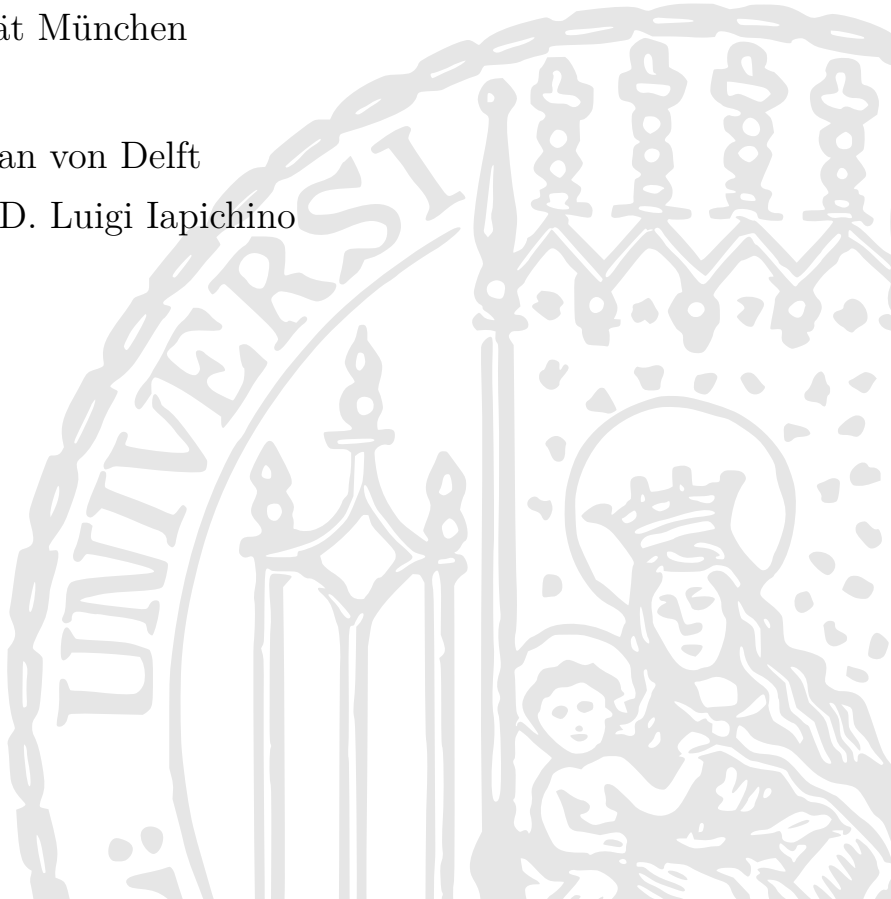
Efficient Tensor Compression through Adaptive Patched Quantics Tensor Cross Interpolation

Gianluca Grosso

M.Sc. Quantum Science & Technology,
Ludwig-Maximilians-Universität München,
Technische Universität München

Supervisor: Prof. Jan von Delft
Co-Supervisor: PhD. Luigi Iapichino

June 2, 2025, Munich



Effiziente Tensorkompression durch Quantics-Tensorkreuzinterpolation mit adaptivem Patching

Gianluca Grosso

M.Sc. Quantum Science & Technology,
Ludwig-Maximilians-Universität München,
Technische Universität München

Betreuer: Prof. Jan von Delft
Zweitbetreuer: PhD. Luigi Iapichino

June 2, 2025, Munich

To my friends and family

Thesis advisor: Prof. Jan von Delft *Gianluca Grosso*

Efficient Tensor Compression through Adaptive Patched Quantics Tensor Cross Interpolation

ABSTRACT

We present a *divide–et–impera* extension of the (Quantics) Tensor Cross Interpolation algorithm [1] that adaptively partitions a high-dimensional tensor into a collection of low-rank TT *patches*. Each patch is compressed with an explicit bond-dimension cap χ_{patch}, that triggers finer partitioning of the configuration space wherever the input tensor has more interesting features (higher local rank). The local cap χ_{patch} not only reduces the memory footprint of tensor-train representation of functions with sharply local features, but also tames the $\mathcal{O}(\chi^4)$ cost of MPO-MPO contractions by decomposing the global product into many rank-$\leq \chi_{\text{patch}}$ sub-contractions; in this context, the choice of MPO patching scheme is essential, as it can markedly enhances—or, if poorly chosen, limits—the overall efficiency of patched contractions.

We derive closed-form bounds that relate χ_{patch} and the patch count N_{patch} to the memory and run-time advantage over a monolithic TCI or MPO contraction, and identify an "over-patching" regime that arises if the cap is chosen too small. The theoretical estimates are validated by comprehensive benchmarks and the advantage is tested on three notorious bottlenecks of many-body physics related to the Hubbard model: (i) the approximaton of two-dimensional Matsubara Green's function, (ii) the computation of the bare susceptibility $\chi_0(\mathbf{q}, i\omega)$ (*bubble* diagram), and (iii) vertex contractions entering the Bethe-Salpeter equation for the single-impurity Anderson model. In all cases the patched strategy yields significant memory savings together with speed-ups of nearly an order of magnitude, enabling computations that remain out of practical reach for the monolithic method.

Acknowledgements

First and foremost, I am deeply grateful to Jan von Delft for introducing me to the realm of tensor networks and allowing me to pursue this project. His many invitations to present my work highlighted its value, kept me motivated, and taught me the importance of scientific collaboration. From him I have also learned the power of clear communication and teamwork in research.

My heartfelt thanks go to my supervisor, Marc Ritter, whose unwavering support and guidance sustained me throughout this project. Our discussions were always illuminating; through them he showed me what it truly means to be a researcher and provided inspiration that will guide me well into the future. I greatly appreciate his patience and reliability, which allowed me to explore the topic independently while never feeling adrift.

I am indebted to the entire tensor4all team—beginning with Hiroshi Shinaoka, without whom this project would not have existed. Close collaboration with him taught me a great deal about developing computational methods. I am equally grateful to Anna Kauch, Markus Frankenbach, Markus Wallerberger, Samuel Badr, Simone Foderà, and Stefan Rohshap for their insightful conversations and generous assistance.

I also thank everyone at the chair for fostering such a supportive work environment. In particular, Anxiang Ge offered invaluable help during the early stages of this endeavor.

Finally, I extend my deepest gratitude to my friends and family. Their unconditional support has been essential to reaching this milestone.

Contents

Chapter 1

Introduction

"Dispelling" the *curse of dimensionality* that "hexes" numerics' computations, has been a challenge of prime interest in different fields of science for many years. Tensor network techniques – and in particular *matrix product states* (MPSs) methodologies – are a well-established [2–8] and standard approach widely employed by the quantum many-body (QMB) physics community to mitigate such exponential increase of computational resources. Numerical simulations of high-dimensional QMB wavefunctions represent, in fact, an exceptionally challenging computational task if not properly addressed.

Mathematicians have dedicated, as well, significant effort to target the same problem – arising from multidimensional tensor approximation of continuous functions or complex numerical linear algebra computations [9, 10] – ultimately converging towards a similar approach: *tensor trains* (TTs)[1] [11].

The rising interest in tensor-network (TN) methods – particularly those based on MPSs — has therefore led to the emergence of a standard "MPS toolbox" [12–14] and a well-defined catalogue of canonical TN applications [15] (variational ansatz of QMB wavefunctions – or DMRG – among the most popular in physics). For a long time, however, techniques for function approximation and manipulation have been beyond the traditional TN portfolio.

Classical numerical techniques for representing and manipulating functions–whether for integration, convolution, differentiation, or related tasks–have advanced considerably over the years [16], yet they remain hindered by significant constraints. Grid-based or naive SVD–style tensor discretizations of multivariate functions, for instance, confront the *curse of dimensionality*: the memory and CPU time required to store and update a high-resolution representation of a $\mathcal{N}$-dimensional function grow exponentially with $\mathcal{N}$. When it comes to integration, standard stochastic approaches such as Monte Carlo and Quantum Monte Carlo fare no better in the high-dimensional regime; their error decreases only algebraically with sample size and they are plagued by additional obstacles—most notably the "sign problem" [17]—that can render simulations impractical for large, strongly correlated systems [18]. Together, these limitations leave many modern, high-dimensional applications beyond the reach of standard numerical methods.

The widespread and increasing interest in TN and MPSs methodologies

[1] From now on *tensor train* and *matrix product state* will be used interchangeably.

among different fields has facilitated the development of a pivotal[2] extension to the "MPS toolbox" in order to integrate the missing function representation capabilities: the Tensor Cross Interpolation (TCI) algorithm.

TCI attempts to address many of the above-mentioned limitations of standard function approximation algorithms, by revealing low-rank structures and leveraging weakly entangled, scale-separated MPS-based function representations. This effort converts otherwise exponential memory and CPU requirements into polynomial ones. The traditional "TT-toolset" already allowed for a similar scaling in resources with truncation-based procedures to reduce tensor's rank, however TCI progressively reveals the rank structure of the input tensor by adaptively increasing the number of tensor evaluations (more on this in Chap. 2) without any loss of information caused by truncation. For this reason TCI can be viewed as an *active–learning* algorithm in the sense of Ref. [19]: it probes the input tensor only at those configurations that most efficiently expose its low-rank structure, thereby minimising the number of function evaluations needed to reach a prescribed approximation accuracy.

Furthermore, the novel approach of Ref. [1, 18] to integrate *quantics* tensor rebasing with *tensor cross approximation* procedures – i.e. Quantics Tensor Cross Interpolation (QTCI) – opened up the possibility to features like *super-high resolution* and *sign problem-free integration* for multi-dimensional functions, while maintaining computational costs still bounded to a polynomial increase. The class of multivariate continuous functions that admit a low-rank tensor representation within a tolerance ε—the so-called ε*-factorisable* functions [18]—appears to be very large, although a rigorous characterisation is still lacking. Nevertheless, in practice this broadness translates into a wide domain of applicability for QTCI.

Among the many applications of QTCI the following are definitely worth mentioning (with their respective achievements): high-order real-time nonequilibrium Schwinger-Keldysh perturbation expansions [18] (integral convergence improved from $1/\sqrt{N_{\text{func\_eval}}}$ to $1/N^2_{\text{func\_eval}}$), multi-dimensional function minimization and quantized reinforcement learning [20] (outperfoming standard gradient-free methods in number of function evaluations and execution time), computation of Brillouin zone integrals for topological invariants evaluation [21] (exponential to polynomial-order scaling of integration costs with respect to the simulation parameters), compact tensorization of atomic orbitals bases with high accuracy [22] (error on g.s. energy of H_2 improved by 85% w.r.t. double zeta calculation), (speed-up of) multi-assets Fourier transform-based European option pricing [23].

A robust reference implementation of the core TCI algorithm is provided by the julia package `TensorCrossInterpolation.jl` [24], and further utilities–such as quantics tensor discretisation–are collected in the libraries catalogued at `tensor4all.org` [25]. This software stack forms the computational backbone of most of the works cited above.

Despite its versatility, the original TCI routine can suffer from *ergodicity* problems: it may stall on very sparse tensors, tensors with discrete symmetries, or tensorised functions featuring sharp local peaks. Global pivoting and related heuristics [1] mitigate these issues but do not always succeed, particularly when the tensor originates from *ab initio* calculations for which no *a priori*

[2] A rather playful wording choice given the context.

information is available. More in general, since all TCI algorithms involve sampling, none of them is fully immune against missing some features of the tensor of interest.

To address these issues we propose in this work a *divide-et-impera* extension of TCI. Inspired from similar distributed TT frameworks [26, 27], our new algorithm adaptively partitions the target tensor into smaller subtensors ("patches"), TCI-compressing each with a fixed bond-dimension cap, and thus it concentrates resources on the most challenging regions of configuration space while keeping the overall memory footprint under control. The same patch philosophy also alleviates other tensor-network bottlenecks: we demonstrate how a similar domain decomposition limits intermediate bond growth in MPO-MPO contractions, substantially reducing their computational cost. All these patch-based operations are implemented as a wrapper around the state-of-the-art `crossinterpolate2` kernel distributed with `TensorCrossInterpolation.jl`, preserving the numerical robustness of the original implementation while extending it seamlessly to the new divide-and-conquer workflow.

The manuscript structured as follows: Chapter 2 reviews the standard (Q)TCI algorithm, its canonical implementation, and representative applications. Chapter 3 introduces the patched variant of (Q)TCI—*patched (Q)TCI*—and analyses theoretical costs and limitations. Chapter 4 extends the patch strategy to MPO–MPO contractions, presenting both greedy and adaptive distributed schemes with cost estimates. Chapter 5 presents numerical benchmarks and showcases the new algorithms on selected problems centred around the Hubbard model.

Chapter 2

Quantics Tensor Cross Interpolation

Cross approximation is a tensor compression technique well-established in the literature [1, 18, 28–31]. This technique, pioneered by Oseledets [28] and improved by Dolgov and Savostyanov [31], aims to find a parsimonious interpolation for multi-index tensors with a limited amount of computational resources. Tensor Cross Interpolation (TCI) permitted TT-representations of multivariate functions at a cheaper cost than any SVD based counterpart. Such compressed TT-representations have been used, among other applications, to compute very complex, multi-dimensional integrations, converging better (with no "sign problem" [17]) than standard sampling routines, such as Monte Carlo [18, 31]. Ritter and collaborators have improved the already powerful implementations of TCI, targeting stability and discretization issues of the standard routine [1]. This renewed TCI, however, is not free from suboptimalities, especially when trying to target very complex, "quasi-singular" type of problems.

In this chapter, we summarize the basic technical details of the state-of-the-art implementation of cross interpolation for tensors. Sec. 2.1 gives a generic introduction to all the mathematical and technical details of the TCI routine, in order to provide the reader the tools to understand and, through further reading, reproduce the current state of the algorithm. Nonetheless, considering the goal of this work, no particular importance is given to such details, and more focus is instead directed towards the strengths of the TCI algorithm. Section 2.2 then shows how a naively modified version of TCI can be tailored to functions with sharply localised structure. Multiple examples are employed for this purpose.

2.1 The algorithm

The algorithm is a rank-revealing algorithm for decomposing low-rank, high-dimensional tensors into tensor trains/matrix product states (MPS). Hence, its implementation requires two prerequisites: the tensor should be **compressible**, and the algorithm used for compressing it should be **rank-revealing**. The properties are defined as follows:

Definition 2.1 (Compressible tensor) A tensor $\mathcal{T}$ is **compressible** or **low-rank** if it can be approximated by a Matrix Product State (MPS) with small rank χ.

Definition 2.2 (Rank-revealing algorithm) An algorithm

$$\begin{array}{rccc} \mathcal{A}: & \mathbb{K}^{d_1\times\cdots\times d_{\mathcal{L}}} & \longrightarrow & \mathbb{K}^{d_1\times\cdots\times d_{\mathcal{L}}} \\ & \mathcal{T} & \longmapsto & \widetilde{\mathcal{T}} \end{array}$$

is said to be **rank-revealing**, if it ouputs a low-rank approximation $\widetilde{\mathcal{T}}$ of any compressible $\mathcal{L}$-dimensional tensor $\mathcal{T}$[1]given as input.

TCI depends on these two properties to provide a competitive numerical approximation technique. Given a tensor $\mathcal{T}$ with a hidden low-rank structure as input, TCI always provides a compressed representation of it at a *polynomial-scaling* cost in CPU time and memory. Even when the tensor $\mathcal{T}$ is high rank, TCI is able to output a TT unfolding $\widetilde{\mathcal{T}}$ (at a slower convergence rate). Before explaining how TCI works, let us first dive into the mathematical tools that supply TCI of this *rank-revealing* and *compression* properties.

Matrix Cross Interpolation (CI)

The TCI algorithm bases its implementation on the following statements: *a $M \times N$ matrix of rank χ can be represented using only $\mathcal{O}(\chi)$ of its entries* and *a compressible (cf. Def. 2.1) $M \times N$ matrix can be approximated using $\mathcal{O}(\tilde{\chi}) \ll MN$ of its entries.*

Let A be a $M \times N$ matrix, we introduce the following notations:

- $\mathbb{I} = \{1, \ldots, M\}$ is the ordered set of all row indices of A;
- $\mathbb{J} = \{1, \ldots, N\}$ is the ordered set of all column indices of A;
- $\mathcal{I} = \{i_1, \ldots, i_{\tilde{\chi}}\} \subseteq \mathbb{I}$ and $\mathcal{J} = \{j_1, \ldots, j_{\tilde{\chi}}\} \subseteq \mathbb{J}$ are, respectively, subsets of rows and columns indices of A.

Therefore, in a julia -like fashion[2]we define

$$A[\mathbb{I}, \mathcal{J}], \qquad A[\mathcal{I}, \mathbb{J}], \qquad P = A[\mathcal{I}, \mathcal{J}] \tag{2.1}$$

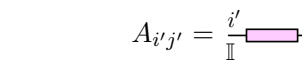
as the submatrices or *slices* containing the intersection elements of $\mathbb{I}$ or $\mathcal{I}$ rows and $\mathbb{J}$ or $\mathcal{J}$ columns (in particular $A = A[\mathbb{I}, \mathbb{J}]$). In a matrix Cross Interpolation (CI) context $P = A[\mathcal{I}, \mathcal{J}]$ is the so-called *pivot matrix* and its elements are the *pivots* of the approximation.

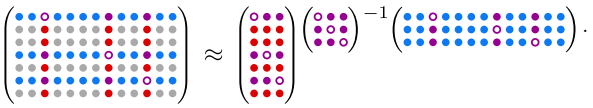
[1]In Def. 2.2 we define a tensor $\mathcal{T}$ as an element of the vector space $\mathbb{K}^{I_1\times\cdots\times I_{\mathcal{L}}}$, this is only true if we consider $\mathcal{T}$ an $\mathcal{L}$-dimensional numerical array, as it is for numerics. More generally $\mathcal{T} \in T^p_q(V) = \{t \mid t : V^{\otimes q} \otimes (V^*)^{\otimes p} \longrightarrow \mathbb{K}\}$ (where $\mathbb{K} = \mathbb{R} \vee \mathbb{C}$ and $V = \mathcal{H}$ for most applications).

[2]From this point onward, we shall consistently use this notation. Slicing notation of the form $A[r_{\text{start}} : r_{\text{start}}, :]$ ($\equiv A_{r,c}$ with $(r, c) \in \{r_{start}, r_{start} + 1, \ldots, r_{end}\} \times \{1, 2, \ldots N\}$) will also be employed.

The CI formula then reads [32]

$$A = A[\mathbb{I}, \mathbb{J}] \approx A[\mathbb{I}, \mathcal{J}] P^{-1} A[\mathcal{I}, \mathbb{J}], \tag{2.2}$$

$A_{i'j'} =$  [3]



Eq. (2.2) gives a rank-$\tilde{\chi}$ approximation of A, where $\tilde{\chi} = \dim(\mathcal{I}) = \dim(\mathcal{J})$. CI is "only" a quasioptimal decomposion of A and its accuracy strongly depends on the choice of the pivots; however, contrarily to its optimal counterparts (e.g. SVD), it doesn't require knowing (and saving in memory) the full $M \times N$ matrix to be computed. Moreover, CI correctly represents the rows and column employed to construct of the approximation on the *r.h.s.* of Eq. (2.2) while also being exact if $\tilde{\chi} = \chi$. Let's consider the following example:

Example 2.1 (5×5 Correlation matrix) For classical Harmonic Oscillator 1D chain mode-like vectors:

$$\boldsymbol{v} = (v_1, v_3, \ldots, v_5) \qquad \boldsymbol{w} = (w_1, w_3, \ldots, w_5)$$

such that $\boldsymbol{v}\boldsymbol{w}^T = 0$, the corresponding position-position correlation matrix can be "cross interpolated" as

$$C = \sigma^2 \boldsymbol{v}^T \boldsymbol{v} + \tilde{\sigma}^2 \boldsymbol{w}^T \boldsymbol{w} =$$

$$= \underbrace{\begin{bmatrix} \sigma^2 v_1 v_1 + \tilde{\sigma}^2 w_1 w_1 & \cdots & \sigma^2 v_1 v_j + \tilde{\sigma}^2 w_1 w_j & \cdots & \sigma^2 v_1 v_5 + \tilde{\sigma}^2 w_1 w_5 \\ \vdots & \ddots & & & \vdots \\ & & \sigma^2 v_i v_j + \tilde{\sigma}^2 w_i w_j & & \\ & & & \ddots & \\ \sigma^2 v_5 v_1 + \tilde{\sigma}^2 w_5 w_1 & \cdots & \cdots & \cdots & \sigma^2 v_5 v_5 + \tilde{\sigma}^2 w_5 w_5 \end{bmatrix}}_{5 \times 5} \tag{2.3}$$

$$\approx \underbrace{C[\mathbb{I}, \mathcal{J}]}_{5 \times 2} \underbrace{\begin{bmatrix} \sigma^2 v_1 v_1 + \tilde{\sigma}^2 w_1 w_1 & \sigma^2 v_1 v_5 + \tilde{\sigma}^2 w_1 w_5 \\ \sigma^2 v_5 v_1 + \tilde{\sigma}^2 w_5 w_1 & \sigma^2 v_5 v_5 + \tilde{\sigma}^2 w_5 w_5 \end{bmatrix}^{-1}}_{2 \times 2} \underbrace{C[\mathcal{I}, \mathbb{J}]}_{5 \times 2}$$

[3] We introduce here a tensor network diagrammatic representation of the matrix multiplication. The internal connecting solid lines represent summation over the respective matrix indices, according to the *Einstein summation convention*. The external lines represent fixed indices.

with σ and $\tilde{\sigma}$ the variances of the two modes and $\mathcal{I} = \{1, 5\}$ and $\mathcal{J} = \{1, 5\}$. From the definition of C we can recognize that $\text{rank}(C) = 2$. This property is correctly highlighted by its CI decomposition ($\dim \mathcal{I} = \dim \mathcal{J} = 2$), since, in this particular case, the approximation is exact (cf. Prop. 2.1.3). The total number of floating point numbers necessary to store the whole matrix in memory is $5 \times 5 = 25$, while for the CI decomposition $5 \times 2 \times 2 + 2 \times 2 = 24$ are sufficient. It is easy to deduce that the reduction in memory costs becomes more important for modes of generic dimension N ($N \times N \gg N \times 2 \times 2 + 2 \times 2$ when $N \gg 1$).

From the example above we can evince the computational advantage of CI, however – in order to make such approximation computationally feasable – some sort of error control is necessary. In particular, the error of the CI approximation is related to the *Schur complement* of the matrix [33].

Definition 2.3 (Schur Complement) Let us block partition a matrix $A \in \mathbb{K}^{M \times N}$ ($\mathbb{K} = \mathbb{R}, \mathbb{C}$) as follows:

$$\begin{matrix} \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} & \begin{matrix} \tilde{\chi} \\ M - \tilde{\chi} \end{matrix} \\ \begin{matrix} \tilde{\chi} & N - \tilde{\chi} \end{matrix} & \end{matrix} \tag{2.4}$$

The **Schur complement** $[A/A_{11}]$ of A is defined by

$$[A/A_{11}] = A_{22} - A_{21}(A_{11})^{-1} A_{12}. \tag{2.5}$$

Proposition 2.1 The following properties hold for a rank-$\tilde{\chi}$ Cross Interpolation decomposition of a matrix A:

1. the error of CI is given by the Schur complement to the pivot matrix;
2. the approximation is exact for any $i \in \mathcal{I}$ or $j \in \mathcal{J}$;
3. the approximation is exact if A has rank $\tilde{\chi}$.

Proof. 1.-2. - The Schur complement is invariant under rows and/or column permutations, therefore let us rearrange $A = A[\mathbb{I}, \mathbb{J}]$ such that

$$A = \begin{bmatrix} A[\mathcal{I}, \mathcal{J}] & A[\mathcal{I}, \mathbb{J}/\mathcal{J}] \\ A[\mathbb{I}/\mathcal{I}, \mathcal{J}] & A[\mathbb{I}/\mathcal{I}, \mathbb{J}/\mathcal{J}] \end{bmatrix}.$$

Then, the r.h.s. of Eq. (2.2) can be rewritten as

$$\tilde{A} = \begin{bmatrix} A[\mathcal{I}, \mathcal{J}] & A[\mathcal{I}, \mathbb{J}/\mathcal{J}] \\ A[\mathbb{I}/\mathcal{I}, \mathcal{J}] & A[\mathbb{I}/\mathcal{I}, \mathcal{J}] \left(A[\mathcal{I}, \mathcal{J}]\right)^{-1} A[\mathcal{I}, \mathbb{J}/\mathcal{J}] \end{bmatrix}$$

which gives (cf. Ref. [1])

$$A - \tilde{A} = \begin{bmatrix} 0 & 0 \\ 0 & [A/A[\mathcal{I}, \mathcal{J}]] \end{bmatrix} \tag{2.6}$$

3. - If $\text{rank}(A) = \tilde{\chi}$ and $P = A[\mathcal{I}, \mathcal{J}]$ is non-singular, then

$$\begin{bmatrix} A[\mathcal{I}, \mathcal{J}] & A[\mathcal{I}, j] \\ A[i, \mathcal{J}] & A[i, j] \end{bmatrix} \qquad \forall\ (i, j) \in \mathbb{I}/\mathcal{I} \times \mathbb{J}/\mathcal{J} \tag{2.7}$$

is singular, which gives $A[i, j] = A[i, \mathcal{J}] \left(A[\mathcal{I}, \mathcal{J}]\right)^{-1} A[\mathcal{I}, j]\ \forall\ (i, j) \in \mathbb{I}/\mathcal{I} \times \mathbb{J}/\mathcal{J}$(cf. App A-B in Ref. [18]). □

Prop. 2.1.1 underlines the importance of the choice of the pivots and of the pivot matrix, specifically with the purpose of minimizing the Schur complement $[A/A[\mathcal{I}, \mathcal{J}]]$. Such procedure is equivalent to maximising $\det A[\mathcal{I}, \mathcal{J}]$ and is known as the *maximum volume principle* [34]. Moreover, from this analysis, we can also get an intuition about why the CI approximation error is at most $O(\tilde{\chi}^2)$ times the optimal one (e.g. $\tilde{\chi}$-truncated SVD error) [35], while requiring only subparts of the original matrix to be known.

Partial rank-revealing LU decomposition (prrLU)

Matrix Cross Interpolation presents itself as a very useful tool when it comes to numerical compression of matrices. Generalization of CI to continuous domains [18, 35] even allows for reduction of numerical complexity of two-dimensional integration and derivation. Nonetheless, for practical, very complex, calculations, CI starts to fail. Numerical instability issues like rounding errors, ill-conditioning or overflowing [33] naturally emerge when CI requires large values of $\tilde{\chi}$ to be accurate, therefore making the pivot matrix *almost singular*.

Partial rank-revealing LU (prrLU) [33, 36] matrix decomposition solves many of the numerical fragilities of CI. The prrLU provides a more stable, but equivalent approximation of our matrix to decompose. prrLU avoids any inversion of the pivot matrix $A[\mathcal{I}, \mathcal{J}]$, whereas still requires a small subset of matrix's elements to be known.

We may summarize the main features of prrLU as follows:

- prrLU is *rank revealing*, i.e. it allows for the iterative determination of the rank of the decomposed matrix;
- prrLU is *partial* (and therefore *controllable*), i.e. the decomposition is stopped after constructing the first $\tilde{\chi}$ rows of L and columns of U, for a fixed $\tilde{\chi}$;
- prrLU is *updatable*, i.e. given pivot lists $\mathcal{I}, \mathcal{J}$ yielding an approximation $\tilde{A}$ of A, new rows and columns can easily be added to $\mathcal{I}, \mathcal{J}$ for an improved approximation.

The prrLU implementation relies on the following LU decomposition (easily inferred from Eq. (2.5)):

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & \mathbb{1}_{22} \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & [A/A_{11}] \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & \mathbb{1}_{22} \end{bmatrix} \tag{2.8}$$

$$L_{11} = U_{11} = \mathbb{1}_{11}, \qquad L_{21} = A_{21}A_{11}^{-1}, \qquad U_{12} = A_{11}^{-1}A_{12}$$

Although matrices L_{21} and U_{12} in Eq. (2.8) contain the inverse of the matrix block A_{11}, the state-of-the-art implementation of prrLU limits A_{11} to a 1×1 slice; hence, no actual matrix inversion is computed directly! The general prrLU algorithmic routine proceeds as outlined below [1]:

Algorithm 2.1: Partial rank revealing LU

Input : $A \in \mathbb{K}^{M \times N}$ matrix, maximum rank $\tilde{\chi}$ and tolerance ε.
Output : Rows permutation Π_r, columns permutation Π_c, L, U, N_{pivots}

$\Pi_r \leftarrow (1, \ldots, M)$, $\Pi_c \leftarrow (1, \ldots, N)$, $n \leftarrow 0$, $\varepsilon_{LU} \leftarrow \varepsilon$;
while $n < \min(\tilde{\chi}, M, N)$ **do**
  $n \leftarrow n + 1$;
  $(r^\star, c^\star) \leftarrow$ `findBestPivot`$\big(A[n\!:\!N, n\!:\!M]\big)$ `// current positions`;
  swap rows $n \leftrightarrow r^\star$ in A and Π_r;
  swap cols $n \leftrightarrow c^\star$ in A and Π_c;
  $\varepsilon_{LU} \leftarrow |A_{n,n}|$;
  if $n > 0$ **and** $\varepsilon_{LU} < \varepsilon$ **then** `// error test`
    break;
  end if
end while
$L \leftarrow$ `LowerTriangular`$\big(A[:, 1\!:\!n]\big)$;
$U \leftarrow$ `strictlyUpperTriangular`$\big(A[1\!:\!n, :]\big)$;
return (Π_r, Π_c, L, U, n);

By iteratively applying Eq. (2.8) on the lower-right block of the internal matrix $\begin{bmatrix} A_{11} & 0 \\ 0 & [A/A_{11}] \end{bmatrix}$, while limiting A_{11} to a 1×1 submatrix at each step, Alg. 2.1 allows us to obtain an approximation of the form

$$A = LDU + \begin{bmatrix} 0 & 0 \\ 0 & [A/A[\mathcal{I}, \mathcal{J}]] \end{bmatrix} = \tilde{A} + \text{err.}, \tag{2.9}$$

equivalent to Eq. (2.2) (cf. Ref. [1]). In Eq. (2.9) D is a diagonal matrix containing – after iterative permutations – $\tilde{\chi}$ pivots of A. Some remarks about Alg. 2.1 are in order

- `findBestPivot`, `LowerTriangular` and `strictlyUpperTriangular` are merely renamed counterparts of the julia routines used in the prrLU implementation of Ref. [1, 24]. The first routine proposes a suitable pivot for the current iteration (often one among several equally valid options),

while the latter two return, respectively, the lower-triangular portion and the strict upper-triangular portion of the input matrix.

- The implementation of prrLU is based upon the search of a good *pivot* (`findBestPivot`). Following the *maxvol principle*, a good pivot is defined as one that attempts to maximise the volume of the submatrix $A[\mathcal{I}, \mathcal{J}]$ ($\equiv A_{11}$ after permutations). Hence, the iterative application of Eq. (2.8) reduces the optimal pivot to the largest element of the submatrix $A[n:N, n:M]$ at iteration n. Such iterative approach, however, does not necessarily extract the pivot matrix $A[\mathcal{I}, \mathcal{J}]$ with larger determinants. This is due to the facto that maximum volume submatrices of larger sizes are not guarantedeed to contain the maximum volume submatrices for smaller sizes. Nonetheless, the algorithm attempts to reach the ideal configuration for $A[\mathcal{I}, \mathcal{J}]$, often with great success.

- Different strategies can be implemented for pivot searching; a naive approach consists in a *full search* scheme scanning the entire submatrix $A[n:N, n:M]$. *Rook search* [37] and *block rook search* [1] are cleverer and cheaper approaches, with comparable robutstness and convergence features, but reduced computational cost $\mathcal{O}(\max(M, N))$.

- The matrix elements explored during *rook search* are sufficient to perform prrLU of A. Therefore, prrLU yields compressibility traits similar to those of CI (see Example 2.1). For this reason, prrLU can also be applied to 2-dimensional continuous functions discretized on a grid, whose full structure is uknown a priori.

- The absolute error of the prrLU approximation is reduced to the modulus of the last inserted pivot, as one can understand from Alg. 2.1. The reason behind this is that we intend to minimize $\| A - \tilde{A} \|_\infty = \| [A/A[\mathcal{I}, \mathcal{J}]] \|_\infty$ which is bounded by $\| A[n:N, n:M] \|_\infty$ at iteration step n.

Tensor Cross Interpolation

Tensor Cross Interpolation is a generalization of matrix Cross Interpolation – and therefore prrLU (see Eq. (2.9)) – to $\mathcal{L}$-dimensional tensors $\mathcal{T}$. Similarly to prrLU, TCI progressively uncovers the *low rank* structure of a given tensor, ultimately rendering a compressed approximation of the input. The main difficulty in the implementation of TCI revolves around bookkeeping of tensor indices – the *pivots* – necessary for a correct approximation. Hence, let us introduce some useful notation:

- $\mathcal{T}_{\boldsymbol{\sigma}} \in \mathbb{K}^{d_1 \times d_2 \times \cdots \times d_{\mathcal{L}}}$ is the TCI input tensor, with indices $\boldsymbol{\sigma} \in I_1 \times I_2 \times \cdots \times I_{\mathcal{L}} := \mathrm{Ind}(\mathcal{T})$ ($|I_i| = d_i$ [4]); $\widetilde{\mathcal{T}}_{\boldsymbol{\sigma}} \in \mathbb{K}^{d_1 \times d_2 \times \cdots \times d_{\mathcal{L}}}$ is the resulting interpolated tensor:

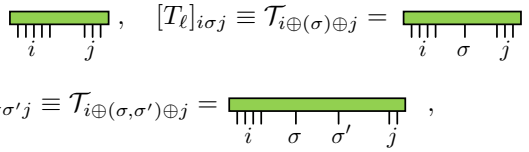
$$\mathcal{T}_{\boldsymbol{\sigma}} = \underbrace{\qquad\qquad}_{\sigma_1 \quad \sigma_2 \quad \cdots \quad \sigma_{\mathcal{L}}} \approx \widetilde{\mathcal{T}}_{\boldsymbol{\sigma}} \,; \tag{2.10}$$

[4] For most application $d_i = d \; \forall i$, with fixed d.

- $\mathbb{I}_\ell = I_1 \times I_2 \times \cdots \times I_\ell$ and $\mathbb{J}_\ell = I_\ell \times I_2 \times \cdots \times I_{\mathcal{L}}$[5] denotes, respectively, the set of all *row multi-indices* and *column multi-indices* up to and from site ℓ (e.g. $i_\ell \in \mathbb{I}_\ell$, $j_\ell \in \mathbb{J}_\ell$ implies $i_\ell = (\sigma_1, \ldots, \sigma_\ell)$, $j_\ell = (\sigma_\ell, \ldots, \sigma_{\mathcal{L}})$);
- $\mathcal{I}_\ell \subseteq \mathbb{I}_\ell$ and $\mathcal{J}_\ell \subseteq \mathbb{J}_\ell$ are, respectively, lists of *pivot rows* and *pivot columns* and contain only a subset of the total row and column multi-indices, the *pivots*;
- the following objects represent *slices* of the original tensor:

$$[P_\ell]_{ij} = \mathcal{T}_{i\oplus j} = \quad , \quad [T_\ell]_{i\sigma j} \equiv \mathcal{T}_{i\oplus(\sigma)\oplus j} = \quad ,$$

$$[\Pi_\ell]_{i\sigma\sigma' j} \equiv \mathcal{T}_{i\oplus(\sigma,\sigma')\oplus j} = \quad , \tag{2.11}$$



where $\oplus$ is the index *concatenation operation*, i.e. $i \oplus j \equiv (\sigma_1, \ldots, \sigma_{\mathcal{L}})$, for fixed $i \in \mathbb{I}_\ell$ and $j \in \mathbb{J}_{\ell+1}$.

The main purpose of TCI is perform the decomposition of a given tensor using only few elements of (few calls to) the tensor $\mathcal{T}_{\boldsymbol{\sigma}}$. Fig. 2.1 below summarizes the main steps of the TCI routine.

The algorithm depicted in Fig. 2.1 represents what is usually referred to as the *2-site TCI algorithm* [1]. The naming *2-site* refers to the fact that the approximation $\widetilde{\mathcal{T}}$ is only built through two-dimensional slices of $\mathcal{T}$. This variant alone enables the recursive extension of the pivot lists and with it the improvement of the approximation, contrarily to the *0-site* and *1-site* that by focusing, respectively, on prrLU optimization of the T and P slices (Eq. (2.11)) simply restore full nesting properties (see below) and remove ill-conditioned pivots. The *2-site* implementation relies on two main ingredients: the partial rank-revealing LU and the *interpolation* properties of TCI. Whilst we extensively described the former in the previous section, a few comments are necessary about the latter. Let us first take a step back and briefly introduce the concept of *nesting conditions*.

The list of pivot rows (columns) is said be *left- (right-) nested* if the following condition holds [11, 31]:

$$\mathcal{I}_0 < \mathcal{I}_1 < \ldots < \mathcal{I}_\ell, \quad (\mathcal{J}_\ell > \mathcal{J}_{\ell+1} > \ldots > \mathcal{J}_{\mathcal{L}+1},) \tag{2.12}$$

where $\mathcal{I}_{\ell-1} < \mathcal{I}_\ell$, if $\mathcal{I}_\ell \subseteq \mathcal{I}_{\ell-1} \times I_\ell$ ($\mathcal{J}_\ell > \mathcal{J}_{\ell+1}$, if $\mathcal{J}_\ell \subseteq J_\ell \times \mathcal{J}_{\ell+1}$). The pivot lists are *fully left- (right-) nested* if $\ell = \mathcal{L} - 1 (= 2)$. *Full nesting* is achieved when the pivots lists are fully left- and right-nested.

The benefit of nesting condition is the already mentioned interpolation characteristics of TCI (we refer the reader to Ref. [1, 18] for proofs and a more detailed explanation). In fact, when pivots are right-nested up to $\ell - 1$ and left-nested from $\ell + 2$ on, then we can define the local error ε_Π

$$\left[\varepsilon_\Pi\right]_{i_{\ell-1}\sigma_\ell\sigma_{\ell+1}j_{\ell+2}} \equiv \left[\Pi_\ell - \widetilde{\Pi}_\ell\right]_{i_{\ell-1}\sigma_\ell\sigma_{\ell+1}j_{\ell+2}} = \left[\mathcal{T} - \widetilde{\mathcal{T}}\right]_{i_{\ell-1}\sigma_\ell\sigma_{\ell+1}j_{\ell+2}}, \tag{2.13}$$

[5] $\mathbb{I}_{\mathcal{L}} = \mathbb{J}_1$ corresponds to the full set of tensor indices, i.e. $\boldsymbol{\sigma} \in \mathbb{I}_{\mathcal{L}} = \mathbb{J}_1 \; \forall \boldsymbol{\sigma}$

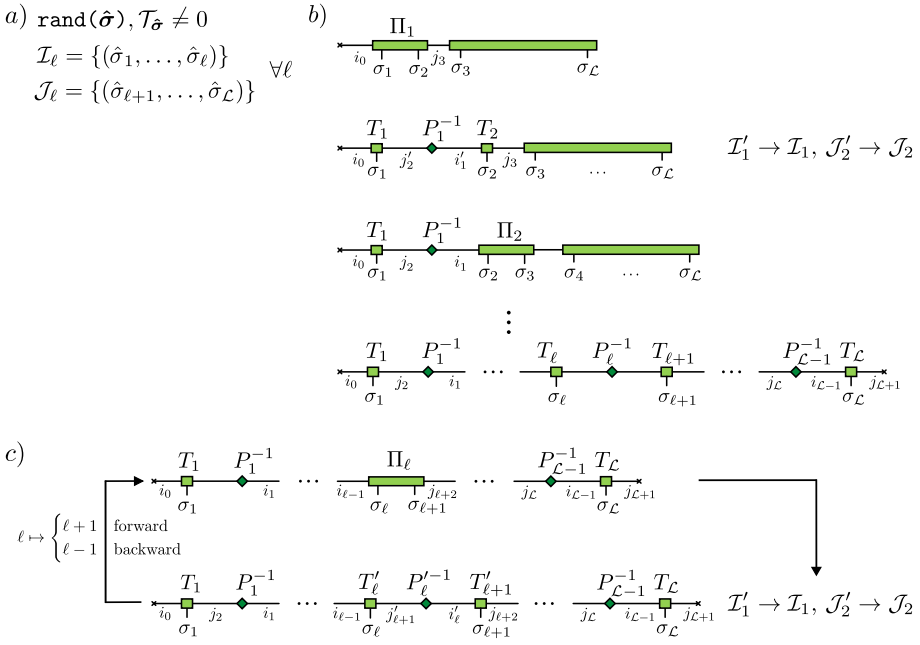
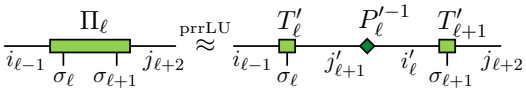


Figure 2.1: Main steps of the TCI algorithm. *a*) A set of indices such that $\mathcal{T}_{\hat{\sigma}} \neq 0$ is chosen randomly from the configuration space $\mathrm{Ind}(\mathcal{T})$. Pivot lists are constructed from the initial multi-indices set. *b*) A first sweep is performed for an initial Matrix Product State representation of our tensor $\mathcal{T}$. At sites ℓ and $\ell+1$, Π_ℓ tensor slices of the form in Eq. (2.11) are constructed from the initial pivot lists $\mathcal{I}_{\ell-1}$ and $\mathcal{J}_{\ell+2}$. prrLU is then perfomed on the "matricized" version of Π_ℓ, namely $\left[\Pi_\ell\right]_{(i_{\ell-1},\sigma_\ell)(\sigma_{\ell+1},j_{\ell+2})}$, and newly found pivots $\mathcal{I}'_\ell$ and $\mathcal{J}'_{\ell+1}$ are added to the initial lists $\mathcal{I}_\ell$ and $\mathcal{J}_{\ell+1}$. This first sweep might resemble the naive approach introduced in Ref. [18], as well as SVD-based MPS tensor unfoldings (cf. Ref. [6]), however it is not different from subsequent iterations of the algorithm. We illustrated it as above to highlight the initial tensor-to-TT transformation. *c*) Sweeping back and forth through index pairs – $\sigma_\ell, \sigma_{\ell+1}$ – the 2-dimensional slices Π_ℓ are reconstructed from the current local pivot lists and prrLU-compressed again. The purpose is to improve the choice of the initial *pivots* and the approximation $\widetilde{\mathcal{T}}$. This last step is performed until convergence.

where



(2.14)

In Eq. (2.13) and Eq. (2.14) (and also Fig. 2.1), Π_ℓ is reconstructed from the current set of local pivots $\mathcal{I}_{\ell-1}$ and $\mathcal{J}_{\ell+2}$; $\widetilde{\Pi}_\ell$ is its approximation through prrLU. Eq. (2.13) allows us to define a error concept consistent along the TT chain ($\forall \ell$), granting the TCI algorithm of a form error control. In fact, step (c) in Fig. 2.1 is performed until the *error* $\left|\Pi_\ell - \widetilde{\Pi}_\ell\right|_{i_{\ell-1}\sigma_\ell\sigma_{\ell+1}j_{\ell+2}}$ is below a fixed tolerance ε. Minimizing the local error means, according to Alg. 2.1 and the *maxvol principle* [31], searching for the largest elements of the tensor $\left|\Pi_\ell - \widetilde{\Pi}_\ell\right|$, adding new pivots that yield the largest improvement to the local accuracy according to the $\|\cdot\|_\infty$ norm. The TCI routine is stopped when the condition $\left|\Pi_\ell - \widetilde{\Pi}_\ell\right| < \tau_\Pi$ is met $\forall \ell \in \{1, \ldots, \mathcal{L}\}$, for the minimal set of pivots possible and a fixed local tolerance τ_Π. The last equality in Eq. (2.13) in not trivial (cf. Ref. [1]) and allows to relate the local error $\left|\Pi_\ell - \widetilde{\Pi}_\ell\right|$ to the accuracy of the global approximation $\widetilde{T}$.

The TCI representation is defined by the selected lists $\mathcal{I}_\ell$ and $\mathcal{J}_\ell$ $\forall \ell$, so an accurate interpolation amounts to optimizing this selection. The TCI routine, as mentioned at the beginning of this work, belongs to the class of "active machine learning" algorithms [19], as it tries to uncover the low-rank structure of a given tensor $\mathcal{T}$ by actively requesting configurations that will better improve its MPS unfolding. Similar to other machine learning (ML) methods, different strategies exist to improve the modelling of our "data set" (e.g. for ML: data augmentation, weighting, prompt engineering etc.). In our case, alongside the local pivot searching strategies – *rook search*, *block rook search* and *full search* – other techniques exist to improve the global approximation of our tensor. The 2-site TCI can be run in *reset mode* or *accumulative mode*. The former recomputes the full lists $\mathcal{I}_\ell$ and $\mathcal{J}_{\ell+1}$ at each prrLU step of the TCI routine, while the latter only adds new pivots to the alredy-existing local lists $\mathcal{I}_\ell$ and $\mathcal{J}_{\ell+1}$. This allows us to discard sub-optimal pivots that were inserted in the pivots lists during the initial exploration of the configuration space, necessary to avoid the pivot matrices P becoming singular. *Global pivot proposals*, similar to multi-start approaches in ML [38], allows the user to incorporate prior information about the tensor $\mathcal{T}$ through a clever choice of the initial configurations $\hat{\boldsymbol{\sigma}}$ (see Fig. 2.1.a), such that all the relevant regions of configuration space are explored.

The above techniques all try to target TCI *ergodicity* issues that arise in different implementations. Although most common TCI applications don't require any further expedient on top of the ones we just mentioned, as we will understand in the rest of this work, there exist many other, especially if modelling very extreme-conditioned physical systems, that call for additional improvements. In particular, *sparse or symmetric tensors* and *narrow peaked multivariate* functions are the main weaknesses for TCI.

Tensor Cross Interpolation, as the $\mathcal{L}$-dimensional extension of CI, conserves its compression properties. The number of elements of $\mathcal{T}$ necessary for its TT decomposition is limited to the $\boldsymbol{\sigma}$ configurations obtained out of concatenation

action	variant		calls to $\mathcal{T}_{\boldsymbol{\sigma}}$	algebra cost
iterate	rook piv.	2-site	$\mathcal{O}(\chi^2 d n_{\text{rook}} \mathcal{L})$	$\mathcal{O}(\chi^3 d n_{\text{rook}} \mathcal{L})$
	full piv.	2-site	$\mathcal{O}(\chi^2 d^2 \mathcal{L})$	$\mathcal{O}(\chi^3 d^2 \mathcal{L})$
	full piv.	1-site	$\mathcal{O}(\chi^2 d \mathcal{L})$	$\mathcal{O}(\chi^3 d \mathcal{L})$
	full piv.	0-site	0	$\mathcal{O}(\chi^3 \mathcal{L})$
achieve full nesting			$\mathcal{O}(\chi^2 d \mathcal{L})$	$\mathcal{O}(\chi^3 d \mathcal{L})$
add n_p global pivots			$\mathcal{O}\big((2\chi + n_p) n_p \mathcal{L}\big)$	$\mathcal{O}\big((\chi + n_p)^3 \mathcal{L}\big)$
compress tensor train	SVD		0	$\mathcal{O}(\chi^3 d \mathcal{L})$
	prrLU			
	CI			

Table 2.1: Computational cost of the main TCI routines. Full nesting routines are useful to restore interpolation properties for our Tensor Train approximation. Rook pivoting and full pivoting are different possible choices for the pivot search in prrLU/CI subroutines. n_{rook} corresponds to the maximum number of rook search moves necessary to find an optimal local pivot ($n_{\text{rook}} < 5$ for most applications). The table is taken directly from Ref. [1].

of the pivots in the *pivot lists* – $\mathcal{I}_\ell$,$\mathcal{J}_{\ell+1}$ $\forall \ell$. Hence, the approximation is systematically controlled by χ, where $\chi = \max_\ell \chi_\ell$ is the *rank* of the tensor $\mathcal{T}$ with $\chi_\ell = \dim \mathcal{I}_\ell = \dim \mathcal{J}_{\ell+1}$ rank of the local pivot matrix P_ℓ. As a consequence, the resources necessary to perform TCI scale strongly with χ.

The number of function calls necessary for the MPS unfolding $\widetilde{\mathcal{T}}$ of $\mathcal{T}$ is $\mathcal{O}(\chi^2 d^2 \mathcal{L})$ compared to the $d^{\mathcal{L}}$ ($= |\text{Ind}(\mathcal{T})|$) of its SVD counterpart, where d is the configuration space dimension. Such an exponential advantage is obtained by fully specifying only $\mathcal{O}(\chi^2 \mathcal{L})$ number of *pivots* from the original tensor. The algebra cost ($\sim$ computational time) of the algorithm scales as $\mathcal{O}(\chi^3 d^2 \mathcal{L})$. Table 2.1 summarises the scaling of the core TCI routines implemented in the `TensorCrossInterpolation.jl` library [24]. As illustrated in Example 2.2, these scalings underscore the pronounced memory efficiency inherent to TCI.

Example 2.2 ($\mathcal{L}$-dimensional function) Let us consider the following $\mathcal{L}$-dimensional function, inspired by [1]

$$f(\boldsymbol{x}) = \frac{2^{\mathcal{L}}}{1 + 2\sum_{\ell=1}^{\mathcal{L}} x_\ell}. \tag{2.15}$$

Such a continuous function can be numerically represented by a tensor $\mathcal{F}_{\boldsymbol{\sigma}}$ through grid discretization over a preferred domain. For this exercise, we take a 61 point Gauss-Kronrod type of grid, over the $[0,1]^{\mathcal{L}}$ hypercube. What this means in practice is that $\mathcal{F}_{\boldsymbol{\sigma}} \in \mathbb{R}^{d_1 \times \cdots \times d_{\mathcal{L}}}$ [6] and $\dim I_i = d = 61$. The TCI approximation of $\mathcal{F}_{\boldsymbol{\sigma}}$, namely $\widetilde{\mathcal{F}}_{\boldsymbol{\sigma}}$, is obtained through a limited number of calls to the original function $f(\boldsymbol{x})$, as depicted in Fig. 2.2 below.

Fig. 2.2 documents the reduced memory footprint required from the TCI representation of functions up to 20 dimensions. Exponential scaling of memory

[6] From this point onward, $\mathcal{F}$ will refer to the tensor discretization of a continuous function, while $\mathcal{T}$ will be the notation for a generic $\mathcal{L}$-dimensional tensor.

requirements, necessary to store the total $61^{\mathcal{L}}$ number of tensor elements of $\mathcal{F}_{\boldsymbol{\sigma}}$, is replaced by a $\mathcal{O}(\chi^2)$ scaling. Moreover, as one can understand from the bottom right plot in Fig. 2.2, out of the total 61^2 ($\mathcal{L} = 2$) Gauss-Kronod grid points, only 49 are actually required for a nearly optimal approximation of f, therefore limiting the bond dimension χ of our tensor train to $\chi = 7$ and with that the number of total stored parameters.

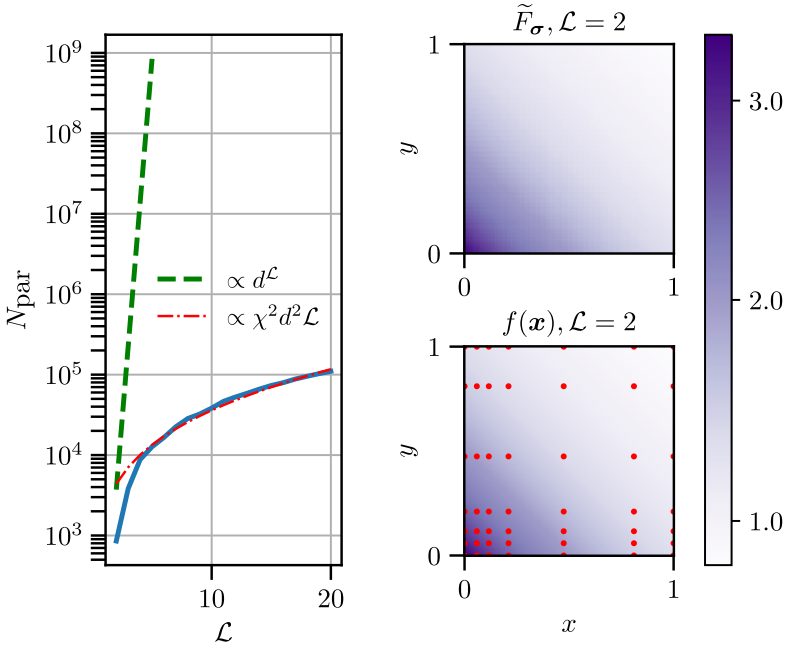


Figure 2.2: TCI approximation of $f(\boldsymbol{x})$ in Eq. (2.15) after discretization on a $\mathcal{L}$-dimensional 61 point Gauss-Kronod grid. *On the left*: number of total floating point elements stored in the compressed tensor $\widetilde{\mathcal{F}}_{\boldsymbol{\sigma}}$ as a function of $\mathcal{L}$ (blue line). The "worst-case-scenario" scaling $d^{\mathcal{L}}$ and the theoretical scaling $\mathcal{O}(\chi^2 d^2 \mathcal{L})$ are also represented (green and red dashed line, respectively). *Top right:* heatmap of the $\mathcal{L} = 2$ approximation $\widetilde{\mathcal{F}}_{\boldsymbol{\sigma}}$ over the 61×61 Gauss-Kronod grid, subset of $[0, 1] \times [0, 1]$, with absolute tolerance $\epsilon = 10^{-8}$. *Bottom right:* heatmap of the $\mathcal{L} = 2$ original function $f(\boldsymbol{x})$ over the domain $[0, 1] \times [0, 1]$. The red dots represent the location of the *pivot* values for the TCI approximation above.

TCI is not only useful for function approximation. A very obvious application of TCI is the computation of integrals in high-dimensional spaces. Detailed examples of this particular usage of TCI are provided in Ref. [1, 18, 31]. TCI-based integration outperform Monte-Carlo and quasi-Monte-Carlo methods in many occasions and the reasons behind it are diverse.

Consider, for example, the following integral

$$I \equiv \int d\boldsymbol{x}\ f(x_1, \ldots, x_{\mathcal{L}}), \tag{2.16}$$

which can be approximated as a Riemann sum,

$$I \approx \sum_{\boldsymbol{\sigma}} \mathcal{F}_{\boldsymbol{\sigma}}, \quad \mathcal{F}_{\boldsymbol{\sigma}} = f(\boldsymbol{x}(\boldsymbol{\sigma})) \tag{2.17}$$

where $\boldsymbol{x}(\boldsymbol{\sigma})$ is our discretization grid (cf. Example 2.2). If we decide to perform TCI unfolding of our numerical tensor $\mathcal{F}_{\boldsymbol{\sigma}}$ before the integration, Eq. (2.17) then reads[7]

$$\sum_{\boldsymbol{\sigma}} \mathcal{F}_{\boldsymbol{\sigma}} \approx \prod_{\ell=1}^{\mathcal{L}} \sum_{\sigma_\ell=1}^{d_\ell} T_\ell^{\sigma_\ell} P_\ell^{-1} \tag{2.18}$$

replacing one $\mathcal{L}$-dimensional integral by $\chi^2\mathcal{L}$ exponentially easier 1-dimensional integrals. Moreover, if the rank of the MPS unfolding of the integrand remains roughly constant as the number of dimensions increases, then the advange provided by TCI increases exponentially. Given the numerical simplicity of the algebra operations in Eq. (2.18), the bottleneck of integration operations is limited to finding the right TCI approximation of the integrand, bounding the parameter scaling to a *polynomial* cost of $\mathcal{O}(\chi^2 d\mathcal{L})$.

The above achievement of TCI relies on the following property of continuous functions:

Definition 2.4 A function f is **almost separable** [1] or **ε-factorizable** [18] if its tensor representation $\mathcal{F}$ is low-rank.

For this class of functions the numerical advantage of TCI integration over more standard approaches – like Monte Carlo sampling – is remarkable (cf. $1/N_{\text{eval}}^4$ vs. $1/\sqrt{N_{\text{eval}}}$ convergence for N_{eval} function evaluations in Fig. 2 of Ref. [1]). In addition, TCI does not suffer from the *"sign problem"* [17] of Monte Carlo methods, *i.e.* slow convergence of the integral error with the number of samples when ingrating over strongly oscillating functions. On the other hand, TCI performs well even when integrating functions oscillating simultaneously on very different scales. The limiting factor of TCI (rank of the ε-factorization) is entirely orthogonal to that of sampling methods.

If the user intends to employ TCI purely to perform integrations, the definition of an *environment-aware error* (from Eq. (2.13) and Eq. (2.18)) might turn out to be very useful. It is defined as

$$\begin{aligned} \left[\varepsilon_\Pi^{\text{env}}\right]_{i_{\ell-1}\sigma_\ell\sigma_{\ell+1}j_{\ell+2}} &\equiv |L_\ell R_\ell| \left[\varepsilon_\Pi\right]_{i_{\ell-1}\sigma_\ell\sigma_{\ell+1}j_{\ell+2}} \\ L_\ell = \prod_{\bar{\ell}=1}^{\ell-1} \sum_{\sigma_{\bar{\ell}}=1}^{d_{\bar{\ell}}} T_{\bar{\ell}}^{\sigma_{\bar{\ell}}} P_{\bar{\ell}}^{-1} \quad & R_\ell = \prod_{\bar{\ell}=\ell+2}^{\mathcal{L}} \sum_{\sigma_{\bar{\ell}}=1}^{d_{\bar{\ell}}} P_{\bar{\ell}-1}^{-1} T_{\bar{\ell}}^{\sigma_{\bar{\ell}}}. \end{aligned} \tag{2.19}$$

The environment error function $\varepsilon_\Pi^{\text{env}}$, by readjusting Substituting the usual definition of local error ε_Π ot be the error of the inegrand, allows the TCI routine

[7] P and T slices are here considered as (σ_ℓ-dependent) matrices. Summation over common indices is therefore implied. $P_{\mathcal{L}} = [1]$.

to be more integral-focused, outperforming the standard implementation when it comes to integral error convergence. By selecting pivots not only based on the absolute value of the integrand but also taking into account a specific point's volume contribution to the integral, $\varepsilon_{\Pi}^{\text{env}}$ allows for more precise computations of complex integrals particularly for multi-scaled functions [18].

The Quantics Representation: QTCI

TCI is a powerful tool that can be used "out-of-the-box" for most applications. The standard implementation however lacks the capabilities of reaching very high resolutions for the approximatino of function of lower dimensions. Let us consider Fig. 2.2 in Example 2.2: one might have noticed that the plot depicting the TCI approximation of Eq. (2.15) shows some subtle visual imperfections. These imperfections arise naturally when discretization errors become dominant in our TCI representation. A naive solution to this issue would be to increase the number of total grid points $d \times d$ used to discretize our 2D function. Despite solving our discretization troubles, such an easy fix will make the number of grid points d the main contributor to TCI scaling, reducing the compression capabilities of the TT unfolding (e.g. for our specific example with $\mathcal{L} = 2$ and, let's say, $d = 1000$ grid points we would obtain a 2-sited MPS with site dimension 1000).

Given a function $f(\boldsymbol{x})$ which we intend to resolve at very high resolution – or more ambitiously at *superhigh resolution* – the *quantics tensor representation* could turn out to be very advantageous. Quantics representation has been a standard approach – not limited to TT approximations – to target resolution issues, well-established in the literature [21, 27, 39–42]. Applications revolving around this type of representation are quite diverse in the many-body physics community, ranging from computations of correlation functions for quantum many body systems [27] to diagrammatic non-equilibrium many-body Green's function-based calculations [41] and compression of imaginary-time propagators in the Frobenious norm [42].

Discretization errors are often a consequence of poor scaling of our numerical approximation tool or of a suboptimal grid choice. While the former is definitely not an issue in the context of TCI [1], the latter can definitely be better addressed.

Given a function of $\mathcal{N}$ variables $f(\boldsymbol{x})$ we discretize each variable through a dyadic grid with $M = 2^{\mathcal{R}}$ points per variable,

$$x_n(m_n) = (x_{n,\max} - x_{n,\min})\frac{m_n}{2^{\mathcal{R}}} + x_{n,\min} \tag{2.20}$$

where each index $m_n \in \{0, \dots, M-1\}$ is written in binary form with $\mathcal{R}$ bits as

$$m_n(\boldsymbol{\sigma}_n) = m_n(\sigma_{n1}, \dots, \sigma_{n\mathcal{R}}) = \sum_{r=1}^{\mathcal{R}} \sigma_{nr}\, 2^{\mathcal{R}-r}, \qquad \sigma_{nr} \in \{0, 1\}, \tag{2.21}$$

so that the $\mathcal{N}$-variate function f is represented by the binary tensor $\mathcal{F}_{\boldsymbol{\sigma}} := f\big(x_1(\boldsymbol{\sigma}_1), \dots, x_{\mathcal{N}}(\boldsymbol{\sigma}_{\mathcal{N}})\big)$ on such a grid [27, 40].

A well-defined tensor $\mathcal{F}_{\boldsymbol{\sigma}}$ requires us to unambiguously specify the *order* of the tensor indices so that an MPS representation can be constructed These aer multiple possible orderings

- *natural ordering*: $\mathcal{F}_{\boldsymbol{\sigma}} \equiv \mathcal{F}_{(\sigma_{11},\ldots,\sigma_{1\mathcal{R}},\sigma_{21},\ldots,\sigma_{2\mathcal{R}},\ldots\ldots,\sigma_{\mathcal{N}1},\ldots,\sigma_{\mathcal{N}\mathcal{R}})}$ combining dimensions;
- *interleaved ordering*: $\mathcal{F}_{\boldsymbol{\sigma}} \equiv \mathcal{F}_{(\sigma_{11},\ldots,\sigma_{\mathcal{N}1},\sigma_{12},\ldots,\sigma_{\mathcal{N}2},\ldots\ldots,\sigma_{1\mathcal{R}},\ldots,\sigma_{\mathcal{N}\mathcal{R}})}$ combining scales;
- *fused ordering*: $\mathcal{F}_{\tilde{\boldsymbol{\sigma}}} \equiv \mathcal{F}_{(\tilde{\boldsymbol{\sigma}}_1,\ldots,\tilde{\boldsymbol{\sigma}}_{\mathcal{R}})}$ fusing scales, where $\tilde{\boldsymbol{\sigma}}_r = \sum_{n=1}^{\mathcal{N}} 2^{n-1}\sigma_{nr}$.

Once the *index ordering* is established, cross interpolation of the tensor $\mathcal{F}_{\boldsymbol{\sigma}}$ yields a TT approximation of our initial function f.

Combining TCI with the quantics representation allows us to represent a given function f through an MPS and resolve it up to a scale of order $1/2^{\mathcal{R}}$; we will name this routine *Quantics Tensor Cross Interpolation* or *QTCI*.

QTCI constructs the local tensors $T_\ell^{\sigma_\ell}$ and P_ℓ^{-1} with a cost of $\mathcal{O}(\mathcal{L}d\chi^2) = \mathcal{O}(\chi^2 d \log M)$, similar to TCI, i.e. linear in the number of bits $\mathcal{R}$ and logarithmic in the grid size (recovering Khoromskij's $O(d\log n)$ scaling [40]), where $\mathcal{L} = \mathcal{N}\mathcal{R}$ or $\mathcal{L} = \mathcal{R}$ depending on the index ordering choice. On the other hand, because the mesh width is $\Delta = 2^{-R}$, analytic f satisfy a spectral error bound

$$\|f - \tilde{f}_R\|_\infty \;\leq\; C\, e^{-cR} = C\,\Delta^{c/\ln 2}, \tag{2.22}$$

so that the discretization error decays exponentially with $\mathcal{R}$ while storage and CPU time grow only linearly with it [40, 43].

Interleaving (fusing) the bits as $\sigma_{\ell(n,r)}$ – where $\ell = n + (r-1)\mathcal{N}$ ($\ell = r$) – arranges (fuses) all sites that resolve the *same* length-scale 2^{-r} next to (with) one another. Whenever cross-scale correlations are weak this ordering yields a tensor-train (TT) of small, $\mathcal{R}$-independent rank χ [27, 40]. QTCI has no problem uncovering such an underlying structure, discarding the weak entanglement betweeen different scales. Let us discuss this further in the following example.

Example 2.3 (2D scale separated function) Consider the following 2D function

$$\begin{aligned} f(x,y) = {} & \exp(-0.4(x^2+y^2)) + 1 + \sin(xy)\exp(-x^2) \\ & + \cos(3xy)\exp(-y^2) + \cos(x+y) \\ & + 2^{-4}\cos(2^{-4}(0.2x - 0.4y)) + 2^{-8}\cos(2^{-4}(-0.2x + 0.7y)) \end{aligned} \tag{2.23}$$

The f might present a very high level of scale separation, where each individual function scale, of order $\mathcal{O}(1)$, $\mathcal{O}(1/2^4)$ and $\mathcal{O}(1/2^8)$ respectively, sees the rest of the function either as a summing constant or as some irrelevant small-magnitude noise. We may expect that performing a QTCI compression of this function, with interleaved ordering, would render a low-ranked TT, given this separation. The level of entanglement between different bit bipartitions can give us an intuition of the amount of scale disconnection present in the function.

An entanglement measure can be achieved through the **Von Neumann** – or **entanglement entropy** S. Given a generic tensor representation of the form $F_{\sigma_1 \dots \sigma_{\mathcal{L}}}$, the Von Neumann entropy at site ℓ can be evaluated through singular value decomposition (SVD) of the matrix

$$\rho_\ell = \rho_{(\sigma_1 \dots \sigma_\ell),(\sigma'_1 \dots \sigma'_\ell)} = \sum_{\sigma_{\ell+1} \dots \sigma_{\mathcal{L}}} F_{(\sigma_1 \dots \sigma_\ell),(\sigma_{\ell+1} \dots \sigma_{\mathcal{L}})} F^\dagger_{(\sigma_{\ell+1} \dots \sigma_{\mathcal{L}}),(\sigma'_1 \dots \sigma'_\ell)}. \quad (2.24)$$

The singular values of ρ_ℓ, namely s_ℓ, can then be used, after normalisation, to calculate the local Von Neumann entropy

$$S_\ell = -\sum_{\alpha=1}^{\chi_\ell} s_\ell^2 \log s_\ell^2 \quad (2.25)$$

as entanglement measure between the bipartions $(\sigma_1 \dots \sigma_\ell)$ and $(\sigma_{\ell+1} \dots \sigma_{\mathcal{L}})$ of the system.

Since bits $\ell = 2(r-1)$ and $\ell = 2(r-1)+1$ (interleaved representation with $\mathcal{N} = 2$) resolve scale 2^{-r} of our function, entanglement measure around these sites can give us an intuition of the amount of information that needs to be propagated from scale 2^{-r} to the other scales for a correct TT representation of f. Fig. 2.3 depicts this entanglement measure for the QTCI compression with $\mathcal{R} = 10$ of $f(x, y)$ in Eq. (2.23) and of `rand(`x, y`)`, that generates random noise. `rand(`x, y`)` is the perfect example of non-compressible function: no hidden low-rank structure and absolutely no scale separation. In the case of f we can observe very low entanglement for all the different bit bipartions, proving once agan its scale separation. On the contrary, the random noise function is strongly entangled along its whole MPS chain and the entire discretized tensor is necessary to represent it as an MPS, such that no compression is achieved by (Q)TCI. Accordingly, the bond dimensions are much larger for `rand()` than for $f(x, y)$

When computing numerically with multivariate functions, one must usually balance two opposing aims: faithfully capturing the function's detail and minimising the memory required for it. Employing the quantics representation together with TCI compression, however, allows us to realise *high resolutions* $(\Delta \to 0)$ at limited computational cost, making QTCI the method of choice whenever scale-separation permits a small TT rank.

`TensorCrossInterpolation.jl` library

In the previous sections we introduced technical details and different applications of the TCI and QTCI algorithms. As briefly mentioned, all of the examples we presented rely on a julia implementation of the algorithm, namely `TensorCrossInterpolation.jl` [24, 25]. We will not dive into this details of the library however, as reference for the rest of the work, we will mention its main functionality: the `crossinterpolate2` function.

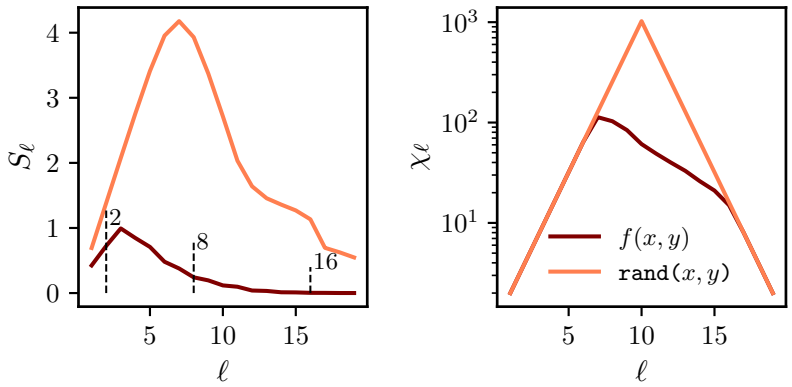


Figure 2.3: *Left:* entanglement entropy per site S_ℓ vs site ℓ for the function of Eq. (2.23) and a 2D random noise function. The vertical dashed lines indicate the bonds where the bits $(\sigma_{1r'}, \sigma_{2r'})$, for $r' \in [1, 4, 8]$, are next to the bipartition cut. *Right:* bond dimension per site χ_ℓ for the same two functions.

`crossinterpolate2`, described in the Listing 2.1 below, performs TCI of a given numerical function `f`; in our context, `f` is either represented by the discretized version of a multi-variate continuous function – $f(\boldsymbol{x}(\boldsymbol{\sigma}))$ – or by any tensor-like numerical function – $\mathcal{F}_{\boldsymbol{\sigma}}$. Given the definition of our tensor $\mathcal{F}_{\boldsymbol{\sigma}}$ as a `function` type, we can understand that its elements don't need to be known and stored beforehand. The output of the computation is an object containing the site tensors of the TT-unfolding, together with the lists of pivots necessary to realize the cross approximation.

```
function crossinterpolate2(
  ::Type{ValueType},          # Return type of f, usually Float64 or ComplexF64
  f,                          # Tensorized function of interest:  f(x(σ)) or F_σ
  localdims::Union{Vector{Int},NTuple{N,Int}},  # Local dimensions (d_1,...,d_L)
  initialpivots::Vector{MultiIndex};  # List of initial pivots {σ̂}.
                                      # Default:  {(1,...,1)}
  tolerance::Float64,         # Global error tolerance τ for TCI. Default:  10^-8
  pivottolerance::Float64,    # Local error tolerance τ_Π for prrLU. Default:  τ
  maxbonddim::Int,            # Maximum bond dimension χ_max.  Default:  no limit
  maxiter::Int,               # Maximum number of half-sweeps.  Default:  20
  pivotsearch::Symbol,        # Full or rook pivot search?  Default:  :full
  normalizeerror::Bool,       # Normalize ε by max_{σ∈samples} F_σ?  Default:  true
  ncheckhistory::Int          # Convergence criterion:
                              # ε_{n_iter} < ε for how many iterations?  Default:  3
) where {ValueType,N}
```

Listing 2.1: Main TCI routine of the `TensorCrossInterpolation.jl` library `crossinterpolate2`. The details of each input variable are described in the relative inline comments.

2.2 QTCI approximation of functions with narrow peaks

Sparse or symmetry-constrained tensors, and–more relevant here–multivariate functions that contain a few sharp local peaks, expose two latent weaknesses of standard TCI. First, the informative regions of the configuration space occupy only a tiny fraction of the full domain; random or purely local pivot searches can miss them, revealing TCI's *ergodicity* problem. Second, fitting the entire (mostly trivial) domain with a *single* tensor train forces one global bond dimension, even though different parts of the domain in fact require widely different ranks. The result is often an over-ranked representation of the function, which mostly traslates to an unnecessary increase in memory costs.

To illustrate these limitations—and to motivate the distributed strategy developed in the following chapters—we introduce a minimal variant of TCI tailored to sharply localised functions.

A "naive" solution

Ergodicity problems render TCI – and QTCI as well – a deficient tool when we attempt to approximate functions with very local and sharp features. This can be solved by *global pivot insertion*, where additional sampling points $\hat{\boldsymbol{\sigma}}$ are inputted by an external source. Proposing clever initial configurations $\hat{\boldsymbol{\sigma}}$ (see Fig. 2.1) to the TCI routine – similar to prompt engineering in deep learning algorithms – can help TCI uncover all the relevant features of the function of interest. The main weakness of this approach is that the target function we intend to TCI compress might not be known *a priori*, so no information can be used to improve its approximation. Let us consider the following function

$$f(\boldsymbol{r}) = \underbrace{A_1\, e^{-\frac{(\boldsymbol{r}-\boldsymbol{r}_1)^2}{2\sigma_1^2}} \sin(k_1 \boldsymbol{r})}_{f_1(\boldsymbol{r})} + \underbrace{A_2\, e^{-\frac{(\boldsymbol{r}-\boldsymbol{r}_2)^2}{2\sigma_2^2}} \sin(k_2 \boldsymbol{r})}_{f_2(\boldsymbol{r})}, \tag{2.26}$$

where

$$\boldsymbol{r} = (x, y), \quad \boldsymbol{r}_{1/2} = \pm(0.5, 0.5),$$
$$A_1 = 10^3, A_2 = 10^6, \quad \sigma_1 = 10^{-1}, \sigma_2 = 10^{-3}, \quad k_1 = 10^4, k_2 = 10^3.$$

$f(\boldsymbol{r})$ is composed by two, almost indipendent terms, with very different absolute scales. $f_1(\boldsymbol{r})$ has wider support but it smaller in absolute value and more slowly oscillating, whereas $f_2(\boldsymbol{r})$ is quickly oscillating but more localised and of larger absolute value. After quantics discretization, the tensor $f(\boldsymbol{r}(\boldsymbol{\sigma})) = \mathcal{F}_{\boldsymbol{\sigma}}$ can be TCI compressed to $\widetilde{\mathcal{F}}_{\boldsymbol{\sigma}}$.

The absolute quality of the local approximation can be measured through

$$\varepsilon_{\log}(\boldsymbol{r}(\boldsymbol{\sigma})) = \log_{10} \left| \mathcal{F}_{\boldsymbol{\sigma}} - \widetilde{\mathcal{F}}_{\boldsymbol{\sigma}} \right|, \quad \boldsymbol{r}(\boldsymbol{\sigma}) \in [0,1]^2 \tag{2.27}$$

where a smaller value indicates better precision. We show this error measure in the *top central* plot in Fig. 2.5, for a QTCI of f with $R = 20$, desired absolute

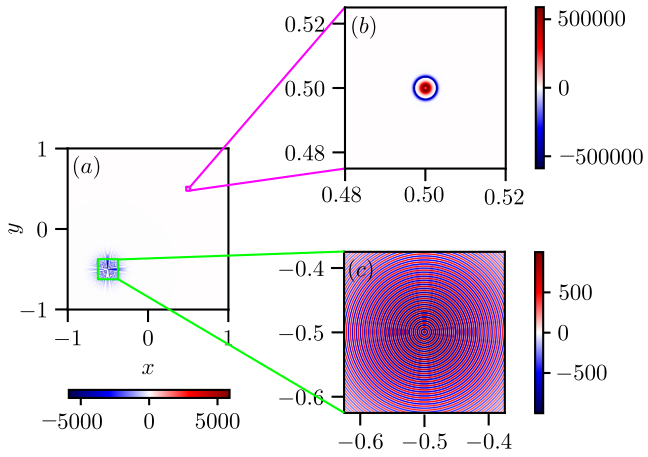


Figure 2.4: (a) Heatmap of the function $f(\boldsymbol{r})$ within the domain $[0,1]^2$. Zoomings into the enviroment of (b) $\boldsymbol{r}_1$ and (c) $\boldsymbol{r}_2$, which are the peak positions of f_1 and f_2, respectively. Eq. (2.26).

tolerance $\tau = 10^{-7}$ and fused quantics index ordering. The image suggests that – as expected from Fig. 2.4 – most of the function domain is easy to represent, while the computational effort is focused around the centers of our local features $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$. Dividing the domain into computationally simple and complex subdomains – or *patches* – could benefit the overall cost and, to some degree, accuracy of the QTCI representation. Other arguments support this strategy, as follows.

Given two tensors $A_{\boldsymbol{\sigma}}$ and $B_{\boldsymbol{\sigma}}$ on the same configuration space $\boldsymbol{\sigma}$, we define the element-wise sum tensor $S_{\boldsymbol{\sigma}} = A_{\boldsymbol{\sigma}} + B_{\boldsymbol{\sigma}}$ as the direct sum – or partial direct sum – of the two tensors. The same operation can be performed for Tensor Trains, by direct summing each site tensor (cf. Sec. 3.1 and Ref. [44]). In our particular example, if we decide to approximate each summand $f_1(\boldsymbol{r})$ and $f_2(\boldsymbol{r})$ in Eq. (2.26) with TTs – $\widetilde{\mathcal{F}}_1$ and $\widetilde{\mathcal{F}}_2$ – then

$$f(\boldsymbol{r}(\boldsymbol{\sigma})) = \mathcal{F}_{\boldsymbol{\sigma}} \approx \widetilde{\mathcal{F}}_{1\boldsymbol{\sigma}} + \widetilde{\mathcal{F}}_{2\boldsymbol{\sigma}} = \widetilde{\mathcal{F}}^{+}_{\boldsymbol{\sigma}} \tag{2.28}$$

with negligible error, due to the independency of the two terms. Fig. 2.5, supports this argument. The first pair of plots *(a.1)* and *(b.1)* portray the error function of the QTCI approximation *focused* on the support region of $f_1(\boldsymbol{r})$ and $f_2(\boldsymbol{r})$. The second pair *(a.2)* and *(b.2)*, on the other hand, measures the error for the QTCI approximations – $\widetilde{\mathcal{F}}_{2\boldsymbol{\sigma}}$ and $\widetilde{\mathcal{F}}_{1\boldsymbol{\sigma}}$ – *limited* [8]to the support of $f_1(\boldsymbol{r})$ and $f_2(\boldsymbol{r})$.

We can observe a slight improvement in the local approximation error in the last row of plots compared to the first one. Whenever QTCI is constricted to each local feature of f it is able to resolve the target with better precision.

[8]When limiting the QTCI approximation, in order to produce results with similar resolution to the standard application, we reduce the number of bits to $\mathcal{R}_1 = 17$ and $\mathcal{R}_2 = 16$, while keeping the other parameters unchanged.

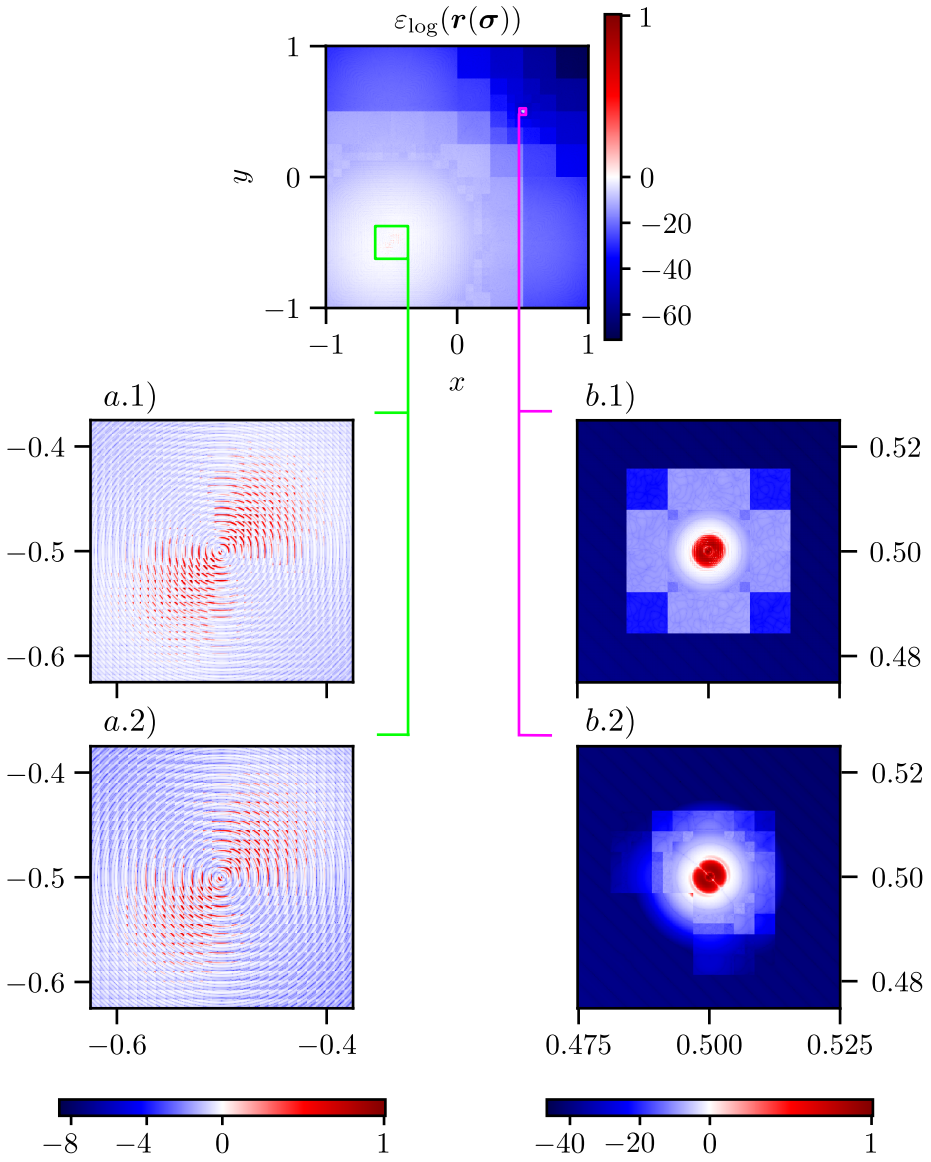


Figure 2.5: Logarithmic error $\varepsilon(\boldsymbol{r}(\boldsymbol{\sigma}))$ for the QTCI representation $\widetilde{\mathcal{F}}_{\boldsymbol{\sigma}}$ of $f(\boldsymbol{r})$ in Eq. (2.26) (*top*) with different zoomings (*a-b.1*). Error measure of QTCI of f *limited* to the surrounding of $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ is also shown (*a-b.2*). We refer to the main manuscript for further details.

Nevertheless, the main advantage of this naive solution is not this small accuracy improvement *per se*.

The computational effort of this *"divide and conquer"* variant of QTCI – apart from being very well suited for parallelization – frees the algorithm from the constraint of representing a single object, made of different independent components, using a single tensor train. The improvement in numerical re-

sources requirement is non-negligible. The bond dimension development along the chains of the MPS unfoldings $\widetilde{\mathcal{F}}_{\boldsymbol{\sigma}}$, $\widetilde{\mathcal{F}}_{2\boldsymbol{\sigma}}$ and $\widetilde{\mathcal{F}}_{1\boldsymbol{\sigma}}$ is a witness of that, as shown in Fig. 2.6.

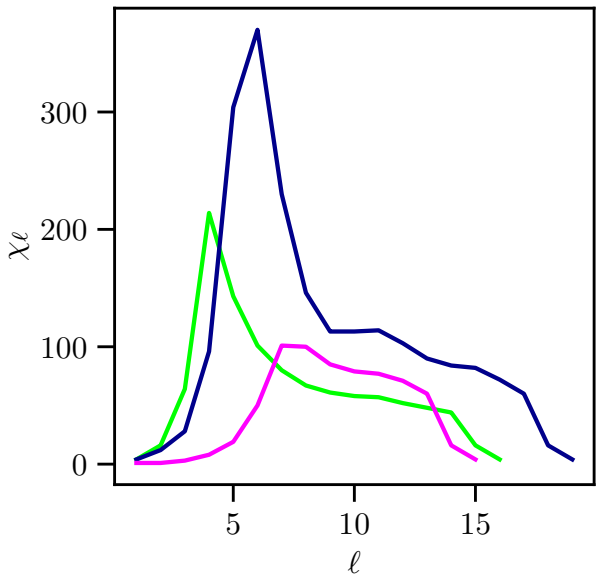


Figure 2.6: Bond dimension per-site of the TT unfolding of $f(\boldsymbol{r})$ (blue line) and of the TT unfoldings of $f_1(\boldsymbol{r})$ and $f_2(\boldsymbol{r})$ in Eq. (2.26), limited to their respective support. The respective Tensor Trains $\widetilde{\mathcal{F}}_{\boldsymbol{\sigma}}$, $\widetilde{\mathcal{F}}_{1\boldsymbol{\sigma}}$ and $\widetilde{\mathcal{F}}_{2\boldsymbol{\sigma}}$ are obtained through QTCI.

The rank χ^+ of the direct sum of MPSs – $\widetilde{\mathcal{F}}^+_{\boldsymbol{\sigma}}$ – in Eq. (2.28), depends only linearly on the bond dimension of its addends, *i.e.*

$$\chi^+ = \chi_1 + \chi_2, \tag{2.29}$$

where χ_1, χ_2 are the ranks of $\widetilde{\mathcal{F}}_{1\boldsymbol{\sigma}}$ and $\widetilde{\mathcal{F}}_{2\boldsymbol{\sigma}}$, respectively.

Since f_1 and f_2 are resolved at markedly different scales, combining them still yields a pronounced drop in the overall bond dimension, despite the extra ranks introduced during the summation that produces the final tensor; as evidence of this, the total number of floating point parameters necessary for the two different implementations is

$$N_{\text{par}}(\widetilde{\mathcal{F}}_{\boldsymbol{\sigma}}) \simeq 1.4 \times 10^6 > N_{\text{par}}(\widetilde{\mathcal{F}}_{2\boldsymbol{\sigma}} + \widetilde{\mathcal{F}}_{1\boldsymbol{\sigma}}) \simeq 5.6 \times 10^5. \tag{2.30}$$

Chapter 3

Patched QTCI

We've come now to the original part of this manuscript. In Chap. 2 we introduced the state-of-the art implementation of the (Quantics) Tensor Cross Interpolation algorithm. We have briefly illustrated the capabilities of this numerical method through several simple examples, which have also been demonstrated in a more thorough manner in other studies [1, 18, 21–23, 28, 31]. Nevertheless, the final section of the previous chapter highlighted a shortcoming of the standard implementation, particularly the approximation of multivariate functions with sharp local features.

Here we present a more structured solution to these shortcomings than the one proposed in Sec. 2.2 above, based on a *"divide and conquer"* variant of the standard QTCI. Similar d.&c. approaches have already been investigated in other tensor-focused works, including the use of an SVD-based Quantics Tensor Train (QTT) method for approximating many-body correlation functions [27, 41], and tensor compression techniques employing standard and High-Order Singular Value Decomposition (HOSVD) within adaptive or greedy frameworks [26, 45, 46]. Our novel implementation of this method – "standing on the shoulder of these giants" – exploits the advantages of TCI and performs TCI tensor compression and function approximation in a *patched* way.

This new version of the TCI algorithm, which we will refer to as *patched (Quantics) Tensor Cross Interpolation* – or *patched (Q)TCI* – distributes the workload of the cross approximation by adaptively splitting the configuration domain and uses the rank of the temporary TT cross approximation as trigger parameter for the splitting. We will show that this *patching* technique helps reduce the computational demand of TCI when attempting to represent narrow peaked, multi-dimensional functions, both in memory requirements and CPU times.

The upcoming chapter will be structured as follows: Sec. 3.1 covers all the technical and mathematical details about the patched QTCI routine and Sec. 3.2 discusses the estimated scaling of computational resources for the algorithm, while also adressing some its weaknesses.

3.1 The algorithm

Patched Quantics Tensor Cross Interpolation (pQTCI) is an adaptive partitioning method [47] based on rank-revealing cross approximation for high-

dimensional tensors. Cross interpolation represents only a subroutine for our algorithm, hence, we will assume that the reader is familiar with all TCI's functionalities (cf. Chap. 2 and Ref. [1] otherwise). Since it is built upon the TCI framework, pQTCI also falls within the class of *rank-revealing* algorithms introduced in Def. 2.2.

Given a tensor $\mathcal{T}$ with hidden low-rank structure and desirably – but not necessarily – very localized features in configurational space (*i.e.* only for a small subset of $\boldsymbol{\sigma}$, $\mathcal{T}_{\boldsymbol{\sigma}}$ is non-trivial) as input, pQTCI returns a *collection* of TT unfoldings as output, that, when resummed, approximate the tensor $\mathcal{T}$ on its whole domain. Let us dive into the prerequisites needed for the construction of pQTCI.

Element-wise addition of TTs

An key component of the pQTCI algorithm is the TT summation operation. Consider the following definition

Definition 3.1 (Direct sum) The **direct sum** of N-dimensional tensors $X_{\boldsymbol{\sigma}} \in \mathbb{K}^{d_1 \times d_2 \times \cdots d_N}$ and $Y_{\boldsymbol{\sigma}'} \in \mathbb{K}^{d'_1 \times d'_2 \times \cdots d'_N}$ is defined by

$$Z_{\boldsymbol{\sigma}''} = X_{\boldsymbol{\sigma}} \oplus Y_{\boldsymbol{\sigma}'} \in \mathbb{R}^{(d_1+d'_1)\times(d_2+d'_2)\times\cdots(d_N+d'_N)} \tag{3.1}$$

with entries

$$Z_{\sigma''_1,\ldots,\sigma''_N} = \begin{cases} X_{\sigma''_1,\ldots,\sigma''_N} & \text{if } 1 \leq \sigma''_n \leq d_n \ \forall n \\ Y_{\sigma''_1-d_1,\ldots,\sigma''_N-d_N} & \text{if } d_n+1 \leq \sigma''_n \leq d_n+d'_n \ \forall n \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

A special case of the direct sum of two tensors is the so-called *partial direct sum* $\boxplus$ [44]. A partial direct sum of two tensors $X_{\boldsymbol{\sigma}} \in \mathbb{K}^{d_1 \times d_2 \times \cdots d_n \times d_{n+1} \times \cdots d_N}$ and $Y_{\boldsymbol{\sigma}'} \in \mathbb{K}^{d'_1 \times d'_2 \times \cdots d'_n \times d_{n+1} \times \cdots d_N}$, that have the same dimensions for their last $N-n$ indices $\sigma_{n+1}, \cdots, \sigma_N$, is defined as

$$Z_{\boldsymbol{\sigma}''} = X_{\boldsymbol{\sigma}} \boxplus Y_{\boldsymbol{\sigma}'} \in \mathbb{K}^{(d_1+d'_1)\times(d_2+d'_2)\times\cdots(d_n+d'_n)\times d_{n+1}\times\cdots d_N} \tag{3.3}$$

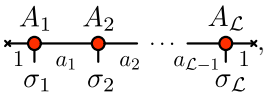
where $Z_{\boldsymbol{\sigma}''}$, by means of the slicing operation introduced in Eq. (2.11), has subtensors

$$Z_{(i''_n \oplus j_{n+1})} = X_{(i_n \oplus j_{n+1})} \oplus Y_{(i'_n \oplus j_{n+1})} \quad \forall j_{n+1} \in \mathbb{J}_{n+1} \text{ fixed.} \tag{3.4}$$

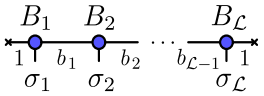
The partial direct sum of $X_{\boldsymbol{\sigma}}$ and $Y_{\boldsymbol{\sigma}}$ is a direct sum performed pairwise between all the subtensors of the two addends, obtained from fixing the indices not included in the sum operation to the same value in both X and Y.

Given now two $\mathcal{L}$-dimensional tensor trains, $\widetilde{A}_{\boldsymbol{\sigma}}$ and $\widetilde{B}_{\boldsymbol{\sigma}}$[1],

[1] We use $\widetilde{\ }$ to refer to a generic MPS or TT unfolding (cf. Chap. 2).

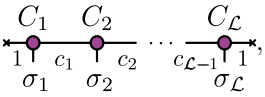
$$\widetilde{A}_{\boldsymbol{\sigma}} = A_1 A_2 \cdots A_{\mathcal{L}} =$$ 

, (3.5)

$$\widetilde{B}_{\boldsymbol{\sigma}} = B_1 B_2 \cdots B_{\mathcal{L}} =$$ 

(3.6)

with identical phyisical dimensions, $d_\ell^A = d_\ell^B \ \forall \ell$, but possibly different virtual dimensions χ_ℓ^A, χ_ℓ^B. Their element-wise sum $\widetilde{C}_{\boldsymbol{\sigma}} = \widetilde{A}_{\boldsymbol{\sigma}} + \widetilde{B}_{\boldsymbol{\sigma}}$ is still a tensor trained shaped as

$$\widetilde{C}_{\boldsymbol{\sigma}} = \big[A_1 \boxplus B_1\big]\big[A_2 \boxplus B_2\big] \cdots \big[A_{\mathcal{L}} \boxplus B_{\mathcal{L}}\big] =$$ 

, (3.7)

where the partial sum between A's and B's sitetensors is performed by fixing the sites indices σ_ℓ, and results in

$$C_1^{\sigma_1} = \begin{bmatrix} A_1^{\sigma_1} & B_1^{\sigma_1} \end{bmatrix}, \quad C_\ell^{\sigma_\ell} = \begin{bmatrix} A_\ell^{\sigma_\ell} & 0 \\ 0 & B_\ell^{\sigma_\ell} \end{bmatrix}, \quad C_{\mathcal{L}}^{\sigma_{\mathcal{L}}} = \begin{bmatrix} A_{\mathcal{L}}^{\sigma_{\mathcal{L}}} \\ B_{\mathcal{L}}^{\sigma_{\mathcal{L}}} \end{bmatrix}. \tag{3.8}$$

The new internal bonds c_ℓ are dependent on the original bonds, in fact $a_\ell = c_\ell$, for $c_\ell \in \{1, \chi_\ell^A\}$, and $b_\ell = c_\ell - \chi_\ell^A$, for $c_\ell \in \{\chi_\ell^A + 1, \chi_\ell^A + \chi_\ell^B\}$. Hence, the bond dimensions[2] of the sum are also additive *w.r.t* to the bond dimensions of its addends

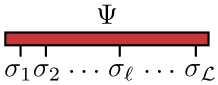
$$\chi_\ell^C = \chi_\ell^A + \chi_\ell^B. \tag{3.9}$$

Tensor subdomain projection

The second requirement needed to construct patched QTCI is, what we will refer to as, the *subdomain projection* operation.

Consider a $\mathcal{L}$-dimensional tensor $\Psi_{\boldsymbol{\sigma}}$; $\Psi_{\boldsymbol{\sigma}}$ can be interpreted as the collection of coefficients of a generic quantum state $|\Psi\rangle$, when represented on a specific basis of the Hilbert space it belongs to, as

$$|\Psi\rangle = \Psi_{\boldsymbol{\sigma}} |\boldsymbol{\sigma}\rangle = |\sigma_1\rangle |\sigma_2\rangle \dots |\sigma_{\mathcal{L}}\rangle \, \Psi_{\sigma_1, \sigma_2, \dots, \sigma_{\mathcal{L}}} \tag{3.10}$$

$$= \Psi$$ 

[3].

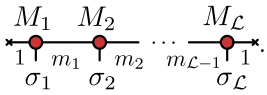
In the same way, its MPS unraveling

[2]The bond dimension at site ℓ ($1 \leq \ell < \mathcal{L}$) is defined as the number of columns of the ℓ^{th} site tensor, when σ_ℓ is fixed to a specific value.

[3]By abuse of notation we identify the tensor $\Psi_{\boldsymbol{\sigma}}$ with the quantum state $|\Psi\rangle$.

$$|\Psi\rangle = \widetilde{\Psi}_{\boldsymbol{\sigma}} |\boldsymbol{\sigma}\rangle = |\sigma_1\rangle [M_1^{\sigma_1}]_{1m_1} |\sigma_2\rangle [M_2^{\sigma_2}]_{m_1 m_2} \cdots |\sigma_{\mathcal{L}}\rangle [M_{\mathcal{L}}^{\sigma_{\mathcal{L}}}]_{m_{\mathcal{L}-1}1} \tag{3.11}$$

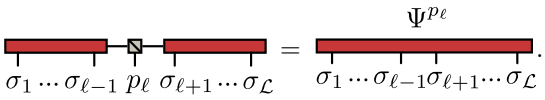
$$= \quad .$$



If we restrict the configuration space to a subset space where $|\sigma_\ell\rangle$ is fixed to a specific basis vector $|p_\ell\rangle$, the corresponding subtensor representation of such a subspace state $|\Psi^{p_\ell}\rangle$ would be

$$\begin{aligned} |\Psi^{p_\ell}\rangle &= |\sigma_1\rangle \cdots |\sigma_{\ell-1}\rangle |p_\ell\rangle |\sigma_{\ell+1}\rangle \cdots |\sigma_{\mathcal{L}}\rangle \Psi_{\sigma_1,\ldots,\sigma_{\ell-1},p_\ell,,\sigma_{\ell+1},\ldots,\sigma_{\mathcal{L}}} \\ &= |\sigma_1\rangle \cdots |\sigma_{\ell-1}\rangle |p_\ell\rangle |\sigma_{\ell+1}\rangle , \cdots |\sigma_{\mathcal{L}}\rangle \Psi^{p_\ell}_{\sigma_1,\ldots,\sigma_{\ell-1},\sigma_{\ell+1},\ldots,\sigma_{\mathcal{L}}} \end{aligned} \tag{3.12}$$

$$= \quad = \quad .$$

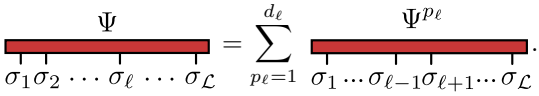


This procedure resembles a quantum projection, *i.e.* $|\Psi^{p_\ell}\rangle = |p_\ell\rangle \langle p_\ell|\Psi\rangle$. For now, the object is only used as a placeholder to relate our subtensor of interest $\Psi^{p_\ell}_{\sigma_1,\ldots,\sigma_{\ell-1},\sigma_{\ell+1},\ldots,\sigma_{\mathcal{L}}}$ to the original configuration space of the full tensor $\Psi_{\boldsymbol{\sigma}}$, but has otherwise no other meaning. Indeed, within the $|p_\ell\rangle$-subspace, $|\Psi^{p_\ell}\rangle$ is completely defined by the $\mathcal{L}-1$ indices $(\sigma_1, \ldots, \sigma_{\ell-1}, \sigma_{\ell+1}, \ldots, \sigma_{\mathcal{L}})$. The items act as *slicing* or *projection* operators that limit us to the subsector of $\Psi_{\boldsymbol{\sigma}}$ where the ℓ^{th} index is fixed to p_ℓ. It is trivial to deduce that

$$|\Psi\rangle = \sum_{p_\ell=1}^{d_\ell} |\Psi^{p_\ell}\rangle , \tag{3.13}$$

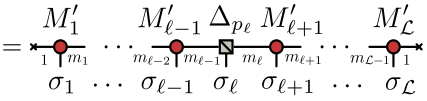
and, thus,

$$\quad = \sum_{p_\ell=1}^{d_\ell} \quad . \tag{3.14}$$



Inside a specific $|p\rangle$-subspace we can perform a TT approximation of the subtensor Ψ^p as follows

$$\widetilde{\Psi}^{p_\ell} = \quad \tag{3.15}$$

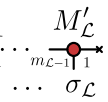


wherein we generalized to be local state dependent in the following manner

$$\begin{matrix} \Delta_{p_\ell} \\ \\ \sigma_\ell \end{matrix} = \begin{cases} [\mathbb{1}]_{m_{\ell-1},m_\ell} & \text{if } \sigma_\ell = p_\ell \\ [0]_{m_{\ell-1},m_\ell} & \text{otherwise,} \end{cases} \tag{3.16}$$

with bonds $m_{\ell-1}$ and m_ℓ of same dimension and where $0_{m,n}$ denotes the element (m,n) of the zero matrix of dimension $\chi_{\ell-1} \times \chi_\ell$. From Eq. (3.14) we can recognize that

$$\Psi_{\boldsymbol{\sigma}} \simeq \sum_{p_\ell=1}^{d_\ell} \widetilde{\Psi}^{p_\ell} = \sum_{p_\ell=1}^{d_\ell} \quad \underset{\sigma_1}{M'_1} \quad \cdots \quad \underset{\sigma_{\ell-1}}{M'_{\ell-1}} \quad \underset{\sigma_\ell}{\Delta_{p_\ell}} \quad \underset{\sigma_{\ell+1}}{M'_{\ell+1}} \quad \cdots \quad \underset{\sigma_{\mathcal{L}}}{M'_{\mathcal{L}}} \tag{3.17}$$



where the first approximation reflects the error associated with an arbitrary MPS approximation and the sum is performed according to the element-wise TT addition described by Eq. (3.7) in the section above. Without loss of generality, the same line of reasoning can be applied to smaller subspaces

$$\Psi_{\boldsymbol{\sigma}} \simeq \widetilde{\Psi}^{+}_{\boldsymbol{\sigma}} = \sum_{p_{\ell_1}=1}^{d_{\ell_1}} \cdots \sum_{p_{\ell_N}=1}^{d_{\ell_N}} \widetilde{\Psi}^{p_{\ell_1},\ldots,p_{\ell_N}} \tag{3.18}$$

taking $(\ell_1, \ldots, \ell_N) \in \{1, \ldots, \mathcal{L}\}^{N(\leq \mathcal{L})}$ ($\ell_i \neq \ell_j$ for every $i \neq j$) without any specific order constraint. Whenever the sum Eq. (3.18) retains a low rank, after recompression, $\widetilde{\Psi}^{+}_{\boldsymbol{\sigma}}$ represents good approximation of the original tensor $\Psi_{\boldsymbol{\sigma}}$.

The Patching scheme

Having laid out all the necessary ingredients, we now return to detailing our patched (Q)TCI algorithm. Consider a generic tensor $\mathcal{T}_{\boldsymbol{\sigma}}$ and its TCI approximation $\widetilde{\mathcal{T}}_{\boldsymbol{\sigma}}$. As noted in Sec. 2.2, when the tensor $\mathcal{T}$ exhibits sharply localised structures, its compression $\widetilde{\mathcal{T}}$ can be made more memory-efficient by adopting a *divide-et-impera* approach. Nevertheless, although the naive domain-splitting approach outlined in Sec. 2.2 proved functional, it provided no means of numerical control and presumed that both the location and spatial extent of the tensor's relevant features were known a priori. Such ideal conditions are rarely encountered in real-world applications, particularly when the tensor $\mathcal{T}$ stems from complex computations that estimate unknown variables. However, the preceding example shows that memory usage for a (local) TT approximation is effectively characterised by its bond dimensions χ_ℓ—specifically, by the maximum bond dimension χ. Recognizing this allows us to devise a more intelligent, adaptive, and general solution to the compression of tensors presenting sharp localized features.

The standard TCI implementation provided by `TCI.jl` [24], as illustrated in Listing 2.1, offers a mechanism to control the bond dimension of the approximation. Specifically, setting the input variable `maxbonddim` within `crossinterpolate2` constrains the size of the pivot lists, thereby restricting the bond dimensions χ_ℓ of the resulting MPS. Although this strategy effectively reduces the memory footprint of the TCI approximation, it also hampers the convergence of the algorithm. Consequently, to ensure convergence, the approximation domain must be suitably reduced.

We partition the approximation domain through *subdomain projection* operation as detailed in the previous section. Once the original tensor has been

decomposed into subtensors, hereafter called *patches* [4], we run TCI on each patch separately. Working on these reduced domains improves the likelihood of convergence since each patch explores a much smaller configuration space, even when the bond dimension is capped at the prescribed threshold χ_{patch}, a bound that will then not be reached. This whole procedure can be repeated iteratively and adaptively, as shown in Fig. 3.1. We shall refer to this routine as the *patching scheme*, or simply *patched TCI*. The algorithm proceeds as follows:

(1) Start the standard TCI routine on the full tensor $\mathcal{T}_{\boldsymbol{\sigma}}$ either from

- a random configuration $\hat{\sigma}$ or
- a tailored set of global pivots, if prior information about local features of $\mathcal{T}$ is available.

The rank reveal of the tensor train $\widetilde{\mathcal{T}}$ is constrained by the prescribed limit χ_{patch}. If $\widetilde{\mathcal{T}}_{\boldsymbol{\sigma}}$ converges, the algorithm terminates.

(2) If convergence is not reached, slice $\mathcal{T}_{\boldsymbol{\sigma}}$ along its first index[5] to produce subtensors

$$\mathcal{T}^{p_1} \quad \forall p_1 \in \{1, \dots, d_1\} \quad \text{(cf. Eq. (3.12))}.$$

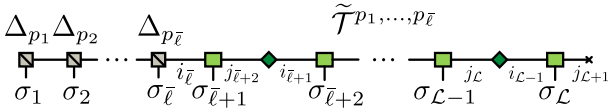
(3) Use the pivot lists $\mathcal{I}_\ell$ and $\mathcal{J}_\ell$ from the preceding (failed) TT to form new global pivots: $\hat{\sigma} = i_\ell \oplus j_{\ell+1}$, with $i_\ell \in \mathcal{I}_\ell$, $j_{\ell+1} \in \mathcal{J}_{\ell+1}$. A pivot $\hat{\sigma}$ is assigned to subtensor $\mathcal{T}^{p_1}$ whenever its first component satisfies $\hat{\sigma}_1 = p_1$.

(4) Compress each subtensor $\mathcal{T}^{p_1}$ with TCI (bond dimension is still capped at χ_{patch}) over the reduced cofiguration domain $(\sigma_2, \dots, \sigma_{\mathcal{L}})$. Store all the converged *patches* $\widetilde{\mathcal{T}}^{p_1}$; discard those that fail to converge.

(5) For every unconverged subtensor $\mathcal{T}^{p_1}$, slice further along the next index to obtain $\mathcal{T}^{p_1,p_2}$ $\forall p_2 \in 1, \dots, d_2$. Build the next set of global pivots subject to $\hat{\sigma}_1 = p_1$ and $\hat{\sigma}_2 = p_2$.

(6) Repeat (4)-(5), recursively increasing the slicing depth, until no *patches* remain to be converged.

Remarks are in order. The output result of the algorithm seems to be a collection of tensor trains of the form



(3.19)

where both the *prefix* indices $(p_1, \dots, p_{\bar{\ell}}) \in \mathbb{I}_{\bar{\ell}}$ and the length of the prefix – the *paching level* – $\bar{\ell} \in \{1, \dots, \mathcal{L}\}$ can take very different values. Nevertheless, two conditions are respected from our implementation:

[4] Henceforth, the term *patch* will denote both the subtensor $\mathcal{T}^{p_1,\dots,p_\ell}$ and its TT-unfolded form $\widetilde{\mathcal{T}}^{p_1,\dots,p_\ell}$.

[5] A different choice for the starting index is possible, see below. For pedagogical purposes we start from σ_1.

a)

$$\texttt{isconverged}(\widetilde{\mathcal{T}}^{(\cdot)}) = \begin{cases} \chi < \chi_{\text{patch}} \wedge \ \|\mathcal{T}^{(\cdot)} - \widetilde{\mathcal{T}}^{(\cdot)}\|_\infty < \tau & \texttt{true} \\ \text{otherwise} & \texttt{false} \end{cases}$$

$$\texttt{results} = \left\{\widetilde{\mathcal{T}}^{(\cdot)} | \texttt{isconverged}(\widetilde{\mathcal{T}}^{(\cdot)}) \texttt{ = true}\right\}$$

$$\texttt{tasks} = \left\{\widetilde{\mathcal{T}}^{(\cdot)} | \texttt{isconverged}(\widetilde{\mathcal{T}}^{(\cdot)}) \texttt{ = false}\right\}$$

b)

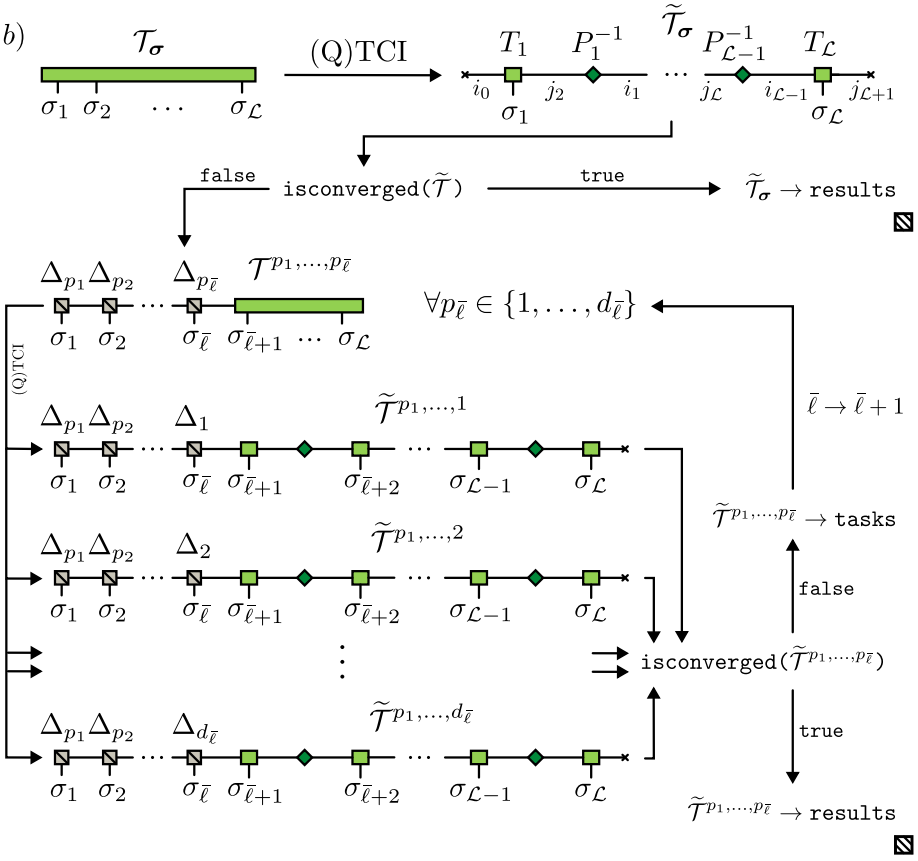


Figure 3.1: Flowchart of the patched TCI algorithm. (*a*) Convergence criterion for the subtensors $\widetilde{\mathcal{T}}^{p_1,\ldots,p_{\bar{\ell}}}$ – or *patches* – defined by the parameters χ_{patch} and τ, and the two sets of subtensors that have already converged – `results` – and those yet to converge – `tasks`. (*b*) Flowchart of the *patching scheme*. The TCI approximation is adaptively decomposed into smaller computations through *slicing* (cf. Eq. (3.12)). Each subtensor is TCI compressed within the smaller domain. The converged *patches* are added to `results`, the yet-to-converge ones to `tasks`. The algorithm terminates – ▧ – when `tasks` is empty. We refer the reader to the main manuscript for additional details.

- any subtensor that is further subdivided has its non-convergent TT approximation discarded, all pairs of index prefixes satisfy

$$(p_1, \dots, p_{\bar{\ell}_1}) \nsubseteq (p_1, \dots, p_{\bar{\ell}_2}) \quad \forall\, \bar{\ell}_1 \neq \bar{\ell}_2 \tag{3.20}$$

  Consequently, the patched-TCI procedure produces TT approximations $\widetilde{\mathcal{T}}^{p_1,\dots,p_{\bar{\ell}}}$ that are strictly *non-overlapping* (*disjointed*) in configuration space;

- the collection of TTs of the form in Eq. (3.19), in order to constitute a good approximation for the full tensor $\mathcal{T}_{\boldsymbol{\sigma}}$, satisfies the condition

$$\mathcal{T}_{\boldsymbol{\sigma}} \approx \sum_{\widetilde{T}^{(\cdot)} \in \texttt{results}} \widetilde{T}^{(\cdot)}. \tag{3.21}$$

  and we can therefore resum them to a single tensor train $\widetilde{\mathcal{T}}^{+}_{\boldsymbol{\sigma}}$, similar to what we did in Eq. (3.18). $\widetilde{\mathcal{T}}^{+}_{\boldsymbol{\sigma}}$ is a good TT representation of $\mathcal{T}$. However due to the adaptivity of pQTCI, differently from Eq. (3.18), here we might be presented with a series of *patches* $\widetilde{\mathcal{T}}^{p_1,\dots,p_{\bar{\ell}}}$ with prefixes $(p_1, \dots, p_{\bar{\ell}})$ of different lengths $\bar{\ell}$; nonetheless, this intricacy does not affect the TT sum operation.

In Fig. 3.1 we displayed a version of the patched TCI where the tensor is sliced sequentially starting from the first index σ_1 of the tensor $\mathcal{T}$. As pointed out already, this constraint is not required to obtain a meaningful outcome from the algorithm. The *prefix* of each subtensor $\mathcal{T}^{p_1,\dots,p_{\bar{\ell}}}$ – that indicates to which slice of $\mathcal{T}$ it corresponds to – can be taken randomly, in an exclusive manner, from the set of tensor indices $\{\sigma_1, \dots, \sigma_{\mathcal{L}}\}$ (considered labels in this context).

Let us consider now the specific case when we intend to TT approximate a multi-variate function, call it $f(\boldsymbol{x})$. Sec. 2.1 taught us that, when we intend to properly resolve all the relevant features of f, it is a clever choice to numerically represent it as a tensor utilizing the quantics discretization (with $\mathcal{R}$ bits per spatial dimention x_i): in fact, the discretization error of the approximation decreases exponentially by increasing $\mathcal{R}$ only linearly.

Assume now that f is characterized by narrow peaks sparsely distributed across its domain. Its tensor representation $\mathcal{F}_{\boldsymbol{\sigma}} = f(\boldsymbol{x}(\boldsymbol{\sigma}))$ inherits the same structure. When we apply the patched TCI procedure to $\mathcal{F}_{\boldsymbol{\sigma}}$, the scale separation introduced by the quantics format (where $\sigma_{\ell(n,r)} \to 2^{-r}$ length scale) implies that each patch $\widetilde{\mathcal{F}}^{p_1,\dots,p_{\bar{\ell}}}$ simply captures $f(\boldsymbol{x}(\boldsymbol{\sigma}))$ restricted to a distinct sub-region of the original domain. Fig. 3.2 illustrates this idea for a bivariate function. The tensor $\mathcal{F}$ is indexed by scale: for example, with an interleaved 2-D ordering, the first two indices σ_1 and σ_2 encode the resolution 2^{-1} in the x- and y-directions. Fixing these two indices—i.e., taking the corresponding slice—yields a smaller subtensor that represents f on one quarter of the domain (see the second panel on the right in Fig. 3.2). This motivates our previous terminology: we call each such subtensor slice a *patch*.

Patched QTCI refines the domain more aggressively in regions where the target function f exhibits higher complexity. Areas that contain sharp, localised

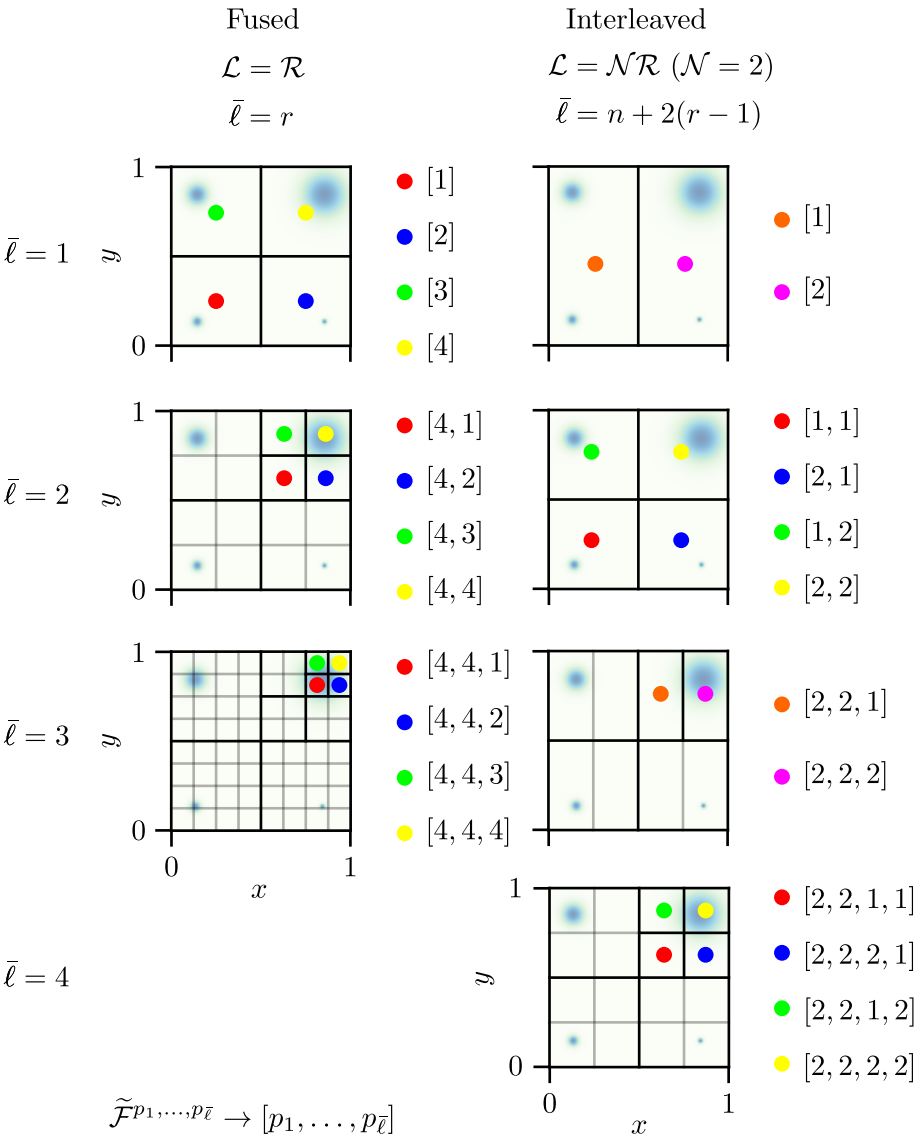


Figure 3.2: Subdivision of a two-dimensional domain caused by the QTCI patching scheme. For clarity, we label each patch $\widetilde{\mathcal{F}}^{p_1,\ldots,p_{\bar{\ell}}}$ by its prefix $[p_1, \ldots, p_{\bar{\ell}}]$. We consider both fused and interleaved index ordering for the TTs $\widetilde{\mathcal{F}}^{p_1,\ldots,p_{\bar{\ell}}}$. Coloured dots show the region represented by each patch, and only selected patches are displayed up to a *patching level* $\bar{\ell} = 4$. Faded grid lines suggest how the remaining parts of the domain could be further split whenever the adaptive-convergence criterion of the patched QTCI algorithm calls for it.

peaks incur a larger memory footprint, driving up the tensor-train rank χ. Panel (a) of Fig. 3.3 illustrates this effect for a generic, single-peaked bivariate function. As noted earlier, patched TCI returns a collection of TT representations that

are mutually disjoint in configuration space; in the context of pQTCI this means that the resulting set of patches forms a non-overlapping cover of the entire domain of f. Panel (b) shows how this cover is built: the algorithm traverses the patch tree and, for each slice $\widetilde{\mathcal{F}}^{p_1,\ldots,p_{\bar{\ell}}}$, checks the stopping criteria:

$$\|\mathcal{F}^{(\cdot)} - \widetilde{\mathcal{F}}^{(\cdot)}\|_\infty < \tau \quad \text{and} \quad \chi < \chi_{\text{patch}}. \tag{3.22}$$

adaptively refining only those slices that violate either boundary. The total number of red colored leaves in Fig. 3.3 is the number of patches, N_{patch}, for a specific approximation.

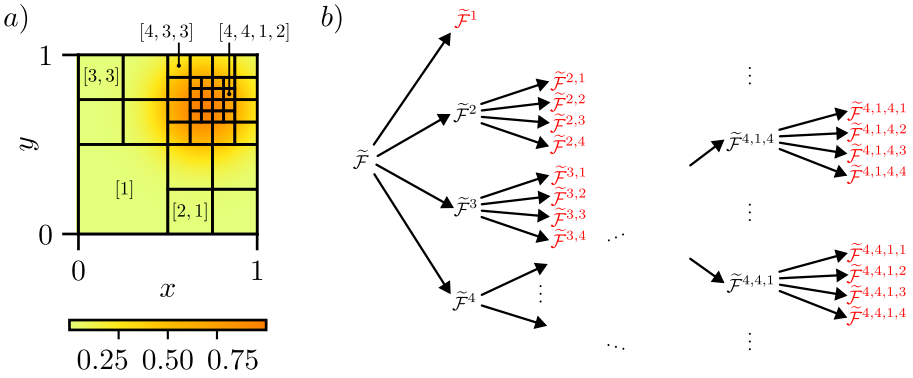


Figure 3.3: Patched QTCI applied to the TT approximation of a bivariate function. a) The domain is adaptively split, yielding finer cells where the function is more intricate. b) Hierarchical tree that records the patch refinement: each tensor slice is attached to its parent, and the red leaves mark the final subtensors produced and kept by pQTCI.

3.2 Computational Costs and Scaling

In many practical settings, applying standard Tensor Cross Interpolation (TCI) directly to large, feature-sparse tensors is wasteful: a lot of the computational effort is spent on regions that do not influence the final accuracy. Moreover, since all TCI algorithms involve sampling, none of them is fully immune against missing some features of the tensor of interest, as already discussed above. Patched QTCI addresses these issues by adaptively dividing the tensor into small, targeted patches and capping the bond dimension in each local solve. The goal is to concentrate resources only where the data truly demands high resolution, while discovering the interesting regions during the process, thereby reducing both memory traffic and run-time.

At first sight, the patched (Q)TCI algorithm shown in Fig. 3.1 appears to burden the original TCI routine with considerable overhead: in the worst-case, `crossinterpolate2` is invoked on the order of $\mathcal{O}(d^{\bar{\ell}})$ times, where $\bar{\ell}$ is the deepest patching level reached (assuming a uniform subdivision and ignoring adaptivity).

In practice, however, patched QTCI can run faster than the plain TCI workflow. Every call to `crossinterpolate2` is restricted by the bond-dimension cap χ_{patch}, so each χ_{patch}-capped local TCI is far cheaper than a non-capped TCI on the entire tensor $\mathcal{T}$. Only when we reach patch spatial dimension containing features we are able to approximate within a bond dimension of χ_{patch} then the TCI procedure runs to its full extent, reaching convergence.

The preceding discussion makes it clear that both the total number of resulting patches, N_{patch}, and relative computational resources expenditure, are governed largely by the bond-dimension cap assigned to each patch, χ_{patch}.

Runtime & Memory vs. χ_{patch}

To estimate actual resource usage of pQTCI, we begin with a theoretical analysis (empirical fits are in Appendix A). Consider a generic $\widetilde{\mathcal{T}}$ of the form given in Eq. (3.11) whose maximum bond dimension is χ. When $\widetilde{\mathcal{T}}$ is constructed by successive SVD unfoldings of the full tensor $\mathcal{T}$, truncating each SVD at the rank χ [2, 6, 7], the bond dimension along the chain typically evolves as illustrated in Fig. 3.4.

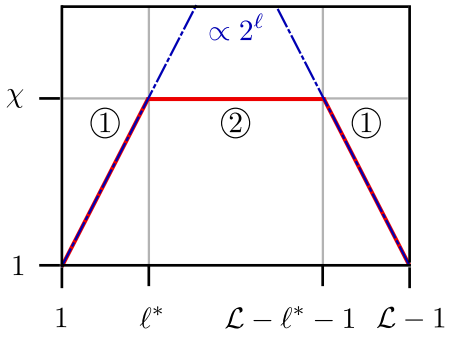


Figure 3.4: Generic χ truncated MPS bond dimension evolution.

A convenient way to gauge the memory footprint of a tensor-train approximation is to tally the total number of floating-point entries it contains. For each of the two TT splittings labelled ① and ② in Fig. 3.4 – assuming all physical dimensions satisfy $d_\ell = d$ – the parameter count is therefore

$$\begin{aligned} N_{\text{par}}^{①} &= 2\sum_{\ell=1}^{\ell^*} d^{2\ell} \\ N_{\text{par}}^{②} &= \sum_{\ell=\ell^*+1}^{\mathcal{L}-\ell^*-1} \chi^2 d \end{aligned} \tag{3.23}$$

where ℓ^* is fixed by the relation $d^{2\ell^*} = \chi$. Hence, the total number of parameters for a generic TT of this form is bounded by

$$N_{\text{par}} \lesssim 2d^2\frac{1-\chi^2}{1-d^2} + \chi^2 d(\mathcal{L} - 2\log_d \chi - 2). \tag{3.24}$$

Assume the worst-case scenario in which patched QTCI subdivides the entire domain into a lattice of patches. After sequential slicing ($\sigma_1 \to \sigma_2 \to \cdots$), the procedure halts at *patching level* $\bar{\ell}$, yielding a total number of patches N_{patch} all with rank χ_{patch} and with bond dimension development as in Fig. 3.4. The memory footprint of this final object would be

$$\begin{aligned} N_{\text{par}} &= N_{\text{patch}}\, N_{\text{par}\times\text{patch}} \\ N_{\text{par}\times\text{patch}} &\lesssim 2d^2 \frac{1-\chi_{\text{patch}}^2}{1-d^2} + \chi_{\text{patch}}^2 d(\mathcal{L} - \bar{\ell} - 2\log_d \chi_{\text{patch}} - 2). \end{aligned} \tag{3.25}$$

As in standard TCI (see Tab. 2.1), the memory footprint of patched QTCI grows with the square of the maximum bond dimension — here evaluated per patch — scaling as

$$\mathcal{O}\big(N_{\text{patch}}\chi_{\text{patch}}^2 d^2 \mathcal{L}\big). \tag{3.26}$$

A comparable reasoning yields an estimate for the runtime of the patched QTCI routine. If a single, full-domain TCI run takes $\mathcal{O}\big(\chi^3 d^2 \mathcal{L}\big)$ CPU time, then the cumulative runtime of patched QTCI becomes

$$t_{\text{patch}} \lesssim \chi_{\text{patch}}^3 d^2 \sum_{\ell=0}^{\bar{\ell}} d^\ell (\mathcal{L} - \ell) \lesssim N_{\text{patch}}\chi_{\text{patch}}^3 d^3 \mathcal{L}, \tag{3.27}$$

where we assumed $\mathcal{L} \gg \bar{\ell}$ in the last inequality. This yields a runtime scaling for pQTCI of

$$\mathcal{O}\big(N_{\text{patch}}\chi_{\text{patch}}^3 d^3 \mathcal{L}\big). \tag{3.28}$$

Beyond the raw cost estimates, the analysis yields some useful thresholds: the patched scheme remains more memory- or time-efficient than standard TCI only if the total number of patches, N_{patch}, stays below the corresponding bounds:

$$N_{\text{patch}} \;<\; \begin{array}{|c|c|}\hline \textbf{memory cost} & \textbf{CPU runtime} \\ \hline \dfrac{\chi^2}{\chi_{\text{patch}}^2} & \dfrac{\chi^3}{d\chi_{\text{patch}}^3} \\ \hline \end{array} \tag{3.29}$$

The table above provides us with rough boundaries on N_{patch} for an optimal pQTCI approximation. In this particular context, χ is the rank of an analogous (Q)TCI compression with the same input tensor of p(Q)TCI

Number of patches vs. χ_{patch}

To explore the dependence of N_{patch} by the fixed parameter χ_{patch}, let us examine a concrete example, in particular let us revisit the function of Eq. (2.26) with the following parameter changes (purely for visualization purposes)

$$\begin{gathered} \boldsymbol{r}_1 = (0.2, 0.2), \quad \boldsymbol{r}_2 = (0.8, 0.8), \\ A_2 = 2\,A_1 = 10^4, \quad \sigma_1 = 2\,\sigma_2 = 10^{-1}, \quad k_1 = 2\,k_2 = 10^3. \end{gathered} \tag{3.30}$$

We restate the formula here

$$f(\boldsymbol{r}) = A_1\, e^{-\frac{(\boldsymbol{r}-\boldsymbol{r}_1)^2}{2\sigma_1^2}} \sin(k_1 \boldsymbol{r}) + A_2\, e^{-\frac{(\boldsymbol{r}-\boldsymbol{r}_2)^2}{2\sigma_2^2}} \sin(k_2 \boldsymbol{r}).$$

We discretize f on the square domain $[0,1]^2$ into the tensor $\mathcal{F}_{\boldsymbol{\sigma}} = f(\boldsymbol{r}(\boldsymbol{\sigma}))$, using interleaved or fused quantics representation. Applying the patched QTCI compression to $\mathcal{F}$, while varying the bond-dimension cap χ_{patch}, let us track how the domain is adaptively partitioned and how the overall patch count N_{patch} changes with χ_{patch}. Lowering the bond-dimension cap χ_{patch} in the patched QTCI algorithm forces each patch to converge on a progressively smaller portion of the configuration space. This behaviour is illustrated in Fig. 3.5.

In Fig. 3.5 each coordinate is discretised with $\mathcal{R} = 17$ bits–enough to resolve every feature of $f(\boldsymbol{r})$ f to a tolerance of $\tau = 10^{-7}$. The total patch count N_{patch} depends sensitively on the chosen bond-dimension cap χ_{patch}. We can then understand that there exists a dependence of N_{patch} by χ_{patch}. From the memory estimate in Eq. (3.25), and assuming the overall parameter budget of the final approximation to stay roughly constant, we anticipate the following scaling law

$$N_{\text{patch}} \propto 1/\chi_{\text{patch}}^2. \tag{3.31}$$

In other words, halving the allowable bond dimension per patch would quadruple the number of patches required to achieve the same accuracy. Fig. 3.6 corroborates this scaling: it plots the patch count against the actual maximum bond dimension, χ^*_{patch}[6], attained by the most demanding patch in the pQTCI compression of $\mathcal{F}_{\boldsymbol{\sigma}}$.

"Over-patching"

The bond-dimension cap χ_{patch} is pivotal to the efficiency of patched QTCI: it must be selected so that the resulting approximation is truly more economical than a direct (Q)TCI compression. The bounds in Eq. (3.29) supply an initial guideline, but they rely on knowing—or at least estimating—the maximum TT rank χ that a standard TCI run would produce. Such information is often unavailable, and even when χ can be inferred (for instance, from a previous but costly TCI attempt), the optimal value of χ_{patch} remains strongly problem-specific. Fig. 3.7 illustrates this point with a toy example.

The target is a piecewise function consisting of an oscillatory segment followed by an exponential tail. The colour map reveals that some ways of partitioning the domain are clearly superior to others. In the bottom row of canvas, where the domain is subdivided too aggressively, the number of resulting patches becomes so large that the overall cost exceeds that of a single,

[6]The value of the bond dimension bound χ_{patch} fixed by the user doesn't always correspond to the maximum bond dimension at which TCI converges within each patch. The largest of these maximum bond dimensions among all the patches is here referred to as χ^*_{patch}.

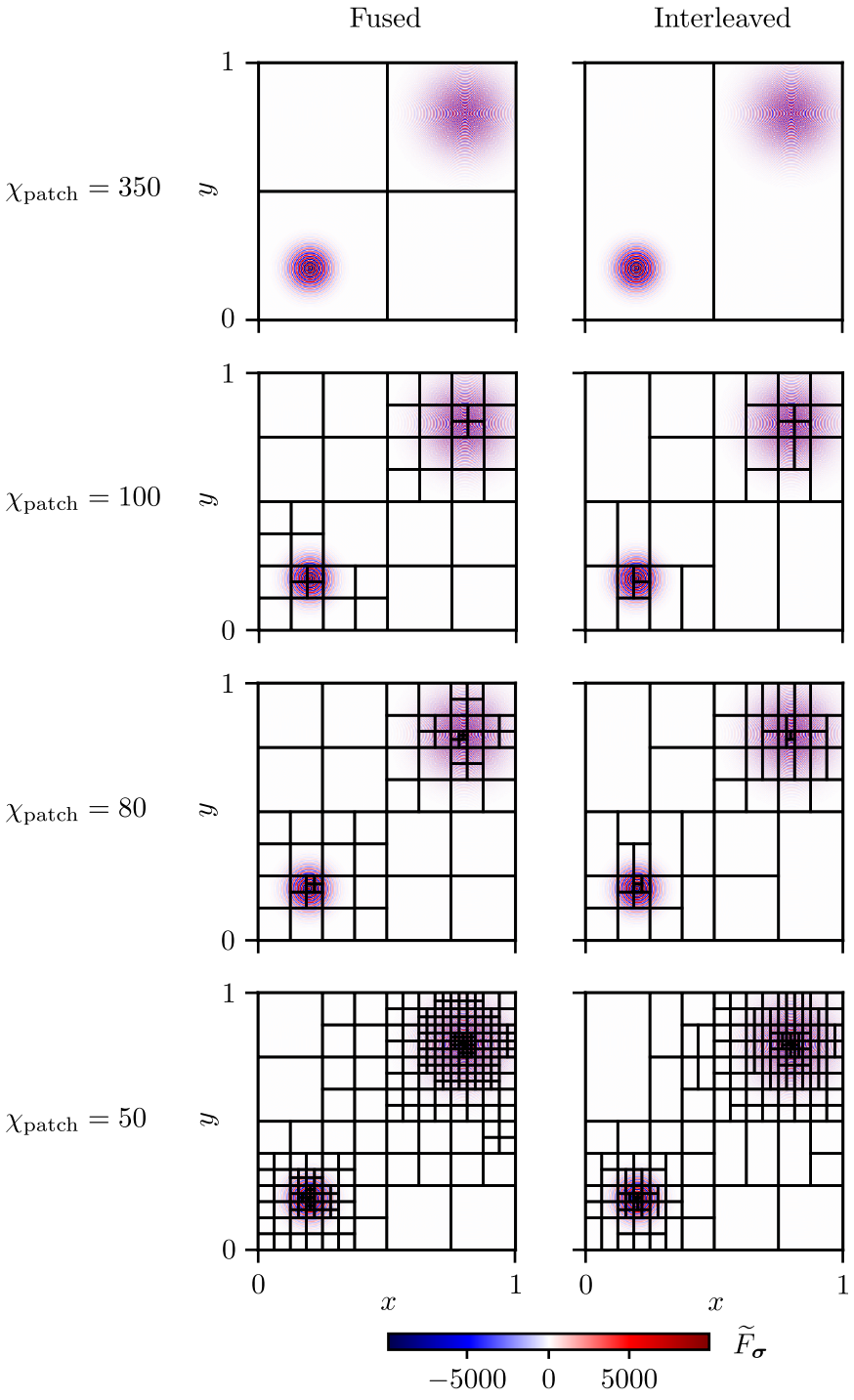


Figure 3.5: Evolution of the adaptive partitioning of the bivariate function $f(\boldsymbol{r})$ domain (cf. Eq. (2.26) and Eq. (3.30)), due to pQTCI. Both interleaved and fused ordering for the tensor $\mathcal{F}_{\boldsymbol{\sigma}} = f(\boldsymbol{r}(\boldsymbol{\sigma}))$ are illustrated. Smaller maximum bond dimensions per patch χ_{patch} render finer partitionings of the domain.

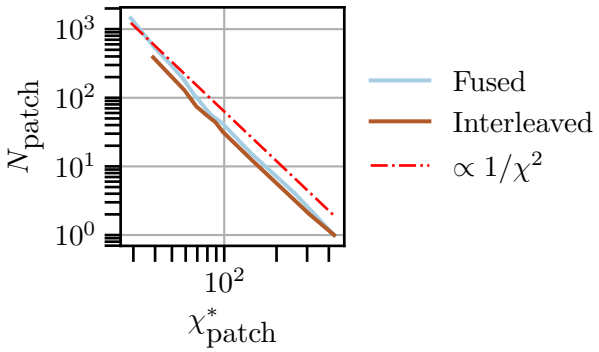


Figure 3.6: Number of total patches vs largest patch maximum bond dimension χ^*_{patch} for the approximation of $f(\boldsymbol{r})$ as in Eq. (2.26). The numerical data is compared with the estimated scaling $1/\chi^2$ (cf. Eq. (3.31)).

full-domain approximation. Over-patching is particularly problematic for functions whose features are not sharply localised: if we split either the exponential tail or the oscillatory region in half, it is impossible to identify a "simpler" versus "harder" sub–interval, and the extra patches provide no computational benefit.

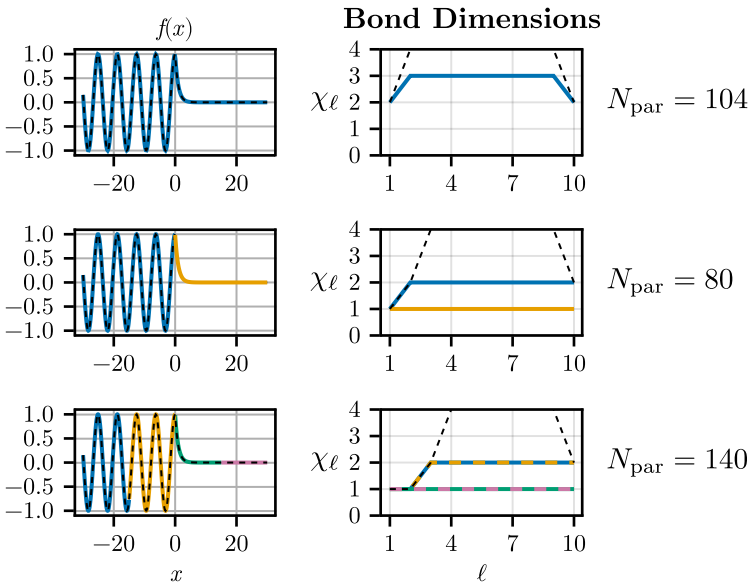


Figure 3.7: Domain partitioning of a one–dimensional piecewise function and the corresponding bond dimensions of the MPS that represents each segment[7]. Colours link each section of the function (left panels) to the bond dimensions of its TT unfolding (right panels). *From top to bottom*: a single, global approximation; an optimally subdivided approximation; and an inefficiently over-subdivided case.

This phenomenon—hereafter termed *over-patching*—is common when patched QTCI is applied to functions lacking sharply localised structure, or when the prescribed cap χ_{patch} is set too small for the task at hand. In either circumstance, once a patch reaches the bond-dimension limit, pQTCI tries to refine the domain further, irrespective of whether the function varies significantly within that patch. The outcome is a proliferation of sub-patches whose spatial extent is smaller, yet whose tensor-train representations inherit essentially the *same* bond dimensions as their parent patch, thereby defeating the purpose of the subdivision.

Fig. 3.8 shows how patched QTCI can fail when confronted with a function whose features are *uniformly spread across the whole domain*. The target here is

$$\begin{aligned} f(x,y) = 1 + e^{-0.4\,(x^2+y^2)} + e^{-x^2}\sin(xy) + e^{-y^2}\cos(3xy) \\ + \cos(x+y) + 0.05\,\cos\!\big[10^2\,(2x-4y)\big] \\ + 5\times10^{-4}\,\cos\!\big[10^3\,(-2x+7y)\big] + 10^{-5}\,\cos\!\big(2\times10^8 x\big), \end{aligned} \tag{3.32}$$

whose oscillatory and exponentially modulated components permeate the entire domain at different length scales. Because no subregion is markedly simpler than another, patched QTCI keeps slicing almost at random, generating numerous small patches whose local tensor-train ranks are *identical* to those of their parent regions. Consequently, the aggregate number of TT parameters (solid curve in panel (*b*)) surpasses that of a single, full-domain TCI approximation (dotted curve). In effect, the algorithm redundantly stores the same information in multiple MPS blocks, negating any potential savings and exemplifying the drawback of *over-patching*.

To curb over-patching, several practical safeguards could be introduced in future versions of the algorithm

- **Minimum patch size**: impose a lower bound $|\Omega|_{\min}$ on the spatial size of any patch $\Omega_{\mathbf{p}}$. If a candidate split would produce sub-domains smaller than this threshold, the recursion is halted and the current patch is accepted as is. This solution is particularly useful when the length scales of interest for a target function are known beforehand.

- **Post-processing merge**: after the recursive phase, inspect neighbouring patches whose TT cores share the same effective ranks. If the error of the combined residual of two siblings stays below τ_{merge}, replace them by a single, merged patch and recompress with TCI.

These mechanisms ensure that domain refinement is driven by actual approximation error rather than the arbitrary attainment of the bond-dimension limit, thereby preventing an explosion in the number of patches and preserving the intended resource savings of patched QTCI.

[7] I owe a debt of gratitude to my supervisor Marc for this figure.

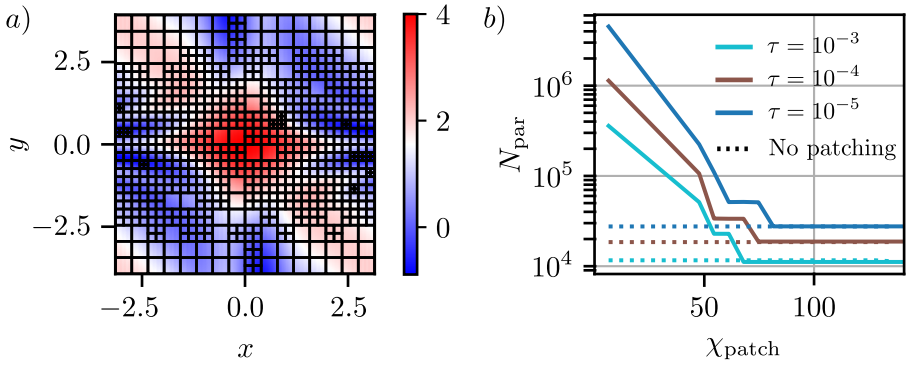


Figure 3.8: Performance of patched QTCI on the two–dimensional oscillatory function of Eq. (3.32). (*a*) Example of the domain subdivision produced with $\mathcal{R} = 15$, $\chi_{\mathrm{patch}} = 10$, and $\tau = 10^{-4}$. The partition is highly redundant and fails to align with the true feature layout of the function. (*b*) Total number of TT parameters returned by pQTCI (solid curves) at three target tolerances, plotted against the bond-dimension cap χ_{patch}. For comparison, the parameter count of a full-domain TCI compression is shown as a dotted line. Once $\chi_{\mathrm{patch}} \gtrsim 80$, every run collapses to a single patch and matches the TCI cost; for smaller caps, over-patching inflates the parameter count sharply.

Chapter 4

Patched MPO-MPO Contractions

MPO-MPO contractions are widely recognized as a critical computational bottleneck in tensor-network algorithms, appearing frequently in applications like real-time evolution [48], finite-temperature simulations [49], two-particle field theory calculations [50] and standard ground-state DMRG computations [5]. Even when employing optimal contraction strategies, the computational complexity typically scales unfavorably with increasing bond dimensions rapidly becoming the dominant cost for large-scale calculations [5, 48, 50].

(Q)TCI makes it possible to encode intricate functions in a highly compact tensor-train form, greatly facilitating the numerical evaluation of otherwise challenging integrals [1, 18, 50]. Besides the favourable complexity of the TCI algorithm itself, in general tensor representations are especially beneficial for convolution–type integrals, e.g. $h(x,y) = \int_0^1 \mathrm{d}t\, f(x,t)\, g(t,y)$, because they allow such integral to be carried out by simply contracting the MPOs representing the two integrands. More importantly, the contraction yields a ready-made, reusable tensor-train representation of the result h.

While (Q)TCI substantially mitigates the cost of setting up convolution–type integrals, the subsequent MPO–MPO contraction still scales steeply with the bond dimensions of the two factors–typically $\mathcal{O}(\chi^4)$ [51]– depending on the algorithm employed. This rank dependence motivates *distributing* the contraction across smaller sub-problems whose bond dimensions are capped, in the spirit of patched (Q)TCI. We refer to such strategies collectively as *patched MPO–MPO contractions*.

The remainder of this chapter is organised as follows: Sec. 4.1 reviews conventional MPO–MPO contraction schemes and establishes the notation used later; Sec. 4.2 introduces the patched approach, analysing its advantages and resource scaling for several representative tensor products; finally, Sec. 4.3 presents a new *adaptive* contraction algorithm that, analogously to pQTCI, dynamically partitions the tensors and caps the local bond dimension, thereby balancing the contraction workload across patches.

4.1 MPO-MPO Contractions: standard algorithms

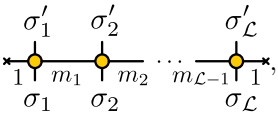
A variety of well–established algorithms can contract two MPOs with high efficiency. Before reviewing these methods, we introduce the notation and

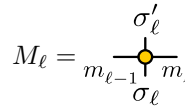
conventions commonly adopted in the tensor-network literature, beginning with the central object of interest: the *Matrix Product Operator* (*MPO*).

Matrix Product Operators

Consider the tensor-train representation

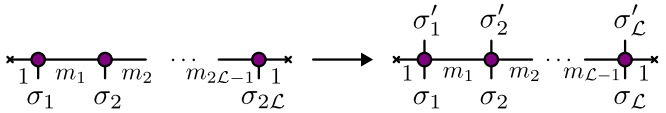
$$\widetilde{M}_{\boldsymbol{\sigma\sigma'}} = \quad \text{with} \quad M_\ell = \tag{4.1}$$



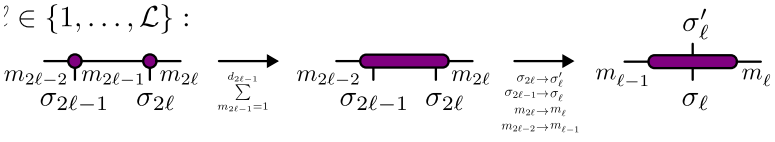


In the quantum-many-body community such a tensor train with two physical legs per site is known as a *Matrix Product Operator* (*MPO*) [5, 6, 52, 53]. In numerical mathematics the same object appears under the name *Tensor-Train Operator* (*TTO*) [11]. Each four-legged core M_ℓ carries two virtual indices and two physical indices, and the largest virtual dimension, $\chi = \max_\ell m_\ell$, is called the rank of the MPO, mirroring the definition for matrix-product states (MPSs). MPOs are routinely built to encode operators—Hamiltonians, density matrices, transfer matrices, projectors—that act on an MPS [15, 52]. They underpin modern DMRG algorithms, time-evolution schemes, finite-temperature purification, and many other tensor-network techniques.

The emergence of cross–approximation techniques has extended the relevance of MPOs to high–dimensional quadrature and convolution problems. A key property is that *any* matrix-product state (MPS) can be recast as an MPO simply by fusing pairs (or groups) of physical indices. For an MPS comprising $2\mathcal{L}$ sites, the transformation is schematically



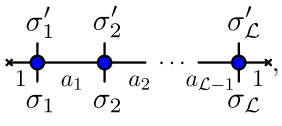
$\ell \in \{1, \ldots, \mathcal{L}\}$: (4.2)

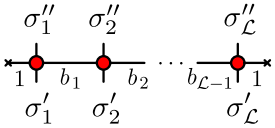


where consecutive physical indices $(\sigma_{2\ell-1}, \sigma_{2\ell})$ are merged into a single input–output pair $\big(\sigma_{2\ell-1}, \sigma_{2\ell}\big) \equiv (\sigma'_\ell, \sigma_\ell)$. This simple regrouping turns the state into an operator, enabling the same compressed representation to serve both as a multidimensional function and as an MPO contraction kernel. For a generic MPS, one simply fuses multiple neighbouring indices—for example $\sigma_\ell, \sigma_{\ell+1}, \sigma_{\ell+2} \to ((\sigma_\ell, \sigma_{\ell+1}), \sigma_{\ell+2}) := (\sigma_\ell, \sigma'_\ell)$—to obtain the desired operator form.

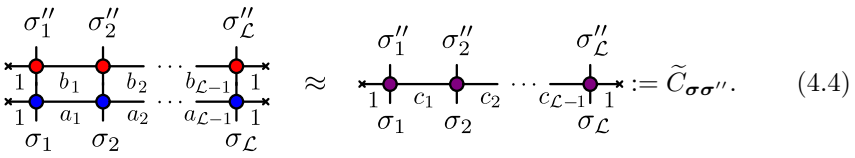
Given two matrix-product operators (MPOs)

$$\widetilde{A}_{\boldsymbol{\sigma\sigma'}} = \quad \widetilde{B}_{\boldsymbol{\sigma'\sigma''}} = \tag{4.3}$$





our goal is to form their product $\sum_{\boldsymbol{\sigma}'} \widetilde{A}_{\boldsymbol{\sigma}\boldsymbol{\sigma}'} \widetilde{B}_{\boldsymbol{\sigma}'\boldsymbol{\sigma}''}$:



(4.4)

If $\widetilde{A}$ and $\widetilde{B}$ both have rank χ we want the final result $\widetilde{C}$ also to have rank $\simeq \chi$. Several contraction strategies exist. Below we outline the familiar *zip-up* algorithm for completeness of this manuscript; alternative schemes available in modern tensor-network toolkits include the *fitting* routine of Stoudenmire and White [51] and the *density-matrix* algorithm described in the tensornetwork.org documentation [7].

The zip-up algorithm

Consider two MPOs of identical bond dimensions χ (cf. Eq. (4.3)) and phyiscal dimensions d.

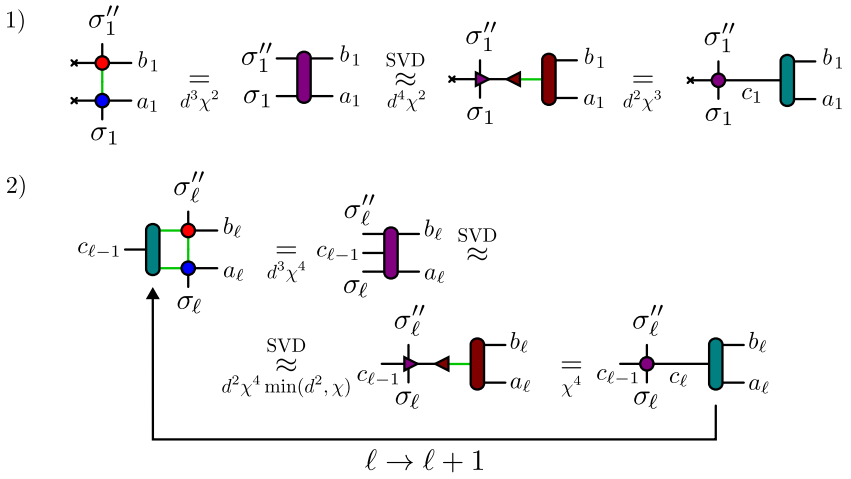


Figure 4.1: The zip-up MPO-MPO contraction algorithm. Each step is labelled with its computational cost. At each iteration the indices coloured green indicate the next contraction to be performed. Refer to the main manuscript for additional details.

Fig. 4.1 sketches the *zip-up* algorithm, an efficient left-to-right MPO contraction procedure:

(1) Begin at the leftmost site. Contract the shared physical index σ'_1 of the two site tensors to form a four-legged object. Treat (σ_1, σ''_1) as the row index and (a_1, b_1) as the column index of a matrix, then apply an SVD. The resulting left isometry ▶ becomes the first core of the product MPO, while the residual (the diagonal and right singular tensors) is contracted into a single "left-over" tensor. Truncate the SVD either by setting a

maximum rank or by discarding singular values below a local tolerance τ_{loc}.

(2) Move one site to the right. Contract the left-over tensor with two new site tensors out of the factor MPOs, fuse the indices $(\sigma_\ell, c_{\ell-1}, \sigma''_\ell)$ and (a_ℓ, b_ℓ) into the new row/column pair and repeat the SVD and truncation as in step (1).

After each SVD step the row index of the singular value matrix will constitute the new bond dimension c_ℓ for the product MPO. The zip-up procedures scales as

$$\mathcal{O}(\chi^4 d^4 \mathcal{L}) \tag{4.5}$$

with the parameters of our input MPOs.

A well–known drawback of the zip-up scheme is that the global approximation error of the resulting MPO is not rigorously bounded by the local SVD tolerances ε_{loc} [51]. In practice, however, tightening ε_{loc} usually reduces the overall error. Several best-practice remedies are commonly employed:

- *Right–canonical preconditioning.* Transform both the two input MPOs to right-canonical form [6]. This stabilises the local factorizations and mitigates ill-conditioning.
- *Error-based truncation.* Instead of imposing a hard rank cap χ, truncate each SVD by discarding singular values whose squared weight falls below a prescribed cutoff, thereby adapting the rank to the local spectrum.
- *Hybrid refinement.* Perform an initial left-to-right zip-up sweep with a fixed rank χ to obtain a rough product MPO, then feed this guess into the *fitting* algorithm of Ref. [51], which iteratively refines the bond dimensions while monitoring the global error.

These strategies balance accuracy and efficiency, providing tighter control over the final approximation error without incurring prohibitive cost. Adopting an error-based type of truncation, while monitoring the local bond dimensions of the resulting MPO, will allow us to use the zip-up routine for our patched version of MPO contraction algorithms.

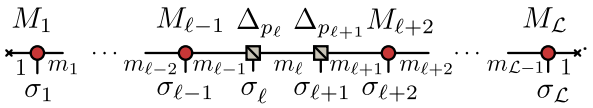
4.2 Patched MPO-MPO Contractions

MPO–MPO contractions rank among the most demanding kernels in tensor–network computations: their arithmetic cost scales roughly as the χ^4 power of the bond dimension χ of the two input operators. Such a steep dependence quickly turns the contraction step into a major bottleneck. Because the bond dimension is the dominant cost driver, a *distributed* strategy that caps the *local* bond dimension—mirroring the philosophy of patched QTCI—promises substantial savings.

In what follows we introduce this strategy, which we call *patched MPO–MPO contraction*, and show how it can already accelerate several representative tensor contractions encountered in practical applications. We start by defining *patched MPOs* and by illustrating how to contract them.

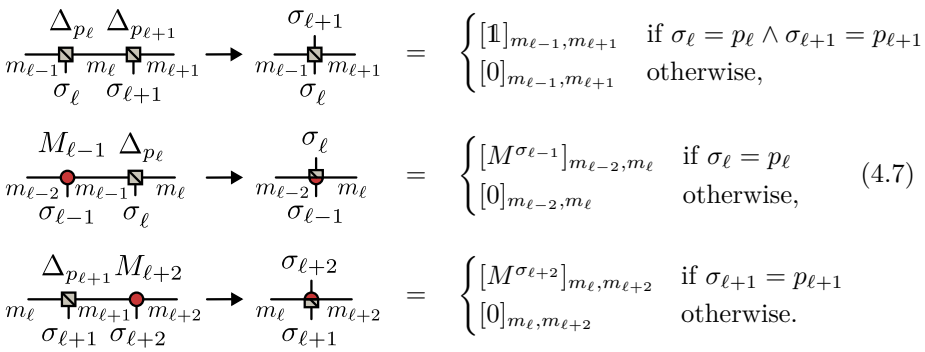
Patched contraction logic

The output produced by pQTCI contains objects with the schematic form



(4.6)

Suppose we promote this patched MPS to an MPO so that it can enter some contraction procedure. The local MPS-to-MPO mapping depends on the site index ℓ, hence on the way the MPS cores are grouped; from Eq. (4.2) one obtains

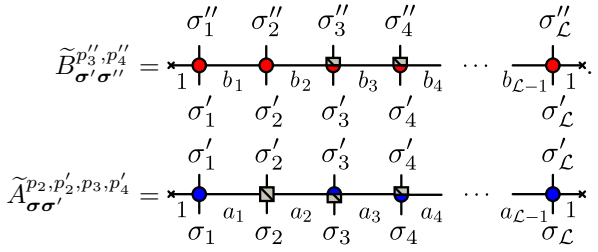


$$= \begin{cases} [\mathbb{1}]_{m_{\ell-1},m_{\ell+1}} & \text{if } \sigma_\ell = p_\ell \wedge \sigma_{\ell+1} = p_{\ell+1} \\ [0]_{m_{\ell-1},m_{\ell+1}} & \text{otherwise,} \end{cases}$$

$$= \begin{cases} [M^{\sigma_{\ell-1}}]_{m_{\ell-2},m_\ell} & \text{if } \sigma_\ell = p_\ell \\ [0]_{m_{\ell-2},m_\ell} & \text{otherwise,} \end{cases} \tag{4.7}$$

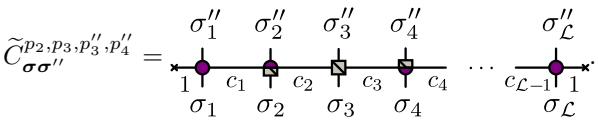
$$= \begin{cases} [M^{\sigma_{\ell+2}}]_{m_\ell,m_{\ell+2}} & \text{if } \sigma_{\ell+1} = p_{\ell+1} \\ [0]_{m_\ell,m_{\ell+2}} & \text{otherwise.} \end{cases}$$

Whenever we are contracting two patched MPOs, which may be folded from MPSs of the type in Eq. (4.6), the product is non–the vanishing only if for those sites on which both internal indices are projected, say $\sigma_\ell'^A = p_\ell'^A$ and $\sigma_\ell'^B = p_\ell'^B$, they are projected to the same value, i.e. if $p_\ell'^A = p_\ell'^B$. Consider the following patched MPOs



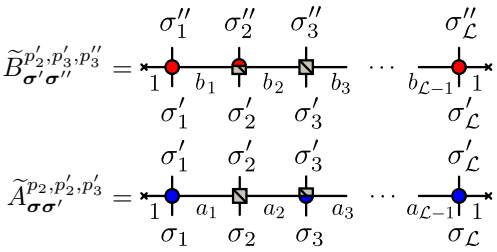
(4.8)

They always yield a result of the form



(4.9)

when contracting over the shared indices $\boldsymbol{\sigma}'$. By contrast, two MPOs shaped as



(4.10)

produce a non–zero contraction only when the projected internal indices match, *i.e.* $p_2'^A = p_2'^B$ and $p_3'^A = p_3'^B$. In other words, to obtain a non–vanishing outcome, any internal index that is projected must be projected onto the *same* value in both factors; the external indices ($\boldsymbol{\sigma}, \boldsymbol{\sigma}''$ in Eq. (4.8)) place no such restriction. All the tensors in Eq. (4.8), Eq. (4.9) and Eq. (4.10) will be hereafter referred to as *patched MPOs* or *patch MPOs*.

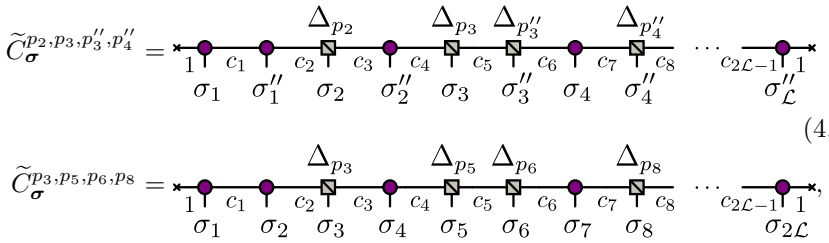
Assume that two tensors $A_{\boldsymbol{\sigma}}$ and $B_{\boldsymbol{\sigma}}$ have been decomposed through p(Q)TCI into collections of patches,

$$A_{\boldsymbol{\sigma}} \approx \sum_{\widetilde{A}^{(i)} \in \texttt{results}_A} \widetilde{A}^{(i)}, \qquad B_{\boldsymbol{\sigma}} \approx \sum_{\widetilde{B}^{(j)} \in \texttt{results}_B} \widetilde{B}^{(j)}. \tag{4.11}$$

After promoting each patch to MPO form (cf. Eq. (4.7)), the full contraction $C_{\boldsymbol{\sigma},\boldsymbol{\sigma}''} = \sum_{\boldsymbol{\sigma}'} A_{\boldsymbol{\sigma},\boldsymbol{\sigma}'} B_{\boldsymbol{\sigma}'\boldsymbol{\sigma}''}$ can be assembled patch-wise: contract every patch $\widetilde{A}^{(i)}$ with each *compatible* patch $\widetilde{B}^{(j)}$ (i.e. those whose projected internal indices match). This yields a family of subtensors $\{\widetilde{C}^{(k)}\}$ which, upon MPO-to-MPS unfolding (cf. Eq. (4.12)), collectively approximate the target tensor $C_{\boldsymbol{\sigma}}$ (unfolding of $C_{\boldsymbol{\sigma},\boldsymbol{\sigma}''}$). In other words, each TT of $\texttt{results}_A$ will be contracted with every TT in $\texttt{results}_B$ and depending on the compatibility of the projection result in a TT $\widetilde{C}^{(k)}$ part of the patched representation of $C_{\boldsymbol{\sigma}}$. The subsequent section will help us clarify this concept with a graphical representation of the *patched contraction* for quantics tensor trains.

2D representation of patched contractions

The product MPO in Eq. (4.9) can be unfolded back into a patched MPS by inverting the mapping of Eq. (4.2). To do so, one may perform an SVD or a CI/prrLU on each MPO site tensor or simply read off the local cores from Eq. (4.7), in the case of a patched MPO. The resulting MPS reads



(4.12)

where the second equality is obtained after relabelling site indices and bond dimensions. The auxiliary cores Δp_ℓ encode the subtensor corresponding to the

external (non–contracted) indices of $\widetilde{A}$ and $\widetilde{B}$, *i.e.* external (non-contracted) projected indices in $\widetilde{A}$ and $\widetilde{B}$ will be projected indices in $\widetilde{C}$.

The tensor-trains in Eq. (4.12) closely mirror the TT form of an individual patch produced by the pQTCI algorithm. If each TT physical index has dimension $d_\ell = 2$ we can assume to be working with a two–dimensional, interleaved quantics representation of some function, where each such MPS corresponds to a distinct sub-region of a 2D domain, uniquely labelled by its prefix indices. A full set of these patches can tile the entire domain of the function.

This insight suggests a convenient two-dimensional diagrammatic view of patched contractions. Let us introduce three quantics variables on the unit interval, each resolved with $\mathcal{R}$ binary digits,

$$\begin{aligned} x \mapsto x(\boldsymbol{\sigma}) &= \sum_{r=1}^{\mathcal{R}} \sigma_r 2^{-r} \qquad & y \mapsto y(\boldsymbol{\sigma}'') &= \sum_{r=1}^{\mathcal{R}} \sigma''_r 2^{-r} \\ s \mapsto s(\boldsymbol{\sigma}') &= \sum_{r=1}^{\mathcal{R}} \sigma'_r 2^{-r} & \sigma_r, \sigma'_r, \sigma''_r &\in \{0,1\} \end{aligned} \tag{4.13}$$

With this notation, the patch ensembles introduced in Eq. (4.11)—and the tensor that results from their pairwise contractions—can be depicted with a two-dimensional schematic (Fig. 4.2).

Fig. 4.2 illustrates the idea for two well-constructed collections of patched MPOs, A and B. Each MPO patch is first unfolded into an MPS and mapped to its assigned region of the two–dimensional domain, exactly as described in the preceding chapter for pQTCI. Panels (a) and (b) show the resulting patch ensembles for A and B, respectively. Every A-patch is then paired with every *compatible* B-patch and contracted over their shared indices $\boldsymbol{\sigma}'$ (s variable), covering all index combinations permitted by the internal projections[1]. The collection of contraction results tiles the square $[0,1]^2$, furnishing a patched representation of the product tensor C.

As highlighted by the first row of Fig. 4.2, this patchwise contraction behaves as a "matrix multiplication" of patches:

- the patching depth along the *columns* (x-direction) of the final object is inherited from the row patching of the left factor A;
- the patching depth along the *rows* (y-direction) is inherited from the column patching of the right factor B.

Although the schematic in Fig. 4.2 employs a simple, regular subdivision to convey the basic logic of *patched contractions*, real applications involve far more intricate patch patterns and additional constraints on the internal indices, making the general contraction problem correspondingly richer and more challenging.

The diagrammatic picture introduced in Fig. 4.2 is not restricted to MPOs that come from a two–dimensional, interleaved quantics TT unfolding. It extends naturally to higher physical dimensions. Suppose that we have a set of

[1] In the particular example of Fig. 4.2 no constraints due to internal indices' projections are imposed. Hence, every A-patch is contracted with every B-patch.

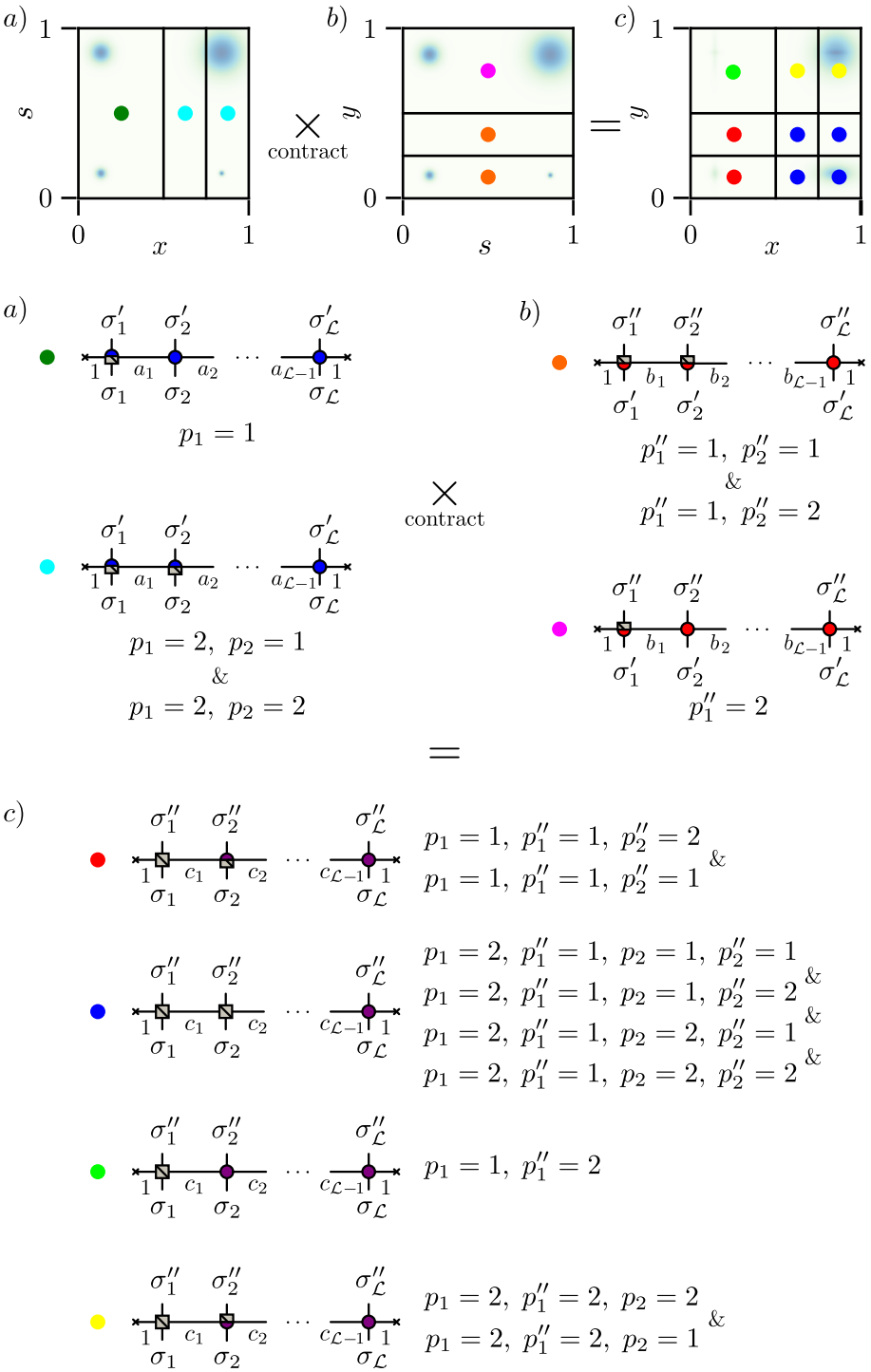


Figure 4.2: 2D representation of patched MPO-MPO contraction.

$\mathcal{R}$ indices each of dimension d, then we can assign to it real variable x discretised in base d such as

$$x \;\mapsto\; x(\boldsymbol{\sigma}) \;=\; \sum_{r=1}^{\mathcal{R}} \sigma_r \, d^{-r}, \qquad \sigma_r \in \{1, \ldots, d\}. \tag{4.14}$$

The corresponding axis will then be partitioned into d intervals at each digit (rather than two), and the same two–dimensional patch contraction picture can be drawn. In fact, the internal indices $\boldsymbol{\sigma'}$ and the external indices $\boldsymbol{\sigma}$, $\boldsymbol{\sigma''}$ may even carry different local dimensions d, d' and d''. Provided that the internal local dimensions are equal for the two factors, i.e. $\dim(\sigma'^{A}_{\ell}) = \dim(\sigma'^{B}_{\ell})\ \forall \ell$, the two–dimensional representation retains its generality.

This grid view is especially convenient when modelling convolution–type integrals—*i.e.*, "matrix multiplications" of two-dimensional functions—such as

$$h(x, y) \;=\; \int_0^1 ds\; f(x, s)\, g(s, y). \tag{4.15}$$

Here each patch pair encodes the actual contribution of a specific $(x{\times}s){\otimes}(s{\times}y)$ block to the result $h(x, y)$, and the patched contraction tiles the $[0, 1]^2$ domain exactly.

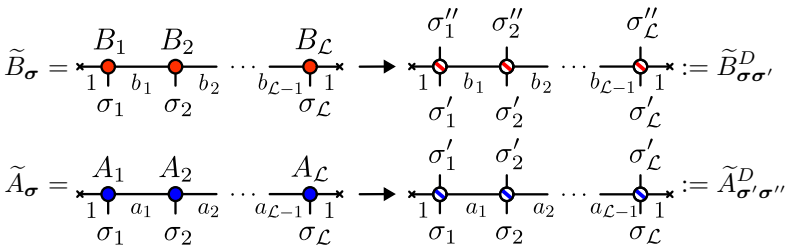
The foregoing analysis suggests that coupling pQTCI with a patch-wise contraction scheme can dramatically curb the computational expense of many tensor-network tasks—particularly when the underlying functions are highly localised. Before presenting concrete benchmarks for this combined approach, we first examine the expected cost reductions and performance benefits of patch-level contractions across several representative classes of MPO–MPO products.

Patched element-wise multiplication

Let $\widetilde{A}_{\boldsymbol{\sigma}}$ and $\widetilde{B}_{\boldsymbol{\sigma}}$ be two MPSs. We seek their *Hadamard (element-wise) product* [44]

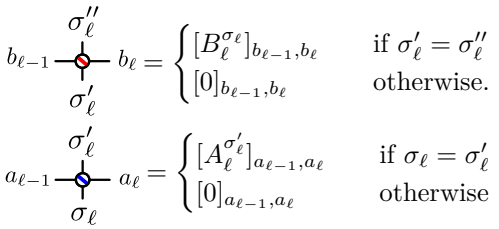
$$C_{\boldsymbol{\sigma}} = A_{\boldsymbol{\sigma}} \circledast B_{\boldsymbol{\sigma}} \tag{4.16}$$

meaning that every entry of C is obtained by multiplying the corresponding entries of A and B; $C_{\boldsymbol{\sigma}} = A_{\boldsymbol{\sigma}} B_{\boldsymbol{\sigma}}$ for all index tuples $\boldsymbol{\sigma}$. To carry out the element-wise product with TTs we first need to embed each tensor train unfolding of the factors in a *diagonal* MPO:



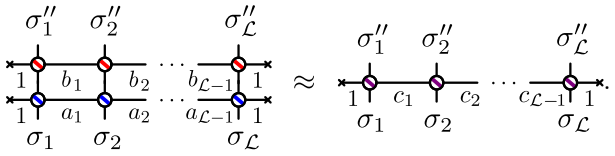
(4.17)

where each local tensor is defined by



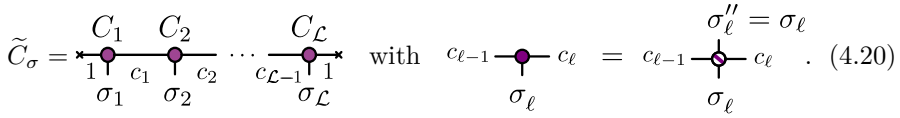
(4.18)

The site indices of the original TTs have been relabelled to make the subsequent contraction explicit:



(4.19)

From this contraction we read off the resulting tensor-train



(4.20)

Hence $\widetilde{C}$ represents the Hadamard product $C_{\boldsymbol{\sigma}} = A_{\boldsymbol{\sigma}} B_{\boldsymbol{\sigma}}$ in TT form. The MPO-MPO contraction in Eq. (4.19) can be performed using the standard MPS-toolkit [12, 13], however if we apply the patching scheme upon the factor tensors A and B, the patched contraction routine can be employed.

Eq. (4.18) shows that a *diagonalised* MPO can sensibly be patched only along *identical* input and output indices, because each site tensor is non–zero solely when $\sigma''_\ell = \sigma'_\ell$ (or, equivalently, when $\sigma'_\ell = \sigma_\ell$). Hence, it is unnecessary to unfold the diagonal MPOs of Eq. (4.17) and Eq. (4.19) into yet another MPS representation. Instead, we may work directly with the original MPS forms $\widetilde{A}_{\boldsymbol{\sigma}}$ and $\widetilde{B}_{\boldsymbol{\sigma}}$ and apply the patching routine to them. Only those patches whose domains *overlap* will yield a non–vanishing contraction, making the two–dimensional patch diagram introduced earlier particularly transparent in this setting. We now illustrate the idea with a concrete example.

Consider two tensor trains $\widetilde{A}_{\boldsymbol{\sigma}}$ and $\widetilde{B}_{\boldsymbol{\sigma}}$ that represent two-dimensional functions on the unit square, both obtained via the pQTCI routine. As before we map the interleaved index string $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots \sigma_{2\mathcal{R}})$ to the auxiliary variables

$$x \mapsto \sum_{r=1}^{\mathcal{R}} \sigma_{2r-1} 2^{-r}, \quad y \mapsto \sum_{r=1}^{\mathcal{R}} \sigma_{2r} 2^{-r}, \quad \sigma_{2r}, \sigma_{2r-1} \in \{0, 1\} \tag{4.21}$$

so that each patch corresponds to a tile in the $[0, 1]^2$ domain. Because we perform an *element-wise* (Hadamard) product, the two operands share the

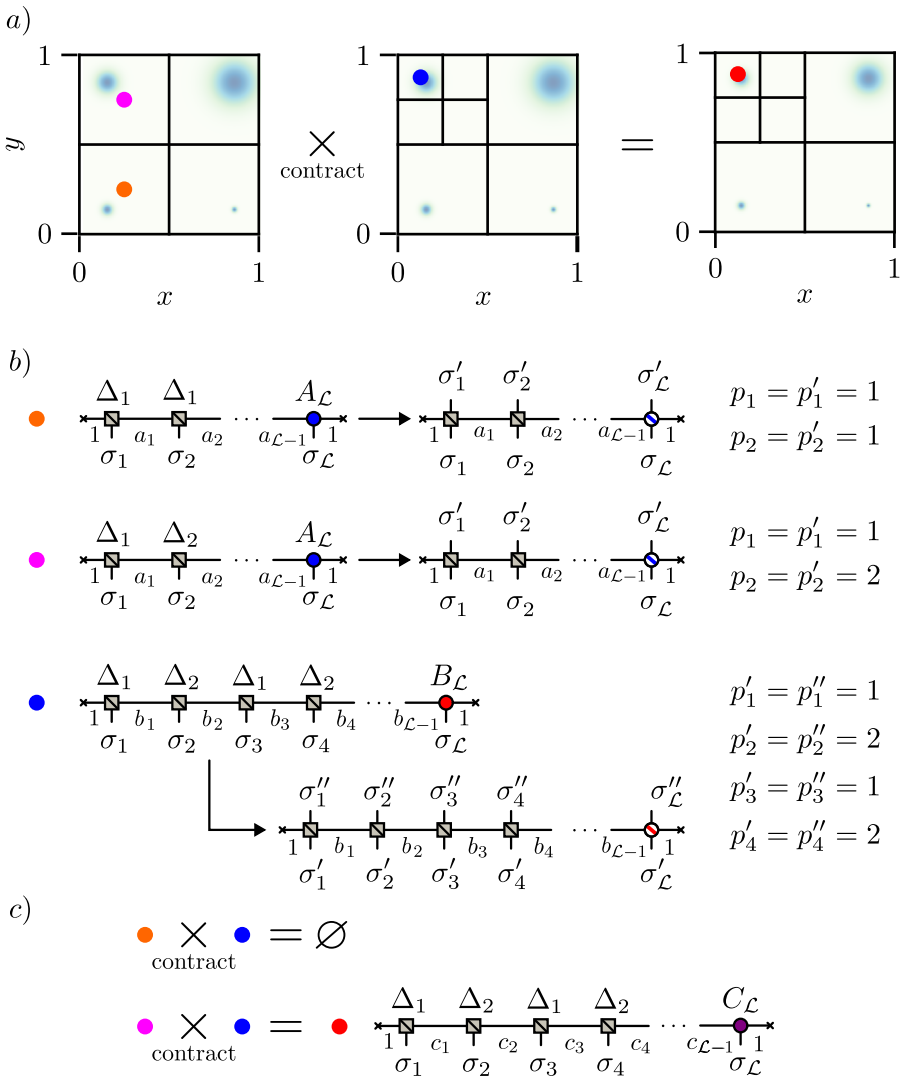


Figure 4.3: Patched element-wise contraction. (*a*) Two–dimensional domain tiled by patches visualised via the auxiliary variables $x(\sigma_1, \sigma_3, \dots)$ and $y(\sigma_2, \sigma_4, \dots)$. (*b*) Representative patches in tensor form: each patch is shown first in the TT format as produced by pQTCI and then in its diagonalised MPO version obtained with Eq. (4.18). (*c*) Patchwise multiplication for the patches marked by coloured dots; only those whose tiles overlap in the domain yield non-zero results.

same (x, y) grid; only patches whose tiles overlap contribute to the result, as sketched in Fig. 4.3.

Assume that both A and B are decomposed into the same number of patches, N_{patch}, and that every patch has a capped bond dimension χ_{patch}. The patched routine executes approximately N_{patch} MPO–MPO contractions. Employing the zip-up algorithm (or an equivalent $\mathcal{O}(\chi^4 d^4 \mathcal{L})$ scheme) for each

local contraction, the total arithmetic cost of element-wise MPS multiplication scales as

$$\mathcal{O}(N_{\text{patch}}\chi_{\text{patch}}^4 d^4 \mathcal{L}). \tag{4.22}$$

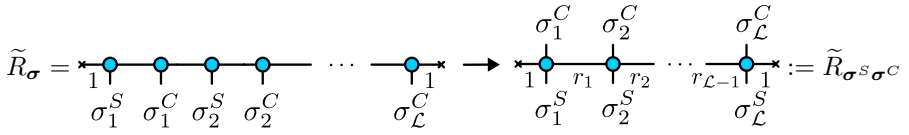
To outperform a single, non-patched contraction of the full-rank MPOs (χ being their common bond dimension), the number of patches must obey

$$N_{\text{patch}} < \frac{\chi^4}{\chi_{\text{patch}}^4}. \tag{4.23}$$

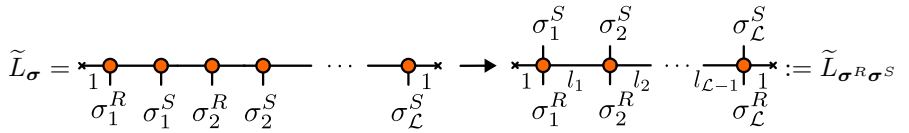
Within this window patch-wise element-wise multiplication delivers a net computational saving by trading rank for patch count.

Patched matrix multiplication

Let $\widetilde{L}_{\boldsymbol{\sigma}}$ and $\widetilde{R}_{\boldsymbol{\sigma}}$ be two generic matrix-product states (MPS). The first carries index sets $\boldsymbol{\sigma}^R, \boldsymbol{\sigma}^S$, the second $\boldsymbol{\sigma}^S, \boldsymbol{\sigma}^C$. Using the mapping of Eq. (4.2), each MPS can be reshaped into an MPO:



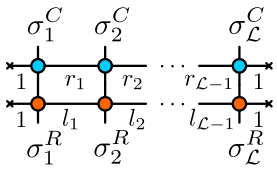
(4.24)



Here each composite index (e.g. σ_ℓ^R) can represent a single index or a block of neighbouring physical indices of a parent MPS and every correspondent MPS core is obtained by contracting the corresponding block of neighbouring MPS site tensors.

With this interpretation, the standard MPO-MPO contraction between $\widetilde{L}$ and $\widetilde{R}$

$$\widetilde{M}_{\boldsymbol{\sigma}^R\boldsymbol{\sigma}^C} = \sum_{\boldsymbol{\sigma}^S} \widetilde{L}_{\boldsymbol{\sigma}^R\boldsymbol{\sigma}^S}\widetilde{R}_{\boldsymbol{\sigma}^S\boldsymbol{\sigma}^C} = \tag{4.25}$$



acts exactly like a *matrix multiplication* between two tensor-train–style objects, producing an MPO indexed by $\boldsymbol{\sigma}^R$ and $\boldsymbol{\sigma}^C$. The labeling becomes then much clearer: $\boldsymbol{\sigma}^R$, $\boldsymbol{\sigma}^C$ and $\boldsymbol{\sigma}^S$ represent the row, column and shared indices of our "matrices" L and R.

Assume that $L_{\boldsymbol{\sigma}^R\boldsymbol{\sigma}^S}$ and $R_{\boldsymbol{\sigma}^S,\boldsymbol{\sigma}^C}$ are pQTCI approximations of two multivariate functions,

$$L(\boldsymbol{x}(\boldsymbol{\sigma}^R), \boldsymbol{s}(\boldsymbol{\sigma}^S)) = L_{\boldsymbol{\sigma}^R,\boldsymbol{\sigma}^S}, \quad R(\boldsymbol{s}(\boldsymbol{\sigma}^S), \boldsymbol{y}(\boldsymbol{\sigma}^C)) = R_{\boldsymbol{\sigma}^S,\boldsymbol{\sigma}^R} \tag{4.26}$$

respectively, their contraction realises the convolution

$$M(\boldsymbol{x}, \boldsymbol{y}) = \int d\boldsymbol{s} L(\boldsymbol{x}, \boldsymbol{s}) R(\boldsymbol{s}, \boldsymbol{y}). \tag{4.27}$$

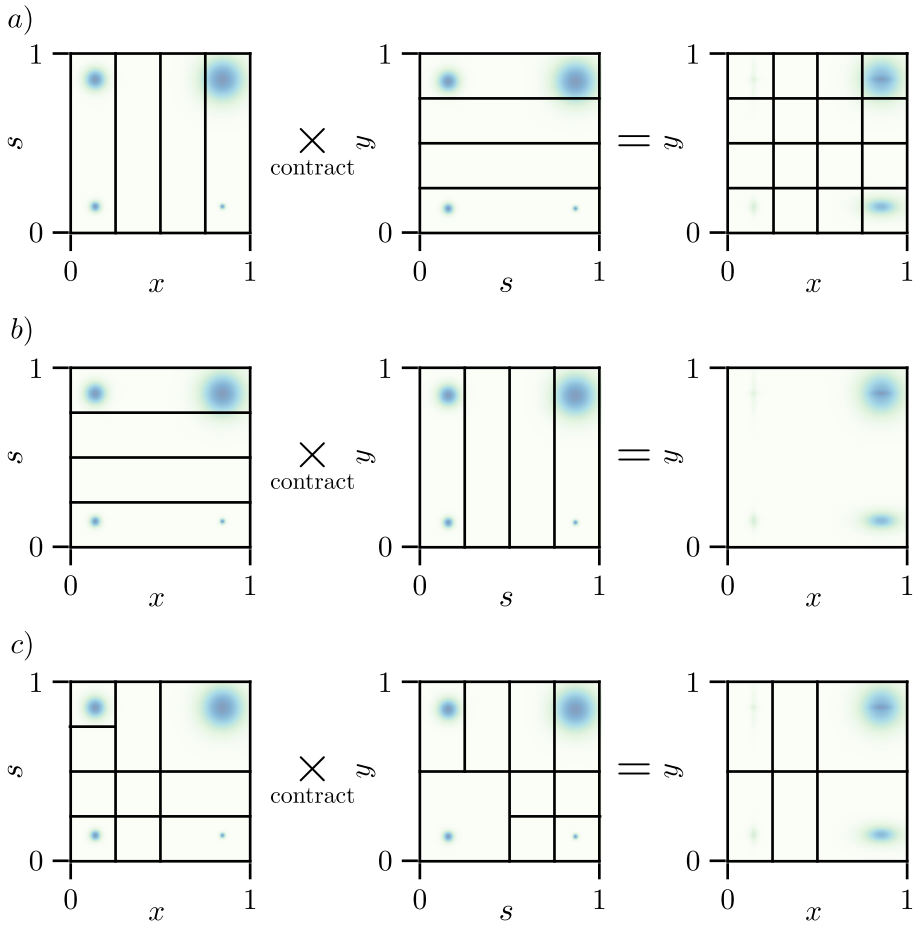


Figure 4.4: Patched *matrix multiplication*. (*a*) Worst-case scenario. Each patch in the first factor is contracted with *each* patch in the second factor. (*b*) Best-case scenario. Each patch in the first factor is contacted with a *single* patch in the second factor. (*c*) Average case scenario. Each patch in the first factor is contracted with a *limited set* of patches in the second factor.

Fig. 4.4 visualises three patching patterns for this "MPO–matrix multiplication," assuming each factor is decomposed into N_{patch} patches of equal bond

dimension χ_{patch}. There the first column corresponds to the L patched tensor, the second column to the R and the last column of panels to the patched contraction result M. The possible scenarios are the following:

(a) **Worst case** (first row): only the external indices are patched, so every L-patch contracts with *every* R-patch. The N_{patch}^2 contractions cost

$$\mathcal{O}\big(N_{\text{patch}}^2\, \chi_{\text{patch}}^4\, d^4\, \mathcal{L}\big), \tag{4.28}$$

improving on a full-rank contraction iff

$$N_{\text{patch}} < \frac{\chi^2}{\chi_{\text{patch}}^2}. \tag{4.29}$$

(b) **Best case** (centre row): the shared index $\boldsymbol{\sigma}^S$ is patched so that each L-patch matches exactly one R-patch. The N_{patch} contraction cost scale as

$$\mathcal{O}\big(N_{\text{patch}}\, \chi_{\text{patch}}^4\, d^4\, \mathcal{L}\big), \tag{4.30}$$

yielding a gain whenever

$$N_{\text{patch}} < \frac{\chi^4}{\chi_{\text{patch}}^4}. \tag{4.31}$$

(c) **Average case** (bottom row): both external and shared indices are patched. After $\bar{\ell}$ subdivision steps on a uniform d-ary grid the number of admissible patch pairs is $d^{\bar{\ell}}\, d^{\bar{\ell}(D-1)/D} = N_{\text{patch}}^{(2D-1)/D}$, where $d^{\bar{\ell}} = N_{\text{patch}}$ for a uniform grid and $D = \mathcal{N}$ for the interleaved ordering (*i.e.* the dimensionality of the functions $L(x_1, \ldots, x_{\mathcal{N}/2}, s_1, \ldots, s_{\mathcal{N}/2})$ and $R(s_1, \ldots, s_{\mathcal{N}/2}, y_1, \ldots, y_{\mathcal{N}/2})$) and $D = 2$ for fused. The overall cost becomes

$$\mathcal{O}\big(N_{\text{patch}}^{\frac{2D-1}{D}}\, \chi_{\text{patch}}^4\, \mathcal{L}\big), \tag{4.32}$$

advantageous as long as

$$N_{\text{patch}} < \left(\frac{\chi}{\chi_{\text{patch}}}\right)^{\frac{4D}{2D-1}}. \tag{4.33}$$

Patch-wise matrix multiplication trades global bond dimension for patch count. When any of the bounds in Eq. (4.31), Eq. (4.29) and Eq. (4.33) are respected, patched MPO–MPO contractions has the possibility[2] to outperform their monolithic counterparts, making them a compelling tool for high-dimensional convolutions.

[2]The typical result out of a pQTCI run is a combination of the patterns shown in Fig. 4.4, due to the adaptivity of the algorithm. Hence, Eq. (4.31), Eq. (4.29) and Eq. (4.33) only give us an intuition of the optimal bounds for N_{patch}.

4.3 Adaptive Patched MPO-MPO Contraction

Building on the principles of pQTCI, we now propose an *adaptive* contraction scheme that further curbs the memory and runtime requirements of patched MPO–MPO products. The primary bottleneck in any contraction routine is the peak RAM footprint, which grows steeply with the bond dimension of the tensors being multiplied. By dynamically refining the factor MPOs whenever their local bond dimension exceeds a prescribed cap, we can keep the intermediate contraction costs in check. The resulting procedure, which we term the *adaptive patched MPO–MPO contraction* (or *adaptive Matrix Multiplication*), combines patched contractions with a bond–dimension–driven refinement loop, similar to pQTCI.

MPO slicing

The key technical ingredient is an MPO analogue of the MPS *slicing* operation (cf. Eq. (3.12)). Because every MPO can be unfolded into an MPS (see Eq. (4.9) and Eq. (4.12)), we may

(i) unfold the operator to an MPS,

(ii) project the state onto a subtensor specified by a prefix, *e.g.* $(p_1, p_2, \ldots, p_{\bar{\ell}})$, and

(iii) fold the sliced MPS back into an MPO via Eq. (4.7).

Fig. 4.5 illustrates this "MPO slicing" workflow.

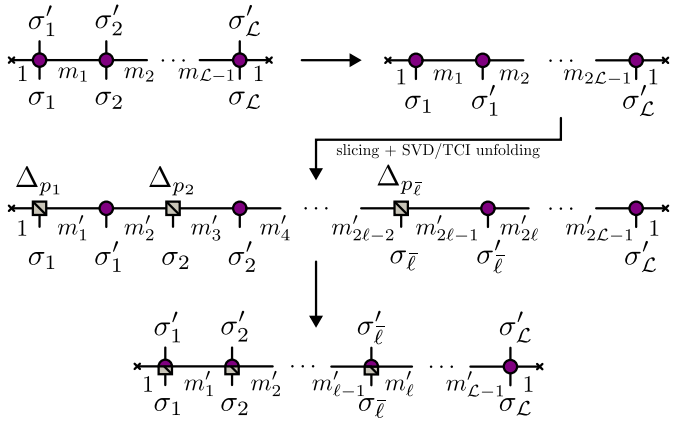


Figure 4.5: Slicing of an MPO

The slicing procedure illustrated in Fig. 4.5 lets our algorithm reuse the TT-projection routines of pQTCI for MPOs. In addition, retaining each patch as an MPS instead simplifies site-tensor manipulation and index bookkeeping; each patched MPS is converted into an MPO only immediately before a contraction step.

The algorithm

Given two MPOs $\widetilde{A}_{\boldsymbol{\sigma}\boldsymbol{\sigma}'}$ and $\widetilde{B}_{\boldsymbol{\sigma}'\boldsymbol{\sigma}''}$ (cf. Eq. (4.3)), the adaptive patched contraction attempts to compute the "matrix multiplication":

$$\widetilde{C}_{\boldsymbol{\sigma}'\boldsymbol{\sigma}''} = \sum_{\boldsymbol{\sigma}'} \widetilde{A}_{\boldsymbol{\sigma}\boldsymbol{\sigma}'} \widetilde{B}_{\boldsymbol{\sigma}'\boldsymbol{\sigma}''} \tag{4.34}$$

as follows

(1) Fix a bond–dimension cap χ_{patch} and a target accuracy τ. Attempt a *single* MPO–MPO contraction. If the product $\widetilde{C}_{\boldsymbol{\sigma}\boldsymbol{\sigma}''}$ converges to tolerance with rank $\chi < \chi_{\text{patch}}$, terminate.

(2) Otherwise, slice both MPOs along their first external physical indices, producing

$$\widetilde{A}^{p_1}_{\boldsymbol{\sigma},\boldsymbol{\sigma}'}, \quad \widetilde{B}^{p_1''}_{\boldsymbol{\sigma}',\boldsymbol{\sigma}''}, \qquad p_1'' \in \{1,\dots,d_1''\},\ p_1 \in \{1,\dots,d_1\}, \tag{4.35}$$

via the MPO–slicing routine of Fig. 4.5.

(3) Contract every *compatible* pair $\left(\widetilde{A}^{\,p_1}, \widetilde{B}^{\,p_1''}\right)$, enforcing the bond cap χ_{patch} on each local product.

(4) For each partial product $\widetilde{C}^{\,p_1,p_1''}$ that still fails to converge (`tasks`), slice it further along the $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma}''$ axes while storing any converged patch (`results`).

(5) Repeat steps (3)–(4), recursively increasing the slicing depth, until no unconverged patched MPOs remain.

A few remarks are in order. At first sight the adaptive routine might seem more expensive than a single MPO–MPO contraction. In practice this overhead is negligible: for a fixed tolerance τ each local contraction is *aborted* as soon as its bond dimension hits the cap χ_{patch} (e.g. by truncating the SVD step at the desired cutoff in the zip–up sweep, Fig. 4.1). Although up to $\mathcal{O}\big(d^{\bar{\ell}}(d'')^{\bar{\ell}}\big)$ patched MPO pairs may be processed—$\bar{\ell}$ being the deepest slicing level of the MPS unfolded factors $\widetilde{A}$ and $\widetilde{B}$—each individual multiply is cheap thanks to the tight rank bound χ_{patch}.

The second remark concerns the choice of patching indices. Our implementation slices only the *external* index sets $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma}''$; consequently the total number of multiplications is $N = N^2_{\text{patch}}$, matching the worst–case scaling in Eq. (4.28). This choice is, however, deliberate:

(a) the arithmetic cost remains $\mathcal{O}\big(N\,\chi^4_{\text{patch}}\,d^4\,\mathcal{L}\big)$, so memory is still controlled by χ_{patch}, which is fixed and therefore independent of the particular slicing choice;

(b) the refinement is in this way *feature–adaptive*: whenever a patch $\widetilde{C}^{p_1,p_1'',\dots}$ fails to converge (its local rank exceeds χ_{patch}) the algorithm subdivides exactly that output configuration space $(\boldsymbol{\sigma},\boldsymbol{\sigma}'')$ region, yielding a finer

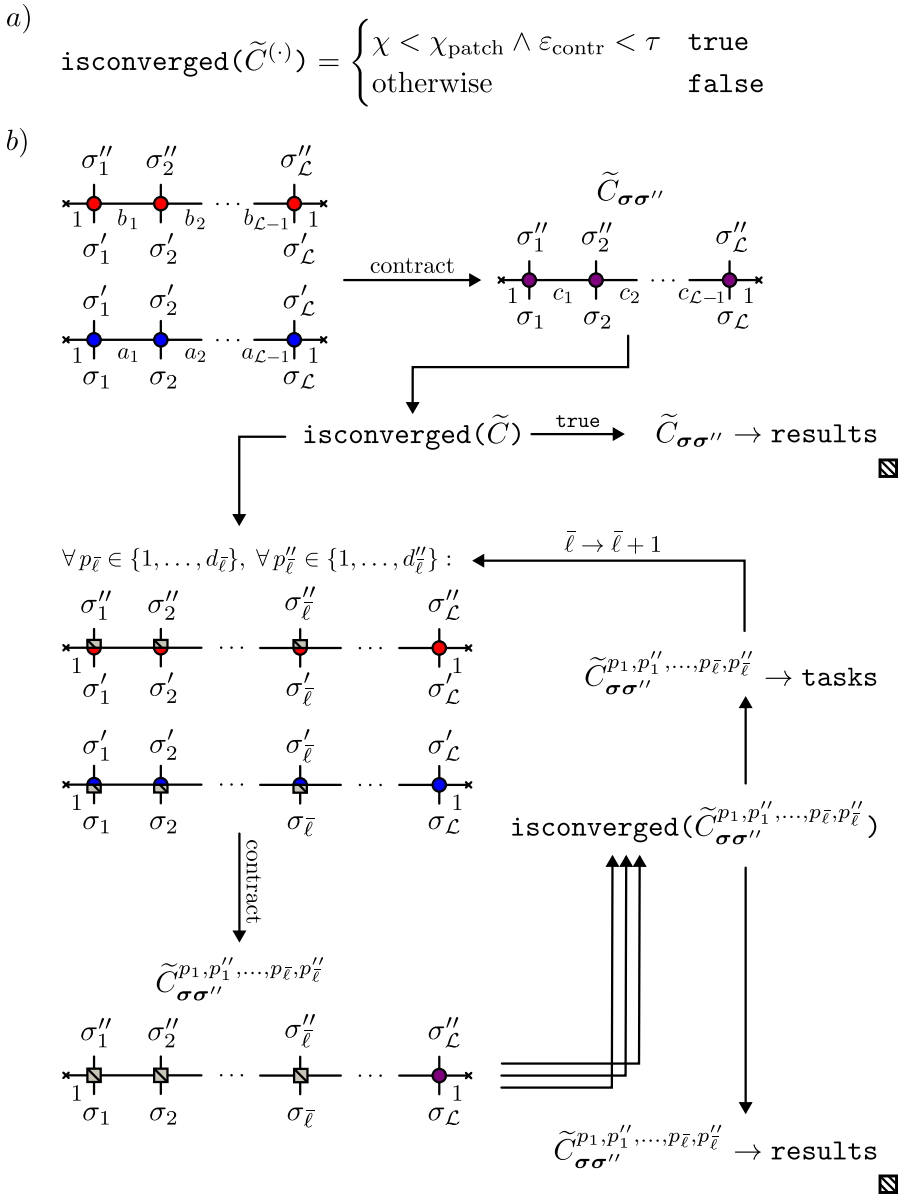


Figure 4.6: Adaptive patched MPO–MPO contraction. (*a*) Convergence test for a patch $\widetilde{C}^{\,p_1,p''_1,\dots,p_{\bar{\ell}},p''_{\bar{\ell}}}$, governed by the bond–dimension cap χ_{patch} and the accuracy tolerance τ. $\varepsilon_{\text{contr}}$ represents the error of the contraction result. (*b*) Flowchart of the algorithm. The global MPO product is recursively broken into smaller sub-problems via *MPO slicing* (Fig. 4.5). Each subtensor pair is contracted under the bond cap χ_{patch}; converged patches are moved to `results`, while the remaining ones are placed back into `tasks`. The routine halts – ▧ – once `tasks` is empty. See the text for a step-by-step description.

discretisation where the product tensor $\widetilde{C}_{\boldsymbol{\sigma}\boldsymbol{\sigma}''}$ is most intricate. Slicing

the internal indices $\boldsymbol{\sigma}'$ instead of the external ones would also reduce the contraction cost by capping the bond dimension at χ_{patch}, but at the expense of this adaptive focus.

The *adaptive patched MPO–MPO contraction* shown in Fig. 4.6 bounds the local bond dimension to limit memory usage while simultaneously producing a patch decomposition that concentrates effort where the result is most complex, thus delivering the same kind of problem–tailored efficiency achieved by pQTCI.

Chapter 5

Numerical results

Chap. 3 and Chap. 4 introduced our *patched QTCI* and *patched MPO–MPO contraction* algorithms in detail. By extending the state-of-the-art implementation of the QTCI algorithm, we have implemented a *divide-and-conquer* version of TCI that targets scenarios in which the standard routine may struggle.

In this chapter we first present representative benchmarks (Sec. 5.1 and Sec. 5.2) that highlight the performance of the new routines. We then apply them to two physics problems that have previously posed computational bottlenecks: the computation of the bare susceptibility for the 2D Hubbard model with momentum dependence (Sec. 5.3) and vertex contractions during the solution of the Bethe-Salpeter for the single-impurity Anderson model (Sec. 5.4). Both cases were recently addressed with QTCI or patched quantics SVD–based tensor trains approaches [27, 50]. Our patched QTCI method complements and extends those efforts, demonstrating improved efficiency on the same benchmark tasks.

Our calculations are constructed on the already mature julia packages `TensorCrossInterpolation.jl` [24] and `QuanticsTCI.jl` [25], which we have extended with the additional features required for the present patched applications.

5.1 Approximation of 2D Green's functions

We begin with the toy Green's function

$$G(\mathbf{k}) \;=\; \frac{1}{\omega + \mu - \varepsilon_{\mathbf{k}} + i\delta}, \tag{5.1}$$

where the non-interacting dispersion is taken as $\varepsilon_{\mathbf{k}} = -2\cos k_x - 2\cos k_y$ and we set the chemical potential to $\mu = 0$. Eq. (5.1) is patterned after the Matsubara Green's function of the two-dimensional Hubbard model at finite temperature [54],

$$G(\boldsymbol{k}, i\nu) \;=\; \frac{1}{i\nu + \mu - \varepsilon_{\mathbf{k}} - \Sigma(\boldsymbol{k}, i\nu)}, \tag{5.2}$$

with two deliberate simplifications: ω plays the role of the (real) self-energy Σ, while δ mimics the Matsubara frequency $\nu = (2n + \xi)\pi/\beta$ ($\xi = 0, 1$ for bosons and fermions) and thus encodes the temperature. Accordingly we treat ω as a fixed input—one may think of it as the self-energy obtained from the previous Dyson iteration—whereas δ is varied to emulate different temperatures.

The figure below shows a density plot of $\mathrm{Re}\, G(\mathbf{k})$ for the representative choice $\omega = 10^{-1}$:

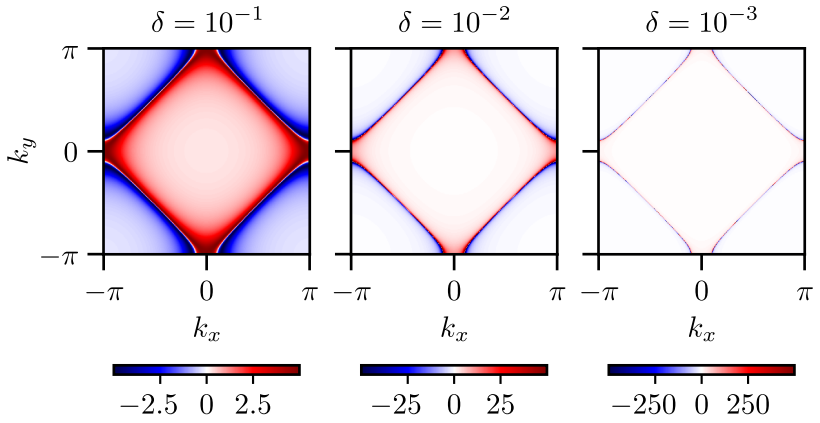


Figure 5.1: Heatmap of the real part of the Green's function in Eq. (5.1), $\mathrm{Re}\,(G(\boldsymbol{k}))$ for different values of δ and fixed $\omega = 10^{-1}$.

Fig. 5.1 shows that the parameter δ controls how sharply the Green's function is localised: as $\delta \to 0$ the poles narrow and $G(\mathbf{k})$ becomes increasingly singular. This makes the function an ideal testbed for patched QTCI.

We discretise $G(\mathbf{k})$ by quantics rebasing,

$$G(\boldsymbol{k}) \;\longrightarrow\; G_{\boldsymbol{\sigma}} = G\big(\boldsymbol{k}(\boldsymbol{\sigma})\big), \tag{5.3}$$

with bit string $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_{\mathcal{R}})$ in the fused ordering, or $\boldsymbol{\sigma} = (\sigma_{k_x 1}, \ldots, \sigma_{k_y \mathcal{R}})$ in the interleaved ordering. Throughout we use $\mathcal{R} = 15$ bits per $\boldsymbol{k}$-component.

After setting a patch bond–dimension cap χ_{patch} and a global tolerance $\tau = 10^{-7}$, we monitor convergence of pQTCI via the pointwise error

$$\varepsilon_{\log}(\mathbf{k}) = \log_{10} \frac{\big|\mathrm{Re}\, \widetilde{G}(\mathbf{k}) - \mathrm{Re}\, G(\mathbf{k})\big|}{\|\,\mathrm{Re}\, \widetilde{G}\|_\infty}, \tag{5.4}$$

where $\|\,\mathrm{Re}\, \widetilde{G}\|_\infty$ is the maximum of $|\,\mathrm{Re}\, \widetilde{G}|$ over all sampling points $\boldsymbol{k}(\boldsymbol{\sigma})$ used in every patch $\mathrm{Re}\, \widetilde{G}^{p_1,\ldots,p_{\bar{\ell}}}$ produced by the patched QTCI routine.

Fig. 5.2 compares the patched QTCI approximation with the exact real part of the Green's function for three values of the broadening parameter δ. The top row shows $\mathrm{Re}\, G(\boldsymbol{k})$ reconstructed from the patched tensor trains, while the bottom row plots the local error $\varepsilon(\boldsymbol{k})$ defined in Eq. (5.4) over the entire Brillouin zone $[-\pi, \pi]^2$. To evaluate the approximate tensor $\mathrm{Re}\, \widetilde{G}_{\boldsymbol{\sigma}}$ on a uniform

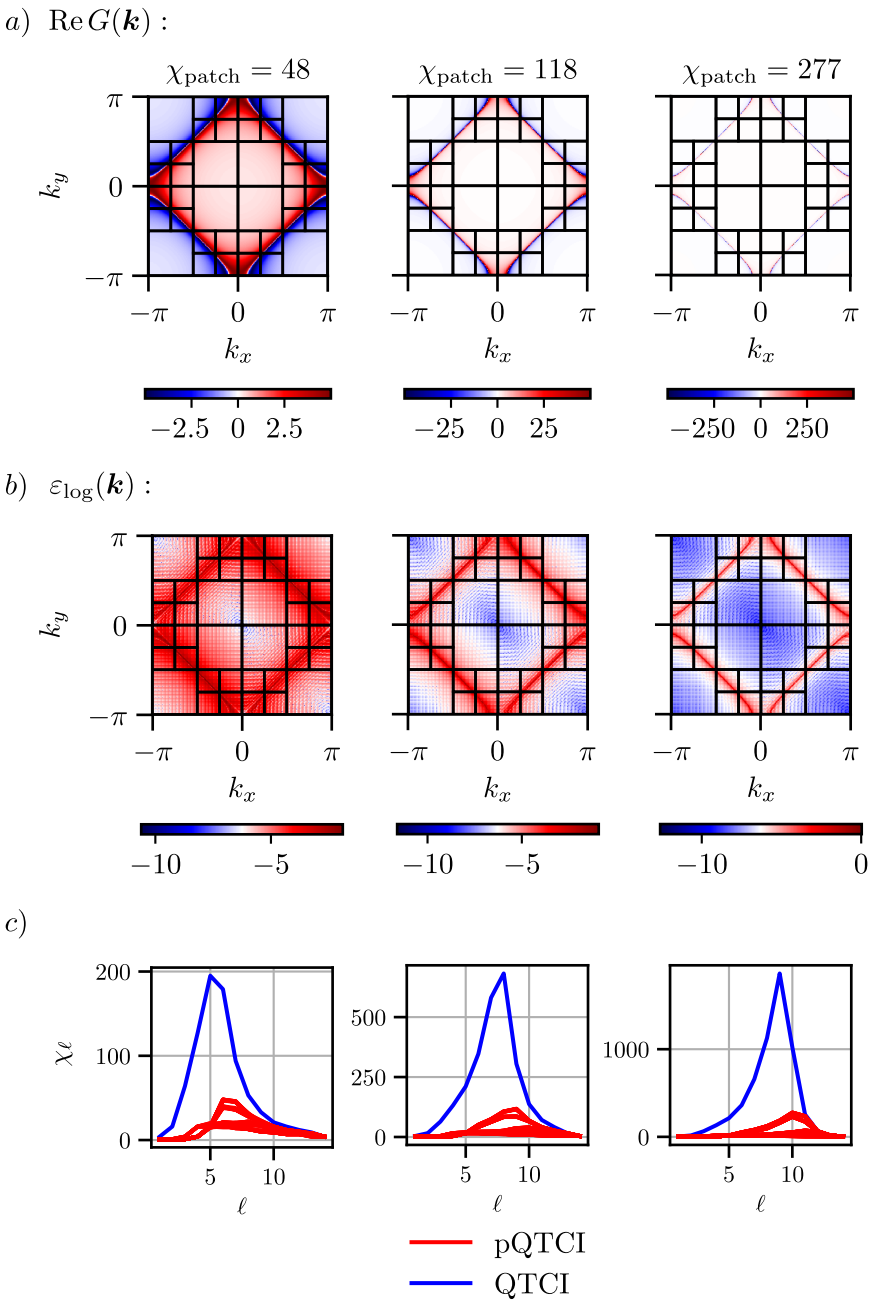


Figure 5.2: Patched QTCI approximation of Re $G(\boldsymbol{k})$. (*a*) Heatmaps of the patched tensor train evaluated on $[-\pi, \pi]^2$ for bond–dimension caps $\chi_{\text{patch}} = 48, 118, 277$ (corresponding to $\delta = 10^{-1}, 10^{-2}, 10^{-3}$, respectively). (*b*) Log–error $\varepsilon(\boldsymbol{k})$ [Eq. (5.4)] for the same patched approximations. (*c*) Comparison of the bond-dimension profiles for the pQTCI and standard QTCI representations of Re $G(\boldsymbol{k})$. Every patch in the pQTCI result adheres to the prescribed cap χ_{patch}.

$\boldsymbol{k}$-grid we first invert the mapping $\boldsymbol{\sigma} \mapsto \boldsymbol{k}(\boldsymbol{\sigma})$ and then, after resumming the whole patches set $\mathrm{Re}\big(\widetilde{G}^{p_1,\dots,p_{\bar{\ell}}}\big)$ to a single TT approximation $\mathrm{Re}\big(\widetilde{G}^{+}_{\boldsymbol{\sigma}}\big)$, we evaluate the correspondent $\boldsymbol{\sigma}(\boldsymbol{k})$ tensor value for each $\boldsymbol{k}$ point of the domain. Fig. 5.2 also displays the bond–dimension distributions for both the patched and the single-TT approximations of $\mathrm{Re}\, G(\boldsymbol{k})$ at the three caps $\chi_{\mathrm{patch}} = 48, 118, 277$ (corresponding to $\delta = 10^{-1}, 10^{-2}, 10^{-3}$). The patches shows a marked rank reduction compared to the QTCI TT. Moreover we can distiguish two patch groupings: large, low-rank patches span the "trivial" regions of the Brillouin zone, while smaller patches—carrying the higher bond dimensions—track the non-trivial, sharply structured areas. The bond dimensions in show this patch separation.

Despite the fact that $\varepsilon(\boldsymbol{k})$ does not drop everywhere below the target tolerance $\tau = 10^{-7}$, its *spatial average* error is of that order. For the same parameters the conventional (Q)TCI routine attains comparable accuracy, as illustrated in Fig. 5.3 for $\delta = 10^{-1}$. A one–dimensional cut at $k_y = \pi/2$ (Fig. 5.4) con-

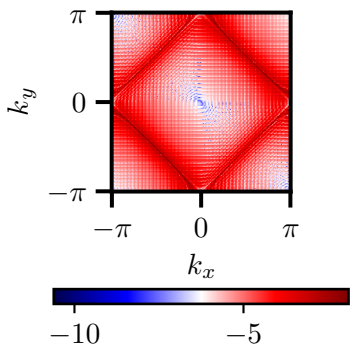


Figure 5.3: Local error $\varepsilon(\boldsymbol{k})$ for a standard QTCI approximation of $\mathrm{Re}\, G(\boldsymbol{k})$ at $\delta = 10^{-1}$.

firms that the patched approximation faithfully reproduces the sharp features of $\mathrm{Re}\, G(\boldsymbol{k})$ for all three δ values.

We now benchmark patched QTCI (pQTCI) against the standard QTCI routine for the Green's function introduced above. For each broadening $\delta \in \{10^{-1}, 10^{-2}, 10^{-3}\}$ we measure

- the *run time* on an Intel® Xeon® W-2245 CPU @ 3.90 GHz, and
- the *memory footprint*, here defined as the total number of floating-point parameters in all patches $\mathrm{Re}\, \widetilde{G}^{p_1,\dots,p_{\bar{\ell}}}$.

The tensor $\mathrm{Re}\, G_{\boldsymbol{\sigma}}$ is discretised with $\mathcal{R} = 15$ bits per momentum component and approximated to a tolerance $\tau = 10^{-7}$. We scan the bond–dimension cap χ_{patch} and compare fused and interleaved bit orderings.

Figures 5.5– 5.6 reveal three main trends:

1. For $\delta = 10^{-2}$ and 10^{-3} the patched routine beats standard QTCI in memory requirements. CPU rutime is smaller only with fused intex ordering. The advantage grows as the poles sharpen (smaller δ).

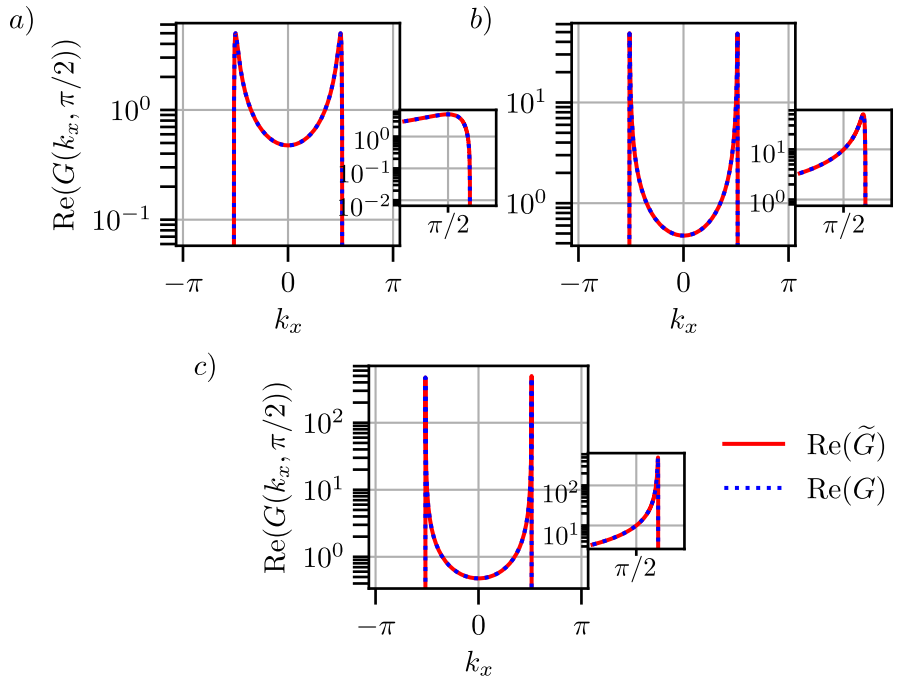


Figure 5.4: One–dimensional slice, $\mathrm{Re}\, G(k_x, k_y = \pi/2)$, comparing the patched approximation (solid red) with the exact function (dotted blue) for $\delta = 10^{-1}$ (a), $\delta = 10^{-2}$ (b), and $\delta = 10^{-3}$ (c). Zoomings centered on the peaks are shown.

2. Each curve exhibits an optimal $\chi^{\text{best}}_{\text{patch}}$: setting the cap too low triggers the "overpatching" effect discussed in Sec. 3.2, inflating the patch count without reducing ranks further.

3. Surprisingly, the fused ordering runtimes performs better than the interleaved one, although,in most cases, both respect the theoretical patch bounds of Eq. (3.29) for the total number of patches N_{patch}. (cf. Sec. A.1 of Appendix A for more details).

For the broadest line, $\delta = 10^{-1}$, pQTCI offers no gain—the function is already smooth enough that a single TT suffices, and the extra slicing merely adds overhead.

A global view of the "return on investment" is given in Fig. 5.7, which plots the ratio of plain QTCI result to the best patched result (in parameters and run time) as a function of δ. The larger is the ratio, the greater the benefit of using pQTCI.

In summary, adaptively partitioning the domain allows one to focus bond dimension where it is truly needed, yielding significant savings for sharply localised Green's functions, while incurring in overhead ("overpatching") when the function is already smooth.

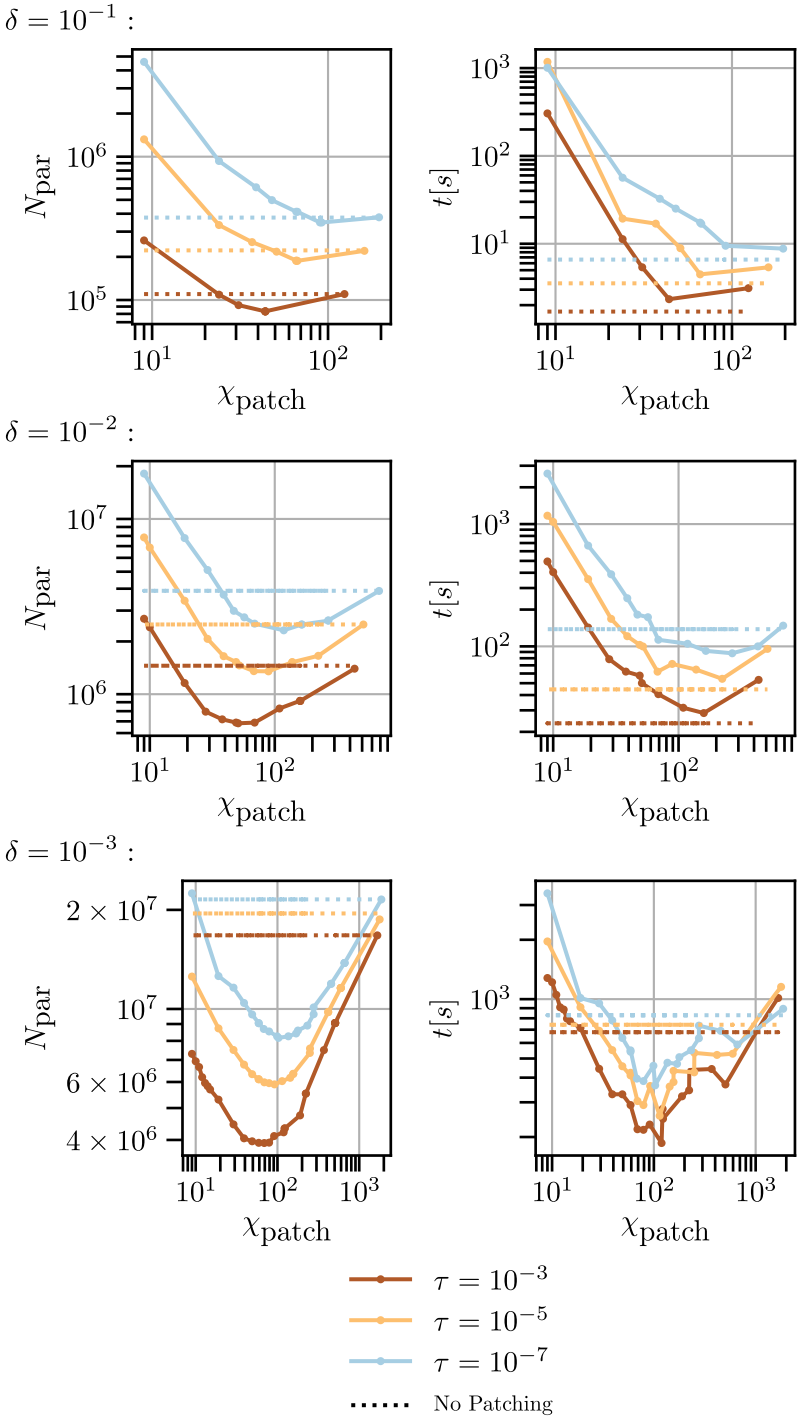


Figure 5.5: **Fused ordering.** Total parameter count (left) and CPU run time (right) versus bond-cap χ_{patch} for $\delta = 10^{-1}, 10^{-2}, 10^{-3}$. Dotted lines show the corresponding standard-QTCI values.

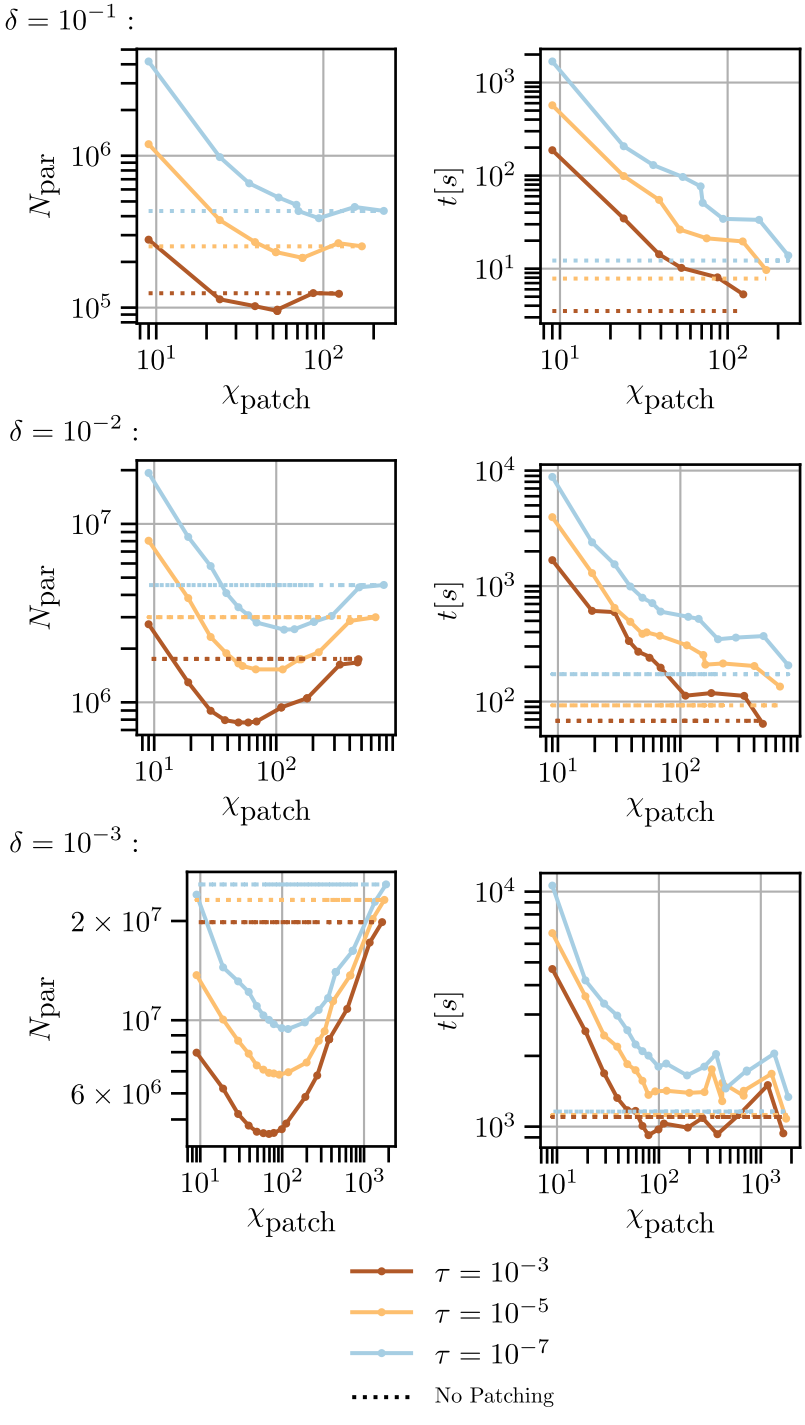


Figure 5.6: **Interleaved ordering.** Same data as Fig. 5.5, but with interleaved bit strings.

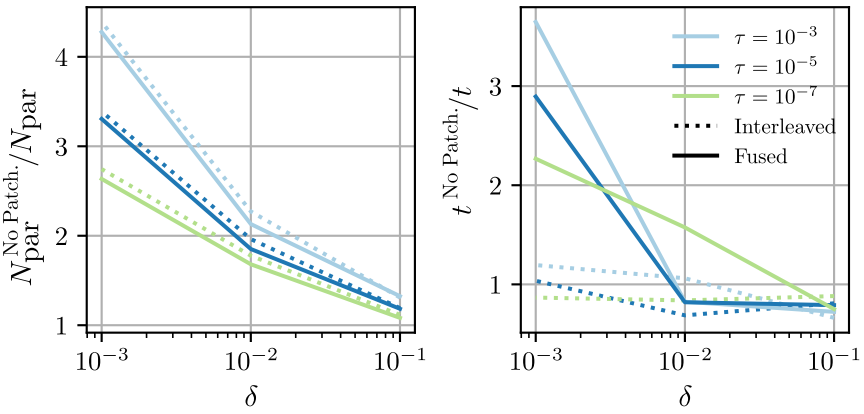


Figure 5.7: Parameter and run-time ratios between the best patched approximation (at $\chi_{\text{patch}}^{\text{best}}$) and a single-TT QTCI approximation, as a function of the broadening δ.

5.2 Benchmarking of Patched MPO-MPO Contractions

Consider the two model functions

$$f(\boldsymbol{x}) = \sum_{j=1}^{4} e^{-(\boldsymbol{x}-\boldsymbol{x}_j)^2/w_j^2}, \quad g(\boldsymbol{x}) = \sum_{j=1}^{4} e^{-|\boldsymbol{x}-\boldsymbol{x}'_j|/w_j}, \tag{5.5}$$

with $\boldsymbol{x}_j = (\cos\phi_j, \sin\phi_j)$, $\phi_j = (j-\frac{1}{2})\frac{\pi}{2}$, $w_j = 2^{-(j+1)}$, and $\boldsymbol{x}'_j = \boldsymbol{x}_j + (2w_j, w_j)$. Fig. 5.8 shows heatmaps of f and g.

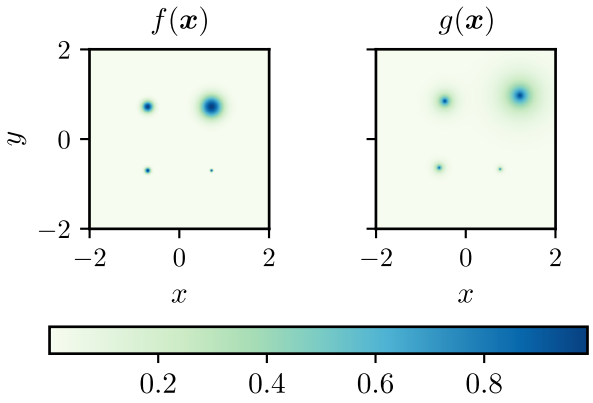


Figure 5.8: Heatmap of $f(\boldsymbol{x})$ and $g(\boldsymbol{x})$ defined in Eq. (5.5).

We convert each function to a tensor via quantics rebasing (fused or interleaved),

$$f(\boldsymbol{x}) \mapsto \mathcal{F}_{\boldsymbol{\sigma}}, \quad g(\boldsymbol{x}) \mapsto \mathcal{G}_{\boldsymbol{\sigma}}. \tag{5.6}$$

Applied to $\mathcal{F}_{\boldsymbol{\sigma}}$ and $\mathcal{G}_{\boldsymbol{\sigma}}$, the patched QTCI routine yields collections of tensor–train patches $\{\mathcal{F}_{\boldsymbol{\sigma}}^{p_1,\ldots,p_{\bar{\ell}}}\}$ and $\{\mathcal{G}_{\boldsymbol{\sigma}}^{p_1,\ldots,p_{\bar{\ell}}}\}$. We use these patch representations, after MPS-to-MPO folding (cf. Eq. (4.2)), as input for our *patched MPO-MPO contraction* routines.

Matrix multiplication

Let the tensors $\mathcal{F}_{\boldsymbol{\sigma}}$ and $\mathcal{G}_{\boldsymbol{\sigma}}$ be stored in the *interleaved* quantics format

$$\boldsymbol{\sigma} = (\sigma_{x,1}, \sigma_{y,1}, \sigma_{x,2}, \sigma_{y,2}, \ldots, \sigma_{x,\mathcal{R}}, \sigma_{y,\mathcal{R}}), \qquad \mathcal{R} = 17.$$

When pQTCI is applied, the slicing (projection) order determines the resulting patch tiling:

- **Row patching:** $p_{x,1} \to p_{x,2} \to \cdots \to p_{x,\mathcal{R}}$ (top–left panel of Fig. 5.9);
- **Column patching:** $p_{y,1} \to p_{y,2} \to \cdots \to p_{y,\mathcal{R}}$ (centre–left panel);
- **Interleaved patching:** $p_{x,1} \to p_{y,1} \to p_{x,2} \to \cdots \to p_{x,\mathcal{R}}$ (bottom–left panel).

Because pQTCI is adaptive, the actual patch pattern follows the feature structure of the functions; from left to right, each column in Fig. 5.9 correspond to the factors in Eq. (5.5) and their patched MPO contraction, respectively.

Each tensor-train patch can be folded into an MPO (Eq. (4.2)) and fed to the patched MPO–MPO contraction routines. For instance, the two–dimensional convolution

$$h(x,y) = \int ds f(x,s)s(x,s) \tag{5.7}$$

maps to the patched "matrix multiplication"

$$\mathcal{H}_{\boldsymbol{\sigma}\boldsymbol{\sigma}''} = \sum_{\boldsymbol{\sigma}'} \mathcal{F}_{\boldsymbol{\sigma}\boldsymbol{\sigma}'}\mathcal{G}_{\boldsymbol{\sigma}'\boldsymbol{\sigma}''}, \tag{5.8}$$

where only mutually compatible patch pairs need be multiplied. Although the labels "row" and "column" may seem inverted when viewed against the two-dimensional tilings in Fig. 5.9, they are perfectly natural in the *matrix* interpretation of each tensor—where the x-bit string forms the row index and the y-bit string the row index.

Fig. 5.9 compares the patch layouts that pQTCI produces for f and g under combinations of the three slicing orders introduced above. Column (1) shows the set $\{\widetilde{\mathcal{F}}^{p_1,\ldots,p_{\bar{\ell}}}\}$ that enters as the left factor of the convolution; column (2) shows the right–factor patches $\{\widetilde{\mathcal{G}}^{p_1,\ldots,p_{\bar{\ell}}}\}$; column (3) displays the patches $\{\widetilde{\mathcal{H}}^{p_1,\ldots,p_{\bar{\ell}}}\}$ produced by the patched MPO–MPO contraction. Colours encode individual patches, allowing one to see at a glance how the domain is split and how the resulting product inherits the finer of the two tilings. From the two-dimensional tilings one sees that the product tensor adopts the row patching of the left factor along the x-direction, while its y-direction patching follows the column pattern of the right factor.

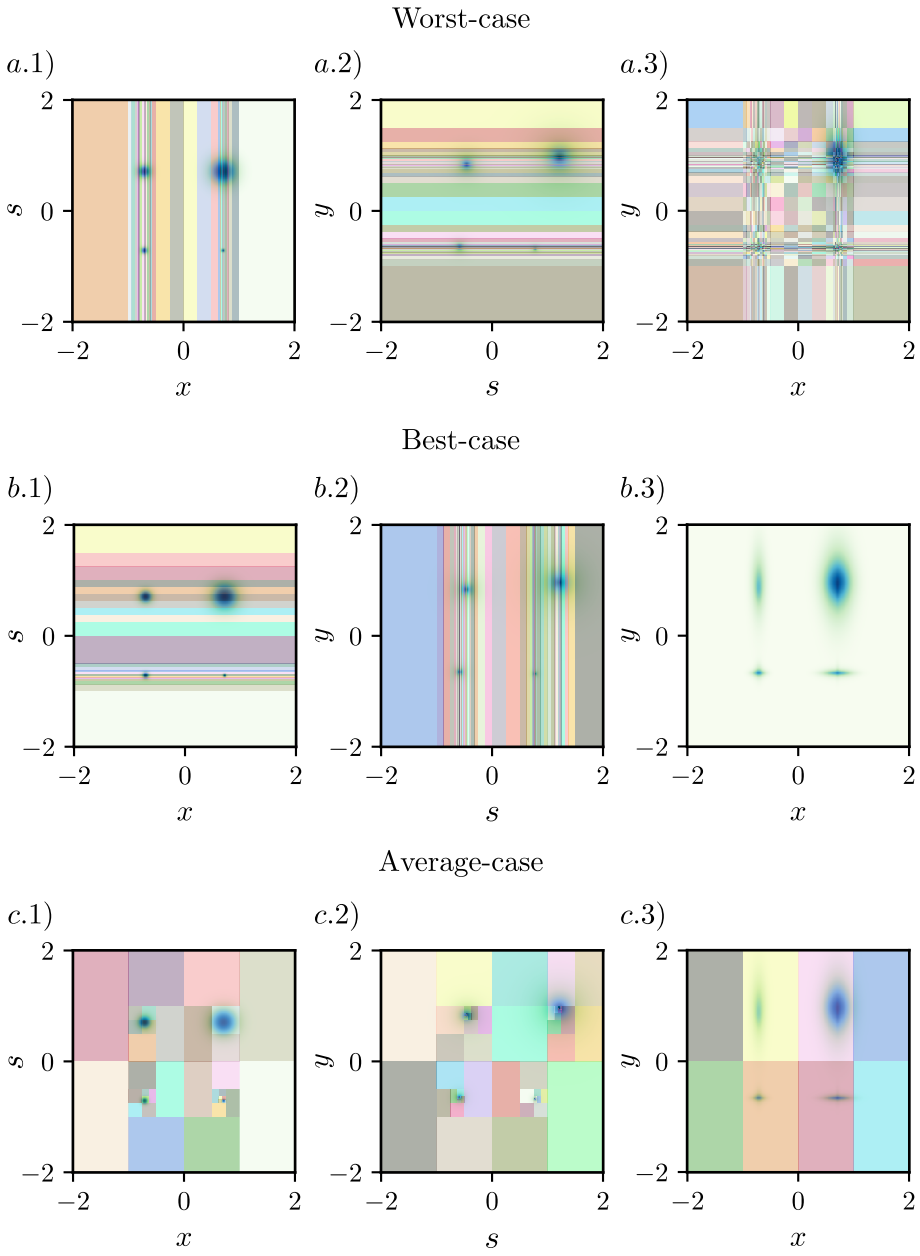


Figure 5.9: Patch tilings for the two factors $\mathcal{F}$ (panels $a-c.1$) and $\mathcal{G}$ ($a-c.2$), together with the corresponding patched product $\mathcal{H}$ ($a-c.3$). Rows illustrate three representative patching contraction strategies as in Fig. 4.4: (a) rows patching against column patching, i.e. *worst-case* in terms of contraction count; (b) column patching against row patching — the *best-case*; (c) interleaved patching (alternating x and y) — the *average* scenario. Each patch is rendered in a distinct colour for more clarity.

We now compare the patched–contraction schemes of Fig. 5.9 with a single, non-patched MPO–MPO contraction. Fig. 5.10 reports the wall-clock time versus the product of the patch bond caps $\chi_{\text{patch},\mathcal{F}}$ and $\chi_{\text{patch},\mathcal{G}}$–fixed in the preliminary pQTCI runs– for the three patch layouts, measured on an Intel® Xeon® W-2245 CPU @ 3.90 GHz.

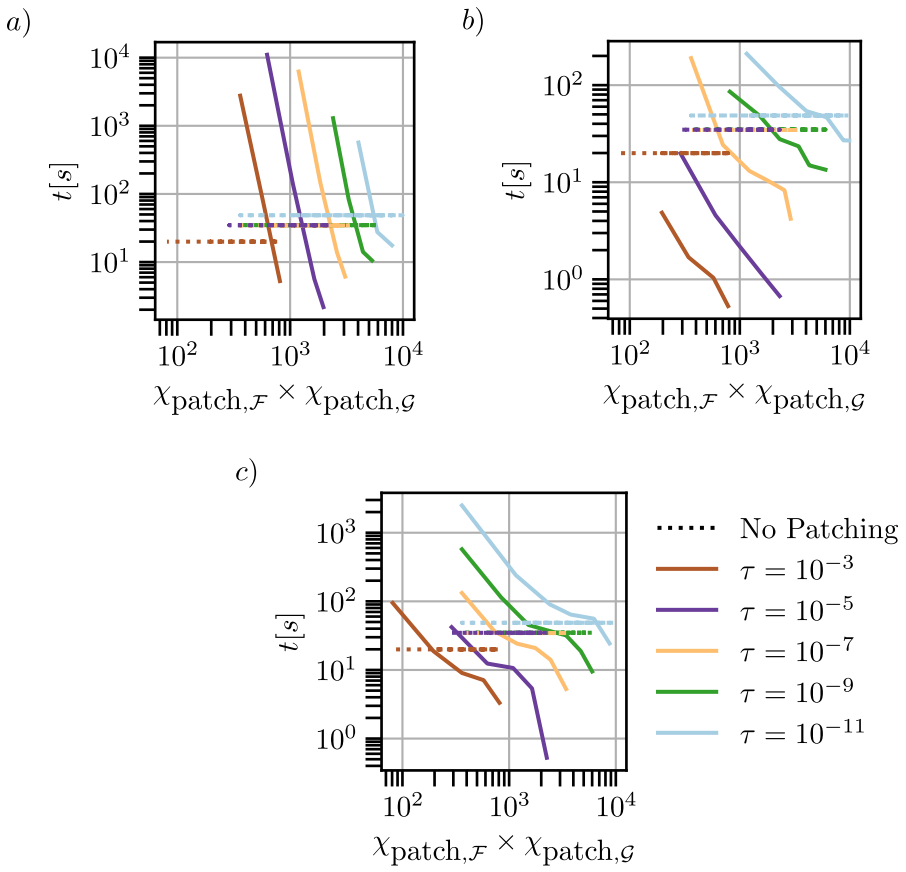


Figure 5.10: Run-time scaling of the patched MPO–MPO contraction for the (*a*) worst, (*b*) best, and (*c*) mixed (average) patch arrangements as in Fig. 5.9. Dotted lines mark the reference time of a monolithic contraction with the same tolerance.

Key observations:

- *Worst-case layout*—column patches on both factors—exhibits the slowest scaling, consistent with the $\mathcal{N}^2_{\text{patch}}$ contraction count predicted in Eq. (4.28).
- For very small χ_{patch} all three patch configurations rise again, signalling *over-patching*: the initial pQTCI step subdivides $\mathcal{F}$ and $\mathcal{G}$ so aggressively that the overhead of launching thousands of tiny contractions outweighs the benefit of lower ranks.

- Around an intermediate, problem-dependent $\chi_{\text{patch}}^{\text{opt}}$ the patched approach becomes advantageous. In the best-case arrangement [panel (b)] the speed-up reaches an order of magnitude relative to the single-core baseline.

The speed-up is consistent with the limits on N_{patch} derived in Eqs (4.29)-(4.33); a detailed analysis is provided in Sec.A.2 of Appendix A.

Element-wise multiplication

Let $\mathcal{F}_{\boldsymbol{\sigma}}$ and $\mathcal{G}_{\boldsymbol{\sigma}}$ be represented in either the *interleaved* or *fused* quantics format. After pQTCI compression and MPO diagonalisation (Eq. (4.18)), the two MPOs can be fed to the *patched element-wise contraction* routine, which produces

$$\big(\mathcal{F}\mathcal{G}\big)_{\boldsymbol{\sigma}} = \mathcal{F}_{\boldsymbol{\sigma}}\mathcal{G}_{\boldsymbol{\sigma}} \tag{5.9}$$

i.e. the tensorised counterpart of the pointwise product

$$(fg)(x,y) = f(x,y)g(x,y). \tag{5.10}$$

Knowing the analytic forms of f and g, we monitor the patchwise error with the metric of Eq. (5.4) (where $\boldsymbol{k} \to \boldsymbol{x}$). Fig. 5.11 shows the local error across the domain for a set tolerance $\tau = 10^{-7}$—applied both in pQTCI and as the square root of the singular value cutoff threshold in each *zip-up* contraction.

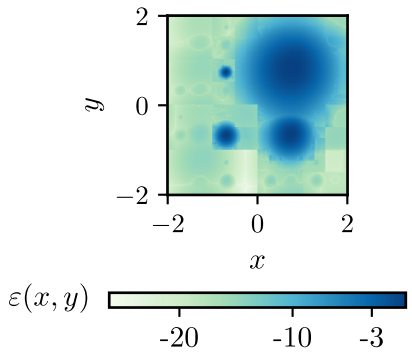


Figure 5.11: Pointwise error $\varepsilon(\boldsymbol{x})$ of the patched element-wise product for $\tau = 10^{-7}$, $\chi_{\text{patch},\mathcal{F}} = \chi_{\text{patch},\mathcal{G}} = 68$ with interleaved ordering.

Because a diagonal MPO is non-zero only on matching input/output indices, two patches contribute to the product only if they cover the *same* (x, y) tile. Fig. 5.12 illustrates this: panels (a) and (b) show the patch layouts of the patched tensors $\mathcal{F}$ and $\mathcal{G}$; panel (c) displays the resulting patches of the product, which appear only where the patched in (a) and (b) overlap (cf. Fig. 4.3). The result is a patched tensor $\big(\mathcal{F}\mathcal{G}\big)$ that inherits – in each region of the domain – the smallest subdivision possible between the two factors.

Element-wise multiplication enjoys the most relaxed patch-count bound (Eq. (4.23); see also Sec. A.2 in Appendix A), and the timing data in Fig. 5.13

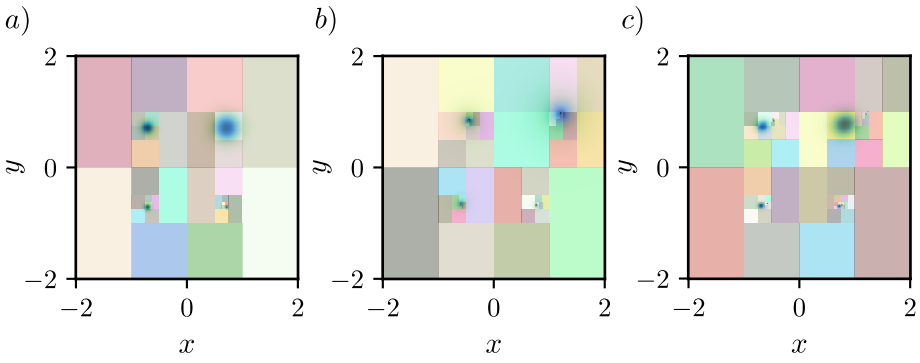


Figure 5.12: Patch tilings of (a) $\mathcal{F}$, (b) $\mathcal{G}$ and (c) their patched element-wise product $(\mathcal{FG})$. Only overlapping tiles produce a non-zero result.

confirm the advantage. With fused ordering the patched routine achieves up to a five-fold speed-up over a monolithic contraction; interleaved ordering is still faster than the baseline in for some choices of $\chi_{\text{patch},\mathcal{F}}$ and $\chi_{\text{patch},\mathcal{G}}$, though by a smaller margin.

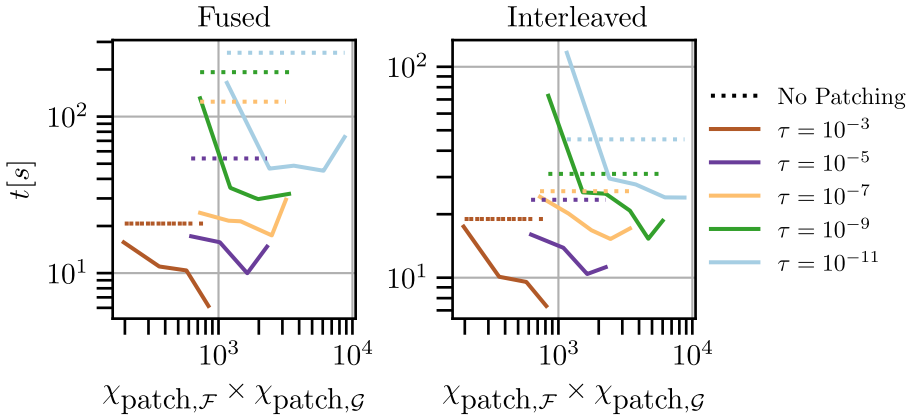


Figure 5.13: Runtimes for patched element-wise multiplication versus the product of the bond caps $\chi_{\text{patch},\mathcal{F}} \times \chi_{\text{patch},\mathcal{G}}$. Solid lines: patched contraction with fused or interleaved index ordering; dotted lines: reference time for a single, non-patched contraction at the same tolerance.

Adaptive matrix multiplication

As a proof of concept for the *adaptive patched MPO–MPO contraction* – or *adaptive matrix multiplication* – algorithm we revisit the tensors $\mathcal{F}_{\boldsymbol{\sigma}}$ and $\mathcal{G}_{\boldsymbol{\sigma}}$, now stored in the *fused* quantics ordering. Suppose we are interested only in a compact representation of their matrix product $\mathcal{H}_{\boldsymbol{\sigma}\boldsymbol{\sigma}''}$ (cf. Eq. (5.8)) and we

expect the result to develop sharp, localised structures. The adaptive workflow proceeds as follows:[1]

(i) Convert $\mathcal{F}$ and $\mathcal{G}$ to MPS form $\widetilde{\mathcal{F}}$ and $\widetilde{\mathcal{G}}$ using (Q)TCI.

(ii) Run the adaptive patched contraction, which recursively slices the two MPOs whenever an intermediate bond exceeds the cap χ_{patch}. The procedure stops when every partial product satisfies the target tolerance τ (see Fig. 4.6).

(iii) Unfold each resulting patch $\widetilde{\mathcal{H}}^{p_1,\dots,p_{\bar{\ell}}}$ back into an MPS for downstream calculations.

Fig. 5.14 demonstrates the outcome. Panel (a) shows how the adaptive algorithm concentrates patches exactly where $\mathcal{H}$ is most structured, while panel (b) compares memory requirements with those of a "naive" single MPO–MPO contraction. The feature–aware tiling reduces the parameter count—analogous to the savings pQTCI achieves over standard QTCI for function approximation.

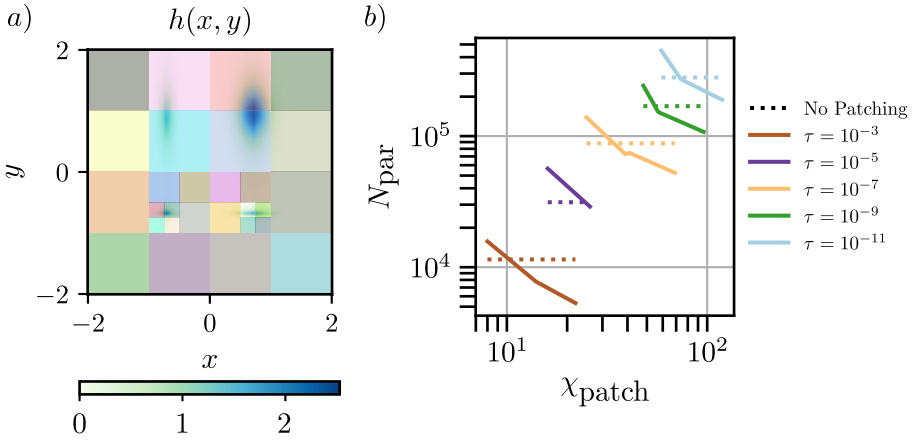


Figure 5.14: Adaptive patched contraction. (a) Patch layout of the product tensor $\mathcal{H}_{\boldsymbol{\sigma}} = h(\boldsymbol{x}(\boldsymbol{\sigma}))$ obtained by the adaptive contraction algorithm; patches cluster around the high-feature regions. (b) Total number of floating-point parameters versus bond–dimension cap χ_{patch}. Solid line: adaptive contraction; dotted line: monolithic MPO–MPO multiplication at the same accuracy.

5.3 Bare Susceptibility Calculation

We consider now the momentum dependence in the case of the single-orbital two-dimensional Hubbard model on the square lattice at half filling. The Hamiltonian of the Hubbard model reads

[1] The code used here incorporates several bug fixes that became available only during the final stage of writing, thus the limited numerical results.

$$\mathcal{H} = \sum_{\langle ij\rangle,\sigma} \hat{c}^{\dagger}_{i\sigma}\hat{c}_{\sigma} + U\sum_{i} \hat{n}_{i\uparrow}\hat{n}_{i\downarrow} - \mu\sum_{i,\sigma}\hat{n}_{i\sigma} \tag{5.11}$$

where $\hat{c}^{\dagger}_{i\sigma}$ is the creation operator of an electron with spin σ at site i and $\hat{n}_{i\sigma} = \hat{c}^{\dagger}_{i\sigma}\hat{c}_{i\sigma}$. U is the onsite repulsion and μ is the chemical potential ($\mu = U/2$ at half filling). The nearest-neighbour hopping is set to one. In the FLEX approximation [55], for a paramagnetic state the self-energy is approximated as

$$\Sigma(\boldsymbol{k}, i\nu) = \frac{1}{\beta N_{\boldsymbol{k}}}\sum_{\boldsymbol{q},i\omega} V(\boldsymbol{q}, i\omega)G(\boldsymbol{k}-\boldsymbol{q}, i\nu - i\omega) \tag{5.12}$$

where $N_{\boldsymbol{k}}$ is the size of the two-dimensional momentum grid, $\boldsymbol{q}$ is a bosonic momentum, ν and ω are Matsubara frequencies, fermionic and bosonic, respectively, and β is the inverse of the temperature. The effective interaction is defined as

$$V(\boldsymbol{q}, i\omega) = U^2\left(\frac{3}{2}\chi_{\mathrm{s}}(\boldsymbol{q}, i\omega) + \frac{1}{2}\chi_c(\boldsymbol{q}, i\omega) - \chi_0(\boldsymbol{q}, i\omega)\right) \tag{5.13}$$

where we introduced the bare $\chi_0(\boldsymbol{q}, i\omega)$, spin $\chi_{\mathrm{s}}(\boldsymbol{q}, i\omega)$ and charge $\chi_c(\boldsymbol{q}, i\omega)$ susceptibilities. In particular the bare susceptibility reads:

$$\chi_0(\boldsymbol{q}, i\omega) = -\frac{1}{N_{\boldsymbol{k}}\beta}\sum_{\boldsymbol{k},i\nu} G(\boldsymbol{k}+\boldsymbol{q}, i\nu + i\omega)G(\boldsymbol{k}, i\nu) \tag{5.14}$$

where the Green's function in the Matsubara axis is given by

$$G(\mathbf{k}, i\nu) \;=\; \frac{1}{i\nu + \mu - \varepsilon_{\mathbf{k}} - \Sigma(\boldsymbol{k}, i\nu)}, \tag{5.15}$$

with $\varepsilon_{\boldsymbol{k}} = -2\cos(k_x) - 2\cos(k_y)$ and $\mu = 0$. Hence the bare susceptibility $\chi_0(\boldsymbol{q}, i\omega)$ is a crucial ingredient in the self-consistent FLEX scheme, where one repeatedly updates the self-energy $\Sigma(\boldsymbol{k}, i\nu)$. Therefore, the convolution in Eq. (5.14) quickly becomes the dominant cost, in particular at low temperatures ($\beta \to \infty$). This is because the fermionic Matsubara grid $\nu_n = (2n+1)\pi/\beta$ grows denser while, at the same time, the Green's function $G(\mathbf{k}, i\nu)$ develops increasingly sharp, localized peaks (cf. Sec. 5.1); direct numerical evaluation is both memory- and time-intensive. These characteristics suggest that a domain-adaptive strategy–specifically, the pQTCI-based patched contraction routines introduced above–can alleviate the cost of the convolution by concentrating bond dimension only where the integrand is genuinely singular.

A direct tensor-train treatment of the convolution in Eq. (5.14) would require handling a six-legged tensor $G(\boldsymbol{k}+\boldsymbol{q}, i\nu + i\omega) = G_{\boldsymbol{\sigma}_{k_x}\boldsymbol{\sigma}_{k_y}\boldsymbol{\sigma}_{q_x}\boldsymbol{\sigma}_{q_x}\boldsymbol{\sigma}_{i\nu}\boldsymbol{\sigma}_{i\omega}}$, a challenging task even for QTCI. The remedy is to migrate the calculation to "real" space–time, where the convolution becomes an *element-wise* product of two functions, after Fourier transform (FT) (cf. Ref. [56]). We define the forward and inverse transforms of the Green's function as

$$G(\boldsymbol{k}, i\nu) = \frac{\beta}{\sqrt{N_{\boldsymbol{k}}}} \int_0^\beta d\tau \sum_{\boldsymbol{r}} e^{+\boldsymbol{k}\boldsymbol{r}+i\nu\tau} G(\boldsymbol{r}, \tau) \tag{5.16}$$

$$G(\boldsymbol{r}, \tau) = \frac{1}{\sqrt{N_{\boldsymbol{k}}}\beta} \sum_{\boldsymbol{k}, i\nu} e^{-\boldsymbol{k}\boldsymbol{r}-i\nu\tau} G(\boldsymbol{k}, i\nu) \tag{5.17}$$

where the sum $\sum_\nu$ runs over the Matsubara frequency grid $\nu_n = (2n+1)\pi/\beta$ with $n = -N_\nu/2, -N_\nu/2+1, \ldots, N_\nu/2-1$.

With these conventions the bare susceptibility reads

$$\chi_0(\boldsymbol{q}, i\omega) = \mathrm{FT}\left[\chi(\boldsymbol{r}, \tau)\right] = \mathrm{FT}\left[G(\boldsymbol{r}, \tau) G(-\boldsymbol{r}, -\tau)\right] \tag{5.18}$$

where $G(-\boldsymbol{r}, -\tau)$ is obtained from Eq. (5.17) by reversing the signs in the exponent. Thus, the convolution is replaced by a pointwise product in $(\boldsymbol{r}, \tau)$ space – well suited for the patched element-wise contraction routines introduced earlier.

The complete workflow for evaluating the bare susceptibility $\chi_0(\mathbf{q}, i\omega)$ is depicted in the flowchart of Fig. 5.15.

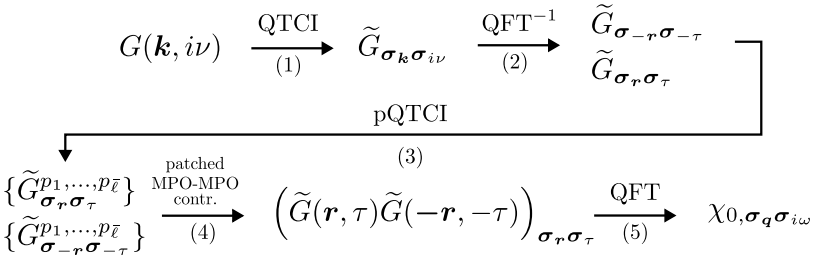


Figure 5.15: Pipeline for computing the bare susceptibility $\chi_0(\boldsymbol{q}, i\omega)$.

We proceed through the following stages:

(1) A QTCI compression of the Green's function in Eq. (5.15) is performed with the momentum/frequency index order $\boldsymbol{k}(\boldsymbol{\sigma_k}) = \mathbf{k}(\sigma_1, \ldots, \sigma_{2\mathcal{R}})$, $i\nu(\boldsymbol{\sigma}_{i\nu}) = i\nu(\sigma_{2\mathcal{R}+1}, \ldots, \sigma_{3\mathcal{R}})$, yielding the TT $\widetilde{G}_{\boldsymbol{\sigma_k}\boldsymbol{\sigma}_{i\nu}}$.

(2) The inverse quantics Fourier transform (QFT^{-1}) is applied separately to the $\boldsymbol{k}$ and $i\nu$ registers, producing the real–space tensors $\widetilde{G}_{\boldsymbol{\sigma_r}\boldsymbol{\sigma}_\tau}$ and $\widetilde{G}_{\boldsymbol{\sigma}_{-\boldsymbol{r}}\boldsymbol{\sigma}_{-\tau}}$, with reordered indices $\boldsymbol{r}(\boldsymbol{\sigma_r}) = \mathbf{r}(\sigma_{\mathcal{R}+1}, \ldots, \sigma_{3\mathcal{R}})$ and $\tau(\boldsymbol{\sigma}_\tau) = \tau(\sigma_1, \ldots, \sigma_{\mathcal{R}})$ (index reversal is an intrinsic feature of the QFT; see Appendix B).

(3) Two copies, $\widetilde{G}(\mathbf{r}, \tau)$ and $\widetilde{G}(-\mathbf{r}, -\tau)$, are each patched with pQTCI, by fixing a bond-dimension cap χ_{patch} and a compression tolerance τ.

(4) The two patch collections are multiplied patch-wise using the patched element-wise contraction routine rendering the product $\widetilde{G}(\mathbf{r}, \tau)\widetilde{G}(-\mathbf{r}, -\tau)$

(5) After summing the patches of the element-wise product into a single TT, a direct QFT maps the result back to $(\mathbf{q}, i\omega)$ space, yielding the sought susceptibility $\chi_0(\mathbf{q}, i\omega)$.

In step (1) we deliberately place the $\boldsymbol{k}$ bits before the $i\nu$ bits to minimise the intermediate bond dimension of $\widetilde{G}_{\boldsymbol{\sigma_k}\boldsymbol{\sigma}_{i\nu}}$. Because the quantics Fourier transform reverses the bit order within each register (see Appendix B), the subsequent real-space representation naturally acquires the ordering $\mathbf{r}(\boldsymbol{\sigma}_\mathbf{r}) = \mathbf{r}(\sigma_{\mathcal{R}+1}, \ldots, \sigma_{3\mathcal{R}})$ and $\tau(\boldsymbol{\sigma}_\tau) = \tau(\sigma_1, \ldots, \sigma_\mathcal{R})$, after index rearragnement. The rearrangement is performed by inverting the site tensors in real space in order to obtain a sequential index ordering, while conserving the links between the transformed site tensors. Hence, the frequency indices move from the back to the front of the TT in real space.

Figure Fig. 5.16 tracks the steps of the FLEX "bubble" χ_0 calculation:

(a) $|G(\mathbf{k}, i\pi/\beta)|$ on the Brillouin zone $[-\pi, \pi]^2$ together with the bond dimension profile of its QTCI representation.

(b) Resulting bare susceptibility $\chi_0(\mathbf{q}, \omega = 0)$ and the bond dimensions of its single-TT compression after the final QFT^{-1}.

(c) Patch layout produced by pQTCI for the real-space Green's function $G(\mathbf{r}, \tau = \beta)$. Colours indicate distinct patches; corresponding color coding is used for the bond dimensionsof each TT patch, contrasted with those of a global non-patched TT approximation of the real space Green's function.

The data in Fig. 5.16 were obtained with $\mathcal{R} = 13$ bits for each spatial and frequency variable, an inverse temperature $\beta = 100$, and a uniform accuracy target $\tau = 10^{-7}$ (applied to the QTCI and pQTCI compressions and–as τ^2–to the local truncation threshold in every MPO-MPO contraction).

Let us now benchmark the results of the computation. We first verify that the bit-depth chosen for the final calculation, $\mathcal{R} = 13$, yields a sufficiently converged susceptibility. Panel (a) of Fig. 5.17 plots the configuration-space error

$$\varepsilon(\mathcal{R}) = \frac{1}{\sqrt{N_\mathrm{err}}} \sqrt{\sum_{\boldsymbol{\sigma_q}, \boldsymbol{\sigma}_{i\omega}} \left| \chi_{0,\boldsymbol{\sigma_q},\boldsymbol{\sigma}_{i\omega}}^{(\mathcal{R},\tau)} - \chi_{0,\boldsymbol{\sigma_q},\boldsymbol{\sigma}_{i\omega}}^{(13,10^{-9})} \right|^2} \tag{5.19}$$

as a function of $\mathcal{R}$, for N_err points extracted randomly from the configuration space. While the curve indicates that χ_0 has not reached the desired precision, the residual error suffices for the present comparative study.

Panel (b) compares the total parameter count of the patched QTCI representation of $G(\mathbf{r}, \tau)$ (the same holds for $G(-\mathbf{r}, -\tau)$) with that of a single-TT QTCI approximation. The patched version saves memory, showing that its patch number respects the theoretical limit (Eq. (3.29)) and, by extension, the element-wise-product bound (Eq. (4.23)). Hence, we expect an advantage for the patched point-wise contraction.

Panel (c) reports the wall-clock time on an Intel® Xeon® E5-2680 v4 @ 2.40 GHz for the patched element-wise contraction versus the "naïve" single-shot contraction, scanned over tolerances τ and inverse temperatures β. The patched routine accelerates the calculation by up to an order of magnitude. For the coldest cases $\beta = 100, 1000$ the monolithic contraction could not be completed

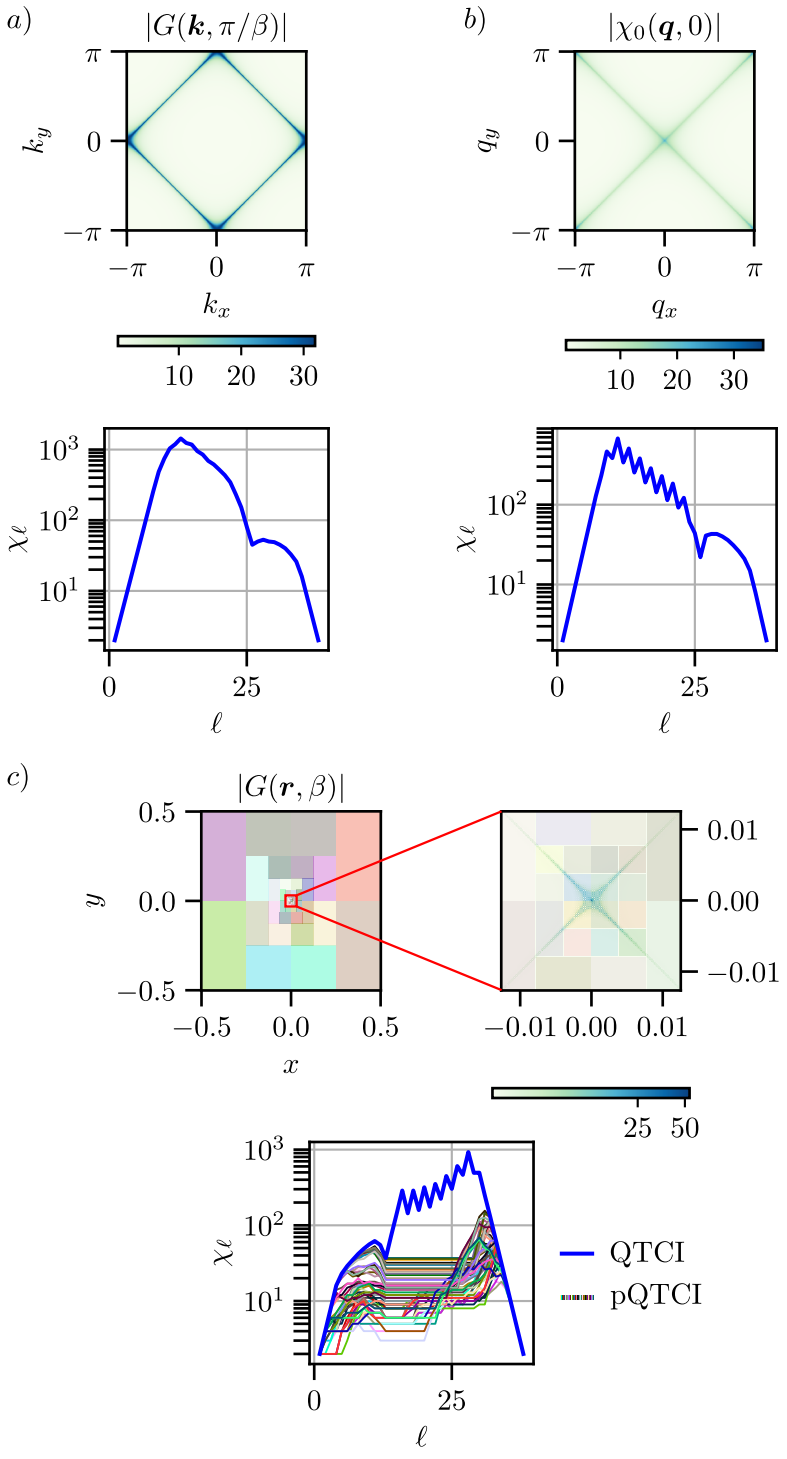


Figure 5.16: Heat-map overview of the bare-susceptibility workflow with $\mathcal{R} = 13$, $\beta = 100$ and tolerance $\tau = 10^{-7}$. (a) Absolute value of the Green's function $G(\boldsymbol{k}, i\nu)$ at the first positive fermionic Matsubara frequency, including QTCI bond dimensions (b) Bare susceptibility $\chi_0(\boldsymbol{q}, \omega = 0)$ with the bond dimensions of its single-TT approximation. (c) Adaptive patching pattern for $G(\mathbf{r}, \tau = \beta)$ with bond dimensions profile of each patch and analogous non-patched approximation.

owing to excessive memory demands, whereas the patched algorithm ran comfortably, illustrating the practical benefit of trading one large χ^4 operation for many lower-rank contractions.

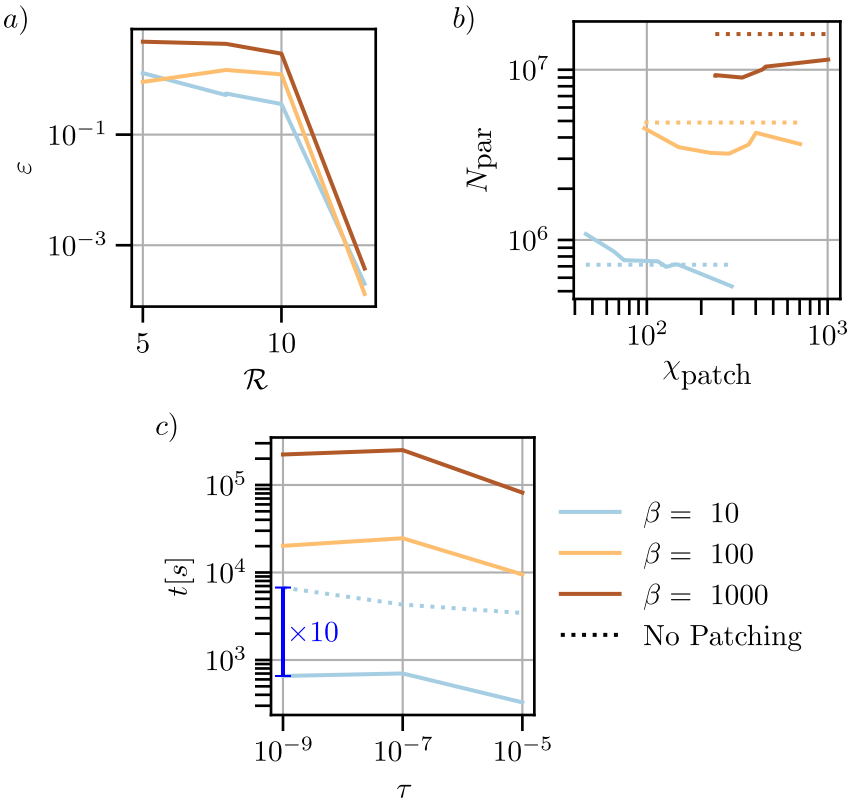


Figure 5.17: Performance of the bubble calculation. (a) Convergence of χ_0 with bit depth $\mathcal{R}$. (b) Total number of floating-point parameters in the patched (solid) versus single-TT (dotted lines) representations of $G(\mathbf{r}, \tau)$ at $\tau = 10^{-9}$ and $\mathcal{R} = 13$. (c) CPU time for the patched element-wise product compared with the monolithic contraction as a function of tolerance τ and at different inverse temperatures β.

Fig. 5.18 tracks the CPU time required for the patched element-wise product as the inverse temperature β is varied at several accuracy targets τ. Within numerical scatter the data follow an *approximately linear* growth, $t_{\text{CPU}} \propto \beta$ as indicated by the dotted line.

In this prototype calculation we applied pQTCI and patched contractions *only* to the element-wise multiplication step, which is the dominant bottleneck. A fully patched treatment of the momentum-space Green's function $G(\boldsymbol{k}, i\nu)$ would in principle yield an additional speed-up, but was not necessary to demonstrate the merits of the method: even this partial application suppresses the χ^4 scaling, delivers order-of-magnitude run-time gains, and keeps the memory footprint below that of the traditional approach. These results underline the practicality of the patched QTCI toolkit for low-temperature many-body calculations.

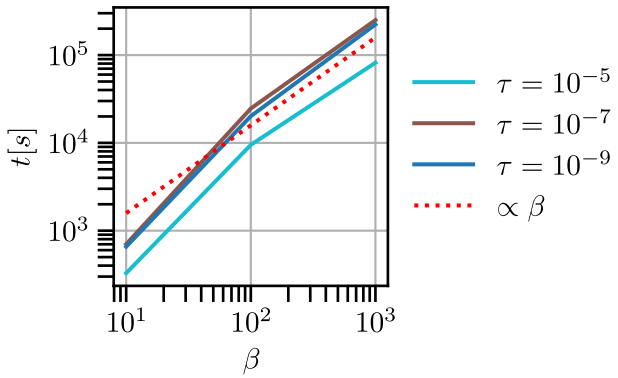


Figure 5.18: Run-time versus inverse temperature β for the patched element-wise contraction (solid curves) at three tolerances τ. Times are measured on an Intel® Xeon® E5-2680 v4 @ 2.40 GHz. The patched algorithm exhibits an almost linear β dependence as highlighted by the fit (dotted curve).

5.4 Bethe-Salpeter equations with pQTCI

The Hubbard atom is a reduction of the Hubbard lattice to a single, isolated site whose Hamiltonian is

$$\mathcal{H} = U\hat{n}_\uparrow\hat{n}_\downarrow - \mu(\hat{n}_\uparrow - \hat{n}_\downarrow) \tag{5.20}$$

where the hopping is naturally set to zero and μ and U are set to the same values as in Eq. (5.11) (half-filling). Despite its simplicity, this *atomic limit* reproduces many key features of the Hubbard model in the strong-coupling regime [57].

In the *single–impurity Anderson model* (SIAM) one embeds this single interacting site in a bath of non-interacting electrons. The SIAM Hamiltonian reads [58]

$$\begin{aligned}\mathcal{H} = &\sum_{\boldsymbol{k}\sigma} \varepsilon_{\boldsymbol{k}} \hat{c}^\dagger_{\boldsymbol{k}\sigma}\hat{c}_{\boldsymbol{k}\sigma} + \sum_{\boldsymbol{k}\sigma}\left(V_{\boldsymbol{k}}\hat{c}^\dagger_{\boldsymbol{k}\sigma}\hat{d}_\sigma + V^*_{\boldsymbol{k}}\hat{d}^\dagger_\sigma\hat{c}_{\boldsymbol{k}\sigma}\right) \\ &+ U\hat{n}_{d\uparrow}\hat{n}_{d\downarrow} + \varepsilon_d(\hat{n}_\uparrow + \hat{n}_\downarrow)\end{aligned} \tag{5.21}$$

where $\hat{d}^{(\dagger)}_\sigma$ annihilates (creates) an electron with spin σ in the impurity level $\varepsilon_d = -U/2$, $\hat{n}_{d,\sigma} = \hat{d}^\dagger_\sigma\hat{d}_\sigma$, and $V_{\boldsymbol{k}}$ hybridises the impurity with the conduction states $\hat{c}^\dagger_{\boldsymbol{k}\sigma}$, $\hat{c}^\dagger_{\boldsymbol{k}\sigma}$. The SIAM provides a microscopic description of Kondo physics in heavy-fermion compounds and Kondo insulators, and underlies many dynamical mean-field-theory (DMFT) calculations [58–60].

We shall not revisit the physics of the SIAM itself; instead, we focus on the computational bottleneck identified in Ref. [50] and show how it can be alleviated by our patched contraction scheme.

In the parquet formalism [61] the single–impurity problem is reformulated in terms of coupled two-particle vertex equations. In the particle–hole (ph)

channel the Bethe–Salpeter equations (BSEs) for the density (d) and magnetic (m) components read

$$F_d^{\nu\nu'\omega} = \Gamma_d^{\nu\nu'\omega} - \frac{1}{\beta^2}\sum_{\nu_1\nu_2} \Gamma_d^{\nu\nu_1\omega}\chi_{0,ph}^{\nu_1\nu_2\omega}F_d^{\nu_2\nu'\omega} \tag{5.22}$$

$$F_m^{\nu\nu'\omega} = \Gamma_m^{\nu\nu'\omega} - \frac{1}{\beta^2}\sum_{\nu_1\nu_2} \Gamma_m^{\nu\nu_1\omega}\chi_{0,ph}^{\nu_1\nu_2\omega}F_m^{\nu_2\nu'\omega}. \tag{5.23}$$

The right–hand side of each equation is a three-indexed three-tensor contraction that must be evaluated many times during the iterative Parquet loop, and thus dominates the overall cost. Rohshap, Ritter *et al.* [50] solved Eqs.(5.22)–(5.23) with great success using QTCI-based tensor trains. Here we revisit the same contraction but replace the monolithic MPO–MPO multiplication by the *patched* strategy developed in Chap. 4, expecting further savings from the control of local bond dimensions.

Fig. 5.19 sketches the workflow for converting a three–frequency vertex $V_{\nu\,\nu'\,\omega}$ (ν, ν' fermionic, ω bosonic) into the MPO format required by the MPO-MPO contraction algorithms.

(a) Each Matsubara variable is discretised on a binary grid of $\mathcal{R}$ bits. Applying QTCI yields a tensor–train[2] $\widetilde{V}_{\nu,\nu',\omega}$.

(b) The TT is reshaped into an MPO by (i) regrouping fermionic indices according to Eq. (4.2), and (ii) "diagonalising" bosonic tensor cores as in Eq. (4.17). This produces an efficient MPO representation whose structure matches the mixed matrix-multiplication and Hadamard operations in the Bethe-Salpeter equations.

(c) For a patched treatment the same sequence is applied *per patch*: the global QTCI compression is replaced by pQTCI, and the subsequent reshaping steps are local to each patch.

The initial stage of the patched BSE scheme is visualised in Fig. 5.20. Using the interleaved slicing order (see Fig. 5.9), each two-particle vertex is compressed with pQTCI so that the resulting patch grid adapts to the structure of the data. For clarity we display two-dimensional cuts at the bosonic frequency $\omega = 0$; the axes correspond to the fermionic frequencies on a $2^{\mathcal{R}} \times 2^{\mathcal{R}}$ mesh with $\mathcal{R} = 7$. The color scale shows $|F_d|$, $|\Gamma_d|$ and $|\chi_{0,ph}|$, respectively, each reconstructed from their patched approximation up to a tolerance $\tau = 10^{-7}$. One sees that smaller, tiles concentrate in the regions where the vertices exhibit pronounced structure, whereas featureless areas are covered by larger patches.

Fig. 5.21 compares the wall-clock time required to evaluate the contraction on the right-hand side of the BSEs with and without patching. Calculations were performed on an Intel® Xeon® E5-2680 v4 @ 2.40 GHz; the horizontal axis shows the number of bits $\mathcal{R}$ used per fermionic frequency (i.e. $\mathcal{R} = \log_2 N_\nu$ with N_ν total frequencies). Four target tolerances τ are reported. Over the entire

[2]For brevity we drop the subscript σ on all quantics bit strings; the frequency is now written simply as $\nu = (\nu_1, \ldots, \nu_{\mathcal{R}})$ which unambiguously denotes the $\mathcal{R}$-bit representation of the Matsubara index.

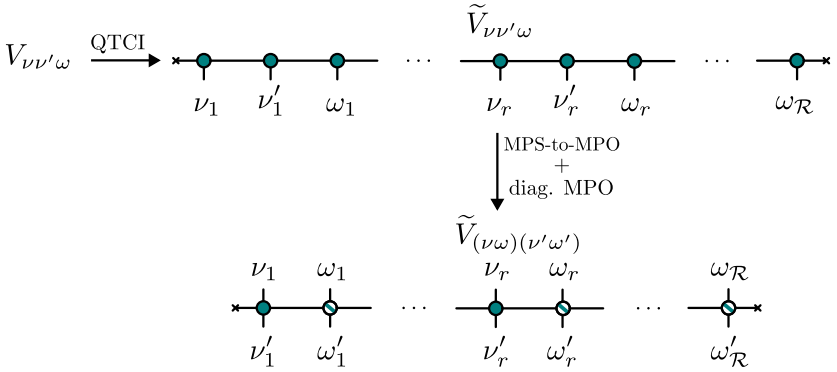


Figure 5.19: Transformation of a three-frequency vertex $V_{\nu\,\nu'\,\omega}$ into an MPO: QTCI compression on a $\mathcal{R}$-bit mesh per frequency; TT $\rightarrow$ MPO mapping via Eqs. (4.2) and (4.17); resulting contraction-ready MPO $\widetilde{V}_{(\nu\omega)(\nu'\omega')}$. For patched calculations the first step is replaced by pQTCI and the second step is performed on each patch.

range the patched algorithm outperforms the conventional single-MPO contraction by up to an order of magnitude. For the stringent tolerance $\tau = 10^{-10}$ the monolithic approach was no longer feasible beyond $\mathcal{R} = 7$ owing to excessive memory requirements, whereas the patched routine remained tractable.

By decomposing the three-index vertex product into many low-rank patch contractions, the patched MPO strategy removes the χ^4 bottleneck that plagues the standard approach. The resulting speed-up highlights the potential of patched tensor-network techniques for self-consistent parquet calculations at high frequency resolution.

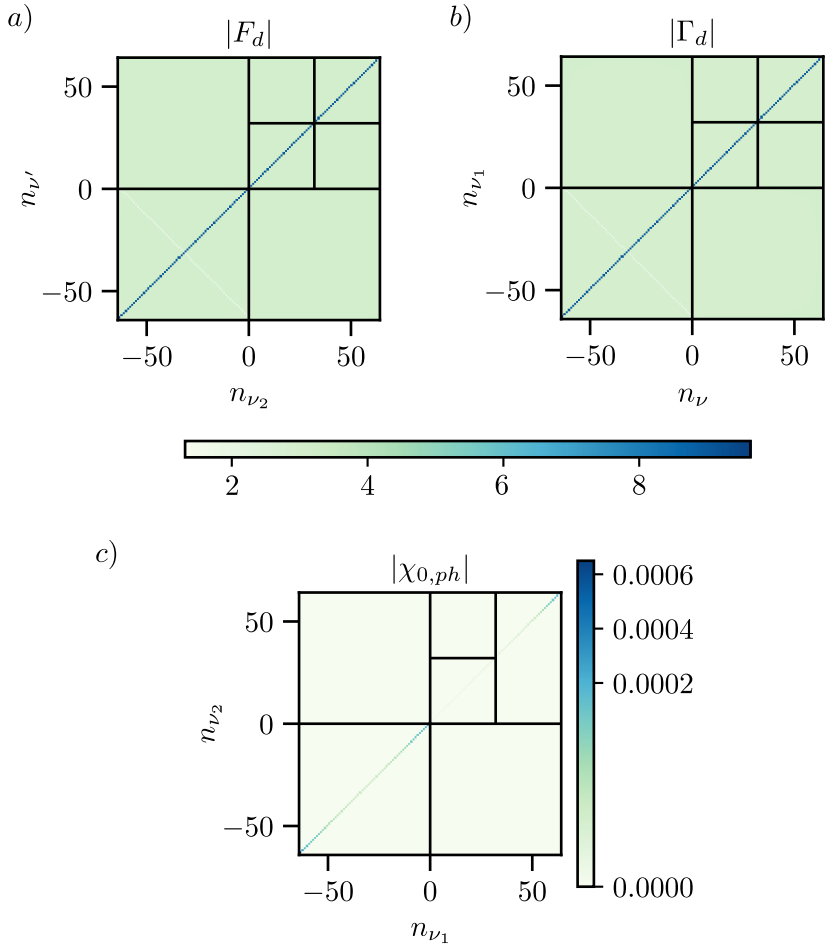


Figure 5.20: pQTCI compression of the Bethe-Salpeter vertices at $\omega = 0$. Panels show $|F_d|$ (a), $|\Gamma_d|$ (b) and $|\chi_{0,ph}|$ (c) on the respective (ν_2, ν'), (ν_1, ν) and (ν_2, ν_1) grids for $\mathcal{R} = 7$ and tolerance $\tau = 10^{-7}$. Patch boundaries reveal how the adaptive slicing refines only those regions where the vertex is more interesting, also for three dimensional objects.

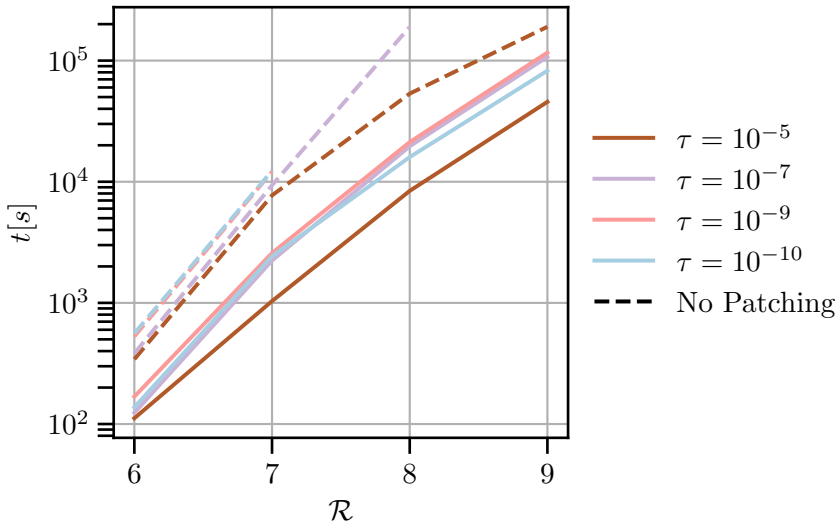


Figure 5.21: CPU time for the BSE vertex contraction versus the frequency resolution $\mathcal{R}$ (bits per Matsubara axis). Solid curves: patched MPO–MPO contraction; dashed curves: conventional (non-patched) contraction. Colours denote the compression tolerance τ.

Chapter 6

Summary and outlook

The work presented in this thesis expands the scope of tensor-train cross-approximation by introducing a patch-based, *divide-et-impera* strategy. Starting from the modern implementation of QTCI developed by Ritter *et al.*, namely `TensorCrossInterpolation.jl` [24] and its quantics extensions [25], we have integrated a new layer of logic that adaptively partitions the input tensor into smaller subtensors, each compressed with a user-defined bond-dimension cap χ_{patch}. This patched variant, *pQTCI*, retains the logarithmic complexity of the original algorithm [1] while addressing two long-standing bottlenecks: the explosive bond growth that plagues functions with narrow peaks, and the χ^4 scaling that renders large matrix-product-operator contractions prohibitive.

The numerical evidence collected throughout the manuscript makes the advantages of the patched approach clear. Whenever the tensor of interest develops strong "localisation", the rank of a single tensor-train representation becomes dictated by the most singular region. pQTCI circumvents this "one-size-fits-all" limitation by assigning high rank only to those patches that actually need it. In the two-dimensional Green's function benchmarks in Sec. 5.1, for instance, as soon as the broadening parameter δ falls below about 10^{-2}, the number of floating-point parameters and the wall-clock time required by the patched approximation decrease by an order of magnitude compared with the standard QTCI. A similar gain appears in the bare susceptibility $\chi_0(\boldsymbol{q}, i\omega)$ calculation in Sec. 5.3, where the computational cost remains well below the one of the monolithic strategy.

Moreover, the patch paradigm shows its full strength in tensor contractions. By expressing each factor in a product as a collection of low-rank patches, the contraction can be decomposed into many smaller and simpler tasks. For the element-wise product of two real-space Green's functions the patched routine delivers a speed-up of nearly ten on a single workstation; furthermore, it completes cases that a single contraction cannot even fit within the RAM memory availability.

A central practical question remains: how should one choose the cap χ_{patch}? Our analysis has derived two sets of bounds, Eqs. (3.29) and (4.23),(4.29)–(4.33), that delimit the patch count required for the patched approximation and patched MPO-MPO contraction, respectively, to beat its monolithic counterpart. These bounds are reassuring *a posteriori*: whenever the observed patch number respects them, the patched scheme is indeed advantageous. They

bounds do not, however, determine χ_{patch} *a priori*. The numerical examples in Chap. 5 suggest that the optimal cap is correlated with the "sharpness" (cf. Fig. 5.2) of the function of interest, yet the relation is intricate and problem specific. For instance, if the cap chosen is too small, the algorithm enters the "*over-patching*" regime, where an explosion of tiny patches negates the expected savings. Caps that are too large, on the other hand, converge to the same resources requirement of a single QTCI approximation (cf. Figs 5.5 and 5.5).

Establishing a predictive rule for the cap is therefore an important topic for future research. Such a rule is likely to involve a refinement of the concept of ε–factorisable functions: just as QTCI succeeds whenever the entire tensor admits a low-rank representation, pQTCI succeeds whenever the configuration space can be divided into a modest number of regions, each of which is well approximated at rank χ_{patch}. One may speak of ε–*patch*– *factorisable* functions and attempt to characterise their feature distributions analytically.

Looking at concrete applications, a natural next step starting from Sec. 5.3 is to transfer the patched-QTCI strategy from imaginary to real frequencies. Computing the retarded bare susceptibility $\chi_0^R(\boldsymbol{q}, \omega)$ is famously delicate. The local error control and adaptive rank allocation built into pQTCI are well suited to tame these difficulties and should enable high-resolution calculations directly on the real-frequency axis. A second avenue (cf. Sec. 5.4) is to apply the patch concept to vertex physics—specifically, to solve the Bethe-Salpeter equation with full momentum dependence. Because the kernel of a momentum-resolved BSE often contains sharply peaked structures that vary from one region of the Brillouin zone to another, distributing the calculation into rank-capped patches could reduce the overall contraction cost in much the same way it does for Green's-function products. Both extensions would further broaden the range of many-body problems that can be tackled efficiently within the QTCI framework.

Finally, the patching strategy is inherently parallel. At the time of writing, a parallel version of the state-of-the-art `crossinterpolate2` routine has already been released, and its capabilities dovetail naturally with the patched techniques introduced here. Because both pQTCI and the patched MPO-MPO algorithms break the workload into many independent, rank-capped subtasks, they lend themselves to *distributed* execution (a first parallel implementation of the patched contraction scheme is already in place in the tensor4all.org julia libraries collection [25]). Harnessing the new internal parallelism of `crossinterpolate2` together with this external domain decomposition promises substantial additional speed-ups once the tuning between the two parallelisation schemes is optimised.

Appendix A

Bounds on N_{patch}

A.1 2D Green Function Approximation

Figures 5.5 and 5.6 in Chapter 5 showed that a patched QTCI (pQTCI) run is advantageous only within a certain window of the patch bond–dimension cap χ_{patch}. To quantify that window we compare the measured patch count $\mathcal{N}_p$ with the theoretical bounds (cf. Eq. (3.29))

$$N_{\text{patch}} < \begin{cases} \chi^2/\chi^2_{\text{patch}}, & \text{(memory)}, \\ \chi^3/(d\,\chi^3_{\text{patch}}), & \text{(CPU runtime)}, \end{cases} \tag{A.1}$$

where χ is the rank of the corresponding single–TT (standard QTCI) approximation at the same discretisation parameters $(\mathcal{R}, \tau)$.

For the pQTCI approximated function $\operatorname{Re} G(\boldsymbol{k})$ [Eq. (5.1)] we plot $N_{\text{patch}} - \left(\chi^2/\chi^2_{\text{patch}}\right)$ and $N_{\text{patch}} - \left[\chi^3/(d\,\chi^3_{\text{patch}})\right]$ for selected (δ) values and index unfoldings. Negative values mean that the bound is not correctly satified.

Several trends emerge:

- For sharp spectral lines ($\delta = 10^{-3}$, fused ordering, Fig. A.1) the bounds are comfortably met in the optimum χ_{patch} range, explaining the clear savings seen in Fig. 5.5.
- When the Green's function is broad ($\delta = 10^{-1}$, interleaved ordering, Fig. A.2) the algorithm slices the domain more than necessary; the patch count exceeds the theoretical limits and the patched run is no longer profitable.
- Although Fig. 5.5 shows time savings only at specific χ_{patch}, the run–time bound in Eq. (A.1) is *always* satisfied. The apparent discrepancy could be due to the current pQTCI implementation, whose task–scheduling overhead masks the benefit except when the single–TT contraction becomes truly expensive. Moreover, the bounds are only an rough estimates. Many more variables play a role in the actual runtime of the implemented pQTCI algorithm

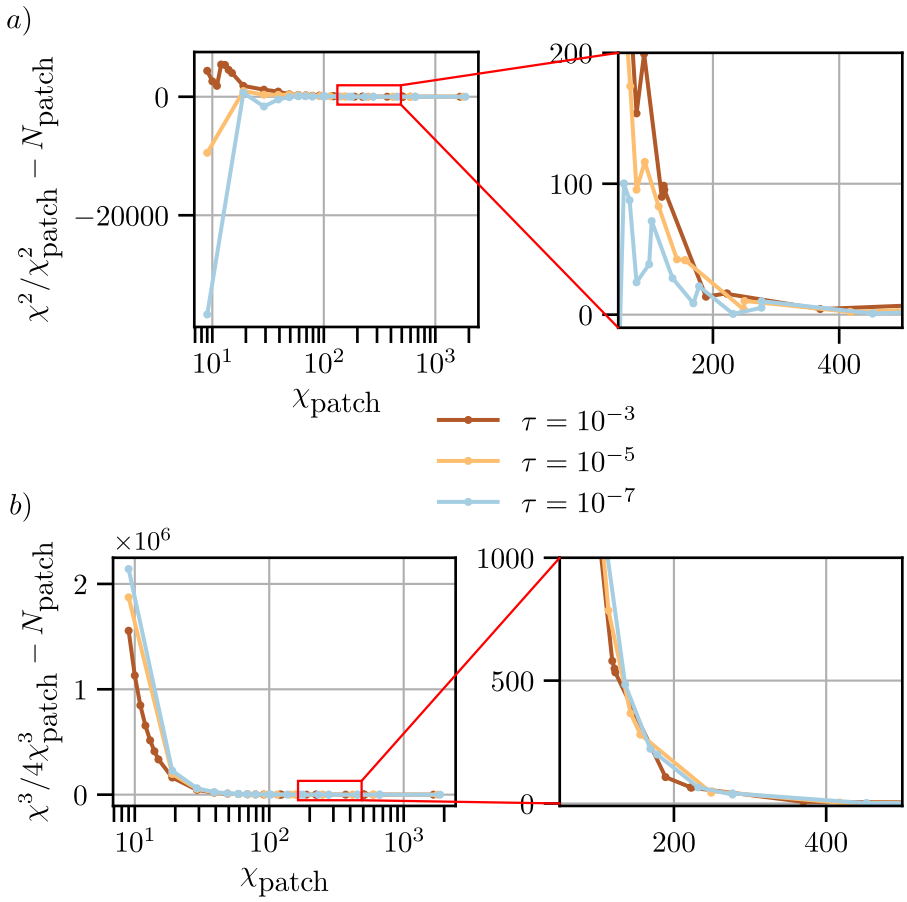


Figure A.1: **Fused ordering, $\delta = 10^{-3}$.** Difference between the actual patch count N_{patch} and the memory (a) and run–time (b) bounds of Eq. (A.1). Negative values indicate that the bound is not respected.

In summary, the bounds of Eq. (A.1) provide a somewhat reliable indicator of when pQTCI will outperform a monolithic QTCI run: the algorithm is advantageous around the parameter regions where both the memory and time inequalities are fulfilled.

A.2 Patched MPO-MPO contractions

Matrix muliplication

The limits derived in Eq. (4.29), Eq. (4.31), and Eq. (4.33) assume that the two MPO factors share the same patching depth $\bar{\ell}$. If the left MPO $\widetilde{\mathcal{F}}_{\boldsymbol{\sigma},\boldsymbol{\sigma}'}$ and the right MPO $\widetilde{\mathcal{G}}_{\boldsymbol{\sigma}',\boldsymbol{\sigma}''}$ are patched to *different* levels, the bounds depend onf both of their patch numbers. With $\chi_{\mathcal{F}}$ and $\chi_{\mathcal{G}}$ denoting the bond dimensions of the corresponding non-patched tensors, and $\chi_{\text{patch},\mathcal{F}}$, $\chi_{\text{patch},\mathcal{G}}$ the caps imposed on

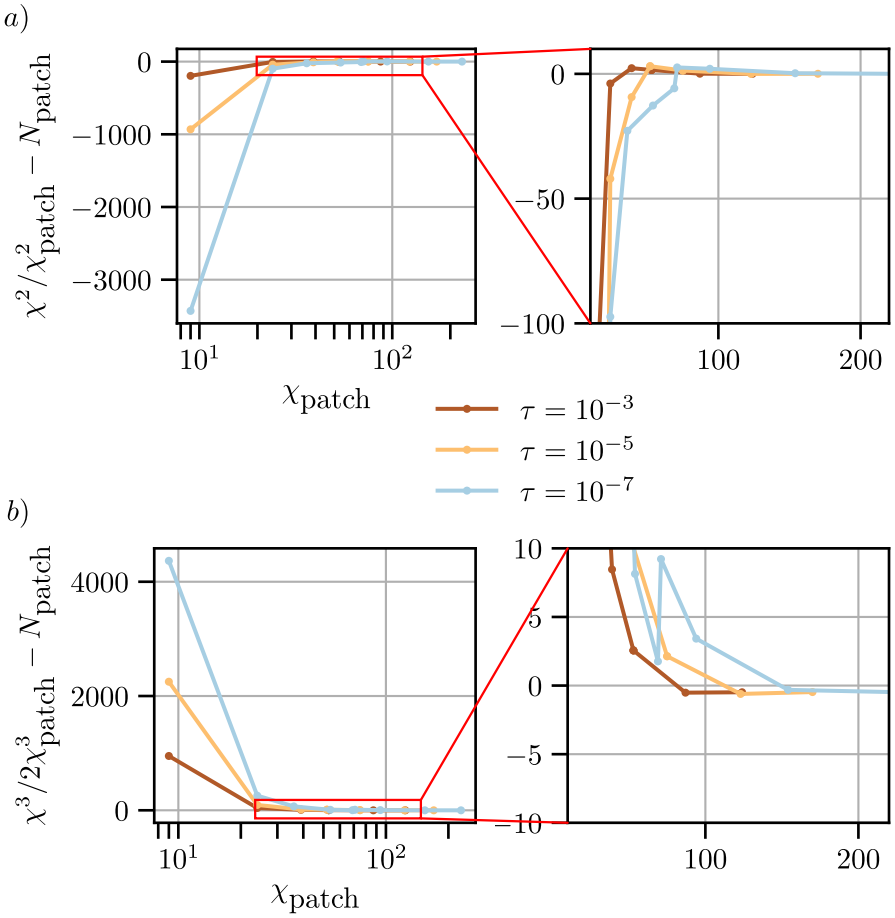


Figure A.2: **Interleaved ordering,** $\delta = 10^{-1}$**.** Same analysis as Fig. A.1. Here pQTCI tends to *violate* the bounds more often because $\mathrm{Re}\, G(\boldsymbol{k})$ is already smooth and the algorithm over–patches the domain.

each factor, the generalised conditions read

(a) **Worst case**

$$N_{\text{patch},\mathcal{F}} N_{\text{patch},\mathcal{G}} < \frac{\chi^2_{\mathcal{F}} \chi^2_{\mathcal{G}}}{\chi^2_{\text{patch},\mathcal{F}} \chi^2_{\text{patch},\mathcal{G}}} \tag{A.2}$$

(b) **Best case**

$$N^{\max}_{\text{patch}} < \frac{\chi^2_{\mathcal{F}} \chi^2_{\mathcal{G}}}{\chi^2_{\text{patch},\mathcal{F}} \chi^2_{\text{patch},\mathcal{G}}} \tag{A.3}$$

where $N^{\max}_{\text{patch}} = \max\{N_{\text{patch},\mathcal{F}}, N_{\text{patch},\mathcal{G}}\}$.

(c) **Average case** Let $\bar{\ell}_{\min}$ be the smaller and $\bar{\ell}_{\max}$ the larger patching level for the two factors. The number of admissible patch pairs is $d^{\bar{\ell}_{\min}} d^{\bar{\ell}_{\min}(D-1)/D} d^{\bar{\ell}_{\max}-\bar{\ell}_{\min}}$ (cf. Eq. (4.33)), leading to

$$N_{\text{patch},\mathcal{F}}^{\frac{D-1}{D}} N_{\text{patch},\mathcal{G}} < \frac{\chi_{\mathcal{F}}^2 \chi_{\mathcal{G}}^2}{\chi_{\text{patch},\mathcal{F}}^2 \chi_{\text{patch},\mathcal{G}}^2}. \tag{A.4}$$

This bounds can be tested for the results we showed in Fig. 5.10. For each data point of the *worst-case*, *best-case* and *average-case* scenario we subtract the r.h.s. and the l.h.s. of the inequalities in Eq. (A.2), Eq. (A.3) and Eq. (A.4), respectively, and plot the resulting "bound difference" in Fig. A.3. We observe an approximate correspondence between the "overpatched" runs in Fig. 5.10 and negative "bound difference". In particular, no data point in the *worst-case-scenario* satisfies the bound, while most of the *best-case-scenario* runs do.

Irrespective of the simulation parameters $(\tau, \mathcal{R})$, the total number of floating-point entries in the result tensor, as measure of the contraction complexity, obeys

(a) **Worst case**

$$\mathcal{O}\big(N_{\text{patch},\mathcal{F}} N_{\text{patch},\mathcal{G}} \chi_{\text{patch},\mathcal{F}}^2 \chi_{\text{patch},\mathcal{G}}^2\big). \tag{A.5}$$

(b) **Best case**

$$\mathcal{O}\big(N_{\text{patch,max}} N_{\text{patch},\mathcal{G}} \chi_{\text{patch},\mathcal{F}}^2 \chi_{\text{patch},\mathcal{G}}^2\big). \tag{A.6}$$

(c) **Average case**

$$\mathcal{O}\big(N_{\text{patch},\mathcal{F}}^{\frac{D}{D-1}} N_{\text{patch},\mathcal{G}} \chi_{\text{patch},\mathcal{F}}^2 \chi_{\text{patch},\mathcal{G}}^2\big). \tag{A.7}$$

Fig. A.4 confirms these scaling laws: the measured parameter counts collapse onto the predicted combinations of N_{patch} and χ_{patch} for the two factors $\mathcal{F}$ and $\mathcal{G}$.

Element-wise multiplication

For an element–wise product of two tensors $\mathcal{F}_{\boldsymbol{\sigma}}$ and $\mathcal{G}_{\boldsymbol{\sigma}}$ the patch–count limit of Eq. (4.23) generalises to

$$N_{\text{patch}}^{\max} < \frac{\chi_{\mathcal{F}}^2 \chi_{\mathcal{G}}^2}{\chi_{\text{patch},\mathcal{F}}^2 \chi_{\text{patch},\mathcal{G}}^2} \tag{A.8}$$

where $N_{\text{patch}}^{\max} = \max\{N_{\text{patch},\mathcal{F}}, N_{\text{patch},\mathcal{G}}\}$. Fig. A.5 plots the difference between the left– and right–hand sides for all data points of Fig. 5.13. Negative bars mark those instances where the patched run exceeds the bound.

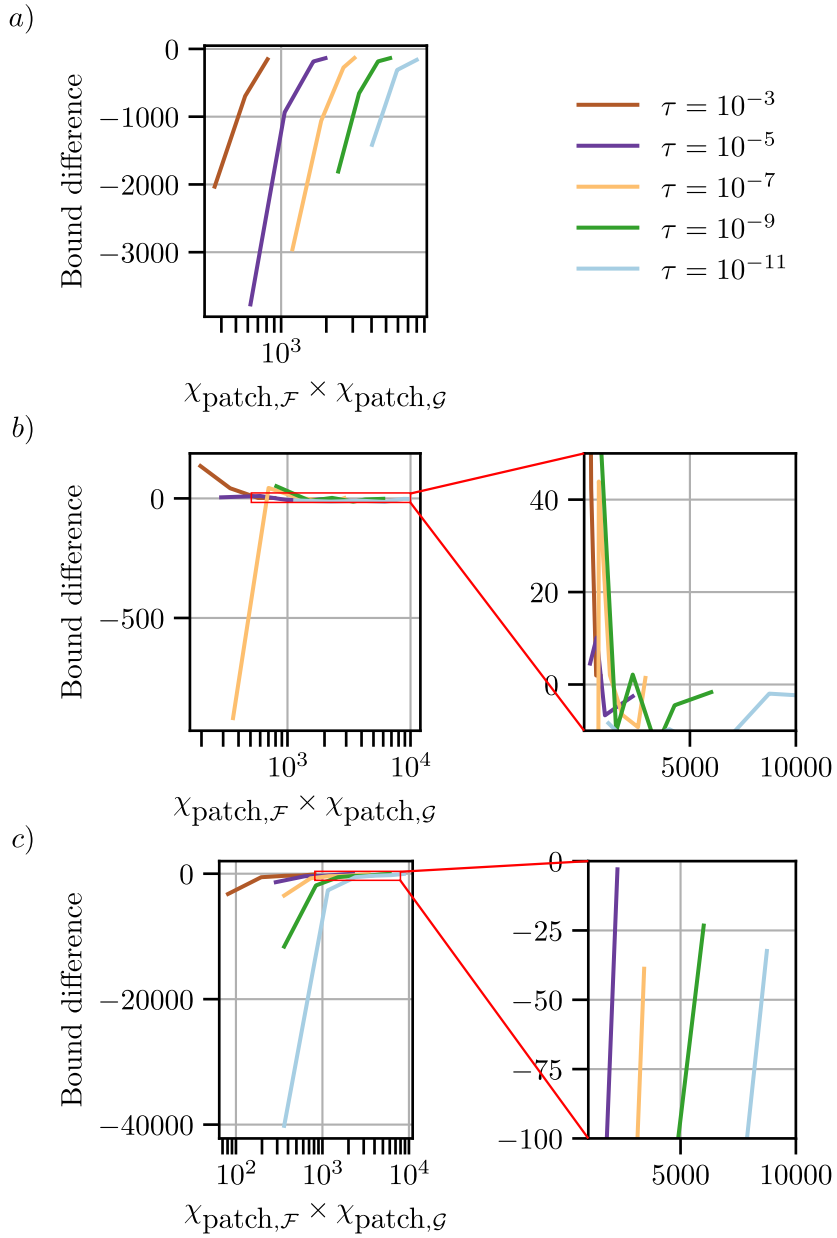


Figure A.3: Deviation of the measured number of patch products from the theoretical limits of Eqs. (A.2), (A.3), and (A.4). Negative values indicate that the bound is *not* satisfied. We illustrate the *worst* (*a*), *best* (*b*) and *average* (*c*) case patche MPO-MPO contraction bounds.

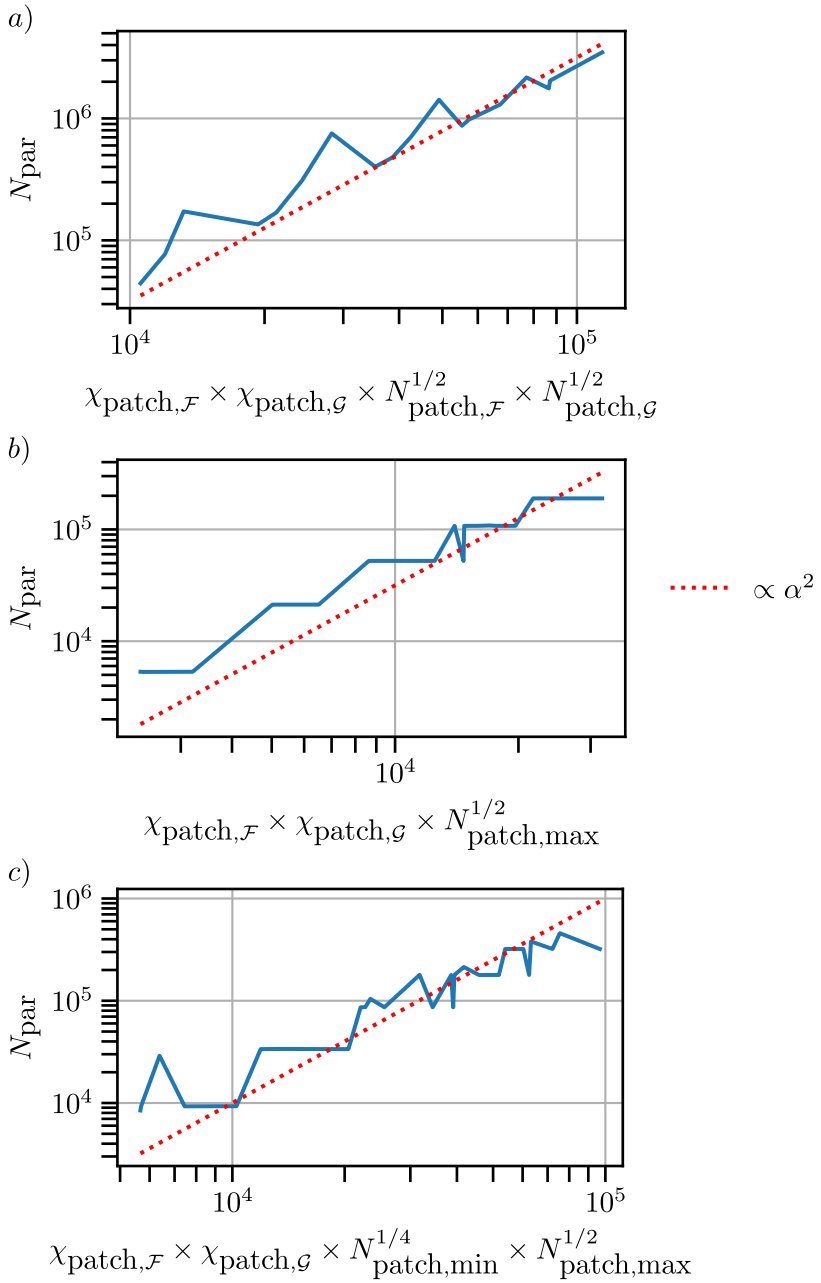


Figure A.4: Total number of parameters in set of patches representing $\mathcal{H}$ versus the scaling variables from Eqs. (A.5)– (A.7). Solid lines illustrate data; dotted lines are the expected theoretical trends.

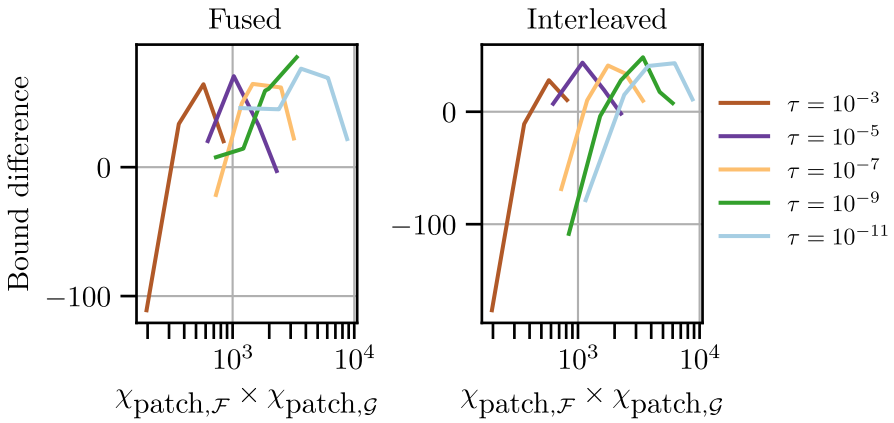


Figure A.5: Deviation from the bound in Eq. (A.8) for element-wise multiplication of $\mathcal{F}_{\boldsymbol{\sigma}}$ and $\mathcal{G}_{\boldsymbol{\sigma}}$. Separate series are shown for fused and interleaved index orderings. Negative values indicate a violation of the bound.

As measure of contraction complexity, the total number of floating-point parameters in the patched product should scale as

$$\mathcal{O}\big(N_{\text{patch,max}}\, N_{\text{patch,min}}\, \chi^2_{\text{patch},\mathcal{F}}\, \chi^2_{\text{patch},\mathcal{G}}\big), \tag{A.9}$$

with $N_{\text{patch,min}} = \min\{N_{\text{patch},\mathcal{F}}, N_{\text{patch},\mathcal{G}}\}$. Fig. A.6 compares the measured parameter counts with this prediction; the dotted lines trace the theoretical trend and closely follow the numerical data.

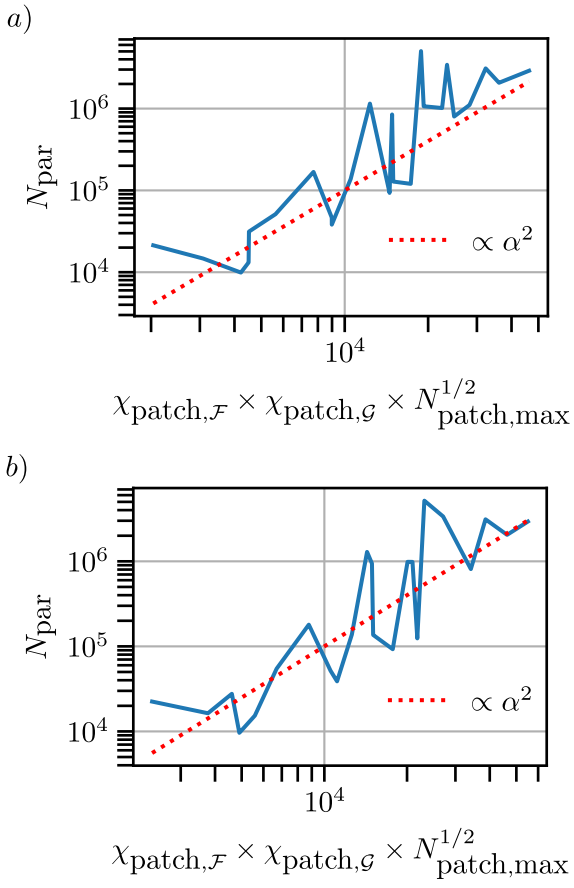


Figure A.6: Total parameter count of the patched tensor $\mathcal{FG}$ versus the scaling variable from Eq. (A.9). Solid lines: data (fused and interleaved orderings); dotted lines: expected scaling.

Appendix B

Quantics Fourier Transform

The quantics representation of tensors [39, 40] makes a Fourier transform almost trivial at the tensor-network level. For a scalar function $G(\boldsymbol{r})$ represented as a Quantics TT, the Fourier transfor

$$\hat{G}(\boldsymbol{k}) = \int d\boldsymbol{r}\; G(\boldsymbol{r}) e^{i\boldsymbol{k}\cdot\boldsymbol{r}} \tag{B.1}$$

results in a very simple MPO-MPO contraction that resembles the operation performed in some quantum computing routines [62] (e.g. Quantum Phase Estimation).

Let us start from the definition of Discrete Fourier Transform (DFT). Consider the variable $G_m = G(x(m))$, discretisation of the one-dimensional function $G(x)$ on a grid with $m = 0, \ldots, M-1$. The discrete Fourier transform (DFT) of G_m reads

$$\hat{G}_k = \sum_{m=0}^{M-1} T_{km} G_m, \quad T_{km} = \frac{1}{\sqrt{M}} e^{-i2\pi k\cdot m/M} \tag{B.2}$$

Represent m and k in binary with $\mathcal{R} = \log_2 M$ bits,

$$m(\boldsymbol{\sigma}) = \sum_{r=1}^{\mathcal{R}} \sigma_r 2^{\mathcal{R}-r}, \quad k(\boldsymbol{\sigma}') = \sum_{r'=1}^{\mathcal{R}} \sigma_{r'} 2^{\mathcal{R}-r'} \tag{B.3}$$

so that $M = 2^{\mathcal{R}}$. Then

$$T_{\boldsymbol{\sigma}'\boldsymbol{\sigma}} = T_{k(\boldsymbol{\sigma}')x(\boldsymbol{\sigma})} = \frac{1}{2^{\mathcal{R}/2}} \exp\left[-i2\pi \sum_{rr'} 2^{\mathcal{R}-r'-r} \sigma'_{r'}\sigma_r\right] \tag{B.4}$$

Re-ordering the bits in "scale–reversed" [1, 27] fashion—fusing $\sigma'_{\mathcal{R}-r+1}$, which encodes the scale 2^{r-1} in k, with σ_r, which encodes the scale $2^{\mathcal{R}-r}$ in x—one can cast T as the remarkably low-rank MPO

$$\widetilde{T}_{\boldsymbol{\sigma}'\boldsymbol{\sigma}} = \quad \sigma_1 \quad \sigma_2 \qquad \cdots \qquad \sigma_r \qquad \cdots \qquad \sigma_{\mathcal{R}} \quad (\text{B.5})$$
$$\qquad\qquad \sigma'_{\mathcal{R}} \quad \sigma'_{\mathcal{R}-1} \qquad\qquad \sigma'_{\mathcal{R}-r+1} \qquad\qquad \sigma'_1$$

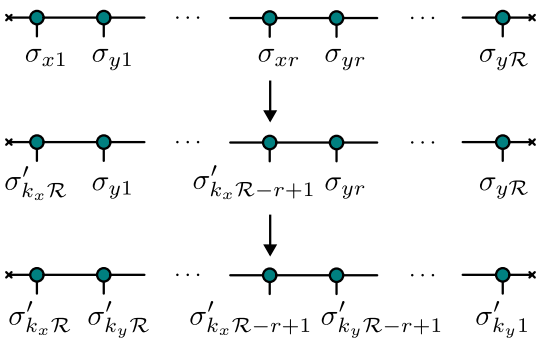
whose maximum bond dimension is just $\chi' = 11$ for machine-precision accuracy [63], $\left|T_{\boldsymbol{\sigma}'\boldsymbol{\sigma}} - \widetilde{T}_{\boldsymbol{\sigma}'\boldsymbol{\sigma}}\right| / \left|\widetilde{T}_{\boldsymbol{\sigma}'\boldsymbol{\sigma}}\right| \sim \epsilon_{\text{mach}}$..

Given a TT of rank χ, its quantics Fourier transform costs only

$$\mathcal{O}\big(\chi^2\,\chi'^2\,\mathcal{R}\big) = \mathcal{O}\big(\chi^2\,\chi'^2\,\log M\big), \tag{B.6}$$

which is exponentially faster than the conventional FFT scaling $\mathcal{O}\big(M \log M\big)$ [1].

For a generic d-variate function $G(\boldsymbol{r})$ the quantics Fourier transform is implemented by applying one 1D QFT per physical dimension. Graphically the operation can be written as



(B.7)

where, in the first step, we act only on the x-bits with the MPO

$$\widetilde{T}^{x}_{\boldsymbol{\sigma}'\boldsymbol{\sigma}} = \quad \sigma_{x1} \quad \sigma_{y1} \qquad \cdots \qquad \sigma_{xr} \quad \sigma_{yr} \qquad \cdots \qquad \sigma_{y\mathcal{R}} \quad (\text{B.8})$$
$$\qquad\qquad \sigma'_{k_x\mathcal{R}} \quad \sigma'_{y1} \qquad\qquad \sigma'_{k_x\mathcal{R}-r+1} \quad \sigma'_{yr} \qquad\qquad \sigma'_{y\mathcal{R}}$$

whose non-transforming site tensor factorises as

$$i \;\; \sigma_{yr} \;\; j \;\; \sigma'_{yr} \;\; = \begin{cases} [\mathbb{1}]_{i,j} & \text{if } \sigma'_{yr} = \sigma_{yr} \\ [0]_{i,j} & \text{otherwise.} \end{cases} \tag{B.9}$$

The construction repeats for the y- (and z-, ...) registers until the full d-dimensional QFT is obtained.

Rearrangement of quantics meshes

In the susceptibility calculation of Sec. 5.3 we are interested in having the inverse temporal Fourier transform at a shifted Matsubara grid whose centre corresponds to the smallest frequency, namely $\nu_n = \pi\frac{2n+\xi}{\beta}$ with $n = -2^{\mathcal{R}-1}, \dots, 2^{\mathcal{R}-1}-1$. Starting from

$$\begin{aligned} G(i\nu_{n'}) &= \beta \int_0^\beta d\tau e^{i\nu_{n'}\tau} G(\tau) = \frac{\beta}{2^{\mathcal{R}/2}} \sum_{m=0}^{2^{\mathcal{R}}-1} e^{i\nu_{n'}\frac{m}{2^{\mathcal{R}}}} G_m \\ &= \frac{\beta}{2^{\mathcal{R}/2}} \sum_{m=0}^{2^{\mathcal{R}}-1} e^{i\pi\frac{(2n'+\xi)}{\beta}m} G_m \end{aligned} \tag{B.10}$$

and redefining the index $n = n' - 2^{\mathcal{R}-1} \in [-2^{\mathcal{R}-1}, 2^{\mathcal{R}-1}-1]$ we obtain

$$\begin{aligned} G(i\nu_n) = G(i\nu_{n'-2^{\mathcal{R}-1}}) &= \frac{\beta}{2^{\mathcal{R}/2}} \sum_{m=0}^{2^{\mathcal{R}}-1} e^{i2\pi\frac{n'm}{2^{\mathcal{R}}}} e^{i\pi m\frac{\xi-2^{\mathcal{R}}}{2^{\mathcal{R}}}} G_m \\ &= \beta \sum_{m=0}^{2^{\mathcal{R}}-1} T^{(+)}_{n'm} e^{i\pi m\frac{\xi-2^{\mathcal{R}}}{2^{\mathcal{R}}}} G_m \end{aligned} \tag{B.11}$$

with the "positive" DFT kernel $T^{(+)}_{n'm} = (2^{\mathcal{R}-1})^{-1/2} e^{+i2\pi n'm/2^{\mathcal{R}}}$ as defined in Eq. (B.2). Equation (B.11) shows that the centred Matsubara transform is realised by a standard inverse QFT followed by a diagonal *phase-rotation* layer $\mathcal{P}$ [27]:

$$\beta\mathrm{QFT}^{-1}\mathcal{P}, \qquad \mathcal{P} = \mathrm{U1}(2^{\mathcal{R}-1}\theta) \cdot \mathrm{U1}(2^{\mathcal{R}-2}\theta) \cdots \mathrm{U1}(\theta) \tag{B.12}$$

where

$$\theta = \pi\frac{\xi - 2^{\mathcal{R}}}{2^{\mathcal{R}}}, \qquad \mathrm{U1}(\alpha) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{pmatrix}. \tag{B.13}$$

The layer $\mathcal{P}$ is an MPO of rank 1, so the extra cost is negligible compared with the inverse QFT itself. An analogous modification is applied to the spatial QFT when, for instance, we would like to center the symmetric Brillouin zone $[-\pi, \pi]^2$ on the quantics spatial grid.

These conventions are all employed for the susceptibility calculations reported in Sec. 5.3.

Bibliography

[1] Y. Núñez 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", SciPost Phys. **18**, 104 (2025).

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

[3] S. R. White, "Density matrix formulation for quantum renormalization groups", Phys. Rev. Lett. **69**, 2863 (1992).

[4] G. Vidal, "Efficient classical simulation of slightly entangled quantum computations", Phys. Rev. Lett. **91**, 147902 (2003).

[5] U. Schollwöck, "The density-matrix renormalization group in the age of matrix product states", Annals of Physics **326**, January 2011 Special Issue, 96 (2011).

[6] J. von Delft, *Lecture Notes on Tensor Networks for Many-Body Physics*, Ludwig-Maximilians University of Munich, Theoretical Solid State Physics, Lecture Notes (2023).

[7] M. Stoudemire, *tensornetwork.org*.

[8] F. Verstraete and J. I. Cirac, "Renormalization algorithms for quantum-many body systems in two and higher dimensions", arXiv:0407066 [cond-mat] (2004).

[9] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications", SIAM Review **51**, 455 (2009).

[10] I. V. Oseledets and E. E. Tyrtyshnikov, "Breaking the curse of dimensionality, or how to use svd in many dimensions", SIAM Journal on Scientific Computing **31**, 3744 (2009).

[11] I. V. Oseledets, "Tensor-train decomposition", SIAM Journal on Scientific Computing **33**, 2295 (2011).

[12] M. Fishman, S. R. White, and E. M. Stoudenmire, "The ITensor Software Library for Tensor Network Calculations", SciPost Phys. Codebases **4** (2022).

[13] A. Weichselbaum, "QSpace - An open-source tensor library for Abelian and non-Abelian symmetries", SciPost Phys. Codebases **40** (2024).

[14] I. V. Oseledets, *ttpy: Python implementation of the TT-toolbox*, `ttpy`, (2012).

[15] F. Verstraete, V. Murg, and J. C. and, "Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems", Advances in Physics **57**, 143 (2008).

[16] E. Isaacson and H. Keller, *Analysis of numerical methods*, Dover Books on Mathematics (Dover Publications, 1994), pp. 176–361.

[17] E. Y. Loh, J. E. Gubernatis, R. T. Scalettar, S. R. White, D. J. Scalapino, and R. L. Sugar, "Sign problem in the numerical simulation of many-electron systems", Phys. Rev. B **41**, 9301 (1990).

[18] Y. Núñez Fernández, M. Jeannin, P. T. Dumitrescu, T. Kloss, J. Kaye, O. Parcollet, and X. Waintal, "Learning Feynman Diagrams with Tensor Trains", Phys. Rev. X **12**, 041018 (2022).

[19] B. Settles, *Active learning*, Synthesis lectures on artificial intelligence and machine learning (Morgan & Claypool, 2012).

[20] K. Sozykin, A. Chertkov, R. Schutski, A.-H. Phan, A. Cichocki, and I. Oseledets, "TTOpt: a maximum volume quantized Tensor Train-based optimization and its application to Reinforcement Learning", arXiv:2205.00293 [cs-LG] (2022) .

[21] 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).

[22] N. Jolly, Y. N. Fernández, and X. Waintal, "Tensorized orbitals for computational chemistry", arXiv:2308.03508 [cond-mat] (2024).

[23] R. Sakurai, H. Takahashi, and K. Miyamoto, "Learning parameter dependence for fourier-based option pricing with tensor trains", arXiv:2405.00701 [q-fin.CP] (2025).

[24] R. Marc, S. Hiroshi, and S. Terasaki, *Tensorcrossinterpolation.jl*, version v0.9.0, https://github.com/tensor4all/TCI.jl.

[25] Y. Núñez 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, *tensor4all.org*.

[26] V. Ehrlacher, L. Grigori, D. Lombardi, and H. Song, "Adaptive hierarchical subtensor partitioning for tensor compression", SIAM Journal on Scientific Computing **43**, A139 (2021).

[27] 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).

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

[29] D. Savostyanov and I. Oseledets, "Fast adaptive interpolation of multi-dimensional arrays in tensor train format", International Workshop on Multidimensional (nD) Systems (2011).

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

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

[32] N. K. Kumar and J. Shneider, "Literature survey on low rank approximation of matrices", Linear and Multilinear Algebra **65**, 2212 (2016).

[33] G. H. Golub and C. F. van Loan, *Matrix computations*, Fourth (JHU Press, 2013).

[34] S. A. Goreinov, N. L. Zamarashkin, and E. E. Tyrtyshnikov, "Pseudo-skeleton approximations by matrices of maximal volume", Mathematical Notes **62**, 515 (1997).

[35] J. Schneider, "Error estimates for two-dimensional cross approximation", Journal of Approximation Theory **162**, 1685 (2010).

[36] C.-T. Pan, "On the existence and computation of rank-revealing lu factorizations", Linear Algebra and its Applications **316**, Special Issue: Conference celebrating the 60th birthday of Robert J. Plemmons, 199 (2000).

[37] G. Poole and L. Neal, "The rook's pivoting strategy", Journal of Computational and Applied Mathematics **123**, Numerical Analysis 2000. Vol. III: Linear Algebra, 353 (2000).

[38] I. Loshchilov and F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts", arXiv:1608.03983 [cs.LG] (2017).

[39] I. V. Oseledets, "Approximation of matrices with logarithmic number of parameters", Doklady Mathematics **80**, 653 (2009).

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

[41] M. Murray, H. Shinaoka, and P. Werner, "Nonequilibrium diagrammatic many-body simulations with quantics tensor trains", Phys. Rev. B **109**, 165135 (2024).

[42] H. Takahashi, R. Sakurai, and H. Shinaoka, "Compactness of quantics tensor train representations of local imaginary-time propagators", SciPost Physics **18** (2025).

[43] M. Lindsey, "Multiscale interpolative construction of quantized tensor trains", arXiv:2311.12445 [math.Na] (2004).

[44] N. Lee and A. Cichocki, "Fundamental tensor operations for large-scale data analysis using tensor network formats", Multidimensional Syst. Signal Process. **29**, 921 (2018).

[45] V. Ehrlacher, M. F. Ruiz, and D. Lombardi, "SOTT: Greedy Approximation of a Tensor as a Sum of Tensor Trains", SIAM Journal on Scientific Computing **44**, A664 (2022).

[46] M. Fuente Ruiz, "Adaptive tensor methods for high dimensional problems", Thesis (Sorbonne Université, Mar. 2023).

[47] P. Deng, W. Wu, and E. Shragowitz, "Adaptive partitions", *Encyclopedia of algorithms*, edited by M.-Y. Kao (Springer US, Boston, MA, 2008), pp. 4–7.

[48] S. Paeckel, T. Köhler, A. Swoboda, S. R. Manmana, U. Schollwöck, and C. Hubig, "Time-evolution methods for matrix-product states", Annals of Physics **411**, 167998 (2019).

[49] 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).

[50] S. Rohshap, M. K. Ritter, H. Shinaoka, J. von Delft, M. Wallerberger, and A. Kauch, "Two-particle calculations with quantics tensor trains: solving the parquet equations", Phys. Rev. Res. **7**, 023087 (2025).

[51] E. M. Stoudenmire and S. R. White, "Minimally entangled typical thermal state algorithms", New Journal of Physics **12**, 055026 (2010).

[52] C. Hubig, I. P. McCulloch, and U. Schollwöck, "Generic construction of efficient matrix product operators", Phys. Rev. B **95**, 035129 (2017).

[53] I. P. McCulloch, "Infinite size density matrix renormalization group, revisited", arXiv:0804.2509 [cond-mat.str-el] (2008).

[54] G. D. Mahan, *Many particle physics, third edition* (Plenum, New York, 2000), pp. 109–183.

[55] N. E. Bickers, D. J. Scalapino, and S. R. White, "Conserving approximations for strongly correlated electron systems: bethe-salpeter equation and dynamics for the two-dimensional hubbard model", Phys. Rev. Lett. **62**, 961 (1989).

[56] M. V. Rakhuba and I. V. Oseledets, "Fast multidimensional convolution in low-rank tensor formats via cross approximation", SIAM Journal on Scientific Computing **37**, A565 (2015).

[57] P. Thunström, O. Gunnarsson, S. Ciuchi, and G. Rohringer, "Analytical investigation of singularities in two-particle irreducible vertex functions of the hubbard atom", Phys. Rev. B **98**, 235107 (2018).

[58] A. C. Hewson, *The kondo problem to heavy fermions*, Cambridge Studies in Magnetism (Cambridge University Press, 1993).

[59] J. R. Schrieffer and P. A. Wolff, "Relation between the anderson and kondo hamiltonians", Phys. Rev. **149**, 491 (1966).

[60] A. Georges, G. Kotliar, W. Krauth, and M. J. Rozenberg, "Dynamical mean-field theory of strongly correlated fermion systems and the limit of infinite dimensions", Rev. Mod. Phys. **68**, 13 (1996).

[61] G. Rohringer, A. Valli, and A. Toschi, "Local electronic correlation at the two-particle level", Phys. Rev. B **86**, 125114 (2012).

[62] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information: 10th anniversary edition* (Cambridge University Press, 2010), pp. 216–242.

[63] J. Chen, E. Stoudenmire, and S. R. White, "Quantum fourier transform has small entanglement", PRX Quantum **4**, 040318 (2023).

This is a blank page...