arXiv:2504.06475v2 [quant-ph] 11 Feb 2026

Successive randomized compression: A randomized algorithm for the compressed MPO-MPS product

Chris Camaño, Ethan N. Epperly, and Joel A. Tropp

Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA, 91125, USA

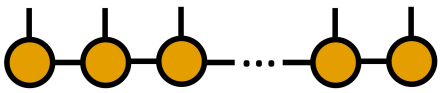
Tensor networks like matrix product states (MPSs) and matrix product operators (MPOs) are powerful tools for representing exponentially large states and operators, with applications in quantum many-body physics, machine learning, numerical analysis, and other areas. In these applications, computing a compressed representation of the MPO–MPS product is a fundamental computational primitive. For this operation, this paper introduces a new single-pass, randomized algorithm, called successive randomized compression (SRC), that improves on existing approaches in speed or in accuracy. The performance of the new algorithm is evaluated on synthetic problems and unitary time evolution problems for quantum spin systems.

1 Introduction

In the past three decades, tensor network methods [1–6] have established themselves as some of the most effective techniques for classical simulation of strongly interacting quantum many-body systems [7–9]. Tensor networks have also shown promise for problems in computational mathematics and machine learning, for instance in numerical analysis [10–12], generative modeling [13,14], and compression of neural networks [15–19]. Given this diverse array of applications, the development of robust, accurate, and scalable tensor network algorithms is of wide interest.

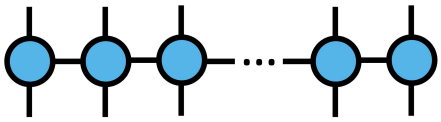
This paper is concerned with matrix product states and matrix product operators, two of the most widely used tensor network formats:

- **Matrix product states (MPSs).** Throughout this article, we will assume basic familiarity with tensor diagram notation; see [6,8,20] for a refresher if needed. Using this notation, an MPS is defined to be a tensor network of the form



  MPSs are used to represent *states* of one-dimensional quantum systems such as chains of interacting spin-S particles. In other applications, they are used to represent *vectors* or *functions*. MPSs are also called *tensor trains* (TTs), particularly in the applied mathematics literature.

- **Matrix product operators (MPOs).** An MPO is a tensor network of the form

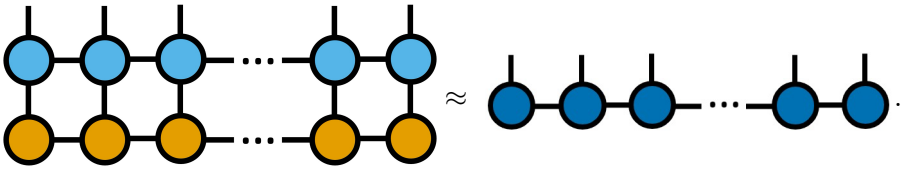


MPOs are used to represent *operators* acting on one-dimensional quantum systems, such as local Hamiltonians [2, 21, 22], time evolution [2, 23, 24], or computational primitives like the quantum Fourier transform [25, 26]. In other applications, MPOs are used to represent *matrices* or *operators on functions*.

In an MPS or MPO, the horizontal and vertical edges are referred to as *bond indices* and *physical indices*, respectively. We denote the number of sites n, the physical dimension d, the MPS bond dimension χ, and the MPO bond dimension D.

This paper is concerned with the following basic and widely applicable problem:

Compressed MPO–MPS multiplication problem. Given an MPS $|\psi\rangle$ and an MPO H, compute an MPS representation $|\eta\rangle \approx H|\psi\rangle$ of the MPO–MPS product:



(1.1)

The MPO–MPS product is a basic primitive which comes up in many computational problems. In quantum physics, MPO–MPS products are used in algorithms for simulating real and imaginary time evolution of 1D quantum states [23, 27], the "boundary MPS" method for contracting tensor networks on higher-dimensional PEPS tensor networks [28, 29], among other use cases [30].

Given the importance of the compressed MPO–MPS multiplication problem, many algorithms have been designed for the task; see section 4 for a description of existing methods. A comparison of these methods appears in Figure 1, which shows the relative error $\|H|\psi\rangle - |\eta\rangle\|_{\mathrm{F}}/\|H|\psi\rangle\|_{\mathrm{F}}$ versus the runtime of several methods for computing the product of a random MPO and a random MPS. The Frobenius norm $\|\cdot\|_{\mathrm{F}}$ is defined as the square root of the sum of squared entries in any tensor. We set both bond dimensions to $D = \chi = 50$, use $n = 100$ sites, and consider a physical dimension $d = 2$. Each tensor is populated with uniformly random entries in the range $[\alpha, 1]$ for $\alpha = -0.5$. The data for this problem is real, but we store it using complex data types (which is the required data format for most quantum tensor network problems). The parameter $\alpha \in [-1, 1]$ sets the difficulty of the problem, with higher α making the problem easier [31–33]; we choose $\alpha = -0.5$ for an intermediate level of difficulty. This is a non-physical example purely intended to demonstrate the performance and accuracy characteristics of the existing methods.

Figure 1 demonstrates limitations of the existing MPO–MPS algorithms. The contract-then-compress (C-T-C) algorithm (also called the "naïve" algorithm), and the randomized contract-than-compress algorithm, and the density matrix algorithm are all accurate but slow (at least on this problem with $D, \chi \gg 1$). The zip-up method, by contrast, is fast but less accurate. On this example, the fitting (i.e., variational) method is accurate but slower than the zip-up method. As we will see in section 4.2.1, the fitting method can require many sweeps to converge (making it significantly slower), or it may fail to converge at all.

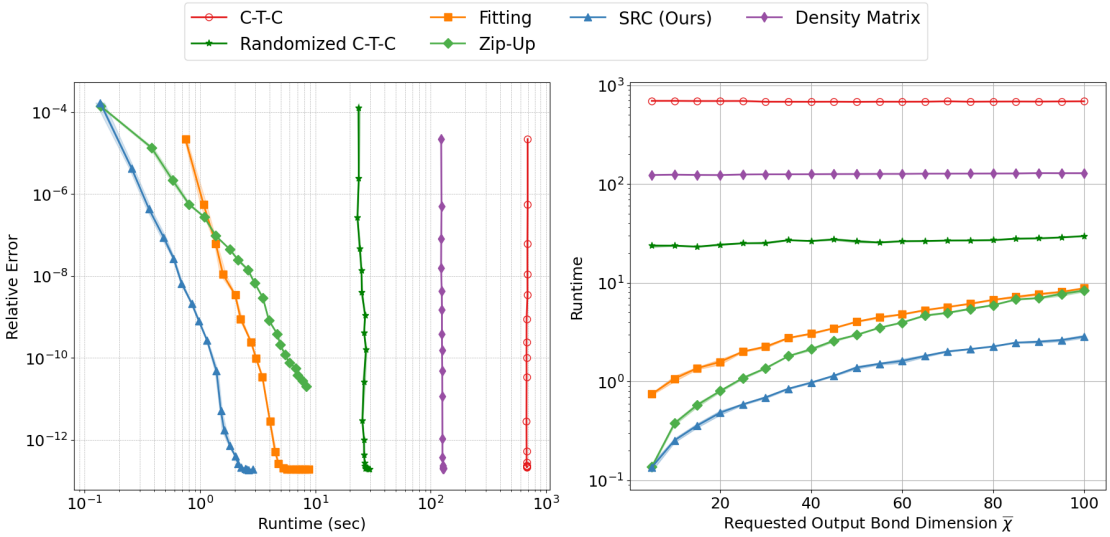


Figure 1: (**MPO–MPS contraction algorithm comparison**). Comparison of relative error versus computation time of several algorithms for the MPO–MPS product of $n = 100$ tensor networks with bond dimension $D = \chi = 50$. The maximum bond dimension $\overline{\chi}$ was varied between 5 and 100, with the compressibility difficulty parameter α set to -0.5. The fitting (variational) method was run for a single left-to-right sweep; more sweeps led to higher computational cost at no appreciable gain in accuracy. C-T-C is an abbreviation for "contract-then-compress". Randomized C-T-C refers to the contract-then-compress method with randomized MPS rounding (see section 4.1.2). Each data point represents the mean of five independent runs.

For faster and more reliable MPO–MPS multiplication, this paper proposes a new *randomized algorithm* for the compressed MPO–MPS product called *successive randomized compression* (SRC). The SRC method has the following desirable properties:

- **Fast.** In terms of both empirical runtime and asymptotic operation count, our algorithm is competitive with the fastest available algorithms for MPO–MPS multiplication. In particular, the SRC method is asymptotically faster than the naïve (contract-then-compress) and density matrix methods.
- **Accurate.** For a given bond dimension, the output $|\eta\rangle \approx H|\psi\rangle$ of the SRC algorithm achieves accuracy comparable to the contract-then-compress method, which is known to achieve near-optimal truncation error [34, Cor. 2.4]. In particular, the accuracy of SRC is often significantly better than the zip-up algorithm.
- **One-shot.** SRC proceeds in a single shot with no need to iterate until convergence. This compares favorably to the fitting method and to nonlinear optimization methods, which can require many iterations to achieve convergence or can fail to converge.

We believe this suite of benefits make the SRC method a compelling candidate for deployment in applications. Indeed, in Figure 1, SRC achieves higher accuracy in less time than other randomized MPO–MPS methods.

1.1 Outline for paper

After introducing background material on randomized matrix approximation and Khatri–Rao products in section 2, we present our algorithm in section 3. Description of and comparison with existing MPO–MPS product methods appears in section 4, and an application to unitary time evolution using the TDVP1-GSE algorithm appears in section 5.

1.2 Notation

The Frobenius norm of a matrix or tensor is $\|\cdot\|_{\mathrm{F}}$. We denote the Hermitian adjoint of a matrix by †. The symbol $\otimes$ refers to the Kronecker product of vectors or matrices, while $\odot$ denotes the Khatri–Rao product of matrices (see section 2.2). We express tensors $|\psi\rangle \in \mathbb{C}^{d\times\cdots\times d}$ using bra–ket notation. The constituent tensors of an MPS $|\psi\rangle$ are denoted $\psi^{(1)}, \ldots, \psi^{(n)}$ (similarly, $H^{(i)}$ for an MPO H).

1.3 Source code

Code for our algorithm, other MPO–MPS multiplication algorithms, and to reproduce the experiments in this paper can be found at the following URL:

https://github.com/chriscamano/RandomMPOMPS

2 Background

This section reviews background material for our randomized method for the compressed MPO–MPS multiplication algorithm, including randomized low-rank approximation (section 2.1) and the Khatri–Rao product (section 2.2).

2.1 Randomized QB approximation

Randomized low-rank approximation algorithms [35–38] are a class of fast, robust methods for approximating a large matrix $A \in \mathbb{C}^{M\times N}$ as a product $A \approx QB$ of thin matrix $Q \in \mathbb{C}^{M\times p}$ times a wide matrix $B \in \mathbb{C}^{p\times N}$. In this section, we review the simplest of such methods, randomized QB approximation. In the next section (section 3), we will show how the randomized QB approximation can be used to give a fast algorithm for the MPO–MPS product.

In its most basic form, the randomized QB approximation consists of four steps:

Randomized QB approximation. On input $A \in \mathbb{C}^{M\times N}$,

(i) **Generate randomness.** Generate a random matrix $\Omega \in \mathbb{C}^{N\times p}$.

(ii) **Collect information.** Compute the matrix product $Y := A\Omega$.

(iii) **Orthonormalize.** Orthonormalize the columns of Y, yielding $Q := \mathrm{orth}(Y)$.

(iv) **Project.** Set $B := Q^\dagger A$, defining an approximation $\widehat{A} = QB$.

The randomized QB approximation $\widehat{A} = QB$ is the same as the approximation produced by the randomized SVD algorithm [35], but presented in the form $\widehat{A} = QB$ rather than as an SVD. The matrix Q has orthonormal columns, and the matrix B has no special structure. When the randomized QB approximation is exact ($A = \widehat{A}$), we call the expression $A = QB$ a (randomized) QB *decomposition*.

The accuracy of randomized QB approximation can be understood using the following result [35, Thm 10.5] (see also [39, Fact A.2]):

Theorem 1 (Randomized QB approximation: Gaussian). *Let $\widehat{A}$ be the rank-p randomized QB approximation for A, formed using a random matrix $\Omega \in \mathbb{C}^{N\times p}$ whose*

entries are independent draws from either the real or complex standard normal distribution. Then

- ***Exactly low-rank, exact recovery.*** *Suppose that* $\operatorname{rank} A \leq p$. *With probability one, randomized QB approximation recovers* A *exactly (that is,* $\widehat{A} = A$*).*

- ***Approximately low-rank, near-optimal approximation.*** *Fix a parameter* $r \leq p - 1 - \alpha$ *and let* A_r *denote the best rank-*r *approximation to* A *with respect to the Frobenius norm. Then*

$$\mathbb{E}\left[\|A - \widehat{A}\|_{\mathrm{F}}^2\right] \leq \left(1 + \frac{r}{p - r - \alpha}\right) \|A - A_r\|_{\mathrm{F}}^2. \tag{2.1}$$

The number $\alpha = 0, 1$ *for the complex or real normal distributions, respectively.*

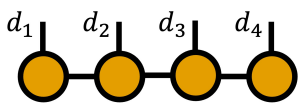
The conclusion of this result is that randomized QB approximation exactly recovers genuinely low-rank matrices and, with modest oversampling such as $p = r + 5$ or $p = 2r$, produces an approximation that is comparable with the best rank-r approximation. See [35, §10], [37, §8], and the references therein for many more analyses of randomized QB approximation. While most of the *analysis* of randomized QB approximation focuses on the Gaussian case, the method usually works quite well *in practice* for a much richer class of random matrices Ω (see [40, §11.5.2] and [41, Fig. 1]).

The accuracy of randomized QB approximation can be improved by using a random matrix Ω that is more aligned with the "important directions" (i.e., right singular vectors) of the matrix A. Such a better Ω can be obtained systematically by applying subspace or Krylov iteration [35, 37, 42, 43] or by using prior knowledge about the matrix [44]. In the context of this work, subspace iteration and Krylov iteration are of prohibitively high cost, and we assume no prior information is available. Therefore, a random Ω, generated independently from the matrix A, is the best-available choice.

Randomized QB approximation or the randomized SVD can be used as a replacement for a truncated full SVD in any algorithm, and there are several papers [45–48] that have proposed using the randomized SVD as a fast alternative to full SVDs in tensor network algorithms. In this paper, we go further by applying the randomized QB approximation directly to an entire tensor network.

2.2 The Khatri–Rao product

In this work, we will apply randomized QB approximations to *matrix unfoldings of tensor networks*. For instance, the following MPS



represents a tensor of size $d_1 \times d_2 \times \cdots \times d_4$, i.e., $|\psi\rangle \in \mathbb{C}^{d_1 \times d_2 \times d_3 \times d_4}$. The tensor $|\psi\rangle$ can be viewed as a matrix in several different ways. For instance, if we choose the third and fourth indices to be rows and the first and second to be columns, we can treat $|\psi\rangle$ as a $(d_3 d_4) \times (d_1 d_2)$ matrix.

To perform QB approximation efficiently for matrix unfoldings of tensor networks, we can use a *tensor-structured* random matrix Ω. There are many different types of tensor-structured random matrices that have been proposed for linear algebraic computations; see [38, Ch. 7] for a survey. In this paper, we will use the simplest example, a Khatri–Rao product of random matrices.

The Khatri–Rao product [49, §2.6] is defined as follows. Let $\Omega^{(1)}, \ldots, \Omega^{(n)} \in \mathbb{C}^{d \times p}$ be matrices, where each matrix $\Omega^{(j)}$ has columns

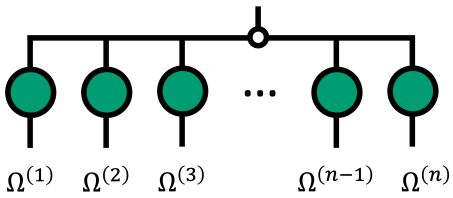
$$\Omega^{(j)} = \begin{bmatrix} \omega_1^{(j)} & \cdots & \omega_p^{(j)} \end{bmatrix}.$$

The *Khatri–Rao product* of $\Omega^{(1)}, \ldots, \Omega^{(n)}$ is defined as the columnwise Kronecker product

$$\Omega^{(1)} \odot \cdots \odot \Omega^{(n)} = \begin{bmatrix} (\omega_1^{(1)} \otimes \omega_1^{(2)} \otimes \cdots \otimes \omega_1^{(n)}) & \cdots & (\omega_p^{(1)} \otimes \omega_p^{(2)} \otimes \cdots \otimes \omega_p^{(n)}) \end{bmatrix} \in \mathbb{C}^{d^n \times p}.$$

To apply randomized QB approximation to MPO–MPS products in this paper, we will define $\Omega := \Omega^{(1)} \odot \cdots \odot \Omega^{(n)}$ to be a Khatri–Rao product of random matrices $\Omega^{(1)}, \ldots, \Omega^{(n)}$ with standard (real or complex) normal entries, all mutually independent. The resulting *Khatri–Rao random matrix* Ω consists of a collection of random product states $\omega_j^{(1)} \otimes \omega_j^{(2)} \otimes \cdots \otimes \omega_j^{(n)}$, stacked columnwise. We can compute the action $A\Omega$ of a matrix A on Ω by applying A to each product state, one at a time.

In tensor diagram notation, the Khatri–Rao product can be denoted as



We emphasize in this diagram that the white circle at the top of this diagram indicates multiplication of the corresponding indices entrywise. See [50, §1.2 & §3.7] for more details on denoting Khatri–Rao products using tensor diagrams.

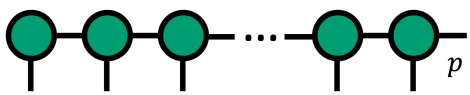
At present, our theoretical understanding for randomized QB approximation with a Khatri–Rao test matrix $\Omega = \Omega^{(1)} \odot \cdots \odot \Omega^{(n)}$ remains limited. It is known that this method supports exact recovery of a low-rank matrix A:

Theorem 2 (Randomized QB approximation: Khatri–Rao). *Let $\widehat{A}$ be the rank-p randomized QB approximation of A where $\Omega = \Omega^{(1)} \odot \cdots \odot \Omega^{(n)} \in \mathbb{C}^{N \times p}$ formed as a Khatri–Rao product of matrices $\Omega^{(i)}$ with independent standard normal entries. Then*

- ***Exactly low-rank, exact recovery.*** *Suppose that* $\operatorname{rank} A \leq p$. *With probability one, randomized QB approximation recovers A exactly (that is, $\widehat{A} = A$).*

Section A contains a proof of Theorem 2. We expect that this result appears in the literature, but we were unable to locate a citation. After the initial release of this paper, the authors developed a quantitative error bound for randomized QB approximation, similar to (2.1), in collaboration with Raphael Meyer [41, §5].

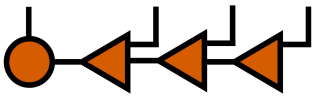
The random test matrices with Khatri–Rao product structure used in this work may be contrasted with the framework of "tensor train sketching" (e.g., see [14, 51–54]). In the latter approach, one chooses Ω to be a random MPS with a single extra exposed index at the right-most site, which has dimension p:



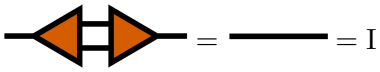
The tensor train sketching framework has advantages and drawbacks over the random Khatri–Rao product matrices used in this work. As advantages, the user has more freedom, as they can tune the bond dimensions of this MPS as a way of trading off the accuracy of the algorithm against computational cost. As disadvantages, the computational effort of tensor train sketching is larger than random Khatri–Rao test matrices, and setting the bond dimensions of the tensor train sketching matrix generally requires either problem-dependent tuning or prior knowledge about the problem.

3 Successive randomized compression

In this section, we introduce a new randomized algorithm, *successive randomized compression* (SRC), for the compressed MPO–MPS product. This algorithm constructs an MPS representation of $|\eta\rangle \approx H|\psi\rangle$ one site at a time in a single pass from right-to-left. The outcome is an MPS representation of $|\eta\rangle \approx H|\psi\rangle$ in right canonical form. Here is an illustration for $n = 4$ sites:



In this network, triangular tensors denote partial isometries, satisfying



(identity operator).

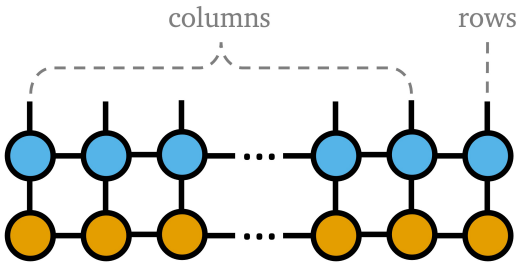
For clarity of exposition, we will describe the SRC algorithm under the following assumptions:

- The product $|\eta\rangle = H|\psi\rangle$ is exactly representable as an MPS with bond dimension $\overline{\chi} < D \cdot \chi$.
- The output bond dimension $\overline{\chi}$ is provided as an input to the algorithm.

In practice, the goal is typically to compute an *approximate* compression $|\eta\rangle \approx H|\psi\rangle$ and, often, to determine $\overline{\chi}$ adaptively to satisfy an error tolerance. The algorithm works without modification for approximate contraction (though a final rounding step can be helpful, see section 3.4). We describe the algorithm in detail in sections 3.1 to 3.3, and we analyze its computational cost in section 3.5. Adaptive determination of bond dimension is deferred to section C.

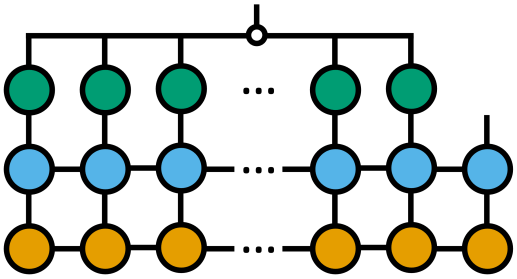
3.1 Step 1: The last site

Begin by viewing $H|\psi\rangle$ as a $d \times d^{n-1}$ matrix, with the last visible index as the rows and the first $n-1$ indices as the columns:

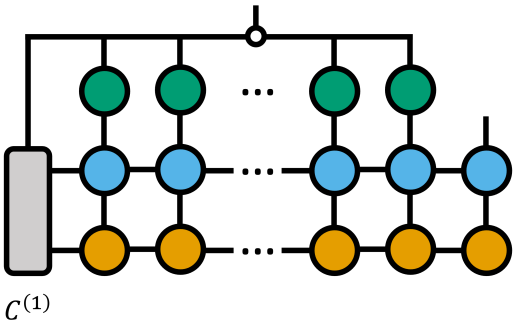


We will compute a randomized QB decomposition of this matrix.

Begin by drawing random matrices $\Omega^{(1)}, \ldots, \Omega^{(n-1)} \in \mathbb{C}^{d \times \overline{\chi}}$ with independent real or complex standard normal entries, instantiating the Khatri–Rao product $\Omega^{(1:n-1)} := \Omega^{(1)} \odot \cdots \odot \Omega^{(n-1)}$, and applying it to $H|\psi\rangle$. This process results in the tensor network

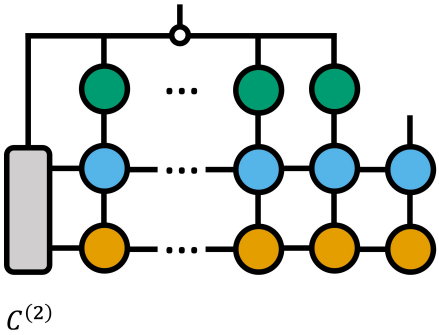
$(H|\psi\rangle) \cdot \Omega^{(1:n-1)} =$  .

We now contract this $(H|\psi\rangle) \cdot \Omega$ network. First, contract the left-most stack of three tensors to obtain a new tensor $C^{(1)}$:

$(H|\psi\rangle) \cdot \Omega^{(1:n-1)} =$ 

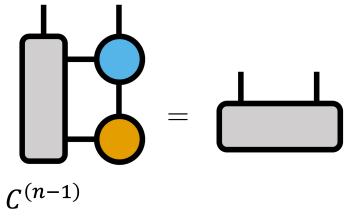
.

Next, contract one step to the right, resulting in a new tensor $C^{(2)}$:

$(H|\psi\rangle) \cdot \Omega^{(1:n-1)} =$ 

.

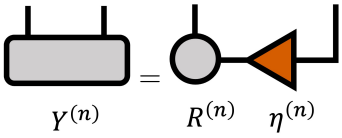
We continue contracting left-to-right, producing tensors $C^{(3)}, C^{(4)}, \ldots, C^{(n-1)}$, which we contract down to a matrix:

$(H|\psi\rangle) \cdot \Omega^{(1:n-1)} =$ 

.

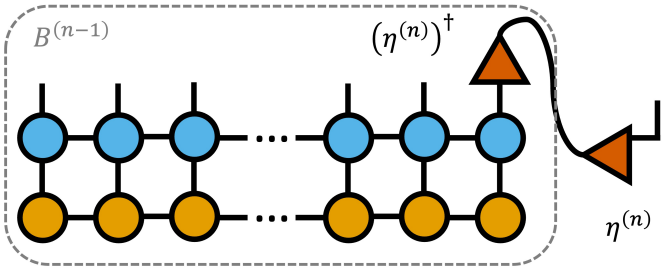
We have now collected $Y^{(n)} := (H|\psi\rangle) \cdot \Omega^{(1:n-1)}$, which is step (ii) of randomized QB decomposition.

Next is step (iii), orthonormalization, which we accomplish by a QR decomposition:

$$Y^{(n)} = (H|\psi\rangle) \cdot \Omega^{(1:n-1)} =$$  $$= \eta^{(n)} \cdot R^{(n)}. \tag{3.1}$$

Since the right index of $Y^{(n)}$ corresponds to the rows, the orthogonal "Q" factor $\eta^{(n)}$ occurs to the right of the triangular "R" factor $R^{(n)}$ in the tensor diagram (3.1), but the factorization is written $Y^{(n)} = \eta^{(n)} \cdot R^{(n)}$ in matrix notation. The tensor $\eta^{(n)}$ will form the final site of the output $|\eta\rangle = H|\psi\rangle$.

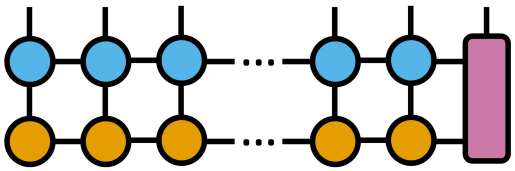
Now, we perform step (iv) of QB decomposition, projection:

$$H|\psi\rangle = \eta^{(n)}(\eta^{(n)})^\dagger(H|\psi\rangle) =$$ 

$$=: \eta^{(n)} \cdot B^{(n-1)}\,.$$

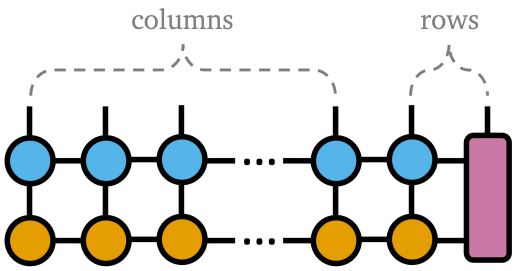
Remember that we are assuming, for now, that $H|\psi\rangle$ is exactly compressible to an MPS of bond dimension $\overline{\chi}$. Thus, by Theorem 2, the randomized QB decomposition is exact with probability one.

In the next step of the algorithm, we will produce site $n-1$ of $H|\psi\rangle$ by applying randomized QB decomposition to $B^{(n-1)}$. In preparation for this step, we contract $(\eta^{(n)})^\dagger$ into the MPO and MPS sites below, obtaining

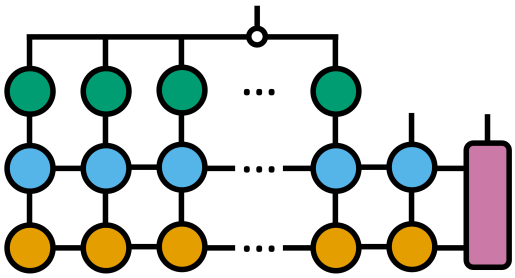
$$B^{(n-1)} =$$  $$. \tag{3.2}$$

3.2 Step 2: Second-to-last site

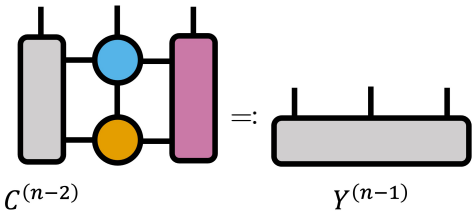
We now compute a randomized QB decomposition of $B^{(n-1)}$. We treat $B^{(n-1)}$ as a $\overline{\chi}d \times d^{n-2}$ matrix, with the last two exposed indices as rows and the first $n-2$ visible indices as columns:



The key step which allows our algorithm to be computationally efficient is *reuse* of the same random matrices $\Omega^{(1)}, \ldots, \Omega^{(n-2)}$ employed in the first step of the algorithm. Specifically, we form the Khatri–Rao product $\Omega^{(1:n-2)} := \Omega^{(1)} \odot \cdots \odot \Omega^{(n-2)}$ and apply it to $B^{(n-1)}$, resulting in the following tensor network:

$B^{(n-1)} \cdot \Omega^{(1:n-2)} =$  .

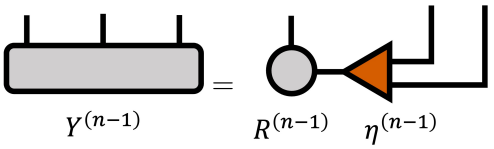
Now, since we use the same random tensors $\Omega^{(1)}, \ldots, \Omega^{(n-2)}$, we can reuse the work from step 1 by using the tensor $C^{(n-2)}$ we saved to obtain

$B^{(n-1)} \cdot \Omega^{(1:n-2)} =$ 

.

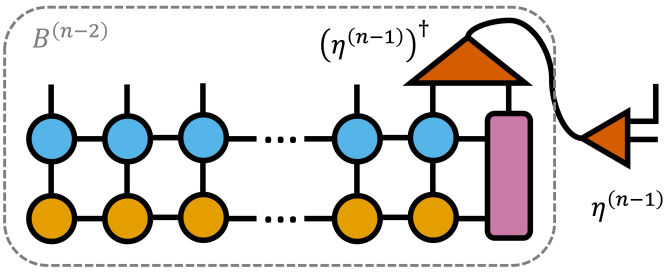
We have further contracted down $B^{(n-1)} \cdot \Omega^{(1:n-2)}$ to a three-mode tensor $Y^{(n-1)}$. This completes step (ii) of QB decomposition for $B^{(n-1)}$.

Next, we do step (iii) of QB decomposition, orthonormalization, by computing a QR decomposition of $Y^{(n-1)}$:

$Y^{(n-1)} = B^{(n-1)} \cdot \Omega^{(1:n-2)} =$ 

$= \eta^{(n-1)} \cdot R^{(n-1)}$.

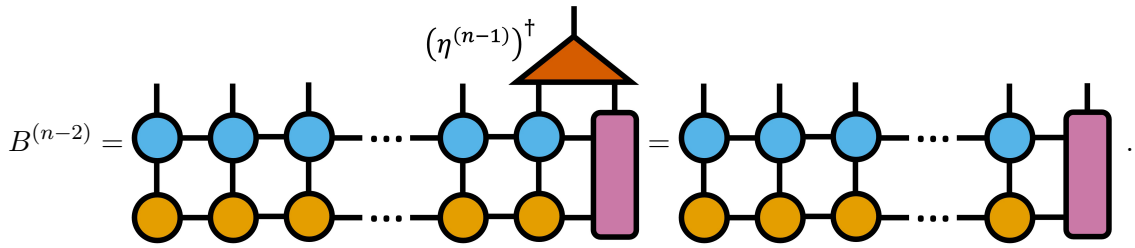
Now, we apply the step (iv) of QB decomposition, projection:

$B^{(n-1)} = \eta^{(n-1)} (\eta^{(n-1)})^\dagger B^{(n-1)} =$ 

.

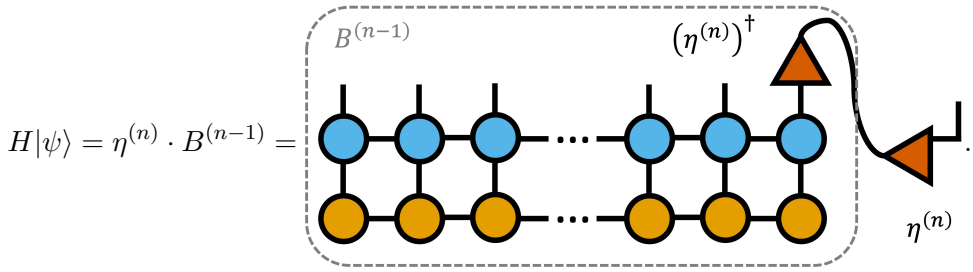
Again, because we have assumed $H|\psi\rangle$ is exactly representable as an MPS with bond dimension $\overline{\chi}$, the QB decomposition $B^{(n-1)} = \eta^{(n-1)} \cdot B^{(n-2)}$ is exact with probability one.

For the next step of the algorithm, we clean up the $B^{(n-2)}$ by contracting in $(\eta^{(n-1)})^\dagger$:

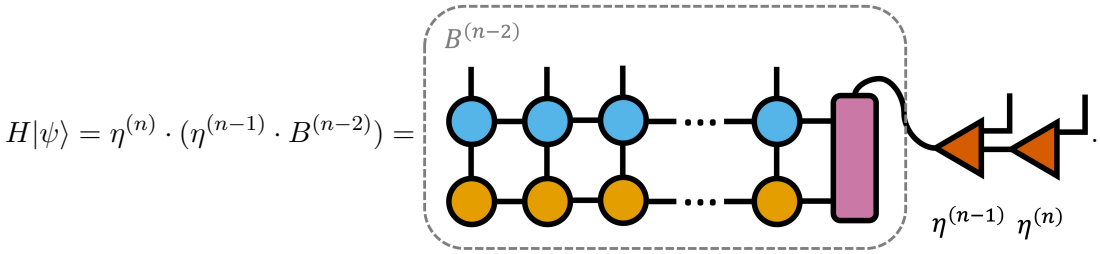


Observe that $B^{(n-2)}$ now has $n-2$ exposed indices of dimension d and 1 exposed index of dimension $\overline{\chi}$; the total number of exposed indices, $n-1$, has been decreased by one.

To see how the algorithm is making progress in representing $H|\psi\rangle$ as an MPS $|\eta\rangle$, observe that in step one, we obtained a factorization $H|\psi\rangle = \eta^{(n)} \cdot B^{(n-1)}$:



Step two yielded a factorization $B^{(n-2)} = \eta^{(n-1)} \cdot B^{(n-2)}$. Thus,

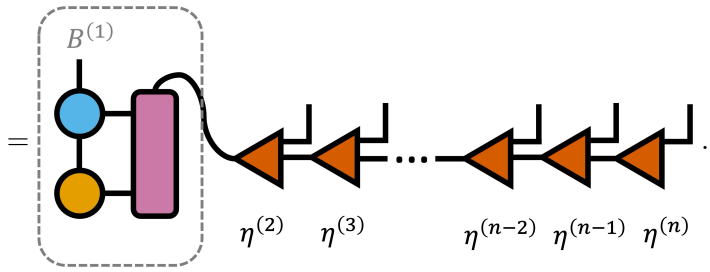


After two steps of the algorithm, we have produced the right two sites $\eta^{(n-1)}$ and $\eta^{(n)}$ of an MPS $|\eta\rangle = H|\psi\rangle$. When completed, the MPS $|\psi\rangle$ will be in left canonical form.

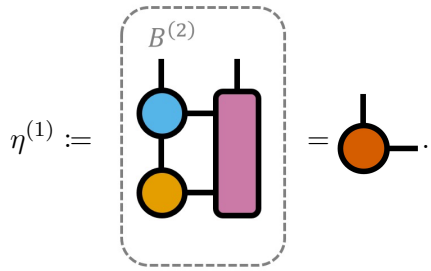
3.3 Finishing up

Proceeding as in step two above, we continue right-to-left producing orthogonal tensors $\eta^{(n)}, \eta^{(n-1)}, \eta^{(n-2)}, \eta^{(n-3)}, \ldots$. By the time we reach site two of $H|\psi\rangle$ and compute $\eta^{(2)}$, we have obtained a tensor network of the form

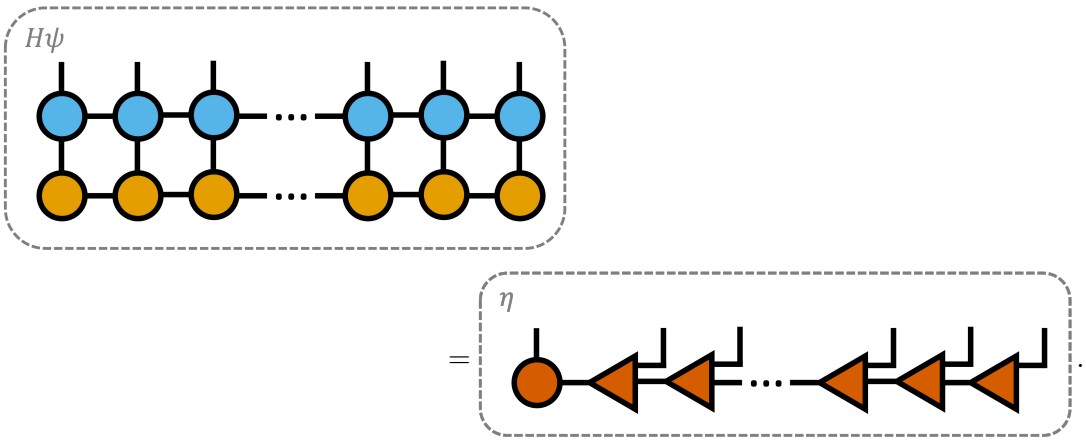
$$H|\psi\rangle = \eta^{(n)} \times \eta^{(n-1)} \times \cdots \times \eta^{(2)} \times B^{(1)}$$



To obtain the first site tensor $\eta^{(1)}$, we take the $B^{(1)}$ tensor network and contract it down:



We conclude, having computed the MPO–MPS product $|\eta\rangle = H|\psi\rangle$ as an MPS in right canonical form:



3.4 Optional step: Oversampling and final round

In sections 3.1 to 3.3, we introduced the SRC algorithm under the assumption that the product $H|\psi\rangle$ is exactly representable as an MPS of known bond dimension $\overline{\chi}$. However, the algorithm works without modification for computing a compressed approximation $H|\psi\rangle$ as an MPS with specified bond dimension $\overline{\chi}$.

When using the SRC method for approximate MPO–MPS multiplication, *oversampling* can be used to achieve accuracy comparable with the best MPS representation of the product $H|\psi\rangle$. To compute a compressed representation $|\eta\rangle \approx H|\psi\rangle$ of bond dimension $\overline{\chi}$, we do the following. First, run SRC with a larger bond dimension $\overline{\chi}' > \overline{\chi}$, obtaining an MPS in right canonical form. Then, use a standard MPS compression algorithm to truncate down to an MPS in left canonical form with desired output bond dimension $\overline{\chi}$. As a sensible default, we recommend $\overline{\chi}' = \max(\lceil 1.5\overline{\chi}\rceil, \overline{\chi} + 10)$.

3.5 Summary: Pseudocode and time complexity

As we have seen, SRC produces an MPS representation $|\eta\rangle$ of the MPO–MPS product $H|\psi\rangle$ using a sequence of QB decompositions or approximations. For ease of presentation, we first presented the algorithm for the case in which $H|\psi\rangle$ is exactly representable as an MPS of specified bond dimension $\overline{\chi}$. In this case, SRC produces an exact representation $|\eta\rangle = H|\psi\rangle$:

Theorem 3 (Successive randomized compression: Exact recovery). *If the MPO–MPS product $H|\psi\rangle$ is exactly representable as an MPS of bond dimension $\overline{\chi}$, then SRC recovers $|\eta\rangle = H|\psi\rangle$ with probability one.*

Algorithm 1 successive randomized compression for MPO–MPS multiplication

Input: MPO H, MPS $|\psi\rangle$, and output bond dimension $\overline{\chi}$ (or tolerance τ, see section C.2)
Output: MPS $|\eta\rangle \approx H|\psi\rangle$ of bond dimension $\overline{\chi}$

Multiply with random matrix
Generate standard Gaussian matrices $\Omega^{(1)}, \ldots, \Omega^{(n-1)} \in \mathbb{C}^{d\times\overline{\chi}}$
Compute tensor $C^{(1)}(a,b,c) \leftarrow \sum_{d,e} \Omega^{(1)}(a,d)H^{(1)}(d,b,e)\psi^{(1)}(e,c)$
for $i = 2, \ldots, n-1$ **do**
    Compute tensor $C^{(i)}(a,b,c) \leftarrow \sum_{d,e,f,g} C^{(i-1)}(a,d,e)\Omega^{(i)}(a,f)H^{(i)}(d,f,b,g)\psi^{(i)}(e,g,c)$
end for

Determine the last site $\eta^{(n)}$
Compute tensor $Y^{(n)}(a,b) \leftarrow \sum_{c,d,e} C^{(n-1)}(a,c,d)H^{(n)}(b,c,e)\psi^{(n)}(d,e)$
Compute QR decomposition $Y^{(n)}(a,b) = \sum_c \eta^{(n)}(c,b)R^{(n)}(a,c)$
[***Optional:*** *Compute error estimate and increase $\overline{\chi}$ if needed; see section C.2*]
Compute tensor $S^{(n)}(a,b,c) \leftarrow \sum_{d,e} \overline{\eta^{(n)}(c,d)}H^{(n)}(b,d,e)\psi^{(n)}(c,e)$

Determine sites $\eta^{(n-1)}, \ldots, \eta^{(2)}$
for $j = n-1, n-2, \ldots, 2$ **do**
    Compute tensor $Y^{(j)} \leftarrow \sum_{d,e,f,g,h} C^{(j-1)}(a,d,e)H^{(j)}(d,b,f,g)\psi^{(j)}(e,g,h)S^{(j+1)}(h,f,c)$
    Compute QR decomposition $Y^{(j)}(a,b,c) = \sum_d \eta^{(j)}(d,b,c)R^{(j)}(a,d)$
    [***Optional:*** *Compute error estimate and increase $\overline{\chi}$ if needed; see section C.2*]
    Compute tensor $S^{(j)}(a,b,c) \leftarrow \sum_{d,e,f,g,h} \overline{\eta^{(j)}(c,d,e)}H^{(j)}(b,d,f,g)\psi^{(j)}(a,g,h)S^{(j+1)}(h,f,e)$
end for

Determine the first site $\eta^{(1)}$
Compute tensor $\eta^1(a,b) \leftarrow \sum_{c,d,e} H^{(1)}(a,c,d)\psi^{(1)}(d,e)S^{(2)}(e,d,b)$
[***Optional:*** *Run an MPS truncation algorithm on $|\eta\rangle$*]

This result follows directly from Theorem 2, though the fact that we use a common set of random matrices $\Omega^{(1)}, \ldots, \Omega^{(n-1)}$ across all steps of the algorithms makes the result not entirely trivial. Section B contains a proof of this result.

When $H|\psi\rangle$ is not exactly equal to an MPS of bond dimension $\overline{\chi}$, SRC returns a compressed approximation $|\eta\rangle \approx H|\psi\rangle$ to the product. We present pseudocode for SRC in Algorithm 1. See section C for a discussion of determining the output bond dimension $\overline{\chi}$ adaptively to meet a tolerance.

The runtime of SRC depends on the number of sites n, the MPO and MPS bond dimensions D and χ, the physical dimension d, and the output bond dimension $\overline{\chi}$. In terms of all of these parameters, SRC has a time complexity of

$$\mathcal{O}(ndD\chi\overline{\chi}(\chi + \overline{\chi} + dD)) \text{ operations.} \tag{3.3a}$$

Making a few assumptions about parameter values, we obtain a simplified runtime estimate

$$\mathcal{O}(nD\chi^3) \text{ operations,} \quad \text{assuming } D \leq \chi = \overline{\chi} \text{ and } d = \text{const.} \tag{3.3b}$$

Using either the full runtime (3.3a) or simplified runtime (3.3b), SRC is as fast as or faster than all existing MPO–MPS multiplication algorithms; see section 4 for a description of other MPO–MPS multiplication algorithms and a runtime comparison with SRC.

3.6 Extension: Linear combinations of MPO–MPS products

As an extension, this algorithm can also be used to form a compressed representation of a linear combination of MPO–MPS products:

$$|\eta\rangle \approx \sum_{i=1}^{t} \alpha H_i |\psi_i\rangle.$$

At each phase of the algorithm, we compute a "local" $Y_i^{(j)}$ tensor separately for each term, and compute the global $Y^{(j)} = \sum_{i=1}^{t} \alpha_i Y_i^{(j)}$ as a linear combination. The global $Y^{(j)} = \eta^{(j)} \cdot R^{(j)}$ tensor is then split using a QR factorization, and we use the same $\eta^{(j)}$ to define a local tensor $B_i^{(j-1)} = (\eta^{(j)})^\dagger B_i^{(j)}$ for each term. We conclude the algorithm by setting $\eta^{(1)} := \sum_{i=1}^{t} \alpha_i B_i^{(1)}$. This randomized algorithm for compressing sums of MPO–MPS products is similar to the algorithm for compressing sums of MPSs in [51].

3.7 Limitation: Physical symmetries

In physical problems, MPOs and MPSs often obey group symmetry relations associated with physical conservation laws (e.g., symmetry under the group $U(1)$ arises from particle-number conservation). When symmetry is present, it can be built into the tensor network representation by requiring that the constituent tensors $\psi^{(1)}, \ldots, \psi^{(n)}$ or $H^{(1)}, \ldots, H^{(n)}$ have a prescribed block-sparsity structure; see [55, 56] for details.

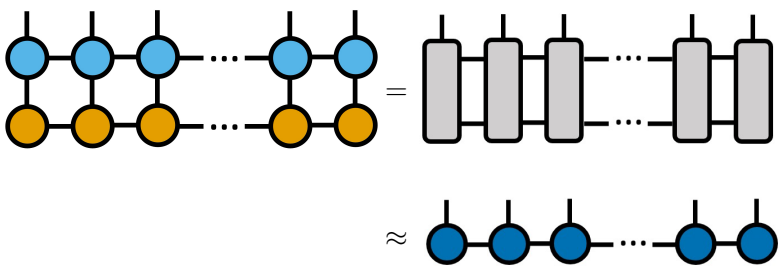
Unlike several other compressed contraction algorithms, the SRC algorithm presented in this paper is not symmetry-preserving: The intermediate tensors $C^{(i)}$ and $S^{(i)}$ computed by the algorithm do not possess sparsity structure, and the algorithm does not ensure the compressed output $|\eta\rangle$ has the appropriate block-sparsity structure. (The algorithm can, however, exploit sparsity of the constituent tensors by using sparse tensor formats to perform intermediate contractions in the algorithm.) Investigating whether the SRC algorithm can be modified to respect symmetries is a natural subject for future research.

4 Existing MPO–MPS product methods

Existing methods for the MPO–MPS product can be divided into three groups: contract-then-compress methods, optimization methods, and explicit construction algorithms. We review each of these classes of algorithms in turn.

4.1 Contract-then-compress methods

Contract then compress methods work by first contracting the MPO–MPS product *exactly* as an MPS of bond dimension $D\chi$, followed by a compression step to truncate the bond dimension:



For the second step, any MPS truncation algorithm can be used, leading to different variants of the contract-then-compress method. We review these variants here.

4.1.1 Basic contract-then-compress

The simplest contract-then-compress method uses the standard SVD-based MPS compression algorithm (see, e.g., [34, Alg. 2]). This scheme is known to produce a near-optimal MPS approximation to $H|\psi\rangle$ [34, Cor. 2.4 and sec. 3], but it is slow, requiring

$$\mathcal{O}(n \cdot [dD^3\chi^3 + d^2D^2\chi^2]) \text{ operations}$$

since it requires converting $H|\psi\rangle$ to canonical form prior to compression. Basic contract-then-compress is typically about D^2 times slower than the SRC, in view of (3.3b).

4.1.2 Randomized contract-then-compress

The contract-than-compress method can be improved by using a *randomized MPS compression method* [51]. Randomized contract-then-compress differs from the randomized method introduced in this paper in that it first represents $H|\psi\rangle$ exactly as an MPS and then compresses this MPS; by contrast, our algorithm avoids ever working with the exact MPS form of $H|\psi\rangle$, resulting in a faster algorithm.

The paper [51] presents three different randomized MPS compression algorithms; for the purpose of this paper, we consider their *randomize-then-orthogonalize* method [51, Alg. 3.2]. Using this method, randomized contract-then-compress has a runtime of

$$\mathcal{O}(n \cdot [dD^2\chi^2\overline{\chi} + d^2D^2\chi^2]) \text{ operations.}$$

This improves on the basic contract-then-compress method, but is still typically a factor D times slower than SRC. Similar to SRC, randomized contract-then-compress is typically a constant factor less accurate than the basic contract-then-compress method. At present, randomized contract-then-compress is limited to the setting where the output bond dimension $\overline{\chi}$ is provided as an input to the algorithm (i.e., not determined adaptively).

4.1.3 Variational contract-then-compress

A third possible implementation of contract-then-compress uses *variational compression* to compress the MPO–MPS product $H|\psi\rangle$. Variational MPS is a cyclic minimization algorithm that attempts to minimize the error $\||\phi\rangle - |\eta\rangle\|_{\mathrm{F}}$ of approximating a provided MPS $|\phi\rangle$ by an MPS $|\eta\rangle$ of smaller dimension; see [57, §4.5.2] and [23, §2.6.2] for details. As with DMRG (and fitting, below), there are one-site and two-site versions of the variational compression method, with the two-site version being somewhat more expensive but adaptive and more robust. With the two-site procedure, variational contract-then-compress costs

$$\mathcal{O}(n \cdot [d^2D^2\chi^2\overline{\chi}]) \text{ operations per sweep.}$$

Similar to the fitting algorithm (below), variational MPS can have slow convergence (requiring many sweeps) or fail to converge at all on some examples.

4.2 Optimization methods

Optimization methods treat the MPO–MPS multiplication problem as an optimization problem

$$\underset{|\eta\rangle=\mathrm{MPS}(\eta^{(1)},\ldots,\eta^{(n)})}{\text{minimize}} \||\eta\rangle - H|\psi\rangle\|_{\mathrm{F}}^2. \tag{4.1}$$

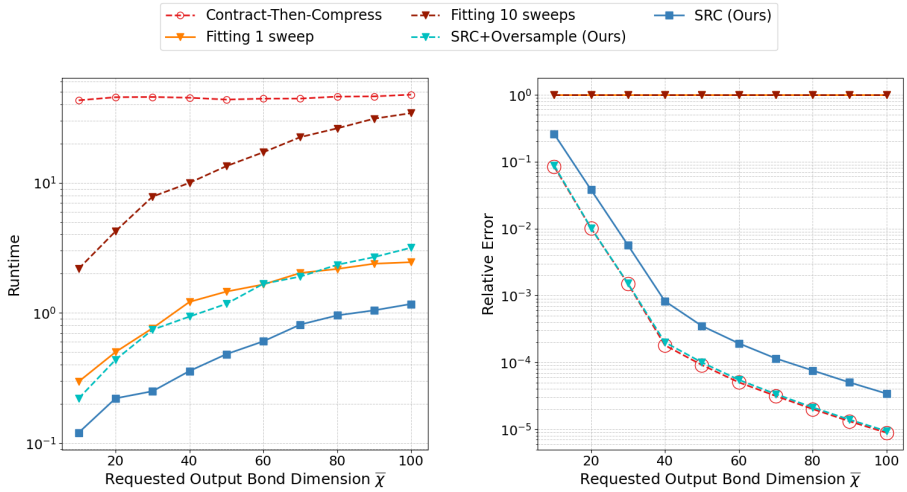


Figure 2: (**Fitting failure mode**). Runtime (*left*) and relative error (*right*) of the fitting algorithm, SRC, and contract-then-compress on the MPO–MPS product between the long range XY Hamiltonian given in section 5.2 and its groundstate.

Here, the minimum is taken over all MPSs $|\eta\rangle$, say, of specified bond dimension $\overline{\chi}$. There are two optimization methods, the fitting method and the global optimization method.

4.2.1 Fitting

The fitting method, proposed by Verstraete and Cirac [28], attempts to solve the optimization problem (4.1) by cyclic minimization, iteratively minimizing over each $\eta^{(i)}$ while holding the other sites. This site-by-site minimization is performed in alternating sweeps left-to-right then right-to-left, *à la* DMRG. See [23, §2.8.2] for details. The fitting method is distinguished from the variational contract-then-compress method because contract-then-compress represents $H|\psi\rangle$ directly as an MPS of bond dimension $D\chi$, which it then compresses; the fitting method avoids ever forming the uncompressed MPO–MPS product $H|\psi\rangle$, leading to a fast algorithm.

The basic Verstraete–Cirac fitting method can suffer from slow convergence and does not admit a natural way to determine the bond dimension of $|\eta\rangle$ adaptively to meet a tolerance. To mitigate these issues, Stoudenmire and White [58] proposed a two-site version of the fitting method which minimizes a pair of sites $\eta^{(i)}, \eta^{(i+1)}$ and then splits by SVD, allowing for adaptive determination of the bond dimension. The relation of one-site and two-site fitting is analogous to one-site and two-site DMRG. All experiments in this paper use the two-site fitting algorithm.

On a per-sweep basis, the fitting method is a fast, requiring $\mathcal{O}\left(n \cdot [dD\chi\overline{\chi}^2 + d^2D^2\chi^2]\right)$ operations. However, as Stoudenmire and White [58] warn, even the more robust two-site fitting algorithm can converge slowly or fail to converge at all, particularly for MPOs that capture long-range interactions. Figure 2 illustrates this behavior for the long range XY Hamiltonian H used in section 5.2. In this example, we compute the MPO–MPS product of H with its ground state $|\psi_0\rangle$, which, again, was obtained using DMRG2 with tolerance 10^{-8}. The fitting algorithm is supplied with the ground state $|\psi_0\rangle$ as the initial guess, which agrees with the output $H|\psi_0\rangle$ up to scaling by the energy $E_0 \neq 0$ and numerical errors. Despite performing 10 sweeps, the fitting algorithm fails to converge, or make any progress towards solving the problem. We observe the same behavior when we initialize the fitting algorithm with a random MPS with Gaussian random entries.

Similar to the fitting algorithm, the DMRG algorithm can also suffer from convergence to local minima. Researchers have suggested adding random noise [59, 60] or using randomized low-rank approximation algorithms [48] as strategies to avoid this behavior, and these strategies can also be applied to the fitting algorithm. We have not evaluated these more sophisticated versions of the fitting algorithm in this paper.

4.2.2 Global optimization

To overcome the shortcomings of cyclic minimization algorithms like fitting, Hauru, Van Damme, and Haegeman investigates global Riemmanian optimization algorithms for tensor network problems [61]. In principle, these methods can have a runtime competitive with other fast MPO–MPS algorithms, but they are relatively complicated to implement. We are unaware of a publicly available implementation of this method for the MPO–MPS product.

4.3 Explicit construction methods

The last family of algorithms, explicit construction methods, directly construct an approximate MPO–MPS product without either the initial representation of $H|\psi\rangle$ as an MPS of bond dimension $D\chi$ in contract-then-compress or the need to iterate until convergence required by optimization methods.

4.3.1 Zip-up

The zip-up method was proposed by Stoudenmire and White [58] as an alternative to the potentially slow convergence of the fitting algorithm. Zip-up begins by left-canonizing the input MPO and MPS at a cost of $\mathcal{O}(n \cdot [dD^3 + d\chi^3])$. Then, it proceeds left-to-right merging sites of the MPO and MPS, truncating as it proceeds. See [23, §2.8.3], [62, §3.2.2] for details.

As advantages, the zip-up algorithm has a fast $\mathcal{O}(n \cdot [dD\chi\overline{\chi}^2 + d^2D^2\chi\overline{\chi}])$ runtime and is a single-shot procedure, immune from the slow or non-convergence than can affect fitting-type methods. However, zip-up is less accurate than other MPO–MPS algorithm because the ith truncation step uses information from the first i sites, rather than the full environment. Describing their experience with the algorithm, Paeckel et al. [23, §2.8.3] write that the zip-up algorithm is "*typically* sufficiently accurate and fast" (emphasis ours) and suggest combining it with a few sweeps of the fitting algorithm to improve accuracy when necessary. By contrast, SRC is both fast and always achieves near-optimal truncation using the full environment.

4.3.2 Density matrix

The density matrix algorithm [63, 64] is conceptually similar to the randomized method developed in this work. In SRC, we construct an MPS approximation $|\eta\rangle$ to $H|\psi\rangle$ by a sequence of randomized QB approximations $B^{(n)} = H|\psi\rangle \approx \eta^{(n)} \cdot B^{(n-1)}$, $B^{(n-1)} \approx \eta^{(n-1)} \cdot B^{(n-2)}$, etc.; each $B^{(j)}$ is a matrix unfolding of an appropriate tensor network. Instead of randomized QB approximation, the density matrix algorithm constructs each $\eta^{(j)}$ as the leading eigenvectors of the *density matrix* $\rho^{(j)} := B^{(j)}(B^{(j)})^\dagger$.

The density matrix has advantages and disadvantages. Neglecting rounding errors, it produces an identical output to the contract-then-compress method, thus achieving near-optimality accuracy. Also, the density method is faster than contract-then-compress and more reliable than the fitting and zip-up methods.

Table 1: Simplified operation counts for methods to apply an MPO (bond dimension D) to an MPS (bond dimension χ). For this table, we assume $d = \mathcal{O}(1)$ and $D \leq \chi \leq \overline{\chi} \leq D\chi$; see Table 2 for a complete operation count that does not make these assumptions.

Method	Op. Count (simplified)
Contract-then-compress	
• Basic	$\mathcal{O}(nD^3\chi^3)$
• Randomized	$\mathcal{O}(nD^2\chi^2\overline{\chi})$
Density Matrix	$\mathcal{O}(nD^2\chi^2\overline{\chi})$
Fitting	$\mathcal{O}(nD\chi\overline{\chi}^2)$ per iteration
Zip-up	$\mathcal{O}(nD\chi\overline{\chi}^2)$
SRC	$\mathcal{O}(nD\chi\overline{\chi}^2)$

The main disadvantage of the density matrix algorithm is speed, being the second slowest method in Figure 1. SRC is nearly as accurate as the density matrix algorithm and often much faster. Another issue with the density matrix algorithm is numerical precision. Forming density matrix $B^{(j)}(B^{(j)})^\dagger$ effectively squares the matrix $B^{(j)}$, potentially halving the effective numerical precision. On certain examples, the relative error of the density matrix can stagnate near 10^{-8} in double precision (10^{-4} in single), where other methods achieve full accuracy ($\approx 10^{-16}$ in double, 10^{-8} in single). The density matrix algorithm faces these precision issues only on some examples, achieving full precision on others; we do not have an explanation for this inconsistency.

4.4 Comparison

In this section, we compare the SRC method to these alternatives and provide guidance about which algorithm to use. In all experiments, we use real standard Gaussian entries in random Khatri–Rao products for the SRC algorithm.

4.4.1 Complexity

We first compare the methods based on their operation counts, which are shown in Table 1. This table displays simplified operation counts where the physical dimension d is treated as a constant and the bond dimensions are ordered as $D \leq \chi \leq \overline{\chi}$; see Table 2 for more detailed operation counts in terms of all five parameters n, d, χ, D, and $\overline{\chi}$.

In terms of pure operation count, we see that the randomized and zip-up methods are fastest followed by density matrix and randomized contract-then-compress, all of which are faster than basic contract-then-compress. These operation counts are consistent with the runtime experiments provided in paper. The *per-iteration* cost of the fitting method is also comparable to the randomized and zip-up methods, but—as Figure 2 shows—the fitting method can take many iterations to converge or fail to converge outright for difficult examples. Finally, we notice that the operation counts of all methods are comparable when $\overline{\chi} \approx D\chi$; therefore, speedups are only possible for MPO–MPS multiplication when a significant amount of compression is used $\overline{\chi} \ll D\chi$.

4.4.2 Comparison experiment 1: Fixed output bond dimension

First, we compare the methods when the output bond dimension $\overline{\chi}$ is fixed by the user. We use the same problem setup as section 1, where we generate random a random MPS and MPO with bond dimensions $D = \chi = 50$ across $n = 100$ sites and uniformly random

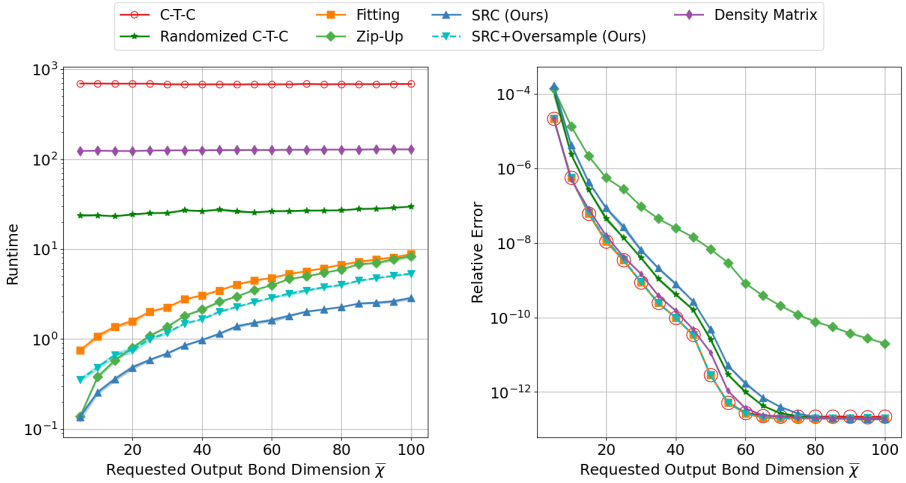


Figure 3: (**Fixed bond dimension**). Runtime (*left*) and relative error (*right*) of several MPO–MPS multiplication algorithms for a synthetic problem as a function of the output bond dimension $\overline{\chi}$, as described in section 4.4.2.

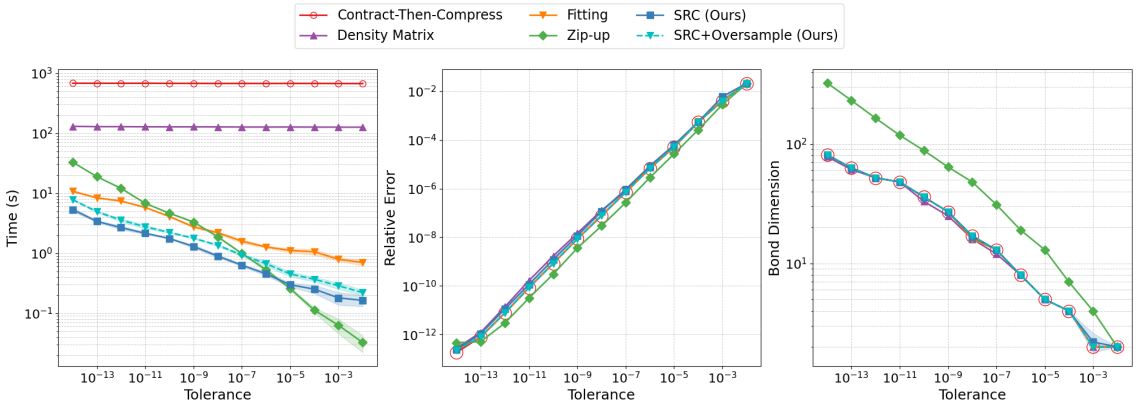


Figure 4: (**Accuracy threshold experiment**). Runtime (*left*), relative error (*center*), and final bond dimension (*right*) of several MPO–MPS multiplication algorithms for a synthetic problem as a specified approximation threshold.

entries $[\alpha, 1]$ with $\alpha = -0.5$. As in the introduction, we use complex data types and apply the contract-then-compress method with a machine-precision tolerance to compute a baseline.

A comparison of existing MPO–MPS methods and SRC is presented in Figure 3 for a range of requested bond dimensions $\overline{\chi}$. We implement SRC both as-is and with the oversampling strategy presented in section 3.4. For this problem, we see that SRC is the fastest method for all requested output bond dimensions $\overline{\chi}$ and, when run with oversampling, achieves comparable accuracy to contract-then-compress at a faster runtime than other accurate methods.

4.4.3 Comparison experiment 2: Adaptive determination of bond dimension

Next, we compare the the methods when the bond dimension is determined adaptively to meet a tolerance; see section C for how to implement SRC in this setting. The runtime behavior of the algorithms in this setting is somewhat changed, as the adaptive determination of bond dimension places a different level of overhead on each algorithm.

Figure 4 shows the results. For both the density matrix and contract-then-compress methods, the runtime behavior remains largely unaffected by the requested tolerance. SRC continues to be the fastest method in this setting, except at large values of the tolerance parameter where the zip-up method becomes fastest. This speed of zip-up comes with a cost, with the zip-up method resulting in meaningfully larger bond dimensions than other algorithms. Therefore, even for high values of the tolerance, SRC (with a final round) may be preferable to obtain an output of the smallest possible bond dimension, with the zip-up method being preferable for fast, approximate computations.

4.4.4 Summary of comparison

Both with a fixed bond dimension and with a tolerance, SRC achieves the greatest speed among the existing methods. When run with oversampling, it achieves comparable accuracy to the best method. We believe these experiments support the deployment of the randomized MPO–MPS multiplication as a method-of-choice in applications.

We conclude with two important caveats and limitations. First, as we already saw, SRC can be slower than the zip-up method when used with a loose tolerance, though the zip-up method results in higher bond dimensions. The zip-up method may still be the method of choice for coarse approximate MPO–MPS multiplication. Second, the speedups of SRC require having a problem with large MPS *and MPO* bond dimensions $D, \chi \gg 1$ and significant compression of the output bond dimension below its maximum value $\overline{\chi} \ll D\chi$. This makes SRC a powerful tool for physical problems with complicated or long-range interactions, but it may not be the best tool for simple Hamiltonians of bond dimension $D \leq 5$.

5 Application: Unitary time evolution

To demonstrate the effectiveness of the randomized contraction algorithm, we compare the performance of several compressed MPO–MPS multiplication algorithms for computing the unitary time evolution $|\psi(t)\rangle = \mathrm{e}^{-\mathrm{i}tH}|\psi(0)\rangle$ of an initial state $|\psi(0)\rangle$ under the influence of a Hamiltonian H. Section 5.1 describes the TDVP-GSE algorithm, a leading algorithm for time evolution that requires compressed MPO–MPS products, and section 5.2 provides a comparison of different MPO–MPS algorithms for use with this algorithm.

5.1 The time-dependent variational principle with global subspace expansion

The time-dependent variational principle (TDVP) algorithm [65] and its variants are currently among the best-performing methods for simulating time-evolution. In its basic form ("TDVP1"), TDVP simulates time evolution by updating individual site of the MPS in sequential left-to-right, right-to-left sweeps. This basic one-site procedure can have issues (similar to one-site DMRG), and even two-site versions of TDVP ("TDVP2", [66, App. 5]) can struggle on difficult examples. To address this issues, Yang and White [27] proposed an enhancement to TDVP by enriching the state at each time step with a set of global Krylov vectors. Given a current state $|\psi(t)\rangle$ and timestep δ, a single step of Yang and White's method proceeds as follows:

1. Generate the Krylov vectors

$$\mathcal{K} = \{|\psi(t)\rangle, (\mathrm{I} - \delta H)|\psi(t)\rangle, \ldots, (\mathrm{I} - \delta H)^{k-1}|\psi(t)\rangle\}.$$

   Here, k sets the number of the Krylov vectors, which Yang and White recommend setting to a small value, e.g., $k = 3$.

2. Perform *basis expansion* on the MPS $|\psi(t)\rangle$, increasing its bond rank by incorporating information from the Krylov space $\mathcal{K}$.

3. Perform a single step of TDVP1 with step size δ.

We refer to [27] for a justification and a description of the basis expansion step. Yang and White refer to their procedure as *global subspace expansion* TDVP (GSE-TDVP1). In our implementation of their method, we use an SVD-based version of the basis expansion procedure, which differs from but is equivalent to the eigendecomposition-based procedure presented in [27].

The GSE-TDVP1 method requires (compressed) MPO–MPS products to form the Krylov space $\mathcal{K}$, and these MPO–MPS products can take up a large fraction of the method's runtime depending on the algorithm used. To respond to this, Yang and White recommend using the density matrix algorithm or the fitting algorithm, depending on the bond dimension of H. In the example documented in next section, we found the density matrix algorithm was too slow and the fitting algorithm was not accurate enough, leading to an explosion of the bond dimension of $|\psi(t)\rangle$ as time evolution progressed. Responding to these issues, we propose the SRC method as an alternative for computing MPO–MPS products in GSE-TDVP1.

5.2 Comparison of MPO–MPS methods for GSE-TDVP1

To compare the performance of GSE-TDVP1 with different MPO–MPS methods, we emulate an experiment performed in [66]. This experiment considers an XY model with power-law interactions

$$H = \frac{1}{2} \sum_{1 \le i < j \le n} \frac{J}{|i-j|^\alpha} \left(X_i X_j + Y_i Y_j\right) \quad \text{with } \alpha = 1.5,\ n = 101.$$

Here, X_i and Y_i denote Pauli matrices acting on the ith site. We construct the initial state $|\psi(0)\rangle = \mathrm{e}^{\mathrm{i}(\pi/4)Y_{\lceil N/2 \rceil}} |\psi_{\min}\rangle$ by applying a local operation to the middle site of the ground state $|\psi_{\min}\rangle$. As demonstrated in previous studies, the difference in magnetization

$$\langle \psi(t) \,|\, X_i \,|\, \psi(t) \rangle - \langle \psi_{\min} \,|\, X_i \,|\, \psi_{\min} \rangle \tag{5.1}$$

between the ground state $|\psi_{\min}\rangle$ and the time-evolved perturbed state $|\psi(0)\rangle$ shows a light cone emanating from the middle cite $i = \lceil N/2 \rceil$ with power law leakage outside the cone.

To simulate this system, we represent H as an MPO using the linear-combination-of-exponentials method of [67, App] with tolerance 10^{-8}, and we compute the ground state $|\psi_{\min}\rangle$ using DMRG2 with tolerance 10^{-8}. The bond dimensions of H and $|\psi(0)\rangle$ are $D = 26$ and $\chi = 32$. Time evolution is performed with GSE-TDVP1 with a time step of $\delta = 0.2/J$ to a final time $tJ = 5$, and all MPO–MPS products $(\mathrm{I} - \delta H)|\phi\rangle$ are truncated to have the same bond dimension as $|\phi\rangle$. Our purpose in presenting this experiment is to compare several MPO–MPS methods for use in GSE-TDVP1, not to claim that the GSE-TDVP1 method is necessary to resolve this particular physical system. (Indeed, the qualitative features of this model can be resolved even using TDVP1.)

Figure 5 shows the results. The left panel shows that the magnetization difference (5.1) exhibits a light cone with power law leakage, in concurrence with the results of previous studies. The middle and right panels show the runtime with various MPO–MPS methods. (The fitting method yielded very poor accuracy and resulted in a rapid explosion in the bond dimension during the basis expansion step, so we omit it.) The SRC method is definitely the fastest, achieving speedups of up to 181× over contract-then-compress, 45× over density matrix, and 3.2× over zip-up for the Krylov vector construction.

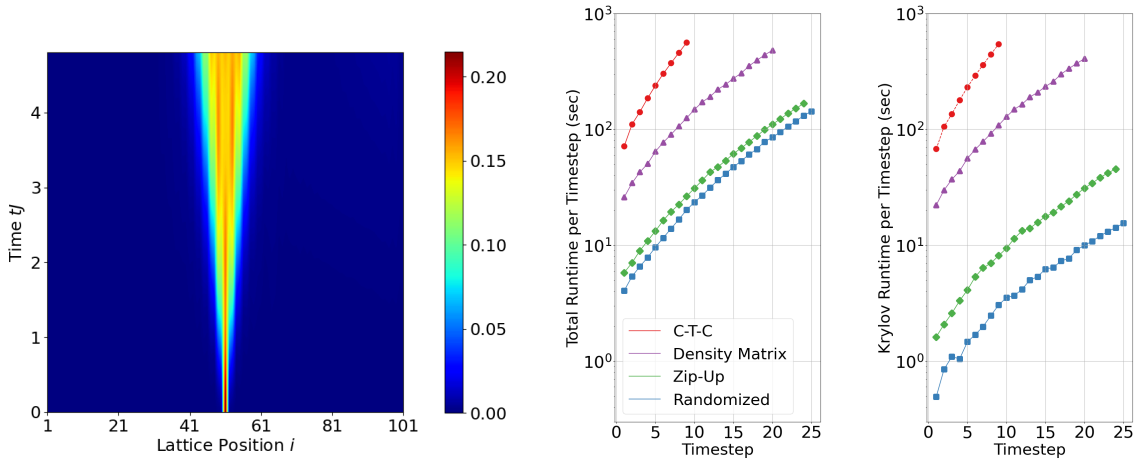


Figure 5: (**TDVP unitary time evolution**). *Left:* Magnetization difference (5.1) at each position $1 \leq i \leq 101$ and time $0 \leq tJ \leq 5$ computed by GSE-TDVP1 with randomized MPO–MPS multiplication, showing a light cone with power law leakage. *Middle and right:* Total runtime (*middle*) and runtime spent during Krylov vector construction (*right*) for different MPO–MPS methods per timestep.

Acknowledgements

We thank Jielun Chen, Garnet Chan, Johnnie Gray, Sandeep Sharma, and Robert Webber for helpful discussions.

References

[1] M. Fannes, B. Nachtergaele, and R. F. Werner. "Finitely correlated states on quantum spin chains". Communications in Mathematical Physics **144**, 443–490 (1992). (Cited on page 1.)

[2] F. Verstraete, J. J. García-Ripoll, and J. I. Cirac. "Matrix product density operators: Simulation of finite-temperature and dissipative systems". Phys. Rev. Lett. **93**, 207204 (2004). (Cited on pages 1 and 2.)

[3] Y.-Y. Shi, L.-M. Duan, and G. Vidal. "Classical simulation of quantum many-body systems with a tree tensor network". Physical Review A**74** (2006). (Cited on page 1.)

[4] F. Verstraete, M. M. Wolf, D. Perez-Garcia, and J. I. Cirac. "Criticality, the area law, and the computational power of projected entangled pair states". Physical Review Letters**96** (2006). (Cited on page 1.)

[5] Guifre Vidal. "Entanglement renormalization: an introduction" (2010). url: `https://arxiv.org/abs/0912.1651v2`. (Cited on page 1.)

[6] Jacob C. Bridgeman and Christopher T. Chubb. "Hand-waving and interpretive dance: An introductory course on tensor networks". Journal of Physics A: Mathematical and Theoretical **50**, 223001 (2017). (Cited on page 1.)

[7] G. Evenbly and G. Vidal. "Tensor network states and geometry". Journal of Statistical Physics **145**, 891918 (2011). (Cited on page 1.)

[8] Román Orús. "A practical introduction to tensor networks: Matrix product states and projected entangled pair states". Annals of Physics **349**, 117–158 (2014). (Cited on page 1.)

[9] Román Orús. "Tensor networks for complex quantum systems". Nature Reviews Physics **1**, 538–550 (2019). (Cited on page 1.)

[10] I. V. Oseledets. "Approximation of $2^d \times 2^d$ matrices using tensor decomposition". SIAM Journal on Matrix Analysis and Applications **31**, 2130–2145 (2010). (Cited on page 1.)

[11] Boris N. Khoromskij. "$O(d \log N)$-quantics approximation of N-d tensors in high-dimensional numerical modeling". Constructive Approximation **34**, 257–280 (2011). (Cited on page 1.)

[12] Michael Lindsey. "Multiscale interpolative construction of quantized tensor trains" (2024). url: https://arxiv.org/abs/2311.12554v3. (Cited on page 1.)

[13] Zhao-Yu Han, Jun Wang, Heng Fan, Lei Wang, and Pan Zhang. "Unsupervised generative modeling using matrix product states". Physical Review X**8** (2018). (Cited on page 1.)

[14] YoonHaeng Hur, Jeremy G. Hoskins, Michael Lindsey, E. M. Stoudenmire, and Yuehaw Khoo. "Generative modeling via tensor train sketching". Applied and Computational Harmonic Analysis **67**, 101575 (2023). (Cited on pages 1 and 6.)

[15] Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. "Tensorizing neural networks". In Proceedings of the 29th International Conference on Neural Information Processing Systems. Volume 1, pages 442–450. Cambridge, MA (2015). MIT Press. (Cited on page 1.)

[16] Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry Vetrov. "Ultimate tensorization: compressing convolutional and FC layers alike" (2016). url: https://arxiv.org/abs/1611.03214v1. (Cited on page 1.)

[17] Yinchong Yang, Denis Krompass, and Volker Tresp. "Tensor-train recurrent neural networks for video classification". In Proceedings of the 34th International Conference on Machine Learning. Pages 3891–3900. Sydney, NSW (2017). JMLR. (Cited on page 1.)

[18] Eva Memmel, Clara Menzen, Jetze Schuurmans, Frederiek Wesel, and Kim Batselier. "Position: Tensor networks are a valuable asset for green AI". In Proceedings of the 41st International Conference on Machine Learning. Volume 235, pages 35340–35353. Vienna, Austria (2024). (Cited on page 1.)

[19] Andrei Tomut, Saeed S. Jahromi, Sukhbinder Singh, Faysal Ishtiaq, Cesar Muoz, Prabdeep Singh Bajaj, Ali Elborady, Gianni del Bimbo, Mehrazin Alizadeh, David Montero, Pablo Martin-Ramiro, Muhammad Ibrahim, Oussama Tahiri Alaoui, John Malcolm, Samuel Mugel, and Roman Orus. "CompactifAI: Extreme compression of large language models using quantum-inspired tensor networks" (2024). url: arXiv:2401.14109v2. (Cited on page 1.)

[20] Jacob Biamonte and Ville Bergholm. "Tensor networks in a nutshell" (2017). url: https://arxiv.org/abs/1708.00006v1. (Cited on page 1.)

[21] Gregory M. Crosswhite, A. C. Doherty, and Guifr Vidal. "Applying matrix product operators to model systems with long-range interactions". Physical Review B**78** (2008). (Cited on page 2.)

[22] C. Hubig, I. P. McCulloch, and U. Schollwck. "Generic construction of efficient matrix product operators". Physical Review B**95** (2017). (Cited on page 2.)

[23] Sebastian Paeckel, Thomas Khler, Andreas Swoboda, Salvatore R. Manmana, Ulrich Schollwck, and Claudius Hubig. "Time-evolution methods for matrix-product states". Annals of Physics **411**, 167998 (2019). (Cited on pages 2, 15, 16, and 17.)

[24] Maarten Van Damme, Jutho Haegeman, Ian McCulloch, and Laurens Vanderstraeten. "Efficient higher-order matrix product operators for time evolution" (2023). url: https://arxiv.org/abs/2302.14181v1. (Cited on page 2.)

[25] Jielun Chen, E.M. Stoudenmire, and Steven R. White. "Quantum Fourier Transform Has Small Entanglement". PRX Quantum **4**, 040318 (2023). (Cited on page 2.)

[26] Jielun Chen and Michael Lindsey. "Direct interpolative construction of the discrete fourier transform as a matrix product operator" (2024). url: https://arxiv.org/abs/2404.03182v1. (Cited on page 2.)

[27] Mingru Yang and Steven R. White. "Time-dependent variational principle with ancillary Krylov subspace". Physical Review B**102** (2020). (Cited on pages 2, 20, and 21.)

[28] F. Verstraete and J. I. Cirac. "Renormalization algorithms for Quantum-Many Body Systems in two and higher dimensions" (2004). url: https://arxiv.org/abs/cond-mat/0407066v1. (Cited on pages 2 and 16.)

[29] J. Jordan, R. Orús, G. Vidal, F. Verstraete, and J. I. Cirac. "Classical simulation of infinite-size quantum lattice systems in two spatial dimensions". Phys. Rev. Lett. **101**, 250602 (2008). (Cited on page 2.)

[30] Paula García-Molina, Luca Tagliacozzo, and Juan José García-Ripoll. "Global optimization of MPS in quantum-inspired numerical analysis" (2024). url: https://arxiv.org/abs/2303.09430v2. (Cited on page 2.)

[31] Johnnie Gray and Garnet Kin-Lic Chan. "Hyperoptimized approximate contraction of tensor networks with arbitrary geometry". Physical Review X **14**, 011009 (2024). (Cited on page 2.)

[32] Jiaqing Jiang, Jielun Chen, Norbert Schuch, and Dominik Hangleiter. "Positive bias makes tensor-network contraction tractable". Foundation of Computer Science 2025, to appear (2024). (Cited on page 2.)

[33] Jielun Chen, Jiaqing Jiang, Dominik Hangleiter, and Norbert Schuch. "Sign problem in tensor-network contraction". PRX Quantum **6**, 010312 (2025). (Cited on page 2.)

[34] I. V. Oseledets. "Tensor-train decomposition". SIAM Journal on Scientific Computing **33**, 2295–2317 (2011). (Cited on pages 3 and 15.)

[35] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions". SIAM Review **53**, 217–288 (2011). (Cited on pages 4 and 5.)

[36] David P. Woodruff. "Sketching as a tool for numerical linear algebra". Foundations and Trends in Theoretical Computer Science **10**, 1–157 (2014). (Cited on page 4.)

[37] Joel A. Tropp and Robert J. Webber. "Randomized algorithms for low-rank matrix approximation: Design, analysis, and applications" (2023). url: https://arxiv.org/abs/2306.12418v1. (Cited on pages 4 and 5.)

[38] Riley Murray, James Demmel, Michael W. Mahoney, N. Benjamin Erichson, Maksim Melnichenko, Osman Asif Malik, Laura Grigori, Piotr Luszczek, Michał Dereziński, Miles E. Lopes, Tianyu Liang, Hengrui Luo, and Jack Dongarra. "Randomized numerical linear algebra : A perspective on the field with an eye to software" (2023). url: https://arxiv.org/abs/2302.11474v2. (Cited on pages 4 and 5.)

[39] Joel A. Tropp, Alp Yurtsever, Madeleine Udell, and Volkan Cevher. "Practical sketching algorithms for low-rank matrix approximation". SIAM Journal on Matrix Analysis and Applications **38**, 1454–1485 (2017). (Cited on page 4.)

[40] Per-Gunnar Martinsson and Joel A. Tropp. "Randomized numerical linear algebra: Foundations and algorithms". Acta Numerica **29**, 403–572 (2020). (Cited on pages 5 and 30.)

[41] Chris Camaño, Ethan N. Epperly, Raphael A. Meyer, and Joel A. Tropp. "Faster linear algebra algorithms with structured random matrices" (2025). arXiv:2508.21189. (Cited on pages 5 and 6.)

[42] Ming Gu. "Subspace Iteration Randomization and Singular Value Problems". SIAM Journal on Scientific Computing **37**, A1139–A1173 (2015). (Cited on page 5.)

[43] Cameron Musco and Christopher Musco. "Randomized block Krylov methods for stronger and faster approximate singular value decomposition". In Proceedings of the 28th International Conference on Neural Information Processing Systems. Pages 1396–1404. MIT Press (2015). url: https://dl.acm.org/doi/10.5555/2969239.2969395. (Cited on page 5.)

[44] Nicolas Boullé, Diana Halikias, Samuel E. Otto, and Alex Townsend. "Operator learning without the adjoint". Journal of Machine Learning Research **25**, 1–54 (2024). url: https://jmlr.org/papers/v25/24-0162.html. (Cited on page 5.)

[45] D. Tamascelli, R. Rosenbach, and M. B. Plenio. "Improved scaling of time-evolving block-decimation algorithm through reduced-rank randomized singular value decomposition". Physical Review E **91**, 063306 (2015). (Cited on page 5.)

[46] Satoshi Morita, Ryo Igarashi, Hui-Hai Zhao, and Naoki Kawashima. "Tensor renormalization group with randomized singular value decomposition". Physical Review E **97**, 033310 (2018). (Cited on page 5.)

[47] Lucas Kohn, Ferdinand Tschirsich, Maximilian Keck, Martin B. Plenio, Dario Tamascelli, and Simone Montangero. "Probabilistic low-rank factorization accelerates tensor network simulations of critical quantum many-body ground states". Physical Review E **97**, 013301 (2018). (Cited on page 5.)

[48] Ian P. McCulloch and Jesse J. Osborne. "Comment on "Controlled Bond Expansion for Density Matrix Renormalization Group Ground State Search at Single-Site Costs" (Extended Version)" (2024). arXiv:2403.00562. (Cited on pages 5 and 17.)

[49] Tamara G. Kolda and Brett W. Bader. "Tensor decompositions and applications". SIAM Review **51**, 455–500 (2009). (Cited on page 6.)

[50] Thomas D. Ahle. "The tensor cookbook". Manuscript in progress, Version: February 2025. (2025). url: https://tensorcookbook.com. (Cited on page 6.)

[51] Hussam Al Daas, Grey Ballard, Paul Cazeaux, Eric Hallman, Agnieszka Międlar, Mirjeta Pasha, Tim W. Reid, and Arvind K. Saibaba. "Randomized algorithms for rounding in the tensor-train format". SIAM Journal on Scientific Computing **45**, A74–A95 (2023). (Cited on pages 6, 14, and 15.)

[52] Daniel Kressner, Bart Vandereycken, and Rik Voorhaar. "Streaming tensor train approximation". SIAM Journal on Scientific Computing **45**, A2610–A2631 (2023). (Cited on page 6.)

[53] Yian Chen and Yuehaw Khoo. "Combining Monte Carlo and Tensor-network Methods for Partial Differential Equations via Sketching" (2023). arXiv:2305.17884. (Cited on page 6.)

[54] Ziang Yu, Shiwei Zhang, and Yuehaw Khoo. "Re-anchoring Quantum Monte Carlo with Tensor-Train Sketching" (2025) math:2411.07194. (Cited on page 6.)

[55] Sukhwinder Singh, Robert N. C. Pfeifer, and Guifré Vidal. "Tensor network decompositions in the presence of a global symmetry". Physical Review A **82**, 050301 (2010). (Cited on page 14.)

[56] Sukhwinder Singh, Robert N. C. Pfeifer, and Guifre Vidal. "Tensor network states and algorithms in the presence of a global U(1) symmetry". Physical Review B **83**, 115125 (2011). (Cited on page 14.)

[57] Ulrich Schollwöck. "The density-matrix renormalization group in the age of matrix product states". Annals of Physics **326**, 96–192 (2011). (Cited on page 15.)

[58] E M Stoudenmire and Steven R White. "Minimally entangled typical thermal state algorithms". New Journal of Physics **12**, 055026 (2010). (Cited on pages 16 and 17.)

[59] Steven R White. "Density matrix renormalization group algorithms with a single center site". Physical Review B—Condensed Matter and Materials Physics **72**, 180403 (2005). (Cited on page 17.)

[60] Claudius Hubig, Ian P McCulloch, Ulrich Schollwöck, and F Alexander Wolf. "Strictly single-site dmrg algorithm with subspace expansion". Physical Review B **91**, 155115 (2015). (Cited on page 17.)

[61] Markus Hauru, Maarten Van Damme, and Jutho Haegeman. "Riemannian optimization of isometric tensor networks". SciPost Physics **10**, 040 (2021). (Cited on page 17.)

[62] Linjian Ma, Matthew Fishman, Edwin Miles Stoudenmire, and Edgar Solomonik. "Approximate contraction of arbitrary tensor networks with a flexible and efficient density matrix algorithm". Quantum **8**, 1580 (2024). (Cited on page 17.)

[63] "MPO-MPS multiplication: Density matrix algorithm". Web page: `https://www.tensornetwork.org/mps/algorithms/denmat_mpo_mps/`. (Cited on page 17.)

[64] Linjian Ma, Matthew Fishman, Edwin Miles Stoudenmire, and Edgar Solomonik. "Approximate contraction of arbitrary tensor networks with a flexible and efficient density matrix algorithm". Quantum **8**, 1580 (2024). (Cited on page 17.)

[65] Jutho Haegeman, J. Ignacio Cirac, Tobias J. Osborne, Iztok Piorn, Henri Verschelde, and Frank Verstraete. "Time-dependent variational principle for quantum lattices". Physical Review Letters**107** (2011). (Cited on page 20.)

[66] Jutho Haegeman, Christian Lubich, Ivan Oseledets, Bart Vandereycken, and Frank Verstraete. "Unifying time evolution and optimization with matrix product states". Physical Review B**94** (2016). (Cited on pages 20 and 21.)

[67] B Pirvu, V Murg, J I Cirac, and F Verstraete. "Matrix product operator representations". New Journal of Physics **12**, 025012 (2010). (Cited on page 21.)

[68] Stanislaw Lojasiewicz. "Triangulation of semi-analytic sets". Annali della Scuola Normale Superiore di Pisa-Classe di Scienze **18**, 449–474 (1964). url: `http://www.numdam.org/item/ASNSP_1964_3_18_4_449_0/`. (Cited on page 27.)

[69] Nate Eldredge. "The Lebesgue measure of zero set of a polynomial function is zero". Math StackExchange post (2016). url: https://math.stackexchange.com/q/1920527. (Cited on page 27.)

[70] Ethan N. Epperly and Joel A. Tropp. "Efficient Error and Variance Estimation for Randomized Matrix Computations". SIAM Journal on Scientific Computing **46**, A508–A528 (2024). (Cited on pages 28 and 29.)

A Proof of Theorem 2

We may assume without loss of generality that $p = \operatorname{rank} A$. The column space of the matrix Y is a subset of the column space of A, and randomized QB approximation is exact precisely when the column space of these two matrices is the same. Thus, randomized QB approximation is exact if and only if $\operatorname{rank} Y = p$, which occurs if and only if $\det(Y^\dagger Y) \neq 0$, so that $Y^\dagger Y$ is nonsingular.

Since $Y = A\Omega$ and $\Omega = \Omega^{(1)} \odot \cdots \odot \Omega^{(n)}$, $p(\Omega^{(1)}, \ldots, \Omega^{(n)}) = \det(Y^\dagger Y)$ is a polynomial in the entries of $\Omega^{(1)}, \ldots, \Omega^{(n)}$. The set of zeros of a nonzero multivariate polynomial is a null set. (See [68] for a sophisticated generalization of this fact; an elementary argument is provided in [69].) Therefore, provided $p(\cdot)$ is not identitically zero as a polynomial, $p(\Omega^{(1)}, \ldots, \Omega^{(n)}) = \det(Y^\dagger Y) \neq 0$ with probability one.

To complete the proof, we must furnish matrices $\Omega^{(1)}, \ldots, \Omega^{(n)}$ for which

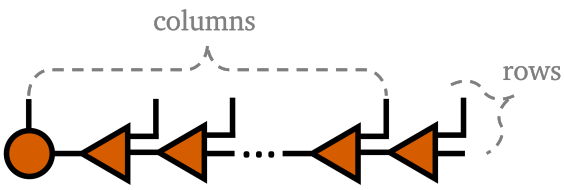
$$p(\Omega^{(1)}, \ldots, \Omega^{(n)}) \neq 0.$$

Since A is rank-p, A contains p linearly independent columns $A(:, i_1), \ldots, A(:, i_p)$. The matrix $\Omega = \begin{bmatrix} e^{(i_1)} & \cdots & e^{(i_p)} \end{bmatrix}$ can be realized as a Khatri–Rao product $\Omega = \Omega^{(1)} \odot \cdots \odot \Omega^{(n)}$, where $e^{(i)}$ denotes the standard basis vector with entries $e^{(i)}_j = \delta_{ij}$. Thus, $p(\Omega^{(1)}, \ldots, \Omega^{(n)}) = \det(A(:, \{i_1, \ldots, i_p\})^\dagger A(:, \{i_1, \ldots, i_p\})) > 0$, completing the proof. ■

B Proof of Theorem 3

At first glance, it might seem that Theorem 3 is a trivial consequence of Theorem 2, which guarantees rank-r QB decomposition is exact with probability one when applied to a matrix of rank at most r. The weakness in this argument is that SRC uses a common set of matrices $\Omega^{(1)}, \ldots, \Omega^{(n-1)}$ throughout the algorithm, and the matrix which we applied QB decomposition at each step depends on previous steps of the algorithm. The dependencies between the steps of the algorithm could be a potential source of trouble.

Fortunately, this potential concern is not a worry. At each step of the SRC algorithm, we apply QB decomposition to a matrix representation $B^{(\ell)}$ of the first ℓ sites of the MPO–MPS product

$B^{(\ell)} =$ 

$\in \mathbb{C}^{\overline{\chi} d \times d^{\ell-1}}$.

The right canonical form of an MPS is unique up to unitary transformations along the bond indices. As such, the $B^{(\ell)}$ matrix that we apply QB decomposition at each step of

the algorithm is fixed independent of the randomness in $\Omega^{(1)}, \ldots, \Omega^{(n-1)}$, up to a potential unitary transformation on the left. The randomized QB algorithm is unitarily covariant (i.e., if we apply the algorithm to both A and a unitary transformation $U \cdot A$ with the same random test matrix, the outputs will be $\widehat{A}$ and $U \cdot \widehat{A}$). Therefore, the randomized QB decompositions are exact at each step of the algorithm with probability one, so SRC outputs an exact MPS representation of $|\eta\rangle = H|\psi\rangle$. ■

C Implementing SRC with a tolerance

In section 3, we presented a version of SRC where the output bond dimension $\overline{\chi}$ is provided as an input to the algorithm. This section will describe an adaptive implementation of SRC that chooses the bond dimension adaptively to meet a tolerance. Section C.1 discusses error estimation, section C.2 uses the error estimate to implement SRC adaptively, and section C.3 optimizes the adaptive implementation using QR factorization updating algorithms.

C.1 Error estimation

Each step of the randomized MPO–MPS multiplication algorithm involves a randomized QB approximation of the form

$$B^{(j)} \approx \eta^{(j)}(\eta^{(j)})^\dagger B^{(j)}.$$

Even the first step of the algorithm has this form, with $B^{(n)} := H|\psi\rangle$. When $H|\psi\rangle$ is exactly representable as an MPS of the specified bond dimension $\overline{\chi}$, this randomized QB approximation is exact with probability one, but this is typically not the case in practice. Thus, in order to choose the appropriate bond dimension $\overline{\chi}$, it is important to understand the error of the approximation

$$\left\| B^{(j)} - \eta^{(j)}(\eta^{(j)})^\dagger B^{(j)} \right\|_{\mathrm{F}}.$$

For this purpose, we will employ the leave-one-out error estimator [70].

C.1.1 Leave-one-out error estimation: Matrix case

Let us first introduce the leave-one-out error estimator in the matrix context. To compute a QB approximation of a matrix $A \in \mathbb{C}^{M \times N}$, we draw a Gaussian or Khatri–Rao random matrix $\Omega \in \mathbb{C}^{N \times p}$, apply A to Ω, and compute a QR factorization

$$A\Omega = QR.$$

The QB approximation of rank p then takes the form

$$\widehat{A} = QQ^\dagger A. \tag{C.1}$$

The root mean-square error of the approximation is

$$\mathrm{Err}_p := \left(\mathbb{E} \left[\|A - \widehat{A}\|_{\mathrm{F}}^2 \right] \right)^{1/2}.$$

We write a subscript p to indicate the rank of the approximation $p = \operatorname{rank} \widehat{A}$.

Surprisingly, the QR factorization (C.1) already contains enough information to compute an estimate of the error Err_p. First, compute the inverse-adjoint of the matrix R:

$$G \coloneqq R^{-\dagger} = \begin{bmatrix} g_1 & \cdots & g_p \end{bmatrix}.$$

The *leave-one-out estimate for the root-mean-square error* is

$$\widehat{\mathrm{Err}} \coloneqq \left(\frac{1}{p} \sum_{i=1}^{p} \|g_i\|^{-2} \right)^{1/2}.$$

For an explanation of the name "leave-one-out estimate", intuition for why $\widehat{\mathrm{Err}} \approx \mathrm{Err}_p$, and empirical evaluation of this estimator for matrix QB approximation, see [70]. The leave-one-out mean-square error estimate satisfies the following guarantee:

Theorem 4 (Leave-one-out error estimation). *Let $\Omega \in \mathbb{C}^{N \times p}$ be a random matrix with independent columns $\omega_1, \ldots, \omega_p$ satisfying the isotropy condition $\mathbb{E}[\omega_i \omega_i^\dagger] = \mathrm{I}$, such as a standard Gaussian matrix or Khatri–Rao product of standard Gaussian matrices. Then, with the prevailing notation,*

$$\mathbb{E}\left[\widehat{\mathrm{Err}}^2\right] = \mathrm{Err}_{p-1}^2 \geq \mathrm{Err}_p^2.$$

That is, $\widehat{\mathrm{Err}}^2$ is a statistically unbiased estimate for the mean-square error Err_{p-1}^2 of the rank-$(p-1)$ randomized QB approximation. Consequently, $\widehat{\mathrm{Err}}^2$ is an overestimate for Err_p^2, on average.

The leave-one-out error estimate is useful because it provides a quick estimate of the error $\|A - \widehat{A}\|_{\mathrm{F}}$ of randomized QB approximation. Indeed, computing this estimate just requires inverting the small $p \times p$ matrix $R^\dagger$ and computing the norms of its columns. The leave-one-out error estimate only costs a measly $\mathcal{O}(p^3)$ operations to compute, dwarfed by the $\Omega(Mp^2)$ cost of computing and orthogonalizing $A\Omega$. The estimator is slightly conservative, typically overestimating MSE_p by a small amount on average.

C.1.2 Leave-one-out error estimation: SRC algorithm

The leave-one-out error estimate can directly be used in the randomized MPO–MPS multiplication algorithm to estimate the error at each step of the algorithm. Simply compute the inverse-adjoint of $R^{(j)}$ and compute the leave-one-out mean-square error estimate:

$$G^{(j)} = \left(R^{(j)}\right)^{-\dagger} = \begin{bmatrix} g_1^{(j)} & \cdots & g_{\overline{\chi}}^{(j)} \end{bmatrix}, \quad \widehat{\mathrm{Err}}^{(j)} \coloneqq \left(\frac{1}{\overline{\chi}} \sum_{i=1}^{\overline{\chi}} \left\| g_i^{(j)} \right\|^{-2} \right)^{1/2}. \tag{C.2}$$

This serves as a drop-in estimate for the error incurred in the current step of the algorithm:

$$\widehat{\mathrm{Err}}^{(j)} \approx \left\| B^{(j)} - \eta^{(j)} \cdot B^{(j-1)} \right\|_{\mathrm{F}}.$$

C.1.3 Norm estimation

In order to implement MPO–MPS truncation adaptively, we will also need access to an estimate of the *norm* $\left\| B^{(j)} \right\|_{\mathrm{F}}$. This can be achieved by the use of another stochastic estimate

$$\widehat{\mathrm{Norm}} \coloneqq \frac{1}{\sqrt{\overline{\chi}}} \left\| Y^{(j)} \right\|_{\mathrm{F}} = \frac{1}{\sqrt{\overline{\chi}}} \left\| R^{(j)} \right\|_{\mathrm{F}}. \tag{C.3}$$

This is an instance of the Girard–Hutchinson norm estimator, closely related to the Girard–Hutchinson trace estimator [40, §4]. In particular, we have the following standard guarantee [40, §4.8]:

Proposition 5 (Norm estimation). *Under the hypotheses of Theorem 4, the squared norm estimate* $\widehat{\mathrm{Norm}}^2$ *is a statistically unbiased estimate for* $\left\|B^{(j)}\right\|_{\mathrm{F}}^2$. *That is,*

$$\mathbb{E}\left[\widehat{\mathrm{Norm}}^2\right] = \left\|B^{(j)}\right\|_{\mathrm{F}}^2.$$

Moreover, the variance of the estimator $\widehat{\mathrm{Norm}}^2$ *decays at a rate* $\mathcal{O}(1/\overline{\chi})$.

C.2 Adaptive determination of bond dimension

Using the leave-one-out error estimate (C.2) and norm estimate (C.3), we can adaptively choose the dimension $\overline{\chi}$ adaptively for each bond in the output MPS using a tolerance. Let τ_{abs} and τ_{rel} be absolute and relative tolerances, and let $\overline{\chi}_0$ and Δ_χ be a user-specified minimum bond dimension and increment. Begin each step j of SRC using the minimum bond dimension $\overline{\chi}_0$, obtaining a jth site tensor $\eta^{(j)}$. Then, do the following steps:

1. **Error estimation.** Compute the error estimate $\widehat{\mathrm{Err}}^{(j)}$ and the norm estimate $\widehat{\mathrm{Norm}}$.
2. **Check tolerance.** Check whether $\widehat{\mathrm{Err}}^{(j)} \leq \tau_{\mathrm{abs}} + \tau_{\mathrm{rel}} \cdot \widehat{\mathrm{Norm}}$. If yes, then exist this loop and continue the algorithm; otherwise, proceed to step 3.
3. **Increase bond dimension.** Increase $\overline{\chi}$ by the increment Δ_χ.
4. **Update random matrix and intermediate tensors.** If $\overline{\chi}$ is larger than the number of columns in $\Omega^{(1:j-1)}$, then append columns to $\Omega^{(1:j-1)}$ until it has $\overline{\chi}$ in total. Update the intermediate tensors $C^{(1)}, \ldots, C^{(j-1)}$.
5. **Update randomized QB approximation.** Compute the additional columns Δ_χ of $Y^{(j)}$ using the (possibly updated) intermediate tensor $C^{(j-1)}$. Recompute (or update, see section C.3) the QR factorization of $Y^{(j)}$. Go to step 1.

When implemented using QR updating rather recomputation from scratch, as described in the next section, this adaptive version of the MPO–MPS multiplication algorithm has the same time complexity as reported in section 3.5. In our experiments in section 4.4.3, we use parameters $\overline{\chi}_0 = 2$, $\Delta_\chi = 3$, and use only a relative tolerance (setting $\tau_{\mathrm{abs}} = 0$). To implement with oversampling, we set the relative tolerance to be 0.1 times the requested tolerance and run a final truncation with the requested tolerance.

C.3 Final optimization: Updating the QR factorization

In the adaptive MPO–MPS multiplication algorithm (section C.2), we continue to accumulate "columns" in the 2- or 3-tensor $Y^{(j)}$ until a tolerance is satisfied. Recomputing a QR factorization each time we do this is computationally expensive, and we can make the algorithm more efficient by instead updating the QR factorization to include new columns. This material is standard but perhaps not well-known, so we provide a brief discussion.

The simplest—but not most numerically stable—way of updating the QR decomposition is through the block Gram–Schmidt process. To simplify notation, drop the superscript

Table 2: Operation counts for methods to apply an MPO (bond dimension D) to an MPS (bond dimension χ), both of n sites and physical dimension d, and truncate the output to bond dimension $\overline{\chi}$.

Method	Operation Count
Contract-then-compress	
• Basic	$\mathcal{O}(n \cdot dD^2\chi^2[D\chi + d])$
• Randomized	$\mathcal{O}(n \cdot dD^2\chi^2[d + \overline{\chi}])$
Density Matrix	$\mathcal{O}(n \cdot [dD\chi(D\chi^2 + dD^2\chi + D\chi\overline{\chi} + d\overline{\chi}^2) + d^3\overline{\chi}^3])$
Fitting	$\mathcal{O}\left(n \cdot [dD\chi\overline{\chi}^2 + d^2D^2\chi^2]\right)$ per iteration
Zip-up	$\mathcal{O}(n \cdot d^2D\chi\overline{\chi}[D + \overline{\chi}])$
SRC	$\mathcal{O}(n \cdot dD\chi\overline{\chi}(\chi + \overline{\chi} + dD))$

$Y = Y^{(j)}$ and let Y' be a set of new columns being appended to $Y = QR$. To update the QR factorization, first orthogonalize the new columns Y' against Q:

$$Z := Y - QR', \quad R' = Q^\dagger Y'.$$

Then, compute a QR factorization $Z = Q'R''$. This results in a QR factorization of the full matrix Y with the appended columns

$$\begin{bmatrix} Y & Y' \end{bmatrix} = \begin{bmatrix} Q & Q' \end{bmatrix} \begin{bmatrix} R & R' \\ 0 & R'' \end{bmatrix}.$$

We have suceeded at updating the QR factorization with the addition of new columns. A similar procedure based on Householder QR factorization is more numerically stable and is preferable. We use the Householder-based procedure in our code.

Another matrix that must be updated to implement the adaptive MPO–MPS multiplication algorithm efficiently is the matrix G used in error estimation (C.2). Again using our superscript-less notation, we begin with $G = R^{-\dagger}$ and with to update G to

$$G' = \begin{bmatrix} R & R' \\ 0 & R'' \end{bmatrix}^{-\dagger}.$$

To do so, invoke the standard formula for the inverse of a block two-by-two matrix, obtaining

$$G' = \begin{bmatrix} R^{-\dagger} & 0 \\ -(R'')^{-\dagger}(R')^\dagger R^{-\dagger} & (R'')^{-\dagger} \end{bmatrix} = \begin{bmatrix} G & 0 \\ -(R'')^{-\dagger}(R')^\dagger G & (R'')^{-\dagger} \end{bmatrix}.$$

We recognize the original G matrix in the top left corner of this matrix. To finish the formula for G', we just need to compute $(R'')^\dagger$ and form the matrix triple product $-((R'')^{-\dagger}(R')^\dagger)G$. Since R'' is a small $\Delta_\chi \times \Delta_\chi$ matrix, this is efficient to do.

D Full operation counts

Full operation counts for the methods in terms of all five parameters n, d, χ, D, and $\overline{\chi}$ are presented in Table 2. For simplified operation counts, see Table 1.