A streamlined quantum algorithm for topological data analysis with exponentially fewer qubits

Sam McArdle,[1,2] András Gilyén,[3] and Mario Berta[1,2,4]

[1] *AWS Center for Quantum Computing, Pasadena, CA 91125, USA*
[2] *Institute for Quantum Information and Matter, California Institute of Technology, Pasadena, USA*
[3] *Alfréd Rényi Institute of Mathematics, Budapest, Hungary*
[4] *Department of Computing, Imperial College London, London, UK*

(Dated: January 30, 2023)

Topological invariants of a dataset, such as the number of holes that survive from one length scale to another (persistent Betti numbers) can be used to analyse and classify data in machine learning applications. We present an improved quantum algorithm for computing persistent Betti numbers, and provide an end-to-end complexity analysis. Our approach provides large polynomial time improvements, and an exponential space saving, over existing quantum algorithms. Subject to gap dependencies, our algorithm obtains an almost quintic speedup in the number of datapoints over rigorous state-of-the-art classical algorithms for calculating the persistent Betti numbers to constant additive error – the salient task for applications. However, this may be reduced to closer to quadratic when compared against heuristic classical methods and observed scalings. We discuss whether quantum algorithms can achieve an exponential speedup for tasks of practical interest, as claimed previously. We conclude that there is currently no evidence that this is the case.

I. Introduction

Modern datasets are growing increasingly large, high-dimensional, and noisy. Topological data analysis (TDA) has been touted as a way to find needles of interpretation in this data haystack [1]. TDA extends methods from algebraic topology (using group theory to classify topological objects) to datapoints sampled from an underlying topological manifold. Topological features (such as the number of connected components, or holes of a given dimension) can be good identifiers for a dataset, as they are often robust to noise and perturbations accrued when collecting the data, unlike geometric features. Topological features may be used directly to interpret the dataset – for example, to find holes in the coverage of sensors in a network [2], identify isolated areas in voter preference data [3], or classify structure in the distribution of matter in the universe [4]. They can also be used as general identifiers to compare different datasets in situations where the topology does not necessarily have an obvious interpretation – such as identifying participant groups in functional magnetic resonance imaging brain scans [5], or detecting patterns in time series data [6] to predict financial crashes [7]. Using topological features as input to machine learning models is growing increasingly popular within physics [8] (such as the unsupervised detection of phase transitions in lattice models [9, 10]) and other areas of science [11].

Classical algorithms for TDA proceed through linear algebra. An approximation of the manifold is constructed from a set of simplices (vertices, line segments, triangles, and higher dimensional generalizations) built by connecting together points within a certain length scale of each other. We are interested in calculating the persistent Betti number $\beta_k^{i,j}$, the number of k-dimensional holes that survive from scale i to scale j, for a range of dimensions and length scales. The number of k-dimensional holes at a single scale is known as the Betti number β_k^i, and is a less informative quantity [12]. A dataset consisting of N points can contain $\binom{N}{k+1}$ k-dimensional simplices, which we represent by orthonormal basis vectors. Classical algorithms for $\beta_k^{i,j}$ (and β_k^i) scale polynomially in time and space with the size of the simplicial complex, which can make them too costly to apply to systems with many datapoints, or for high k. However, large k values are likely also less prevalent in applications due to their reduced interpretability [8].

A quantum algorithm for computing Betti numbers has been proposed [13], demonstrated as a proof-of-principle [24], refined [15, 16, 25], and recently extended to the more informative persistent Betti numbers [17] – with claims of an exponential speedup over classical algorithms. The quantum algorithm exploits the ability of N qubits to represent and perform linear algebra in a vector space of dimension 2^N, in some cases with cost polynomial in N. However, the quantum algorithm actually returns the (persistent) Betti number, normalised by the number of k-simplices in the complex. While tasks closely related to normalised Betti number estimation have been shown to be DQC1-hard [25, 26] (note that the task of estimating normalised (persistent) Betti numbers of typically considered clique complexes has not been shown to be DQC1-hard [26]), there is currently no evidence for a regime with practical use cases. When using quantum algorithms to compute the salient quantity $\beta_k^{i,j}$, the exponential advantage is lost (a similar issue afflicts quantum algorithms for approximating the Jones polynomial [27, 28] and other #P problems [29]). Therefore, quantum algorithms for TDA should be optimised, if they are to achieve significant polynomial advantage over classical approaches for applications of interest.

	Algorithm	β_k^i	$\beta_k^{i,j}$	Operator encoding topology	Time complexity	Spatial complexity
Quantum	LGZ [13]	✓	✗[a]	Hermitian embedding of boundary operators ∂_k^i, ∂_{k+1}^i	$\mathcal{O}\left(\frac{N^3\binom{N}{k+1}}{\Delta^2 \mathrm{Min}\left(\Lambda_{\partial_k^i}, \Lambda_{\partial_{k+1}^i}\right)}\right)$	$\mathcal{O}(N^2)$
	GK [15]	✓	✗	$\bigoplus_k \partial_k + \partial_k^\dagger$	$\tilde{\mathcal{O}}\left(\frac{N^2 k\sqrt{\binom{N}{k+1}}}{\Delta\Lambda}\right)$	$\mathcal{O}(N)$
	UAS+ [16]	✓	✗	Combinatorial Laplacian	$\mathcal{O}\left(\frac{N\binom{N}{k+1}^{1.5}}{\Delta^3\Lambda}\right)$[b]	$\mathcal{O}(N)$
	Hay [17]	✓	✓	Persistent combinatorial Laplacian	$\tilde{\mathcal{O}}\left(\frac{N^8 k^4\binom{N}{k+1}}{\Delta^2\Lambda_1^2\Lambda_2}\right)$	$\mathcal{O}(N)$
	AMS [18]	✓	✓[c]	Hermitian embedding of boundary operators	Unspecified	$\mathcal{O}(N^2)$
	This work	✓	✓	$\partial_k^i, \partial_{k+1}^j$	$\tilde{\mathcal{O}}\left(\frac{N^{3/2}\sqrt{\binom{N}{k+1}}}{\Delta\Lambda_{\Pi\Pi}\mathrm{Min}\left(\Lambda_{\partial_k^i}, \Lambda_{\partial_{k+1}^j}\right)}\right)$	$\mathcal{O}(k\log(N))$
Classical	Textbook [19]	✓	✓	Boundary operator	$\mathcal{O}\big(\lvert S_{k+1}^j\rvert^3\big)$	$\mathcal{O}\big(\lvert S_{k+1}^j\rvert^2\big)$
	Optimised textbook [20, 21]	✓	✓	Boundary operator	$\mathcal{O}\big(\lvert S_{k+1}^j\rvert^\omega\big)$	$\mathcal{O}\big(\lvert S_{k+1}^j\rvert^2\big)$
	Heuristic sparsification [22]	✓	✓	Boundary operator	$\mathcal{O}\big(\lvert S_{k+1}^j\rvert + \lvert\bar{S}_{k+1}^j\rvert^\omega\big)$	$\mathcal{O}\big(\mathrm{Max}(\lvert S_{k+1}^j\rvert, \lvert\bar{S}_{k+1}^j\rvert^2)\big)$
	Power method [23]	✓	✗	Combinatorial Laplacian	$\tilde{\mathcal{O}}\left(\frac{\lvert S_k^i\rvert(k^2\beta_k^i + k(\beta_k^i)^2)}{\Lambda}\right)$	$\mathcal{O}\big(\lvert S_k^i\rvert(k+\beta_k^i)\big)$

[a] It is suggested in Ref. [14] that this approach can be used for computing persistent Betti numbers by exploiting the quantum Zeno effect. We have been unable to reproduce this line of reasoning, as we discuss in Appendix E 2.
[b] One gets this scaling via substituting $|S_k^i|$ by the upper bound $\binom{N}{k+1}$.
[c] We have been unable to verify the correctness of this approach, due to nuances associated with changing the basis from simplices in the complex at scale i to those at scale j. We discuss this in more detail in Appendix E 1.

TABLE I. A comparison of quantum and classical algorithms for topological data analysis. $|S_k^i|$ denotes the number of k-simplices in the complex at scale i. The complexities in Refs. [13, 17, 18] were not explicitly stated in terms of the parameters that we use in this paper, hence the above bounds are our best estimates based on the subroutines used in those works. Initial quantum algorithms were only able to compute regular Betti numbers, not persistent Betti numbers. To compare the scalings of all quantum algorithms on a level footing, we estimate the quantities $\beta_k^i, \beta_k^{i,j}$ to additive error Δ, and work in a regime which we find practically most relevant, where $k \ll N$, $|S_k^i|$ is approximately $\binom{N}{k+1}$ and the evaluated quantity $\beta_k^i, \beta_k^{i,j} = \mathcal{O}\Big(\mathrm{polylog}\binom{N}{k+1}\Big)$. The quantity Λ denotes the smallest non-zero singular value of the matrix used for encoding the topology ($\partial_k^i, \partial_{k+1}^i$, or the combinatorial Laplacian). The gap $\Lambda_{\Pi\Pi}$ appearing in our work is defined in Sec. V D, and equals 1 when computing the non-persistent β_k^i values. The quantity $\bar{S}_k^i$ denotes the number of k-simplices in a sparsified complex, and $\omega \approx 2.4$ is the matrix multiplication exponent.

II. Overview of results

We present and analyse a streamlined quantum algorithm for estimating persistent Betti numbers. We reduce the problem to estimating the normalised rank of a projector that encodes the relevant topological feature of the data, and show how this problem can be efficiently solved using quantum singular value transformation (QSVT) [30]. QSVT provides efficient means for applying a polynomial function to the singular values of a matrix given by a unitary block encoding.

We show how to construct block encodings of operators that represent the topology of a simplicial complex, and how to use QSVT to transform these into projectors onto the relevant subspaces.[1] We consider two possible ways of mapping simplices to qubits; a direct approach (taken by previous quantum algorithms) that uses N qubits for N datapoints, and a compact approach, introduced herein, that uses $(k+1)\log(N)$ qubits to store a k-simplex state. The compact mapping provides an exponential space saving over classical and quantum algorithms for $k = \mathcal{O}(\mathrm{polylog}(N))$.

For a dense simplicial complex constructed from N points in $\mathbb{R}^d$, such that the number of k-simplices scales

[1] We were motivated by Ref. [17], which also used QSVT for computing persistent Betti numbers. Our approach is similar at a high level, but we work with different and more elementary operators to encode the topology. We provide an explicit comparison between our approach and the algorithm of Ref. [17] in Sec. VI A.

roughly as $\binom{N}{k+1}$, and assuming $\beta_k^{i,j} \ll \binom{N}{k+1}$, we have the following main result:[2] (see Appendix D for details)

Theorem 1. *(Persistent Betti number estimation) For the above parameters, we can estimate $\beta_k^{i,j}$ to additive error Δ, with success probability greater than $(1-\eta)$ using*

$$\tilde{\mathcal{O}}\left(\log\left(\frac{\sqrt{\binom{N}{k+1}}}{\Delta}\right)\log\left(\frac{1}{\eta}\right)\right) \tag{1}$$

repetitions of a quantum circuit that has depth

$$\tilde{\mathcal{O}}\left(\frac{N^{3/2}\sqrt{\binom{N}{k+1}}}{\Delta\Lambda_{\Pi\Pi}\mathrm{Min}\left(\Lambda_{\partial_k^i},\Lambda_{\partial_{k+1}^j}\right)}\right), \tag{2}$$

where $\Lambda_{\partial_k^i}, \Lambda_{\partial_{k+1}^j}$ denotes the size of the smallest non-zero singular value of the specified boundary operators, and $\Lambda_{\Pi\Pi}$ is another gap parameter defined in Sec. V D, which has value 1 if $i = j$. The quantum circuit acts on $\mathcal{O}(\min\{k\log(N), N\})$ qubits.

We compare our approach to existing quantum algorithms in Table I. Only the more recent approaches have been able to compute both regular and persistent Betti numbers – the latter being the key quantity of interest for topological data analysis [12]. Prior works took more circuitous routes to computing (persistent) Betti numbers – either by implementing the relevant projections in less efficient ways (e.g., quantum phase estimation) [13, 15, 18], encoding the topology in more complex operators [13, 15, 17, 18], or using incoherent, asymptotically less efficient approaches to estimate subspace dimensions [13, 16–18]. In particular, compared to the existing quantum algorithm [17] for computing $\beta_k^{i,j}$, our approach provides an exponential reduction in the number of qubits required for $k = \mathcal{O}(\mathrm{polylog}(N))$, as well as a time complexity improvement of $\mathcal{O}\left(N^{6.5}\sqrt{\binom{N}{k+1}}\Delta^{-1}\right)$ (assuming similar overheads from the respective gaps Λ of the different operators used for encoding the topology).

For dense complexes, our quantum algorithm achieves a polynomial speedup for the practically relevant task of computing the actual $\beta_k^{i,j}$ values, subject to the dependence of the gap parameters on N. The speedup could be bigger for larger values of k, but is less prominent when compared to heuristic methods that sparsify the complex. Our quantum algorithm also achieves an exponential space saving compared to classical approaches for the practically relevant case of dense complexes in the small k regime. An extended comparison between our quantum algorithm and classical approaches can be found in Sec. VI D.

[2] For technical reasons we also need that $\log(d)\log(b) < N$, where b is the number of bits required to describe the coordinates.

The rest of this manuscript is laid out as follows. In Sec. III we provide a self-contained introduction to the aspects of topological data analysis required to understand our quantum algorithm (we provide a more extended and pedagogical introduction in Appendix A). We then introduce the quantum algorithm used for computing persistent Betti numbers in Sec. IV. In Sec. V we outline the costs of the individual building blocks of our quantum algorithm, with detailed discussion in the Appendices. In Sec. VI we discuss the complexity of our quantum algorithm, classical algorithms, regimes for quantum advantage, and quantum speedups for practical problems of interest. We conclude in Sec. VII.

III. Background on topological data analysis

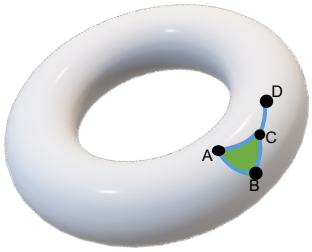


FIG. 1. An example subset of datapoints arising from an underlying toroidal manifold. The datapoints (0-simplices A, B, C, D) are ordered alphabetically. After defining a length scale within which to connect datapoints, we add the relevant simplices. We show 1-simplices AB, BC, AC, CD (blue) and a 2-simplex ABC (green) that form elements of the resulting simplicial complex for this dataset. Simplices inherit the ordering of the vertices. We ascribe an orientation to the simplices based on the ordering of the vertices. An odd permutation of the vertices changes the orientation of the simplex; i.e. $AC = -CA$.

We seek to compute $\beta_k^{i,j}$ for a sufficient number of dimensions k and scales i, j so that we can capture the main topological features of the dataset at hand. As illustrated in Fig. 1, to understand the topology at scale i we connect datapoints within distance μ_i, generating a graph from the dataset. We use the so-called *clique complex* of the graph, where simplices correspond to the cliques in the graph: a $(k+1)$-clique defines a k-simplex. This way every $k+1$ mutually connected vertices induce a k-simplex, which we can think of as the convex hull of the corresponding data points. The set of simplices and all of their constituent faces (e.g., the 2-simplex ABC has 1-simplices AB, BC, AC and 0-simplices A, B, C as faces) at scale i is referred to as the simplicial complex S^i, while S_k^i denotes the k-simplices in S^i. Accordingly, the number of simplices in the complex at scale i is denoted by $|S^i|$, and the number of k-simplices by $|S_k^i|$.

This simplicial complex is known as a Vietoris-Rips complex, and is one possible choice [31] for approximat-

ing the underlying topological space. The maximum number of k-simplices $|S_k^i|$ is $\binom{N}{k+1}$. Mapping the simplices to orthonormal basis vectors enables the use of linear algebraic tools for understanding the topology.

The goal is to identify holes in the complex; a hole is a region of empty space, demarcated by its boundary. Studying these boundaries is crucial for finding holes, and so we define a boundary operator. The k-th boundary operator maps k-simplices to their oriented boundaries. The action of the boundary operator ∂ on a k-simplex is

$$\partial[v_0, ..., v_k] = \sum_{l=0}^{k} (-1)^l [v_0, ...\hat{v}_l, ...v_k], \tag{3}$$

where $v_0...v_k$ are the ordered vertices in the k-simplex, and $\hat{v}_l$ means the vertex is excluded from the simplex. We denote by $\partial_k^i \colon \langle S_k^i \rangle \to \langle S_{k-1}^i \rangle$ the boundary operator on the complex at length scale i restricted to the subspaces $\langle S_k^i \rangle$ and $\langle S_{k-1}^i \rangle$ spanned by the basis vectors corresponding to k and $k-1$ simplices present at scale i. For example, $\partial_1^i[CD] = D - C$, and $\partial_2^i[ABC] = BC - AC + AB$.

We can form 'chains' of k-simplices by taking linear combinations of their corresponding vectors. The boundary operator extends linearly onto chains of k-simplices. When applying the boundary operator to a closed chain of k-simplices (known as a k-cycle), each $(k-1)$-face appears twice, each time with opposite signs. For example,

$$\partial_1^i[AB + BC - AC] = B - A + C - B - C + A = 0. \tag{4}$$

More formally, all k-cycles are in the kernel of ∂. A k-cycle can surround either empty space, or will correspond to the boundary of a $(k+1)$-chain in the complex (the example above corresponds to this latter case). The latter such k-cycles are in the image of ∂.

To count holes in the complex at scale i, one can consider all k-cycles, and remove those that are the boundaries of $(k+1)$-chains in the complex. Accordingly, the k-th Betti number at scale i is defined [32] as

$$\beta_k^i = \dim\big(\mathrm{Ker}(\partial_k^i)\big) - \dim\big(\mathrm{Im}(\partial_{k+1}^i)\big), \tag{5}$$

where the first term counts the number of 'hole-like' objects, and the latter removes those that are boundaries of $(k+1)$-simplices, leaving the number of true holes in the complex. Note that since ∂_ℓ^i is restricted to the subspaces $\langle S_\ell^i \rangle$ and $\langle S_{\ell-1}^i \rangle$ we have $\mathrm{Ker}(\partial_k^i), \mathrm{Im}(\partial_{k+1}^i) \subseteq \langle S_k^i \rangle$.

Computing the Betti numbers at different scales is not sufficient to determine the persistence of topological features, as the Betti number does not uniquely identify the holes [12]. In order to determine how many holes at scale i are still present at scale j, one can consider the simplicial complexes formed at both scales (S^i, S^j), and modify Eq. (5) to give the persistent Betti numbers [19] as

$$\beta_k^{i,j} = \dim\big(\mathrm{Ker}(\partial_k^i)\big) - \dim\Big(\mathrm{Ker}(\partial_k^i) \cap \mathrm{Im}(\partial_{k+1}^j)\Big). \tag{6}$$

The first term in this expression again counts all hole-like objects at scale i. The second term removes all holes that were present at scale i, but have been 'filled-in' by linear combinations of $(k+1)$-simplices at scale j (which automatically includes any $(k+1)$-simplices at scale i that fill-in holes at scale i). As above $\mathrm{Ker}(\partial_k^i) \subseteq \langle S_k^i \rangle$ and $\mathrm{Im}(\partial_{k+1}^j) \subseteq \langle S_k^j \rangle$, while $\langle S_k^i \rangle \subseteq \langle S_k^j \rangle$.

Once the persistent Betti numbers have been computed for all pairs of scales, they are typically summarised in a persistence diagram, that shows when topological features are created and destroyed. Different datasets can be compared by computing the similarities between these diagrams, either using stable distance measures, or vectorization of the diagrams [1, 8, 31].

IV. Quantum algorithm for persistent Betti numbers

Unlike in our quantum algorithm, prior works did not directly use the formula in Eq. (6) for computing (persistent) Betti numbers, for a number of reasons. One such reason is that the boundary operator is not Hermitian, so one cannot directly use traditional approaches such as quantum phase estimation. Therefore, earlier works mostly computed Betti numbers by utilizing the combinatorial Laplacian and related objects (see Appendix A), that either made it difficult to extend methods from Betti numbers to persistent Betti numbers [13, 15, 16, 25] or resulted in higher computational complexity [17].

We reduce the problem of estimating persistent Betti numbers to that of estimating the ratio of the ranks of two orthogonal projectors. We present a quantum native approach in the QSVT framework for solving this problem. Consider the orthogonal projectors $\Pi \subseteq \widetilde{\Pi}$. If we treat Π as an observable and $\rho := \widetilde{\Pi}/\mathrm{rank}(\widetilde{\Pi})$ as a quantum state, then the ratio of the ranks $\frac{\mathrm{rank}(\Pi)}{\mathrm{rank}(\widetilde{\Pi})}$ equals Π's expectation value when evaluated on the mixed state ρ. We can speed up the estimation of this expectation value using amplitude estimation if we replace the mixed state ρ by its purification $|\psi\rangle$. For example, when $\widetilde{\Pi} = I$, we choose $|\psi\rangle$ to be the maximally entangled state. We require a state preparation unitary V_ψ for $|\psi\rangle$ and a projected unitary encoding [30] V_Π of Π (we provide a definition in Appendix B). This leads to the following result (we provide a full proof in Appendix B 1 where we formulate amplitude estimation as a singular value estimation problem with the same asymptotic complexity, in order to track the propagation of errors).

Theorem 2. *(Normalized projector rank estimation) Given the above ingredients, we can estimate* $\sqrt{\frac{\mathrm{rank}(\Pi)}{\mathrm{rank}(\widetilde{\Pi})}}$ *to additive error* δ *with success probability* $\geq 1 - \eta$ *using* $\mathcal{O}\big(\log(\eta^{-1})\big)$ *incoherent repetitions of a quantum circuit, which makes* $\mathcal{O}\big(\frac{1}{\delta}\big)$ *calls to* V_ψ *and* V_Π *(and their inverses), and uses* $\widetilde{\mathcal{O}}\big(\frac{1}{\delta}\big)$ *additional two-qubit gates.*

Proof. We employ amplitude estimation with precision δ to the following process: prepare $|\psi\rangle$ and measure Π. We repeat this process $\mathcal{O}\big(\log(\eta^{-1})\big)$ times and take the median of the estimates, which boosts the success probability exponentially due to the Chernoff bound. □

We compute (persistent) Betti numbers by the above subroutine via estimating the normalized rank of $\Pi_{\mathrm{Ker}(\partial_k^i)} - \Pi_{\mathrm{Ker}(\partial_k^i)\cap\mathrm{Im}(\partial_{k+1}^j)} \preceq \Pi_{\langle S_k^i\rangle}$ to sufficient precision, cf. Appendix B 2.

To obtain unitary block-encodings $V_{\Pi_{\mathrm{Ker}}}$ (of $\Pi_{\mathrm{Ker}(\partial_k^i)}$) and $V_{\Pi_{\mathrm{Im}}}$ (of $\Pi_{\mathrm{Im}(\partial_{k+1}^j)}$) we apply approximate threshold functions to ∂_k^i and $(\partial_{k+1}^j)^\dagger$ via QSVT, which introduces a reciprocal dependence on the smallest non-zero singular values $\Lambda_{\partial_k^i}$, $\Lambda_{\partial_{k+1}^j}$ of ∂_k^i and ∂_{k+1}^j respectively.

In general $\mathrm{Ker}(\partial_k^i)\cap\mathrm{Im}(\partial_{k+1}^j)$ is spanned by those singular vectors of $\Pi_{\mathrm{Ker}(\partial_k^i)}\cdot\Pi_{\mathrm{Im}(\partial_{k+1}^j)}$ that have singular value 1. We can (approximately) implement a block encoding of $\Pi_{\mathrm{Ker}(\partial_k^i)\cap\mathrm{Im}(\partial_{k+1}^j)}$ by once again applying a threshold function via QSVT to $\Pi_{\mathrm{Ker}(\partial_k^i)}\cdot\Pi_{\mathrm{Im}(\partial_{k+1}^j)}$ (but now with a threshold of approximately 1). The required sharpness of the threshold is determined by the quantity $\Lambda_{\Pi\Pi}$, the deviation of the largest $\neq 1$ singular value of $\Pi_{\mathrm{Ker}(\partial_k^i)}\cdot\Pi_{\mathrm{Im}(\partial_{k+1}^j)}$ from 1. This results in an additional factor of $\Lambda_{\Pi\Pi}^{-1}$ in the complexity of the algorithm. For ordinary Betti numbers (i.e., $i=j$) we have $\Lambda_{\Pi\Pi}=1$ since $\mathrm{Im}(\partial_{k+1}^i)\subseteq\mathrm{Ker}(\partial_k^i)$ and thus $\Pi_{\mathrm{Ker}(\partial_k^i)}\cdot\Pi_{\mathrm{Im}(\partial_{k+1}^j)}=\Pi_{\mathrm{Im}(\partial_{k+1}^i)}$.

In order to determine the overall resource costs for estimating persistent Betti numbers using our quantum algorithm, we need to determine the cost of the block encodings of the relevant operators. In the following section, we discuss how to implement the block encodings listed above. These require a number of ingredients. The most complex case of $\Pi_{\mathrm{Ker}\cap\mathrm{Im}}$ (similarly to the other cases), uses the following ingredients, whose costs we establish for both the direct and compact mappings:

- Implement the 'membership oracle' that determines if a given k-simplex is present in the complex at scale i, using a sequence of elementary operations, including those used for querying some form of quantum memory that stores the coordinates of the data points.
- Obtain block encodings of ∂_k^i, ∂_{k+1}^j using elementary operations and the membership oracle.
- Obtain block encodings of Π_{Ker}, Π_{Im}, using QSVT applied to block encodings of ∂_k^i, ∂_{k+1}^j, respectively.

We determine these resource in order to bound the end-to-end complexity of our algorithm.

V. Resource costs of building blocks

A. Mapping simplices to qubits

We consider two approaches to map simplices to quantum states: the direct mapping using $\mathcal{O}(N)$ qubits for a k-simplex on N datapoints, and the compact mapping using $\mathcal{O}(k\log(N))$ qubits, introduced herein.

The direct mapping, considered in previous quantum algorithms [13], encodes k-simplices as Hamming weight-$(k+1)$ computational basis states of N qubits. Each qubit corresponds to a datapoint, and its value is $|1\rangle$ if and only if the vertex corresponding to the datapoint is incident to the given simplex.[3]

The compact mapping encodes a k-simplex built from an N-point dataset using $(k+1)\lceil\log(N+1)\rceil$ qubits. Each of the $(k+1)$ registers represents a vertex present in the simplex. The vertices are enumerated in binary, starting from 1. The all-0 state $|\bar{0}\rangle$ denotes the absence of a vertex, and is required for implementing the boundary operator. The ordering of the vertices in the state is the same as the ordering chosen for the datapoints.

k	Simplex	Direct	Compact
0	A	$\|1000000\rangle$	$\|001\rangle$
1	BD	$\|0101000\rangle$	$\|010\rangle\,\|100\rangle$
2	CEG	$\|0010101\rangle$	$\|011\rangle\,\|101\rangle\,\|111\rangle$

TABLE II. Example mappings from simplices to qubits for simplices built from a vertex set $\{A,...,G\}$.

For N datapoints, there are $\binom{N}{k+1}$ possible k-simplices. Classically storing generic linear combinations of these requires $\Omega\binom{N}{k+1}$ memory, while a quantum register can be put into a superposition of all possible k-simplex states. The compact mapping can thus provide an exponential reduction in spatial resources over classical approaches, for dense complexes. The compact mapping is also an exponential improvement over the direct mapping when $k=\mathcal{O}(\mathrm{polylog}(N))$.

B. Membership oracle cost

In order to prepare $|\psi_m\rangle$ a purification of $\Pi_{\langle S_k^i\rangle}/|S_k^i|$ and to restrict the boundary operators to act only on simplices present in the complex, the algorithm makes calls to a membership oracle $O_{m_k^i}$. The membership oracle determines if a k-simplex is present in the complex at scale i (or not) based on the positions of the vertices $\{\vec{r}_\alpha\}$, and the length scale μ_i considered. The membership oracle

[3] Here we mean that the vertex is part of the defining $(k+1)$-clique in the corresponding graph, rather than the datapoint being in the complex hull of the points geometrically corresponding to the simplex.

acts as

$$O_{m_k^i}|s_k\rangle|a\rangle = \begin{cases} |s_k\rangle|a\oplus 1\rangle & \text{if } s_k \in S_k^i \\ |s_k\rangle|a\oplus 0\rangle & \text{if } s_k \notin S_k^i \end{cases} \tag{7}$$

We explicitly show how to implement the membership oracle for both the direct and compact mappings in Appendix C 1. It is often claimed that the quantum algorithm for TDA does not require quantum RAM. While it is possible to implement the algorithm without it [16], typical formulations do in fact use QRAM [13]. However, in contrast to other quantum machine learning algorithms, the QRAM used in TDA is not exponentially larger than the number of qubits used for storing the main register.

	Memory	Space	Non-Clifford depth
Direct	QROM	N	$N\log(N)$
Compact	QROM	$\max\{k\log(N), \log(N)+db^2\}$	$k+\sqrt{k}\log(\epsilon^{-1})\times (N+\log(d)\log(b)+b)$
	QRAM	$Ndb+db^2$	$k+\sqrt{k}\log(\epsilon^{-1})\times (\log(N)+\log(d)\log(b)+b)$

TABLE III. A comparison of the asymptotic time and space complexity of implementing the membership oracle in the two encodings, using the optimal memory models for each. The datapoints are in $\mathbb{R}^d$ and each coordinate is represented with b bits, chosen sufficiently large to suppress under/overflow errors in the distance calculation. The membership oracle for the compact mapping has an operator norm error of ϵ.

Our membership oracle for direct mapped simplices follows the approach to clique finding in Ref. [33], and uses a QROM (quantum read-only memory) that can be implemented with a purely classical memory by going through a (classical) list of present edges, and applying a Toffoli gate on the adjacent vertices to increment a counter. The counter is used for verifying that all edges of the simplex are present.

Our membership oracle for compact mapped simplices also makes use of quantum memory [34]; either QRAM (performs fast loads using many ancilla qubits) or QROM [35] (performs slow loads using few ancilla qubits and a classical memory). We coherently load the datapoints in the simplex, and calculate the distance between each pair of them. Comparing the distance against the length scale μ_i flags edges that are not present and yields a membership oracle implementation. The compact membership oracle also verifies that the vertices are stored in increasing order for consistency.

TDA is widely applied to high dimensional data. In order to reduce the cost of the membership oracle in the compact mapping, we can implement a classical preprocessing step. We can use the Johnson–Lindenstrauss lemma to embed the high-dimensional vertices into a lower dimensional space, while approximately maintaining the distances between vertices. The dimension d can be reduced to $\mathcal{O}\big(\log(N)/\epsilon_{JL}^2\big)$ to ensure the distances between all the points stay accurate to a factor of $(1\pm\epsilon_{JL})$.

C. Boundary operator cost

In this section we discuss the resource costs to construct block encodings of the boundary operator ∂_k^i and ∂_{k+1}^j according to Eq. (A2). Essentially, we construct a unitary $V_{\partial_k^i}$ such that $(\langle 0|^{\otimes a}\otimes I)V_{\partial_k^i}(|0\rangle^{\otimes a}\otimes I) = \frac{1}{\alpha}\partial_k^i$ (and analogously for ∂_{k+1}^j). For resource efficiency, we are actually working with a generalization of block encodings called projected unitary encoding defined in Appendix B. Since these work very similarly to block encoding and can be often converted to actual block encodings with minimal overheads we use the terms interchangeably in the paper. The costs are summarised in Table IV, and are derived in Appendix C 2.

Our compact mapped approach proceeds by coherently swapping the vertex to be deleted into the final position of the register, and then setting it to $|\bar{0}\rangle$ to represent the absence of a vertex.

Our direct mapped approach relies on a previously observed correspondence between the boundary operator and second quantized fermionic operators: $\partial+\partial^\dagger = \sum_i a_i + a_i^\dagger$ [16, 36, 37]. We considered existing approaches for implementing projected unitary encodings of such operators [37, 38]. The chosen approach [37] for the direct mapping is a $(\sqrt{N},0,0)$ projected unitary encoding of ∂_k^i that can be implemented with $\mathcal{O}(\log(N))$ non-Clifford gate depth. This can be compared with the approach of Ref. [17] that implemented a $(N(k+1),\mathcal{O}(\log(N),0)$ projected unitary encoding of ∂_k^i requiring $\Omega(N)$ non-Clifford depth.

D. Subspace projector cost

As discussed in Sec. IV we use unitary block encodings $V_{\Pi_{\mathrm{Ker}}}$ (of $\Pi_{\mathrm{Ker}(\partial_k^i)}$) and $V_{\Pi_{\mathrm{Im}}}$ (of $\Pi_{\mathrm{Im}(\partial_{k+1}^j)}$), which are in turn used for building $V_{\Pi\Pi}$ (of $\Pi_{\mathrm{Ker}\cap\mathrm{Im}}$). In this section we present an outline of the methods used for building these block encodings, and state their cost in Table V and Table VI. We refer to Appendix C 3 for details.

The QSVT circuit for implementing $V_{\Pi_{\mathrm{Ker}}}$ uses singular value threshold projectors [30]. This is a generalization of eigenvalue threshold projectors that are used for example in [39], where quantum eigenvalue transformation is used for applying a filter function that projects out eigenvectors above a given threshold. In [39], to prepare the ground state, the threshold is set lower than the gap Λ between the ground and first excited state. We can use the polynomial approximations of the threshold function used in either of [30, 39], but we need to carefully track the propagation of errors stemming from the approximation error ϵ_i. A technical detail is that we need to use even singular value transformation ensuring that the left and right singular vectors coincide, so that we end up with an (approximate) orthonormal projector.

We can implement $V_{\Pi_{\mathrm{Im}}}$ similarly, by constricting an approximate singular value threshold projector Π from

	(α, m, ϵ)	Non-Clifford gate depth	Projectors
Compact	$(\sqrt{(N+1)(k+1)}, \log(k+1), 0)$	$\mathcal{O}(k \log\log(N+1))$	$\Pi_L = \Pi_{m^i_{k-1}} \otimes \lvert 0\rangle\langle 0\rvert^{\otimes \log(N+1)}$ $\Pi_R = \Pi_{m^i_k}$
Direct	$(\sqrt{N}, 0, 0)$	$\mathcal{O}(\log(N))$	$\Pi_L = \Pi_{m^i_{k-1}}$ $\Pi_R = \Pi_{m^i_k}$

TABLE IV. Parameters for the projected unitary encoding $V_{\partial^i_k}$ of ∂^i_k. The parameters for the projected unitary encoding $V_{\partial^j_{k+1}}$ of ∂^j_{k+1} follow from replacing $k \to k+1$ and $i \to j$. The projector $\Pi_{m^i_k}$ projects onto k-simplices in the complex at scale i. The implementation of this operation differs for the compact and direct encodings, as discussed in Sec. V B.

	(α, m, ϵ)	Costs
Compact	$(1, \log(k+1)+3, \epsilon_{\text{ker}})$	$\mathcal{O}\left(\frac{\sqrt{(N+1)(k+1)}}{\Lambda_{\partial^i_k}} \log\left(\frac{1}{\epsilon_k}\right)\right)$ $\times V_{\partial^i_k}, V^\dagger_{\partial^i_k}, O_{m^i_k}, O_{m^i_{k-1}}$
Direct	$(1, 3, \epsilon_k)$	$\mathcal{O}\left(\frac{\sqrt{N}}{\Lambda_{\partial^i_k}} \log\left(\frac{1}{\epsilon_k}\right)\right)$ $\times V_{\partial^i_k}, V^\dagger_{\partial^i_k}, O_{m^i_k}, O_{m^i_{k-1}}$

TABLE V. A summary of the costs to implement the block encoding $V_{\Pi_{\text{Ker}}}$. The error parameter ϵ_{ker} depends on a number of other parameters (including ϵ_m), as shown in Appendix C 3 b.

$(\partial^j_{k+1})^\dagger$, and then taking $I - \Pi$.

Given the above methods for implementing $V_{\Pi_{\text{Ker}}}$ and $V_{\Pi_{\text{Im}}}$, we can implement $V_{\Pi\Pi}$. We cannot simply take a product of the two projectors, as in general they do not commute with each other (because of new simplices that enter the complex at scale j). However, we can implement $V_{\Pi\Pi}$ by first taking the product of the block-encoded matrices Π_{Ker} and Π_{Im} [30], and then applying QSVT to the resulting block encoding of $\Pi_{\text{Ker}} \cdot \Pi_{\text{Im}}$. We apply a function that sends all $\neq 1$ singular values to 0. In reality, we can only send singular values below a threshold $(1 - \Lambda_{\Pi\Pi})$ to zero, and we choose $\Lambda_{\Pi\Pi}$ so that all $\neq 1$ singular values of $\Pi_{\text{Ker}} \cdot \Pi_{\text{Im}}$ are below $(1 - \Lambda_{\Pi\Pi})$. Such a thresholding costs $\mathcal{O}\big(\Lambda^{-1}_{\Pi\Pi} \log(\epsilon_p^{-1})\big)$ calls to $V_{\Pi_{\text{Ker}}}$ and $V_{\Pi_{\text{Im}}}$ to implement $V_{\Pi\Pi}$ with precision ϵ_p.

E. State preparation cost

In this section we describe how to implement V_{ψ_m} preparing $|\psi_m\rangle$ from the all-0 state, where $|\psi_m\rangle$ is a purified maximally mixed state over k-simplices in the complex at scale i. We discuss our approach in more detail in Appendix C 4.

We use fixed-point amplitude amplification to construct V_{ψ_m}. In both the compact and direct encodings case the algorithm uses $\widetilde{\mathcal{O}}\left(\sqrt{\frac{\binom{N}{k+1}}{|S^i_k|}} \log(\epsilon_\psi^{-1})\right)$ calls to $O_{m^i_k}$, and to an operator U_{uni} (and its inverse) that prepares a purification of the maximally mixed state over all possible k-simplices to prepare the state with precision ϵ_ψ.

In the direct mapping, k-simplices are Hamming weight $(k+1)$ computational basis states on N qubits. We can implement U_{uni} using the approaches in Ref. [40, 41] for preparing Dicke states, the most efficient of which has $\mathcal{O}(k \log(N))$ depth.

In the compact mapping, we can construct a uniform superposition of k-simplices (with correctly ordered vertices) by first placing all vertex registers into an equal superposition. We then use fixed-point amplitude amplification to convert this state to a superposition with no repeated vertices (but ignoring the ordering of vertices). Finally, we can use a reversible quantum sorting network [42], to prepare an equal superposition over all k-simplices, with their vertices correctly ordered. This requires

$$\mathcal{O}\left((\log(N) + k\log(k))\left(\sqrt{\frac{(k+1)^{(k+1)}}{(k+1)!}} \log(\epsilon_s^{-1})\right)\right) \tag{8}$$

non-Clifford gate depth, and uses $\mathcal{O}(k\log(k) + k\log(N))$ ancilla qubits. The unitary circuit has an error of $\sqrt{\epsilon_s}$.

VI. Discussion

A. Quantum complexity

In Appendix D we determine the overall complexity of our quantum algorithm for computing persistent Betti numbers, using the costs of the building blocks introduced in the previous sections. We assume that $\text{Ker}(\partial^i_k) \cap \text{Im}(\partial^j_{k+1}) \sim \text{Ker}(\partial^i_k) \ll |S^i_k|$, which lets us bound the complexity of our algorithm by the complexity of estimating $\text{Ker}(\partial^i_k) \cap \text{Im}(\partial^j_{k+1})$. We also make the approximation that all calls to the membership oracle have the same cost for $(k-1), k, (k+1)$. The cost of the quantum circuit (in either encoding) is given by:

$$\mathcal{O}\left(\frac{\sqrt{|S^i_k|}}{\Delta} \log\left(\frac{1}{\epsilon_f}\right) \times (V_{\psi_m} + V_{\Pi\Pi})\right) \tag{9}$$

Using the direct mapping, we show in Appendix D 1 that the algorithm has complexity given by the expression in Table VII, to estimate the persistent Betti number to additive error Δ with success probability $\geq 1-\eta$. The quantum circuit acts on $\mathcal{O}(N)$ qubits. Taking $|S^i_k| \sim \binom{N}{k+1}$

	(α, m, ϵ)	Costs
Compact	$\left(1, 2\log(k+1)+5, \frac{1}{\Lambda_{\Pi\Pi}}\log\left(\epsilon_p^{-1}\right)\sqrt{\epsilon_{\text{ker}}+\epsilon_{\text{im}}+8\sqrt{\epsilon_m}}+\epsilon_p\right)$	$\mathcal{O}\left(\frac{1}{\Lambda_{\Pi\Pi}}\log\left(\frac{1}{\epsilon_p}\right)\right)\times$ $V_{\Pi_{\text{Ker}}}, V^\dagger_{\Pi_{\text{Ker}}}, V_{\Pi_{\text{Im}}}, V^\dagger_{\Pi_{\text{Im}}}, O_m$
Direct	$\left(1, 5, \frac{1}{\Lambda_{\Pi\Pi}}\log\left(\epsilon_p^{-1}\right)\sqrt{\epsilon_k+\epsilon_i}+\epsilon_p\right)$	

TABLE VI. A summary of the costs to implement the block encoding $V_{\Pi\Pi}$. The error parameters ϵ_{ker} and ϵ_{im} depend on a number of other parameters (including ϵ_m), as shown in Appendix C 3 d. We use the shorthand O_m to indicate that we need membership oracles for $O_{m^i_k}$, $O_{m^i_{k-1}}$, $O_{m^j_k}$, $O_{m^j_{k+1}}$.

and hiding logarithmic factors with the big-$\tilde{\mathcal{O}}$ notation recovers Theorem 1. We can compare this scaling to the corresponding complexity of the quantum algorithm for calculating persistent Betti numbers in Ref. [17], which we also show in Table VII. That work used the QSVT to construct the persistent combinatorial Laplacian (see Appendix A for details), and to project onto its kernel, which encodes the persistent Betti numbers [43]. We see that our approach provides a large polynomial improvement in $|S^i_k|, N, k$ over the prior state-of-the-art.

The corresponding expression for the compact mapping (with a QROM-based implementation of the membership oracle) contains additional terms, as shown in Appendix D 2. We will simplify the expression by noting that the most interesting regime for the quantum algorithm is $|S^i_k| \sim \binom{N}{k+1}$, as this is the regime that is challenging for classical algorithms, which scale with $|S^i_k|$. We further assume $N \gg k, \log(x)\log(b_x), b_x$, as this is the regime where classical algorithms are typically applied. The resulting complexity is

$$\tilde{\mathcal{O}}\left(\frac{\sqrt{\binom{N}{k+1}}}{\Delta}\log^2\left(\frac{\sqrt{\binom{N}{k+1}}}{\Delta}\right)\log\left(\frac{1}{\epsilon_m}\right)\log\left(\frac{1}{\eta}\right)\times\left[\left((\log(N)+k\log(k))\sqrt{\frac{(k+1)^{k+1}}{(k+1)!}}\right)+\frac{N^{3/2}k}{\Lambda_{\Pi\Pi}\text{Min}\left(\Lambda_{\partial^i_k},\Lambda_{\partial^j_{k+1}}\right)}\right]\right) \tag{10}$$

where the expression for ϵ_m is specified in Appendix D 2. The quantum circuit acts on $\mathcal{O}\big(k\log(N)\big)$ qubits. Taking k constant and hiding logarithmic factors with the big-$\tilde{\mathcal{O}}$ notation recovers Theorem 1. We see that in this regime, our compact mapped approach provides an improvement of $\mathcal{O}\left(\frac{N^{6.5}\sqrt{\binom{N}{k+1}}}{\Delta}\right)$ over Ref. [17] (assuming similar scalings for the respective gaps Λ), as well as an exponential reduction in the number of qubits required.

Both of these quantum algorithms have dependencies on two different gap parameters; in our case, $\Lambda_{\Pi\Pi}, \text{Min}\left(\Lambda_{\partial^i_k}, \Lambda_{\partial^j_{k+1}}\right)$ and in the case of Ref. [17] Λ_1, Λ_2 (where Λ_1 is the gap of a submatrix of $\partial^j_{k+1}\left(\partial^j_{k+1}\right)^\dagger$, and Λ_2 is a the gap of the persistent combinatorial Laplacian). The gaps $\text{Min}\left(\Lambda_{\partial^i_k}, \Lambda_{\partial^j_{k+1}}\right)$ and Λ_2 arise due to the need to project into the relevant kernel/image spaces, and their non-persistent analogues are also present when calculating regular Betti numbers. However, the gaps $\Lambda_{\Pi\Pi}$ and Λ_1 have a more subtle origin, that appears to be specific to the problem of calculating persistent Betti numbers. In Ref. [17], this gap arises from performing a change of basis necessary to build the persistent combinatorial Laplacian (see Appendix E for further discussion). In our algorithm, this gap arises from transforming the projector $\Pi_{\text{Ker}(\partial^i_k)}\cdot\Pi_{\text{Im}(\partial^j_{k+1})}$ to $\Pi_{\text{Ker}\cap\text{Im}}$ by applying the QSVT. This is necessary because $\Pi_{\text{Ker}(\partial^i_k)}, \Pi_{\text{Im}(\partial^j_{k+1})}$ do not commute in general. In both cases, this additional complexity arises due to the additional simplices that are present at stage j, that are not present at stage i. This makes it necessary to perform an effective change of basis, to ensure only the relevant simplices in j that can fill holes in i are considered. It remains an open question as to how the gaps of the boundary operators $\Lambda_{\partial^i_k}, \Lambda_{\partial^j_{k+1}}$, and of the combined projector $\Pi_{\text{Ker}}\Pi_{\text{Im}}$ scale as a function of N, k. A number of conjectures about the scaling of the gap of the combinatorial Laplacian were given in Ref. [23], but we are unaware of any proofs or further evidence towards these conjectures, or extensions to persistent operators. The scaling of the gap parameters will determine the viability of quantum algorithms for topological data analysis, and so it is critical to establish how these gaps scale for problems of interest.

B. Classical complexity

There are a number of classical algorithms for persistent Betti numbers. The most widely used approach [19] (referred to as the 'textbook', 'standard', or 'column' algorithm) proceeds by 'reducing' an $|S^j_{k,k+1}| \times |S^j_{k-1,k}|$

Approach	Complexity
This work (Direct mapping)	$\tilde{\mathcal{O}}\left(\frac{N\log(N)\sqrt{\lvert S_k^i\rvert}}{\Delta}\log^2\left(\frac{\sqrt{\lvert S_k^i\rvert}}{\Delta}\right)\log\left(\frac{1}{\eta}\right)\times\left(\sqrt{\frac{\binom{N}{k+1}}{\lvert S_k^i\rvert}}+\frac{\sqrt{N}}{\Lambda_{\Pi\Pi}\mathrm{Min}\left(\Lambda_{\partial_k^i},\Lambda_{\partial_{k+1}^j}\right)}\right)\right)$
Ref. [17] (Direct mapping)	$\tilde{\mathcal{O}}\left(\frac{N^2\lvert S_k^i\rvert}{\Delta^2}\log\left(\frac{\sqrt{\lvert S_k^i\rvert}}{\Delta}\right)\times\left(\sqrt{\frac{\binom{N}{k+1}}{\lvert S_k^i\rvert}}+\frac{N^6k^4}{\Lambda_1^2\Lambda_2}\log\left(\frac{\sqrt{\lvert S_k^i\rvert}}{\Delta}\right)\right)\right)$

TABLE VII. A comparison of the asymptotic complexity of our algorithm using the direct encoding, and the approach of Ref. [17]. The overall complexity of the latter approach is not explicitly stated in that work, and so we have used our best estimates based on the subroutines given. The gap parameters Λ_1, Λ_2 refer to the gaps of a submatrix of $\partial_{k+1}^j(\partial_{k+1}^j)^\dagger$, and of the persistent combinatorial Laplacian, respectively.

boundary matrix to identify pairs of simplices that create and (later) destroy a topological feature. The algorithm requires only a single repetition to compute the persistent Betti numbers at all length scales. This algorithm has worst-case runtime of $\mathcal{O}(|S_{k,k+1}^j|^3)$ [44], although the runtime can approach linear in practice, due to sparsity in the complex [20], and its constant factors can be optimised [45, 46]. The complexity can be improved to $\mathcal{O}(|S_{k,k+1}^j|^\omega)$ [20, 21] (where $\omega \approx 2.4$ is the exponent for matrix multiplication).

An alternative approach, which has parallels with the quantum algorithm for TDA, is to use the power method to find the dimension of the kernel of the combinatorial Laplacian matrix [23]. This approach has only been used for computing Betti numbers. Like quantum algorithms for TDA, this approach also has a runtime that depends on the gap between the ground and first excited state of the operator used to encode the topology of the complex. This approach has a time complexity of $\tilde{\mathcal{O}}\left(\frac{|S_k^i|(k^2\beta_k^i+k(\beta_k^i)^2)}{\Lambda}\right)$ and a spatial complexity of $\mathcal{O}\left(|S_k^i|(k+\beta_k^i)\right)$. This complexity provides an apples-to-apples comparison for using quantum algorithms for computing regular Betti numbers. In the dense complex regime of $|S_k^i| \sim \binom{N}{k+1}$, the quantum algorithm only achieves an approximately quadratic speedup in N.

With the recent introduction of the persistent combinatorial Laplacian [43, 47, 48], the power method could also be used for computing persistent Betti numbers. However, the asymptotic cost of building the persistent combinatorial Laplacian is the same as that of diagonalizing it [43], so there would likely be no benefit from performing the power method with this object.

Classical techniques have also been introduced that prune simplices from the complex. While it is NP-hard to find the maximally pruned complex [49], a number of heuristic approaches have been developed [22, 50–54] (and see Ref. [31] Secs. 3.1 and 5.2.6). If the number of simplices in the reduced complex is $|\bar{S}|$, the algorithm of Ref. [22] reduces the complexity of calculating persistent Betti numbers from $\mathcal{O}(|S_{k,k+1}^j|^\omega)$ to $\mathcal{O}(|S_{k,k+1}^j| + |\bar{S}|^\omega)$. When $|\bar{S}| \ll |S_{k,k+1}^j|$ the algorithm runs approximately linearly in the number of simplices in the original complex.

C. Regimes for quantum advantage

We can consider the regimes in which quantum algorithms may provide a speedup over the classical approaches listed above. For k constant, both the quantum and classical algorithms run in polynomial time, preventing quantum algorithms from achieving an exponential speedup. For k scaling with N, all known classical algorithms run in superpolynomial time. This is likely to be a fundamental result; for the independence complex (closely related to the Vietoris-Rips complex considered here) determining if β_k^i is non-zero is NP-hard (but not generally in NP), and calculating β_k^i is #P-hard [55]. We are not aware of any applications that use large k persistent Betti numbers – but it is unclear if this is due to the computational difficulties of obtaining such values, or if they are of no intrinsic value.

When our task is to estimate $\beta_k^{i,j}$ to constant additive error Δ, all known quantum algorithms are similarly unable to achieve an exponential speedup over classical algorithms. The limitation of quantum algorithms stems from their competing subroutines. The algorithms need to efficiently find simplices in the complex, so $|S_k^i|$ should only be polynomially smaller than $\binom{N}{k+1}$. On the other hand, the algorithms compute $\beta_k^{i,j}/|S_k^i|$ to additive error δ with an overhead of $poly(\delta^{-1})$. Converting the estimate of $\beta_k^{i,j}/|S_k^i|$ to an estimate of $\beta_k^{i,j}$ reintroduces the superpolynomial factor $|S_k^i|$, and eliminates the previously claimed [13, 16–18] exponential speedup when one needs to compute these quantities to constant additive error. A similar normalisation issue arises in quantum algorithms for the approximating the Jones polynomial [27, 28] and other #P problems [29].

The only hope for an exponential quantum advantage appears to be the case where $\beta_k^{i,j}$ is sufficiently large that estimating $\beta_k^{i,j}/|S_k^i|$ to constant additive error is a meaningful task. In Ref. [26] it was shown that estimating the normalised quasi-Betti numbers (which accounts for miscounting low-lying but non-zero singular values) of general cohomology groups is DQC1-hard. The hardness of estimating normalised (persistent) Betti numbers of a clique complex, subject to a gap assumption – which is the problem solved by existing quantum algorithms – has not been established [26]. Even if quan-

tum algorithms provide an exponential speedup for this problem, there is currently no evidence that this problem is useful in practice. The largest bounds on β_k^i are $\mathcal{O}(N^k)$ [56] (though this bound is obtained inductively, and so may be loose). For complexes constructed from points positioned randomly in space, the Betti numbers are much smaller; $\beta_k^i \in \mathcal{O}(N)$, on average [57, 58]. For complexes constructed from graphs with random edges, $\beta_k^i \in \mathcal{O}\Big(\binom{N}{k+1}/N^{k/2}\Big)$, on average [59]. Both of these cases lead to superpolynomially vanishing values of $\beta_k^i/|S_k^i|$, when $|S_k^i|$ scales as $\mathcal{O}\Big(\binom{N}{k+1}/Poly(N)\Big)$.

D. Quantum speedup for problems of practical interest

Given the above discussion, it appears prudent to consider the polynomial speedup resulting from applying the quantum algorithm in the regime of practical applications. Let us consider $k = 3$, which is sufficiently low-dimensional to be practically relevant, but is large enough so that it is already challenging to compute classically. Assuming a dense complex with $|S_{k+1}^j| \sim \binom{N}{k+1}$ this results in a scaling of approximately N^{10} classically [20]. Our quantum algorithm scales as $N^{3.5}$, with the additional dependencies on the precision and gap parameters noted in Table I. Subject to these dependencies, this is an almost cubic speedup in N. For larger values of k, subject to the dependencies on the gap parameters, the quantum speedup approaches quintic: $\binom{N}{k+1}^{0.5}$ vs $\binom{N}{k+1}^{\omega}$. Nevertheless, in practice the classical algorithm often scales as $\mathcal{O}\Big(|S_{k+1}^j|\Big)$ [20], or this scaling can be achieved using heuristic sparsification methods [22]. This would reduce the quantum speedup to approximately quadratic, regardless of k. It is an open question as to whether quantum versions of this complex sparsification could be developed, and if they would be beneficial. In addition, the classical algorithm computes $\beta_k^{i,j}$ for all length scales using a single repetition of the algorithm, because it has a record of the boundary operator at all length scales stored in memory. In contrast, the quantum algorithm must be repeated a large number of times to fully analyse the data at all length scales.

VII. Conclusion

We have developed a streamlined quantum algorithm for the calculation of persistent Betti numbers, and carried out an end-to-end complexity analysis. Our algorithm uses the QSVT framework to tackle the problem in a quantum native way, which enables improved complexities compared to existing quantum algorithms. We introduce a new compact mapping from simplices to qubits, that provides an exponential reduction in spatial resources for $k = \mathcal{O}(\text{polylog}(N))$. By compiling all steps of our algorithm down to primitive gates, we can provide a fair comparison to classical algorithms. We see that when the caveats of all known quantum algorithms are taken into account, they can only achieve a polynomial speedup over classical algorithms for the salient problem of estimating low-dimensional persistent Betti numbers to constant additive error – in contrast to previous claims. While it may be possible for quantum algorithms to achieve an exponential speedup for the related problem of estimating normalised persistent Betti numbers (the hardness of this task has not yet been established, but there are currently no known efficient classical algorithms for this problem), there is currently no evidence that this will be of practical use.

The polynomial speedup of the quantum algorithm is contingent upon the as yet unknown dependence of the gap parameters on N. Speedups are only achieved for dense complexes, and vary from quadratic ($k = 1$) to almost quintic (large k) in N (subject to the gap dependencies). In addition, the speedup may be closer to quadratic in practice, compared to typical observed scaling of the standard classical algorithm [20], or heuristic sparsification methods [22]. Moreover, the quantum algorithm must be repeated at a range of length scales, unlike the classical algorithm, introducing a large constant factor overhead – in addition to the overhead in each iteration from e.g. Δ.

The compact mapping introduced in this work provides an exponential space saving over existing quantum and classical methods for the practically relevant regime of $k = \mathcal{O}(\text{polylog}(N))$. Nevertheless, despite the modest number of qubits to store the main register and QROM, we do not expect this to be an application suitable for the early fault-tolerant era [60–63]. For classically challenging instances, we anticipate a large number of non-Clifford gates, which will need to be distilled in parallel. This will increase the footprint of the resulting quantum hardware considerably.

Our algorithm provides an efficient and universal lens through which to investigate topological data analysis on quantum computers. It may provide large polynomial speedup over classical algorithms in some regimes, if favourable gap scaling can be established. Future work should establish its practicality by considering concrete resource estimates for problems of interest. Quantum advantage would be made more likely by identifying scenarios where high-dimensional persistent Betti numbers are of interest, or where calculating the normalised persistent Betti numbers would be meaningful.

VIII. Acknowledgements

We thank Alex Dalzell and Eric Peterson for productive discussions on various aspects of this work, and Fernando Brandão for discussions and support throughout the project. A.G. acknowledges funding from the AWS

Center for Quantum Computing.

[1] Gunnar Carlsson. Topological methods for data modelling. *Nature Reviews Physics*, 2(12):697–708, 2020.

[2] Vin De Silva and Robert Ghrist. Coverage in sensor networks via persistent homology. *Algebraic & Geometric Topology*, 7(1):339–358, 2007.

[3] Michelle Feng and Mason A Porter. Persistent homology of geospatial data: A case study with voting. *SIAM Review*, 63(1):67–99, 2021.

[4] Pratyush Pranav, Herbert Edelsbrunner, Rien Van de Weygaert, Gert Vegter, Michael Kerber, Bernard JT Jones, and Mathijs Wintraecken. The topology of the cosmic web in terms of persistent betti numbers. *Monthly Notices of the Royal Astronomical Society*, 465(4):4281–4310, 2017.

[5] Bastian Rieck, Tristan Yates, Christian Bock, Karsten Borgwardt, Guy Wolf, Nicholas Turk-Browne, and Smita Krishnaswamy. Uncovering the topology of time-varying fmri data using cubical persistence. *Advances in neural information processing systems*, 33:6900–6912, 2020.

[6] Jose A Perea and John Harer. Sliding windows and persistence: An application of topological methods to signal analysis. *Foundations of Computational Mathematics*, 15 (3):799–838, 2015.

[7] Marian Gidea and Yuri Katz. Topological data analysis of financial time series: Landscapes of crashes. *Physica A: Statistical Mechanics and its Applications*, 491:820–834, 2018.

[8] Daniel Leykam and Dimitris G. Angelakis. Topological data analysis and machine learning. 2022. doi: 10.48550/ARXIV.2206.15075. URL https://arxiv.org/abs/2206.15075.

[9] Nicholas Sale, Jeffrey Giansiracusa, and Biagio Lucini. Quantitative analysis of phase transitions in two-dimensional xy models using persistent homology. *Phys. Rev. E*, 105:024121, Feb 2022. doi: 10.1103/PhysRevE.105.024121. URL https://link.aps.org/doi/10.1103/PhysRevE.105.024121.

[10] Andrea Tirelli and Natanael C. Costa. Learning quantum phase transitions through topological data analysis. *Phys. Rev. B*, 104:235146, Dec 2021. doi: 10.1103/PhysRevB.104.235146. URL https://link.aps.org/doi/10.1103/PhysRevB.104.235146.

[11] Felix Hensel, Michael Moor, and Bastian Rieck. A survey of topological machine learning methods. *Frontiers in Artificial Intelligence*, 4:52, 2021.

[12] Niels Neumann and Sterre den Breeijen. Limitations of clustering using quantum persistent homology. *arXiv preprint arXiv:1911.10781*, 2019.

[13] Seth Lloyd, Silvano Garnerone, and Paolo Zanardi. Quantum algorithms for topological and geometric analysis of data. *Nature communications*, 7(1):1–7, 2016.

[14] Seth Lloyd. Quantum algorithms for topological and geometric analysis of data, 2021. URL https://youtu.be/G4t7Pdn9R6c?t=3853. Timecode: 1:04:13.

[15] Sam Gunn and Niels Kornerup. Review of a quantum algorithm for betti numbers. *arXiv preprint arXiv:1906.07673*, 2019.

[16] Shashanka Ubaru, Ismail Yunus Akhalwaya, Mark S Squillante, Kenneth L Clarkson, and Lior Horesh. Quantum topological data analysis with linear depth and exponential speedup. *arXiv preprint arXiv:2108.02811*, 2021.

[17] Ryu Hayakawa. Quantum algorithm for persistent betti numbers and topological data analysis. *arXiv preprint arXiv:2111.00433*, 2021.

[18] Bernardo Ameneyro, Vasileios Maroulas, and George Siopsis. Quantum persistent homology. *arXiv preprint arXiv:2202.12965*, 2022.

[19] Edelsbrunner, Letscher, and Zomorodian. Topological persistence and simplification. *Discrete & Computational Geometry*, 28(4):511–533, Nov 2002. ISSN 1432-0444. doi:10.1007/s00454-002-2885-2. URL https://doi.org/10.1007/s00454-002-2885-2.

[20] Nikola Milosavljević, Dmitriy Morozov, and Primoz Skraba. Zigzag persistent homology in matrix multiplication time. In *Proceedings of the twenty-seventh Annual Symposium on Computational Geometry*, pages 216–225, 2011.

[21] Nikola Milosavljevic, Dmitriy Morozov, and Primoz Skraba. Zigzag Persistent Homology in Matrix Multiplication Time. Research Report RR-7393, INRIA, September 2010. URL https://hal.inria.fr/inria-00520171.

[22] Konstantin Mischaikow and Vidit Nanda. Morse theory for filtrations and efficient computation of persistent homology. *Discrete & Computational Geometry*, 50(2): 330–353, 2013.

[23] Joel Friedman. Computing betti numbers via combinatorial laplacians. *Algorithmica*, 21(4):331–346, 1998.

[24] He-Liang Huang, Xi-Lin Wang, Peter P Rohde, Yi-Han Luo, You-Wei Zhao, Chang Liu, Li Li, Nai-Le Liu, Chao-Yang Lu, and Jian-Wei Pan. Demonstration of topological data analysis on a quantum processor. *Optica*, 5(2): 193–198, 2018.

[25] Casper Gyurik, Chris Cade, and Vedran Dunjko. Towards quantum advantage via topological data analysis. *arXiv preprint arXiv:2005.02607*, 2020.

[26] Chris Cade and P Marcos Crichigno. Complexity of supersymmetric systems and the cohomology problem. *arXiv preprint arXiv:2107.00011*, 2021.

[27] Dorit Aharonov, Vaughan Jones, and Zeph Landau. A polynomial quantum algorithm for approximating the jones polynomial. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 427–436, 2006.

[28] Peter W Shor and Stephen P Jordan. Estimating jones polynomials is a complete problem for one clean qubit. *arXiv preprint arXiv:0707.2831*, 2007.

[29] Magnus Bordewich, MICHAEL Freedman, László Lovász, and D Welsh. Approximate counting and quantum computation. *arXiv preprint arXiv:0908.2122*, 2009.

[30] András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages

193–204, 2019.

[31] Nina Otter, Mason A Porter, Ulrike Tillmann, Peter Grindrod, and Heather A Harrington. A roadmap for the computation of persistent homology. *EPJ Data Science*, 6:1–38, 2017.

[32] Allen Hatcher. *Algebraic topology*. Cambridge University Press, 2005.

[33] Sara Ayman Metwalli, François Le Gall, and Rodney Van Meter. Finding small and large k-clique instances on a quantum computer. *IEEE Transactions on Quantum Engineering*, 1:1–11, 2020. doi: 10.1109/TQE.2020.3045692.

[34] Connor T. Hann, Gideon Lee, S.M. Girvin, and Liang Jiang. Resilience of quantum random access memory to generic noise. *PRX Quantum*, 2:020311, Apr 2021. doi: 10.1103/PRXQuantum.2.020311. URL https://link.aps.org/doi/10.1103/PRXQuantum.2.020311.

[35] Ryan Babbush, Craig Gidney, Dominic W Berry, Nathan Wiebe, Jarrod McClean, Alexandru Paler, Austin Fowler, and Hartmut Neven. Encoding electronic spectra in quantum circuits with linear t complexity. *Physical Review X*, 8(4):041015, 2018.

[36] Ismail Yunus Akhalwaya, Yang-Hui He, Lior Horesh, Vishnu Jejjala, William Kirby, Kugendran Naidoo, and Shashanka Ubaru. Efficient quantum computation of the fermionic boundary operator. *arXiv preprint arXiv:2201.11510*, 2022.

[37] Iordanis Kerenidis and Anupam Prakash. Quantum machine learning with subspace states. *arXiv preprint arXiv:2202.00054*, 2022.

[38] Kianna Wan. Exponentially faster implementations of select (h) for fermionic hamiltonians. *Quantum*, 5:380, 2021.

[39] Lin Lin and Yu Tong. Near-optimal ground state preparation. *Quantum*, 4:372, 2020.

[40] Andreas Bärtschi and Stephan Eidenbenz. Deterministic preparation of dicke states. In *Fundamentals of Computation Theory*, pages 126–139. Springer International Publishing, 2019. doi:10.1007/978-3-030-25027-0˙9. URL https://doi.org/10.1007%2F978-3-030-25027-0_9.

[41] Andreas Bärtschi and Stephan Eidenbenz. Short-depth circuits for dicke state preparation. 2022. doi: 10.48550/ARXIV.2207.09998. URL https://arxiv.org/abs/2207.09998.

[42] András Gilyén. Quantum walk based search methods and algorithmic applications. Master's thesis, Eötvös Loránd University, 2014. URL http://web.cs.elte.hu/blobs/diplomamunkak/msc_mat/2014/gilyen_andras_pal.pdf.

[43] Facundo Mémoli, Zhengchao Wan, and Yusu Wang. Persistent laplacians: Properties, algorithms and implications. *arXiv preprint arXiv:2012.02808*, 2020.

[44] Dmitriy Morozov. Persistence algorithm takes cubic time in worst case. *BioGeometry News, Dept. Comput. Sci., Duke Univ*, 2, 2005.

[45] Chao Chen and Michael Kerber. Persistent homology computation with a twist. In *Proceedings 27th European workshop on computational geometry*, volume 11, pages 197–200, 2011.

[46] Vin De Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson. Dualities in persistent (co) homology. *Inverse Problems*, 27(12):124003, 2011.

[47] Rui Wang, Duc Duy Nguyen, and Guo-Wei Wei. Persistent spectral graph. *International journal for numerical methods in biomedical engineering*, 36(9):e3376, 2020.

[48] Rui Wang, Rundong Zhao, Emily Ribando-Gros, Jiahui Chen, Yiying Tong, and Guo-Wei Wei. Hermes: Persistent spectral graph software. *Foundations of data science (Springfield, Mo.)*, 3(1):67, 2021.

[49] Michael Joswig and Marc E Pfetsch. Computing optimal morse matchings. *SIAM Journal on Discrete Mathematics*, 20(1):11–25, 2006.

[50] Marian Mrozek, Paweł Pilarczyk, and Natalia Żelazna. Homology algorithm based on acyclic subspace. *Computers & Mathematics with Applications*, 55(11):2395–2412, 2008.

[51] Afra Zomorodian. The tidy set: a minimal simplicial set for computing homology of clique complexes. In *Proceedings of the twenty-sixth annual symposium on Computational geometry*, pages 257–266, 2010.

[52] Jonathan Ariel Barmak and Elias Gabriel Minian. Strong homotopy types, nerves and collapses. *Discrete & Computational Geometry*, 47(2):301–328, 2012.

[53] Paweł Dłotko and Hubert Wagner. Simplification of complexes for persistent homology computations. *Homology, Homotopy and Applications*, 16(1):49–63, 2014.

[54] Jean-Daniel Boissonnat, Siddharth Pritam, and Divyansh Pareek. Strong collapse for persistence. *arXiv preprint arXiv:1809.10945*, 2018.

[55] Michał Adamaszek and Juraj Stacho. Complexity of simplicial homology and independence complexes of chordal graphs. *Computational Geometry*, 57:8–18, 2016.

[56] Michael Goff. Extremal betti numbers of vietoris-rips complexes. *Discrete & Computational Geometry*, 46(1), 2011.

[57] Matthew Kahle. Random geometric complexes. *Discrete & Computational Geometry*, 45(3):553–573, jan 2011. doi:10.1007/s00454-010-9319-3. URL https://doi.org/10.1007%2Fs00454-010-9319-3.

[58] Omer Bobrowski and Matthew Kahle. Topology of random geometric complexes: a survey. 2014. doi: 10.48550/ARXIV.1409.4734. URL https://arxiv.org/abs/1409.4734.

[59] Matthew Kahle. Topology of random simplicial complexes: a survey. 2013. doi:10.48550/ARXIV.1301.7165. URL https://arxiv.org/abs/1301.7165.

[60] Earl T Campbell. Early fault-tolerant simulations of the hubbard model. *Quantum Science and Technology*, 7(1): 015007, 2021.

[61] Ruizhe Zhang, Guoming Wang, and Peter Johnson. Computing Ground State Properties with Early Fault-Tolerant Quantum Computers. *Quantum*, 6:761, July 2022. ISSN 2521-327X. doi:10.22331/q-2022-07-11-761. URL https://doi.org/10.22331/q-2022-07-11-761.

[62] Lin Lin and Yu Tong. Heisenberg-limited ground-state energy estimation for early fault-tolerant quantum computers. *PRX Quantum*, 3:010318, Feb 2022. doi: 10.1103/PRXQuantum.3.010318. URL https://link.aps.org/doi/10.1103/PRXQuantum.3.010318.

[63] Kianna Wan, Mario Berta, and Earl T. Campbell. Randomized quantum algorithm for statistical phase estimation. *Phys. Rev. Lett.*, 129:030503, Jul 2022. doi: 10.1103/PhysRevLett.129.030503. URL https://link.aps.org/doi/10.1103/PhysRevLett.129.030503.

[64] Gunnar Carlsson and Mikael Vejdemo-Johansson. *Topological Data Analysis with Applications*. Cambridge University Press, 2021. doi:10.1017/9781108975704.

[65] Chad Topaz. Chad's self-help homology tutorial for the simple(x) minded, 2016. URL https://drive.google.com/file/d/0B3Www1z6Tm8xV3ozTmN5RE94bDg/view?resourcekey=0-tE7y-zXFtV3OWSGmjUebYA. Last accessed 7 April 2022.
[66] David Richeson. Topology 101: The hole truth, January 2021. URL https://dsweb.siam.org/The-Magazine/Article/topological-data-analysis-1. Last accessed 7 April 2022.
[67] Chad Topaz. One applied mathematician's heartwarming story of struggle, triumph, and (ultimately) more struggle, 2016. URL https://dsweb.siam.org/The-Magazine/Article/topological-data-analysis-1. Last accessed 7 April 2022.
[68] Michelle Feng, Abigail Hickok, Yacoub Kureh, Mason Porter, and Chad Topaz. Connecting the dots: Discovering the 'shape' of data. *Front. Young Minds*, 9 (551557), 2021. doi:10.3389/frym.2021.551557. URL https://kids.frontiersin.org/articles/10.3389/frym.2021.551557.
[69] Bastian Rieck. Topological data analysis for machine learning i: Algebraic topology, September 2020. URL https://www.youtube.com/watch?v=gVq_xXnwV-4. Last accessed 7 April 2022.
[70] Michelle Feng. Michelle feng: Topological techniques, February 2021. URL https://www.youtube.com/watch?v=M3TU4NmHDkM. Last accessed 7 April 2022.
[71] Rasmus Hedegaard. Wolfram mathworld: Homologous. URL https://mathworld.wolfram.com/Homologous.html. Last accessed 7 April 2022.
[72] Lek-Heng Lim. Hodge laplacians on graphs. *Siam Review*, 62(3):685–715, 2020.
[73] Timothy E Goldberg. Combinatorial laplacians of simplicial complexes. *Senior Thesis, Bard College*, 2002.
[74] John M. Martyn, Zane M. Rossi, Andrew K. Tan, and Isaac L. Chuang. Grand unification of quantum algorithms. *PRX Quantum*, 2:040203, Dec 2021. doi: 10.1103/PRXQuantum.2.040203. URL https://link.aps.org/doi/10.1103/PRXQuantum.2.040203.
[75] Guang Hao Low, Vadym Kliuchnikov, and Luke Schaeffer. Trading t-gates for dirty qubits in state preparation and unitary synthesis. *arXiv preprint arXiv:1812.00954*, 2018.
[76] Thomas G Draper, Samuel A Kutin, Eric M Rains, and Krysta M Svore. A logarithmic-depth quantum carry-lookahead adder. *arXiv preprint quant-ph/0406142*, 2004.
[77] Shouvanik Chakrabarti, Rajiv Krishnakumar, Guglielmo Mazzola, Nikitas Stamatopoulos, Stefan Woerner, and William J. Zeng. A Threshold for Quantum Advantage in Derivative Pricing. *Quantum*, 5:463, June 2021. ISSN 2521-327X. doi:10.22331/q-2021-06-01-463. URL https://doi.org/10.22331/q-2021-06-01-463.
[78] András Pál Gilyén. *Quantum walk based search methods and algorithmic applications*. PhD thesis, MSc Thesis, Eötvös Loránd University, 2014.
[79] C Gidney. Quantum computing stack exchange, 2020. URL https://quantumcomputing.stackexchange.com/questions/11734/what-is-the-complexity-of-splitting-a-state-into-a-superpo
[80] S Pallister. Quantum computing stack exchange, 2022. URL https://quantumcomputing.stackexchange.com/questions/27864/creating-a-uniform-superposition-of-a-subset-of-basis-stat

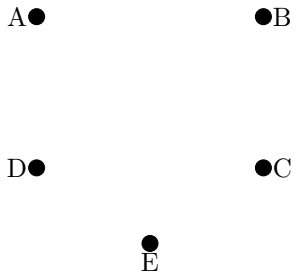
A. Pedagogical introduction to topological data analysis

In this section, we provide a more pedagogical introduction to the mathematical basis and notation used in topological data analysis, to compliment the introduction given in the main text. This area is known as simplicial persistent homology. We will first discuss the case of computing regular Betti numbers, before moving on to persistent Betti numbers. We refer readers to Ref. [64] for a more detailed and rigorous treatment of this area. For pedagogical introductions to algebraic topology and computing persistent homology, we recommend the following review [31], notes [65], popular science articles [66–68], and video lectures [69, 70]. We will only focus on the calculation of persistent homology for point-cloud data, and do not consider extensions to graphs or pixel-based data [31].

1. Computing Betti numbers

Symbol	Meaning
σ_k, τ_k, s_k	Individual k-simplices
S^i	Simplicial complex at scale i defined by length scale μ_i
S^i_k	The set of k-simplices in S^i
$C_k(S^i)$	The k-th chain group of S^i_k (here, over the coefficients $\{0, \pm 1\}$)
∂^i_k	The k-th boundary operator restricted to k-simplices in the complex at scale i
H^i_k	The k-th Homology group (i.e. group of k-holes) at scale i
β^i_k	The k-th Betti number (i.e. number of k-dimensional holes) at scale i
Δ^i_k	The k-th combinatorial Laplacian at scale i

We will assume that our data consists of N points distributed in $\mathbb{R}^d$, sampled from an underlying topological manifold – for example, $N = 5$ in $\mathbb{R}^2$:



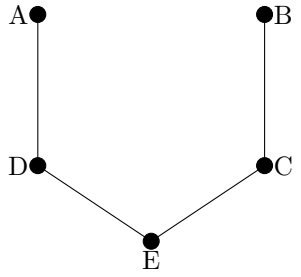
These datapoints will also be referred to as vertices, or 0-simplices. The datapoints have an associated ordering; for example, we choose to order the datapoints alphabetically by their labels. We define a length scale μ, and connect vertices by an edge if they are within μ of each other. For example, at two values of μ, we obtain:

This creates a graph for each value of μ. We use the so-called *clique complex* of the graph, where simplices correspond to the cliques in the graph: a $(k+1)$-clique defines a k-simplex. This way every $(k+1)$ mutually connected vertices induce a k-simplex, which we can think of as the convex hull of the corresponding data points. For example:

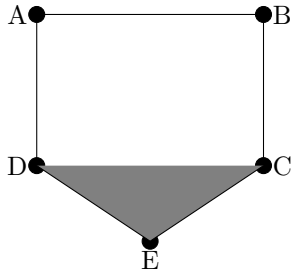
Geometric realisation	Simplex	Examples
	0-Simplex	A, B
	1-Simplex	BC, CD
	2-Simplex	CDE

A k-simplex has cardinality $k+1$ (it contains $k+1$ vertices), and is of dimension k. For two simplices τ, σ, if $\tau \subset \sigma$, then we say τ is a face of σ. For example, the 1-simplex CD is a face of the 2-simplex CDE. Simplices inherit the ordering of the vertices. We ascribe an orientation to the simplices based on the ordering of the vertices. An odd permutation of the vertices changes the orientation of the simplex; for example, $DA = -AD$. A set of simplices (and all of their constituent faces) is referred to as a simplicial complex S. The dimension of a simplicial complex is the highest dimension of a simplex in the complex. We define the number of simplices in the complex as $|S|$, and the number of k-simplices in the complex as $|S_k|$. As the graphs above only contain vertices and edges (0 and 1-simplices), they are 1-dimensional simplicial complexes, known as the 1-skeleton of the complex. We can then add to the complex all k-simplices represented by $(k+1)$-cliques in the graph:

$S^{\mu_1} = \{A, B, C, D, E, AD, BC, DE, CE\}$
$|S_0^{\mu_1}| = 5,\ |S_1^{\mu_1}| = 4,\ |S_2^{\mu_1}| = 0$



$S^{\mu_2} = \{A, B, C, D, E, AD, BC, DE, CE, AB, CD, CDE\}$
$|S_0^{\mu_2}| = 5,\ |S_1^{\mu_2}| = 6,\ |S_2^{\mu_2}| = 1$



This complex is known as a Vietoris-Rips complex, and provides an approximation to the underlying topological space whose homology we wish to compute. We refer the reader to Ref. [31] for discussion of other types of simplicial complexes that can be used to approximate the topological space. The maximum number of k-simplices is $\binom{N}{k+1}$. We represent the k-simplices by orthonormal basis vectors.

Under addition with field coefficients, these vectors form an abelian group. We define the k-th chain group of the complex $C_k(S)$, over field coefficients $\{-1, 0, 1\}$ (i.e. the group $\mathbb{Z}_3$ defined over addition modulo 3).

The group elements are linear combinations of k-simplices with coefficients ± 1, referred to as k-chains. For example:

$$\begin{aligned} AB + CD &\in C_1(S^{\mu_1}) \\ AB + BC + CD - AD &\in C_1(S^{\mu_2}) \end{aligned} \tag{A1}$$

The simplicial complex that we have constructed acts as a scaffold that approximates the underlying topological space of interest. Promoting the simplices in the complex to vectors in a vector space enables the use of linear algebraic tools to perform the computation of simplicial homology. We are trying to identify holes in the complex; a hole is a region of empty space, demarcated by its boundary. As studying these boundaries is crucial for finding holes, we define a boundary operator.

The boundary operator ∂ is a linear map which sends k-chains to their oriented boundaries. The action of the boundary operator on a k-simplex in the complex at the given scale i is

$$\partial[v_0, ..., v_k] = \sum_{l=0}^{k} (-1)^l [v_0, ...\hat{v}_l, ...v_k] \tag{A2}$$

where $v_0...v_k$ are the ordered vertices in the k-simplex, and $\hat{v}_l$ means the vertex is excluded from the simplex. We denote by $\partial_k^i \colon \langle S_k^i \rangle \to \langle S_{k-1}^i \rangle$ the boundary operator on the complex at length scale i restricted to the subspaces $\langle S_k^i \rangle$ and $\langle S_{k-1}^i \rangle$ spanned by the basis vectors corresponding to k and $k-1$ simplices present at scale i.

The boundary operator has the following actions on simplices in our example complex

$$\begin{aligned} \partial_1[AB] &= B - A \\ \partial_2[CDE] &= DE - CE + CD \end{aligned} \tag{A3}$$

A closed cycle of k-simplices in the complex can surround either empty space, or can surround a chain of $(k+1)$-simplices in the complex. The former case corresponds to a hole, while the latter case corresponds to a hole that has been filled-in by $(k+1)$-simplices. An example of the former case in our example complex is the 1-cycle $AB + BC + CD - AD$ (where the minus sign appears due to $DA = -AD$). An example of the latter case in our example complex is the 1-cycle $DE - CE + CD = \partial_2[CDE]$. Linear combinations of these two types of k-cycles are also valid k-cycles. All k-cycles are themselves boundaryless (when applying the boundary operator to a closed cycle of k-simplices, each $(k-1)$-face appears twice, each time with opposite signs). As a result, k-cycles are objects in the kernel of ∂.

To identify the holes in the complex, we consider all possible k-cycles, and remove those that are the k-boundaries of a $(k+1)$-chain in the complex. These k-boundaries are the k-chains in the image of ∂. As both the k-cycles and k-boundaries form subgroups of $C_k(S^i)$, we can remove the filled-in holes using quotient groups (this also eliminates the issue of linear combinations of k-holes and k-boundaries being valid k-cycles - such objects are said to be homologous [71] to the hole in question - and either the hole, or a homologous k-cycle can be chosen to represent the hole). The k-th homology group is defined as the group of k-holes in the complex

$$\mathrm{H}_k^i = \mathrm{Ker}(\partial_k^i) \ / \ \mathrm{Im}(\partial_{k+1}^i) \tag{A4}$$

and the k-th Betti number is the rank of the k-th homology group

$$\begin{aligned}\beta_k^i &= \text{rank}\big(\text{H}_k^i\big) \\ &= \dim\big(\text{Ker}(\partial_k^i)\big) - \dim\big(\text{Im}(\partial_{k+1}^i)\big).\end{aligned} \tag{A5}$$

β_0^i gives the number of connected components in S^i, β_1^i gives the number of holes, β_2^i gives the number of voids, and so on. For our example complex above, $\beta_0(S^{\mu_2}) = 1$, and $\beta_1(S^{\mu_2}) = 1$, so the complex has one connected component, and a single 1-dimensional hole.

As discussed in Ref. [72], it can be undesirable to have a number of equivalent homologous choices for the representative of the hole. The 'harmonic representative' is a chain that is orthogonal to the chains in $\text{Im}(\partial_{k+1}^i)$. The group of harmonic representatives is given by $\text{Ker}(\Delta_k^i)$, where

$$\Delta_k^i = \partial_{k+1}^i(\partial_{k+1}^i)^\dagger + (\partial_k^i)^\dagger\partial_k^i \tag{A6}$$

is known as the k-th combinatorial Laplacian of the simplicial complex, which is a higher-order analogue of the graph Laplacian. It can be shown that [72]

$$\beta_k^i = \dim\big(\text{Ker}(\Delta_k^i)\big). \tag{A7}$$

This formulation of the problem is widely used in a number of quantum algorithms [13, 15, 16, 25]. Note that the operator $(\partial_k^i)^\dagger$ maps $C_{k-1}(S^i) \to C_k(S^i)$. The combinatorial Laplacian is Hermitian, $(k+1)$-sparse (see Ref. [73], Theorem 3.3.4), and its entries are bounded by N [25]. As discussed above, the harmonic representatives are not necessarily in the 'obvious' form. For our example complex, we find that $\text{Ker}(\Delta_1(S^{\mu_2})) = -3(AB + BC + CD - AD) + \partial_2(CDE)$, rather than the obvious choice $(AB + BC + CD - AD)$. Some of the quantum algorithms [13, 15] also consider a 'k-th Dirac operator' that when squared, embeds the k-th combinatorial Laplacian in one of its sub-blocks.

The above procedures can be used for computing the Betti numbers for a fixed length scale μ_i that determines the simplicial complex at scale i. Unfortunately, it is not obvious *a priori* how the length scale should be chosen. Moreover, computing the Betti number for different values of μ does not provide information about the persistence of topological features beyond $k = 0$. As discussed in Ref. [12], the Betti numbers do not uniquely identify topological features. For example, if there was one 1-hole created at scale μ_1, which is destroyed at scale μ_2, at the same time as a different 1-hole is created, then we would find $\beta_1(S^{\mu_1}) = \beta_1(S^{\mu_2}) = 1$, from which we would incorrectly infer that the topological feature created at μ_1 persists at length scale μ_2. In the following subsection we discuss how the methods introduced above can be extended to compute persistent Betti numbers.

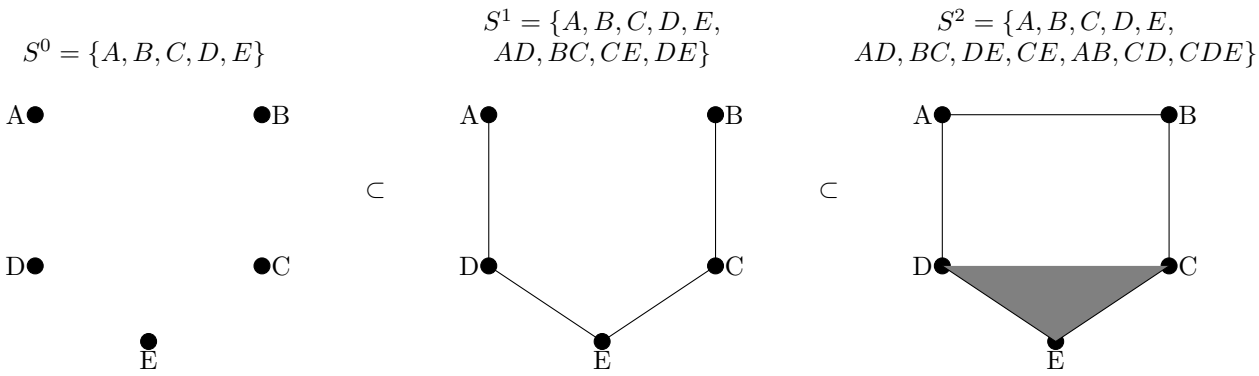
2. Computing persistent Betti numbers

Symbol	Meaning
$\text{H}_k^{i,j}$	The $(j-i)$-th persistent k-th Homology group
$\beta_k^{i,j}$	The $(j-i)$-th persistent k-th Betti number (i.e. number of k-holes present at i still present at j)
$C_{k+1}^{i,j}(S^j)$	The subgroup of $(k+1)$-chains in $C_{k+1}(S^j)$ mapped to k-chains in $C_k(S^i)$ by ∂_{k+1}^j.
$\partial_{k+1}^{i,j}$	The boundary operator ∂_{k+1} restricted to elements of $C_{k+1}^{i,j}(S^j)$.
$\Delta_k^{i,j}$	The $(j-i)$-th persistent k-th combinatorial Laplacian

In order to compute the persistent homology of the complex S, we consider a nested sequence of increasingly dense simplicial complexes – known as a filtered simplicial complex

$$S^0 \subset S^1 \subset ... \subset S^L = S \tag{A8}$$

This filtration is determined by the scale μ_i; as μ_i is increased additional simplices enter the complex. If two simplices enter the complex at the same time, then we consider the lower dimensional simplex to enter first. If the two simplices are of the same dimension, an order can be assigned arbitrarily. For example, we have the following filtration:



As above, we denote the k-th boundary operator restricted to k-simplices in S^i as ∂_k^i. The k-holes in complex S^i may become 'filled in' by new $(k+1)$-simplices present in the later complex S^j. There may also be new k-holes in S^j that were not in S^i. We can count the holes in S^i that are still present in S^j by:

1. Counting all the k-holes and k-boundaries in S^i (i.e. the k-cycles $\mathrm{Ker}(\partial_k^i)$)
2. Removing those k-cycles in S^i that are k-boundaries in S^j (this is given by $\mathrm{Ker}(\partial_k^i) \cap \mathrm{Im}(\partial_{k+1}^j)$ [19])

We do not have to worry about double-counting the k-boundaries in S^i and S^j, as all k-boundaries in S^i are also k-boundaries in S^j, because all simplices in S^i are also in S^j. Formally, we can define the $(j-i)$-th persistent k-th homology group as [19]

$$\mathrm{H}_k^{i,j} := \mathrm{Ker}(\partial_k^i) \;/\; \Big(\mathrm{Ker}(\partial_k^i) \cap \mathrm{Im}(\partial_{k+1}^j)\Big) \tag{A9}$$

The $(j-i)$-th persistent k-th Betti number is the rank of this group,

$$\begin{aligned} \beta_k^{i,j} &= \mathrm{rank}\Big(\mathrm{H}_k^{i,j}\Big) \\ &= \dim\big(\mathrm{Ker}(\partial_k^i)\big) - \dim\Big(\mathrm{Ker}(\partial_k^i) \cap \mathrm{Im}(\partial_{k+1}^j)\Big). \end{aligned} \tag{A10}$$

As discussed in the main text, our quantum algorithm uses this expression for computing $\beta_k^{i,j}$, by estimating the size of these subgroups.

Recently, the task of computing persistent Betti numbers has been reformulated in terms of a persistent combinatorial Laplacian [43, 47]. Although our quantum algorithm does not proceed through this approach, the approach of Ref. [17] does, and it is instructive to compare this approach with our own. One first defines a subgroup $C_{k+1}^{i,j}(S^j)$ containing $(k+1)$-chains in the complex at scale S^j whose images under the boundary operator ∂_{k+1}^j are k-chains at scale S^i. Formally,

$$C_{k+1}^{i,j}(S^j) = \{c \in C_{k+1}(S^j) : \partial_{k+1}^j(c) \in C_k(S^i)\}. \tag{A11}$$

Intuitively, the chains in $C_{k+1}^{i,j}(S^j)$ are mapped to k-cycles in $C_k(S^i)$, which can be divided into the k-boundaries already present in $C_k(S^i)$, and a subset of the k-holes in $C_k(S^i)$. In other words, the additional $(k+1)$-chains in $C_{k+1}^{i,j}(S^j)$ (beyond those already present in $C_{k+1}(S^i)$) have the action of filling-in k-holes in $C_k(S^i)$. Defining $\partial_{k+1}^{i,j}$ as ∂ restricted to the chains in $C_{k+1}^{i,j}(S^j)$ it follows that [43]

$$\mathrm{Im}(\partial_{k+1}^{i,j}) \cong \mathrm{Ker}(\partial_k^i) \cap \mathrm{Im}(\partial_{k+1}^j), \tag{A12}$$

In Refs. [43, 47] the $(j-i)$-th persistent k-th combinatorial Laplacian is defined as

$$\Delta_k^{i,j} = \partial_{k+1}^{i,j} \circ (\partial_{k+1}^{i,j})^\dagger + (\partial_k^i)^\dagger \partial_k^i. \tag{A13}$$

It is shown in Ref. [43] that $\beta_k^{i,j} = \dim\Big(\mathrm{Ker}(\Delta_k^{i,j})\Big)$. We have used $\circ$ in Eq. (E4) to highlight that the expression of $\partial_{k+1}^{i,j}$ in terms of matrices is more complex than expressing ∂_k^i, as discussed in Ref. [43]. We elaborate more on this point in Appendix E. The reason for this complexity is that the restricted boundary operator $\partial_{k+1}^{i,j}$ must be expressed in a basis of the chains in $C_{k+1}^{i,j}(S^j)$, rather than in the basis of S_{k+1}^j. This change of basis adds complexity to both classical [43] and quantum algorithms [17]. A similar phenomenon underpins the additional round of QSVT required in our algorithm to convert $\Pi_{\mathrm{Ker}}\Pi_{\mathrm{Im}}$ to $\Pi_{\mathrm{Ker}\cap\mathrm{Im}}$, discussed in Sec. V D.

B. Proofs of Theorems, Corollaries and Lemmas

As discussed in the main text, we tackle the problem of estimating persistent Betti numbers within the QSVT framework. The QSVT provides quantum circuits that can apply a polynomial function to the singular values of a matrix given by a projected unitary encoding. This is captured by the following Definitions and Lemma, which are less formal restatements of the main results of Ref. [30].

Definition 1. *We say an $n+m$ qubit operator U is an (α, m, ϵ) projected unitary encoding of the n qubit operator A if $||A - \alpha\Pi_L U \Pi_R|| \leq \epsilon$, where Π_L, Π_R are orthogonal projectors.*

Definition 2. *We define a $C_\Pi NOT$ gate as*

$$\mathrm{C}_\Pi\mathrm{NOT} = (I - \Pi) \otimes I + \Pi \otimes X. \tag{B1}$$

For example, if $\Pi = |1\rangle\langle 1|$, $C_\Pi NOT$ is a regular CNOT gate, while if $\Pi = |11\rangle\langle 11|$, $C_\Pi NOT$ is a Toffoli gate.

We are given an (α, m, ϵ) projected unitary encoding U of an operator A, which has singular value decomposition $A/\alpha = W\Sigma V^\dagger$. We are also given the associated $\mathrm{C}_{\Pi_L}\mathrm{NOT}$ and $\mathrm{C}_{\Pi_R}\mathrm{NOT}$ gates, as well as a degree d real polynomial $P(x)$, $|P(x)| \leq 1$ for $x \in [-1, 1]$. The QSVT provides the following Lemma

Lemma 1. *We can implement a projected unitary encoding U_o for an odd polynomial, such that $\Pi_L U_o \Pi_R = P_o(A/\alpha) := W P_o(\Sigma) V^\dagger$ (and equivalently U_e for an even polynomial such that $\Pi_R U_e \Pi_R = P_e(A/\alpha) := V P_e(\Sigma) V^\dagger$). The QSVT circuit makes d calls to U, $U^\dagger$, $2d$ calls to $C_{\Pi_L}NOT$, $C_{\Pi_R}NOT$, and uses $\mathcal{O}(d)$ other single- and two-qubit gates.*

As discussed in the main text, we reduce the problem to that of estimating the normalised rank of a projector. In Theorem 2, we informally show how this problem can be efficiently solved using a quantum algorithm. In the following subsection, we provide and prove a more formal version of this Theorem. Our approach formulates amplitude estimation as a singular value estimation problem, tackled within the QSVT framework. The benefit of this approach is that while it has the same asymptotic complexity as the textbook approach to amplitude estimation considered in Theorem 2, it allows more careful propagation of errors.

1. Theorem 2: Formal version and proof

We will prove a formal version of Theorem 2, which we state here:

Consider the orthogonal projectors $\Pi \subseteq \widetilde{\Pi}$, with $\mathrm{rank}(\Pi) := d_g$, $\mathrm{rank}(\widetilde{\Pi}) := d$. If we treat Π as an observable and $\rho := \widetilde{\Pi}/\mathrm{rank}(\widetilde{\Pi})$ as a quantum state, then the ratio of the ranks $\frac{\mathrm{rank}(\Pi)}{\mathrm{rank}(\widetilde{\Pi})}$ equals Π's expectation value when evaluated on the mixed state ρ. We can speed up the estimation of this expectation value using amplitude estimation if we replace the mixed state ρ by its purification $|\psi\rangle$. For the case of $\widetilde{\Pi} = I$, we chose $|\psi\rangle$ to be the maximally entangled state. We are given access to these objects through projected unitary encodings: we are given a $(1, m_1, \chi_\psi)$-projected unitary encoding V_ψ of $|\psi\rangle\langle\bar{0}|$, and a $(1, m_2, \chi_\pi)$-projected unitary encoding V_Π of Π, and their associated $\mathrm{C}_\Pi\mathrm{NOT}$ gates with errors $\epsilon_L^i, \epsilon_R^i$. We then have the following theorem:

(Normalised projector rank estimation, formal) Given the above ingredients, we can estimate $\sqrt{d_g/d}$ to additive error δ with success probability $\geq 1 - \eta$ using $\tilde{\mathcal{O}}\left(\frac{\log(\delta^{-1})\log(\eta^{-1})}{(1-\chi_\psi)^2(1-2\epsilon)^2}\right)$ incoherent repetitions of a quantum circuit, where $\epsilon := \frac{4}{\delta}\log(\epsilon_f^{-1})\sqrt{2\chi_\psi + \chi_\pi + \sum_i(\epsilon_L^i + \epsilon_R^i)} + \epsilon_f$, with error parameter ϵ_f that contributes to the failure probability. The quantum circuit makes $\mathcal{O}\left(\frac{1}{\delta}\log(\epsilon_f^{-1})\right)$ calls to V_ψ and V_Π (and their inverses) and to the corresponding $C_\Pi NOT$ gates, and uses $\mathcal{O}\left(\frac{1}{\delta}\log(\epsilon_f^{-1})\right)$ additional single- and two-qubit gates.

Proof. As shown in Lemma 2 in Appendix B 3, the purified maximally mixed state $|\psi\rangle$ can be expressed as $|\psi\rangle = \sqrt{\frac{d_g}{d}}|\psi_g\rangle + |\phi^\perp\rangle$, where $|\psi_g\rangle$ is defined by the projector $\Pi = \sum_{i=1}^{d_g}|\pi_i\rangle\langle\pi_i|$. In the 2D subspace $|\psi_g\rangle, |\phi^\perp\rangle$, Π acts as $|\psi_g\rangle\langle\psi_g|$.

Using a product of projected unitary encodings [30], we can combine $V_\Pi \times V_\psi \times V_\psi^\dagger$, to give a $(1, 2m_1 + m_2, 2\chi_\psi + \chi_\pi)$ projected unitary encoding of

$$\Pi|\psi\rangle\langle\bar{0}|\bar{0}\rangle\langle\psi| \tag{B2}$$

$$
\begin{aligned}
&= \Pi \left|\psi\right\rangle \left\langle\psi\right| \\
&= \Pi \left(\sqrt{\frac{d_g}{d}} \left|\psi_g\right\rangle + \left|\phi^{\perp}\right\rangle \right) \left\langle\psi\right| \\
&= \sqrt{\frac{d_g}{d}} \left|\psi_g\right\rangle \left\langle\psi\right|
\end{aligned}
$$

This operator has a lone non-zero singular value of $\sqrt{\frac{d_g}{d}}$, which we seek to estimate to additive error δ.

We can adapt the analysis in Ref. [74], which considered eigenvalue thresholding. Using the QSVT we apply $f_c(x)$ – an even-polynomial approximation of the step-function with transition point c – to the singular-value of the encoded operator. The degree of the polynomial scales as $\mathcal{O}\left(\delta^{-1}\log(\epsilon_f^{-1})\right)$. The dependence on δ ensures that the slope of the transition from 0 to 1 is sufficiently sharp to resolve the transition point to error δ. The error ϵ_f bounds the maximum deviation from 0 or 1 away from the transition point, and determines the success probability of the algorithm. In order to bound the error in the resulting QSVT circuit, we must include the error in both the projected unitary encoding, and the faulty $\mathrm{C}_{\Pi_L^i}\mathrm{NOT}$ and $\mathrm{C}_{\Pi_R^i}\mathrm{NOT}$ gates we are given access to. We can combine the $\mathrm{C}_{\Pi_L^i}\mathrm{NOT}$ and $\mathrm{C}_{\Pi_R^i}\mathrm{NOT}$ gates to form a pair $\mathrm{C}_{\Pi_L}\mathrm{NOT}$ and $\mathrm{C}_{\Pi_R}\mathrm{NOT}$ for the combined projected unitary encodings. The errors in these operators can be bounded by $\sum_i \epsilon_L^i$ and $\sum_i \epsilon_R^i$, respectively. We can then use a slight modification of Lemma 3 (here we have 3 additional qubits, rather than 1, because we have 3 different $\mathrm{C}_{\Pi}\mathrm{NOT}$ gates which we would use to target ancilla qubits $\left|000\right\rangle$. We can then replace the CNOT gate in Lemma 3 by a Toffoli gate controlled on these 3 qubits) to show that the QSVT circuit V_{f_c} is a $(1, 2m_1 + m_2 + 4, \epsilon)$ projected unitary encoding of the operator $f_c\left(\sqrt{\frac{d_g}{d}}\right)\left|\psi_g\right\rangle\left\langle\psi\right|$ with $\epsilon := \frac{4}{\delta}\log(\epsilon_f^{-1})\sqrt{2\chi_\psi + \chi_\pi + \sum_i(\epsilon_L^i + \epsilon_R^i)} + \epsilon_f$.

We use V_ψ to prepare the state $\left|\psi\right\rangle$, which is the sole non-zero singular vector of $f_c\left(\sqrt{\frac{d_g}{d}}\right)\left|\psi_g\right\rangle\left\langle\psi\right|$. We can thus implement

$$
V_{f_c}\left|\psi\right\rangle\left|\bar{0}\right\rangle_a \to f_c\left(\sqrt{\frac{d_g}{d}}\right)\left|\psi_g\right\rangle\left|\bar{0}\right\rangle_a + \left|\phi, \bar{0}_a^{\perp}\right\rangle \tag{B3}
$$

where $\left|\phi, \bar{0}_a^{\perp}\right\rangle$ is a state orthogonal to $\left|\bar{0}\right\rangle_a$. As $f_c(x)$ approximates the step function with transition at point c, by varying the value of c, we can estimate the value of a by monitoring the state of the ancilla register at the end of the algorithm. If the step function were perfect, the ancilla register would be in $\left|\bar{0}\right\rangle_a$ for $c < a$, and $\left|\bar{0}^{\perp}\right\rangle_a$ for $c > a$. The presence of polynomial approximations in both V_ψ and V_{f_c} introduces a deviation from the ideal case outlined above. We utilise the error analysis presented in Ref. [74] for eigenvalue thresholding.

The projected unitary encoding V_ψ introduces an error of χ_ψ. This means that rather than preparing the state $\left|\psi\right\rangle$, we instead prepare

$$
\left|\tilde{\psi}\right\rangle = (1-\chi_\psi)\left|\psi\right\rangle + \sqrt{\chi_\psi(2-\chi_\psi)}\left|\psi^{\perp}\right\rangle. \tag{B4}
$$

The error ϵ for the projected unitary encoding V_{f_c} leads us to prepare the state

$$
V_{f_c}\left|\psi\right\rangle\left|\bar{0}\right\rangle_a \to \begin{cases} \epsilon(1-\chi_\psi)\left|\psi_g\right\rangle\left|\bar{0}\right\rangle_a + \left|\phi', \bar{0}_a^{\perp}\right\rangle, & \text{if } c > a \\ (1-\epsilon)(1-\chi_\psi)\left|\psi_g\right\rangle\left|\bar{0}\right\rangle_a + \left|\phi, \bar{0}_a^{\perp}\right\rangle & \text{if } c < a \end{cases} \tag{B5}
$$

Using Lemma 5 of Ref. [74] with $P(\bar{0}_a|c > a)$ and $P(\bar{0}_a|c < a)$, we can distinguish these two cases with success probability greater than $(1-\theta)$ using

$$
\begin{aligned}
& 6\frac{P(\bar{0}_a|c<a) + P(\bar{0}_a|c>a)}{\left(P(\bar{0}_a|c<a) - P(\bar{0}_a|c>a)\right)^2}\log\left(\frac{1}{\theta}\right) \\
&= 3\left(\frac{1}{(1-\chi_\psi)^2} + \frac{1}{(1-\chi_\psi)^2(1-2\epsilon)^2}\right)\log\left(\frac{1}{\theta}\right) \\
&\in \mathcal{O}\left(\frac{\log\left(\frac{1}{\theta}\right)}{(1-\chi_\psi)^2(1-2\epsilon)^2}\right)
\end{aligned} \tag{B6}
$$

repetitions.

We can use the above procedure to determine if $\sqrt{\frac{d_g}{d}}$ is greater than or less than the chosen value c. We use the incoherent binary search discussed in Ref. [39] to estimate the value of $\sqrt{\frac{d_g}{d}}$. That work considered learning the ground state energy. The maximum bound on the energy eigenvalues was $\pm\alpha$. To determine the ground state energy to precision h with success probability $\geq (1-\eta)$ (given a success probability of each individual step of binary search of $\geq (1-\theta)$), this approach required $\mathcal{O}\big(\log\big(\frac{\alpha}{h}\big)\big)$ steps, each of which requires $\mathcal{O}\Big(\log\Big(\frac{2\log(4\alpha/h)}{\eta}\Big)\Big)$ repetitions of the quantum algorithm. Applying this to our case, where the variable $\sqrt{\frac{d_g}{d}} \in [0,1] \to \alpha = 1$, and $h = \delta$, we require

$$\begin{aligned} &\mathcal{O}\left(\frac{\log\big(\frac{1}{\delta}\big)\log\Big(\frac{\log(1/\delta)}{\eta}\Big)}{(1-\chi_\psi)^2(1-2\epsilon)^2}\right) \\ \in\ &\tilde{\mathcal{O}}\left(\frac{\log\big(\frac{1}{\delta}\big)\log\Big(\frac{1}{\eta}\Big)}{(1-\chi_\psi)^2(1-2\epsilon)^2}\right) \end{aligned} \tag{B7}$$

incoherent repetitions of the quantum circuit to learn $\sqrt{\frac{d_g}{d}}$ to additive error δ with success probability $\geq (1-\eta)$.

The quantum part of the algorithm has two components; using V_ψ to prepare $|\psi\rangle$ (which makes 1 call to V_ψ) and applying V_{f_c}. The circuit V_{f_c} makes $\mathcal{O}\Big(\frac{1}{\delta}\log(\epsilon_f^{-1})\Big)$ calls to the projected unitary encoding of $\sqrt{\frac{d_g}{d}}\,|\psi_g\rangle\langle\psi|$ (and its inverse), as well as the corresponding $\mathrm{C}_{\Pi^i_{L/R}}\mathrm{NOT}$ gates. The projected unitary encoding used one call to V_ψ, one call to $V_\psi^\dagger$, and one call to V_Π, and a constant number of additional single- and two-qubit gates. Hence the circuit requires

$$\mathcal{O}\left(\frac{1}{\delta}\log(\epsilon_f^{-1})\right) \tag{B8}$$

calls to V_ψ and V_Π (and their inverses), and to the corresponding $\mathrm{C}_{\Pi^i_{L/R}}\mathrm{NOT}$ gates, as well as $\mathcal{O}\Big(\frac{1}{\delta}\log(\epsilon_f^{-1})\Big)$ additional single- and two-qubit gates. □

2. Corollary 2.1: Applying Theorem 2 to estimate persistent Betti numbers

As discussed in the main text, we can estimate persistent Betti numbers using the following expression

$$\beta_k^{i,j} = \dim\big(\mathrm{Ker}(\partial_k^i)\big) - \dim\Big(\mathrm{Ker}(\partial_k^i)\cap\mathrm{Im}(\partial_{k+1}^j)\Big). \tag{B9}$$

We estimate the dimensions of projectors Π_{Ker} and $\Pi_{\mathrm{Ker}\cap\mathrm{Im}}$, where Π_{Ker} projects onto $\mathrm{Ker}(\partial_k^i)$ and Π_{Im} projects onto $\mathrm{Im}(\partial_{k+1}^j)$ (and thus $\Pi_{\mathrm{Ker}\cap\mathrm{Im}}$ projects onto $\mathrm{Ker}(\partial_k^i)\cap\mathrm{Im}(\partial_{k+1}^j)$). We can obtain these quantities, normalised by the number of k-simplices in the complex, using the quantum algorithm of Theorem 2. Hence, by rescaling the estimation error, we obtain the following corollary of (the formal version of) Theorem 2:

Corollary 2.1. *To estimate $\beta_k^{i,j}$ to additive error Δ with success probability $\geq 1-\eta$, we solve three instances of normalised projector rank estimation. Each instance x uses $\tilde{\mathcal{O}}\Big(\frac{\log(\delta_x^{-1})\log(\eta^{-1})}{(1-\chi_\psi)^2(1-2\epsilon_x)^2}\Big)$ incoherent repetitions of a quantum circuit, where $\epsilon_x := \frac{4}{\delta_x}\log(\epsilon_f^{-1})\sqrt{2\chi_\psi+\chi_\pi+\sum_i(\epsilon_L^i+\epsilon_R^i)} + \epsilon_f$. The quantum circuit for each instance makes $\mathcal{O}\Big(\frac{1}{\delta_x}\log(\epsilon_f^{-1})\Big)$ calls to V_{ψ_x} and V_{Π_x} (and their inverses), as well as the corresponding faulty $C_\Pi NOT$ gates that induce the projected unitary encodings. The algorithm also uses $\mathcal{O}\Big(\frac{1}{\delta_x}\log(\epsilon_f^{-1})\Big)$ additional single- and two-qubit gates. The error parameters δ_x and quantities $|\psi_x\rangle, \Pi_x$ are shown in Table VIII.*

Proof. To prove this corollary, we define the following three random variables, which each instance of normalised projector rank estimation will estimate:

$$W := \sqrt{\frac{d}{D}} \tag{B10}$$

Instance x	1	2	3
Estimates	$\sqrt{\lvert S_k^i\rvert/\binom{N}{k+1}}$	$\sqrt{\dim(\mathrm{Ker}(\partial_k^i))/\lvert S_k^i\rvert}$	$\sqrt{\dim\big(\mathrm{Ker}(\partial_k^i)\cap\mathrm{Im}(\partial_{k+1}^j)\big)/\lvert S_k^i\rvert}$
δ_x	$\frac{\Delta}{2\sqrt{3}\beta_k^{i,j}}\sqrt{\frac{\lvert S_k^i\rvert}{\binom{N}{k+1}}}$	$\frac{\Delta}{2\sqrt{3\lvert S_k^i\rvert\dim\left(\mathrm{Ker}(\partial_k^i)\right)}}$	$\frac{\Delta}{2\sqrt{3\lvert S_k^i\rvert\dim\left(\mathrm{Ker}(\partial_k^i)\cap\mathrm{Im}(\partial_{k+1}^j)\right)}}$
Π	$\Pi_{m_k^i}$: Projects onto k-simplices in complex	Π_{Ker}: Projects onto $\mathrm{Ker}(\partial_k^i)$	$\Pi_{\mathrm{Ker}\cap\mathrm{Im}}$: Projects onto $\mathrm{Ker}(\partial_k^i)\cap\mathrm{Im}(\partial_{k+1}^j)$
$\lvert\psi\rangle$	$\lvert\psi_s\rangle$: Uniform superposition of all possible k-simplices	$\lvert\psi_m\rangle$: Purification of maximally mixed state over all k-simplices in complex at scale i	

TABLE VIII. Details of the error parameters and projected unitary encoded operators for each of the instances of normalised projector rank estimation used to estimate the persistent Betti number to additive error Δ using the algorithm of Theorem 2.

$$X := \sqrt{\frac{\dim\big(\mathrm{Ker}(\partial_k^i)\big)}{d}} \tag{B11}$$

$$Y := \sqrt{\frac{\dim\Big(\mathrm{Ker}(\partial_k^i)\cap\mathrm{Im}(\partial_{k+1}^j)\Big)}{d}} \tag{B12}$$

where $d := |S_k^i|$, $D := \binom{N}{k+1}$. As discussed in the main text, we can compute the persistent Betti numbers using

$$\beta_k^{i,j} = \dim\big(\mathrm{Ker}(\partial_k^i)\big) - \dim\Big(\mathrm{Ker}(\partial_k^i)\cap\mathrm{Im}(\partial_{k+1}^j)\Big). \tag{B13}$$

This gives an expression for the estimated persistent Betti number $\bar{\beta}_k^{i,j} := Z$ in terms of the random variables W, X, Y:

$$Z = W^2 D(X^2 - Y^2). \tag{B14}$$

We define the estimation errors as $\alpha_z, \alpha_w, \alpha_x, \alpha_y$, respectively. The error in Z is given by

$$\begin{aligned}\alpha_z^2 &= \sum_{j\in W,X,Y}\left(\frac{\partial Z}{\partial J}\right)^2\alpha_j^2 \\ &= 4W^2D^2(X^2-Y^2)^2\alpha_w^2 + 4W^4D^2X^2\alpha_x^2 + 4W^4D^2Y^2\alpha_y^2\end{aligned} \tag{B15}$$

To ensure $\alpha_z = \Delta$, we set each term equal to $\Delta^2/3$. This yields

$$\alpha_w = \frac{\Delta}{2\sqrt{3}Z}\sqrt{\frac{d}{D}} \tag{B16}$$

$$\alpha_x = \frac{\Delta}{2\sqrt{3d\dim\big(\mathrm{Ker}(\partial_k^i)\big)}} \tag{B17}$$

$$\alpha_y = \frac{\Delta}{2\sqrt{3d\dim\Big(\mathrm{Ker}(\partial_k^i)\cap\mathrm{Im}(\partial_{k+1}^j)\Big)}} \tag{B18}$$

Substituting for d, D yields the required values of δ specified in the corollary.

It remains to show that the projected unitary encodings specified encode the required random variables. For W, V_{ψ_s} is a projected unitary encoding of $|\psi_s\rangle\langle\bar{0}|$, where

$$\begin{aligned}|\psi_s\rangle &= \frac{1}{\sqrt{D}}\sum_{s_k}|s_k\rangle \\ &= \frac{1}{\sqrt{D}}\sum_{s_k\in S_k^i}|s_k\rangle + \frac{1}{\sqrt{D}}\sum_{s_k'\notin S_k^i}|s_k'\rangle\end{aligned} \tag{B19}$$

We can express the projector $\Pi_{m_k^i}$ as $\Pi_{m_k^i} = \sum_{s_k \in S_k^i} |s_k\rangle\langle s_k|$. Hence, taking a product of the projected unitary encodings $V_{\Pi_{m_k^i}} \times V_{\psi_s} \times V_{\psi_s}^\dagger$ gives a projected unitary encoding of

$$\begin{aligned}
&\left(\sum_{\sigma_k \in S_k^i} |\sigma_k\rangle\langle\sigma_k|\right)\left(\frac{1}{\sqrt{D}}\sum_{s_k \in S_k^i}|s_k\rangle + \frac{1}{\sqrt{D}}\sum_{s_k' \notin S_k^i}|s_k'\rangle\right)\langle\psi| \\
&= \frac{1}{\sqrt{D}}\sum_{s_k \in S_k^i}|s_k\rangle\langle\psi| \\
&= \sqrt{\frac{d}{D}}\,|\bar{\psi}_{\text{Mem}}\rangle\langle\psi|
\end{aligned} \tag{B20}$$

where we have defined $|\bar{\psi}_{\text{Mem}}\rangle := \frac{1}{\sqrt{d}}\sum_{s_k \in S_k^i}|s_k\rangle$. This encodes W, as required. We see that the purification was not required in this example because we can prepare the uniform superposition in the basis in which $\Pi_{m_k^i}$ is diagonal.

For X and Y, we do require a purification, as Π_{Ker} and $\Pi_{\text{Ker}\cap\text{Im}}$ are not diagonal in the basis of k-simplices in the complex. We focus on Y, as the proof for X follows the same steps. Using Lemma 2 in Appendix B 3, we can express

$$|\psi_m\rangle := \frac{1}{\sqrt{|S_k^i|}}\sum_{s_k \in S_k^i}|s_k\rangle|s_k\rangle \tag{B21}$$

as

$$|\psi_m\rangle = \sqrt{\frac{\dim\Big(\text{Ker}(\partial_k^i)\cap\text{Im}(\partial_{k+1}^j)\Big)}{|S_k^i|}}\,|\psi_g\rangle + |\phi^\perp\rangle \tag{B22}$$

where $|\psi_g\rangle$ is defined by $\Pi_{\text{Ker}\cap\text{Im}}$, and ensures that $\Pi_{\text{Ker}\cap\text{Im}}$ acts as $|\psi_g\rangle\langle\psi_g|$ in the 2D subspace spanned by $|\psi_g\rangle, |\phi^\perp\rangle$. We can then implement a projected unitary encoding of

$$\begin{aligned}
&\Pi_{\text{Ker}\cap\text{Im}}\,|\psi_m\rangle\langle\psi_m| \\
&= \sqrt{\frac{\dim\Big(\text{Ker}(\partial_k^i)\cap\text{Im}(\partial_{k+1}^j)\Big)}{|S_k^i|}}\,|\psi_g\rangle\langle\psi_m|
\end{aligned} \tag{B23}$$

which encodes the desired random variable Y. This shows that the chosen projected unitary encodings are able to encode the necessary random variables. The number of calls required to these projected unitary encodings follow from Theorem 2 and the above error bounds. □

3. Technical Lemmas

Lemma 2. *Given a projector $\Pi = \sum_{i=1}^{d_g}|\pi_i\rangle\langle\pi_i|$, define $|\psi\rangle$ as a purified maximally mixed state over a d dimensional basis (with $d \geq d_g$), such that Π has full support on $|\psi\rangle$. We can express $|\psi\rangle = \sqrt{\frac{d_g}{d}}\,|\psi_g\rangle + |\phi^\perp\rangle$, where $|\psi_g\rangle, |\phi^\perp\rangle$ are defined by their action under Π, such that Π acts as $|\psi_g\rangle\langle\psi_g|$ in the 2D subspace spanned by $|\psi_g\rangle, |\phi^\perp\rangle$.*

Proof. The state $|\psi\rangle$ is defined as $\frac{1}{\sqrt{d}}\sum_x |x\rangle|x\rangle$ for a set of states $\{|x\rangle\}$ such that $d \geq d_g$, and Π has full support on $|\psi\rangle$. As a result, the states $|x\rangle$ can be re-expressed in the basis of $\{|\pi_i\rangle\}$ that includes both the states in the image of Π, and states in its kernel; $|x\rangle = \sum_{\alpha=1}^d U_{\alpha x}|\pi_\alpha\rangle$. We can then express $|\psi\rangle$ in the basis $\{|\pi_i\rangle\}$ as

$$\begin{aligned}
|\psi\rangle &= \frac{1}{\sqrt{d}}\sum_{x=1}^d |x\rangle|x\rangle \\
&= \frac{1}{\sqrt{d}}\sum_{x=1}^d\sum_{\alpha,\beta=1}^d U_{\alpha x}|\pi_\alpha\rangle U_{\beta x}|\pi_\beta\rangle
\end{aligned} \tag{B24}$$

$$= \frac{1}{\sqrt{d}} \left(\sum_{x=1}^{d} \sum_{\alpha=1}^{d_g} \sum_{\beta=1}^{d} U_{\alpha x} |\pi_\alpha\rangle U_{\beta x} |\pi_\beta\rangle + \sum_{x=1}^{d} \sum_{\alpha=d_g+1}^{d} \sum_{\beta=1}^{d} U_{\alpha x} |\pi_\alpha\rangle U_{\beta x} |\pi_\beta\rangle \right)$$
$$:= \frac{1}{\sqrt{d}} |\tilde{\psi}_g\rangle + |\phi^\perp\rangle$$

where $|\tilde{\psi}_g\rangle$ is the unnormalized state of interest. The normalisation is given by

$$\begin{aligned}
\langle\tilde{\psi}_g|\tilde{\psi}_g\rangle &= \sum_{\alpha,\alpha'=1}^{d_g} \sum_{\beta,\beta'=1}^{d} \sum_{x,x'=1}^{d} U_{\alpha x} U^*_{\alpha' x'} \langle\pi_{\alpha'}|\pi_\alpha\rangle U_{\beta x} U^*_{\beta' x'} \langle\pi_{\beta'}|\pi_\beta\rangle \\
&= \sum_{\alpha,\alpha'=1}^{d_g} \sum_{x,x'=1}^{d} U_{\alpha x} U^*_{\alpha x'} [UU^\dagger]_{x x'} \\
&= \sum_{\alpha,\alpha'=1}^{d_g} \sum_{x=1}^{d} U_{\alpha x} U^*_{\alpha x} \\
&= \sum_{\alpha,\alpha'=1}^{d_g} [UU^\dagger]_{\alpha,\alpha} \\
&= d_g
\end{aligned} \tag{B25}$$

As a result, we can define the normalised state $|\psi_g\rangle = \frac{1}{\sqrt{d_g}} |\tilde{\psi}_g\rangle$, and re-express $|\psi_A\rangle$ as

$$|\psi_A\rangle = \sqrt{\frac{d_g}{d}} |\psi_g\rangle + |\phi^\perp\rangle \tag{B26}$$

as required.

It remains to be shown that $\Pi := \sum_{i=1}^{d_g} |\pi_i\rangle\langle\pi_i|$ acts as $|\psi_g\rangle\langle\psi_g|$ in the 2D subspace $|\psi_g\rangle, |\phi^\perp\rangle$. This can be verified from the definitions of $|\psi_g\rangle, |\phi^\perp\rangle$ above by applying Π to the first register in the purification. □

Lemma 3. *We are given a $(1, m, \epsilon)$ projected unitary encoding U such that $\Pi_L U \Pi_R = A$. We are also given gates $C_{\Pi_R}NOT$ and $C_{\Pi_L}NOT$ with associated errors ϵ_R and ϵ_L. A QSVT circuit V that uses d applications of U, $U^\dagger$, and $2d$ applications of each of $C_{\Pi_R}NOT$ and $C_{\Pi_L}NOT$ is a $(1, m+2, 4d\sqrt{\epsilon_L + \epsilon_R + \epsilon})$ projected unitary encoding of $P(A)$ (given a polynomial such that $|P(x)| \leq 1$ for $x \in [-1, 1]$).*

Proof. The QSVT circuit uses an alternating sequence of calls to U, $\mathrm{C}_{\Pi_L}\mathrm{NOT}$, $U^\dagger$, $\mathrm{C}_{\Pi_R}\mathrm{NOT}$, as shown in Fig. 2.

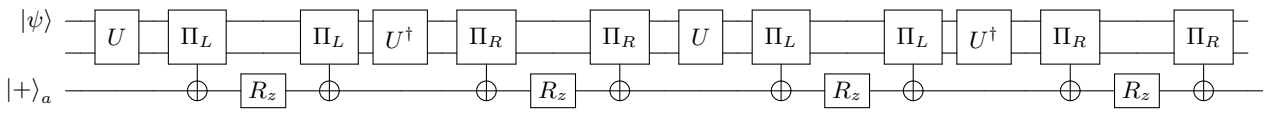


FIG. 2. A standard QSVT circuit template. See Ref. [30] for additional details.

By introducing an additional ancilla qubit, we can rewrite each block of this circuit as shown in Fig. 3.

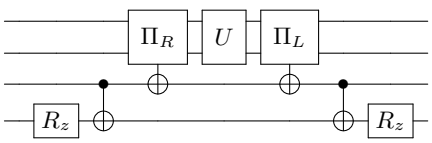


FIG. 3. Recompiling a block of the above QSVT circuit using an additional ancilla qubit.

We can now group together $\mathrm{C}_{\Pi_L}\mathrm{NOT} \cdot U \cdot \mathrm{C}_{\Pi_R}\mathrm{NOT} = \bar{U}$ as a new projected unitary encoding, induced by CNOT gates from the new ancilla qubit to the QSVT ancilla. We can now bound the error in the new projected unitary encoding $\bar{U}$ (here $\Pi'_L = \Pi'_R = |1\rangle_a \langle 1|_a$)

$$\begin{aligned} &||\Pi'_L \bar{U} \Pi'_R - A|| \\ &\leq ||\Pi'_L \bar{U} \Pi'_R - \Pi'_L \bar{U}^0 \Pi'_R|| + ||\Pi'_L \bar{U}^0 \Pi'_R - A|| \end{aligned} \tag{B27}$$

where $\bar{U}^0$ is the new projected unitary encoding if the $\mathrm{C}_\Pi\mathrm{NOT}$ gates are noiseless. Continuing from above

$$\begin{aligned} &\leq ||\bar{U} - \bar{U}^0|| + \epsilon \\ &= ||\mathrm{C}_{\Pi_L}\mathrm{NOT} \cdot U \cdot \mathrm{C}_{\Pi_R}\mathrm{NOT} - \mathrm{C}_{\Pi_L}\mathrm{NOT}^0 \cdot U \cdot \mathrm{C}_{\Pi_R}\mathrm{NOT}^0|| + \epsilon \\ &\leq ||\mathrm{C}_{\Pi_L}\mathrm{NOT} \cdot U \cdot \mathrm{C}_{\Pi_R}\mathrm{NOT} - \mathrm{C}_{\Pi_L}\mathrm{NOT}^0 \cdot U \cdot \mathrm{C}_{\Pi_R}\mathrm{NOT}|| + ||\mathrm{C}_{\Pi_L}\mathrm{NOT}^0 \cdot U \cdot \mathrm{C}_{\Pi_R}\mathrm{NOT} - \mathrm{C}_{\Pi_L}\mathrm{NOT}^0 \cdot U \cdot \mathrm{C}_{\Pi_R}\mathrm{NOT}^0|| + \epsilon \\ &\leq \epsilon_L + \epsilon_R + \epsilon \end{aligned} \tag{B28}$$

We have thus converted the $(1, m, \epsilon)$ projected unitary encoding U with noisy $\mathrm{C}_\Pi\mathrm{NOT}$ gates into a $(1, m+1, \epsilon_L+\epsilon_R+\epsilon)$ projected unitary encoding $\bar{U}$ with noiseless $\mathrm{C}_{\Pi'}\mathrm{NOT}$ gates.

We can then apply the robustness of QSVT (Ref. [30], Lemma 22) to recover the stated result. □

C. Resource costs

In this section we determine the resource costs of the individual building blocks of our algorithm.

1. Membership oracle construction

As discussed in the main text, the algorithm makes regular calls to a membership oracle $O^i_{m_k}$ which determines if a simplex is present in the complex at scale i (or not) based on the positions of the vertices $\{\vec{r}_\alpha\}$, and the length scale μ_i considered. The membership oracle acts as

$$O^i_{m_k} |s_k\rangle |a\rangle = |s_k\rangle |a \oplus m(s_k)\rangle \tag{C1}$$

where we have defined the membership function

$$m(s_k) = \begin{cases} 1 & \text{if } s_k \in S^i_k \\ 0 & \text{if } s_k \notin S^i_k \end{cases}. \tag{C2}$$

We discuss below how the membership oracle is implemented for both the compact and direct mappings.

a. *Compact mapping*

The membership oracle in the compact encoding checks that the vertices present in the simplex have the correct ordering, and that the distances between these vertices are less than the length scale μ_i. To implement the latter step, we will load the positions of the vertices from a quantum memory. The quantum memory can be implemented as either quantum read-only memory (QROM) or quantum random-access memory (QRAM). A quantum memory implements the following transformation

$$\frac{1}{\sqrt{D}} \sum_{t=0}^{D-1} |t\rangle |0\rangle^{\otimes b} \to \frac{1}{\sqrt{D}} \sum_{t=0}^{D-1} |t\rangle |d_t\rangle \tag{C3}$$

where d_t are b-bit data values to be loaded. Loading D pieces of data with a QROM requires $\mathcal{O}(D \log(b))$ time (with only $\mathcal{O}(D)$ non-Clifford gates [35]), and $\mathcal{O}(\log(D))$ ancilla qubits. To load D pieces of b-bit data with a QRAM requires $\mathcal{O}(\log(D))$ time, and $\mathcal{O}(Db)$ ancilla qubits. It is also possible to consider intermediate space-time tradeoffs [34].

The quantum memory is used to load the position of the vertices in the compactly encoded simplex. As a result, for a set of points in $\mathbb{R}^d$ with each coordinate stored with b_d bits, each piece of loaded data requires db_d

bits. We choose b_d large enough to keep over/underflow errors introduced when calculating the squared distance between points sufficiently small. The distance between the datapoints is then calculated coherently, and used to mark any edges in the simplex that are above the length scale μ_i. If such an edge exists, the simplex does not belong in the complex. We can use this marking of edges to mark bad simplex states (via amplitude amplification).

To implement the above process, we introduce two ancilla registers, each with $\log(k+1)$ qubits, and place them in an equally weighted superposition

$$\frac{1}{k+1}\sum_{i,j=0}^{k}|i\rangle\,|j\rangle\,|V_0\rangle\,...\,|V_k\rangle\,. \tag{C4}$$

We then use quantum memory to load the coordinates of the i-th and j-th vertices in the simplex. We can use a controlled swap network [75] requiring $\mathcal{O}(\log(k+1))$ depth to swap the i-th vertex to the first position in the simplex, and then use quantum memory to load the position of the simplex stored in the first vertex position. We can then swap the i-th vertex back to its original position, and repeat the process for the j-th vertex. The resulting state is

$$\frac{1}{k+1}\sum_{i,j=0}^{k}|i\rangle\,|j\rangle\,|V_1\rangle\,...\,|V_{k+1}\rangle\,|\vec{r}_i\rangle\,|\vec{r}_j\rangle \tag{C5}$$

We coherently evaluate the squared distance $d_{ij}^2 = |\vec{r}_i - \vec{r}_j|^2$ between the positions

$$\frac{1}{k+1}\sum_{i,j=0}^{k}|i\rangle\,|j\rangle\,|V_1\rangle\,...\,|V_{k+1}\rangle\,|\vec{r}_i\rangle\,|\vec{r}_j\rangle\,|d_{ij}^2\rangle \tag{C6}$$

and set a flag based on whether this value is below or above the squared value of the length scale μ_i

$$\frac{1}{k+1}\sum_{i,j=0}^{k}|i\rangle\,|j\rangle\,|V_1\rangle\,...\,|V_{k+1}\rangle\,|\vec{r}_i\rangle\,|\vec{r}_j\rangle\,|d_{ij}^2\rangle\,|d_{ij}^2 \overset{?}{\leq} \mu_i^2\rangle \tag{C7}$$

such that $d_{ij}^2 \overset{?}{\leq} \mu_i^2$ evaluates to 0 if true, and 1 if false. For a given simplex, this marks any edges that are present but should not be – which in turn signifies that the simplex should not be in the complex. We use fixed-point amplitude amplification to map the equal superposition initial state to the state (we do not keep track of the normalisation factor here)

$$\sum_{i<j;E_{ij}\notin\mathcal{E}}|i\rangle\,|j\rangle\,|V_1\rangle\,...\,|V_{k+1}\rangle\,|1\rangle \tag{C8}$$

(assuming that this state exists) where $\mathcal{E}$ denotes the edges present in the complex, and E_{ij} is the edge between vertices V_i, V_j. The probability of finding an incorrect edge is $(k+1)^2/k$ in the worst case of one incorrect vertex (which leads to k incorrect edges in a k-simplex). A typical analysis of amplitude amplification considers boosting the success probability to $\geq 1-\epsilon_m$ using $\mathcal{O}(\sqrt{k}\log(\epsilon_m^{-1}))$ rounds of fixed-point amplitude amplification. Because this membership oracle subroutine will be called a large number of times in the algorithm, we instead consider amplitude amplification to succeed with probability 1, but to introduce an error, which we bound by the operator norm error in the amplitude amplification unitary, which is $\sqrt{\epsilon_m}$ for $\mathcal{O}(\sqrt{k}\log(\epsilon_m^{-1}))$ rounds of fixed-point amplitude amplification. We can copy the flag, and apply the inverse of the amplitude amplification circuit to prepare the state

$$|V_0\rangle\,..\,|V_k\rangle\,|m(V_0...V_k)\rangle \tag{C9}$$

which thus implements the membership oracle with an error of $2\sqrt{\epsilon_m}$. Each round of amplitude amplification uses four calls to the quantum memory, and two evaluations of the squared distance, and two comparison operations.

Using a QRAM structure, the memory load can be performed with $\mathcal{O}(N)$ gates, in $\mathcal{O}(\log(N))$ depth, and $\mathcal{O}(Ndb_d)$ ancilla qubits. If instead we opted to use a QROM structure, the memory load would require $4N$ T gates, and $\log(N)$ ancilla qubits [35]. Using the primitives shown in Table IX, we can calculate the squared distance using $\mathcal{O}(\log(d)\log(b_d)+b_d)$ non-Clifford depth, using $\mathcal{O}(db_d^2)$ ancilla qubits, and compare it to the length scale using $\mathcal{O}(\log(b_d))$ non-Clifford depth and $\mathcal{O}(b_d)$ ancilla qubits.

Primitive	Non-Clifford depth	Ancilla qubits
Addition/Subtraction (two a-bit numbers) [76]	$\log(a)$	a
Squaring (a-bit number) [77]	a	a^2
Comparison (two a-bit numbers)	$\log(a)$	a

TABLE IX. Costs of primitives used for coherently calculating the squared distances between vertices.

To verify that the vertices have the correct ordering, we can introduce $\mathcal{O}(k)$ ancilla qubits. We use these to perform comparisons between V_i and V_{i+1} for i even. We perform all $\lceil (k+1)/2 \rceil$ comparisons in parallel. We then do the same for V_i and V_{i+1} for i odd. This requires a circuit depth of $\mathcal{O}(\log\log(N))$, and $\mathcal{O}(k\log(N))$ ancilla qubits. We can use a single $\mathcal{O}(k)$-controlled Toffoli gate to flag if the vertices have the correct ordering. This requires a circuit depth of $\mathcal{O}(k)$. As the order checking can be performed after the distance checking, we can use the same ancilla qubits for both protocols. The overall complexity for the membership oracle, factoring in both steps, is shown in Table X.

Memory model	Non-Clifford depth	Ancilla qubits
QROM	$\sqrt{k}\log(\epsilon_m^{-1})(N+\log(d)\log(b_d)+b_d)+k$	$\mathrm{Max}(\log(N)+db_d^2, k\log(N))$
QRAM	$\sqrt{k}\log(\epsilon_m^{-1})(\log(N)+\log(d)\log(b_d)+b_d)+k$	$\mathrm{Max}(Ndb_d+db_d^2, k\log(N))$

TABLE X. Asymptotic complexities of implementing the compact mapped membership oracle with error $2\sqrt{\epsilon_m}$, using either QROM or QRAM to perform loading of the positions of the vertices.

b. Direct mapping

Our implementation of the membership oracle for the direct mapping follows the approach in Ref. [33] for finding cliques in graphs. We classically store a list of edges in the complex, and then iterate through this list, checking if each edge is present in the given k-simplex. This can be checked using a Toffoli gate controlled on the vertices of the edge, targeting an ancilla qubit. We then increment a counter, controlled on the value of the ancilla qubit, and then uncompute the ancilla qubit value. After all edges have been sequentially checked, we verify that the counter register contains the correct number of edges for a k-simplex ($0.5(k+1)k$), and use this to flag if the simplex is in the complex or not, and uncompute the counter register. In the worst case there are $0.5N(N-1)$ edges in the complex. By introducing $\mathcal{O}(N)$ ancilla qubits, we can check $N/2$ edges in parallel, and add their values (in a tree-like structure) to increment the counter. This reduces the non-Clifford gate depth of the circuit to $\mathcal{O}(N\log(N))$. For edge-sparse complexes, the complexity may be further reduced.

We also explored to use of quantum memory to implement the membership oracle for the direct encoding. However, this proved less efficient than the approach outlined above. The idea is to create a superposition over the $\mathcal{O}(N^2)$ edges, using two ancilla registers $|i\rangle|j\rangle$, each of $\log(N)$ qubits. We can then perform a QRAM/QROM load of the positions $\vec{r}_i, \vec{r}_j$, and compute whether the corresponding edge E_{ij} is present in the complex. We also compute whether that edge is actually present in the given simplex ($E_{ij}(s)$) using a Toffoli gate between vertices i, j. We then compute $(E_{ij}=0 \wedge E_{ij}(s)=1)$, which flags that an edge is present that should not be present. We require $O(N/\sqrt{k})$ rounds of amplitude amplification to prepare the state solely composed of 'bad edges'. Each round requires two memory loads, which in the best case take $\mathcal{O}(\log(N))$ time (i.e. using $\mathcal{O}(Ndb_d)$ qubits). Amplitude amplification would also introduce error in the approximation. Hence, the classical memory-based approach is preferable for typical values of k.

2. Boundary operator implementation

In this section we determine the resource estimates to construct projected unitary encodings of the relevant boundary operators ∂_k^i and ∂_{k+1}^j. A summary of our results are present in Table XI. Our compact mapped approach proceeds by swapping the vertex to be deleted into the final position of the register, and then setting it to $|\bar{0}\rangle$ to represent the absence of a vertex. Our direct mapped approach relies on a previously observed correspondence between the

	(α, m, ϵ)	Non-Clifford gate depth	Projectors
Compact	$(\sqrt{(N+1)(k+1)}, \log(k+1), 0)$	$\mathcal{O}(k \log\log(N+1))$	$\Pi_L = \Pi_{m^i_{k-1}} \otimes \left\|0\right\rangle\left\langle 0\right\|^{\otimes \log(N+1)}$ $\Pi_R = \Pi_{m^i_k}$
Direct	$(\sqrt{N}, 0, 0)$	$\mathcal{O}(\log(N))$	$\Pi_L = \Pi_{m^i_{k-1}}$ $\Pi_R = \Pi_{m^i_k}$

TABLE XI. Parameters for the projected unitary encoding $V_{\partial^i_k}$ of ∂^i_k. The parameters for the projected unitary encoding $V_{\partial^j_{k+1}}$ of ∂^j_{k+1} follow from replacing $k \to k+1$ and $i \to j$. The projector $\Pi_{m^i_k}$ projects onto k-simplices in the complex at scale i. The implementation of this operation differs for the compact and direct encodings, as discussed in Sec. V B.

boundary operator and second quantized fermionic operators [16, 36, 37], and efficient circuits for implementing such operators [36–38].

a. *Compact mapping*

To simplify the analysis, we choose N to be expressible as $2^m - 1$ for integer m. We reserve the state $\left|\bar{0}\right\rangle$ to express an absence of a vertex. For example, if we had vertices A, B, C, we would map these as $A = \left|01\right\rangle, B = \left|10\right\rangle, C = \left|11\right\rangle$.

Under the compact mapping, we represent the boundary operator as

$$\partial_k = \sum_{V_0<\ldots<V_k} \sum_{j=0}^{k} (-1)^j \Big(\left|V_0\right\rangle \ldots \left|V_{j-1}\right\rangle \left|V_{j+1}\right\rangle \ldots \left|V_k\right\rangle \left|\bar{0}\right\rangle \Big) \Big(\left\langle V_0\right| \ldots \left\langle V_j\right| \ldots \left\langle V_k\right| \Big) \tag{C10}$$

which acts on $(k+1)\log(N)$ qubits. We focus first on the k-th boundary operator, as the $(k+1)$-th will follow as a simple modification of the former. We see that this operator projects from a k-simplex onto a $(k-1)$-simplex, where the final register storing $\left|\bar{0}\right\rangle$ denotes the missing vertex.

We will show how to implement an $(\alpha, \log(k+1), 0)$ projected unitary encoding $V_{\partial^i_k}$ such that

$$\Big(\left|0\right\rangle\left\langle 0\right|^{\otimes \log(k+1)} \otimes \Pi_L\Big) V_{\partial^i_k} \Big(\left|0\right\rangle\left\langle 0\right|^{\otimes \log(k+1)} \otimes \Pi_R\Big) = \frac{\partial^i_k}{\alpha} \tag{C11}$$

where α is a normalisation factor, and L/R denote left/right. As before, we define $\Pi_{m^i_k}$ as the projector onto k-simplices in the complex at scale i. We choose the projector $\Pi_L = \Pi_{m^i_{k-1}} \otimes \left|0\right\rangle\left\langle 0\right|^{\otimes \log(N+1)}$ and the projector $\Pi_R = \Pi_{m^i_k}$. These projectors ensure that the operator (and its Hermitian conjugate) only act on simplices in the complex (with the correct vertex ordering), and that the output simplices contain the correct number of vertices.

We can verify our projected unitary encoding by considering the matrix element (for simplices present in the complex and ordered correctly, ensured by the projectors)

$$\Big(\left\langle V_0\right| \ldots \left\langle V_{j-1}\right| \left\langle V_{j+1}\right| \ldots \left\langle V_k\right| \left\langle \bar{0}\right| \otimes \left\langle 0\right|^{\otimes \log(k+1)}\Big) V_{\partial^i_k} \Big(\left|V_0\right\rangle \ldots \left|V_j\right\rangle \ldots \left|V_k\right\rangle \otimes \left|0\right\rangle^{\otimes \log(k+1)}\Big) \tag{C12}$$

which should equal $(-1)^j/\alpha$. To show the implementation of $V_{\partial^i_k}$, we consider its effect on the state $\left|V_0\right\rangle \ldots \left|V_j\right\rangle \ldots \left|V_k\right\rangle \otimes \left|0\right\rangle^{\otimes \log(k+1)}$, and step through the circuit.

We first prepare a uniform superposition over the ancilla register, which can be done with $\log(k+1)$ Hadamard gates

$$\frac{1}{\sqrt{k+1}} \sum_{j=0}^{k} \left|V_0\right\rangle \ldots \left|V_k\right\rangle \otimes \left|j\right\rangle. \tag{C13}$$

Using a work ancilla qubit, we perform a j-conditioned permutation that shuffles the j-th state into the final register

$$\frac{1}{\sqrt{k+1}} \sum_{j=0}^{k} \left|V_0\right\rangle \ldots \left|V_{j-1}\right\rangle \left|V_{j+1}\right\rangle \ldots \left|V_k\right\rangle \left|V_j\right\rangle \otimes \left|j\right\rangle. \tag{C14}$$

This circuit requires k controlled-SWAP gates, and $2k$-comparisons of the $\log(k+1)$-bit number j. We apply a Z gate to the final bit of $|j\rangle$

$$\frac{1}{\sqrt{k+1}}\sum_{j=0}^{k}(-1)^j\left|V_0\right\rangle ...\left|V_{j-1}\right\rangle\left|V_{j+1}\right\rangle ...\left|V_k\right\rangle\left|V_j\right\rangle\otimes|j\rangle\,. \tag{C15}$$

We then compute the value of j into another work register, by coherently checking the value of the final register against the values of each neighbouring pairs of registers (e.g. we check $|V_j\rangle$ against $|V_{l+1}\rangle, |V_l\rangle$, and so on). Because the registers store simplices in ascending order, we seek the only pair $|V_l\rangle, |V_{l+1}\rangle$ such that $V_l < V_j < V_{l+1}$. We require $\mathcal{O}(k)$ comparisons of $\log(N+1)$ qubit registers, which takes $\mathcal{O}(k\log\log(N+1))$ depth, and $\log(N+1)$ ancilla qubits. This enables us to uniquely identify j, which lets us uncompute the initial superposition over $|j\rangle$. We then uncompute the 2nd register $|j\rangle$ using inverse of the inequality checking outlined above. This yields the state

$$\frac{1}{\sqrt{k+1}}\sum_{j=0}^{k}(-1)^j\left|V_0\right\rangle ...\left|V_{j-1}\right\rangle\left|V_{j+1}\right\rangle ...\left|V_k\right\rangle\left|V_j\right\rangle\otimes|0\rangle^{\otimes\log(k+1)}\,. \tag{C16}$$

We then apply $\log(N+1)$ Hadamard gates to the $|V_j\rangle$ register, resulting in the state

$$\frac{1}{\sqrt{(N+1)(k+1)}}\sum_{j=0}^{k}(-1)^j\left|V_0\right\rangle ...\left|V_{j-1}\right\rangle\left|V_{j+1}\right\rangle ...\left|V_k\right\rangle\left(|\bar{0}\rangle+\sum_{x=1}^{N}\pm|x\rangle\right)\otimes|0\rangle^{\otimes\log(k+1)}\,. \tag{C17}$$

We can then consider the matrix element

$$\begin{aligned}
&\langle V_0|...\langle V_{j-1}|\langle V_{j+1}|...\langle V_k|\langle\bar{0}|\otimes\langle 0|^{\otimes\log(k+1)}\left(V_{\partial_k^i}\right)|V_0\rangle ...|V_j\rangle ...|V_k\rangle\otimes|0\rangle^{\otimes\log(k+1)} \\
=&\left(\langle V_0|...\langle V_{j-1}|\langle V_{j+1}|...\langle V_k|\langle\bar{0}|\otimes\langle 0|^{\otimes\log(k+1)}\right) \\
&\times\left(\frac{1}{\sqrt{(N+1)(k+1)}}\sum_{j=0}^{k}(-1)^j\left|V_0\right\rangle ...\left|V_{j-1}\right\rangle\left|V_{j+1}\right\rangle ...\left|V_k\right\rangle\left(|\bar{0}\rangle+\sum_{x=1}^{N}\pm|x\rangle\right)\otimes|0\rangle^{\otimes\log(k+1)}\right) \\
=&\frac{(-1)^j}{\sqrt{(N+1)(k+1)}}
\end{aligned} \tag{C18}$$

which shows that $V_{\partial_k^i}$ is a $\left(\sqrt{(N+1)(k+1)},\log(k+1),0\right)$ projected unitary encoding of ∂_k^i. Implementing this projected unitary encoding requires a gate-depth of $\mathcal{O}(k\log\log(N+1))$.

To implement a projected unitary encoding $V_{\partial_{k+1}^j}$

$$\left(|0\rangle\langle 0|^{\otimes\log(k+2)}\otimes\Pi_L\right)V_{\partial_{k+1}^j}\left(|0\rangle\langle 0|^{\otimes\log(k+2)}\otimes\Pi_R\right)=\frac{\partial_{k+1}^j}{\alpha} \tag{C19}$$

we likewise choose the projectors $\Pi_L=\Pi_{m_k^j}\otimes|0\rangle\langle 0|^{\otimes\log(N+1)}$, $\Pi_R=\Pi_{m_{k+1}^j}$. We can reuse the analysis above to show that $V_{\partial_{k+1}^j}$ is a $\left(\sqrt{(N+1)(k+2)},\log(k+2),0\right)$ projected unitary encoding of ∂_{k+1}^j.

b. Direct mapping

To implement the projected unitary encoding of the boundary operators in the direct mapping, we exploit a previously observed link to second quantized fermionic creation and annihilation operators [16, 36, 37]. Specifically, the unrestricted boundary operator ∂ (which acts on all possible simplices, of all dimensions) can be expressed as

$$\partial+\partial^\dagger=\sum_{i=0}^{N-1}a_i+a_i^\dagger \tag{C20}$$

where $a_i^\dagger, a_i$ are second quantized fermionic creation and annihilation operators, respectively. We can take two routes to implementing a projected unitary encoding of this operator. The first, as outlined in Ref. [37], uses a unitary circuit with no additional ancilla qubits to implement a scaled version of this operator. This circuit has a non-Clifford depth of $\mathcal{O}(\log(N))$ (composed of $2(N-1)$ single-qubit rotations), and introduces a scaling factor of $\alpha = \sqrt{N}$. Alternatively, one could use techniques for implementing projected unitary encodings of fermionic operators introduced in the literature on quantum chemistry. For example, the approach of Ref. [38] would use $\log(N)$ ancilla qubits, and a T gate depth of $24\lceil\log(N)\rceil$ (with a T count of $24(N-1)$) to implement a projected unitary encoding of $\partial + \partial^\dagger$ with a scaling factor of $\alpha = N$. As a result of the reduced ancilla count and improved scaling factor, we expect the first approach to have better performance.

The projection operations required for this approach are $\Pi_L = \Pi_{m^i_{k-1}}$, $\Pi_R = \Pi_{m^i_k}$, which ensures that the operator acts on simplices in the complex of the correct dimension. The unitary embedding approach thus provides a $(\sqrt{N}, 0, 0)$ projected unitary encoding of ∂^i_k. In Table. XII we compare the costs the unitary embedding approach to that of the method used to provide a projected unitary encoding of ∂^i_k in Ref. [17]. We see that the unitary embedding approach is preferable in all parameters.

	Unitary embedding [37]	Ref [17]
Ancilla qubits	0	$\sim (\lceil\log(N)\rceil + 2\lceil\log(k)\rceil + 5)$
α	$\sqrt{N}$	$N(k+1)$
Non-Clifford gates	$2(N-1)$ single-qubit rotations	$\mathcal{O}(N^2)$ Toffoli gates
Non-Clifford depth	$\mathcal{O}(\log(N))$	$\Omega(N)$

TABLE XII. A comparison of the unitary embedding approach for implementing a projected unitary encoding of ∂^i_k in the direct mapping against the approach of Ref. [17].

3. Projected unitary encodings of subspace projectors

In this section we discuss how to implement the following projected unitary encodings:

- $V_{\Pi_{\text{Ker}}}$: A projected unitary encoding of Π_{Ker}, the projector onto the kernel of ∂^i_k.
- $V_{\Pi_{\text{Im}}}$: A projected unitary encoding of Π_{Im}, the projector onto the image of ∂^j_{k+1}.
- $V_{\Pi\Pi}$: A projected unitary encoding of $\Pi_{\text{Ker}\cap\text{Im}}$, the projector onto $\text{Ker}(\partial^i_k) \cap \text{Im}(\partial^j_{k+1})$.

These will be constructed from QSVT circuits applied to the projected unitary encodings of ∂^i_k and ∂^j_{k+1}. We use the same approach for both the compact and direct mappings. As a result, we will only discuss our methods for the case of the compact mapping.

a. *Implementation of $C_\Pi NOT$ gates*

As shown in Fig. 2, a QSVT circuit requires repeated calls to $\text{C}_{\Pi_L}\text{NOT}$ and $\text{C}_{\Pi_R}\text{NOT}$ gates. We discuss how to construct these gates here. For the compact mapping, we have $\Pi_L = \Pi_{m^i_{k-1}} \otimes |0\rangle\langle 0|^{\otimes \log(N+1)}$ and $\Pi_R = \Pi_{m^i_k}$ (and similar for the direct mapping). We can thus implement the gate $\text{C}_{\Pi_L}\text{NOT}$ as shown in Fig. 4. We introduce an ancilla qubit, use the membership oracle $O_{m^i_{k-1}}$ applied to the first k registers to flip an ancilla qubit, and then apply a multicontrol-Toffoli gate controlled on the ancilla qubit and the remaining vertex register to flip the target ancilla. We then uncompute the intermediate ancilla using the membership oracle. The approach is similar for the $\text{C}_{\Pi_R}\text{NOT}$ gate, but now applying the membership oracle $O_{m^i_k}$ to all $(k+1)$ vertices (which therefore does not require an additional ancilla qubit).

Because the C_ΠNOT gates are applied in pairs, separated only by an R_z gate on the ancilla qubit, we can cancel the applications of the membership oracle inside the C_ΠNOT block. Hence, the degree d QSVT circuit makes d calls to each type of the C_ΠNOT gates. As a result, it uses d applications each of $O_{m^i_{k-1}}$ and $O_{m^i_k}$. As discussed in Sec. C 1 a, our implementation of the compact membership oracle is approximate, with an error of $2\sqrt{\epsilon_m}$. As a result, the repeated sequence of $V_{\partial^i_k}$ and C_ΠNOT gates deviates from the ideal QSVT circuit. The error is bounded by $4d\sqrt{\epsilon_m}$. In the following sections, we will determine the value of d required for the QSVT transformations.

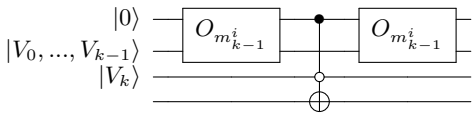


FIG. 4. Implementing the $\mathrm{C}_{\Pi_L}\mathrm{NOT}$ for $\Pi_L = \Pi_{m^i_{k-1}} \otimes |0\rangle\langle 0|^{\otimes \log(N+1)}$ using two calls to the membership oracle.

b. *Projector onto kernel*

The projector onto the kernel can be implemented using the approach of Ref. [39], which considered how to optimally prepare ground states. While that work considered Hermitian matrices, and so used the quantum eigenvalue transform, the results can easily be adapted to the non-Hermitian case, using the QSVT formalism. Using Lemma 5 of Ref. [39], given an $(\alpha, m, 0)$ projected unitary encoding of an operator A, we can implement $V_{\Pi_{\mathrm{Ker}(A)}}$, which is a $(1, m+3, \epsilon_k)$ projected unitary encoding of $\Pi_{\mathrm{Ker}(A)}$. We require $\mathcal{O}\big(\frac{\alpha}{\Lambda}\log(\epsilon_k^{-1})\big)$ calls to an $(\alpha, m, 0)$ projected unitary encoding of A, and its inverse (where Λ is the gap between the zero and lowest non-zero singular values), and to the necessary $\mathrm{C}_{\Pi}\mathrm{NOT}$ gates.

As a result, in the compact mapping (assuming first that the membership oracle is perfect) we can implement a $(1, \log(k+1)+3, \epsilon_k)$ projected unitary encoding of Π_{Ker} using $\mathcal{O}\left(\frac{\sqrt{(N+1)(k+1)}}{\Lambda_{\partial^i_k}}\log(\epsilon_k^{-1})\right)$ calls to $V_{\partial^i_k}$, and its inverse, and to the membership oracles $O_{m^i_k}$, $O_{m^i_{k-1}}$. When these membership oracles are imperfect, each introduces an error of $2\sqrt{\epsilon_m}$. The projected unitary encoding then becomes a $\left(1, \log(k+1)+3, \epsilon_k + \frac{2\sqrt{\epsilon_m(N+1)(k+1)}}{\Lambda_{\partial^i_k}}\log(\epsilon_k^{-1})\right)$ projected unitary encoding of Π_{Ker}.

For the direct mapping, we can implement a $(1, 3, \epsilon_k)$ projected unitary encoding of Π_{Ker} using $\mathcal{O}\left(\frac{\sqrt{N}}{\Lambda_{\partial^i_k}}\log(\epsilon_k^{-1})\right)$ calls to $V_{\partial^i_k}$, and its inverse, and to the membership oracles $O_{m^i_k}$, $O_{m^i_{k-1}}$.

c. *Projector onto image*

To implement a projector onto the image of a matrix A, consider its singular value decomposition $A = \sum_i \sigma_i |L_i\rangle\langle R_i|$. A projector onto the image of A is given by $\Pi_{\mathrm{Im}(A)} = \sum_{j;\sigma_j>0} |L_j\rangle\langle L_j|$. We can implement a projected unitary encoding of this projector by using the QSVT to apply a even polynomial approximating a threshold function to the singular values of the encoded operator $A^\dagger$ (which can be implemented using $V_A^\dagger$, when V_A is a projected unitary encoding of A). This ensures that all singular values above the threshold Λ/α are set to 1, and all values below are set to zero. The use of an even polynomial ensures that we only keep the left singular values [30]. We can implement $V_{\Pi_{\mathrm{Im}(A)}}$, which is a $(1, m+1, \epsilon_i)$ projected unitary encoding of $\Pi_{\mathrm{Im}(A)}$ using $\mathcal{O}\big(\frac{\alpha}{\Lambda}\log(\epsilon_i^{-1})\big)$ calls to an $(\alpha, m, 0)$ projected unitary encoding of A, and its inverse, and to the corresponding $\mathrm{C}_{\Pi}\mathrm{NOT}$ gates.

As a result, in the compact mapping (assuming first that the membership oracle is perfect) we can implement a $(1, \log(k+1)+1, \epsilon_i)$ projected unitary encoding of Π_{Im} using $\mathcal{O}\left(\frac{\sqrt{(N+1)(k+2)}}{\Lambda_{\partial^j_{k+1}}}\log(\epsilon_i^{-1})\right)$ calls to $V_{\partial^j_{k+1}}$, and its inverse, and to the membership oracles $O_{m^j_k}$, $O_{m^j_{k+1}}$. When these membership oracles are imperfect, each introduces an error of $2\sqrt{\epsilon_m}$. The projected unitary encoding then becomes a $\left(1, \log(k+1)+1, \epsilon_i + \frac{2\sqrt{\epsilon_m(N+1)(k+2)}}{\Lambda_{\partial^j_{k+1}}}\log(\epsilon_i^{-1})\right)$ projected unitary encoding of Π_{Im}.

For the direct mapping, we can implement a $(1, 1, \epsilon_i)$ projected unitary encoding of Π_{Im} using $\mathcal{O}\left(\frac{\sqrt{N}}{\Lambda_{\partial^j_{k+1}}}\log(\epsilon_i^{-1})\right)$ calls to $V_{\partial^j_{k+1}}$, and its inverse, and to the membership oracles $O_{m^j_k}$, $O_{m^j_{k+1}}$.

	(α, m, ϵ)	Costs
Compact	$\left(1, 2\log(k+1)+5, \frac{1}{\Lambda_{\Pi\Pi}}\log\left(\epsilon_p^{-1}\right)\sqrt{\epsilon_{\text{ker}}+\epsilon_{\text{im}}+8\sqrt{\epsilon_m}}+\epsilon_p\right)$	$\mathcal{O}\left(\frac{1}{\Lambda_{\Pi\Pi}}\log\left(\frac{1}{\epsilon_p}\right)\right)\times$ $V_{\Pi_{\text{Ker}}}, V_{\Pi_{\text{Ker}}}^\dagger, V_{\Pi_{\text{Im}}}, V_{\Pi_{\text{Im}}}^\dagger, O_m$
Direct	$\left(1, 5, \frac{1}{\Lambda_{\Pi\Pi}}\log\left(\epsilon_p^{-1}\right)\sqrt{\epsilon_k+\epsilon_i}+\epsilon_p\right)$	

TABLE XIII. A summary of the costs to implement the projected unitary encoding $V_{\Pi\Pi}$. The error parameters ϵ_{ker} and ϵ_{im} depend on a number of other parameters (including ϵ_m), as shown in Appendix C 3 d. We use the shorthand O_m to indicate that we need membership oracles for $O_{m_k^i}$, $O_{m_{k-1}^i}$, $O_{m_k^j}$, $O_{m_{k+1}^j}$.

d. *Projector onto kernel and image*

Given the above methods for implementing $V_{\Pi_{\text{Ker}}}$ and $V_{\Pi_{\text{Im}}}$, we can implement $V_{\Pi\Pi}$. We cannot simply take a product of the two projectors, as in general they do not commute with each other (because of new simplices that enter the complex at scale j). We can implement $V_{\Pi\Pi}$ by first using a product of projected unitary encodings [30] to obtain a product of the projectors $\Pi_{\text{Ker}}\Pi_{\text{Im}}$. We can use Lemma 53 of Ref. [30], which states that:
If U is an (α, a, δ) projected unitary encoding of A and V is a (β, b, ϵ) projected unitary encoding of B, then $(I_b \otimes U)(I_a \otimes V)$ is an $(\alpha\beta, a+b, \alpha\epsilon+\beta\delta)$ projected unitary encoding of AB.

As a result, in the compact mapping we can implement a

$$\left(1, 2\log(k+1)+4, \left(\epsilon_k + \frac{2\sqrt{\epsilon_m(N+1)(k+1)}}{\Lambda_{\partial_k^i}}\log(\epsilon_k^{-1}) + \epsilon_i + \frac{2\sqrt{\epsilon_m(N+1)(k+2)}}{\Lambda_{\partial_{k+1}^j}}\log(\epsilon_i^{-1})\right)\right) \tag{C21}$$

projected unitary encoding of $\Pi_{\text{Ker}}\Pi_{\text{Im}}$ using one call each to of $V_{\Pi_{\text{Ker}}}$ and $V_{\Pi_{\text{Im}}}$. We define

$$\epsilon_{\text{ker}} := \epsilon_k + \frac{2\sqrt{\epsilon_m(N+1)(k+1)}}{\Lambda_{\partial_k^i}}\log(\epsilon_k^{-1}) \tag{C22}$$

$$\epsilon_{\text{im}} := \epsilon_i + \frac{2\sqrt{\epsilon_m(N+1)(k+2)}}{\Lambda_{\partial_{k+1}^j}}\log(\epsilon_i^{-1}). \tag{C23}$$

In the direct mapping we can implement a

$$(1, 4, \epsilon_k + \epsilon_i) \tag{C24}$$

projected unitary encoding of $\Pi_{\text{Ker}}\Pi_{\text{Im}}$ using one call each to of $V_{\Pi_{\text{Ker}}}$ and $V_{\Pi_{\text{Im}}}$.

We then apply the QSVT to this projected unitary encoding. We apply a function that sends all non-unity singular values to zero. In reality, we can only send singular values below a threshold $(1-\Lambda_{\Pi\Pi})$ to zero. We require $\mathcal{O}\left(\Lambda_{\Pi\Pi}^{-1}\log\left(\epsilon_p^{-1}\right)\right)$ calls to $V_{\Pi_{\text{Ker}}}$ and $V_{\Pi_{\text{Im}}}$ (and their inverses) to implement $V_{\Pi\Pi}$ in this way. The error in the above projected unitary encoding is scaled according to the robustness of QSVT Lemma. The QSVT circuit also makes a number of calls to the corresponding C_ΠNOT gates. Each $\text{C}_{\Pi_{L/R}}\text{NOT}$ gate requires two calls to the membership oracle. For the compact mapping, using Lemma 3 this effectively increases the error in the above projected unitary encoding by a factor of $8\sqrt{\epsilon_m}$.

This yields the costs shown in Table XIII to implement the projected unitary encoding $V_{\Pi\Pi}$.

4. Preparing projected unitary encoding V_{ψ_m}

In this section we will discuss how to prepare the projected unitary encoding V_{ψ_m} of the operator $|\psi_m\rangle\langle\bar{0}|$ where $|\psi_m\rangle$ is the purified maximally mixed state over k-simplices in the complex at scale i. We do this by constructing circuits that prepare a purification of the maximally mixed state over all possible k-simplices. These unitaries provide a projected unitary encoding of $|\psi_m\rangle\langle\bar{0}|$ (scaled by a factor of $\sqrt{|S_k^i|/\binom{N}{k+1}}$), which can then be converted into the desired operator using the QSVT. Our approach can be viewed as a QSVT-phrased version of fixed-point amplitude amplification [30, 74].

a. *Preparing purification of maximally mixed state over all possible k-simplices*

In this section we consider unitary circuits U_{uni} that prepare the purification of the maximally mixed state over all $\binom{N}{k+1}$ possible k-simplices in the complex. We will discuss how to implement unitaries that first prepare an equal

superposition over all possible k-simplices in the complex. We can then introduce a second register of equal size, and apply CNOT gates between the corresponding qubits in each register, to generate the purification.

The direct encoding stores k-simplices as Hamming weight $(k+1)$ computational basis states of N qubits. An equally weighted superposition of states with fixed Hamming weight is known as a Dicke state. We can prepare Dicke states using the approaches in Refs. [40, 41], the most efficient of which requires $\mathcal{O}((k+1)\log(N/k+1))$ depth.

Preparing an equal superposition of k-simplices in the compact encoding proceeds as follows. We will consider $N+1=2^m$ for some integer m to simplify the analysis. The uniform superposition over k-simplices built from N datapoints is

$$\frac{1}{\sqrt{\binom{N}{k+1}}}\sum_{s_k}|s_k\rangle = \frac{1}{\sqrt{\binom{N}{k+1}}}\sum_{V_1=1}^{[N-k]}\sum_{V_2>V_1}^{[N-(k-1)]}\ldots\sum_{V_{k+1}>V_k}^{N}|V_1\rangle|V_2\rangle\ldots|V_{k+1}\rangle \tag{C25}$$

where each register acts on $m=\log(N+1)$ qubits. We can prepare this state by first placing each register in an equal superposition of N states

$$\left(\frac{1}{\sqrt{N}}\sum_{V_i=1}^{N}|V_i\rangle\right)^{\otimes(k+1)}. \tag{C26}$$

The dimension of the state above is $N^{(k+1)}$. Preparing the superpositions on each register can be done (for each register in parallel) using $\mathcal{O}(\log(N))$ gates [35] (see also Refs. [78–80]). We need to isolate the components of this state that are permutations of our desired orderings (i.e. we need to remove branches of the superposition with repeated vertices)

$$\frac{1}{\sqrt{(k+1)!\binom{N}{k+1}}}\sum_{V_1\neq V_2\ldots\neq V_{k+1}}^{N}|V_1\rangle|V_2\rangle\ldots|V_{k+1}\rangle \tag{C27}$$

We check all pairs of registers (which can be done in $\mathcal{O}(k\log(k))$ depth, using $\mathcal{O}(k\log(N+1))$ ancilla qubits) to trigger a flag qubit if the branch of the superposition contains any repeated vertices. The dimension of the desired 'permutation state' is $N\times\ldots\times(N-k)=\frac{N!}{(N-k-1)!}$. Using bounds on binomial coefficients, we require $\mathcal{O}\left(\sqrt{\frac{(k+1)^{(k+1)}}{(k+1)!}}\log(\epsilon_s^{-1})\right)$ rounds of amplitude amplification to prepare the permutation state with an error bounded by $\sqrt{\epsilon_s}$. The complexity increases rapidly with the value of k, but still contributes a factor of <100 for $k\leq 10$.

Once the permutation state is prepared it can be sorted into the correct order using a reversible quantum sorting network, as described in Sec. 6.3.1 of Ref. [78]. The sorting network requires $\mathcal{O}(k\log(k)\log\log(N))$ depth and $\mathcal{O}(k(\log(k)+\log(N)))$ additional ancilla qubits. At the end of the sorting network, the ancilla register is unentangled with the desired state, and so can be discarded. The overall complexity of preparing the equal superposition over simplices in the compact mapping is thus

$$\mathcal{O}\left((\log(N)+k\log(k))\left(\sqrt{\frac{(k+1)^{(k+1)}}{(k+1)!}}\log(\epsilon_s^{-1})\right)+(k\log(k)\log\log(N)\right) \tag{C28}$$

and requires

$$\mathcal{O}(k(\log(k)+\log(N))) \tag{C29}$$

additional ancilla qubits. The unitary circuit has an error of $\sqrt{\epsilon_s}$.

b. Preparing the purified maximally mixed state over k-simplices in the complex via fixed-point amplitude amplification

Given the above unitary circuits that prepare the purification of the maximally mixed state over all possible k-simplices in the complex, we can use fixed point amplitude amplification (implemented using the QSVT) to generate

V_{ψ_m}. This is a projected unitary encoding of the operator $|\psi_m\rangle\langle\bar{0}|$, where $|\psi_m\rangle$ is the purified maximally mixed state over k-simplices in the complex. We follow the approach to fixed-point amplitude amplification outlined in Sec.3 of Ref. [74].

We define the following operations: U_{uni} (as above) prepares the purified maximally mixed state over all possible k-simplices, $\Pi_0 = |\bar{0}\rangle\langle\bar{0}|$, $\Pi_m = |\psi_m\rangle\langle\psi_m|$. We note that the membership oracle $O_{m_k^i}$ acts as $|\psi_m\rangle\langle\psi_m|$ in the 2D subspace spanned by $|\psi_m\rangle, |\psi_m^{\perp}\rangle$, which we remain in during the course of this subroutine. We then see that

$$||\Pi_m U_{\text{uni}} \Pi_0 - \sqrt{\frac{|S_k^i|}{\binom{N}{k+1}}}\,|\psi_m\rangle\langle\bar{0}|\,|| \leq \sqrt{\epsilon_s} \tag{C30}$$

i.e. U_{uni} provides a $(1, 0, \sqrt{\epsilon_s})$ projected unitary encoding of $\sqrt{\frac{|S_k^i|}{\binom{N}{k+1}}}\,|\psi_m\rangle\langle\bar{0}|$. We can use the QSVT to map the singular value $\sqrt{\frac{|S_k^i|}{\binom{N}{k+1}}}$ to 1. A layer of the resulting QSVT circuit in shown in Fig. 5.

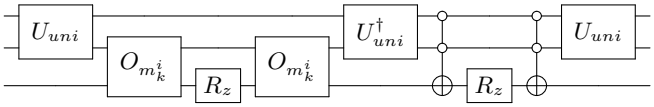


FIG. 5. A layer of the QSVT implementation of fixed-point amplitude amplification used to implement the projected unitary encoding V_{ψ_m}.

Because both U_{uni} and $O_{m_k^i}$ (used to implement $\text{C}_{\Pi_m}\text{NOT}$) have errors, we can invoke Lemma 3, applied to the QSVT circuit for fixed-point amplitude amplification. The resulting QSVT circuit V_{ψ_m} is a

$$\left(1, 2, \epsilon_\psi + 4\sqrt{\frac{\binom{N}{k+1}}{|S_k^i|}}\log(\epsilon_\psi^{-1})\sqrt{\sqrt{\epsilon_s} + \sqrt{\epsilon_m}}\right) \tag{C31}$$

projected unitary encoding of $|\psi_m\rangle\langle\bar{0}|$, and uses $\mathcal{O}\left(\sqrt{\frac{\binom{N}{k+1}}{|S_k^i|}}\log(\epsilon_\psi^{-1})\right)$ calls to $O_{m_k^i}$, U_{uni}, and $U_{\text{uni}}^{\dagger}$.

D. Determining the overall complexity of the algorithm

In this section we determine the overall complexity of our algorithm.

Instance x	1	2	3
Estimates	$\sqrt{\lvert S_k^i\rvert/\binom{N}{k+1}}$	$\sqrt{\dim(\text{Ker}(\partial_k^i))/\lvert S_k^i\rvert}$	$\sqrt{\dim\big(\text{Ker}(\partial_k^i)\cap\text{Im}(\partial_{k+1}^j)\big)/\lvert S_k^i\rvert}$
δ_x	$\frac{\Delta}{2\sqrt{3}\beta_k^{i,j}}\sqrt{\frac{\lvert S_k^i\rvert}{\binom{N}{k+1}}}$	$\frac{\Delta}{2\sqrt{3\lvert S_k^i\rvert\dim\big(\text{Ker}(\partial_k^i)\big)}}$	$\frac{\Delta}{2\sqrt{3\lvert S_k^i\rvert\dim\big(\text{Ker}(\partial_k^i)\cap\text{Im}(\partial_{k+1}^j)\big)}}$
Π	$\Pi_{m_k^i}$: Projects onto k-simplices in complex	Π_{Ker}: Projects onto $\text{Ker}(\partial_k^i)$	$\Pi_{\text{Ker}\cap\text{Im}}$: Projects onto $\text{Ker}(\partial_k^i)\cap\text{Im}(\partial_{k+1}^j)$
$\lvert\psi\rangle$	$\lvert\psi_s\rangle$: Uniform superposition of all possible k-simplices	$\lvert\psi_m\rangle$: Purification of maximally mixed state over all k-simplices in complex at scale i	

TABLE XIV. Details of the error parameters and projected unitary encoded operators for each of the instances of normalised projector rank estimation used to estimate the persistent Betti number to additive error Δ using the algorithm of Theorem 2.

We invoke Corollary 2.1, which we restate here:
To estimate $\beta_k^{i,j}$ to additive error Δ with success probability $\geq 1-\eta$, we solve three instances of normalised projector rank estimation. Each instance x uses $\tilde{\mathcal{O}}\left(\frac{\log(\delta_x^{-1})\log(\eta^{-1})}{(1-\chi_\psi)^2(1-2\epsilon_x)^2}\right)$ incoherent repetitions of a quantum circuit, where $\epsilon_x := \frac{4}{\delta_x}\log(\epsilon_f^{-1})\sqrt{2\chi_\psi + \chi_\pi + \sum_i(\epsilon_L^i + \epsilon_R^i)} + \epsilon_f$. The quantum circuit for each instance makes $\mathcal{O}\left(\frac{1}{\delta_x}\log(\epsilon_f^{-1})\right)$ calls

to V_{ψ_x} and V_{Π_x} (and their inverses), as well as the corresponding faulty $C_\Pi NOT$ gates that induce the projected unitary encodings. The algorithm also uses $\mathcal{O}\left(\frac{1}{\delta_x}\log(\epsilon_f^{-1})\right)$ additional single- and two-qubit gates. The error parameters δ_x and quantities $|\psi_x\rangle, \Pi_x$ are shown in Table XIV.

We will make use of the following facts and assumptions throughout:

- We expect instance 3 to be the most costly, as it uses the most complex projection operator; we thus bound the cost of the algorithm by the asymptotic cost of instance 3.
- We assume that $\delta_2 = \delta_3 \sim \frac{\Delta}{\sqrt{|S_k^i|}}$.
- We assume the cost of calls to membership oracles in the algorithm have a cost equal to $O_{m_k^i}$ (for $k-1, k, k+1$ and i, j).

The cost of the quantum circuit is then given by:

$$\begin{aligned} &\mathcal{O}\left(\frac{1}{\delta_3}\log\left(\frac{1}{\epsilon_f}\right)\times\left(V_{\psi_m}+V_{\Pi\Pi}+O_{m_k^i}\right)\right) \\ \in\ &\mathcal{O}\left(\frac{1}{\delta_3}\log\left(\frac{1}{\epsilon_f}\right)\times\left(V_{\psi_m}+V_{\Pi\Pi}\right)\right) \end{aligned} \tag{D1}$$

because each of $V_{\Pi\Pi}$ and V_{ψ_m} make many calls to $O_{m_k^i}$. In the following subsections, we will determine these costs in terms of the building blocks introduced in the preceding sections. We will do this for both the direct mapping, and the compact mapping with QROM. We will not treat the compact mapping with QRAM explicitly, as it is a small modification of the QROM results.

1. Direct mapping

We focus first on the comparatively more simple case of the direct mapping. We have the following costs:

- V_{ψ_m} : $(1, 1, \epsilon_\psi)$ projected unitary encoding, using $\sqrt{\frac{\binom{N}{k+1}}{|S_k^i|}}\log\left(\frac{1}{\epsilon_\psi}\right)\times[U_{\text{Uni}}+O_{m_k^i}]$.
  - U_{Uni} : $k\log(N)$
  - $O_{m_k^i}$: $N\log(N)$
- $V_{\Pi\Pi}$: $\left(1, 5, \frac{1}{\Lambda_{\Pi\Pi}}\log(\epsilon_p^{-1})\sqrt{\epsilon_k+\epsilon_i}+\epsilon_p\right)$ projected unitary encoding, using $\frac{1}{\Lambda_{\Pi\Pi}}\frac{\sqrt{N}}{\text{Min}\left(\Lambda_{\partial_k^i},\Lambda_{\partial_{k+1}^j}\right)}\log\left(\frac{1}{\epsilon_p}\right)\log\left(\frac{1}{\epsilon_{i/k}}\right)\times[V_\partial+O_{m_k^i}]$.
  - V_∂ : $\log(N)$

We have used that the cost of $V_{\partial_k^i}$ and $V_{\partial_{k+1}^j}$ are the same. We refer to the error $\epsilon_{i/k}$ because as we shall show below, ϵ_i and ϵ_k are approximately equal.

Substituting these costs into the overall algorithmic cost (and dropping the subleading terms gives:

$$\mathcal{O}\left(\frac{N\log(N)}{\delta_3}\log\left(\frac{1}{\epsilon_f}\right)\times\left(\sqrt{\frac{\binom{N}{k+1}}{|S_k^i|}}\log\left(\frac{1}{\epsilon_\psi}\right)+\frac{\sqrt{N}}{\Lambda_{\Pi\Pi}\text{Min}\left(\Lambda_{\partial_k^i},\Lambda_{\partial_{k+1}^j}\right)}\log\left(\frac{1}{\epsilon_{i/k}}\right)\log\left(\frac{1}{\epsilon_p}\right)\right)\right) \tag{D2}$$

We must determine suitable values for the errors $\epsilon_\psi, \epsilon_f, \epsilon_{i/k}, \epsilon_p$. As stated in Corollary 2.1, the algorithm uses $\tilde{\mathcal{O}}\left(\frac{\log(\delta_3^{-1})\log(\eta^{-1})}{(1-\chi_\psi)^2(1-2\epsilon_3)^2}\right)$ incoherent repetitions of the quantum circuit, where $\epsilon_3 := \frac{4}{\delta_3}\log(\epsilon_f^{-1})\sqrt{2\chi_\psi+\chi_\pi+\sum_i(\epsilon_L^i+\epsilon_R^i)}+\epsilon_f$. This expression simplifies considerably here, because the C_ΠNOT gates are noiseless. We choose to bound both $(1-\chi_\psi)^{-2}$ and $(1-2\epsilon_3)^{-2}$ by constants. The former value will not appear in our algorithmic scalings. This is because it is the error due to state preparation, which is only called once in the algorithm, and so is dominated by

the rest of the circuit. Hence we can make this factor close to 1 with no difference in the asymptotic complexity of the circuit. We set $(1-2\epsilon_3)^{-2} = C$ (i.e. a factor of C increase in the number of repetitions required), which implies $\epsilon_3 = \frac{1}{2}\left(1 - \frac{1}{\sqrt{C}}\right) = C'$ constant. We can then choose:

$$\begin{aligned} \epsilon_f &\sim \epsilon_3 \\ \epsilon_\psi, \epsilon_p &\sim \frac{\delta_3^2 \epsilon_3^2}{\log^2\left(\frac{1}{\epsilon_3}\right)} \\ \epsilon_i, \epsilon_k &\sim \frac{\delta_3^4 \epsilon_3^4 \Lambda_{\Pi\Pi}^2}{\log^4\left(\frac{1}{\epsilon_3}\right)\log^2\left(\frac{\log\left(\frac{1}{\epsilon_3}\right)}{\epsilon_3 \delta_3}\right)} \end{aligned} \tag{D3}$$

Substituting these values into the cost of the algorithm, and hiding constant terms using the big-$\mathcal{O}$ notation yields

$$\mathcal{O}\left(\frac{N\log(N)}{\delta_3}\log\left(\frac{1}{\delta_3^2}\right) \times \left(\sqrt{\frac{\binom{N}{k+1}}{|S_k^i|}} + \frac{\sqrt{N}}{\Lambda_{\Pi\Pi}\mathrm{Min}\left(\Lambda_{\partial_k^i}, \Lambda_{\partial_{k+1}^j}\right)}\log\left(\frac{1}{\delta_3^4\Lambda_{\Pi\Pi}^2}\right)\right)\right). \tag{D4}$$

We consider $|S_k^i| \sim \binom{N}{k+1}$, as this is the regime in which classical algorithms are least efficient. The resulting complexity is:

$$\mathcal{O}\left(\frac{N^{3/2}\log(N)\sqrt{\binom{N}{k+1}}}{\Delta\Lambda_{\Pi\Pi}\mathrm{Min}\left(\Lambda_{\partial_k^i}, \Lambda_{\partial_{k+1}^j}\right)}\log\left(\frac{\sqrt{\binom{N}{k+1}}}{\Delta}\right)\log\left(\frac{\binom{N}{k+1}}{\Delta^2\Lambda_{\Pi\Pi}}\right)\right) \tag{D5}$$

non-Clifford depth. We require

$$\tilde{\mathcal{O}}\left(\log\left(\frac{\sqrt{\binom{N}{k+1}}}{\Delta}\right)\log\left(\frac{1}{\eta}\right)\right) \tag{D6}$$

incoherent repetitions of this circuit. Hence, the overall complexity is

$$\tilde{\mathcal{O}}\left(\frac{N^{3/2}\sqrt{\binom{N}{k+1}}}{\Delta\Lambda_{\Pi\Pi}\mathrm{Min}\left(\Lambda_{\partial_k^i}, \Lambda_{\partial_{k+1}^j}\right)}\right). \tag{D7}$$

The quantum circuit used acts on $\mathcal{O}(N)$ qubits.

2. Compact mapping

We now focus on the compact mapping using a QROM to implement the membership oracle. We have the following costs:

- V_{ψ_m} : $\left(1, 2, \epsilon_\psi + 4\sqrt{\frac{\binom{N}{k+1}}{|S_k^i|}}\log\left(\frac{1}{\epsilon_\psi}\right)\sqrt{\sqrt{\epsilon_s} + \sqrt{\epsilon_m}}\right)$ projected unitary encoding, using $\sqrt{\frac{\binom{N}{k+1}}{|S_k^i|}}\log\left(\frac{1}{\epsilon_\psi}\right) \times [U_{\mathrm{Uni}} + O_{m_k^i}]$.
  - $U_{\mathrm{Uni}} : (\log(N) + k\log(k))\sqrt{\frac{(k+1)^{k+1}}{(k+1)!}}\log\left(\frac{1}{\epsilon_s}\right)$
  - $O_{m_k^i} : \sqrt{k}\log\left(\frac{1}{\epsilon_m}\right)(N + \log(x)\log(b_x) + b_x) + k$
- $V_{\Pi\Pi}$: $\left(1, 2\log(k+1) + 5, \frac{1}{\Lambda_{\Pi\Pi}}\log\left(\epsilon_p^{-1}\right)\sqrt{\epsilon_{\ker} + \epsilon_{\mathrm{im}} + 8\sqrt{\epsilon_m}} + \epsilon_p\right)$ projected unitary encoding, using $\frac{\sqrt{(N+1)(k+1)}}{\Lambda_{\Pi\Pi}\mathrm{Min}\left(\Lambda_{\partial_k^i}, \Lambda_{\partial_{k+1}^j}\right)}\log\left(\frac{1}{\epsilon_p}\right)\log\left(\frac{1}{\epsilon_{i/k}}\right) \times [V_{\partial_k^i} + O_{m_k^i}]$.

- $V_{\partial_k^i} : k \log\log(N+1)$
- $\epsilon_{\text{ker}} = \epsilon_k + \frac{2\sqrt{\epsilon_m (N+1)(k+1)}}{\Lambda_{\partial_k^i}} \log(\epsilon_k^{-1})$
- $\epsilon_{\text{im}} = \epsilon_i + \frac{2\sqrt{\epsilon_m (N+1)(k+2)}}{\Lambda_{\partial_{k+1}^j}} \log(\epsilon_i^{-1})$

We have made the simplifications that the costs of $V_{\partial_k^i}$ and $V_{\partial_{k+1}^j}$ are asymptotically the same. We refer to the error $\epsilon_{i/k}$ because as we shall show below, ϵ_i and ϵ_k are approximately equal.

Substituting these costs into the overall algorithmic cost yields

$$\begin{aligned}\mathcal{O}\left(\frac{1}{\delta_3}\log\left(\frac{1}{\epsilon_f}\right)\times\left[\sqrt{\frac{\binom{N}{k+1}}{|S_k^i|}}\log\left(\frac{1}{\epsilon_\psi}\right)\times\left[(\log(N)+k\log(k))\sqrt{\frac{(k+1)^{k+1}}{(k+1)!}}\log\left(\frac{1}{\epsilon_s}\right)\right.\right.\right.\\ \left.\left.\left.+\sqrt{k}\log\left(\frac{1}{\epsilon_m}\right)(N+\log(x)\log(b_x)+b_x)+k\right]\right.\right.\\ \left.\left.+\frac{\sqrt{(N+1)(k+1)}}{\Lambda_{\Pi\Pi}\text{Min}\left(\Lambda_{\partial_k^i},\Lambda_{\partial_{k+1}^j}\right)}\log\left(\frac{1}{\epsilon_{i/k}}\right)\log\left(\frac{1}{\epsilon_p}\right)\times\left[k\log\log(N+1)+\right.\right.\right.\\ \left.\left.\left.+\sqrt{k}\log\left(\frac{1}{\epsilon_m}\right)(N+\log(x)\log(b_x)+b_x)+k\right]\right]\right)\end{aligned} \tag{D8}$$

We must determine suitable values for the errors $\epsilon_\psi, \epsilon_f, \epsilon_{i/k}, \epsilon_m, \epsilon_s, \epsilon_p$. As stated in Corollary 2.1, the algorithm uses $\tilde{\mathcal{O}}\left(\frac{\log(\delta_3^{-1})\log(\eta^{-1})}{(1-\chi_\psi)^2(1-2\epsilon_3)^2}\right)$ incoherent repetitions of the quantum circuit, where $\epsilon_3 := \frac{4}{\delta_3}\log(\epsilon_f^{-1})\sqrt{2\chi_\psi+\chi_\pi+6\sqrt{\epsilon_m}}+\epsilon_f$ here. We choose to bound both $(1-\chi_\psi)^{-2}$ and $(1-2\epsilon_3)^{-2}$ by constants. The former value will not appear in our algorithmic scalings. This is because it is the error due to state preparation, which is only called once in the algorithm, and so is dominated by the rest of the circuit. Hence we can make this factor close to 1 with no difference in the asymptotic complexity of the circuit. We set $(1-2\epsilon_3)^{-2} = C$ (i.e. a factor of C increase in the number of repetitions required), which implies $\epsilon_3 = \frac{1}{2}\left(1-\frac{1}{\sqrt{C}}\right) = C'$ constant. We can then choose:

$$\begin{aligned}\epsilon_f &\sim \epsilon_3\\ \epsilon_\psi, \epsilon_p &\sim \frac{\delta_3^2\epsilon_3^2}{\log^2\left(\frac{1}{\epsilon_3}\right)}\\ \epsilon_k, \epsilon_i &\sim \frac{\Lambda_{\Pi\Pi}^2\delta_3^4\epsilon_3^4}{\log^4\left(\frac{1}{\epsilon_3}\right)\log^2\left(\frac{\log\left(\frac{1}{\epsilon_3}\right)}{\epsilon_3\delta_3}\right)}\\ \epsilon_m &\sim \frac{\Lambda_{\Pi\Pi}^4\delta_3^8\epsilon_3^8\text{Min}\left(\Lambda_{\partial_k^i},\Lambda_{\partial_{k+1}^j}\right)^2}{(N+1)(k+1)\log^8\left(\frac{1}{\epsilon_3}\right)\log^4\left(\frac{\log\left(\frac{1}{\epsilon_3}\right)}{\epsilon_3\delta_3}\right)\log^2\left(\frac{1}{\Lambda_{\Pi\Pi}^2\epsilon_3^4\delta_3^4}\right)}\\ \epsilon_s &\sim \frac{\delta_3^8\epsilon_3^8}{\log^8\left(\frac{1}{\epsilon_3}\right)\log^4\left(\frac{\log^2\left(\frac{1}{\epsilon}\right)}{\epsilon_3^2\delta_3^2}\right)}\left(\frac{|S_k^i|}{\binom{N}{k+1}}\right)^2\end{aligned} \tag{D9}$$

Substituting these values into the cost of the algorithm, and hiding constant terms using the big-$\mathcal{O}$ notation yields

$$\begin{aligned}\mathcal{O}\left(\frac{1}{\delta_3}\log\left(\frac{1}{\delta_3}\right)\times\left[\sqrt{\frac{\binom{N}{k+1}}{|S_k^i|}}\times\left[(\log(N)+k\log(k))\sqrt{\frac{(k+1)^{k+1}}{(k+1)!}}\log\left(\frac{\binom{N}{k+1}^2\log^4\left(\frac{1}{\delta_3^2}\right)}{|S_k^i|^2\delta_3^8}\right)\right.\right.\right.\\ \left.\left.\left.+\sqrt{k}\log\left(\frac{1}{\epsilon_m}\right)(N+\log(x)\log(b_x)+b_x)+k\right]\right.\right.\end{aligned} \tag{D10}$$

$$+\frac{\sqrt{(N+1)(k+1)}}{\Lambda_{\Pi\Pi}\mathrm{Min}\Big(\Lambda_{\partial_k^i},\Lambda_{\partial_{k+1}^j}\Big)}\log\left(\frac{\log\Big(\frac{1}{\delta_3}\Big)}{\Lambda_{\Pi\Pi}\delta_3^2\epsilon_3^2}\right)\times\Big[k\log\log(N+1)+$$
$$+\sqrt{k}\log\left(\frac{1}{\epsilon_m}\right)(N+\log(x)\log(b_x)+b_x)+k\Big]\Bigg)$$

We consider $|S_k^i|\sim\binom{N}{k+1}$, as this is the regime in which classical algorithms are least efficient. We also assume $N\gg k,\log(x)\log(b_x),b_x$. The resulting complexity is:

$$\mathcal{O}\left(\frac{1}{\delta_3}\log\left(\frac{1}{\delta_3}\right)\log\left(\frac{1}{\epsilon_m}\right)\times\left[\left((\log(N)+k\log(k))\sqrt{\frac{(k+1)^{k+1}}{(k+1)!}}+N\sqrt{k}\right)\right.\right. \tag{D11}$$
$$\left.\left.+\frac{\sqrt{(N+1)(k+1)}}{\Lambda_{\Pi\Pi}\mathrm{Min}\Big(\Lambda_{\partial_k^i},\Lambda_{\partial_{k+1}^j}\Big)}\times N\sqrt{k}\right]\right).$$

We are interested in growing N asymptotically. We thus choose k constant. The resulting complexity is

$$\mathcal{O}\left(\frac{1}{\delta_3}\log\left(\frac{1}{\delta_3}\right)\log\left(\frac{(N+1)(k+1)\log^4\Big(\frac{1}{\delta_3}\Big)\log^2\Big(\frac{1}{\Lambda_{\Pi\Pi}\delta_3^2}\Big)}{\Lambda_{\Pi\Pi}^4\delta_3^8\mathrm{Min}\Big(\Lambda_{\partial_k^i},\Lambda_{\partial_{k+1}^j}\Big)^2}\right)\times\frac{N^{3/2}k}{\Lambda_{\Pi\Pi}\mathrm{Min}\Big(\Lambda_{\partial_k^i},\Lambda_{\partial_{k+1}^j}\Big)}\right) \tag{D12}$$
$$\in\tilde{\mathcal{O}}\left(\frac{N^{3/2}k\sqrt{\binom{N}{k+1}}}{\Delta\Lambda_{\Pi\Pi}\mathrm{Min}\Big(\Lambda_{\partial_k^i},\Lambda_{\partial_{k+1}^j}\Big)}\right)$$

non-Clifford depth. We require

$$\tilde{\mathcal{O}}\left(\log\left(\frac{\sqrt{\binom{N}{k+1}}}{\Delta}\right)\log\left(\frac{1}{\eta}\right)\right) \tag{D13}$$

incoherent repetitions of this circuit. Hence, the overall complexity is

$$\tilde{\mathcal{O}}\left(\frac{N^{3/2}\sqrt{\binom{N}{k+1}}}{\Delta\Lambda_{\Pi\Pi}\mathrm{Min}\Big(\Lambda_{\partial_k^i},\Lambda_{\partial_{k+1}^j}\Big)}\right). \tag{D14}$$

The quantum circuit used acts on $\mathcal{O}\big(k\log(N)\big)$ qubits.

E. Nuances of persistent Betti numbers

1. Performing a change of basis

As discussed in the main text, there are a number of nuances associated with building operators that can encode persistent Betti numbers. In this work, we compute $\beta_k^{i,j}$ as

$$\beta_k^{i,j}=\dim\big(\mathrm{Ker}(\partial_k^i)\big)-\dim\Big(\mathrm{Ker}(\partial_k^i)\cap\mathrm{Im}(\partial_{k+1}^j)\Big). \tag{E1}$$

Nevertheless, there are other ways this problem can be formulated. For example, the task of computing persistent Betti numbers has been framed in terms of a persistent combinatorial Laplacian [43, 47]. This is achieved by first defining a subgroup $C_{k+1}^{i,j}(S^j)$ containing $(k+1)$-chains in the complex at scale j whose images under the boundary operator ∂_{k+1}^j are k-chains at scale i. Formally,

$$C_{k+1}^{i,j}(S^j)=\{c\in C_{k+1}(S^j):\partial_{k+1}^j(c)\in C_k(S^i)\}. \tag{E2}$$

Intuitively, the chains in $C^{i,j}_{k+1}(S^j)$ are mapped to k-cycles in $C_k(S^i)$, which can be divided into the k-boundaries already present in $C_k(S^i)$, and a subset of the k-holes in $C_k(S^i)$. In other words, the additional $(k+1)$-chains in $C^{i,j}_{k+1}(S^j)$ (beyond those already present in $C_{k+1}(S^i)$) have the action of filling-in k-holes in $C_k(S^i)$. Defining $\partial^{i,j}_{k+1}$ as ∂_{k+1} restricted to the chains (not just the simplices) in $C^{i,j}_{k+1}(S^j)$ it follows that [43]

$$\mathrm{Im}(\partial^{i,j}_{k+1}) \cong \mathrm{Ker}(\partial^i_k) \cap \mathrm{Im}(\partial^j_{k+1}). \tag{E3}$$

Moving from the simplex basis to the chain basis is the main subject of this section.

In Refs. [43, 47] the $(j-i)$-th persistent k-th combinatorial Laplacian is defined as

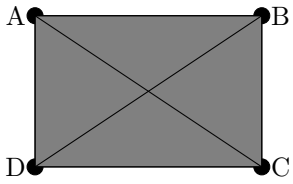
$$\Delta^{i,j}_k = \partial^{i,j}_{k+1} \circ (\partial^{i,j}_{k+1})^\dagger + (\partial^i_k)^\dagger \partial^i_k. \tag{E4}$$

It is shown in Ref. [43] that $\beta^{i,j}_k = \dim\Big(\mathrm{Ker}(\Delta^{i,j}_k)\Big)$. We have used $\circ$ in Eq. (E4) to highlight that the expression of $\partial^{i,j}_{k+1}$ in terms of matrices is more complex than expressing ∂^i_k, as discussed in Ref. [43], and as we shall show below. This is because the $(k+1)$-simplices in S^j_{k+1} do not form an appropriate basis for the group $C^{i,j}_{k+1}(S^j)$. Even if a chain $\sigma_1 + \sigma_2 \in C^{i,j}_{k+1}(S^j)$ (such that $\partial^j_{k+1}(\sigma_1+\sigma_2) \in C_k(S^i)$), it can be the case that $\partial^j_{k+1}(\sigma_1), \partial^j_{k+1}(\sigma_2) \notin C_k(S^i)$. The required change of basis adds complexity to both the classical and quantum algorithms for persistent Betti numbers. We elaborate more on this point below.

In this Appendix we provide a worked example for constructing the restricted boundary operator $\partial^{i,j}_{k+1}$, in order to build intuition for the increased complexity. Below, we show a pair of simplicial complexes at different scales i and j. Clearly there are zero 1-holes that persist from scale i to scale j, as the hole is filled by the new 2-simplices at scale j.

$$S^i_0 = \{A, B, C, D\}$$
$$S^i_1 = \{AB, BC, CD, AD\}$$
$$S^i_2 = \{\}$$

$$S^j_0 = \{A, B, C, D\}$$
$$S^j_1 = \{AB, BC, CD, AD, AC, BD\}$$
$$S^j_2 = \{ABC, ACD, ABD, BCD\}$$



We are interested in the elements of the restricted chain group

$$C^{i,j}_{k+1}(S^j) = \{c \in C_{k+1}(S^j) : \partial^j_{k+1}(c) \in C_k(S^i)\}. \tag{E5}$$

In this example, it is straightforward to find the elements of this group manually:

$$\begin{aligned}
\partial^j_2[ABC] &= BC - AC + AB \\
\partial^j_2[ACD] &= CD - AD + AC \\
\partial^j_2[ABD] &= BD - AD + AB \\
\partial^j_2[BCD] &= CD - BD + BC.
\end{aligned} \tag{E6}$$

The 1-chains AC and BD are present at scale j, but are not present at scale i. Consequently, we need to take linear combinations of the 2-simplices in j that will cancel out these errant 1-chains to form the elements of $C^{i,j}_2(S^j)$. These are given by

$$\begin{aligned}
c_1 &= ABC + ACD; \quad \partial^j_2[c_1] = AB + BC + CD - AD \\
c_2 &= ABD + BCD; \quad \partial^j_2[c_2] = AB + BC + CD - AD,
\end{aligned} \tag{E7}$$

or any linear combination of these chains. We see that while $c_1, c_2 \in C^{i,j}_2(S^j)$, their constituent simplices are not, as mentioned above. This is why we must represent $\partial^{i,j}_2$ in the basis of elements of $C^{i,j}_2(S^j)$, rather than in the basis of simplices. We can see that the dimension of $\partial^{i,j}_2$ is two, and its rank is one. The persistent Betti number is given by

$$\beta^{i,j}_1 = \dim\big(\mathrm{Ker}(\partial^i_1)\big) - \dim\Big(\mathrm{Im}(\partial^{i,j}_2)\Big) \tag{E8}$$

$$= 1 - 1$$
$$= 0$$

as expected.

We now consider what the matrix representation of $\partial_2^{i,j}$ should be. We closely follow the steps outlined in Ref. [43], in particular the proof of Lemma 3.4 in that work.

The dimension 2 boundary matrix at scale j, projected onto simplices present at scale j is given by

$$\partial_2^j P_2^j = \begin{array}{c} \begin{matrix} ABC & ACD & ABD & BCD \end{matrix} \\ \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & -1 & -1 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{pmatrix} \end{array} \begin{matrix} AB \\ BC \\ CD \\ AD \\ AC \\ BD \end{matrix}$$

As an aside, we note that the quantum algorithm in Ref. [18] defines the restricted boundary operator as $P_1^i \partial_2^j P_2^j$. We have been unable to verify that this quantum algorithm is able to correctly obtain the persistent Betti numbers. The matrix representation of this operator for our example is

$$P_1^i \partial_2^j P_2^j = \begin{array}{c} \begin{matrix} ABC & ACD & ABD & BCD \end{matrix} \\ \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & -1 & -1 & 0 \end{pmatrix} \end{array} \begin{matrix} AB \\ BC \\ CD \\ AD \end{matrix}$$

This operator has rank 3, and would therefore give $\beta_1^{i,j} = -2$, which is not possible. Another issue with this operator can be seen by considering its action on ABC (or any of the 2-simplices). The operator maps ABC to $AB + BC$, and we have that $ABC \in C_2(S^j)$, $AB, BC \in C_1(S^i)$, from which we would conclude that $ABC \in C_2^{i,j}(S^j)$. However, this is not the case, as $\partial_2^j(ABC) = BC - AC + AB$, and $AC \notin C_1(S^i)$. As discussed above, the chain group $C_2^{i,j}(S^j)$ is made up of a linear combination of the 2-simplices in S^j, and therefore we need a change of basis from simplices to chains, which is not present in the above expression for $\partial_2^{i,j}$. We provide additional discussion on issues encountered when using this expression for computing persistent Betti numbers at the end of this section.

To obtain the correct expression for the restricted boundary operator, we follow the procedure of Ref. [43], and consider the projection onto the 1-chains that are not in S_1^i

$$(I - P_1^i)\partial_2^j P_2^j = \begin{array}{c} \begin{matrix} ABC & ACD & ABD & BCD \end{matrix} \\ \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{pmatrix} \end{array} \begin{matrix} AC \\ BD \end{matrix}$$

This operator maps 2-chains in j to 1-chains in j that are not in i. Therefore objects in its kernel are 2-chains in j that are in i. This coincides with the definition of $C_2^{i,j}(S^j)$, and therefore the elements of this group are in the kernel of $(I - P_1^i)\partial_2^j P_2^j := D_2^j$. We can identify these chains by performing column reduction on the matrix D_2^j. We need to find a non-singular matrix Y such that $R_2^j = D_2^j Y$ is column-reduced. Such a matrix Y can be found by doing a singular value decomposition of D_2^j. In this example, we use the matrix

$$Y = \begin{array}{c} \begin{matrix} \alpha & \beta & \gamma & \delta \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{matrix} ABC \\ ACD \\ ABD \\ BCD \end{matrix}$$

such that

$$R_2^j = \begin{array}{c} \begin{matrix} \alpha & \beta & \gamma & \delta \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \begin{matrix} AC \\ BD \end{matrix}$$

We can see from R_2^j and Y that the columns indexed by α, δ are in the kernel of D_2^j, and correspond to the chains $\alpha = c_1 = ABC + ACD$, $\delta = c_2 = ABD + BCD$, as found earlier. As a result, Y has acted as a change of basis, from the simplicial basis, to a basis of chains that has, as a subset, a basis of $C_2^{i,j}(S^j)$.

The matrix representation for $\partial_2^{i,j}$ is then obtained by changing the column space of ∂_2^j using Y, and projecting into the columns corresponding to elements of $C_2^{i,j}(S^j)$, and the rows corresponding to 1-simplices in S^i:

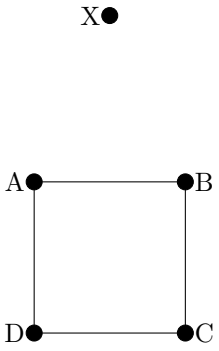
$$
\begin{aligned}
\partial_2^{i,j} &= \Big((\partial_2^j P_2^j)\cdot Y\Big)[AB:AD][\alpha,\delta] \\
&= \begin{pmatrix}
 & ABC & ACD & ABD & BCD \\
AB & 1 & 0 & 1 & 0 \\
BC & 1 & 0 & 0 & 1 \\
CD & 0 & 1 & 0 & 1 \\
AD & 0 & -1 & -1 & 0 \\
AC & -1 & 1 & 0 & 0 \\
BD & 0 & 0 & 1 & -1
\end{pmatrix}
\cdot
\begin{pmatrix}
 & \alpha & \beta & \gamma & \delta \\
ABC & 1 & 0 & 0 & 0 \\
ACD & 1 & 1 & 0 & 0 \\
ABD & 0 & 0 & 1 & 1 \\
BCD & 0 & 0 & 0 & 1
\end{pmatrix}
[AB:AD][\alpha,\delta] \\
&= \begin{pmatrix}
 & \alpha & \beta & \gamma & \delta \\
AB & 1 & 0 & 1 & 1 \\
BC & 1 & 0 & 0 & 1 \\
CD & 1 & 1 & 0 & 1 \\
AD & -1 & -1 & -1 & -1 \\
AC & 0 & 1 & 0 & 0 \\
BD & 0 & 0 & 1 & 0
\end{pmatrix}
[AB:AD][\alpha,\delta] \\
&= \begin{pmatrix}
 & \alpha & \delta \\
AB & 1 & 1 \\
BC & 1 & 1 \\
CD & 1 & 1 \\
AD & -1 & -1
\end{pmatrix}
\end{aligned}
$$

This matrix maps 2-chains in $C_2(S^j)$ into 1-chains in $C_1(S^i)$, and has a rank of one. As a result, it correctly gives $\beta_1^{i,j} = 0$ for this filtration.

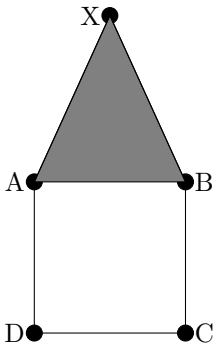
We see that finding the matrix representation of $\partial_{k+1}^{i,j}$ is more complex than finding the matrices $\partial_k^i, \partial_k^j$, etc. This results from the need to identify the subspace $C_{k+1}^{i,j}(S^j)$, and change to a basis that spans this subspace. In this example, this was achieved using the column reduction of D_2^j, and finding the rotation matrix Y. A more algorithmic approach was also introduced in Ref. [43], which builds the matrix representation of the persistent combinatorial Laplacian by expressing the matrix representation of $\partial_{k+1}^{i,j} \circ \left(\partial_{k+1}^{i,j}\right)^\dagger$ in terms of the Schur complement of the submatrix $\partial_{k+1}^j \left(\partial_{k+1}^j\right)^\dagger [\{I_k^j\}][\{I_k^j\}]$ in the matrix $\partial_{k+1}^j \left(\partial_{k+1}^j\right)^\dagger$, where the $\{I_k^j\}$ is the set of k-simplices in S^j that are not in S^i. The implementation of this Schur complement requires taking four submatrices of $\partial_{k+1}^j \left(\partial_{k+1}^j\right)^\dagger$, multiplying two of the submatrices by the Moore-Penrose pseudo-inverse of the third, and subtracting this product from the first submatrix. This approach was adapted for use in the quantum algorithm for finding persistent Betti numbers by Hayakawa [17]. Implementing the Schur complement requires quantum subroutines for block encoding the matrix $\partial_{k+1}^j \left(\partial_{k+1}^j\right)^\dagger$, performing projections into the subspaces S^i, S^j (and their complements), and being able to take the Moore-Penrose pseudo-inverse of block encoded matrices.

We now provide an additional discussion of issues encountered when using $P_k^i \partial_{k+1} P_{k+1}^j$ as a representation of the restricted boundary operator $\partial_{k+1}^{i,j}$, as was done in Ref. [18]. We consider the following pair of complexes:

$$S_0^i = \{A, B, C, D, X\}$$
$$S_1^i = \{AB, BC, CD, AD\}$$
$$S_2^i = \{\}$$



$$S_0^j = \{A, B, C, D, X\}$$
$$S_1^j = \{AB, BC, CD, AD, AX, BX\}$$
$$S_2^j = \{ABX\}$$



The persistent Betti number $\beta_1^{i,j}$ is 1 for this system. However, the matrix representation of $P_1^i \partial_2 P_2^j$ is

$$\partial_2 P_2^j = \begin{array}{c} ABX \\ \left(\begin{array}{c} 1 \\ -1 \\ 1 \end{array} \right) \begin{array}{l} AB \\ AX \\ BX \end{array} \end{array}$$

and

$$P_1^i \partial_2 P_2^j = \begin{array}{c} ABX \\ \left(\begin{array}{c} 1 \\ 0 \\ 0 \end{array} \right) \begin{array}{l} AB \\ AX \\ BX \end{array} \end{array}$$

This matrix has a rank of 1. Hence, if we try to compute the persistent Betti number as

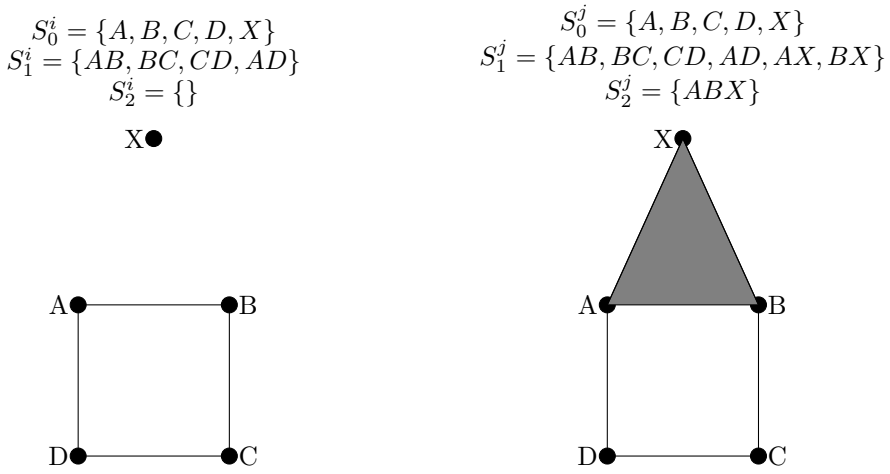
$$\begin{aligned} \beta_1^{i,j} &= \dim\big(\mathrm{Ker}(\partial_1^i)\big) - \dim\Big(\mathrm{Im}(\partial_2^{i,j})\Big) \\ &= 1 - 1 \\ &= 0 \end{aligned} \tag{E9}$$

which is not the expected value.

2. Persistent Betti numbers from the quantum Zeno effect

In Ref. [14] it is suggested that it is possible to compute the persistent Betti numbers using only the original quantum algorithm presented in Ref. [13] for the calculation of Betti numbers. The suggested approach is to initially perform phase estimation with the combinatorial Laplacian at scale μ_1, repeating until an eigenvalue of 0 is found. Phase estimation is then performed again with the combinatorial Laplacian at scale $\mu_1 + \delta\mu$, and the hole is considered to persist for at least $\delta\mu$ if an eigenvalue of 0 is measured. Eventually, at scale μ_2 the hole will close fully, at which point the measured eigenvector no longer has eigenvalue 0. It is suggested that if the steps of phase estimation are performed at small enough increments of $\delta\mu$, the probability of projecting into the hole eigenstate stays high, making use of the quantum Zeno effect.

We have been unable to recover the suggested result, based on the following example. Consider the system



The relevant distances in this diagram are: $d_{AB} = 2, d_{AX} \approx 2.42, d_{AC} = 2\sqrt{2}$. At scale i with $\mu_i = d_{AB}$, the hole $AB + BC + CD - AD$ is created. As the value of μ used to construct the combinatorial Laplacian is increased, there is no change until $\mu_j = d_{AX}$. As this stage, we can see that the hole still persists. However, objects in the kernel of the combinatorial Laplacian are the harmonic representative of the homology group, and so can have unexpected forms [72]. In this case, the (unnormalized) zero eigenvector of Δ_1^j is given by $3(AB + BC + CD - AD) - \partial_2(ABX)$. The squared overlap between the (normalised) new and original holes is not equal to one. As a result, there is a non-zero probability of projecting into a different eigenstate, that is not in the kernel of Δ_1^j. Imagine a complex that consists of many copies of this system, separated by a large distance. As the quantum algorithm for Betti numbers works by sampling eigenstates at random, we are only able to count the fraction of eigenstates of Δ_1^i with zero eigenvalue. If we then try to project these zero eigenstates onto the eigenstates of Δ_1^j, we would erroneously conclude that some non-zero fraction of the holes had closed – leading us to believe that we have two types of holes, short-lived holes that close by stage j, and longer-lived holes that are still open at stage j. It appears that the Zeno-like procedure cannot conclusively determine persistent Betti numbers. The issue appears to be that while the parameter μ changes smoothly, the Laplacian changes discontinuously whenever a new simplex enters the filtration. As a result, the eigenstates undergo sudden jumps to include new basis states, required by the harmonic form. If these basis states were not present in the original eigenstate, then to ensure that probability is conserved, the state must also gain overlap with other (non-zero eigenvalued) eigenstates that also include the new basis states.

A possible modification to the idea outlined above would be to consider adiabatic state preparation between the initial combinatorial Laplacian Δ_k^i and the final combinatorial Laplacian Δ_k^j. For example, we could start in the ground state of Δ_k^i (whose normalised form is $0.5(AB + BC + CD - AD)$) and then time evolve the state under a time-dependent Hamiltonian $H(s) = (1 - s)\Delta_k^i + s\Delta_k^j$, interpolating slowly between $s = 0$ and $s = 1$. We would then perform phase estimation with Δ_k^j. If we measured an eigenvalue of 0, we would conclude that the hole is still open. However, if we measured a non-zero eigenvalue we would conclude that the hole has closed. Unfortunately, we have been unable to obtain the correct result with this approach. The initial Laplacian has three relevant degenerate ground states; $0.5(AB + BC + CD - AD)$, AX, and BX. Only the first of these is present in the complex at stage i, and it is this state that we start in. As soon as $s \neq 0$, the degeneracy between the states is broken, and all three states have non-zero eigenvalues (until $s = 1$, when one of the states recovers an eigenvalue of 0). To see this rigorously, we expand the initial state in the non-zero eigenstates of $Q_0(\partial H/\partial s)Q_0 = Q_0(H(1) - H(0))Q_0$, where Q_0 is the projector onto the 3 relevant zero-valued eigenvectors of $H(0)$. This determines the amplitudes of the final states, assuming we move along the path infinitely slowly. We observe that the initial state has squared overlap of 0.945 and 0.055 with two of these eigenstates (it is also possible to check the squared overlap with the low-lying eigenstates of $H(s \ll 1)$, and our results are in agreement). As a result, there will be a non-zero probability of finding the state in a non-zero eigenstate at the end of the adiabatic path. As with the example above, this would lead us to conclude that some fraction of the holes present in a complex have closed.