n = 25$. Left panels: Sketch of the functions considered over $x \in [0, 1]$. Inset of **d)** shows a zoomed-in region of the function with data points corresponding to the values obtained from the Bident tree tensor network with $\chi = 36$. Middle panels: Error versus bond dimension of the tensor networks. Right panels: Error versus memory requirement for storing the tensor networks —assuming 8 bytes for a floating point number.

any sampling bias. The same subset g is used when comparing the efficacy of different TTNS for the same function $f(\mathbf{x})$.

1D Function Construction - We consider two emblematic single-variable functions: a Laguerre polynomial and a truncated series representation of the Weierstrass function. We compare these functions represented on three different labeled trees in Fig. 3: a one-dimensional path (i.e. a QTT) with the most-to-least significant bits ordered from left to right, a binary tree with the more significant bits nearer the root, and a three-pronged tree with the most significant digit on the end of one of the prongs. We use our direct constructive methods to build an exact tensor network representation of a given target function $f(\mathbf{x})$ on the specified labelled tree and then compare the effect on the error ϵ when systematically truncating down the bond dimension of the network (via an singular value decomposition of local pairs of tensors) from $\chi = \chi_{\max}$ to $\chi = 1$.

For the Laguerre polynomial the function is continuous and smooth. We find, despite the high-order nature of the polynomial, that the function can be represented with an error of $\epsilon \sim 10^{-12}$ with maximum bond dimension $\chi = 7$ on any of the labelled trees. The tensor train with sequential digit ordering $(x_1, x_2, ..., x_n)$ is slightly more effective in terms of error vs required memory to store the tensor network. The Weierstrass function, meanwhile, is a more complex function. Whilst we consider only a truncated version of its infinite trigonometric series, the limit is a nowhere differentiable function. This complexity explains why a much larger bond dimension is required to exactly capture the finite-series realisation of the function ($\chi \sim 40$). Here we find that the tensor train ansatz with sequential

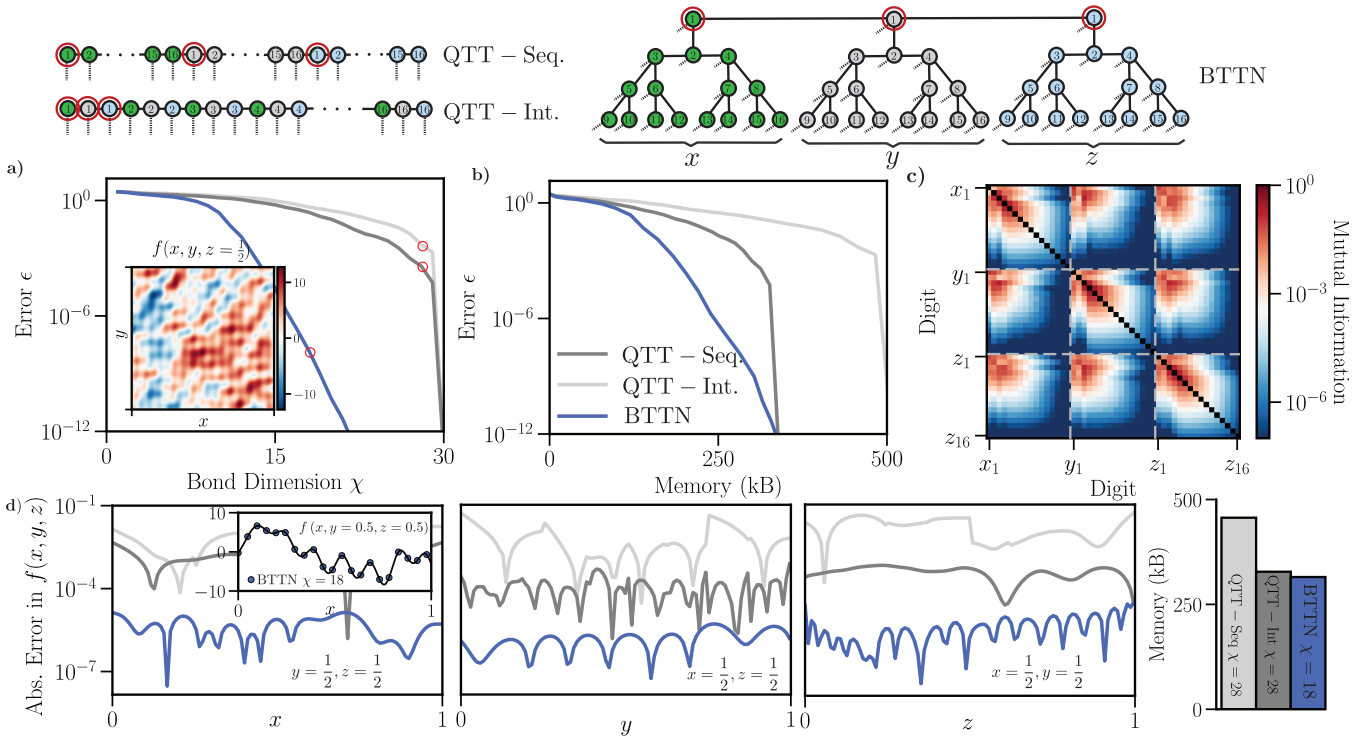


FIG. 4. Comparison of different tree tensor networks for compressing the three dimensional function $f(\mathbf{r}) = \sum_{j=1}^{n} \cos(j\mathbf{k_j} \cdot \mathbf{r})$ with $n = 30$, $\mathbf{r} = (x, y, z)$ and $\mathbf{k_j} = (k_j^x, k_j^y, k_j^z)$ with $k_j^\alpha \in \mathcal{N}(0, 1)$. Error is calculated as ϵ via Eq.(6), sampling the function over 10^3 random grid points. Different trees used are shown at the top. Bits for x, y and z are coloured in green, grey and light blue respectively and numbered sequentially from most significant to least significant. **a)** Error versus bond dimension. Inset shows a heatmap of the function at $z = \frac{1}{2}$. **b)** Error versus memory requirement for storing the tensor networks — assuming 8 bytes for a single floating point number. **c)** Mutual information matrix encoding the correlations between the binary digits. Calculated by sampling the function 10^4 times to build up an approximate reduced density matrix for the given paur of bits. Each dashed block encodes the matrix of correlations between the bits of two given dimensions. **d)** Absolute error over one-dimensional slices of the function. Bar chart shows memory requirements on the right for the specified TTN with the maximum bond dimension of the network also shown. Inset (black line) shows a slice of the function for $y = 1/2$ and $z = 1/2$, with corresponding results for the coupled binary tree tensor network at $\chi = 18$.

digit ordering is noticeably more effective than the other labelled trees. The tensor train corresponds to a network with the lowest co-ordination number whilst still being connected. This directly translates into the lowest memory cost of χ^2 for the tensors in the network. Moreover for typical one-dimensional functions the leading bits, and their correlations with each other, are the most important and so by clustering them all at the start of the train this allows the ansatz to capture those fundamental correlations at a low cost.

3D Function Construction - In Fig. 4 we move on to consider a three-dimensional function which is the sum of $n = 30$ random plane waves of increasing frequency and compare three topologies: two tensor trains with different, commonly used, digit orderings (interleaved and sequential) and a tree consisting of three separate binary trees, one for each dimension, coupled together at their roots (where the most significant bits are placed). Here we find that this coupled binary tree tensor network (BTTN) is a better ansatz for the function at hand and compresses much more effectively under truncation of the internal bonds. For a fixed memory cost it can achieve orders of magnitude lower error than the trains. For instance, it is able to represent the function with an error $\epsilon \sim 10^{-6}$ with a bond dimension of $\chi = 15$ whilst the tensor trains each require a bond dimension of $\chi = 30$ to reach such an error. As $n = 30$ the function can be represented exactly on any network with a bond dimension $\chi = 30$ and thus this shows that the tensor trains are an ineffective representation. In Fig. 4d) we show the absolute error over multiple one-dimensional slices of the function of the tree TTNs for fixed bond dimensions. Despite having a slightly lower memory cost than the tensor trains, the error in the BTTN is consistently several orders of magnitude lower than them.

To support our analysis, we also compute the correlation measure $M(x_A, x_B)$ between two binary digits $x_A = x_{i,j}$ and $x_B = x_{i',j'}$ for a given function $f(\mathbf{x})$ by interpreting the function values as coefficients of a wavefunction and computing the quantum mutual information $M(x_A, x_B)$ [56] by building an approximate representation of the two-

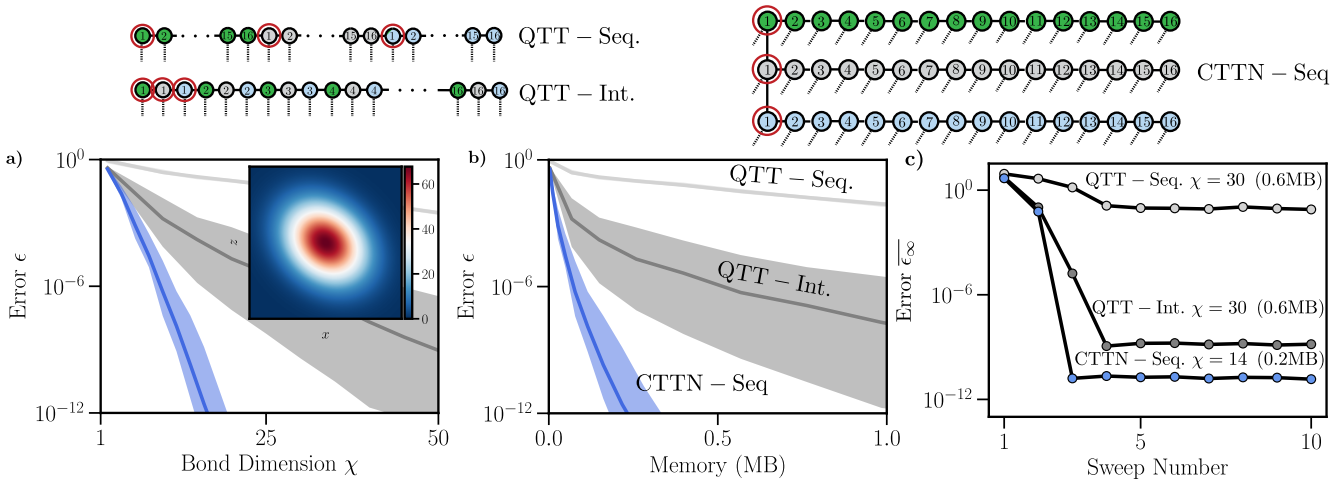


FIG. 5. Comparison of the effectiveness of different tree tensor networks with $L = 16$ bits per dimension for learning the multinormal probability density function $f(\mathbf{r}) \propto \exp(-((\mathbf{r}-\mu)^T M^{-1}(\mathbf{r}-\mu))$ via the tensor cross interpolation algorithm. Here M is an $n \times n$ covariance matrix and $\mu = (\mu_1, \mu_2, ..., \mu_n)$ is the mean vector. We consider $n = 3$ with $\mathbf{r} = (x, y, z) \in [0, 10)^3$ and $\mu = (5, 5, 5)$. Results are obtained from drawing $N = 10$ instances of M $(M_1, M_2, \dots M_{10})$ from the LKJ distribution [57] with shape parameter $\eta = 50$. Different trees used are shown at the top. Bits for x, y and z are coloured in green, grey and light blue respectively. The most significant digit in each dimension is circled in red. **a-b)** Error ϵ, calculated via Eq. (6) after $n = 10$ sweeps of the TCI algorithm, versus bond dimension and memory cost for the tensor networks. The solid lines shows the mode of the error over the 10 realizations of M whilst the shaded area shows the range of the error, i.e. for any M sampled the error ϵ for a given bond dimension. Inset) shows a heatmap of the function over $(x, z) \in [3, 7]^2$ with $y = \frac{1}{2}$. **c)** Average value for the infinity norm ϵ_∞ (see Eq. (6)) over a given sweep of the TCI algorithm for $M = M_1$ and the three tensor networks at the specified bond dimensions and memory cost.

body reduced density matrix $\rho_{A,B}$ via sampling of the function (see Supplementary Material for calculcation details). The value of $M(x_A, x_B)$ is a good proxy for the correlations between the two bits and the larger this value the closer bits x_A and x_B will need to be in the tensor network in order to accurately encode their correlations with a fixed bond dimension. The plot of the mutual information matrix in Fig. 4c show that the most significant binary digits are strongly correlated with the other significant bits within their dimension and with those in other dimensions. Meanwhile, the least significant bits are typically correlated only with bits within their respective dimension. These lead us to understand why the structured tree is far more effective: it keeps the more significant bits in a given dimension clustered together and close to their counterparts in other dimensions. Meanwhile the least significant bits are far from those in other dimensions, which is acceptable because they are very weakly correlated. We emphasize that our results here are not specific to the random frequencies and amplitudes chosen: we observed the same qualitative results for any random realisation of the plane wave frequencies.

TCI Function Construction - In Fig. 5 we use the TCI algorithm to build representations of the trivariate gaussian probability density function $f(\mathbf{r}) \propto \exp(-(\mathbf{r}-\mu)^T M^{-1}(\mathbf{r}-\mu))$ where $\mathbf{r} = (x, y, z)$, $\mu = (\mu_x, \mu_y, \mu_z)$ is the mean vector, and M is a covariance matrix which we sample from the Lewandowski-Kurowicka-Joe (LKJ) distribution with shape parameter $\eta = 50$ [57], which controls the weight of the off-diagonal correlations. We compare results from the TCI algorithm when varying the maximum allowed bond dimension for different tree tensor networks: two tensor trains with commonly used digit orderings (sequential and interleaved) for multi-dimensional functions and a comb tree tensor network consisting of sequentially ordered tensor trains (CTTN - Seq) coupled by their most significant digit.

Similarly to our results for random plane waves we find that for all covariance matrices we sample, the comb tree systematically outperforms the tensor trains. Firstly, for a fixed memory cost it achieves orders of magnitude lower error ϵ than the tensor trains (see Fig. 5b). Moreover, the variance in the error vs memory curves is much lower than for the quantics tensor train with an interleaved ordering, indicating it is a much more effective and consistent ansatz. The variance for the QTT with sequential ordering is also very low, but the errors are drastically worse than the other two ansatzes (we are unable to converge the error to below $\epsilon \sim 10^{-2}$ for any of the covariance matrices sampled). This low variance is therefore just an indicator that the ansatz is consistently poor.

We emphasize that the effectiveness of the comb tree in comparison to the tensor trains is not at all specific to the shape parameter η we chose. In the Supplemental Material we also show results for $\eta = 1$, which is equivalent to sampling uniformly from the space of all covariance matrices. For any given M the structured tree offers a significantly

better ansatz than the tensor trains: with many orders of magnitude lower errors for a given memory cost. Due to the lower value of η, however, the variance (i.e. the size of the shaded area in Fig. 5) in the bond dimension / memory required to accurately represent the function is much higher because certain matrices are sampled which have very significant correlations between the continuous variables. This makes the function harder to represent with a TTN ansatz.

V. Application: Solving Non-linear Fredholm Equations with Tree Tensor Networks

Integral equations arise in many different scientific domains [58]. Analytical solutions of such equations are typically difficult to find and thus numerical methods are vital[59–61]. Here we use the methods introduced in this paper to define an iterative tree tensor network (TTN) based numerical algorithm for Fredholm integral equations of the second kind. The solution is represented as a TTN and following the methods introduced in the paper, there is complete flexibility over the structure of the tree chosen.

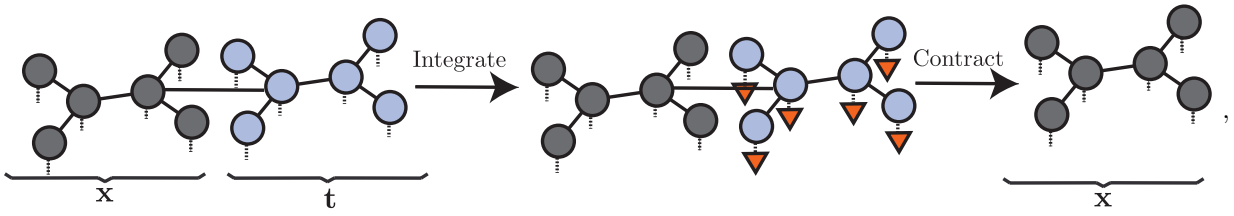
We focus on the following Fredholm equations of the second kind

$$f(\mathbf{x}) = g(\mathbf{x}) + \lambda \int_{\mathbf{t}\in[0,1)^n} K(\mathbf{x},\mathbf{t}) f^{\alpha}(\mathbf{t}) d\mathbf{t} \qquad \alpha \in \mathbb{N}, \tag{7}$$

where $\mathbf{x} = (x_1, x_2, \dots x_n) \in [0,1)^n$ and $\mathbf{t} = (t_1, t_2, \dots t_n) \in [0,1)^n$. The integral kernel is $K(\mathbf{x},\mathbf{t}) : [0,1)^n \times [0,1)^n \to \mathbb{R}$ and $g(\mathbf{x}) : [0,1)^n \to \mathbb{R}$ is a given function. We wish to find the solution $f(\mathbf{x}) : [0,1)^n \to \mathbb{R}$, setting $\lambda = 1$ without loss of generality. In our examples we will focus on the non-linear case ($\alpha > 1$), however our algorithms and analysis also apply straightforwardly to the linear case ($\alpha = 1$) as well.

We consider a tree tensor network defined over a labelled tree $\mathcal{T}_{\mathbf{x}}$ as the ansatz for $f(\mathbf{x})$ and perform the iterative procedure illustrated in Fig. 6) to attempt to solve Eq. (7). The procedure is also given in "pseudo code" in Algorithm 1. The kernel $K(\mathbf{x},\mathbf{t})$ is constructed over the tree $\mathcal{T}_{\mathbf{x}} \cup \mathcal{T}_{\mathbf{t}}$, where $\mathcal{T}_{\mathbf{t}}$ is identical to $\mathcal{T}_{\mathbf{x}}$ except the vertices have been relabelled $x_{(i,j)} \to t_{(i,j)}$. A single edge $e_{\mathbf{x}\leftrightarrow\mathbf{t}}$ is added to $\mathcal{T}_{\mathbf{x}} \cup \mathcal{T}_{\mathbf{t}}$ between one of the binary digits in $\mathbf{x}$ and one in $\mathbf{t}$. The larger the rank of the kernel, the larger the dimension of the virtual index $\alpha_{e_{\mathbf{x}\leftrightarrow\mathbf{t}}}$ corresponding to $e_{\mathbf{x}\leftrightarrow\mathbf{t}}$ will need to be for an accurate representation. One can view such a TTN construction of the kernel as the finite-rank decomposition $K(\mathbf{x},\mathbf{t}) = \sum_{i=1}^{\dim(\alpha_{e_{\mathbf{x}\leftrightarrow\mathbf{t}}})} u^{(i)}(\mathbf{x}) v^{(i)}(\mathbf{t})$ where the function $u^{(i)}(\mathbf{x})$ is represented with a TTN with structure specified by $\mathcal{T}_{\mathbf{x}}$ and $v^{(i)}(\mathbf{t})$ is represented with a TTN with identical structure specified by $\mathcal{T}_{\mathbf{t}}$. Importantly, our generic TCI algorithm provides us with a method to approximately identify such a decomposition.

The partial integration of $f^{\alpha}(\mathbf{t})K(\mathbf{x},\mathbf{t})$ over $\mathbf{t}$ is performed by multiplying the tensors with external indices corresponding to bits in $\mathbf{t}$ with the vector $T_{i,j} = (\frac{1}{2}, \frac{1}{2})$ and then contracting away those tensors. This process can be written diagrammatically as:



Importantly, the bond dimension of the tree tensor network at the end of each iteration is guaranteed to be bounded by $\chi_g + \chi_{K,\mathcal{T}}$ where χ_g is the bond dimension of $g(\mathbf{x})$ and $\chi_{K,\mathcal{T}}$ is the bond dimension of $K(\mathbf{x},\mathbf{t})$ in the subtree $\mathcal{T}_{\mathbf{x}}$. Thus the success of the algorithm relies on finding accurate, low bond dimension representations of $K(\mathbf{x},\mathbf{t})$ and $g(\mathbf{x})$ on the given labelled tree. This is because if an accurate tensor network representation of $K(\mathbf{x},\mathbf{t})$ and $g(\mathbf{x})$ can be found with bond dimensions that do not scale with the size of the network, then the algorithm complexity scales with $\mathcal{O}(L)$ whilst the error on the integration scales as $\mathcal{O}(\exp(-L))$. Thus the complexity of the algorithm is based on the representation of $K(\mathbf{x},\mathbf{t})$ and $g(\mathbf{x})$ as opposed to the integration itself: which can be a limitation of DNS solvers such as that outlined in Ref. [61].

In Fig. 6 we present results from this method for two example non-linear Fredholm equations, with known two-dimensional solutions [61]. In both examples we take $\mathcal{T}_{\mathbf{x}}$ to be a tree formed from a pair (one for each dimension) of binary trees of depth k coupled by their roots (see Fig. 6 for more details). The more significant bits are placed nearer the roots of the tree. The examples we use correspond to

- Example I: $K(x_1, x_2, t_1, t_2) = x_1 x_2^2 t_1/6$, $\alpha = 3$ and $g(x_1, x_2) = \sin(x_1) - c x_1 x_2^2$ with $c = (1 - \cos(1)(\sin^2(1)/2 + 1))/18$. The exact solution is $f(x_1, x_2) = \sin(x_2)$.

Algorithm 1 Tree tensor network method for solving the non-linear Fredholm equation — see Eq. (7)

Given (by using the TTN construction methods specified in Sec. III): Tree tensor network (TTN) representing the initial guess $f_1(\mathbf{x})$ with structure specified by the labelled tree $\mathcal{T}_\mathbf{x}$. TTN representing $g(\mathbf{x})$ with structure given by $\mathcal{T}_\mathbf{x}$. TTN representing $K(\mathbf{x},\mathbf{t})$ with structure represented by $\mathcal{T}_\mathbf{x} \cup \mathcal{T}_\mathbf{t}$. Where $\mathcal{T}_\mathbf{t}$ is a copy of $\mathcal{T}_\mathbf{x}$ but with the external indices mapped: $x_{(i,j)} \to t_{(i,j)}$. There is *one* additional edge between $\mathcal{T}_\mathbf{x}$ and $\mathcal{T}_\mathbf{t}$ in $\mathcal{T}_\mathbf{x} \cup \mathcal{T}_\mathbf{t}$.

for i in $1 : \mathrm{N}$ **do**

Remap Variables: $f_i(\mathbf{t}) \leftarrow f_i(\mathbf{x})$ ▷ The mapping is achieved via the index relabelling $x_{i,j} \to t_{i,j}$.

Multiply (Sec. III A): $f_i(\mathbf{x},\mathbf{t}) \leftarrow K(\mathbf{x},\mathbf{t}) f_i^\alpha(\mathbf{t})$

Integrate (Sec. V): $f_i(\mathbf{x}) \leftarrow f_i(\mathbf{x},\mathbf{t}) \prod_{i=1}^{n} \prod_{j=1}^{L} T_{i,j}(t_{i,j})$ ▷ $T_{i,j}(t_{i,j}) = \frac{1}{2}\ \forall i,j.$

Add (Sec. III A): $f_{i+1}(\mathbf{x}) \leftarrow f_i(\mathbf{x}) + g(\mathbf{x})$

end for

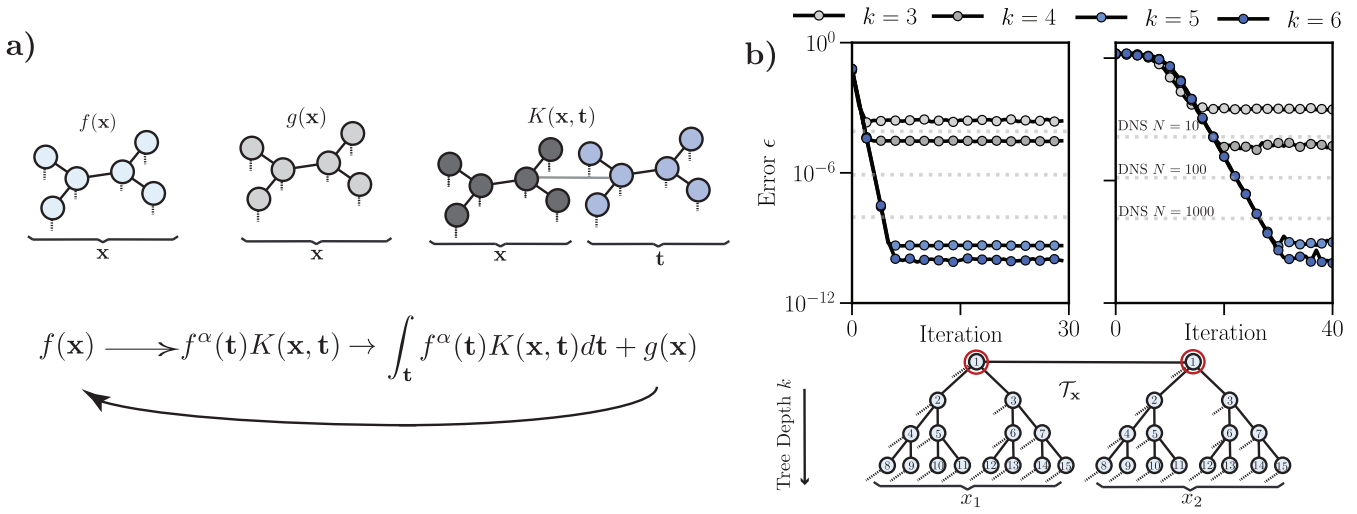


FIG. 6. Tree tensor network method for solving (non-linear) Fredholm equations of the second kind $f(\mathbf{x}) = g(\mathbf{x}) + \int_\mathbf{t} f^\alpha(\mathbf{t}) K(\mathbf{x},\mathbf{t}) d\mathbf{t}$. **a)** An initial guess for the target function $f(\mathbf{x})$ is constructed as a tensor network over a labelled tree. The additive function $g(\mathbf{x})$ is constructed as a tree tensor network of the same structure. The kernel $K(\mathbf{x},\mathbf{t})$ is then constructed over as a tree tensor network formed from two copies of the tree, with one encoding the 'real' variables $\mathbf{x}$ and the other the auxiliary variables $\mathbf{t}$. A single edge is also added between the two trees. The Fredholm equation can then be solved by the iterative steps expressed at the bottom and outlined in Alg. 1 **b**) Results for two different Fredholm equations with a two dimensional solution $f(\mathbf{x}) = f(x_1, x_2)$. Explicit expressions for $K(x_1, x_2, t_1, t_2)$ and $g(x_1, x_2)$ are given in the main text. The structure of the underlying tree for $f(x)$ is shown at the bottom with the binary digits numbered by their significance: the most significant digit in each dimension is circled in red. Error ϵ is calculated as in Eq. (6), sampling over 100 random grid points and comparing to the values for the exact solution, versus iteration of the solver. Dashed lines show the converged error from the Direct Numerical Simulation (DNS) results of Ref. [61] with $N = 10, 100$ and 1000 grid points. Left plot) Results for example 1 where the exact solution is $f(x_1, x_2) = \sin(x_2)$. Right plot) Results for example 2 where the exact solution is $f(x_1, x_2) = 1/(1 + x_1 + x_2)^2$.

- Example II: $K(x_1, x_2, t_1, t_2) = \frac{x_1(1+t_1+t_2)}{1+x_2}$, $\alpha = 2$ and $g(x_1, x_2) = 1/(1 + x_1 + x_2)^2 - x_1/(1 + x_2)/6$. The exact solution is $f(x_1, x_2) = 1/(1 + x_1 + x_2)^2$.

In the first example all of the relevant functions can be constructed exactly using our direct construction method for polynomials. In the second we are able to use TCI to construct g and K on the trees $\mathcal{T}_\mathbf{x}$ and $\mathcal{T}_\mathbf{x} \cup \mathcal{T}_\mathbf{t}$ with errors on the order of the grid spacing $\epsilon \sim O(2^{-L})$ whilst only using a bond dimension $\chi \leq 10$. The structure of $\mathcal{T}_\mathbf{x}$ is shown in Fig. 5.

We show the results of the solver in Fig. 6b starting from a tensor network of bond dimension $\chi = 1$ representing the constant function $f_1(x_1, x_2) = 1$. In each example, we observe convergence of our solution to a given error ϵ (in comparison to the exact solution) which is controlled by the number of bits L which we take to decompose each continuous variable. Notably, the errors we achieve are always on the order of the grid spacing $\epsilon \sim O(2^{-L})$, which is the error in our representation of K and g and our integration technique. For sufficient L this exponential precision allows us to reach a much higher accuracy than the quadrature-based DNS methods benchmarked in Ref. [61].

VI. Conclusion

In this paper we have introduced the tree tensor network (TTN) ansatz for representing functions and solving problems in continuous space, generalising beyond the almost-exclusively used one-dimensional tensor train ansatz. We provided direct and indirect (via an extension of the tensor cross interpolation algorithm) methods for constructing tree tensor network (TTN) representations of mathematical functions. We identified a direct construction of polynomial functions, with an upper bound on the maximum bond dimension in the tree which is independent of the network topology. For multi-dimensional functions we find that TTNs with more complex structure — such as comb tree tensor network (CTTN) and coupled binary tree tensor networks (BTTN) — can be a much more effective ansatz than the (quantics) tensor train. This is because these more structured TTNs can simultaneously keep the leading binary digits close to their counterparts within the same dimension and in other dimensions. Using the tools introduced in this paper we introduced a new iterative TTN-based solver for non-linear Fredholm integral equations. For our algorithm, provided the iterative solver converges, the bond dimension of the solution is boundable in terms of the bond dimension of the kernel.

Looking forward, an important outstanding question is identifying a relevant cost function, and a search algorithm for finding the tree corresponding to its extrema, which allows one to identify good candidates for the correct tree structure for representing a given function. Potential heuristics could include those based on the quantum mutual information which have had success in determining the optimal ordering for sites in matrix product states when applied to quantum chemistry problems [62–64].

Finally, we wish to emphasize that the generality of the work described here means that a wide range of TTN-based numerical algorithms can be built, significantly broadening the scope and potential of tensor-network based numerical methods. For problems where the solution is a multivariate function with significant inter-dimensional correlations, our results suggest moving away from the tensor train ansatz and working with more structured tree tensor networks could push the state-of-the-art for problem solving. Such functions, for instance, arise in turbulent solutions of the Navier-Stokes equation and are currently pushing the limits of the tensor train ansatz [65, 66].

Acknowledgments

We would like to thank Matthew Fishman for helpful discussions and advice and input on software design. We also thank Ryan Anselm and Marc Ritter for useful discussions. The authors are grateful for ongoing support through the Flatiron Institute, a division of the Simons Foundation. The tools and methods described in this paper are contained within the ITensorNumericalAnalysis.jl library [67]: an open-source Julia package built on-top of ITensors.jl [68] and *TensorNetworkQuantumSimulator.jl* [69] for constructing and optimizing tree tensor network representations of continuous functions, with the tree structure freely definable by the user.

VII. Supplementary Material

A. Proof the TTN construction in Sec III contracts to the polynomial $f(x) = \sum_{k=0}^{d} c_k x^k$.

Here we prove that for a tree tensor network whose local tensors have elements specified by Eq. (2) will contract down to the polynomial $f(x) = \sum_{k=0}^{d} c_k x^k$. First, we observe that the elements of the local tensors satisfy the following properties,

$$\sum_{\alpha_{z_j-1}=0}^{d} (x_{j'})^{\alpha_{z_j-1}} \mathcal{T}_j(x_j)_{\alpha_1,\alpha_2,\ldots,\alpha_{z_j-1},\beta} = C_{\alpha_1,\alpha_2,\ldots,\alpha_{z_j-2},\beta} \left(\frac{x_j}{2^j} + \frac{x_{j'}}{2^{j'}}\right)^{f_{\alpha_1,\alpha_2,\ldots,\alpha_{z_j-2},\beta}} \tag{S1}$$

$$= \quad \mathcal{T}_{(j,j')}(x_j + x_{j'})_{\alpha_1,\alpha_2,\ldots,\alpha_{z_j-2},\beta} \qquad z_j > 2,$$

$$\sum_{\alpha_1=0}^{d} (x_{j'})^{\alpha_1} \mathcal{T}_j(x_j)_{\alpha_1,\beta} = (x_j + x_{j'})^{\beta} = \mathcal{T}_{(j,j')}(x_j + x_{j'})_{\beta} \qquad z_j = 2. \tag{S2}$$

This means that the contraction $\mathcal{T}_j \cdot \mathcal{T}_{j'}$ of one of the tensors on the leaves of the tree $\mathcal{T}_j$ and its parent $\mathcal{T}_{j'}$ yields a new tensor $\mathcal{T}_{(j,j')}$ (with two external indices corresponding to x_j and $x_{j'}$ and $z_{j,j'} = z_j + z_{j'} - 2 = z_{j'} - 1$ virtual indices corresponding to the set difference of the virtual indices of $\mathcal{T}_j$ and $\mathcal{T}_{j'}$) whose properties are still completely specified by Eq. (2).

We can utilize this property to iteratively prove the contraction of the tree yields $f(x)$. Following Eq. (S2) the binary digits effectively sum under contraction of the tensors on the leaves of the tree with their parents. This means we can repeat the process of contracting the leaves of the tree onto their parents until we are left with the root tensor $\tilde{\mathcal{T}}_r$ surrounded by z_r tensors each with a single virtual index β whose elements are specified by $(z_k)^{\beta} = (\sum_{x_j \in \text{Branch}(\beta)} x_j)^{\beta}$ where the sum runs over all of the bits x_j which the branch specified by the virtual index β connects to x_r. The following property holds for the elements of $\tilde{\mathcal{T}}_r$

$$\sum_{\alpha_1=0}^{d} (z_1)^{\alpha_1} \sum_{\alpha_2=0}^{d} (z_2)^{\alpha_2} \ldots \sum_{\alpha_r=0}^{d} (z_r)^{\alpha_{z_r}} \tilde{\mathcal{T}}_r(x_r)_{\alpha_1,\alpha_2,\ldots,\alpha_{z_r}} = \sum_{k=0}^{d} c_k (z_1 + z_2 + z_3 + \ldots z_r + x_r)^k \qquad z_i \in \mathbb{C} \tag{S3}$$

and so we can contract these satellite tensors onto $\tilde{\mathcal{T}}_r(x_r)$ and arrive at $f(x) = \sum_{k=0}^{d} c_k \left(\sum_{i=1}^{L} \frac{x_j}{2^j}\right)^k$. The proof is complete. *Elevating to higher dimensions* - We can also represent polynomials as tree tensor networks in the case there are multiple continuous variables: i.e. we wish to build a TTN representation of $f(\mathbf{x}) = f(x_1, x_2, \ldots x_n) = f(x) = \sum_{k=0}^{d} c_k x^k$, where $x \in \{x_1, x_2, \ldots x_n\}$. The construction is achieved as before, with the tensors $\mathcal{T}_{(j)}$ with an external index corresponding to x, defined via Eq. (2). For the tensors $\mathcal{T}_{(i,j)}$ where $x_i \neq x$ their elements are given as

$$\mathcal{T}_{(i,j)}(x_{i,j})_{\alpha_1,\alpha_2,\ldots,\alpha_{z_{i,j}-1},\beta} = \begin{cases} 1 & \sum_{l=1}^{z_{i,j}-1} \alpha_l = \beta \\ 0 & \text{otherwise} \end{cases} \qquad z_{i,j} > 1$$

$$\mathcal{T}_{(i,j)}(x_{i,j})_{\beta} = 1 \qquad z_{i,j} = 1 \tag{S4}$$

which will guarantee the tree tensor network contracts to $f(x)$, with the virtual indices all of dimension $\chi = d + 1$.

B. Mutual Information Measurement

In Fig. 4 we plot the "quantum mutual information" between two binary digits $x_A = x_{i,j}$ and $x_B = x_{i',j'}$ for the given function $f(\mathbf{x})$. To compute this quantity we treat the binary digits as spin degrees of freedom and the amplitudes of the function as analogous to the amplitudes of a quantum wavefunction $\psi(\mathbf{x}) \leftrightarrow f(\mathbf{x})$. We then form the two-body reduced density matrix $\rho_{A,B} = \rho((x_A, x_B), (x_A, x_B)')$ by taking the partial trace over all bits excluding x_A and x_B of the full density matrix whose elements are specified as $\rho(\mathbf{x}, \mathbf{x}') = f(\mathbf{x}) f^*(\mathbf{x}')$. We have used the notation $\mathbf{x} = (x_{1,1}, x_{1,2}, \ldots x_{1,L}, x_{2,1}, x_{2,2}, \ldots x_{n,L})$ and $\mathbf{x}' = (x'_{1,1}, x'_{1,2}, \ldots x'_{1,L}, x'_{2,1}, x'_{2,2}, \ldots x'_{n,L})$.

In practice, due to the exponentially large size of the grid in the tensor network size L, we form an approximate reduced density matrix by performing the partial trace via sampling of the function over N randomly selected grid points. For the example plotted in Fig. 4 we find good convegrence and 10^4 grid points is sufficient to get an accurate approximation of the two-body reduced density matrix for a given pair of binary digits.

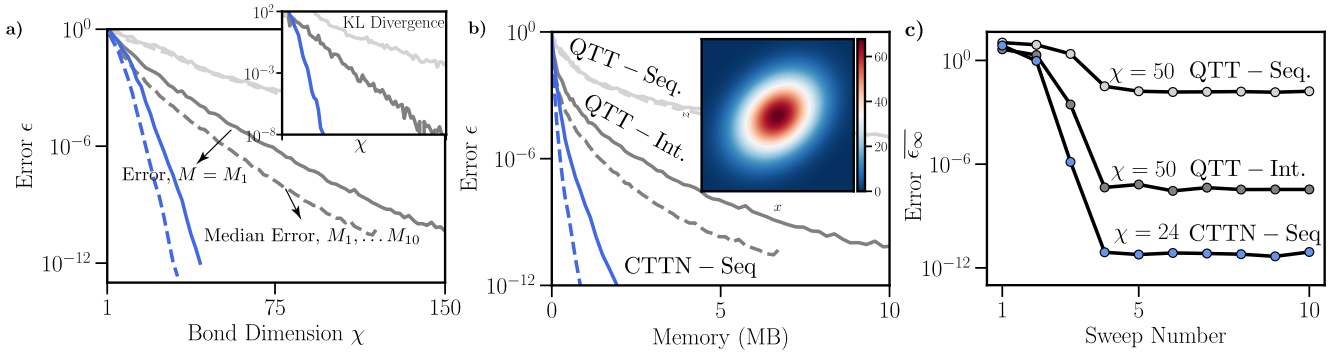


Figure S1. Comparison of the effectiveness of different tree tensor networks (diagrams of the different networks can be found in Fig. 5) with $L = 16$ bits per dimension for learning the multinormal probability density function $f(\mathbf{r}) \propto \exp(-((\mathbf{r}-\mu)^T M^{-1}(\mathbf{r}-\mu))$ via the tensor cross interpolation algorithm. Here M is an $n \times n$ covariance matrix and $\mu = (\mu_1, \mu_2, ..., \mu_n)$ is the mean vector. We consider $n = 3$ with $\mathbf{r} = (x, y, z) \in [0, 10)^3$ and $\mu = (5, 5, 5)$. Results are obtained from drawing $N = 10$ instances of M $(M_1, M_2, \dots M_{10})$ from the LKJ distribution [57] with shape parameter $\eta = 1$, a distribution uniform in the space of all covariance matrices. **a-b)** Error ϵ, calculated via Eq. (6) after $n = 10$ sweeps of the TCI algorithm, versus bond dimension and memory cost for the tensor networks. The dashed lines shows the mode of the error over the 10 realizations of M whilst the solid line shows the specific error for $M = M_1$. Inset in **a)** shows the Kullback-Leibler divergence (calculated from the function samples) as a function of bond dimension evaluated from the function samples for $M = M_1$. Inset in **b)** shows a heatmap of the function for $y = \frac{1}{2}$. **c)** Average value for the infinity norm ϵ_∞ — see Eq. (6) — over a given sweep of the TCI algorithm for $M = M_1$ and the three tensor networks at the specified bond dimensions.

From this approximate matrix we can form the one-body reduced density matrices $\rho_A = \mathrm{Tr}_B \rho_{A,B}$ and $\rho_B = \mathrm{Tr}_A \rho_{A,B}$, allowing the mutual information $M(x_A, x_B)$ between the two bits to be be calculated via

$$\begin{aligned} M(x_A, x_B) &= S(x_A) + S(x_B) - S(x_A \cup x_B) \\ &= -\mathrm{Tr}\left(\rho_A \ln(\rho_A)\right) - \mathrm{Tr}\left(\rho_B \ln(\rho_B)\right) + \mathrm{Tr}\left(\rho_{A,B} \ln(\rho_{A,B})\right), \end{aligned} \tag{S5}$$

where $S(\cdot)$ denotes the standard Von-Neumann entropy.

C. Further TCI Data

In Fig. S1 we present further data comparing the effectiveness of the tensor cross interpolation (TCI) algorithm for learning the trivariate probability density function $f(\mathbf{r}) \propto \exp(-(\mathbf{r}-\mu)^T M^{-1}(\mathbf{r}-\mu))$ where $\mathbf{r} = (x, y, z)$, $\mu = (\mu_x, \mu_y, \mu_z)$ is the mean vector, and M is a covariance matrix. Here we sample M from the Lewandowski-Kurowicka-Joe (LKJ) [57] distribution with shape parameter $\eta = 1$, which is equivalent to sampling uniformly from the space of all covariance matrices. As in the main text, we find the comb tree systematically outperforms the tensor trains for a given covariance matrix M. This is most transparent in Fig. S1c), where the TCI algorithm converges to a value for the infinity norm ϵ_∞ that is orders of magnitude lower for the comb tree than for the tensor trains - despite the train ansatzes utilizing a higher bond dimension and memory. Due to the lower value of the shape parameter η we find the bond dimensions required to reach a certain error, however, vary much much more strongly with different covariances matrices than in Fig. 5 where we set $\eta = 50$. This is because, for $\eta = 1$, matrices are drawn with significant inter-dimensional correlations. These are in general more difficult to represent with a tensor network ansatz due to the presence of significant inter-dimensional correlations between binary digits.

[1] F. Verstraete, V. Murg, and J. Cirac, Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems, Advances in Physics **57**, 143 (2008).
[2] U. Schollwöck, The density-matrix renormalization group in the age of matrix product states, Annals of Physics **326**, 96 (2011), january 2011 Special Issue.
[3] R. Orús, A practical introduction to tensor networks: Matrix product states and projected entangled pair states, Annals of Physics **349**, 117 (2014), arXiv:1306.2164 [cond-mat, physics:hep-lat, physics:hep-th, physics:quant-ph].
[4] A. Cichocki, Tensor Networks for Big Data Analytics and Large-Scale Optimization Problems (2014), arXiv:1407.3124 [cs, math] type: article.

[5] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, and D. P. Mandic, Tensor Networks for Dimensionality Reduction and Large-scale Optimization: Part 1 Low-Rank Tensor Decompositions, Foundations and Trends® in Machine Learning **9**, 249 (2016).

[6] J. C. Bridgeman and C. T. Chubb, Hand-waving and interpretive dance: an introductory course on tensor networks, Journal of Physics A: Mathematical and Theoretical **50**, 223001 (2017).

[7] J. Haegeman and F. Verstraete, Diagonalizing Transfer Matrices and Matrix Product Operators: A Medley of Exact and Computational Methods, Annual Review of Condensed Matter Physics **8**, 355 (2017).

[8] A. Cichocki, A.-H. Phan, Q. Zhao, N. Lee, I. Oseledets, M. Sugiyama, and D. P. Mandic, Tensor Networks for Dimensionality Reduction and Large-scale Optimization: Part 2 Applications and Future Perspectives, Foundations and Trends® in Machine Learning **9**, 431 (2017).

[9] R. Orús, Tensor networks for complex quantum systems, Nature Reviews Physics **1**, 538 (2019), arXiv:1812.04011 [cond-mat, physics:hep-lat, physics:quant-ph].

[10] L. Vanderstraeten, J. Haegeman, and F. Verstraete, Tangent-space methods for uniform matrix product states, SciPost Physics Lecture Notes , 007 (2019).

[11] J. I. Cirac, D. Perez-Garcia, N. Schuch, and F. Verstraete, Matrix product states and projected entangled pair states: Concepts, symmetries, theorems, Rev. Mod. Phys. **93**, 045003 (2021).

[12] The Tensor Network: Resources for tensor network algorithms, theory, and software (2024).

[13] G. Evenbly, Tensors.net: Resources for learning and implementing tensor network methods to study quantum many-body systems. (2024).

[14] J. Tindall and M. Fishman, Gauging tensor networks with belief propagation, SciPost Phys. **15**, 222 (2023).

[15] S. R. White, Density matrix formulation for quantum renormalization groups, Physical Review Letters **69**, 2863 (1992).

[16] S. R. White, Density-matrix algorithms for quantum renormalization groups, Physical Review B **48**, 10345 (1993).

[17] G. Vidal, Efficient Classical Simulation of Slightly Entangled Quantum Computations, Physical Review Letters **91**, 147902 (2003).

[18] W. Merbis, C. de Mulatier, and P. Corboz, Efficient simulations of epidemic models with tensor networks: Application to the one-dimensional susceptible-infected-susceptible model, Phys. Rev. E **108**, 024303 (2023).

[19] S. Dolgov and D. Savostyanov, Tensor product approach to modelling epidemics on networks, Applied Mathematics and Computation **460**, 128290 (2024).

[20] I. V. Oseledets, Approximation of $2^d \times 2^d$ matrices using tensor decomposition, SIAM Journal on Matrix Analysis and Applications **31**, 2130 (2010), https://doi.org/10.1137/090757861.

[21] B. N. Khoromskij, O(dlogn)-quantics approximation of n-d tensors in high-dimensional numerical modeling, Constructive Approximation **34**, 257–280 (2011).

[22] S. V. Dolgov, B. N. Khoromskij, and I. V. Oseledets, Fast Solution of Parabolic Problems in the Tensor Train/Quantized Tensor Train Format with Initial Application to the Fokker–Planck Equation, SIAM Journal on Scientific Computing **34**, A3016 (2012).

[23] B. N. Khoromskij, Tensor Numerical Methods for High-dimensional PDEs: Basic Theory and Initial Applications (2014), arXiv:1408.4053 [math] type: article.

[24] M. Lubasch, P. Moinier, and D. Jaksch, Multigrid renormalization, Journal of Computational Physics **372**, 587 (2018).

[25] J. J. García-Ripoll, Quantum-inspired algorithms for multivariate analysis: from interpolation to partial differential equations, Quantum **5**, 431 (2021).

[26] L. Richter, L. Sallandt, and N. Nüsken, Solving high-dimensional parabolic PDEs using the tensor train format (PMLR, 2021) pp. 8998–9009.

[27] N. Gourianov, M. Lubasch, S. Dolgov, Q. Y. van den Berg, H. Babaee, P. Givi, M. Kiffner, and D. Jaksch, A quantum-inspired approach to exploit turbulence structures, Nature Computational Science **2**, 30 (2022).

[28] N. Gourianov, *Exploiting the structure of turbulence with tensor networks*, Ph.D. thesis, University of Oxford (2022).

[29] Y. N. Fernández, M. K. Ritter, M. Jeannin, J.-W. Li, T. Kloss, T. Louvet, S. Terasaki, O. Parcollet, J. von Delft, H. Shinaoka, and X. Waintal, Learning tensor networks with tensor cross interpolation: new algorithms and libraries (2024), arXiv:2407.02454 [physics.comp-ph].

[30] M. K. Ritter, Y. Núñez Fernández, M. Wallerberger, J. von Delft, H. Shinaoka, and X. Waintal, Quantics tensor cross interpolation for high-resolution parsimonious representations of multivariate functions, Phys. Rev. Lett. **132**, 056501 (2024).

[31] D. Adak, D. P. Truong, G. Manzini, K. Ø. Rasmussen, and B. S. Alexandrov, Tensor network space-time spectral collocation method for time-dependent convection-diffusion-reaction equations, Mathematics **12**, 10.3390/math12192988 (2024).

[32] M. Lindsey, Multiscale interpolative construction of quantized tensor trains (2024), arXiv:2311.12554 [math.NA].

[33] J. Tindall, M. Fishman, E. M. Stoudenmire, and D. Sels, Efficient tensor network simulation of ibm's eagle kicked ising experiment, PRX Quantum **5**, 010308 (2024).

[34] L. Pavešić, D. Jaschke, and S. Montangero, Constrained dynamics and confinement in the two-dimensional quantum ising model (2024), arXiv:2406.11979 [quant-ph].

[35] J. Tindall and D. Sels, Confinement in the Transverse Field Ising model on the Heavy Hex lattice (2024), arXiv:2402.01558 [quant-ph].

[36] A. Kshetrimayum, H. Weimer, and R. Orús, A simple tensor network algorithm for two-dimensional steady states, Nature Communications **8**, 1291 (2017).

[37] J. Tindall, A. Mello, M. Fishman, M. Stoudenmire, and D. Sels, Dynamics of disordered quantum systems with two- and three-dimensional tensor networks (2025), arXiv:2503.05693 [quant-ph].

[38] T. Begušić, J. Gray, and G. K.-L. Chan, Fast and converged classical simulations of evidence for the utility of quantum computing before fault tolerance, Science Advances **10**, eadk4321 (2024), https://www.science.org/doi/pdf/10.1126/sciadv.adk4321.

[39] S. Dolgov and B. Khoromskij, Two-level qtt-tucker format for optimized tensor calculus, SIAM Journal on Matrix Analysis and Applications **34**, 593 (2013).

[40] E. Ye and N. Loureiro, Quantized tensor networks for solving the vlasov-maxwell equations (2024), arXiv:2311.07756 [physics.comp-ph].

[41] A. Gorodetsky, S. Karaman, and Y. Marzouk, A continuous analogue of the tensor-train decomposition, Computer Methods in Applied Mechanics and Engineering **347**, 59 (2019).

[42] M. B. Soley, P. Bergold, A. A. Gorodetsky, and V. S. Batista, Functional tensor-train chebyshev method for multidimensional quantum dynamics simulations, Journal of Chemical Theory and Computation **18**, 25 (2022).

[43] J. Ballani, L. Grasedyck, and M. Kluge, Black box approximation of tensors in hierarchical tucker format, Linear Algebra and its Applications **438**, 639 (2013), tensors and Multilinear Algebra.

[44] L. Grasedyck, *Polynomial Approximation in Hierarchical Tucker Format by Vector-Tensorization*, Preprint 308 (Institut für Geometrie und Praktische Mathematik, RWTH Aachen, 2010).

[45] I. Oseledets and E. Tyrtyshnikov, Tt-cross approximation for multidimensional arrays, Linear Algebra and its Applications **432**, 70 (2010).

[46] D. Savostyanov and I. Oseledets, Fast adaptive interpolation of multi-dimensional arrays in tensor train format, in *The 2011 International Workshop on Multidimensional (nD) Systems* (2011) pp. 1–8.

[47] In this work, for simplicity, we assume binary strings which are all of length L and thus the total number of external indices is nL. The generalization of our work to $n-$ary decompositions with a variable number of binary digits for each variable is straightforward.

[48] L. Grasedyck, Hierarchical Singular Value Decomposition of Tensors, SIAM Journal on Matrix Analysis and Applications **31**, 2029 (2010).

[49] H. Shinaoka, M. Wallerberger, Y. Murakami, K. Nogaki, R. Sakurai, P. Werner, and A. Kauch, Multiscale space-time ansatz for correlation functions of quantum systems based on quantics tensor trains, Phys. Rev. X **13**, 021015 (2023).

[50] J. J. Rodríguez-Aldavero, P. García-Molina, L. Tagliacozzo, and J. J. García-Ripoll, Chebyshev approximation and composition of functions in matrix product states for quantum-inspired numerical analysis (2024), arXiv:2407.09609 [quant-ph].

[51] D. V. Savostyanov, Quasioptimality of maximum-volume cross interpolation of tensors, Linear Algebra and its Applications **458**, 217 (2014).

[52] S. Dolgov and D. Savostyanov, Parallel cross interpolation for high-precision calculation of high-dimensional integrals, Computer Physics Communications **246**, 106869 (2020).

[53] Y. Núñez Fernández, M. Jeannin, P. T. Dumitrescu, T. Kloss, J. Kaye, O. Parcollet, and X. Waintal, Learning feynman diagrams with tensor trains, Physical Review X **12**, 041018 (2022).

[54] Y. N. Fernández, M. K. Ritter, M. Jeannin, J.-W. Li, T. Kloss, T. Louvet, S. Terasaki, O. Parcollet, J. von Delft, H. Shinaoka, *et al.*, Learning tensor networks with tensor cross interpolation: new algorithms and libraries, arXiv preprint arXiv:2407.02454 (2024).

[55] M. Fornace and M. Lindsey, Column and row subset selection using nuclear scores: algorithms and theory for nyström approximation, cur decomposition, and graph laplacian reduction, arXiv preprint arXiv:2407.01698 (2024).

[56] M. A. Valdez, D. Jaschke, D. L. Vargas, and L. D. Carr, Quantifying complexity in quantum phase transitions via mutual information complex networks, Phys. Rev. Lett. **119**, 225301 (2017).

[57] D. Lewandowski, D. Kurowicka, and H. Joe, Generating random correlation matrices based on vines and extended onion method, Journal of Multivariate Analysis **100**, 1989 (2009).

[58] A. J. Jerri, *Introduction to integral equations with applications* (John Wiley & Sons, 1999).

[59] H. Guoqiang and W. Jiong, Extrapolation of nystrom solution for two dimensional nonlinear fredholm integral equations, Journal of Computational and Applied Mathematics **134**, 259 (2001).

[60] A. H. Borzabadi and O. S. Fard, A numerical scheme for a class of nonlinear fredholm integral equations of the second kind, Journal of Computational and Applied Mathematics **232**, 449 (2009).

[61] M. Kazemi, H. M. Golshan, R. Ezzati, and M. Sadatrasoul, New approach to solve two-dimensional fredholm integral equations, Journal of Computational and Applied Mathematics **354**, 66–79 (2019).

[62] J. Rissler, R. M. Noack, and S. R. White, Measuring orbital interaction using quantum information theory, Chemical Physics **323**, 519 (2006).

[63] G. Barcza, O. Legeza, K. H. Marti, and M. Reiher, Quantum-information analysis of electronic states of different molecular structures, Phys. Rev. A **83**, 012508 (2011).

[64] M. Ali, On the ordering of sites in the density matrix renormalization group using quantum mutual information (2021).

[65] M. Kiffner and D. Jaksch, Tensor network reduced order models for wall-bounded flows, Phys. Rev. Fluids **8**, 124101 (2023).

[66] N. Gourianov, P. Givi, D. Jaksch, and S. B. Pope, Tensor networks enable the calculation of turbulence probability distributions (2024), arXiv:2407.09169 [physics.flu-dyn].

[67] ITensorNumericalAnalysis.jl, `https://github.com/jtindall/ITensorNumericalAnalysis.jl` (2024).

[68] M. Fishman, S. R. White, and E. M. Stoudenmire, Codebase release 0.3 for ITensor, SciPost Phys. Codebases , 4 (2022).

[69] M. S. Rudolph and J. Tindall, Simulating and sampling from quantum circuits with 2d tensor networks (2025), arXiv:2507.11424 [quant-ph].