SIAM J. MATRIX ANAL. APPL.
Vol. 46, No. 1, pp. 172–209

MULTIVARIATE TRACE ESTIMATION USING QUANTUM STATE SPACE LINEAR ALGEBRA*

LIRON MOR-YOSEF†, SHASHANKA UBARU‡, LIOR HORESH‡, AND HAIM AVRON†

Abstract. In this paper, we present a quantum algorithm for approximating multivariate traces, i.e., the traces of matrix products. Our research is motivated by the extensive utility of multivariate traces in elucidating spectral characteristics of matrices, as well as by recent advancements in leveraging quantum computing for faster numerical linear algebra. Central to our approach is a direct translation of a multivariate trace formula into a quantum circuit, achieved through a sequence of low-level circuit construction operations. To facilitate this translation, we introduce *quantum matrix states linear algebra* (qMSLA), a framework tailored for the efficient generation of state preparation circuits via primitive matrix algebra operations. Our algorithm relies on sets of state preparation circuits for input matrices as its primary inputs and yields two state preparation circuits encoding the multivariate trace as output. These circuits are constructed utilizing qMSLA operations, which enact the aforementioned multivariate trace formula. We emphasize that our algorithm's inputs consist solely of state preparation circuits, eschewing harder to synthesize constructs such as block encodings. Furthermore, our approach operates independently of the availability of specialized hardware like QRAM, underscoring its versatility and practicality.

Key words. trace estimation, quantum computing, QBLAS, multivariate trace

MSC codes. 68Q12, 15A18, 65F99

DOI. 10.1137/24M1654749

1. Introduction.

In this paper we propose a quantum algorithm for approximating multivariate traces, i.e., traces of the products of matrices. Formally, the multivariate trace of matrices $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_k$ of compatible dimensions is defined as

$$\mathbf{MTr}_k\left(\mathbf{A}_1, \ldots, \mathbf{A}_k\right) := \mathbf{Tr}\left(\mathbf{A}_1 \mathbf{A}_2 \cdots \mathbf{A}_k\right). \tag{1}$$

Approximating multivariate traces is motivated as follows. For a square $\mathbf{A} \in \mathbb{R}^{n \times n}$, its matrix moments $\mathbf{Tr}(\mathbf{A}^k) = \mathbf{MTr}_k(\mathbf{A}, \ldots, \mathbf{A})$ for $k = 1, 2, \ldots$, which reveal useful spectral properties of $\mathbf{A}$ with applications in scientific computing [32] and other fields, are multivariate traces. Furthermore, by introducing a set of shifts we have $\mathbf{MTr}_k(\mathbf{A} - \alpha_1 \mathbf{I}, \ldots, \mathbf{A} - \alpha_k \mathbf{I}) = \mathbf{Tr}(p(\mathbf{A}))$ for some polynomial $p(x)$ of degree k and leading coefficient 1. The right-hand-side $\mathbf{Tr}(p(\mathbf{A}))$ is equal to $\sum_{i=1}^{n} p(\lambda_i)$ where $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of $\mathbf{A}$, which is a spectral sum. Since many smooth functions $f(x)$ can be approximated well using polynomials [66], it is a common practice to approximate spectral sums of the general form $\sum_{i=1}^{n} f(\lambda_i)$ using restricted forms $\mathbf{Tr}(p(\mathbf{A}))$. A well-known example is $\log\det \mathbf{A} = \sum_{i=1}^{n} \log(\lambda_i)$. Many machine learning techniques estimate spectral properties of various matrices by approximating appropriate spectral sums via polynomial spectral sums $\mathbf{Tr}(p(\mathbf{A}))$, e.g., Gaussian processes

*Received by the editors April 16, 2024; accepted for publication (in revised form) by L. Lin October 11, 2024; published electronically January 10, 2025.
https://doi.org/10.1137/24M1654749
Funding: This research was supported by the US-Israel Binational Science Foundation (grant 2017698), the Israel Science Foundation (grant 1524/23), and an IBM Faculty Award. The first author acknowledges support by the Milner Foundation and the Israel Council for Higher Education.
†Tel Aviv University, Tel Aviv, Israel (lm2@mail.tau.ac.il, haimav@tauex.tau.ac.il).
‡IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA (Shashanka.Ubaru@ibm.com, lhoresh@us.ibm.com).

[55], kernel learning [22], Bayesian learning [47], matrix completion [12], differential privacy problems [34], graph analysis [25], Hessian and neural network property analysis [54, 27], and many more [68, 14]. Other applications of multivariate traces appear in computational physics, e.g., entanglement estimation [36, 42, 9, 26, 53], quantum error mitigation [43], and quantum distinguishability measures [10].

Due to the ubiquity of matrix moments and spectral sums (and more generally multivariate traces), there is an extensive literature on efficient algorithms for spectral sum approximation. Examples of classical (conventional) algorithms for spectral sum approximation[1] are [33, 67]. Many of these classical methods are based on (a) estimating multivariate traces using randomized trace estimation techniques (Hutchinson's method [37] and related methods [5, 4, 56, 49, 19]), and (b) approximating the function or quadratic form using polynomial approximations, Lanczos Gaussian quadrature, histogram, or some other rational approximation. While these methods are powerful and useful, with the constant increase in the size of matrices encountered in applications, and ever increasing accuracy requirements (which lead to larger degrees in approximating polynomials), developing novel algorithms for trace estimation is an active research topic.

Quantum computers can efficiently perform certain linear-algebraic operations on large computational spaces (exponential in the number of qubits) and offer the potential to achieve significant speedups over classical computations. In recent years, a variety of quantum numerical linear algebra (qNLA) [52, 46, 28, 11] and quantum machine learning (QML) [57, 59, 7] methods have been proposed to harness the computational power of quantum computing to achieve polynomial to exponential speedups over the best-known classical methods. In this paper, we consider the problem of estimating multivariate traces.

1.1. Contributions. Most of the previous literature on qNLA and QML proposed techniques that operate under stringent quantum input models, e.g., ones that assume availability of speculative components such as QRAM [29]. However, in practice, availability of such components is nontrivial, e.g., efficient QRAMs are not currently available (and are not expected to be available in the near-term future). Moreover, recent results on *classical dequantization* (aka *quantum-inspired classical algorithms*) [15, 16, 65] have shown that under analogous (to QRAM) classical data structure assumptions, "dequantized" classical algorithms can achieve similar runtimes as their quantum counterparts. Therefore, we advocate the development of qNLA algorithms that operate under only lax quantum input models. In particular, algorithms that make no quantum access assumptions, i.e., start with a classical description of input data, are especially attractive. Thus, in this work, we develop a quantum algorithm for multivariate trace estimation that uses an input model where both matrices and vectors are presented as quantum state preparation circuits, rather than relying on more restrictive input models.

In designing our quantum algorithm for trace estimation we take an approach that differs from the aforementioned qNLA and QML works. Our algorithm for multivariate trace estimation is based on *quantum matrix state linear algebra* (qMSLA), a novel framework for performing numerical linear algebra computations on quantum computers. qMSLA utilizes matrix state preparation circuits as a practical and efficient construct for representing and manipulating matrices and vectors in quantum

[1] In the context of this paper, whenever we talk about "classical algorithms" we mean algorithms that use only classical computing, as opposed to "quantum algorithms" that also use quantum computing (but may have a classical component).

algorithms. The key concept behind qMSLA is the development of a set of basic linear algebra operations that can be executed directly on circuits that represent matrices in probability amplitudes of quantum states. For example, given matrix state preparation circuits for matrices $\mathbf{A}$ and $\mathbf{B}$, qMSLA operation qmsla.kronecker outputs a matrix state preparation circuit for $\mathbf{A} \otimes \mathbf{B}$. These operations enable the execution of various matrix algebra tasks entirely within the quantum domain. In section 3 we propose a limited yet useful set of qMSLA operations and show how they can be implemented efficiently.

Although there are foundational quantum circuits and algorithms, QLA and otherwise, and common patterns for designing quantum circuits, these represent high-level operations and do not provide general purpose matrix algebra subroutines, which are necessary building blocks for computational linear algebra. qMSLA allows practitioners to overcome this limitation by offering a matrix-level framework for doing linear algebra on quantum computers, and thus designing quantum circuits by directly synthesizing quantum circuits that implement matrix equations. We believe that the idea of constructing quantum circuits of target matrix computations (such as the trace) using a series of primitive matrix algebra operations at the circuit level might be of independent interest, as a new paradigm for doing matrix computations in the quantum domain. Indeed, our approach is very much inspired by BLAS, and we view qMSLA as a (as of yet limited) BLAS for QLA.

We leverage qMSLA to design a novel algorithm for solving multivariate trace problems. Along with fulfilling the main goal of this study, this also showcases the potential of the qMSLA approach. The algorithm seamlessly executes a sequence of high-level qMSLA operation, eliminating the need for a separate, custom-built quantum circuit. The algorithm's input are predesigned circuits, denoted as $\mathcal{U}_{\mathbf{A}_i}$ for matrices $\mathbf{A}_i$ (i from 1 to $2k$).[2] These circuits encode the matrices into quantum states, that is, $\mathcal{U}_{\mathbf{A}_i}$ implements an amplitude encoding procedure for $\mathbf{A}_i$ [58, equation 3.60, p. 115]. Our algorithm outputs two circuits, $\mathcal{U}_\psi$ and $\mathcal{U}_\phi$, that prepare two quantum states, $|\psi\rangle$ and $|\phi\rangle$, such that their dot product (overlap in quantum computing parlance) is equal to the multivariate trace. This overlap can then be estimated using various existing quantum algorithms (such as the Hadamard test and the swap test). Alternatively, the circuits $\mathcal{U}_\psi$ and $\mathcal{U}_\phi$ can be used in the context of a larger quantum algorithm. We emphasize that our approach is able to handle matrices that are not necessarily Hermitian or even square. We also discuss how our algorithm can be used to approximate multivariate traces of matrices given in classical memory, showing that even in this case, under certain conditions, our algorithm entertains a small computational gain over state-of-the-art classical trace estimators. We also discuss how our algorithm can be used to approximate spectral sums.

The proposed algorithm makes only mild assumptions on the input matrices, namely that it has access to matrix state preparation circuits for each input matrix. Construction of such state preparation circuits for some matrices can be inexpensive (possibly polylogarithmic in matrix size) [31, 38], [58, section 4.2.2]. At the baseline, for a general matrix available in classical memory, it is possible to construct a state preparation circuit for it in time that is linear in the number of entries in the matrix and has similar depth [62], and this is the basis of a end-to-end classical-to-classical-via-quantum use of our algorithm.

[2]In this study, we have developed an algorithm specifically tailored for the multiplication of an even number of matrices. Odd numbers of matrices can be dealt with by introducing the identity matrix as one of the input matrices.

1.2. Related work. For the classical setting, there is scarce literature on computing multivariate traces per se, but there is extensive literature on stochastic estimation of the trace of implicit matrices. The latter can be used to estimate multivariate traces. Virtually all methods for stochastic trace estimation trace their origin to Hutchinson's work [37], and the well-known Hutchinson's estimator, though Hutchinson actually cites Girard [30] in his abstract. Extensive followup, such as [5, 4, 56, 19], improved on Hutchinson's results. The state-of-the-art estimator is Hutch++ [49], which demonstrates improved convergence rates through randomized low-rank approximations. Classical stochastic trace estimators are often combined with polynomial approximations to design stochastic estimators for spectral sums [33, 67].

Quantum computing offers a unique opportunity for faster linear algebra computations due to its ability to represent and manipulate vectors in a superposition state. This, combined with the power of entanglement, opens doors for potential speedups compared to classical matrix algorithms. Recent advancements in this field have explored various quantum techniques for fundamental linear algebra operations, including principal component analysis [7], matrix multiplication [60], solving linear systems of equations [35], and others [17, 28, 64, 2, 45]. Using quantum algorithms for trace estimation and spectral sum approximation has been considered as well.

Early work on quantum trace estimation traces its origin to the one clean qubit model of computations [63]. Given a circuit that implements a unitary operator $\mathbf{A}$, access to a maximally entangled state, and one clean pure ancilla qubit, they show how to estimate $\frac{1}{N}\mathbf{Tr}(\mathbf{A})$, where N is the size of the matrix, as a state overlap measured via the Hadamard test. However, due to the N^{-1} coefficient, the statistical error is much larger than the one commonly observed by Hutchinson's method. Another early work on quantum trace estimation is [24], which introduced a direct method for multivariate trace estimation of multiple density matrices using a controlled-SWAP gate. However, their circuit design is impractical for near-term devices. More recently, three preprints suggested algorithms related to multivariate trace estimation or spectral sum estimation. Luongo and Shao suggested several quantum algorithms for spectral sum estimation [46]. Their algorithms are based on stringent quantum access assumptions, such as block encoding and QRAM. Such assumptions pose significant challenges [23, 51], e.g., implementing QRAMs might require a vast number of qubits (trillions for 8 GB of RAM), and might even be infeasible. Shen et al. propose a quantum algorithm that essentially implements Hutchison's estimator in the quantum domain [61]. However, since creating random vector samples from the Rademacher distribution is not efficient in the quantum domain, they show how to construct a random state which they call the "quantum Hutchinson state." The distribution of state vector of the quantum Hutchinson state entertains statistical properties on par with the Rademacher vector when it comes to stochastic trace estimation. Bravyi et al. consider approximating partition functions, i.e., $\mathbf{Tr}(e^{-\beta\mathbf{H}})$ for some inverse temperature β and Hamiltonian $\mathbf{H}$ [8]. They consider both classical and quantum algorithms for local and nonlocal Hamiltonians. Finally, Quek, Kaur, and Wilde, who were the first to use the term "multivariate trace estimation," suggested a constant quantum depth circuit to estimate multivariate traces of density operators [53]. Their method assumes that inputs are presented as copies of quantum states, and it computes the multivariate trace of the density operators associated with those states.

As explained in the previous section, our algorithm is based on a framework (qMSLA) for operating in the quantum domain on matrices via a set of primitive operations. This approach is inspired by the well-known BLAS library in numerical linear algebra [41]. Other efforts in this vein have been proposed in the quantum

domain. Although not labeled as "BLAS," Gilyén et al. show how to do basic matrix arithmetic, including matrix addition, subtraction, and multiplication, using block encodings [28], and use these techniques to propose their seminal quantum singular value transformation (QSVT) algorithm. In a review paper by Biamonte et al., the authors catalogued functionalities crucial for quantum machine learning algorithms under the umbrella term "qBLAS" [7]. These functionalities, which include fast Fourier transforms and eigenvalue/eigenvector computations, are actually present (in the classical domain) in LAPACK and not BLAS implementations. Another effort is a Q#-specific library called QBLAS [20]. QBLAS assumes the presence of QRAM hardware and implements a simulated version. QBLAS offers functionalities like vector inner products, an HHL linear solver, matrix eigenvalue decomposition, and quantum phase estimation. Despite the name referencing BLAS, the QBLAS library, like qBLAS, focuses on higher-order linear algebra operations typically found in LAPACK and not in BLAS implementations.

2. Preliminaries. In this section, we present the notations, definitions, and background information necessary for describing our algorithms and their analyses.

2.1. Linear algebra notation. We denote scalars using Greek letters or using $x, y, \ldots$. Vectors are denoted by $\mathbf{x}, \mathbf{y}, \ldots$ and matrices by $\mathbf{A}, \mathbf{B}, \ldots$. The $s \times s$ identity matrix is denoted $\mathbf{I}_s$. We assume 0-based indexing for vectors and matrices. This is less common in the numerical linear algebra literature, but is more convenient in the context of quantum computing. We use the convention that vectors are column vectors, unless otherwise stated. For a vector $\mathbf{x}$ or a matrix $\mathbf{A}$, the notation $\mathbf{x}^*$ or $\mathbf{A}^*$ denotes the Hermitian conjugate. We say that a matrix $\mathbf{A}$ is normalized if $\|\mathbf{A}\|_F = 1$. The vectorization of $\mathbf{A} \in \mathbb{C}^{m \times n}$, denoted by $\mathbf{vec}(\mathbf{A}) \in \mathbb{C}^{mn}$, is a column vector obtained by stacking the columns of the matrix A on top of one another (i.e., column-major vectorization): $\mathbf{vec}(\mathbf{A}) = [a_{0,0}, \ldots, a_{m-1,0}, a_{01}, \ldots, a_{m-1,1}, \ldots, a_{m-1,n-1}]^{\mathrm{T}}$.

Given matrices $\mathbf{A} \in \mathbb{C}^{m \times n}$ and $\mathbf{B} \in \mathbb{C}^{p \times q}$, their direct sum, denoted by $\mathbf{A} \oplus \mathbf{B}$, is an $(m+p) \times (n+q)$ matrix obtained by putting $\mathbf{A}$ and $\mathbf{B}$ on the diagonal. That is,

$$\mathbf{A} \oplus \mathbf{B} := \begin{bmatrix} \mathbf{A} & 0_{m \times q} \\ 0_{p \times n} & \mathbf{B} \end{bmatrix}.$$

Allowing for matrices in which one of the dimensions has size zero, we can use the $\oplus$ notation to also denote padding by columns or rows:

$$\mathbf{A} \oplus \mathbf{0}_{p \times 0} := \begin{bmatrix} \mathbf{A} \\ \mathbf{0}_{p \times n} \end{bmatrix}, \qquad \mathbf{A} \oplus \mathbf{0}_{0 \times q} := \begin{bmatrix} \mathbf{A} & \mathbf{0}_{m \times q} \end{bmatrix}.$$

Although we can view vectors as matrices in which either the column dimension or the row dimension is 1, we introduce specialized $\oplus$ notations for vectors:

$$\mathbf{x} \oplus \mathbf{y} := \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}, \qquad \mathbf{x}^{\mathrm{T}} \oplus \mathbf{y}^{\mathrm{T}} := \begin{bmatrix} \mathbf{x}^{\mathrm{T}} & \mathbf{y}^{\mathrm{T}} \end{bmatrix}.$$

We rely on context to resolve any ambiguity in the use of $\oplus$.

Given matrices $\mathbf{A} \in \mathbb{C}^{m \times n}$ and $\mathbf{B} \in \mathbb{C}^{p \times q}$, their Kronecker product $\mathbf{A} \otimes \mathbf{B}$ is an $(mp) \times (nq)$ matrix with elements defined by $a_{i,j} b_{k,l}$. A useful representation is

$$\mathbf{A} \otimes \mathbf{B} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \sum_{k=0}^{p-1} \sum_{l=0}^{q-1} a_{i,j} b_{k,l} \mathbf{E}^{mp \times nq}_{pi+k, qj+l},$$

where $\mathbf{E}^{m \times n}_{i,j}$ denotes an $m \times n$ matrix with 1 in the (i,j)th entry and 0 otherwise.

2.2. Matrices and quantum states: Basic notations and definitions. We use the state vector based formulation of quantum computing, i.e., the system's state is represented by a unit vector in a Hilbert space. We assume without loss of generality that the Hilbert space is $\mathbb{C}^n$, where $n = 2^q$ for q qubits. We let $|0\rangle_q, |1\rangle_q, |2\rangle_q, \ldots$ denote the computational basis in a q-qubit system, and $|\phi\rangle_q, |\psi\rangle_q, \ldots$ some abstract states in a q-qubit system. Given two states $|\phi\rangle_q$ and $|\psi\rangle_p$ on q and p qubits (respectively), we denote by $|\psi\rangle_q|\phi\rangle_p$ the $q+p$ qubit state obtained by tensoring the two states to obtain a state in $\mathbb{C}^{mn}$ where $n = 2^q$ and $m = 2^p$. Similar to many physics textbooks, we assume that the MSB is in the lowest index in binary expansions, e.g., $|i\rangle_q = |b_0\rangle_1|b_1\rangle_1 \cdots |b_{q-1}\rangle_1 =: |b_0 \cdots b_{q-1}\rangle_q$, where $b_0 \cdots b_{q-1}$ is the binary expansion of i.

Given an amplitude vector $\alpha \in \mathbb{C}^n$, where n is a power of 2, we use the ket $|\alpha\rangle$ to denote the $\log_2 n$-qubit system's state whose amplitudes are given by α after normalization. That is, $|\alpha\rangle := \frac{1}{\|\alpha\|_2} \sum_{i=0}^{n-1} \alpha_i |i\rangle_{\log_2 n}$, where α_i is the ith entry of α. Note that under these conventions we have $|i\rangle_{\log_2 n} = |\mathbf{e}_i^n\rangle$, where $\mathbf{e}_0^n, \mathbf{e}_1^n, \ldots$ are the n-dimensional identity vectors (when the dimension of the vectors is clear from the context we omit it). We also define $|\overline{\alpha}\rangle := \frac{1}{\|\overline{\alpha}\|_2} \sum_i \overline{\alpha_i} |i\rangle_{\log_2 n}$, where $\overline{\alpha_i}$ is the complex conjugation of $\alpha_i \in \mathbb{C}$. Given two amplitude vectors $\alpha \in \mathbb{C}^n$ and $\beta \in \mathbb{C}^m$, we have $|\alpha\rangle|\beta\rangle = |\alpha \otimes \beta\rangle$ and this extends naturally to a higher number of multiplicands.

We use calligraphic letters to denote both quantum circuits and operators on the state's Hilbert space, e.g., $\mathcal{U}$, $\mathcal{S}$, and $\mathcal{T}$, also using the same letter for a circuit and the operator it induces. For a circuit $\mathcal{U}$ on q qubits, we use $\mathbf{M}(\mathcal{U}) \in \mathbb{C}^{n \times n}$ $(n = 2^q)$ to denote the unique unitary matrix such that for every $\alpha \in \mathbb{C}^n$ applying $\mathcal{U}$ on the state $|\alpha\rangle$ results in the state $|\mathbf{M}(\mathcal{U})\alpha\rangle$. We say that a circuit $\mathcal{U}^*$ is the inverse (or adjoint) of circuit $\mathcal{U}$ if $\mathbf{M}(\mathcal{U}) = \mathbf{M}(\mathcal{U})^*$. Given a quantum system, we denote the application of circuits sequentially with $\cdot$ or omit totally.

A quantum register is defined as a contiguous subset of qubits of a multiqubit system. The size of the register is determined by the number of qubits it encompasses. Given a system with several quantum registers, we use the standard tensor product $\otimes$ operator to denote the application of each circuit on the corresponding register, but also use $\cdot_i$ to denote the action of a circuit $\mathcal{U}$ to only register i of the system, e.g., $\mathcal{U}\cdot_i |\alpha\rangle$ (where the operation on the other register is the identity). We use 0-based indexing for register numbering within a system.

Given a system with two quantum registers, with qubit sizes $q^{(0)} = \log_2 n$ and $q^{(1)} = \log_2 m$, and a matrix $\mathbf{A} \in \mathbb{C}^{m \times n}$, we use $||\mathbf{A}\rangle\rangle$ to denote the state whose amplitude vectors are $\frac{\mathbf{vec}(\mathbf{A})}{\|\mathbf{A}\|_F}$. That is,

$$||\mathbf{A}\rangle\rangle := \frac{1}{\|\mathbf{A}\|_F} \sum_{j=0}^{n-1} \sum_{i=0}^{m-1} a_{i,j} \, |mj + i\rangle_{q^{(0)}+q^{(1)}} = \frac{1}{\|\mathbf{A}\|_F} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} a_{i,j} \, \big|\big|\mathbf{E}_{i,j}^{m \times n}\big\rangle\big\rangle,$$

where a_{ij} denotes the (i,j)th entry of $\mathbf{A}$. Since $\mathbf{E}_{i,j}^{m \times n} = \mathbf{e}_i^m (\mathbf{e}_j^n)^{\mathrm{T}}$ we have $||\mathbf{E}_{i,j}^{m \times n}\rangle\rangle := |\mathbf{e}_j^n\rangle|\mathbf{e}_i^m\rangle = |j\rangle_{q^{(0)}}|i\rangle_{q^{(1)}}$. We use the notation $||\mathbf{A}\rangle\rangle$ (instead of $|\mathbf{vec}(\mathbf{A})\rangle$) to help the reader distinguish state descriptions that are based on matrices as opposed to vectors. We remark that the use of double kets to denote matrix states can be traced to [21].

For a matrix $\mathbf{A} \in \mathbb{C}^{m \times n}$ we denote by $q_{\mathbf{A}}$ the number of qubits we need to hold the state $||\mathbf{A}\rangle\rangle$. That is, $q_{\mathbf{A}} = \log_2 mn$. For a circuit $\mathcal{U}$, we denote $g_{\mathcal{U}}$ the number of gates in $\mathcal{U}$, and by $d_{\mathcal{U}}$ the depth (critical path) of $\mathcal{U}$, and by $q(\mathcal{U})$ the number of qubits of $\mathcal{U}$.

2.3. Overlap between two quantum states. Two foundational quantum algorithms for measuring the overlap (i.e., dot product or similarity) between two

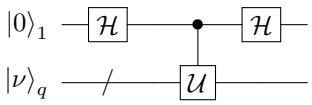

FIG. 1. *Circuit implementing of Hadamard test.*

quantum states are the Hadamard and swap tests. For completeness, we now recall both.

For our purposes, it is useful to describe the Hadamard test as an algorithm that accepts a description of a circuit $\mathcal{U}$ and construct a new circuit that implements the test.

PROPOSITION 1 (Hadamard test). *Given a classical description of the circuit $\mathcal{U}$ on q qubit, there exists an algorithm that constructs a circuit $\mathrm{HT}_{\mathcal{U}}$ on $q+1$ qubits (see Figure 1 for the circuit) such that for any q-qubit state $|\nu\rangle_q$ we have*

$$\mathrm{HT}_{\mathcal{U}}\left(|0\rangle_1|\nu\rangle_q\right)=\frac{1}{2}|0\rangle_1(|\nu\rangle_q+\mathcal{U}|\nu\rangle_q)+\frac{1}{2}|1\rangle_1(|\nu\rangle_q-\mathcal{U}|\nu\rangle_q). \tag{2}$$

The circuit $\mathrm{HT}_{\mathcal{U}}$ serves as an implementation of the Hadamard test. Upon applying the Hadamard test to $|0\rangle_1|\nu\rangle_q$, the probability of obtaining 0 after measurement of the first qubit is

$$\frac{1}{2}(1+\mathbf{Re}(\langle\nu|\mathcal{U}|\nu\rangle).$$

The cost of constructing $\mathrm{HT}_{\mathcal{U}}$ is $O(g_{\mathcal{U}})$ and the depth is $d_{\mathcal{U}}+2$. To obtain the imaginary part, the phase gate $\mathcal{S}=\left[\begin{smallmatrix}1&0\\0&i\end{smallmatrix}\right]$ can be employed. In this complex version, we simply apply $\mathcal{S}^\dagger$ after the initial Hadamard gate, and the probability of obtaining 0 after measurement of the first qubit is

$$\frac{1}{2}(1+\mathbf{Im}(\langle\nu|\mathcal{U}|\nu\rangle)).$$

Remark 2. If we have access to a circuit $\mathcal{U}_\nu$ such that $\mathcal{U}_\nu|0\rangle_q=|\nu\rangle_q$, and a circuit $\mathcal{U}_\eta$ such that $\mathcal{U}_\eta|0\rangle_q=|\eta\rangle_q$, then one can use the Hadamard test for overlap estimation (with relative phase information as well) by the following identity: $\langle\nu|\eta\rangle=\langle 0|\mathcal{U}_\nu^*\mathcal{U}_\eta|0\rangle$, e.g., in that case we have that

$$p(0)=\frac{1}{2}(1+\mathbf{Re}(\langle 0|\mathcal{U}_\nu^*\mathcal{U}_\eta|0\rangle))=\frac{1}{2}(1+\mathbf{Re}(\langle\nu|\eta\rangle)) \tag{3}$$

for the output of $\mathrm{HT}_{\mathcal{U}_\nu^*\mathcal{U}_\eta}$, where $p(0)$ denotes the probability of measuring 0 in the first qubit (when only that qubit is measured).

PROPOSITION 3 (swap test). *For q qubits, there exists a circuit ST_q on $2q+1$ qubits (see Figure 2 for the circuit) such that for any two q-qubit states $|\psi\rangle_q$ and $|\phi\rangle_q$ we have*

$$\mathrm{ST}_q\left(|0\rangle_1|\psi\rangle_q|\phi\rangle_q\right)=\frac{1}{2}|0\rangle_1\left(|\psi\rangle_q|\phi\rangle_q+|\phi\rangle_q|\psi\rangle_q\right)+\frac{1}{2}|1\rangle_1\left(|\psi\rangle_q|\phi\rangle_q-|\phi\rangle_q|\psi\rangle_q\right).$$

The circuit ST_q serves as an implementation of the swap test. Upon applying the swap test to $|0\rangle_1|\psi\rangle_q|\phi\rangle_q$, the probability of obtaining 0 after measurement of the first qubit is

$$\frac{1}{2}(1+|\langle\phi|\psi\rangle|^2).$$

The cost of constructing ST_q is $q+2$ and the depth is 3.

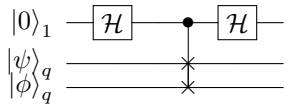

FIG. 2. *Circuit implementing of swap test.*

3. Quantum matrix state linear algebra (qMSLA). Since quantum circuits essentially evolve quantum states by applying unitary transformations on them, one can envision performing matrix computations using quantum circuits. Quantum speedups on numerical linear algebra tasks will immediately lead to faster algorithms in many scientific computing and machine learning tasks, since at the core, such algorithms rely heavily on matrix computations [7]. Indeed, quite a few quantum algorithms for scientific computing and machine learning have been proposed recently, many of them relying on QLA breakthroughs such as the HHL algorithm [35] and others [17, 28, 64, 2, 45].

All the aforementioned works make assumptions on the mechanism in which the input is presented to the quantum algorithm, and that mechanism is of crucial importance [13]. We refer to the mechanism in which input is presented to the quantum algorithm as the *input model* of the quantum algorithm. Several input models have been presented in the literature. Two prominent examples are the sparse-data access model [1, 17] and various quantum data structure based input models [39, 40]. Recently, Chakraborty, Gilyén, and Jeffery [13] showed that a variety of widely used input models can be reduced to an input model in which matrices are input using *block encodings* and vectors are input as *state preparation circuits*.

DEFINITION 4 (state preparation circuit). *We say that a* $\log_2 n$*-qubit circuit* $\mathcal{U}$ *is a* state preparation circuit *for a vector* $\mathbf{x} \in \mathbb{C}^n$ *if applying* $\mathcal{U}$ *to the state* $|0\rangle_{\log_2 n}$ *results in the state* $|\mathbf{x}\rangle$.

DEFINITION 5 (block encoding of a matrix [28]). *For* $\alpha \geq \|\mathbf{A}\|_F$*, a circuit* $\mathcal{U}$ *is a* α*-block encoding of* $\mathbf{A} \in \mathbb{C}^{m\times n}$ *if*

$$\alpha \mathbf{M}(\mathcal{U}) = \begin{bmatrix} \mathbf{A} & * \\ * & * \end{bmatrix},$$

where $*$ *denotes arbitrary entries. We refer to* α *as the* scale.

We refer to the input model in which matrices are accessed using block encodings and vectors are accessed as state preparation circuits as the *block encoding input model*. There are powerful algorithms that operate under the block encoding model. In particular, in the block encoding model we can perform QSVT [28], a powerful technique that leads to efficient algorithms for solving linear equations, amplitude amplification, quantum simulation, and more [48].

In this paper we do not assume matrices are given as block encodings, and instead consider an input model in which *both* matrices and vectors are input as state preparation circuits; for matrices we use a *matrix state preparation circuit.*

DEFINITION 6 (matrix state preparation circuit). *We say that a* $(\log_2 n + \log_2 m)$*-qubit circuit* $\mathcal{U}$ *is a* matrix state preparation circuit *for a matrix* $\mathbf{A} \in \mathbb{C}^{m\times n}$ *if applying* $\mathcal{U}$ *to the state* $|0\rangle_{\log_2 mn}$ *results in the state* $||\mathbf{A}\rangle\rangle$*. Equivalently, the first column of* $\mathbf{M}(\mathcal{U})$ *is* $\mathbf{vec}(\mathbf{A})$*. For convenience, where appropriate, we add the matrix as subindex when denoting state preparation circuits, e.g.,* $\mathcal{U}_{\mathbf{A}}$*. In such cases, with an abuse of notation, the number of gates in* $\mathcal{U}_{\mathbf{A}}$ *is denoted by* $g_{\mathbf{A}}$*, and the depth by* $d_{\mathbf{A}}$.

Remark 7. A matrix state preparation circuit $\mathcal{U}_{\mathbf{A}}$ should be viewed as a data structure bundling the actual circuit, along with meta-data that describes the size of the resulting matrix (i.e., m and n). The meta-data in essence describes the partition of the qubits on which the circuit operates into two registers, one corresponding to column indices (MSB qubits, "top" qubits in circuit visualizations) and the other corresponding to row indices (LSB qubits, "bottom" qubits in circuit visualizations). We refer to the qubits that correspond to the column indices as *column qubits,* and the register that corresponds to these indices as the *column register,* and likewise for *row qubits* and *row register.*

Remark 8. It is also possible to track classically the Frobenius norm of the matrix (and include it in the meta-data), though this is not necessary for the purpose of multivariate trace estimation.

Remark 9. A vector is also a matrix where one of the dimensions has size 1. So, our definition of a matrix state preparation circuit (Definition 6) subsumes vector state preparation circuits (Definition 4). In vector state preparation circuits one of the two registers has zero size: for a column vector we have only row qubits, and for a row vector we have only column qubits.

We refer to the input model in which both matrices and vectors are accessed via state preparation circuits as the *state preparation input model.* Are the block encoding model and the state preparation model equivalent? We do not have a complete answer to this question, though we do have some partial observations. First, given a matrix state preparation circuit for $\mathbf{A}$ and a state preparation circuit for a vector $\mathbf{w}$ whose entries are the row norms of $\mathbf{A}$, it is possible to construct a block encoding of $\mathbf{A}$ [18, section I.D]. We are unaware of any efficient algorithm that given *only* a matrix state preparation circuit for $\mathbf{A}$ constructs a block encoding of $\mathbf{A}$. Second, in section 3.2.2 we show that given a circuit $\mathcal{U}$ we can construct a matrix state preparation of $\mathbf{M}(\mathcal{U})$. Thus, given a block encoding of $\mathbf{A}$ we can construct a matrix state preparation circuit for a matrix that contains $\mathbf{A}$. Both observations together lead us to conjecture that the state preparation model uses less stringent assumptions than the block encoding model, though we leave formalizing this conjecture and proving its correctness to future work.

There is one additional way in which the state preparation model has an advantage over the block encoding model. The value of the scale parameter α of the block encoding is important to downstream efficiency. Roughly speaking, the smaller α is, shallower circuits suffice or less shots are required. Recalling that the minimum value for α for a block encoding of $\mathbf{A}$ is $\|\mathbf{A}\|_F$, we note that it is impossible to construct a block encoding with the minimum α unless $\mathbf{A}$ is unitary. In contrast, for *any* $m \times n$ matrix it is possible to construct an $O(mn)$ depth matrix state preparation circuit in $O(mn)$ (classical) time, without any loss [62].

In this paper we show how to estimate multivariate traces and spectral sums in the state preparation model. The main challenge is that in the state preparation model we do not have access to powerful tools like QSVT. Our approach is based on showing that given classical descriptions of state preparation circuits, we can construct more complex state preparation circuits that implement several important matrix algebra operations, thus providing a toolbox for performing matrix algebra in the state preparation model. We refer to our framework as "quantum Matrix State Linear Algebra" (qMSLA). Using this toolbox, we can encode multivariate traces in the state preparation circuit via simple matrix identities. In this section we outline qMSLA, while in the next section (section 4) we use qMSLA to propose a quantum algorithm for multivariate trace estimation.

TABLE 1

Level 1 *qMSLA operations. These are primitive operations that create or operate on state preparation circuits using low-level circuit building tools. The classical cost for these operations is consistently* $O(g(output))$.

Input	Output	q(output)	g(output)	d(output)	Operation name	Subsection
$n = 2^q$	$\mathcal{U}_{\mathbf{I}_n}$	$2q$	$2q$	2	qmsla.identity	3.2.1
$\mathcal{U}$	$\mathcal{U}_{\mathbf{M}(\mathcal{U})}$	$2q(\mathcal{U})$	$g(\mathcal{U})+2q$	$d(\mathcal{U})+2$	qmsla.matrix	3.2.2
$\mathcal{U}_{\mathbf{A}}$	$\mathcal{U}_{\overline{\mathbf{A}}}$	$q_{\mathbf{A}}$	$g_{\mathbf{A}}$	$d_{\mathbf{A}}$	qmsla.conjugate	3.2.3
$\mathcal{U}_{\mathbf{A}}$	$\mathcal{U}_{\mathbf{A}^{\mathrm{T}}}$	$q_{\mathbf{A}}$	$g_{\mathbf{A}}$	$d_{\mathbf{A}}$	qmsla.transpose	3.2.4
$\mathcal{U}_{\mathbf{A}}$	$\mathcal{U}_{\mathbf{vec}(\mathbf{A})}$	$q_{\mathbf{A}}$	$g_{\mathbf{A}}$	$d_{\mathbf{A}}$	qmsla.vec	3.2.5
$\mathcal{U}_{\mathbf{A}}, k$	$\mathcal{U}_{\mathbf{A}\oplus 0_{0\times(2^k-1)n}}$	$q_{\mathbf{A}}+k$	$g_{\mathbf{A}}$	$d_{\mathbf{A}}$	qmsla.pad_zero_columns	3.2.6
$\mathcal{U}_{\mathbf{A}},\mathcal{U}_{\mathbf{b}}$	$\mathcal{U}_{\frac{\mathbf{A}\mathbf{b}}{\|\mathbf{A}\|_F\|\mathbf{b}\|_2}\oplus\xi}$	$q_{\mathbf{A}}$	$g_{\mathbf{A}}+g_{\mathbf{b}}$	$d_{\mathbf{A}}+d_{\mathbf{b}}$	qmsla.matrix_vec	3.2.7
$\mathcal{U}_{\mathbf{A}},\mathcal{U}_{\mathbf{B}}$	$\mathcal{U}_{\mathbf{A}\otimes\mathbf{B}}$	$q_{\mathbf{A}}+q_{\mathbf{B}}$	$g_{\mathbf{A}}+g_{\mathbf{B}}$	$\max(d_{\mathbf{A}}, d_{\mathbf{B}})$	qmsla.kronecker	3.2.8
$\mathcal{U}_{\mathbf{A}_1},\dots,\mathcal{U}_{\mathbf{A}_k}$	$\mathcal{U}_{\mathbf{A}_1\otimes\mathbf{A}_2\otimes\cdots\otimes\mathbf{A}_k}$	$\sum_i q_{\mathbf{A}_i}$	$\sum_i g_{\mathbf{A}_i}$	$\max(d_{\mathbf{A}_1},\cdots,d_{\mathbf{A}_k})$	qmsla.kronecker	3.2.9

TABLE 2

Level 2 *qMSLA operations. These are composite operations built using level* 1 *operations. The classical cost for these operations is consistently* $O(g(output))$.

Input	Output	q(output)	g(output)	d(output)	Operation name	Subsection
$\mathcal{U}_{\mathbf{A}}, r$	$\mathcal{U}_{\mathbf{A}\oplus 0_{(2^r-1)m\times 0}}$	$q_{\mathbf{A}}+r$	$g_{\mathbf{A}}$	$d_{\mathbf{A}}$	qmsla.pad_zero_rows	3.3.1
$\mathcal{U}_{\mathbf{A}}, k, r$	$\mathcal{U}_{\mathbf{A}\oplus 0_{(2^r-1)m\times(2^k-1)n}}$	$q_{\mathbf{A}}+k+r$	$g_{\mathbf{A}}$	$d_{\mathbf{A}}$	qmsla.pad	3.3.1
$\mathcal{U}_{\mathbf{A}}$	$\mathcal{U}_{\mathbf{A}^*}$	$q_{\mathbf{A}}$	$g_{\mathbf{A}}$	$d_{\mathbf{A}}$	qmsla.adjoint	3.3.2
$\mathcal{U}_{\psi},\mathcal{U}_{\phi}$	$\mathcal{U}_{\frac{\psi^*\phi}{\|\psi\|_2\|\phi\|_2}\oplus\xi}$	q_{ψ}	$g_{\psi}+g_{\phi}$	$d_{\psi}+d_{\phi}$	qmsla.overlap	3.3.3

In the following subsections we present various qMSLA operations, along with discussion of their complexity. With few exceptions, each operation receives a variable number of classical descriptions of a matrix state preparation circuit, and outputs a classical description of a new state preparation circuit that implements some matrix algebra operation between the input matrices. Thus, the algorithms are classical-to-classical, but provide circuits to be executed on a quantum computer. For each operation, we provide visualization of the output circuit and analyze the gate complexity and depth of the output circuit and the (classical) cost of forming the output circuit.

We split qMSLA into two levels. Level 1 includes primitive operations that create or operate on state preparation circuits using low-level circuit building procedures, while level 2 includes composite operations built using level 1 primitive operations. Both levels are summarized in Tables 1 and 2.

In this section we present a high-level description on how to implement various qMSLA operations. Low-level implementation details are deferred to the appendix. We also introduce in the appendix another qMSLA level: level 0. Level 0 operates on the circuit level and does not deal with matrix states at all. In essence, these are the basic circuit building operations we require from an underlying quantum computing framework (such as QISKIT) so that we can build qMSLA on top of that framework. Most of the operations are implemented in various frameworks, but some are not, so we explain how they can be implemented using only the ability to add and remove gates. We then show how to implement level 1 operations using only level 0 operations.

Henceforth in this section, unless otherwise specified, we assume that $\mathbf{A} \in \mathbb{C}^{m\times n}$, $\mathcal{U}, \mathcal{W}, \mathcal{Q}$ are quantum circuits, σ is a permutation function on all qubits, $\mathcal{S}_\sigma$ denotes an implementation of a permutation σ using SWAP gates, and $\xi \in \mathbb{C}^{m(n-1)}$ represents auxiliary information (garbage), which may be arbitrary.

3.1. Prelude: Operating on a subset of the qubits and qubit permutations. In this subsection, we investigate quantum circuit operations targeting specific subsets of qubits (i.e., registers), qubit permutation procedures, circuit composition, and an efficient method for eliminating SWAP gates in state preparation circuits. Understanding each of these operations is crucial for the implementation of various qMSLA operations.

3.1.1. Operating on single registers of a matrix state. Consider a matrix state $||\mathbf{A}\rangle\rangle$ for $\mathbf{A} \in \mathbb{C}^{m\times n}$. qMSLA is based on three types of operations: operating on the column register alone, operating on the row register alone, and rearranging the qubits between registers (using SWAP gates). In this subsection we seek to understand how the first two evolve matrix states.

First, let us consider how the state evolves when we apply a circuit only on one of the two registers. Looking at the product states of the two registers, assuming that $\alpha \in \mathbb{C}^n$ and $\beta \in \mathbb{C}^m$, we have the following immediate identities:

1. $\mathcal{U} \cdot |\alpha\rangle|\beta\rangle$ results in the state $|\mathbf{M}(\mathcal{U})(\alpha \otimes \beta)\rangle$.
2. $\mathcal{Q}_{\cdot 0}\,|\alpha\rangle|\beta\rangle = (\mathcal{Q} \otimes \mathcal{I}_m)|\alpha\rangle|\beta\rangle$ results in the state $|\mathbf{M}(\mathcal{Q})\alpha\rangle|\beta\rangle$.
3. $\mathcal{W}_{\cdot 1}\,|\alpha\rangle|\beta\rangle = (\mathcal{I}_n \otimes \mathcal{W})|\alpha\rangle|\beta\rangle$ results in the state $|\alpha\rangle|\mathbf{M}(\mathcal{W})\beta\rangle$.

Using these identities, we prove the following lemma.

LEMMA 10. *Consider the state* $||\mathbf{A}\rangle\rangle$ *for* $\mathbf{A} \in \mathbb{C}^{m\times n}$*, and circuits* $\mathcal{U}$ *and* $\mathcal{Q}$ *where* $\mathcal{U}$ *operates on* $\log_2 n$ *qubits and* $\mathcal{Q}$ *operates on* $\log_2 m$ *qubits. The following holds:*

1. $\mathcal{U}_{\cdot 0}\,||\mathbf{A}\rangle\rangle$ *results in the state* $||\mathbf{A}\mathbf{M}(\mathcal{U})^{\mathrm{T}}\rangle\rangle$.
2. $\mathcal{Q}_{\cdot 1}\,||\mathbf{A}\rangle\rangle$ *results in the state* $||\mathbf{M}(\mathcal{Q})\mathbf{A}\rangle\rangle$.

Proof. We use the following identity: $\mathbf{vec}(\mathbf{ABC}) = (\mathbf{C}^{\mathrm{T}} \otimes \mathbf{A})\mathbf{vec}(\mathbf{B})$) [6, Proposition 7.1.9]. We have

$$\begin{aligned}\mathcal{U}_{\cdot 0}\,||\mathbf{A}\rangle\rangle = (\mathcal{U}\otimes\mathcal{I})\,|\mathbf{vec}\,(\mathbf{A})\rangle &= |\mathbf{M}(\mathcal{U}\otimes\mathcal{I})\mathbf{vec}\,(\mathbf{A})\rangle \\ &= |(\mathbf{M}(\mathcal{U})\otimes\mathbf{I})\mathbf{vec}\,(\mathbf{A})\rangle \\ &= \left|\mathbf{vec}\left(\mathbf{I}\mathbf{A}\mathbf{M}(\mathcal{U})^{\mathrm{T}}\right)\right\rangle \\ &= \left|\left|\mathbf{A}\mathbf{M}(\mathcal{U})^{\mathrm{T}}\right\rangle\right\rangle\end{aligned}$$

and

$$\begin{aligned}\mathcal{Q}_{\cdot 1}\,||\mathbf{A}\rangle\rangle = (\mathcal{I}\otimes\mathcal{Q})\,|\mathbf{vec}\,(\mathbf{A})\rangle &= |\mathbf{M}(\mathcal{I}\otimes\mathcal{Q})\mathbf{vec}\,(\mathbf{A})\rangle \\ &= |(\mathbf{I}\otimes\mathbf{M}(\mathcal{Q}))\mathbf{vec}\,(\mathbf{A})\rangle \\ &= |\mathbf{vec}\,(\mathbf{M}(\mathcal{Q})\mathbf{A}\mathbf{I})\rangle \\ &= ||\mathbf{M}(\mathcal{Q})\mathbf{A}\rangle\rangle\,.\end{aligned}$$

∎

3.1.2. Permutation of qubits and registers. Let q be the number of qubits in a system, and consider a permutation $\sigma : \{0,\ldots,q-1\} \to \{0,\ldots,q-1\}$. Denote by $\mathcal{S}_\sigma$ the unitary that permutes the qubits according to σ. In the computational basis we have

$$\mathcal{S}_\sigma\,|b_0\cdots b_{q-1}\rangle_q = \left|b_{\sigma(0)}\cdots b_{\sigma(q-1)}\right\rangle_q,$$

where $b_0\cdots b_{q-1}$ is the binary expansion of the index of a basis state. A circuit implementing $\mathcal{S}_\sigma$ can be built using SWAP gates. A simple algorithm for forming $\mathcal{S}_\sigma$ is the following. Given the sequence $(\sigma(0),\ldots,\sigma(q-1))$, find a sequence of swaps that sort the sequence (e.g., by tracking the swaps of a sorting algorithm). That is,

find $(i_1, j_1), \ldots, (i_T, j_T)$ such that $\sigma_{(i_T,j_T)} \circ \cdots \circ \sigma_{(i_1,j_1)} \circ \sigma = \sigma_I$, where $\sigma_{(i,j)}$ is the permutation that swaps the entry i and j, and σ_I is the identity permutation. Let $\mathcal{S}_{(i,j)}$ denote the circuit which swaps qubits i and j (via a single SWAP gate), but keeps other qubits in place. Now, $\mathcal{S}_{(i_T,j_T)} \cdots \mathcal{S}_{(i_1,j_1)} \mathcal{S}_\sigma = \mathcal{I}$. By applying the inverse of these swap operations from the left, we obtain an implementation of $\mathcal{S}_\sigma$ as $\mathcal{S}_{(i_1,j_1)} \cdots \mathcal{S}_{(i_T,j_T)}$. The set of swaps can be found using a sorting algorithm in $O(q \log q)$ operations. This is also the bound on the number of gates in $\mathcal{S}_\sigma$, and on the depth.

However, our algorithms typically permute full registers rather than individual bits. In such cases, to distinguish register permutations from qubit permutations, we use the bar notation $\bar{\sigma}$ to indicate that the permutation is to be applied to registers. It is assumed to have signature $\bar{\sigma} : \{0, \ldots, r-1\} \to \{0, \ldots, r-1\}$ for r registers. In particular, suppose we have r registers, of sizes $q_0, \ldots, q_{r-1}$, and consider a permutation $\bar{\sigma} : \{0, \ldots, r-1\} \to \{0, \ldots, r-1\}$; then defining the circuit $\mathcal{S}^R_{\bar{\sigma}}$ as the circuit on $\sum_{i=0}^{m-1} q_i$ qubits for which, for all states $|\psi_0\rangle_{q_0}, \ldots, |\psi_{r-1}\rangle_{q_{r-1}}$ we have

$$\mathcal{S}^R_{\bar{\sigma}} \cdot |\psi_0\rangle_{q_0} \cdots |\psi_{r-1}\rangle_{q_{r-1}} = \left|\psi_{\bar{\sigma}(0)}\right\rangle_{q_{\bar{\sigma}(0)}} \cdots \left|\psi_{\bar{\sigma}(r-1)}\right\rangle_{q_{\bar{\sigma}(r-1)}}.$$

Example 11. Given the $(\log_2 m + \log_2 n)$-qubit system and a permutation $\bar{\sigma}_{\mathrm{T}} = (1, 0)$ (i.e., swap the order of the two registers) we have that

$$\mathcal{S}^R_{\bar{\sigma}_{\mathrm{T}}} |\phi\rangle_{\log_2 n} |\psi\rangle_{\log_2 m} = |\psi\rangle_{\log_2 m} |\phi\rangle_{\log_2 n}.$$

We show in section 3.2.4 that $\mathcal{S}^R_{\bar{\sigma}_{\mathrm{T}}}$ applied to $||\mathbf{A}\rangle\rangle$ results in $\left|\left|\mathbf{A}^{\mathrm{T}}\right\rangle\right\rangle$.

3.1.3. Eliminating permutations in state preparation circuits. In the subsequent sections, various qMSLA operations are described, and from these descriptions, a common structure emerges in the output of straightforward implementations. By observing this structure, it is possible to operate on the resulting circuit $\mathcal{U}_{\mathbf{X}}$ so as to build an alternative circuit $\mathcal{U}'_{\mathbf{X}}$ which is also a matrix state preparation for $\mathbf{X}$ (even though it might be the case that $\mathbf{M}(\mathcal{U}_{\mathbf{X}}) \neq \mathbf{M}(\mathcal{U}'_{\mathbf{X}})$), but is much more efficient since it removes many SWAP gates. Details appear in Appendix A.4.

We call the elimination process EliminatePermutations. In translating the high-level descriptions of sections 3.2 and 3.3 to low-level pseudocode descriptions, we implicitly use EliminatePermutations as a meta-algorithm that transforms the output circuit to more efficient ones. The high-level descriptions of sections 3.2 and 3.3 do not use EliminatePermutations. However, the circuit diagrams do show the boundaries on which EliminatePermutations is applied, and various complexity statements assume that SWAP gates have been eliminated using EliminatePermutations.

3.2. Level-1 qMSLA operations. Level 1 operations create or operate on state preparation circuits using low-level circuit building operations. For more details on implementation, including pseudocode for all level 1 operations, see Appendix B.

3.2.1. Preparing an identity matrix: $\mathcal{U}_{\mathbf{I}_n} \leftarrow$ qmsla.identity(n). Given $2q$-qubits, split between two q-qubit registers, consider the maximally entangled state

$$|\psi\rangle_{2q} = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} |k\rangle_q \otimes |k\rangle_q,$$

where $n = 2^q$. We have

$$|\psi\rangle_{2q} = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} |k\rangle_q \otimes |k\rangle_q = \frac{1}{\sqrt{n}} \sum_{n=0}^{n-1} \left|\left|\mathbf{e}_k \mathbf{e}_k^{\mathrm{T}}\right\rangle\right\rangle = ||\mathbf{I}_n\rangle\rangle.$$

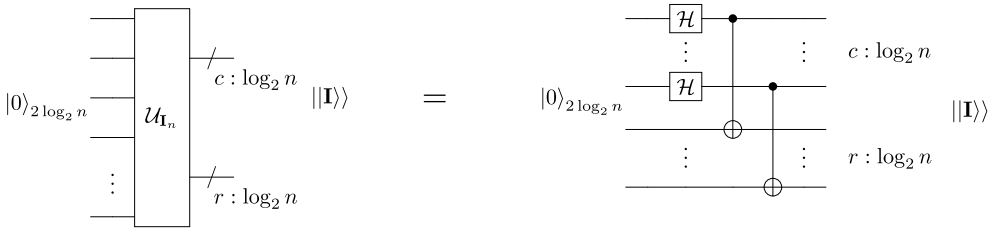

FIG. 3. *Graphical description of qmsla.identity (input: n; output: $\mathcal{U}_{\mathbf{I}_n}$).*

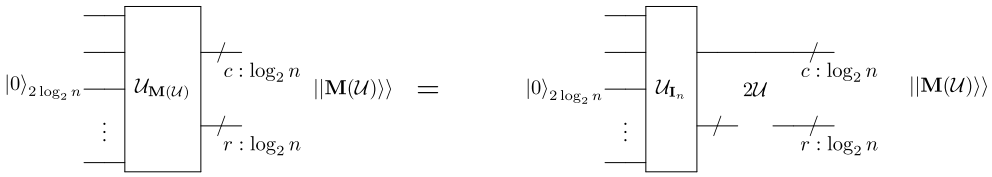

FIG. 4. *Graphical description of qmsla.matrix (Input: $\mathcal{U}$, Output: $\mathcal{U}_{\mathbf{M}(\mathcal{U})}$).*

Thus, to implement a state preparation circuit $\mathcal{U}_{\mathbf{I}_n}$ for $\mathbf{I}_n$ we need to find a circuit that given $|0\rangle_{2q}$ creates the maximally entangled state $|\psi\rangle_{2q}$. It is well known that the circuit depicted in Figure 3 prepares this state. Thus, $g_{\mathbf{I}_n} = 2q$ and $d_{\mathbf{I}_n} = 2$.

3.2.2. State preparation for $\mathbf{M}(\mathcal{U})$: $\mathcal{U}_{\mathbf{M}(\mathcal{U})} \leftarrow$ qmsla.matrix($\mathcal{U}$). Consider a circuit $\mathcal{U}$ on q qubits and denote $n = 2^q$. Due to Lemma 10 we have

$$\begin{aligned}\mathcal{U}_{\cdot_1}\mathcal{U}_{\mathbf{I}_n}|0\rangle_{2q} &= \mathcal{U}_{\cdot_1}||\mathbf{I}_n\rangle\rangle \\ &= ||\mathbf{M}(\mathcal{U})\mathbf{I}_n\rangle\rangle \\ &= ||\mathbf{M}(\mathcal{U})\rangle\rangle\,.\end{aligned}$$

Thus, $\mathcal{U}_{\cdot_1}\mathcal{U}_{\mathbf{I}_n}$ is a state preparation circuit for $\mathbf{M}(\mathcal{U})$. See Figure 4 for a graphical description. We have $g_{\mathbf{M}(\mathcal{U})} = g(\mathcal{U}) + 2q$ and $d_{\mathbf{M}(\mathcal{U})} = d(\mathcal{U}) + 2$. The complexity of forming $\mathcal{U}_{\mathbf{M}(\mathcal{U})}$ is $O(g_{\mathbf{M}(\mathcal{U})})$. The idea of applying a circuit to one share of a maximally entangled vector in order to encode the circuit's matrix in the amplitudes also appears in [50].

In a sense, the ability to build $\mathcal{U}_{\mathbf{M}(\mathcal{U})}$ from a circuit $\mathcal{U}$ hints that usage of the state preparation model constitutes making weaker assumptions than the ones made when using the block encoding model, since block encodings can be converted to state preparation circuits.

3.2.3. Matrix conjugate: $\mathcal{U}_{\overline{\mathbf{A}}} \leftarrow$ qmsla.conjugate($\mathcal{U}_{\mathbf{A}}$). Given a state preparation circuit for a matrix $\mathbf{A} \in \mathbb{C}^{m\times n}$, we can construct a state preparation circuit for $\overline{\mathbf{A}}$ by conjugating the circuit itself. This is shown in the following proposition.

PROPOSITION 12. *For a matrix state preparation circuit $\mathcal{U}_{\mathbf{A}}$, we have*

$$\overline{\mathcal{U}_{\mathbf{A}}} = \mathcal{U}_{\overline{\mathbf{A}}}.$$

Thus, given $\mathcal{U}_{\mathbf{A}}$ we can compute a description of $\mathcal{U}_{\overline{\mathbf{A}}}$ with depth $d_{\mathbf{A}}$ in cost $O(g_{\mathbf{A}})$.

Proof. Recall that the first column of $\mathbf{M}(\mathcal{U}_{\mathbf{A}})$ is $\mathbf{vec}(\mathbf{A})$, so

$$\begin{aligned}\mathbf{M}(\overline{\mathcal{U}_{\mathbf{A}}}) &= \overline{\mathbf{M}(\mathcal{U}_{\mathbf{A}})} \\ &= \overline{\begin{bmatrix} | & * & \cdots & * \\ \mathbf{vec}(\mathbf{A}) & \vdots & & \vdots \\ | & * & \cdots & * \end{bmatrix}} \\ &= \begin{bmatrix} | & * & \cdots & * \\ \mathbf{vec}(\overline{\mathbf{A}}) & \vdots & & \vdots \\ | & * & \cdots & * \end{bmatrix} \\ &= \mathbf{M}(\mathcal{U}_{\overline{\mathbf{A}}}).\end{aligned}$$

∎

3.2.4. Matrix transpose: $\mathcal{U}_{\mathbf{A}^{\mathrm{T}}} \leftarrow$ qmsla.transpose($\mathcal{U}_{\mathbf{A}}$). Given a matrix state preparation circuit for a matrix $\mathbf{A} \in \mathbb{C}^{m\times n}$, we can construct a matrix state preparation circuit for $\mathbf{A}^{\mathrm{T}}$ based on the following proposition. See Figure 5 for a graphical description. It is easy to see then that $g_{\mathbf{A}^{\mathrm{T}}} = g_{\mathbf{A}}$ and $d_{\mathbf{A}^{\mathrm{T}}} = d_{\mathbf{A}}$, if EliminatePermutations is used to reduce the gate complexity and depth. The cost of qmsla.transpose is $O(g_{\mathbf{A}^{\mathrm{T}}})$.

Proposition 13. *Given* $(\log_2 n + \log_2 m)$*-qubit* state preparation circuit $\mathcal{U}_{\mathbf{A}}$*, the circuit* $\mathcal{S}^R_{(1,0)} \cdot \mathcal{U}_{\mathbf{A}}$ *is a state preparation circuit for* $\mathbf{A}^{\mathrm{T}}$*.*

Proof. Denote $\bar{\sigma}_{\mathrm{T}} = (1,0)$. We have that

$$\begin{aligned}\mathcal{S}^R_{\bar{\sigma}_{\mathrm{T}}} \cdot \mathcal{U}_{\mathbf{A}} |0\rangle_{\log_2 mn} &= \mathcal{S}^R_{\bar{\sigma}_{\mathrm{T}}} ||\mathbf{A}\rangle\rangle \\ &= \mathcal{S}^R_{\bar{\sigma}_{\mathrm{T}}} \left(\frac{1}{\|\mathbf{A}\|_F} \sum_{j=0}^{n-1} \sum_{i=0}^{m-1} a_{i,j} |mj+i\rangle_{\log_2 mn} \right) \\ &= \mathcal{S}^R_{\bar{\sigma}_{\mathrm{T}}} \left(\frac{1}{\|\mathbf{A}\|_F} \sum_{j=0}^{n-1} \sum_{i=0}^{m-1} a_{i,j} |j\rangle_{\log_2 n} |i\rangle_{\log_2 m} \right) \\ &= \frac{1}{\|\mathbf{A}^{\mathrm{T}}\|_F} \sum_{j=0}^{n-1} \sum_{i=0}^{m-1} a_{i,j} |i\rangle_{\log_2 m} |j\rangle_{\log_2 n} \\ &= \frac{1}{\|\mathbf{A}^{\mathrm{T}}\|_F} \sum_{j=0}^{n-1} \sum_{i=0}^{m-1} a_{i,j} |ni+j\rangle_{\log_2 mn} \\ &= \left|\left|\mathbf{A}^{\mathrm{T}}\right\rangle\right\rangle.\end{aligned}$$

So, we see that $\mathcal{S}^R_{(1,0)} \cdot \mathcal{U}_{\mathbf{A}}$ is a state preparation circuit for $\mathbf{A}^{\mathrm{T}}$. ∎

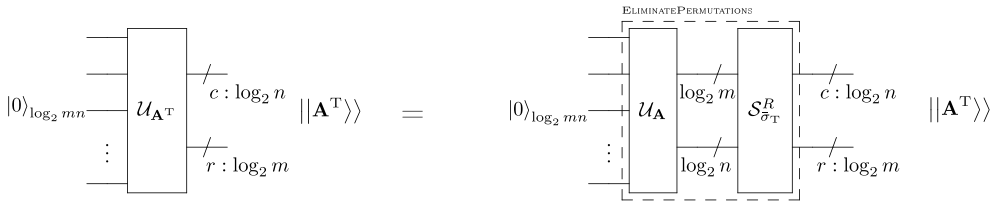

Fig. 5. *Graphical description of qmsla.transpose (input:* $\mathcal{U}_{\mathbf{A}}$*; output:* $\mathcal{U}_{\mathbf{A}^{\mathrm{T}}}$*).*

3.2.5. Vectorize: $\mathcal{U}_{\mathbf{vec(A)}} \leftarrow$ qmsla.vec($\mathcal{U}_{\mathbf{A}}$). Notice that $||\mathbf{A}\rangle\rangle = ||\mathbf{vec}(\mathbf{A})\rangle\rangle$ when considering only probability amplitudes. Indeed, each state is obtained by a different logical split of the qubits between the two registers. In other words, the matrix state preparation circuit $\mathcal{U}_{\mathbf{A}}$ can be interpreted also as $\mathcal{U}_{\mathbf{vec}(\mathbf{A})}$, where the circuit itself is the same, but the meta-data (partition of the qubits to registers) is different. Thus, qmsla.vec can be implemented via meta-data transformation. Gate complexity and depth do not change, and the cost of the operation is $O(g_{\mathbf{A}})$.

3.2.6. Pad with zero columns: $\mathcal{U}_{\mathbf{A}\oplus\mathbf{0}_{0\times(2^k-1)n}} \leftarrow$ qmsla.pad_zero_columns($\mathcal{U}_{\mathbf{A}}, k$) ($k > 0$). Consider a state preparation circuits $\mathcal{U}_{\mathbf{A}}$, where $\mathbf{A} \in \mathbb{C}^{m\times n}$. Given an integer $k > 0$, the following identity shows how to construct a matrix state preparation circuit for $[\ \mathbf{A} \quad \mathbf{0}_{m\times(2^k-1)n}\ ]$:

$$\begin{aligned}\mathcal{I}_k \otimes \mathcal{U}_{\mathbf{A}} \left|0\right\rangle_k \left|0\right\rangle_{\log_2 mn} &= \left|0\right\rangle_k ||\mathbf{A}\rangle\rangle \\ &= \left|\left|\left[\ \mathbf{A} \quad \mathbf{0}\ \right]\right\rangle\right\rangle,\end{aligned}$$

where $\mathcal{I}_k$ is the k-qubit empty (identity) circuit. So, $\mathcal{U}_{\mathbf{A}\oplus 0_{0\times(2^k-1)n}} = \mathcal{I}_k \otimes \mathcal{U}_{\mathbf{A}}$. See Figure 6 for a graphical description of qmsla.pad_zero_columns. It is easy to see then that $g_{\mathbf{A}\oplus 0_{0\times(2^k-1)n}} = g_{\mathbf{A}}$ and $d_{\mathbf{A}\oplus 0_{0\times(2^k-1)n}} = d_{\mathbf{A}}$ and the cost of the algorithm is $O(g_{\mathbf{A}})$.

3.2.7. Matrix-vector product: $\mathcal{U}_{\mathbf{y}} \leftarrow$ qmsla.matrix_vec($\mathcal{U}_{\mathbf{A}}, \mathcal{U}_{\mathbf{b}}$) where $\mathbf{y} = \frac{\mathbf{Ab}}{\|\mathbf{A}\|_F\|\mathbf{b}\|_2} \oplus \boldsymbol{\xi} \in \mathbb{C}^{mn}$ for some garbage $\boldsymbol{\xi} \in \mathbb{C}^{m(n-1)}$ and $\|\mathbf{y}\|_2 = 1$. Assume that $\mathbf{A} \in \mathbb{C}^{m\times n}$ and $\mathbf{b} \in \mathbb{C}^n$. Consider applying $\mathcal{U}_{\mathbf{b}}^{\mathrm{T}}$ to the MSB register (i.e., column register) of the state $||\mathbf{A}\rangle\rangle$. We have (Lemma 10)

$$\begin{aligned}\mathcal{U}_{\mathbf{b}}^{\mathrm{T}} \cdot_0 ||\mathbf{A}\rangle\rangle &= \left|\left|\mathbf{A}\mathbf{M}(\mathcal{U}_{\mathbf{b}}^{\mathrm{T}})^{\mathrm{T}}\right\rangle\right\rangle \\ &= ||\mathbf{A}\mathbf{M}(\mathcal{U}_{\mathbf{b}})\rangle\rangle \\ &= \left|\left|\mathbf{A}\begin{bmatrix} | & * & \cdots & * \\ \frac{\mathbf{b}}{\|\mathbf{b}\|_2} & \vdots & & \vdots \\ | & * & \cdots & * \end{bmatrix}\right\rangle\right\rangle \\ &= \left|\left|\begin{bmatrix} | & * & \cdots & * \\ \frac{\mathbf{Ab}}{\|\mathbf{b}\|_2} & \vdots & & \vdots \\ | & * & \cdots & * \end{bmatrix}\right\rangle\right\rangle \\ &= \left|\begin{bmatrix} \frac{\mathbf{Ab}}{\|\mathbf{b}\|_2} \\ *\vdots \\ \vdots \\ * \end{bmatrix}\right\rangle = \left|\begin{bmatrix} \frac{\mathbf{Ab}}{\|\mathbf{A}\|_F\|\mathbf{b}\|_2} \\ *\vdots \\ \vdots \\ * \end{bmatrix}\right\rangle,\end{aligned}$$

where $*$ denotes arbitrary amplitudes. The third equality follows from the fact that the first column of $\mathbf{M}(\mathcal{U}_{\mathbf{b}})$ is equal to $\mathbf{b}/\|\mathbf{b}\|_2$. The equality follows since the ket notation $|\cdot\rangle$ we use is oblivious to a global scaling by a positive real number (i.e., $|\alpha\mathbf{x}\rangle = |\mathbf{x}\rangle$ for any real $\alpha > 0$), and so we can rescale the first element and the additional arbitrary amplitudes below it. We now denote the values in the lower part of the vector by ξ. The normalization we introduced by dividing by $\|\mathbf{A}\|_F$ ensures that $\|\mathbf{y}\|_2 = 1$.

See Figure 7 for a graphical description of qmsla.matrix_vec. The gate complexity is $g_{\mathbf{A}} + g_{\mathbf{b}}$ and the depth is bounded by $d_{\mathbf{A}} + d_{\mathbf{b}}$. The number of qubits is $q_{\mathbf{A}}$. The cost of the algorithm is $O(g_{\mathbf{A}} + g_{\mathbf{b}}) = O(g_{\mathbf{y}})$.

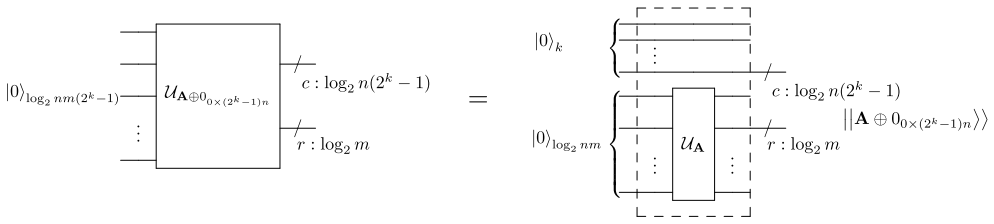

FIG. 6. *Graphical description of qmsla.pad_zero_columns (input: $\mathcal{U}_{\mathbf{A}}$ and k; output: $\mathcal{U}_{\mathbf{A}\oplus 0_{0\times(2^k-1)n}}$).*

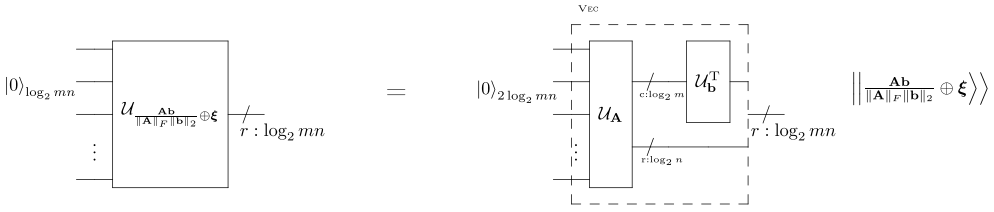

FIG. 7. *Graphical description of qmsla.matrix_vec (input: $\mathcal{U}_{\mathbf{A}}$ and $\mathcal{U}_{\mathbf{b}}$; output: $\mathcal{U}_{\frac{\mathbf{Ab}}{\|\mathbf{A}\|_F\|\mathbf{b}\|_2}\oplus\xi}$).*

3.2.8. Kronecker product: $\mathcal{U}_{\mathbf{A}\otimes\mathbf{B}} \leftarrow$ qmsla.kronecker($\mathcal{U}_{\mathbf{A}}$,$\mathcal{U}_{\mathbf{B}}$). Consider two state preparation circuits $\mathcal{U}_{\mathbf{A}}$ and $\mathcal{U}_{\mathbf{B}}$. We have that $(\mathcal{U}_{\mathbf{A}}\otimes\mathcal{U}_{\mathbf{B}})\cdot|0\rangle_{q_{\mathbf{A}}+q_{\mathbf{B}}} = ||\mathbf{A}\rangle\rangle||\mathbf{B}\rangle\rangle$. The state $||\mathbf{A}\rangle\rangle||\mathbf{B}\rangle\rangle = ||\mathbf{A}\rangle\rangle\otimes||\mathbf{B}\rangle\rangle$ is generally not equal to $||\mathbf{A}\otimes\mathbf{B}\rangle\rangle$, and $\mathcal{U}_{\mathbf{A}}\otimes\mathcal{U}_{\mathbf{B}}$ is not a state preparation circuit for $\mathbf{A}\otimes\mathbf{B}$. Nevertheless, the following proposition shows that by permuting the order of the registers of $||\mathbf{A}\rangle\rangle||\mathbf{B}\rangle\rangle$ we can obtain the state $||\mathbf{A}\otimes\mathbf{B}\rangle\rangle$. This allows us to construct a matrix state preparation circuit $\mathcal{U}_{\mathbf{A}\otimes\mathbf{B}}$ for $\mathbf{A}\otimes\mathbf{B}$.

PROPOSITION 14. *Let*

$$\overline{\sigma}_{\otimes} := (0,2,1,3).$$

Then

$$\mathcal{S}^R_{\bar{\sigma}_{\otimes}}\cdot||\mathbf{A}\rangle\rangle\,||\mathbf{B}\rangle\rangle = ||\mathbf{A}\otimes\mathbf{B}\rangle\rangle\,.$$

Proof. Suppose that $\mathbf{A}$ is $m\times n$ and $\mathbf{B}$ is $p\times q$. First, we show that

$$\mathcal{S}^R_{\bar{\sigma}_{\otimes}}\left|\left|\mathbf{E}^{m\times n}_{i,j}\right\rangle\right\rangle\left|\left|\mathbf{E}^{p\times q}_{k,l}\right\rangle\right\rangle = \left|\left|\mathbf{E}^{mp\times nq}_{pi+k,qj+l}\right\rangle\right\rangle$$

for all i,j,k, and l. Indeed,

$$\begin{aligned}
\mathcal{S}^R_{\bar{\sigma}_{\otimes}}\left|\left|\mathbf{E}^{m\times n}_{i,j}\right\rangle\right\rangle\left|\left|\mathbf{E}^{p\times q}_{k,l}\right\rangle\right\rangle &= \mathcal{S}^R_{\bar{\sigma}_{\otimes}}|j\rangle_{\log_2 n}|i\rangle_{\log_2 m}|l\rangle_{\log_2 q}|k\rangle_{\log_2 p}\\
&= |j\rangle_{\log_2 n}|l\rangle_{\log_2 q}|i\rangle_{\log_2 m}|k\rangle_{\log_2 p}\\
&= \left|\left|\mathbf{E}^{q\times n}_{l,j}\right\rangle\right\rangle\left|\left|\mathbf{E}^{p\times m}_{k,i}\right\rangle\right\rangle\\
&= \left|\mathbf{e}^{nq}_{qj+l}\right\rangle\left|\mathbf{e}^{mp}_{pi+k}\right\rangle\\
&= \left|\left|\mathbf{E}^{mp\times nq}_{pi+k,qj+l}\right\rangle\right\rangle.
\end{aligned}$$

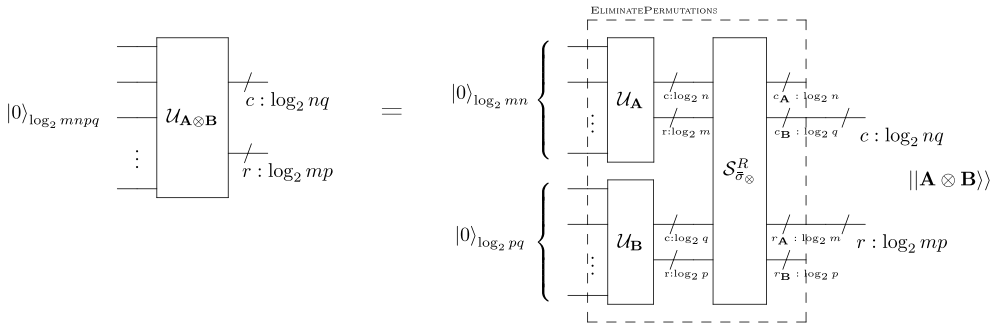

FIG. 8. *Graphical description of qmsla.kronecker (input: $\mathcal{U}_{\mathbf{A}}$ and $\mathcal{U}_{\mathbf{B}}$; output: $\mathcal{U}_{\mathbf{A}\otimes\mathbf{B}}$).*

Now,

$$\begin{aligned}
\mathcal{S}^R_{\bar{\sigma}_\otimes} ||\mathbf{A}\rangle\rangle ||\mathbf{B}\rangle\rangle &= \mathcal{S}^R_{\bar{\sigma}_\otimes} \left(\frac{1}{\|\mathbf{A}\|_F} \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} a_{ij} \left|\left|\mathbf{E}^{m\times n}_{i,j}\right\rangle\right\rangle \right) \left(\frac{1}{\|\mathbf{B}\|_F} \sum_{k=0}^{p-1}\sum_{l=0}^{q-1} b_{kl} \left|\left|\mathbf{E}^{p\times q}_{k,l}\right\rangle\right\rangle \right) \\
&= \mathcal{S}^R_{\bar{\sigma}_\otimes} \left(\frac{1}{\|\mathbf{A}\|_F\|\mathbf{B}\|_F} \sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\sum_{k=0}^{p-1}\sum_{l=0}^{q-1} a_{ij}b_{kl} \left|\left|\mathbf{E}^{m\times n}_{i,j}\right\rangle\right\rangle \left|\left|\mathbf{E}^{p\times q}_{k,l}\right\rangle\right\rangle \right) \\
&= \frac{1}{\|\mathbf{A}\|_F\|\mathbf{B}\|_F} \sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\sum_{k=0}^{p-1}\sum_{l=0}^{q-1} a_{ij}b_{kl} \mathcal{S}_{\bar{\sigma}_\otimes} \left|\left|\mathbf{E}^{m\times n}_{i,j}\right\rangle\right\rangle \left|\left|\mathbf{E}^{p\times q}_{k,l}\right\rangle\right\rangle \\
&= \frac{1}{\|\mathbf{A}\otimes\mathbf{B}\|_F} \sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\sum_{k=0}^{p-1}\sum_{l=0}^{q-1} a_{ij}b_{kl} \left|\left|\mathbf{E}^{mp\times nq}_{pi+k,qj+l}\right\rangle\right\rangle \\
&= ||\mathbf{A}\otimes\mathbf{B}\rangle\rangle . \qquad \square
\end{aligned}$$

Based on the last proposition, given matrix state preparation circuits $\mathcal{U}_{\mathbf{A}}$ and $\mathcal{U}_{\mathbf{B}}$ we can create a matrix state preparation circuit for $\mathcal{U}_{\mathbf{A}\otimes\mathbf{B}}$. See Figure 8 for a graphical description. Thus, we have $g_{\mathbf{A}\otimes\mathbf{B}} = g_{\mathbf{A}} + g_{\mathbf{B}}$ and $d_{\mathbf{A}\otimes\mathbf{B}} = \max(d_{\mathbf{A}}, d_{\mathbf{B}})$, once we apply EliminatePermutations to reduce the gate complexity and depth. The cost of qmsla.kronecker is $O(g_{\mathbf{A}\otimes\mathbf{B}})$.

3.2.9. Multiple Kronecker products: $\mathcal{U}_{\mathbf{A_1}\otimes\mathbf{A_2}\otimes\cdots\otimes\mathbf{A}_k} \leftarrow$ qmsla.kronecker $(\mathcal{U}_{\mathbf{A_1}}, \ldots, \mathcal{U}_{\mathbf{A}_k})$. Given a list of state preparation circuits $\mathcal{U}_{\mathbf{A}_1}, \ldots \mathcal{U}_{\mathbf{A}_k}$ we can compute $\mathcal{U}_{\mathbf{A}_1\otimes\mathbf{A}_2\otimes\cdots\otimes\mathbf{A}_k}$ using recursion, with the algorithm from the previous subsection used to combine results. That is, if $k = 1$ we simply return the circuit itself. For $k \geq 2$ we recursively form $\mathcal{U}_{\mathbf{A}_1\otimes\cdots\otimes\mathbf{A}_{\lfloor k/2\rfloor}}$ and $\mathcal{U}_{\mathbf{A}_{\lfloor k/2\rfloor+1}\otimes\cdots\otimes\mathbf{A}_k}$, and then combine the results to form $\mathcal{U}_{\mathbf{A}_1\otimes\mathbf{A}_2\otimes\cdots\otimes\mathbf{A}_k}$ using the algorithm from the previous subsection. Assuming $\mathbf{A}_i \in \mathbb{C}^{m_i\times n_i}$ for $i = 1, \ldots, k$, we have $g_{\mathbf{A}_1\otimes\cdots\otimes\mathbf{A}_k} = g_{\mathbf{A}_1} + \cdots + g_{\mathbf{A}_k}$ and $d_{\mathbf{A}_1\otimes\cdots\otimes\mathbf{A}_k} = \max(d_{\mathbf{A}_1}, \ldots, d_{\mathbf{A}_k})$, once we apply EliminatePermutations to reduce the gate complexity and depth.

The recursive algorithm has total complexity of $O(\log k \sum_{i=2}^{2k} g_{\mathbf{A}_i})$. This arises from the fact that each matrix appears in $O(\log k)$ EliminatePermutations operations, with a cost of $O(g(\text{output}))$ each. Thus, it is better to use a nonrecursive algorithm: In particular, we could compute the permutation associated with the Kronecker products upfront, and apply the EliminatePermutations once, to reduce cost to $O(g_{\mathbf{A}_1\otimes\cdots\otimes\mathbf{A}_k})$.

3.3. Level 2 qMSLA operations. Having defined a basic set of qMSLA primitives, we can start composing them into more complex operations. Here, we describe a few such operations that prove useful for estimating multivariate traces.

3.3.1. Pad with zero rows and padding diagonally: $\mathcal{U}_{\mathbf{A}\oplus 0_{(2^r-1)m\times 0}} \leftarrow$ qmsla.pad_zero_rows$(\mathcal{U}_{\mathbf{A}}, r)$ $(r>0)$ and $\mathcal{U}_{\mathbf{A}\oplus 0_{(2^r-1)m\times(2^k-1)n}} \leftarrow$ qmsla.pad $(\mathbf{A}, r, k)$ $(k, r>0)$. Assuming we already have qmla.pad_zero_columns($\mathcal{U}_{\mathbf{A}}$,r), then

$$\begin{aligned}&\text{qmlsa.pad\_zero\_rows}(\mathcal{U}_{\mathbf{A}}, r)\\ &\quad := \text{qmsla.transpose}(\text{qmsla.pad\_zero\_columns}(\text{qmsla.transpose}(\mathcal{U}_{\mathbf{A}}), r)).\end{aligned}$$

However, it is easy to see that this amounts to adding an empty register between the column and the row register. Thus, $g_{\mathbf{A}\oplus 0_{(2^r-1)m\times 0}} = g_{\mathbf{A}}$ and $d_{\mathbf{A}\oplus 0_{(2^r-1)m\times 0}} = d_{\mathbf{A}}$.

Both padding operations can be combined to a unified single padding operation:

$$\text{qmlsa.pad}(\mathbf{A}, r, k) := \text{qmsla.pad\_zero\_rows}(\text{qmsla.pad\_zero\_columns}(\mathcal{U}_{\mathbf{A}}, k), r)).$$

This amounts to adding two empty registers: one before the column register and one between the two registers. Again, $g_{\mathbf{A}\oplus 0_{(2^r-1)m\times(2^k-1)n}} = g_{\mathbf{A}}$ and $d_{\mathbf{A}\oplus 0_{(2^r-1)m\times(2^k-1)n}} = d_{\mathbf{A}}$, and the cost is $O(g_{\mathbf{A}})$.

3.3.2. Matrix adjoint: $\mathcal{U}_{\mathbf{A}^*} \leftarrow$ qmsla.adjoint$(\mathcal{U}_{\mathbf{A}})$. Since $\mathbf{A}^* = (\overline{\mathbf{A}})^{\mathrm{T}}$ we can compose qmsla.conjugate and qmsla.transpose (the order does not matter) to compute $\mathcal{U}_{\mathbf{A}^*}$ given $\mathcal{U}_{\mathbf{A}}$. Thus, $g_{\mathbf{A}^*} = g_{\mathbf{A}}$ and $d_{\mathbf{A}^*} = d_{\mathbf{A}}$. The cost is $O(g_{\mathbf{A}})$.

3.3.3. Overlap: $\mathcal{U}_{\frac{\psi^*\phi}{\|\psi\|_2\|\phi\|_2}\oplus\xi} \leftarrow$ qmsla.overlap$(\mathcal{U}_{\psi}, \mathcal{U}_{\phi})$. Given two state preparation circuits $\mathcal{U}_{\psi}$ and $\mathcal{U}_{\phi}$ of vectors ψ and ϕ of the same size, we can build a state preparation circuit for a unit vector of the form

$$\mathbf{y} = \begin{bmatrix} \frac{\psi^*\phi}{\|\psi\|_2\|\phi\|_2} \\ \vdots \\ \vdots \end{bmatrix}.$$

Thus, the overlap (dot product) appears in the first probability amplitude. The idea is to first form $\mathcal{U}_{\psi^*}$ from $\mathcal{U}_{\psi}$ using qmsla.adjoint, and then apply qmsla.matrix_vector to obtain $\mathcal{U}_{\mathbf{y}}$ (where we normalize the vector by dividing by $\|\psi\|_2$). A summary appears in Algorithm 1. Thus, the number of gates in the resulting circuit is $g_{\psi} + g_{\phi}$ and the depth is $d_{\psi} + d_{\phi}$, and the cost of the algorithm is $O(g_{\psi} + g_{\phi})$. Note that the resulting circuit does not use any additional control operation beyond the ones in $\mathcal{U}_{\psi}$ and $\mathcal{U}_{\phi}$, and no additional qubits.

Algorithm 1 QMSLA.OVERLAP.

1: **Input:** Classical description of the circuits $\mathcal{U}_{\psi}$,$\mathcal{U}_{\phi}$
2:
3: $\mathcal{U}_{\psi^*} \leftarrow$ qmsla.adjoint$(\mathcal{U}_{\psi})$
4: $\mathcal{U}_{\mathbf{y}} \leftarrow$ qmsla.matrix_vec$(\mathcal{U}_{\psi^*}, \mathcal{U}_{\phi})$
5: **return** $\mathcal{U}_{\mathbf{y}}$

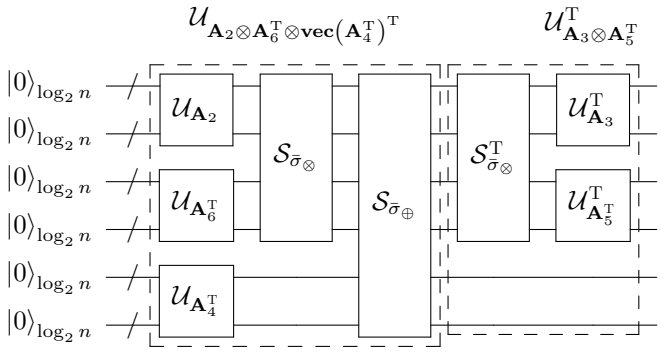

FIG. 9. *Visualization of the unitary transform* $\mathcal{U}_\phi(\mathcal{U}_{\mathbf{A}_2},\ldots,\mathcal{U}_{\mathbf{A}_6})$ *from the* MVTRACEPREP *algorithm, where* $\mathbf{A}_i \in \mathbb{C}^{n\times n}$.

4. Encoding multivariate traces in quantum states. In this section, we leverage qMSLA to introduce our main algorithm: MVTRACEPREP. Given state preparation circuits for $2k$ matrices, MVTRACEPREP produces two (vector) state preparation circuits, for which the overlap between the vectors is equal to the multivariate trace of the matrices up to normalization. More formally, assuming the inputs to MVTRACEPREP are $\mathcal{U}_{\mathbf{A}_1},\ldots,\mathcal{U}_{\mathbf{A}_{2k}}$, where $\mathcal{U}_{\mathbf{A}_i}$ is a matrix state preparation circuit for $\mathbf{A}_i$, MVTRACEPREP outputs two state preparations circuits $\mathcal{U}_\psi$ and $\mathcal{U}_\phi$ for vectors ψ and ϕ (respectively) such that $\psi^*\phi = \frac{\mathbf{Tr}(\mathbf{A}_1\mathbf{A}_2\cdots\mathbf{A}_{2k})}{\|\mathbf{A}_1\|_F\cdots\|\mathbf{A}_{2k}\|_F}$.

A pseudocode description of MVTRACEPREP appears in Algorithm 2. For illustration purposes, a high-level block circuit diagram of $\mathcal{U}_\phi(\mathcal{U}_{\mathbf{A}_2},\ldots,\mathcal{U}_{\mathbf{A}_6})$ (i.e., $k=3$) is shown in Figure 9 ($\mathcal{U}_\psi$, which depends only on $\mathcal{U}_{\mathbf{A}_1}$, is very simple and is based on qmsla.pad) with the SWAPs written explicitly (they can be eliminated using ELIMINATEPERMUTATIONS). In the following subsections, we explain MVTRACEPREP and its rationale, and we finally state and prove Theorem 18, which summarizes the main properties of MVTRACEPREP.

4.1. Warm-up I: $k=2$. As a warm-up, let us first consider the case of $k=2$, i.e., the trace of the product of four matrices. To keep notation simple, we denote the four matrices by $\mathbf{A},\mathbf{B},\mathbf{C}$, and $\mathbf{D}$, and we want to compute $\mathbf{Tr}(\mathbf{ABCD})$. The proposed algorithm is based on the following fact.

FACT 15 (Fact 7.4.9 from [6]). *Let* $\mathbf{A}\in\mathbb{C}^{n\times m}$, $\mathbf{B}\in\mathbb{C}^{m\times l}$, $\mathbf{C}\in\mathbb{C}^{l\times k}$, $\mathbf{D}\in\mathbb{C}^{k\times n}$. *Then,*

$$\mathbf{Tr}\left(\mathbf{ABCD}\right) = \mathbf{vec}\left(\mathbf{A}\right)^{\mathrm{T}}\left(\mathbf{B}\otimes\mathbf{D}^{\mathrm{T}}\right)\mathbf{vec}\left(\mathbf{C}^{\mathrm{T}}\right). \tag{4}$$

This fact can easily be used to implement, via qMSLA operations, an algorithm that takes $\mathcal{U}_\mathbf{A},\mathcal{U}_\mathbf{B},\mathcal{U}_\mathbf{C}$, and $\mathcal{U}_\mathbf{D}$ and outputs two state preparation circuits whose overlap is $\mathbf{Tr}\left(\mathbf{ABCD}\right)$ (in the parentheses we show which qMSLA operation is used in each step):

Step 1: Given $\mathcal{U}_\mathbf{C}$ and $\mathcal{U}_\mathbf{D}$, using qmsla.tranpose twice, compute circuits for $\mathbf{C}^{\mathrm{T}}$ and $\mathbf{D}^{\mathrm{T}}$, i.e., $\mathcal{U}_{\mathbf{C}^{\mathrm{T}}},\mathcal{U}_{\mathbf{D}^{\mathrm{T}}}$.

Step 2: Given $\mathcal{U}_\mathbf{B}$ and $\mathcal{U}_{\mathbf{D}^{\mathrm{T}}}$, using qmsla.kronecker, compute a circuit for $\mathbf{B}\otimes\mathbf{D}^{\mathrm{T}}$, i.e., $\mathcal{U}_{\mathbf{B}\otimes\mathbf{D}^{\mathrm{T}}}$.

Step 3: Given $\mathcal{U}_{\mathbf{C}^{\mathrm{T}}}$, using qmsla.vec, compute a circuit for $\mathbf{vec}(\mathbf{C}^{\mathrm{T}})$, i.e., $\mathcal{U}_{\mathbf{vec}(\mathbf{C}^{\mathrm{T}})}$.

Algorithm 2 MVTracePrep.

Input: Classical description of the circuits $\mathcal{U}_{\mathbf{A}_1}, \cdots \mathcal{U}_{\mathbf{A}_{2k}}$

{Constructing $\mathcal{U}_\psi$}
$\mathcal{U}_{\overline{\mathbf{A}}_1} \leftarrow$ qmsla.conjugate($\mathcal{U}_{\mathbf{A}_1}$)
$\mathcal{U}_\psi \leftarrow$ qmsla.pad_zero_rows(qmsla.vec($\mathcal{U}_{\overline{\mathbf{A}}_1}$), $\log_2 n_3 n_4 \cdots n_{2k}$)

{Constructing $\mathcal{U}_\phi$}
$p \leftarrow k+1$
for $i = p$ **to** $i = 2k$ **do**
  $\mathcal{U}_{\mathbf{A}_i^{\mathrm{T}}} \leftarrow$ qmsla.transpose($\mathcal{U}_{\mathbf{A}_i}$)
end for
if k is odd **then**
  $l_{\text{even}} \leftarrow (k+1)/2$ and $l_{\text{odd}} \leftarrow (k-1)/2$
  $\mathcal{U}_{\mathbf{F}^{(1)}_{\text{even}}} \leftarrow \mathcal{U}_{\mathbf{A}_p^{\mathrm{T}}}$ {State prep for $\mathbf{E}_{(k+1)/2,k} = \mathbf{A}_p^{\mathrm{T}}$}
  $\mathcal{U}_{\mathbf{F}^{(1)}_{\text{odd}}} \leftarrow$ qmsla.kronecker($\mathcal{U}_{\mathbf{A}_{p-1}}, \mathcal{U}_{\mathbf{A}^{\mathrm{T}}_{p+1}}$) {State prep for $\mathbf{O}_{i,k} = \mathbf{A}_{p+1} \otimes \mathbf{A}^{\mathrm{T}}_{p-1}$}
else
  $l_{\text{even}} \leftarrow k/2$ and $l_{\text{odd}} \leftarrow k/2$
  $\mathcal{U}_{\mathbf{F}^{(1)}_{\text{even}}} \leftarrow$ qmsla.kronecker($\mathcal{U}_{\mathbf{A}_{p-1}}, \mathcal{U}_{\mathbf{A}^{\mathrm{T}}_{p+1}}$) {State prep for $\mathbf{E}_{i,k} = \mathbf{A}_{p+1} \otimes \mathbf{A}^{\mathrm{T}}_{p-1}$}
  $\mathcal{U}_{\mathbf{F}^{(1)}_{\text{odd}}} \leftarrow \mathcal{U}_{\mathbf{A}_p^{\mathrm{T}}}$ {State prep for $\mathbf{O}_{k/2,k} = \mathbf{A}_p^{\mathrm{T}}$}
end if
for $i = 2$ **to** $i = 2k - p$ **do**
  if $p + i$ is odd **then**
    $\mathcal{U}_{\mathbf{O}_{\mathrm{i,k}}} \leftarrow$ qmsla.kronecker($\mathcal{U}_{\mathbf{A}_{p-i}}, \mathcal{U}_{\mathbf{A}^{\mathrm{T}}_{p+i}}$)
    $\mathcal{U}_{\mathbf{F}^{(i)}_{\text{odd}}} \leftarrow$ qmsla.kronecker($\mathcal{U}_{\mathbf{O}_{\mathrm{i,k}}}$, qmsla.rvec($\mathcal{U}_{\mathbf{F}^{(i-1)}_{\text{odd}}}$))
  else
    $\mathcal{U}_{\mathbf{E}_{\mathrm{i,k}}} \leftarrow$ qmsla.kronecker($\mathcal{U}_{\mathbf{A}_{p-i}}, \mathcal{U}_{\mathbf{A}^{\mathrm{T}}_{p+i}}$)
    $\mathcal{U}_{\mathbf{F}^{(i)}_{\text{even}}} \leftarrow$ qmsla.kronecker($\mathcal{U}_{\mathbf{E}_{\mathrm{i,k}}}$, qmsla.rvec($\mathcal{U}_{\mathbf{F}^{(i-1)}_{\text{even}}}$))
  end if
end for
$\mathcal{U}_\phi \leftarrow$ qmsla.matrix_vec($\mathcal{U}_{\mathbf{F}^{(l_{\text{odd}})}_{\text{even}}}$, qmsla.vec($\mathcal{U}_{\mathbf{F}^{(l_{\text{odd}})}_{\text{odd}}}$))

return $\mathcal{U}_\psi, \mathcal{U}_\phi$

Step 4: Given $\mathcal{U}_{\mathbf{B}\otimes\mathbf{D}^{\mathrm{T}}}$ and $\mathcal{U}_{\mathbf{vec}(\mathbf{C}^{\mathrm{T}})}$, using qmsla.matrix_vec, compute a circuit for $\frac{(\mathbf{B}\otimes\mathbf{D}^{\mathrm{T}})\mathbf{vec}(\mathbf{C}^{\mathrm{T}})}{\|\mathbf{B}\|_F\|\mathbf{D}\|_F\|\mathbf{C}\|_F} \oplus \xi'$ for some garbage $\xi' \in \mathbb{C}^{kl(mn-1)}$. Note that we use the fact that $\|\mathbf{vec}(\mathbf{C}^{\mathrm{T}})\|_2 = \|\mathbf{C}\|_F$ and $\|\mathbf{B}\otimes\mathbf{D}^{\mathrm{T}}\|_F = \|\mathbf{B}\|_F\|\mathbf{D}\|_F$. This is output $\mathcal{U}_\phi$.

Step 5: Given $\mathcal{U}_{\mathbf{A}}$, using qmsla.conjugate, compute a circuit for $\overline{\mathbf{A}}$, i.e., $\mathcal{U}_{\overline{\mathbf{A}}}$.

Step 6: Given $\mathcal{U}_{\overline{\mathbf{A}}}$, using qmsla.vec, compute a circuit for $\mathbf{vec}(\overline{\mathbf{A}})$, i.e., $\mathcal{U}_{\mathbf{vec}(\overline{\mathbf{A}})}$.

Step 7: Given $\mathcal{U}_{\mathbf{vec}(\overline{\mathbf{A}})}$, using qmsla.pad_zero_rows, compute a circuit for $\mathbf{vec}(\overline{\mathbf{A}}) \oplus 0_{kl(mn-1)}$. This is output $\mathcal{U}_\psi$.

Once we have $\mathcal{U}_\phi$ and $\mathcal{U}_\psi$, we can estimate the overlap in order to estimate $\mathbf{Tr}(\mathbf{ABCD})$. For example, we can use Algorithm 1.

PROPOSITION 16. *Consider circuits $\mathcal{U}_\psi$ and $\mathcal{U}_\phi$ which are the output of the steps described above. Consider the circuit which is the result of Algorithm 1 applied to $\mathcal{U}_\psi$ and $\mathcal{U}_\phi$. It is a state preparation circuit for the vector*

$$\frac{\mathbf{Tr}\left(\mathbf{ABCD}\right)}{\|\mathbf{A}\|_F\|\mathbf{B}\|_F\|\mathbf{C}\|_F\|\mathbf{D}\|_F} \oplus \xi$$

for some vector $\xi \in \mathbb{C}^{mnkl-1}$.

Proof. Clearly $\mathcal{U}_\phi$ is a state preparation circuit for $\phi = \frac{(\mathbf{B}\otimes\mathbf{D}^{\mathrm{T}})\mathbf{vec}(\mathbf{C}^{\mathrm{T}})}{\|\mathbf{B}\|_F\|\mathbf{D}\|_F\|\mathbf{C}\|_F} \oplus \xi'$ for some garbage $\xi' \in \mathbb{C}^{kl(mn-1)}$, and $\mathcal{U}_\psi$ is a state preparation circuit for $\psi = \mathbf{vec}(\overline{\mathbf{A}}) \oplus 0_{kl(mn-1)}$. We have that

$$\begin{aligned}
\psi^*\phi &= \left(\mathbf{vec}\left(\overline{\mathbf{A}}\right) \oplus 0_{lk(mn-1)}\right)^* \left(\frac{(\mathbf{B}\otimes\mathbf{D}^{\mathrm{T}})\mathbf{vec}\left(\mathbf{C}^{\mathrm{T}}\right)}{\|\mathbf{B}\|_F\|\mathbf{D}\|_F\|\mathbf{C}\|_F} \oplus \xi'\right) \\
&= \left(\mathbf{vec}\left(\mathbf{A}\right) \oplus 0_{lk(mn-1)}\right)^{\mathrm{T}} \left(\frac{(\mathbf{B}\otimes\mathbf{D}^{\mathrm{T}})\mathbf{vec}\left(\mathbf{C}^{\mathrm{T}}\right)}{\|\mathbf{B}\|_F\|\mathbf{D}\|_F\|\mathbf{C}\|_F} \oplus \xi'\right) \\
&= \mathbf{vec}\left(\mathbf{A}\right)^{\mathrm{T}} \frac{(\mathbf{B}\otimes\mathbf{D}^{\mathrm{T}})\mathbf{vec}\left(\mathbf{C}^{\mathrm{T}}\right)}{\|\mathbf{B}\|_F\|\mathbf{D}\|_F\|\mathbf{C}\|_F} \\
&= \frac{\mathbf{Tr}\left(\mathbf{ABCD}\right)}{\|\mathbf{B}\|_F\|\mathbf{D}\|_F\|\mathbf{C}\|_F}.
\end{aligned}$$

Algorithm 1 on $\mathcal{U}_\psi$ and $\mathcal{U}_\phi$ computes a state preparation circuit for the vector

$$\frac{\psi^*\phi}{\|\psi\|_2\|\phi\|_2} \oplus \xi = \frac{\mathbf{Tr}\left(\mathbf{ABCD}\right)}{\|\mathbf{A}\|_F\|\mathbf{B}\|_F\|\mathbf{C}\|_F\|\mathbf{D}\|_F} \oplus \xi$$

for some garbage $\xi \in \mathbb{C}^{mnkl-1}$. This is because $\|\psi\|_2 = \|\mathbf{A}\|_F$ and $\|\phi\|_2 = 1$ (since it is the result of qmsla.matrix_vec). ∎

4.2. Warm-up II: $k = 3$. For six matrices ($k = 3$) there is no simple multivariate trace formula. However, we can twice clamp together two of the matrices to reduce the number of matrices to four, and apply (4). Further algebra reduces the trace to a formula that can be encoded using qMSLA operations. More concretely,

$$\begin{aligned}
\mathbf{Tr}\left(\mathbf{A}_1\mathbf{A}_2\mathbf{A}_3\mathbf{A}_4\mathbf{A}_5\mathbf{A}_6\right) &= \mathbf{Tr}\left(\mathbf{A}_1(\mathbf{A}_2\mathbf{A}_3)\mathbf{A}_4(\mathbf{A}_5\mathbf{A}_6)\right) \\
&= \mathbf{vec}\left(\mathbf{A}_1\right)^{\mathrm{T}}\left(\mathbf{A}_2\mathbf{A}_3\otimes\mathbf{A}_6^{\mathrm{T}}\mathbf{A}_5^{\mathrm{T}}\right)\mathbf{vec}\left(\mathbf{A}_4^{\mathrm{T}}\right) \\
&= \mathbf{vec}\left(\mathbf{A}_1\right)^{\mathrm{T}}\left(\mathbf{A}_2\otimes\mathbf{A}_6^{\mathrm{T}}\right)\left(\mathbf{A}_3\otimes\mathbf{A}_5^{\mathrm{T}}\right)\mathbf{vec}\left(\mathbf{A}_4^{\mathrm{T}}\right) \\
&= \mathbf{Tr}\left(\mathbf{vec}\left(\mathbf{A}_1\right)^{\mathrm{T}}\left(\mathbf{A}_2\otimes\mathbf{A}_6^{\mathrm{T}}\right)\left(\mathbf{A}_3\otimes\mathbf{A}_5^{\mathrm{T}}\right)\mathbf{vec}\left(\mathbf{A}_4^{\mathrm{T}}\right)\right) \\
&= \mathbf{vec}\left(\mathbf{vec}\left(\mathbf{A}_1\right)^{\mathrm{T}}\right)^{\mathrm{T}}\left(\mathbf{A}_2\otimes\mathbf{A}_6^{\mathrm{T}}\otimes\mathbf{vec}\left(\mathbf{A}_4^{\mathrm{T}}\right)^{\mathrm{T}}\right)\mathbf{vec}\left(\mathbf{A}_3\otimes\mathbf{A}_5^{\mathrm{T}}\right) \\
&= \mathbf{vec}\left(\mathbf{A}_1\right)^{\mathrm{T}}\left(\mathbf{A}_2\otimes\mathbf{A}_6^{\mathrm{T}}\otimes\mathbf{vec}\left(\mathbf{A}_4^{\mathrm{T}}\right)^{\mathrm{T}}\right)\mathbf{vec}\left(\mathbf{A}_3\otimes\mathbf{A}_5^{\mathrm{T}}\right). \qquad (5)
\end{aligned}$$

In the above, we applied (4) to matrices $\mathbf{A}_1, \mathbf{A}_2\mathbf{A}_3, \mathbf{A}_4$ and $\mathbf{A}_5\mathbf{A}_6$. In the third equality, we used the Kronecker mixed product property. In the fourth equality we used the fact that $\mathbf{Tr}(c) = c$ for scalars. Finally, we apply (4) again, this time on the matrices $\mathbf{vec}(\mathbf{A}_1)^{\mathrm{T}}, \mathbf{A}_2\otimes\mathbf{A}_6^{\mathrm{T}}, \mathbf{A}_3\otimes\mathbf{A}_5^{\mathrm{T}}$ and $\mathbf{vec}(\mathbf{A}_4^{\mathrm{T}})$.

Equation (5) can be used to design, via qMSLA operations, circuits $\mathcal{U}_\phi$ and $\mathcal{U}_\psi$ for

$$\phi = \frac{\left(\mathbf{A}_2\otimes\mathbf{A}_6^{\mathrm{T}}\otimes\mathbf{vec}\left(\mathbf{A}_4^{\mathrm{T}}\right)^{\mathrm{T}}\right)\mathbf{vec}\left(\mathbf{A}_3\otimes\mathbf{A}_5^{\mathrm{T}}\right)}{\|\mathbf{A}_2\|_F\cdots\|\mathbf{A}_6\|_F} \oplus \xi'$$

and

$$\psi = \mathbf{vec}\left(\overline{\mathbf{A}_1}\right) \oplus 0_w$$

for some garbage ξ' and appropriate size w. Computing overlap using Algorithm 1 produces the state

$$\frac{\mathbf{Tr}\left(\mathbf{A}_1\mathbf{A}_2\cdots\mathbf{A}_6\right)}{\|\mathbf{A}_1\|_F\cdots\|\mathbf{A}_6\|_F} \oplus \xi$$

for some garbage ξ. We omit the details and move directly to arbitrary $k>2$, as the details are very similar to the case of $k=2$.

4.3. Multivariate trace formula for $k>2$. We now consider the trace of the product of $\mathbf{A}_1,\ldots,\mathbf{A}_{2k}$. We clamp $\mathbf{A}_2$ with $\mathbf{A}_3$ and $\mathbf{A}_{2k-1}$ with $\mathbf{A}_{2k}$, finding that we need to compute the trace of the product of $\mathbf{A}_1,\mathbf{A}_2\mathbf{A}_3,\mathbf{A}_4,\ldots\mathbf{A}_{2k-2},\mathbf{A}_{2k-1}\mathbf{A}_{2k}$. These are $2k-2$ matrices. So, we recursively apply the formula for $2k-2$ inputs, and after using again the Kronecker mixed product property and the fact that $\mathbf{Tr}\left(c\right)=c$ for scalars, we obtain a multivariate trace formula. The details are somewhat technical, so we first give the final result, and then prove it inductively.

First, we need a few additional notations. Given matrices $\mathbf{A}_1,\ldots\mathbf{A}_{2k}$, we define the following set of matrices:

$$\mathbf{E}_{i,k} := \mathbf{A}_{2i}\otimes\mathbf{A}_{2(k+1-i)}^{\mathrm{T}},\quad i=1,\ldots,\left\lfloor\frac{k}{2}\right\rfloor,$$

$$\mathbf{E}_{(k+1)/2,k} := \mathbf{A}_{k+1}^{\mathrm{T}}\quad (k \text{ is odd}),$$

$$\mathbf{O}_{i,k} := \mathbf{A}_{2i+1}\otimes\mathbf{A}_{2(k-i)+1}^{\mathrm{T}},\quad i=1,\ldots,\left\lfloor\frac{k-1}{2}\right\rfloor,$$

$$\mathbf{O}_{k/2,k} := \mathbf{A}_{k+1}^{\mathrm{T}}\quad (k \text{ is even}).$$

Each of the matrices $\{\mathbf{E}_{i,k}\}$ and $\{\mathbf{O}_{i,k}\}$ pairs two matrices (except for the pivotal matrix, which is not paired) of the sequence $\mathbf{A}_2,\ldots,\mathbf{A}_{2k}$ via the Kronecker product. Each matrix appears exactly once. The $\mathbf{E}$ matrices pair even indexed matrices, and the $\mathbf{O}$ matrices pair odd indexed matrices. Next, we define a series of functions, $F^{(1)},F^{(2)},\ldots$, where $F^{(p)}$ is p-ary and given by

$$F^{(1)}(\mathbf{X}) := \mathbf{X},$$
$$F^{(p)}(\mathbf{X}_1,\ldots,\mathbf{X}_p) := \mathbf{X}_1\otimes\mathbf{vec}\left(F^{(p-1)}(\mathbf{X}_2,\ldots,\mathbf{X}_p)\right).$$

THEOREM 17. *With the above notations, if k is even*

$$\begin{aligned}&\mathbf{MTr}_{2k}\left(\mathbf{A}_1,\ldots,\mathbf{A}_{2k}\right)\\&\quad=\mathbf{vec}\left(\mathbf{A}_1\right)^{\mathrm{T}}F^{(k/2)}(\mathbf{E}_{1,k},\ldots,\mathbf{E}_{k/2,k})\mathbf{vec}\left(F^{(k/2)}(\mathbf{O}_{1,k},\ldots,\mathbf{O}_{k/2,k})\right)\end{aligned}$$

and if k is odd

$$\begin{aligned}&\mathbf{MTr}_{2k}\left(\mathbf{A}_1,\ldots,\mathbf{A}_{2k}\right)\\&\quad=\mathbf{vec}\left(\mathbf{A}_1\right)^{\mathrm{T}}F^{((k+1)/2)}(\mathbf{E}_{1,k},\ldots,\mathbf{E}_{(k+1)/2,k})\mathbf{vec}\left(F^{((k-1)/2)}(\mathbf{O}_{1,k},\ldots,\mathbf{O}_{(k-1)/2,k})\right).\end{aligned}$$

Proof. We prove the theorem by induction on k. For the base case, note that $k=2$ is exactly (4):

$$\begin{aligned}\mathbf{Tr}\left(\mathbf{A}_1\cdots\mathbf{A}_4\right) &= \mathbf{vec}\left(\mathbf{A}_1\right)^{\mathrm{T}}\mathbf{A}_2\otimes\mathbf{A}_4^{\mathrm{T}}\mathbf{vec}\left(\mathbf{A}_3^{\mathrm{T}}\right)\\ &= \mathbf{vec}\left(\mathbf{A}_1\right)^{\mathrm{T}}F^{(1)}(\mathbf{E}_{1,k})\mathbf{vec}\left(F^{(1)}(\mathbf{O}_{1,k})\right).\end{aligned}$$

The inductive step is slightly different if k is even or odd. However, the difference is very minor and sums up with slightly modified indices. Thus, for conciseness, we show the inductive step only for even k to odd $k+1$. That is, assuming the formula holds for an even k, we prove that the following formula (which is the formula for $k+1$) holds:

$$\begin{aligned}&\mathbf{Tr}\left(\mathbf{A}_1\mathbf{A}_2\cdots\mathbf{A}_{2k}\mathbf{A}_{2k+1}\mathbf{A}_{2k+2}\right)\\ &\quad= \mathbf{vec}\left(\mathbf{A}_1\right)^{\mathrm{T}}F^{((k+2)/2)}(\mathbf{E}_{1,k+1},\ldots,\mathbf{E}_{(k+2)/2,k+1})\\ &\qquad\times\mathbf{vec}\left(F^{(k/2)}(\mathbf{O}_{1,k+1},\ldots,\mathbf{O}_{k/2,k+1})\right).\end{aligned}$$

To see that, we first note that

$$\begin{aligned}&\mathbf{Tr}\left(\mathbf{A}_1\cdots\mathbf{A}_{2k+2}\right)\\ &\quad= \mathbf{Tr}\left(\mathbf{A}_1(\mathbf{A}_2\mathbf{A}_3)\cdots\mathbf{A}_{2k}(\mathbf{A}_{2k+1}\mathbf{A}_{2k+2})\right)\\ &\quad= \mathbf{vec}\left(\mathbf{A}_1\right)^{\mathrm{T}}F^{\left(\frac{k}{2}\right)}(\mathbf{A}_2\mathbf{A}_3\otimes\mathbf{A}_{2k+2}^{\mathrm{T}}\mathbf{A}_{2k+1}^{\mathrm{T}},\mathbf{O}_{2,k+1},\ldots,\mathbf{O}_{\frac{k}{2},k+1})\\ &\qquad\times\mathbf{vec}\left(F^{\left(\frac{k}{2}\right)}(\mathbf{E}_{2,k+1},\ldots,\mathbf{E}_{\frac{k+1}{2},k+1})\right)\\ &\quad= \mathbf{vec}\left(\mathbf{A}_1\right)^{\mathrm{T}}\left(\mathbf{A}_2\mathbf{A}_3\otimes\mathbf{A}_{2k+2}^{\mathrm{T}}\mathbf{A}_{2k+1}^{\mathrm{T}}\otimes\mathbf{vec}\left(F^{\left(\frac{k-2}{2}\right)}(\mathbf{O}_{2,k+1},\ldots,\mathbf{O}_{\frac{k}{2},k+1})\right)^{\mathrm{T}}\right)\\ &\qquad\times\mathbf{vec}\left(F^{\left(\frac{k}{2}\right)}(\mathbf{E}_{2,k+1},\ldots,\mathbf{E}_{\frac{k+1}{2},k+1})\right)\\ &\quad= \mathbf{vec}\left(\mathbf{A}_1\right)^{\mathrm{T}}(\mathbf{A}_2\otimes\mathbf{A}_{2k+2}^{\mathrm{T}})(\mathbf{A}_3\otimes\mathbf{A}_{2k+1}^{\mathrm{T}})\otimes\mathbf{vec}\left(F^{\left(\frac{k-2}{2}\right)}(\mathbf{O}_{2,k+1},\ldots,\mathbf{O}_{\frac{k}{2},k+1})\right)^{\mathrm{T}}\\ &\qquad\times\mathbf{vec}\left(F^{\left(\frac{k}{2}\right)}(\mathbf{E}_{2,k+1},\ldots,\mathbf{E}_{\frac{k+1}{2},k+1})\right)\\ &\quad= \mathbf{vec}\left(\mathbf{A}_1\right)^{\mathrm{T}}(\mathbf{A}_2\otimes\mathbf{A}_{2k+2}^{\mathrm{T}})\left(\mathbf{A}_3\otimes\mathbf{A}_{2k+1}^{\mathrm{T}}\otimes\mathbf{vec}\left(F^{\left(\frac{k-2}{2}\right)}(\mathbf{O}_{2,k+1},\ldots,\mathbf{O}_{\frac{k}{2},k+1})\right)^{\mathrm{T}}\right)\\ &\qquad\times\mathbf{vec}\left(F^{\left(\frac{k}{2}\right)}(\mathbf{E}_{2,k+1},\ldots,\mathbf{E}_{\frac{k+1}{2},k+1})\right).\end{aligned}$$

In the second equality, we applied the inductive assumption. However, note that due to index shift the even labeled pairs ($\mathbf{E}$ matrices) become odd labeled pairs ($\mathbf{O}$ matrices), except for the first index. In the third equality we apply the recursive definition of $F^{(k/2)}$, and in the fourth equality we use the Kronecker mixed product property. The fifth equality follows by the fact that for any two matrices $\mathbf{A},\mathbf{B}$ and row vector $\mathbf{x}$, we have $(\mathbf{AB})\otimes\mathbf{x}=\mathbf{A}(\mathbf{B}\otimes\mathbf{x})$ [6, Fact 7.4.20]. Finally, we can always put a trace around scalars and utilize (4) once again, to find that

$$\begin{aligned}&\mathbf{Tr}\left(\mathbf{A}_1\cdots\mathbf{A}_{2k+2}\right)\\ &\quad= \mathbf{vec}\left(\mathbf{A}_1\right)^{\mathrm{T}}\mathbf{A}_2\otimes\mathbf{A}_{2k+2}^{\mathrm{T}}\otimes\mathbf{vec}\left(F^{\left(\frac{k}{2}\right)}(\mathbf{E}_{2,k+1},\ldots,\mathbf{E}_{\frac{k+1}{2},k+1})\right)^{\mathrm{T}}\\ &\qquad\times\mathbf{vec}\left(\mathbf{A}_3\otimes\mathbf{A}_{2k+1}^{\mathrm{T}}\otimes\mathbf{vec}\left(F^{\left(\frac{k-2}{2}\right)}(\mathbf{O}_{2,k+1},\ldots,\mathbf{O}_{\frac{k}{2},k+1})\right)^{\mathrm{T}}\right)\\ &\quad= \mathbf{vec}\left(\mathbf{A}_1\right)^{\mathrm{T}}F^{\left(\frac{k+2}{2}\right)}(\mathbf{E}_{1,k+1},\ldots,\mathbf{E}_{\frac{k+2}{2},k+1})\mathbf{vec}\left(F^{\left(\frac{k}{2}\right)}(\mathbf{O}_{1,k+1},\ldots,\mathbf{O}_{\frac{k}{2},k+1})\right),\end{aligned}$$

where in the second equality we applied the recursive definition of $F^{((k+2)/2)}$ and $F^{(k/2)}$ in reverse. ∎

4.4. MVTracePrep. Algorithm MVTracePrep implements the general trace formula described in the previous subsection using qMSLA operations.

THEOREM 18. *Given classical descriptions of matrix state preparation circuits* $\mathcal{U}_{\mathbf{A}_1}, \ldots \mathcal{U}_{\mathbf{A}_{2k}}$ *of matrices* $\mathbf{A}_1, \ldots, \mathbf{A}_{2k}$*, where the size of* $\mathbf{A}_i$ *is* $n_{i-1} \times n_i$ *for* $n_0, \ldots, n_{2k}$*, such that* $n_0 = n_{2k}$*, all of which are powers of* 2*, Algorithm* MVTracePrep *(Algorithm* 2*) outputs two (vector) state preparation circuits,* $\mathcal{U}_\psi$ *for a vector* ψ *and* $\mathcal{U}_\phi$ *for a vector* ϕ*, on* $q = 2\sum_{i=1}^{2k} \log_2 n_i$ *qubits, such that*

$$\langle \psi | \phi \rangle = \frac{\mathbf{Tr}\left(\mathbf{A}_1 \cdots \mathbf{A}_{2k}\right)}{\|\mathbf{A}_1\|_F \cdots \|\mathbf{A}_{2k}\|_F}.$$

$\mathcal{U}_\psi$ *depends only on* $\mathbf{A}_1$*, while* $\mathcal{U}_\phi$ *depends only on* $\mathbf{A}_2, \ldots, \mathbf{A}_{2k}$*. The classical cost of the algorithm is* $O(\sum_{i=1}^{2k} g_{\mathbf{A}_i})$*. The depth of* $\mathcal{U}_\psi$ *is* $d_{\mathbf{A}_1}$ *and the gate complexity is* $g_{\mathbf{A}_1}$*. The depth of* $\mathcal{U}_\phi$ *is* $\max_{i\in[2,4,\cdots,2k]} d_{\mathbf{A}_i} + \max_{i\in[3,5,\cdots,2k-1]} d_{\mathbf{A}_i}$ *and the gate complexity is* $\sum_{i=2}^{2k} g_{\mathbf{A}_i}$*.*

Proof. Correctness of the algorithm follows Theorem 17, since it simply implements the various parts of the formula using qMSLA operations. Indeed, $\mathcal{U}_{\mathbf{F}^{(i)}_{\text{even}}}$ and $\mathcal{U}_{\mathbf{F}^{(i)}_{\text{odd}}}$ are state preparation circuits for

$$\mathbf{F}^{(i)}_{\text{even}} := F^{(i)}(\mathbf{E}_{1,k}, \ldots, \mathbf{E}_{i,k}),$$

$$\mathbf{F}^{(i)}_{\text{odd}} := F^{(i)}(\mathbf{O}_{1,k}, \ldots, \mathbf{O}_{i,k}),$$

respectively, and $\mathcal{U}_{\mathbf{O}_{i,k}}$ and $\mathcal{U}_{\mathbf{E}_{i,k}}$ are state preparation for $\mathbf{O}_{i,k}$ and $\mathbf{E}_{i,k}$ (respectively). However, further explanations are needed for lines 4 and 29. Since the output of qmsla.matrix_vec is a state preparation for the product with garbage, line 29 creates a state preparation circuit for

$$\phi = \begin{cases} \dfrac{F^{(k/2)}(\mathbf{E}_{1,k}, \ldots, \mathbf{E}_{k/2,k})\mathbf{vec}\left(F^{(k/2)}(\mathbf{O}_{1,k}, \ldots, \mathbf{O}_{k/2,k})\right)}{\|\mathbf{A}_2\|_F \cdots \|\mathbf{A}_{2k}\|_F} \oplus \xi' & k \text{ is even,} \\ \dfrac{F^{((k+1)/2)}(\mathbf{E}_{1,k}, \ldots, \mathbf{E}_{(k+1)/2,k})\mathbf{vec}\left(F^{((k-1)/2)}(\mathbf{O}_{1,k}, \ldots, \mathbf{O}_{(k-1)/2,k})\right)}{\|\mathbf{A}_2\|_F \cdots \|\mathbf{A}_{2k}\|_F} \oplus \xi' & k \text{ is odd.} \end{cases}$$

To avoid the garbage contaminating the overlap, the algorithm set ψ to be equal to $\mathbf{vec}(\overline{\mathbf{A}}_1)$ padded with zeros. Thus, line 4 creates a state preparation circuit for

$$\psi = \mathbf{vec}\left(\overline{\mathbf{A}}_1\right) \oplus 0^{0\times \log_2 n_3 n_4 \cdots n_{2k}}.$$

According to Theorem 17, we have that $\psi^*\phi = \frac{\mathbf{Tr}(\mathbf{A}_1 \cdots \mathbf{A}_{2k})}{\|\mathbf{A}_1\|_F \cdots \|\mathbf{A}_{2k}\|_F}$.

We are left with analyzing the gate and depth complexities, along with the running time, considering qubit sizes. For $\mathcal{U}_\psi$, following the conjugation and padding of $\mathcal{U}_{\mathbf{A}_1}$, the gate and depth complexities are $g_{\mathbf{A}_1}$ and $d_{\mathbf{A}_1}$, respectively, resulting in a running time of $O(g_{\mathbf{A}_1})$. Subsequently, Kronecker products and the matrix vector product introduce no additional complexity in terms of depth and gate complexity. Consequently, the depth $d_{\mathcal{U}_\phi}$ is $\max_{i\in[2,4,\cdots,2k]} d_{\mathbf{A}_i} + \max_{i\in[3,5,\cdots,2k-1]} d_{\mathbf{A}_i}$, and the gate complexity is $\sum_{i=2}^{2k} g_{\mathbf{A}_i}$. The qubit count of both circuits is determined by the sizes of matrices with even indices, given that $\mathcal{U}_{\mathbf{A}_i}$ has dimensions $n_{i-1} \times n_i$ for $n_0, \ldots, n_{2k}$, where $n_0 = n_{2k}$. Thus, the qubit count is $q = 2\sum_{i=1}^{2k} \log_2 n_i$.

Now consider the running time. In the description in Algorithm 2 we use qMSLA operations, each having complexity of $O(g(\text{output}))$. Unfortunately, this does not imply that the final output of a sequence of qMSLA operations has complexity of $O(g(\text{output}))$. Specifically, the description of MVTracePrep results in complexity which is $O(\log k \sum_{i=2}^{2k} g_{\mathbf{A}_i})$ (this is because each matrix appears in $O(\log k)$ qMSLA operations, and for each one you pay for its gate complexity). However, a careful implementation of the algorithm that writes parts of $\mathcal{U}_{\mathbf{A}_1}, \ldots, \mathcal{U}_{\mathbf{A}_{2k}}$ to their final place, and delays EliminatePermutations to the end, allows us to shave off the $\log k$ factor. We defer the details to Appendix C. ∎

5. Estimating multivariate traces. Algorithm MVTracePrep encodes $\mathbf{MTr}_{2k}(\mathbf{A}_1, \ldots, \mathbf{A}_{2k})$ using two state preparation circuits $\mathcal{U}_\phi$ and $\mathcal{U}_\psi$, where to overlap between $|\phi\rangle$ and $|\psi\rangle$ is equal to $\mathbf{MTr}_{2k}(\mathbf{A}_1, \ldots, \mathbf{A}_{2k})$ up to normalization. These two circuits can be used in the context of a larger quantum algorithm, or used to directly estimate $\mathbf{MTr}_{2k}(\mathbf{A}_1, \ldots, \mathbf{A}_{2k})$ using various algorithms for estimating overlap (e.g., Hadamard test, swap test, and Algorithm 1; each can be combined with amplitude estimation acceleration [44, Example 4.5]). Here we analyze the overall cost needed for estimating $\mathbf{MTr}_{2k}(\mathbf{A}_1, \ldots, \mathbf{A}_{2k})$ up to additive ϵ error and discuss implications in the context of an end-to-end algorithm that starts with matrices $\mathbf{A}_1, \ldots, \mathbf{A}_{2k}$ explicitly stored in classical memory. We focus on using the Hadamard test, and for simplicity we assume the matrices are real. For complex matrices we need to employ the Hadamard test twice (the real and imaginary versions), but the analysis is essentially the same.

Following (2), we have

$$\mathrm{HT}_{\mathcal{U}_\psi^* \mathcal{U}_\phi} |0\rangle_1 |0\rangle_q = \frac{1}{2}(|0\rangle_1 (\mathcal{I} + \mathcal{U}_\psi^* \mathcal{U}_\phi) |0\rangle_q + |1\rangle_1 (\mathcal{I} - \mathcal{U}_\psi^* \mathcal{U}_\phi) |0\rangle_q)$$

for $q = 2\sum_{i=1}^{2k} \log_2 n_i$. Due to (3) and the fact that the matrices are real we get that

$$\mathbf{p}(0) = \frac{1}{2}\left(1 + \mathbf{Tr}\left(\frac{\mathbf{A}_1 \cdots \mathbf{A}_{2k}}{\|\mathbf{A}_1\|_F \cdots \|\mathbf{A}_{2k}\|_F}\right)\right),$$

where $\mathbf{p}(0)$ denotes the probability of measuring 0 in the first qubit (when only that qubit is measured). Next, we compute an approximation $\tilde{\mathbf{p}}$ of $\mathbf{p} := \mathbf{p}(0)$, and then we can build an approximation $\tilde{t} \approx t := \mathbf{Tr}(\frac{\mathbf{A}_1 \cdots \mathbf{A}_{2k}}{\|\mathbf{A}_1\|_F \cdots \|\mathbf{A}_{2k}\|_F})$ via

$$\tilde{\mathbf{t}} := 2\tilde{\mathbf{p}} - 1.$$

Using quantum phase estimation we can estimate $\tilde{\mathbf{p}}$ s.t. $|\mathbf{p} - \tilde{\mathbf{p}}| \le \epsilon'$ with total complexity (depth times shots) of $O(\epsilon'^{-1} \max_i(d_{\mathbf{A}_i}))$ (see [44, section 4.2]) and we have that

$$\begin{aligned} \left|\mathbf{t} - \tilde{\mathbf{t}}\right| &= |(2\mathbf{p} - 1) - (2\tilde{\mathbf{p}} - 1)| \\ &= 2\,|\mathbf{p} - \tilde{\mathbf{p}}| \\ &\le 2\epsilon'. \end{aligned}$$

Thus, the total complexity is $O(\epsilon^{-1} \max_i(d_{\mathbf{A}_i}))$ for approximating $\mathbf{Tr}(\frac{\mathbf{A}_1 \cdots \mathbf{A}_{2k}}{\|\mathbf{A}_1\|_F \cdots \|\mathbf{A}_{2k}\|_F})$ to additive error ϵ. Since $\frac{\mathbf{Tr}(\mathbf{A}_1 \cdots \mathbf{A}_{2k})}{\|\mathbf{A}_1\|_F \cdots \|\mathbf{A}_{2k}\|_F} = \mathbf{Tr}(\frac{\mathbf{A}_1 \cdots \mathbf{A}_{2k}}{\|\mathbf{A}_1\|_F \cdots \|\mathbf{A}_{2k}\|_F})$, the total complexity is $O(\epsilon^{-1} \|\mathbf{A}_1\|_F \cdots \|\mathbf{A}_{2k}\|_F \max_i(d_{\mathbf{A}_i}))$ for approximating $\mathbf{Tr}(\mathbf{A}_1 \cdots \mathbf{A}_{2k})$ to ϵ additive error.

5.1. Quantum multivariate trace estimation with classical inputs. Consider the case that the matrices $\mathbf{A}_1, \ldots, \mathbf{A}_{2k} \in \mathbb{R}^{n\times n}$ are given in classical memory, and our goal is to approximate $\mathbf{MTr}_{2k}(\mathbf{A}_1, \ldots, \mathbf{A}_{2k})$ to additive error ϵ. For simplicity of analysis, we assume that all matrices are $n \times n$. We do not assume sparsity of $\mathbf{A}_i$ (i.e., for each i we treat $\mathbf{A}_i$ as dense) or *any* other special property, like positive definiteness. To employ MVTracePrep, we need state preparation circuits $\mathcal{U}_{\mathbf{A}_1}, \ldots, \mathcal{U}_{\mathbf{A}_{2k}}$. Given a classically stored matrix $\mathbf{A} \in \mathbb{C}^{m\times n}$, a state preparation circuit for $\mathbf{A}$ can be built using the algorithm described in [62] in $O(mn)$. So the total combined cost (classical, one time) of building $\mathcal{U}_{\mathbf{A}_1}, \ldots, \mathcal{U}_{\mathbf{A}_{2k}}$ is $O(kn^2)$, and the depth of each circuit is $O(n^2)$. The costs of the output of the MVTracePrep are

$$q = 4k \log_2 n + 1, \qquad g = O(kn^2), \qquad d = O(n^2).$$

The classical cost of MVTracePrep is also $O(kn^2)$.

Using the Hadamard test and quantum phase estimation, as in the previous subsection, we get total quantum complexity for ϵ additive error of $O(\epsilon^{-1}\|\mathbf{A}_1\|_F \cdots \|\mathbf{A}_{2k}\|_F n^2)$. Thus, the total cost, classical and quantum, of approximating $\mathbf{MTr}_{2k}(\mathbf{A}_1, \ldots, \mathbf{A}_{2k})$ to additive ϵ error with constant probability is

$$O\left(n^2(k + \epsilon^{-1}\|\mathbf{A}_1\|_F \cdots \|\mathbf{A}_{2k}\|_F)\right).$$

By taking the median of $O(\log(1/\delta))$ different executions of the algorithm, we can boost the success probability to at least $1 - \delta$, so the running time is $O(n^2(k + \epsilon^{-1}\|\mathbf{A}_1\|_F \cdots \|\mathbf{A}_{2k}\|_F \log(1/\delta)))$ for an additive (ϵ, δ) estimator.

Comparison to classical methods. We now compare our algorithm to multivariate trace estimators that use only classical computation. The simplest classical approach is to compute $\mathbf{A}_1\mathbf{A}_2 \cdots \mathbf{A}_{2k}$ and then sum the diagonal entries. This algorithm requires $O(n^3k)$ arithmetic operations and provides an exact value for the matrix product trace. Our algorithm reduces the dependence on n from n^3 to n^2, but provides a stochastic estimate.

More appropriate baselines are classical stochastic trace estimators. The state-of-the-art algorithm is Hutch++ [49], which requires $O(\epsilon^{-1}\log(1/\delta))$ matrix vector products for a relative (ϵ, δ) estimator for a symmetric positive definite $\mathbf{A}$. Computing the product of $\mathbf{A} = \mathbf{A}_1 \cdots \mathbf{A}_{2k}$ with a vector can be accomplished in $O(n^2k)$ using repeated matrix-vector products. Overall, the cost of Hutch++ for the case we consider in this section is $O(n^2k\epsilon^{-1}\log(1/\delta))$. In comparison, in our algorithm the factor n^2k does not appear with ϵ and δ. So, as long as $\|\mathbf{A}_1\|_F \cdots \|\mathbf{A}_{2k}\|_F = o(k)$ we improve the running time compared to Hutch++, though Hutch++ guarantees are relative and not additive.

However, one can argue that using parallel computing with k compute nodes can trivially reduce the cost of Hutch++ to $O(n^2\epsilon^{-1}\log(1/\delta))$, and this is a fairer comparison since our algorithm uses $O(k)$ qubits. In fact, when input data is given in an unstructured dense classical manner, it is hard to achieve any quantum advantage since encoding the data will immediately incur an exponential cost in the number of qubits. The classical algorithm for encoding a vector of size 2^N as the amplitudes of an N-qubit circuit (the algorithm from [62], which is implemented in various quantum computing frameworks) requires $O(2^N)$ gate complexity and depth. Yet, there is a trade-off between the number of qubits and depth: recently it was shown [3] that a quantum circuit with a depth of $O(N^2)$ and $O(2^N)$ qubits can effectively load a 2^N-dimensional vector into a quantum state (here the quantum system has ancilla qubits).

Regardless, an exponential cost is involved, and any exponential (or even polynomial) reduction in cost (compared to classical) in any downstream algorithm (as is the case for our algorithm) gets nullified.

The main potential for our algorithm is when the input matrices are huge matrices with compact quantum circuit description. In such scenarios we have an encoding of the matrices in an exponentially sized Hilbert space, but only pay costs that are (hopefully) linear or polynomial in the number of qubits. Such situations can only occur for scenarios where we can construct meaningful matrices directly on the quantum state. The most likely candidates are Hamiltonians of quantum systems. Another avenue worth exploring involves encoding a clever approximation of matrix $\mathbf{A}$, rather than encoding $\mathbf{A}$ directly. One can hope that such approximations could be generated using cost-effective operations in the quantum domain, particularly those that have low depth. Presumably, one can attempt to find such approximations that are not necessarily cheaper in terms of matrix size, but more easily encoded into the quantum state (depthwise), while preserving the spectral properties of $\mathbf{A}$ up to a small error. We leave exploring these directions to future research.

We stress that the above discussion holds only for positive definite matrices, while our algorithm is more general and can even be used to estimate multivariate traces of nonsymmetric input matrices.

5.2. From multivariate traces to spectral sums. Multivariate trace estimation can be used for spectral sum estimation, via polynomial approximation. A well- established approach is to estimate $\mathbf{Tr}(f(\mathbf{A}))$ using $\mathbf{Tr}(p(\mathbf{A}))$ where $p(\cdot)$ is a polynomial that approximates $f(\cdot)$. We do not consider how $p(\cdot)$ is computed; we focus solely on how $\mathbf{Tr}(p(\mathbf{A}))$ can be computed via multivariate traces.

We suggest two methods for this task. The first is to use matrix moments. Suppose $p(x) = \sum_{k=0}^{L} c_k x^k$ is an L-degree polynomial. Then,

$$\mathbf{Tr}\left(p(\mathbf{A})\right) = \sum_{k=0}^{L} c_k \mathbf{Tr}\left(\mathbf{A}^k\right).$$

Therefore, we can approximate $\mathbf{Tr}(\mathbf{A}^k)$ for $k = 0, \ldots, L$ using our algorithm.[3] As long as we have access to enough qubits, we can compute them in parallel. Using these trace values, the sum is computed classically.

The second approach translates $\mathbf{Tr}(p(\mathbf{A}))$ to a single multivariate trace. We first rewrite $p(x)$ in a factored form:

$$p(x) = c_L(x - r_1)(x - r_2)\cdots(x - r_L).$$

The shifts $r_1, \ldots, r_L$ are the roots of $p(\cdot)$, and as long as the polynomial has positive degree they exist (perhaps with multiplicity in some of the roots). Now,

$$p(\mathbf{A}) = c_L(\mathbf{A} - r_1\mathbf{I})(\mathbf{A} - r_2\mathbf{I})\cdots(\mathbf{A} - r_L\mathbf{I})$$

so $\mathbf{Tr}(p(\mathbf{A})) = c_L\mathbf{MTr}_L(\mathbf{A} - r_1\mathbf{I}, \ldots, \mathbf{A} - r_L\mathbf{I})$.

It is well known that the roots of a polynomial expressed as a linear combination of monomials, i.e., $p(x) = \sum_{k=0}^{L} c_k x^k$, are equal to the eigenvalues of the $L \times L$ companion matrix:

[3] For odd degrees we add a state preparation circuit for the identity matrix as input.

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ -c_1/c_0 & -c_2/c_0 & \cdots & -c_{L-1}/c_0 & -c_L/c_0 \end{bmatrix}.$$

Indeed, the MATLAB roots function implements exactly this method. However, this method has numerical stability issues, and there are better methods based on expressing the polynomial in the Chebyshev basis. See [66, Chapter 18] for a discussion.

The polynomial factorization based approach requires L distinct but related input matrix state preparation circuits: each one of them is a preparation circuit of an $\mathbf{A}$ shifted by some constant. This is in contrast to the moments based method, which requires a single preparation circuit (for $\mathbf{A}$ only). Thus, this approach is especially appealing in cases where we can generate circuits of shifted variants of $\mathbf{A}$ once we have some base preparation circuit $\mathbf{A}$ itself.

6. Conclusions. In this work, we have presented a new quantum algorithm for approximating multivariate traces. To that end we introduce the qMSLA framework. qMSLA builds a set of fundamental matrix algebra building block operations for circuits that encode matrices in quantum states. qMSLA facilitates the seamless composition of basic operations to construct circuits, as we demonstrate with our construction of circuits that encode multivariate traces.

We view the qMSLA-based approach as a conceptually promising paradigm for designing a quantum algorithm for QLA and QML via high-level building blocks. However, the current set of operations supported is limited, and this limits the set of matrix computations that can be implemented. In particular, qMSLA is not expressive enough to solve linear equations, solve least squares, compute eigenvalues, etc. For broader applicability, qMSLA must be expanded. We envision several avenues for future expansion of qMSLA via additional research:

- **Enrich the set of qMSLA operations:** Incorporating additional important level-1 operations will allow us to express a wider array of matrix algorithms. The two most important operations are matrix addition and matrix-matrix multiplication. The former can be implemented using the linear combination of unitaries technique, while the latter is quite similar to the already present matrix-vector product. Also, further expanding the existing set of level 2 operations within qMSLA will further enhance its versatility and enable the efficient construction of low-depth circuits for complex matrix computations.
- **Extended matrix state preparation circuits:** Developing an extended version of the qMSLA framework that can handle "garbage" in ancilla qubits of inputs and outputs is crucial. Quantum computations often involve intermediate results that are not essential for the final outcome. A framework capable of efficiently dealing with such garbage data will significantly improve the overall efficiency, as it will allow a wider set of basic matrix operations.
- **Methods for building efficient state preparation circuits:** While the qMSLA approach holds promise, the inputs are state preparation circuits, so real-world application requires efficient state preparation circuits so that we can even consider using qMSLA for them. Developing a suite of algorithms for constructing state preparation circuits will dramatically enhance the applicability of qMSLA.

Appendices. The structure of the appendices is outlined as follows:

- In Appendix A, we provide a detailed description of level 0 qMSLA operations. These fundamental circuit-level operations serve as building blocks for higher levels of qMSLA operations.
- Appendix B gives implementation details for level 1 qMSLA. In particular, Appendix B defines a data structure for describing matrix state preparation circuits and provides pseudocodes for all level 1 operations.
- Appendix C describes a more efficient variant of MVTracePrep.

Appendix A. Level 0 qMSLA. In order to describe the implementation of qMSLA in a concise yet complete framework independent manner, we base qMSLA's implementation on a set of basic circuit level operations. We group these operations as "level 0 qMSLA." However, we stress that these operations are at the circuit level and do not operate on state preparation circuits. In the next appendix, we explain how to implement all level 1 qMSLA operations in terms of level 0 operations. We emphasize that while our description of level 0 operations is independent of various quantum computing frameworks (e.g., QISKIT, Q#, PennyLane), nearly all operations are present in such frameworks under various names. There are a couple of operations that are not always present (circuit transpose and conjugate). For these we provide implementation details.

As a design choice, all qMSLA operations we describe (here, and in subsequent sections) are assumed to be nondestructive; they accept circuits and return a new circuit, without overwriting inputs. Of course, in the context of an algorithm that is implemented using a sequence of qMSLA operations, it is possible to use destructive operations for more efficient implementation. We view this as a type of complier optimization, which we leave for future research.

Table 3 provides a summary of all level 0 qMSLA operations. In the following subsections, we provide additional details.

A.1. Creating new circuits: $\mathcal{U} \leftarrow$ qmsla.qc_empty(q), $\mathcal{U} \leftarrow$ qmsla.qc_tensor($\mathcal{Q}, \mathcal{W}$), $\mathcal{U} \leftarrow$ qmsla.qc_compose($\mathcal{W}, Q, \sigma$). The first set of level 0 operations concern creating a new circuit. Circuits can be created in three ways: initiating an empty circuit, taking the tensor product of two circuits, and composing two circuits.

Table 3

Level 0 qMSLA operations. These are basic circuit-level operations that allow us to implement higher-level qMSLA operations. The classical cost for these operations remains consistently $O(g(\text{output}))$. In the table, $\mathcal{U}, \mathcal{W}, \mathcal{Q}$ are quantum circuits, $\mathcal{G}$ is a quantum gate, σ is a permutation, and δ is a list of qubit indices.

Input	Output	q(output)	g(output)	d(output)	Operation name	Subsection
q	$\mathcal{I}_q$	q	0	0	qmsla.qc_empty	A.1
$\mathcal{W}, \mathcal{Q}$	$\mathcal{Q} \otimes \mathcal{W}$	$q(\mathcal{W}) + q(\mathcal{Q})$	$g(\mathcal{W}) + g(\mathcal{Q})$	$\max(d(\mathcal{W}), d(Q))$	qmsla.qc_tensor	A.1
$\mathcal{W}, \mathcal{Q}, \sigma$	$\mathcal{S}_{\sigma^{-1}} \mathcal{Q} \mathcal{S}_\sigma \mathcal{W}$	$q(\mathcal{W})$	$g(\mathcal{W}) + g(\mathcal{Q})$	$d(\mathcal{W}) + d(Q)$	qmsla.qc_compose	A.1
$\mathcal{Q}$, $\mathcal{G}$,δ	$\mathcal{G}_\delta \mathcal{Q}$	$q(\mathcal{Q})$	$g(\mathcal{Q}) + 1$	$d(\mathcal{Q}) + 1$	qmsla.qc_add_gate	A.2
$\mathcal{U}$	$\mathcal{U}^{\mathrm{T}}$	$q(\mathcal{U})$	$g(\mathcal{U})$	$d(\mathcal{U})$	qmsla.qc_transpose	A.3
$\mathcal{U}$	$\overline{\mathcal{U}}$	$q(\mathcal{U})$	$g(\mathcal{U})$	$d(\mathcal{U})$	qmsla.qc_conjugate	A.3
$\mathcal{U}$	$\mathcal{U}^{-1}$	$q(\mathcal{U})$	$g(\mathcal{U})$	$d(\mathcal{U})$	qmsla.qc_inverse	A.3
$\mathcal{W}, \sigma$	$\mathcal{S}_\sigma \mathcal{W}$[4]	$q(\mathcal{W})$	$g(\mathcal{W})$	$d(\mathcal{W})$	qmsla.qc_permute_bits	A.4

[4]Equality here holds only on the ground state. That is, if $\mathcal{U}$ is the output, we only require that $\mathcal{U}|0\rangle_{q(\mathcal{W})} = \mathcal{S}_\sigma \mathcal{W}|0\rangle_{q(\mathcal{W})}$.

$\mathcal{U} \leftarrow$ qmsla.qc_empty(q). Create a new empty (no gates) circuit $\mathcal{U}$ on q qubits. The unitary matrix associated with the circuit is the identity matrix, i.e., $\mathbf{M}(\mathcal{U}) = \mathbf{I}_{2^q}$.

$\mathcal{U} \leftarrow$ qmsla.qc_tensor($\mathcal{Q}, \mathcal{W}$). Create a new circuit $\mathcal{U}$ on $q(Q) + q(\mathcal{W})$ qubits. Apply $\mathcal{Q}$ to the $q(\mathcal{Q})$ MSB qubits and $\mathcal{W}$ to the $q(\mathcal{W})$ LSB qubits. The corresponding unitary is the Kronecker (aka tensor) product of the unitaries, i.e., $\mathbf{M}(\mathcal{U}) = \mathbf{M}(\mathcal{Q}) \otimes \mathbf{M}(\mathcal{W})$.

$\mathcal{U} \leftarrow$ qmsla.qc_compose($\mathcal{W}, Q, \sigma$). Create a new circuit $\mathcal{U}$ on $\max(q(\mathcal{Q}), q(\mathcal{W}))$ qubits which is made of first applying $\mathcal{W}$ (on the MSB $q(\mathcal{W})$ qubits), and then applying $\mathcal{Q}$. However, for $i = 1, \ldots, q(\mathcal{W})$ qubit $\sigma(i)$ in $\mathcal{W}$'s output is wired to qubit i in $\mathcal{Q}$. Mathematically, the operation is $\mathcal{U} = \mathcal{S}_{\sigma^{-1}} \mathcal{Q} \mathcal{S}_\sigma \mathcal{W}$. However, no actual SWAP gates are used. We note that this is exactly the "compose" operation in QISKIT.

A.2. Adding a gate: $\mathcal{U} \leftarrow$ qmsla.qc_add_gate($\mathcal{Q}, \mathcal{G}, \delta$). The qmsla.qc_add_gate_function constructs a new circuit by composing an existing circuit $\mathcal{Q}$ with the gate $\mathcal{G}$, which applied the qubits of $\mathcal{Q}$ whose indices appear in the qubit list δ. The gate is added at the end of the circuits, on the output of $\mathcal{Q}$, where the wires in δ outgoing from $\mathcal{Q}$ are connected to $\mathcal{G}$ using the ordering in δ. Mathematically, the operation is $\mathcal{U} = \mathcal{G}_\delta \mathcal{Q}$, where $\mathcal{G}_\delta$ denotes the circuit on $q(\mathcal{Q})$ qubits that consist solely of $\mathcal{G}$ applied to the qubits listed in δ.

A.3. Transpose, conjugate, and inverse of a circuit: $\mathcal{U}^{\mathrm{T}} \leftarrow$ qmsla.qc_transpose($\mathcal{U}$), $\overline{\mathcal{U}} \leftarrow$ qmsla.qc_conjugate($\mathcal{U}$), $\mathcal{U}^* \leftarrow$ qmsla.qc_inverse($\mathcal{U}$). The operations qmsla.qc_transpose, qmsla.qc_conjugate, and qmsla.qc_inverse implement the transpose of the unitary, the conjugate of the unitary, and the combination of both to obtain the inverse (adjoint) of the unitary (respectively). To obtain the inverse/adjoint circuit $\mathcal{U}^{-1}$ from a given circuit $\mathcal{U}$, reverse the order of gates and conjugate each gate. To create a conjugated circuit $\overline{\mathcal{U}}$ from a given circuit $\mathcal{U}$, conjugate each gate without changing the order. To create a transposed circuit $\mathcal{U}^{\mathrm{T}}$ from a circuit $\mathcal{U}$, we simply invert and conjugate. The inverse operation is typically implemented in most quantum computing frameworks (e.g., the "inverse" function in QISKIT), while the transpose and conjugate operations are relatively straightforward to implement in most frameworks, based on the description above. We summarize in the following lemma.

LEMMA 19.

1. *Given a classical description for a circuit $\mathcal{U}$ with n gates we can create a classical description of a circuit $\mathcal{U}^{\mathrm{T}}$ for which $\mathbf{M}(\mathcal{U}^{\mathrm{T}}) = \mathbf{M}(\mathcal{U})^{\mathrm{T}}$ in $O(n)$ operations. The depth and number of gates in the circuit are the same as for $\mathcal{U}$.*
2. *Given a classical description for a circuit $\mathcal{U}$ with n gates we can create a classical description of a circuit $\overline{\mathcal{U}}$ for which $\mathbf{M}(\overline{\mathcal{U}}) = \overline{\mathbf{M}(\mathcal{U})}$ in $O(n)$ operations. The depth and number of gates in the circuit are the same as for $\mathcal{U}$.*
3. *Given a classical description for a circuit $\mathcal{U}$ with n gates we can create a classical description of a circuit $\mathcal{U}^*$ for which $\mathbf{M}(\mathcal{U}^*) = \mathbf{M}(\mathcal{U})^*$ in $O(n)$ operations. The depth and number of gates in the circuit are the same as for $\mathcal{U}$.*

A.4. Efficient qubit rearrangement: $\mathcal{U} \leftarrow$ qmsla.qc_permute_bits($\mathcal{W}, \sigma$). Given a circuit $\mathcal{W}$ and qubit permutation σ, the goal of qmlsa.qc_permute_bits is to return a circuit that applies the qubit permutation σ on the output of $\mathcal{W}$, e.g., return $\mathcal{S}_\sigma \mathcal{W}$. However, for our purposes, it suffices that this holds only when the input

Algorithm 3 qmsla.qc_permute_bits.

Input: Classical description of circuit $\mathcal{W}$ and permutation σ

$\mathcal{W}^{-1} \leftarrow$ qmsla.qc_inverse($\mathcal{W}$)
$\mathcal{I} \leftarrow$ qmsla.qc_empty($q(\mathcal{W})$)
$\mathcal{Q} \leftarrow$ qmsla.qc_compose($\mathcal{I}, \mathcal{W}^{-1}, \sigma^{-1}$)
$\mathcal{U} \leftarrow$ qmsla.qc_inverse($\mathcal{Q}$)
return $\mathcal{U}$

is the ground state. That is, qmsla.qc_permute_bits($\mathcal{W}, \sigma$) returns a circuit $\mathcal{U}$ such that $\mathcal{U}|0\rangle_{q(\mathcal{W})} = \mathcal{S}_\sigma \mathcal{W}|0\rangle_{q(\mathcal{W})}$. Since we do require equality for all input states, just for the ground state, we can implement $\mathcal{U}$ without any SWAP gates, and with the same depth as $\mathcal{U}$. A pseudocode description is provided in Algorithm 3. We remark qmsla.qc_permute_bits is different from other level 0 operations in that it utilizes level 0 qMSLA operations, yet it does not qualify for level 1 operation due to its lack of clear linear algebraic interpretation.

Mathematically, we can verify that qmsla.qc_permute_bits indeed achieves its desired goals:

$$\begin{aligned}
\mathcal{U}\,|0\rangle_q &= \mathcal{Q}^{-1}\,|0\rangle_q \\
&= (\mathcal{S}_{\sigma_1}\mathcal{W}^{-1}\mathcal{S}_{\sigma_1^{-1}}\mathcal{I})^{-1}\,|0\rangle_q \\
&= (\mathcal{S}_{\sigma_1}(\mathcal{S}_{\sigma_1}\mathcal{W})^{-1}\mathcal{I})^{-1}\,|0\rangle_q \\
&= \mathcal{I}^{-1}\mathcal{S}_{\sigma_1}\mathcal{W}\mathcal{S}_{\sigma_1^{-1}}\,|0\rangle_q \\
&= \mathcal{S}_{\sigma_1}\mathcal{W}\,|0\rangle_q\,, \qquad (6)
\end{aligned}$$

where we used the fact that for any permutation σ we have that $\mathcal{S}_\sigma|0\rangle_q = |0\rangle_q$. This demonstrates that the output circuit $\mathcal{U}$ applies the permutation circuit $\mathcal{S}_\sigma$ to the original circuit $\mathcal{Q}$ on the ground state, as intended.

EliminatePermutations. We now explain the EliminatePermutations process mentioned in the main text. Suppose the output is a state preparation circuit $\mathcal{U}_\mathbf{X}$ for some matrix $\mathbf{X}$. Typically, $\mathcal{U}_\mathbf{X}$ starts with a series of SWAP gates that effectively implements a permutation on the qubits, next some circuit $\mathcal{Q}$ is executed, and finally another series of SWAP operations are executed, effectively implementing another permutation on the qubits. In short, the output can be written as $\mathcal{U}_\mathbf{X} = \mathcal{S}_{\sigma_1}\mathcal{Q}\mathcal{S}_{\sigma_2}$. However, conceptually, since qMSLA concerns only the matrix its outputs prepare (i.e., their operation on the ground state $|0\rangle$), any circuit $\mathcal{U}'_\mathbf{X}$ will do equally well as long as it is a state preparation circuit for $\mathbf{X}$ as well, even if $\mathbf{M}(\mathcal{U}_\mathbf{X}) \neq \mathbf{M}(\mathcal{U}'_\mathbf{X})$. Given σ_1, σ_2, and $\mathcal{Q}$, EliminatePermutations constructs a circuit $\mathcal{U}'_\mathbf{X}$ that prepares the same matrix as $\mathcal{U}_\mathbf{X} = \mathcal{S}_{\sigma_1}\mathcal{Q}\mathcal{S}_{\sigma_2}$ but does not use any SWAPs.

EliminatePermutations proceeds as follows. First, it eliminates any leading SWAP gates, as swaps applied to $|0\rangle$ have no effect, i.e., $\mathcal{S}_{\sigma_2}|0\rangle = |0\rangle$. Next, EliminatePermutations computes the inverse permutation σ_1^{-1}, inverts the circuit $\mathcal{Q}$, and composes the empty circuit with $\mathcal{Q}^{-1}$. However, in the composition, wire i is connected to wire $\sigma_1(i)$ in $\mathcal{Q}^{-1}$ input. Finally, EliminatePermutations inverts the output circuit, resulting in $\mathcal{U}'_\mathbf{X}$.

Relationship between qmsla.qc_permute_bits and ELIMINATE PERMUTATIONS. qMSLA operation qmsla.qc_permute_bits is key in applying the ELIMINATEPERMUTATIONS process. Suppose that for some output state preparation circuit $\mathcal{U}_{\mathbf{X}}$ we have identified a circuit $\mathcal{W}$ and two permutations σ_1, σ_2 such that $\mathcal{U}_{\mathbf{X}} = \mathcal{S}_{\sigma_1}\mathcal{W}\mathcal{S}_{\sigma_2}$. Then $\mathcal{U}'_{\mathbf{X}} = \text{qc\_permute\_bits}(\mathcal{W}, \sigma_1)$ is also a state preparation circuit for $\mathbf{X}$. This can be verified as follows:

$$\begin{aligned}\mathcal{U}'_{\mathbf{X}}\left|0\right\rangle_q &= \mathcal{S}_{\sigma_1}\mathcal{Q}\left|0\right\rangle_q \\ &= \mathcal{S}_{\sigma_1}\mathcal{Q}\mathcal{S}_{\sigma_2}\left|0\right\rangle_q \\ &= \mathcal{U}_{\mathbf{X}}\left|0\right\rangle_q.\end{aligned}$$

In the main text of the paper, wherever we write ELIMINATEPERMUTATIONS (in circuit diagrams and the text itself) we mean that when preparing that pseudocode for Appendix B we have identified the permutations and applied qc_permute_bits.

Appendix B. Implementation details for level 1 qMSLA. In this section, we give implementation details (pseudocode) for all level 1 qMSLA operations.

Pseudocodes are provided in Algorithms 4 and 5. However, to understand the pseudocode we need first to discuss how matrix state preparation circuits are represented therein. A state preparation circuit is a structure with three fields: the circuit

Algorithm 4 Pseudocode for all level-1 qMSLA operations (part 1).

qmsla.identity(n):
$q \leftarrow \log_2 n$
$\mathcal{I}_{2q} \leftarrow$ qmsla.qc_empty(2q)
$\mathcal{U} \leftarrow$ qmsla.qc_add_gate($\mathcal{I}_{2q}, \mathcal{H}, 0:q$){Add Hadamard gates on each qubit of the columns register}
$\mathcal{U} \leftarrow$ qmsla.qc_add_gate($\mathcal{U}$, CNOT, 0:q,q:2q){Add CNOT gates on row qubit controlled by corresponding column qubits}
return MatrixStatePreparation($\mathcal{U}, n, n$)

qmsla.matrix($\mathcal{U}$):
$q \leftarrow q(\mathcal{U})$
$\mathcal{U}_{\mathbf{I}_{2^q}} \leftarrow$ qmsla.identity($2q(\mathcal{U})$)
$\mathcal{U}_{\mathbf{M}(\mathcal{U})} \leftarrow$ qmsla.qc_compose($\mathcal{U}_{\mathbf{I}_{2^{q(\mathcal{U})}}}, \mathcal{U}_{\mathbf{I}_{2^{q(\mathcal{U})}}}$.rreg)
return MatrixStatePreparation($\mathcal{U}_{\mathbf{M}(\mathcal{U})}, 2^q, 2^q$)

qmsla.conjugate($\mathcal{U}_{\mathbf{A}}$):
$\mathcal{U}_{\overline{\mathbf{A}}} \leftarrow$ qmsla.qc_conjugate($\mathcal{U}_{\mathbf{A}}$)
return MatrixStatePreparation($\mathcal{U}_{\overline{\mathbf{A}}}$,$\mathcal{U}_{\mathbf{A}}.m$,$\mathcal{U}_{\mathbf{A}}.n$)

qmsla.transpose($\mathcal{U}_{\mathbf{A}}$):
$\sigma_{\mathrm{T}} \leftarrow \mathcal{U}_{\mathbf{A}}$.creg+$\mathcal{U}_{\mathbf{A}}$.rreg
$\mathcal{U}_{\mathbf{A}^{\mathrm{T}}} \leftarrow$ qmsla.qc_permute_bits($\mathcal{U}_{\mathbf{A}}$, σ_{T})
return MatrixStatePreparation($\mathcal{U}_{\mathbf{A}^{\mathrm{T}}}$,$\mathcal{U}_{\mathbf{A}}.n$,$\mathcal{U}_{\mathbf{A}}$.m)

qmsla.vec($\mathcal{U}_{\mathbf{A}}$):
return MatrixStatePreparation($\mathcal{U}_{\mathbf{A}}$,$\mathcal{U}_{\mathbf{A}}.m \cdot \mathcal{U}_{\mathbf{A}}$.n,1)

Algorithm 5 Pseudocode for all level-1 qMSLA operations (part 2).

qmsla.pad_zero_columns($\mathcal{U}_{\mathbf{A}}, k$):
$\mathcal{I}_{k+q(\mathcal{U}_{\mathbf{A}})} \leftarrow$ qmsla.qc_empty$(k + q(\mathcal{U}_{\mathbf{A}}))$
$\mathcal{U}_{\mathbf{A}\oplus 0_{0\times(2^k-1)n}} \leftarrow$ qmsla.qc_compose$(\mathcal{I}_{k+q(\mathcal{U}_{\mathbf{A}})}, \mathcal{U}_{\mathbf{A}}, k : (\log_2 mn + k))$
return MatrixStatePreparation$(\mathcal{U}_{\mathbf{A}\oplus 0_{0\times(2^k-1)n}}, \mathcal{U}_{\mathbf{A}}.m, (2^k-1)\cdot\mathcal{U}_{\mathbf{A}}.n)$

qmsla.matrix_vec($\mathcal{U}_{\mathbf{A}}$,$\mathcal{U}_{\mathbf{b}}$):
$\mathcal{U}_{\mathbf{b}}^{\mathrm{T}} \leftarrow$ qmsla.qc_transpose$(\mathcal{U}_{\mathbf{b}})$
$\mathcal{W} \leftarrow$ qmsla.qc_compose$(\mathcal{U}_{\mathbf{A}}, \mathcal{U}_{\mathbf{b}}^{\mathrm{T}}, \mathcal{U}_{\mathbf{A}}.\mathrm{creg}))$
$\mathcal{U}_{\frac{\mathbf{Ab}}{\|\mathbf{A}\|_F\|\mathbf{b}\|_2}\oplus\xi} \leftarrow$ qmsla.vec(MatrixStatePreperation$(\mathcal{W}, \mathcal{U}_{\mathbf{A}}.m, \mathcal{U}_{\mathbf{A}}.n)$)
return $\mathcal{U}_{\frac{\mathbf{Ab}}{\|\mathbf{A}\|_F\|\mathbf{b}\|_2}\oplus\xi}$

qmsla.kronecker($\mathcal{U}_{\mathbf{A}}$,$\mathcal{U}_{\mathbf{B}}$):
$\sigma_{\otimes} \leftarrow \mathcal{U}_{\mathbf{A}}.\mathrm{rreg}+\mathrm{shift}(\mathcal{U}_{\mathbf{B}}.\mathrm{rreg}, q_{\mathbf{A}}) + \mathcal{U}_{\mathbf{A}}.\mathrm{creg}+\mathrm{shift}(\mathcal{U}_{\mathbf{B}}.\mathrm{creg}, q_{\mathbf{A}})$
$\mathcal{U} \leftarrow$ qmsla.qc_tensor$(\mathcal{U}_{\mathbf{A}}, \mathcal{U}_{\mathbf{B}})$
$\mathcal{U} \leftarrow$ qmsla.qc_permute_bits$(\mathcal{U}, \sigma_{\otimes})$
return MatrixStatePreparation$(\mathcal{U}, \mathcal{U}_{\mathbf{A}}.m\cdot\mathcal{U}_{\mathbf{B}}.m, \mathcal{U}_{\mathbf{A}}.n\cdot\mathcal{U}_{\mathbf{B}}.n)$

qmsla.kronecker($\mathcal{U}_{\mathbf{A}_1}, \ldots, \mathcal{U}_{\mathbf{A}_k}$): (Assume $k \geq 2$)
$\mathcal{U} \leftarrow$ qmsla.qc_empty$(\sum_1^k q(\mathcal{U}_i))$
for $i = 1$ **to** $i = k$ **do**
  $\mathcal{U} \leftarrow$ qmsla.qc_compose$(\mathcal{U}, \mathcal{U}_{\mathbf{A}_i}, \sum_{j=1}^{i-1}\log_2 m_j n_j : \sum_{j=0}^{i}\log_2 m_j n_j)$
end for
$\sigma_{\otimes} \leftarrow \mathcal{U}_{\mathbf{A}_1}.\mathrm{rreg} + \mathrm{shift}(\mathcal{U}_{\mathbf{A}_2}.\mathrm{rreg}, q_{\mathbf{A}_1}) + \cdots + \mathrm{shift}(\mathcal{U}_{\mathbf{A}_k}.\mathrm{rreg}, \sum_{i=1}^{k-1} q_{\mathbf{A}_i}) +$
$\mathcal{U}_{\mathbf{A}_1}.\mathrm{creg} + \mathrm{shift}(\mathcal{U}_{\mathbf{A}_2}.\mathrm{creg}, q_{\mathbf{A}_1}) + \cdots + \mathrm{shift}(\mathcal{U}_{\mathbf{A}_k}.\mathrm{creg}, \sum_{i=1}^{k-1} q_{\mathbf{A}_i})$
$\mathcal{U}_{\mathbf{A}_1\otimes\mathbf{A}_2\otimes\cdots\otimes\mathbf{A}_k} \leftarrow$ qmsla.qc_permute_bits$(\mathcal{U}, \sigma_{\otimes})$
return MatrixStatePreperation$(\mathcal{U}_{\mathbf{A}_1\otimes\mathbf{A}_2\otimes\cdots\otimes\mathbf{A}_k}, \prod_{i=1}^k \mathcal{U}_{\mathbf{A}_i}.m, \prod_{i=1}^k \mathcal{U}_{\mathbf{A}_i}.n)$

($\mathcal{U}$), the number of rows in the prepared matrix (m), and the number of columns (n). In the pseudocode, a state preparation circuit is constructed using the constructor MatrixStatePreperation$(\mathcal{U}, m, n)$. The various fields of a state preparation structure $\mathcal{U}_{\mathbf{A}}$ are accessed in the pseudocode as follows: $\mathcal{U}_{\mathbf{A}}.m$ for the number of rows, $\mathcal{U}_{\mathbf{A}}.n$ for the number of columns, and $\mathcal{U}_{\mathbf{A}}$ for the circuit itself. Pseudocodes also use two logical accessors: $\mathcal{U}_{\mathbf{A}}$.creg and $\mathcal{U}_{\mathbf{A}}$.rreg. $\mathcal{U}_{\mathbf{A}}$.creg returns the list of qubits that correspond to the column register, and $\mathcal{U}_{\mathbf{A}}$.rreg returns the list of qubits that correspond to the row register.

If qubits are provided as a numerical list, we use colon notation: $0:n$ represents elements from 0 to $n-1$ (inclusive), and $m:n$ represents elements from m to $n-1$ (inclusive). Concatenating lists is denoted by the plus operator. The shift operator shifts the indices in a list by a specified number.

Appendix C. A more efficient MVTracePrep. In this section, we propose MVTracePrepOptimized—a more efficient variant of MVTracePrep than the variant in the main text. MVTracePrep translates Theorem 18 directly into a circuit via the series of qMSLA operations. This showcases the appeal of qMSLA's approach. Each qMSLA operation is efficient, with cost that is proportional

to the size of the output (i.e., $O(g(\text{output}))$), so the overall cost is rather attractive: $O(\log k \sum_{i=2}^{2k} g_{\mathbf{A}_i})$. Nevertheless, careful inspection of the final output reveals that a more efficient construction is possible, one that costs only $O(\sum_{i=2}^{2k} g_{\mathbf{A}_i})$, though it is no longer a straightforward translation of Theorem 18.

The key observation is that if we do not apply EliminatePermutation at all, the output circuit $\mathcal{U}_\phi$ has a very special structure. First, there is a layer which consists of the tensor product of a subset of the circuits $\{\mathcal{U}_{\mathbf{A}_i}\}$. Then there is a series of SWAPs that implements a permutation. Finally, there is another layer which consists of the tensor product of another subset of the circuits $\{\mathcal{U}_{\mathbf{A}_i}\}$. See Figure 9 for an illustration. Thus, to build the circuit, we can build the first layer, build the last layer, compute the middle permutation, and finally compose the first layer with the final layer with the qubit permutation in a gate efficient manner via qmsla.qc_compose.

MVTracePrepOptimized (Algorithm 7) does exactly this. The main challenge is in finding the qubit permutation of the SWAP layer. MVTracePrepOptimized finds the permutation by tracking the execution of the various qMSLA operations in MVTracePrep logically, i.e., computing only the qubit permutation they induce. See Algorithm 6 for pseudocode for implementing this. MVTracePrepOptimized uses Algorithm 6 as a subroutine.

Algorithm 6 GenerateMVTracePrepPermutation.

Input: List of reindexed qubit registers from 0 to total number of qubits: $[\text{rreg}_1, \text{creg}_1, \ldots, \text{rreg}_{2k}, \text{creg}_{2k}]$
$p \leftarrow k+1$
if k is odd **then**
  $l_{\text{even}} \leftarrow (k+1)/2$ and $l_{\text{odd}} \leftarrow (k-1)/2$
  $\tau_{\mathbf{F}^{(1)}_{\text{even}}} \leftarrow \{\text{creg} = \text{rreg}_p, rreg = \text{creg}_p\}$ {Permutation for $\mathbf{E}_{(k+1)/2,k} = \mathbf{A}_p^{\mathrm{T}}$}
  $\tau_{\mathbf{F}^{(1)}_{\text{odd}}} \leftarrow \{\text{creg} = \text{creg}_{p+1} + \text{rreg}_{p-1}, \text{rreg} = \text{rreg}_{p+1} + \text{creg}_{p-1}\}$ {Permutation for $\mathbf{O}_{i,k} = \mathbf{A}_{p+1} \otimes \mathbf{A}_{p-1}^{\mathrm{T}}$}
else
  $l_{\text{even}} \leftarrow k/2$ and $l_{\text{odd}} \leftarrow k/2$
  $\tau_{\mathbf{F}^{(1)}_{\text{even}}} \leftarrow \{\text{creg} = \text{creg}_{p+1} + \text{rreg}_{p-1}, \text{rreg} = \text{rreg}_{p+1} + \text{creg}_{p-1}\}$ {Permutation for $\mathbf{E}_{i,k} = \mathbf{A}_{p+1} \otimes \mathbf{A}_{p-1}^{\mathrm{T}}$}
  $\tau_{\mathbf{F}^{(1)}_{\text{odd}}} \leftarrow \{\text{creg} = \text{rreg}_p, \text{rreg} = \text{creg}_p\}$ {Permutation for $\mathbf{O}_{k/2,k} = \mathbf{A}_p^{\mathrm{T}}$}
end if
for $i = 2$ **to** $i = 2k - p$ **do**
  if $p + r$ is odd **then**
    $\tau_{\mathbf{O}_{\text{i,k}}} \leftarrow \{\text{creg} = \text{creg}_{p+i} + \text{rreg}_{p-i}, \text{rreg} = \text{rreg}_{p+i} + \text{creg}_{p-i}\}$
    $\tau_{\mathbf{F}^{(i)}_{\text{odd}}} \leftarrow \{\text{creg} = \tau_{\mathbf{O}_{\text{i,k}}}.\text{creg}, \text{rreg} = \tau_{\mathbf{O}_{\text{i,k}}}.\text{rreg} + \tau_{\mathbf{F}^{(i-1)}_{\text{odd}}}.\text{creg} + \tau_{\mathbf{F}^{(i-1)}_{\text{odd}}}.\text{rreg}\}$
  else
    $\tau_{\mathbf{E}_{\text{i,k}}} \leftarrow \{\text{creg} = \text{creg}_{p+i} + \text{rreg}_{p-i}, \text{rreg} = \text{rreg}_{p+i} + \text{creg}_{p-i}\}$
    $\tau_{\mathbf{F}^{(i)}_{\text{even}}} \leftarrow \{\text{creg} = \tau_{\mathbf{E}_{\text{i,k}}}.\text{creg}, \text{rreg} = \tau_{\mathbf{E}_{\text{i,k}}}.\text{rreg} + \tau_{\mathbf{F}^{(i-1)}_{\text{even}}}.\text{creg} + \tau_{\mathbf{F}^{(i-1)}_{\text{even}}}.\text{rreg}\}$
  end if
end for
$\sigma^{-1}_{\text{even}} \leftarrow \mathbf{F}^{(l_{\text{even}})}_{\text{even}}.\text{creg} + \mathbf{F}^{(l_{\text{even}})}_{\text{even}}.\text{rreg}$
$\sigma^{-1}_{\text{odd}} \leftarrow \mathbf{F}^{(l_{\text{odd}})}_{\text{odd}}.\text{creg} + \mathbf{F}^{(l_{\text{odd}})}_{\text{odd}}.\text{rreg}$
return $\sigma := \sigma_{\text{even}} \sigma^{-1}_{\text{odd}}$

Algorithm 7 MVTracePrepOptimized.

Input: Classical description of the circuits $\mathcal{U}_{\mathbf{A}_1}, \ldots, \mathcal{U}_{\mathbf{A}_{2k}}$ where $\mathbf{A}_i \in \mathbb{C}^{m_i \times n_i}$

{Constructing $\mathcal{U}_\psi$}
$\mathcal{U}_{\overline{\mathbf{A}_1}} \leftarrow$ qmsla.conjugate($\mathcal{U}_{\mathbf{A}_1}$)
$\mathcal{U}_\psi \leftarrow$ qmsla.pad(qmsla.vec($\mathcal{U}_{\overline{\mathbf{A}_1}}$), $\log_2 n_3 n_4 \cdots n_{2k}, 0$)

{Constructing $\mathcal{U}_\phi$}
all_regs $\leftarrow$ []
idx $= 0$
for $i = 1$ **to** $i = 2k$ **do**
    reg_i $\leftarrow$ idx + [$\mathcal{U}_{\mathbf{A}_i}$.rreg + $\mathcal{U}_{\mathbf{A}_i}$.creg] {concat qubits list and add idx for each element in the list}
    idx $\leftarrow$ idx + size(reg_i)
    all_regs $\leftarrow$ all_regs.append(reg_i)
end for
$\sigma \leftarrow$ GenerateMVTracePrepPermutation(all_regs)
$\mathcal{U}_{\text{even}} \leftarrow \mathcal{U}_{\mathbf{A}_2}$
$\mathcal{U}_{\text{odd}} \leftarrow \mathcal{U}^{-1}_{\mathbf{A}_3}$
for $i = 2$ **to** $i = k$ **do**
    $\mathcal{U}_{\text{even}} \leftarrow$ qmsla.qc_tensor($\mathcal{U}_{\text{even}}, \mathcal{U}_{\mathbf{A}_{2i}}$)
    if $2i < 2k - 2i$ **then**
        $\mathcal{U}_{\text{even}} \leftarrow$ qmsla.qc_tensor($\mathcal{U}_{\text{even}}, \mathcal{U}_{\mathbf{A}_{2k-2i}}$)
    end if
    $\mathcal{U}_{\text{odd}} \leftarrow$ qmsla.qc_tensor($\mathcal{U}_{\text{odd}}, \mathcal{U}^{-1}_{\mathbf{A}_{2i+1}}$)
    if $2i < 2k - 2i$ **then**
        $\mathcal{U}_{\text{odd}} \leftarrow$ qmsla.qc_tensor($\mathcal{U}_{\text{odd}}, \mathcal{U}^{-1}_{\mathbf{A}_{2k-2i-1}}$)
    end if
end for
$\mathcal{U}_\phi \leftarrow$ qmsla.compose($\mathcal{U}_{\text{even}}, \mathcal{U}_{\text{odd}}, \sigma$)
return $\mathcal{U}_\psi, \mathcal{U}_\phi$

REFERENCES

[1] A. Ambainis, *Variable time amplitude amplification and quantum algorithms for linear algebra problems*, in 29th Symposium on Theoretical Aspects of Computer Science, LIPIcs. Leibniz Int. Proc. Inform. 14, Schloss Dagstuhl – Leibniz Center for Informatics, Wadern, 2012, pp. 636–647.

[2] D. An and L. Lin, *Quantum linear system solver based on time-optimal adiabatic quantum computing and quantum approximate optimization algorithm*, ACM Trans. Quantum Comput., 3 (2022), pp. 1–28, https://doi.org/10.1145/3498331.

[3] I. F. Araujo, D. K. Park, F. Petruccione, and A. J. da Silva, *A divide-and-conquer algorithm for quantum state preparation*, Sci. Rep., 11 (2021), 6329, https://doi.org/10.1038/s41598-021-85474-1.

[4] H. Avron and S. Toledo, *Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix*, J. ACM, 58 (2011), pp. 1–34, https://doi.org/10.1145/1944345.1944349.

[5] Z. Bai, G. Fahey, and G. Golub, *Some large-scale matrix computation problems*, J. Comput. Appl. Math., 74 (1996), pp. 71–89, https://doi.org/10.1016/0377-0427(96)00018-0.

[6] D. S. Bernstein, *Matrix Mathematics: Theory, Facts, and Formulas*, 2nd ed., Princeton University Press, Princeton, NJ, 2011.

[7] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, *Quantum machine learning*, Nature, 549 (2017), pp. 195–202, https://doi.org/10.1038/nature23474.

[8] S. Bravyi, A. Chowdhury, D. Gosset, and P. Wocjan, *Quantum Hamiltonian complexity in thermal equilibrium*, Nat. Phys., 18 (2022), pp. 1367–1370, https://doi.org/10.1038/s41567-022-01742-5.

[9] T. Brydges, A. Elben, P. Jurcevic, B. Vermersch, C. Maier, B. P. Lanyon, P. Zoller, R. Blatt, and C. F. Roos, *Probing Rényi entanglement entropy via randomized measurements*, Science, 364 (2019), pp. 260–263, https://doi.org/10.1126/science.aau4963.

[10] H. Buhrman, R. Cleve, J. Watrous, and R. De Wolf, *Quantum fingerprinting*, Phys. Rev. Lett., 87 (2001), 167902, https://doi.org/10.1103/PhysRevLett.87.167902.

[11] C. Cade and A. Montanaro, *The quantum complexity of computing Schatten p-norms*, in 13th Conference on the Theory of Quantum Computation, Communication and Cryptography, S. Jeffery, ed., LIPIcs. Leibniz Int. Proc. Inform. 111, Schloss Dagstuhl – Leibniz Center for Informatics, Wadern, 2018, pp. 4:1–4:20, https://doi.org/10.4230/LIPIcs.TQC.2018.4.

[12] E. J. Candès and B. Recht, *Exact matrix completion via convex optimization*, Found. Comput. Math., 9 (2009), pp. 717–772, https://doi.org/10.1007/s10208-009-9045-5.

[13] S. Chakraborty, A. Gilyén, and S. Jeffery, *The power of block-encoded matrix powers: Improved regression techniques via faster Hamiltonian simulation*, in 46th International Colloquium on Automata, Languages, and Programming, C. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi, eds., LIPIcs. Leibniz Int. Proc. Inform. 132, Schloss Dagstuhl – Leibniz Center for Informatics, Wadern, 2019, pp. 33:1–33:14, https://doi.org/10.4230/LIPIcs.ICALP.2019.33.

[14] T. Chen, T. Trogdon, and S. Ubaru, *Analysis of stochastic Lanczos quadrature for spectrum approximation*, in Proceedings of the International Conference on Machine Learning, PMLR, 2021, pp. 1728–1739.

[15] N. Chepurko, K. Clarkson, L. Horesh, H. Lin, and D. Woodruff, *Quantum-inspired algorithms from randomized numerical linear algebra*, in Proceedings of the International Conference on Machine Learning, PMLR, 2022, pp. 3879–3900.

[16] N.-H. Chia, A. Gilyén, T. Li, H.-H. Lin, E. Tang, and C. Wang, *Sampling-based sublinear low-rank matrix arithmetic framework for dequantizing quantum machine learning*, in Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, 2020, pp. 387–400.

[17] A. M. Childs, R. Kothari, and R. D. Somma, *Quantum algorithm for systems of linear equations with exponentially improved dependence on precision*, SIAM J. Comput., 46 (2017), pp. 1920–1950, https://doi.org/10.1137/16M1087072.

[18] B. D. Clader, A. M. Dalzell, N. Stamatopoulos, G. Salton, M. Berta, and W. J. Zeng, *Quantum resources required to block-encode a matrix of classical data*, IEEE Trans. Quantum Eng., 3 (2022), pp. 1–23, https://doi.org/10.1109/TQE.2022.3231194.

[19] A. Cortinovis and D. Kressner, *On randomized trace estimates for indefinite matrices with an application to determinants*, Found. Comput. Math., 22 (2022), pp. 1–29, https://doi.org/10.1007/s10208-021-09525-9.

[20] X. Cui and Y. Shi, *QBLAS: A Quantum Basic Linear Algebra and Simulation Library*, https://github.com/xpclove/qblas.

[21] G. M. D'Ariano and P. Lo Presti, *Quantum tomography for measuring experimentally the matrix elements of an arbitrary quantum operation*, Phys. Rev. Lett., 86 (2001), pp. 4195–4198, https://doi.org/10.1103/PhysRevLett.86.4195.

[22] J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon, *Information-theoretic metric learning*, in Proceedings of the 24th International Conference on Machine Learning, ACM, 2007, pp. 209–216.

[23] O. Di Matteo, V. Gheorghiu, and M. Mosca, *Fault-tolerant resource estimation of quantum random-access memories*, IEEE Trans. Quantum Eng., 1 (2020), pp. 1–13, https://doi.org/10.1109/TQE.2020.2965803.

[24] A. K. Ekert, C. M. Alves, D. K. L. Oi, M. Horodecki, P. Horodecki, and L. C. Kwek, *Direct estimations of linear and nonlinear functionals of a quantum state*, Phys. Rev. Lett., 88 (2002), 217901, https://doi.org/10.1103/PhysRevLett.88.217901.

[25] E. Estrada, *Characterization of 3D molecular structure*, Chem. Phys. Lett., 319 (2000), pp. 713–718, https://doi.org/10.1016/S0009-2614(00)00158-5.

[26] N. Feldman, A. Kshetrimayum, J. Eisert, and M. Goldstein, *Entanglement estimation in tensor network states via sampling*, PRX Quantum, 3 (2022), 030312, https://doi.org/10.1103/PRXQuantum.3.030312.

[27] B. Ghorbani, S. Krishnan, and Y. Xiao, *An investigation into neural net optimization via Hessian eigenvalue density*, in Proceedings of the International Conference on Machine Learning, PMLR, 2019, pp. 2232–2241.

[28] A. Gilyén, Y. Su, G. H. Low, and N. 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, 2019, pp. 193–204.

[29] V. Giovannetti, S. Lloyd, and L. Maccone, *Quantum random access memory*, Phys. Rev. Lett., 100 (2008), 160501, https://doi.org/10.1103/PhysRevLett.100.160501.

[30] D. Girard, *Un algorithme simple et rapide pour la validation croisée generalisée sur des problèmes de grande taille [A fast and efficient procedure for generalized cross-validation with large data sets]: Applications à la restauration d'image [Applications for image restoration]*, Rapport de recherche: Institut d'Informatique et de Mathématiques Appliquées, IMAG, 1987, https://books.google.co.il/books?id=_lILrgEACAAJ.

[31] N. Gleinig and T. Hoefler, *An efficient algorithm for sparse quantum state preparation*, in Proceedings of the 58th ACM/IEEE Design Automation Conference, IEEE, 2021, pp. 433–438.

[32] G. H. Golub and G. Meurant, *Matrices, Moments and Quadrature with Applications*, Princeton Ser. Appl. Math. 30, Princeton University Press, Princeton, NJ, 2009.

[33] I. Han, D. Malioutov, H. Avron, and J. Shin, *Approximating spectral sums of large-scale matrices using stochastic Chebyshev approximations*, SIAM J. Sci. Comput., 39 (2017), pp. A1558–A1585, https://doi.org/10.1137/16M1078148.

[34] M. Hardt, K. Ligett, and F. McSherry, *A simple and practical algorithm for differentially private data release*, in Advances in Neural Information Processing Systems, 2012, pp. 2339–2347.

[35] A. W. Harrow, A. Hassidim, and S. Lloyd, *Quantum algorithm for linear systems of equations*, Phys. Rev. Lett., 103 (2009), 150502, https://doi.org/10.1103/PhysRevLett.103.150502.

[36] P. Horodecki and A. Ekert, *Method for direct detection of quantum entanglement*, Phys. Rev. Lett., 89 (2002), 127902, https://doi.org/10.1103/PhysRevLett.89.127902.

[37] M. F. Hutchinson, *A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines*, Commun. Stat. Simul. Comput., 19 (1990), pp. 433–450, https://doi.org/10.1080/03610919008812866.

[38] J. Iaconis, S. Johri, and E. Y. Zhu, *Quantum state preparation of normal distributions using matrix product states*, npj Quantum Inform., 10 (2024), 15, https://doi.org/10.1038/s41534-024-00805-0.

[39] I. Kerenidis and A. Prakash, *Quantum recommendation systems*, in 8th Innovations in Theoretical Computer Science Conference, LIPIcs. Leibniz Int. Proc. Inform. 67, C. H. Papadimitriou, ed., Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, 2017, pp. 49:1–49:21, https://doi.org/10.4230/LIPIcs.ITCS.2017.49.

[40] I. Kerenidis and A. Prakash, *Quantum gradient descent for linear systems and least squares*, Phys. Rev. A, 101 (2020), 022316, https://doi.org/10.1103/PhysRevA.101.022316.

[41] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, *Basic linear algebra subprograms for Fortran usage*, ACM Trans. Math. Software, 5 (1979), pp. 308–323, https://doi.org/10.1145/355841.355847.

[42] M. S. Leifer, N. Linden, and A. Winter, *Measuring polynomial invariants of multiparty quantum states*, Phys. Rev. A, 69 (2004), 052304, https://doi.org/10.1103/PhysRevA.69.052304.

[43] J.-M. Liang, Q.-Q. Lv, Z.-X. Wang, and S.-M. Fei, *Unified multivariate trace estimation and quantum error mitigation*, Phys. Rev. A, 107 (2023), 012606, https://doi.org/10.1103/PhysRevA.107.012606.

[44] L. Lin, *Lecture Notes on Quantum Algorithms for Scientific Computation*, preprint, arXiv:2201.08309, 2022.

[45] L. Lin and Y. Tong, *Optimal polynomial based quantum eigenstate filtering with application to solving quantum linear systems*, Quantum, 4 (2020), 361, https://doi.org/10.22331/q-2020-11-11-361.

[46] A. Luongo and C. Shao, *Quantum Algorithms for Spectral Sums*, preprint, arXiv:2011.06475, 2020.

[47] D. J. C. MacKay, *Information Theory, Inference & Learning Algorithms*, Cambridge University Press, Cambridge, UK, 2002.

[48] J. M. Martyn, Z. M. Rossi, A. K. Tan, and I. L. Chuang, *Grand unification of quantum algorithms*, PRX Quantum, 2 (2021), 040203, https://doi.org/10.1103/PRXQuantum.2.040203.

[49] R. A. Meyer, C. Musco, C. Musco, and D. P. Woodruff, *Hutch++: Optimal stochastic trace estimation*, in Proceedings of the Symposium on Simplicity in Algorithms, SIAM, Philadelphia, 2021, pp. 142–155, https://doi.org/10.1137/1.9781611976496.16.

[50] D. Patel and M. M. Wilde, *Wave matrix Lindbladization* I*: Quantum programs for simulating Markovian dynamics*, Open Syst. Inf. Dyn., 30 (2023), 2350010, https://doi.org/10.1142/S1230161223500105.

[51] K. Phalak, A. Chatterjee, and S. Ghosh, *Quantum random access memory for dummies*, Sensors, 23 (2023), 7462, https://doi.org/10.3390/s23177462.

[52] A. Prakash, *Quantum Algorithms for Linear Algebra and Machine Learning*, Technical report UCB/EECS-2014-211, University of California, Berkeley, 2014, http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-211.html.

[53] Y. Quek, E. Kaur, and M. M. Wilde, *Multivariate trace estimation in constant quantum depth*, Quantum, 8 (2024), 1220, https://doi.org/10.22331/q-2024-01-10-1220.

[54] A. Ramesh and Y. LeCun, *Backpropagation for Implicit Spectral Densities*, preprint, arXiv:1806.00499, 2018.

[55] C. E. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*, MIT Press, Cambridge, MA, 2006.

[56] F. Roosta-Khorasani and U. Ascher, *Improved bounds on sample size for implicit matrix trace estimators*, Found. Comput. Math., 15 (2015), pp. 1187–1212, https://doi.org/10.1007/s10208-014-9220-1.

[57] M. Schuld and N. Killoran, *Quantum machine learning in feature Hilbert spaces*, Phys. Rev. Lett., 122 (2019), 040504, https://doi.org/10.1103/PhysRevLett.122.040504.

[58] M. Schuld and F. Petruccione, *Machine Learning with Quantum Computers*, Springer, New York, 2021.

[59] M. Schuld, I. Sinayskiy, and F. Petruccione, *An introduction to quantum machine learning*, Contemp. Phys., 56 (2015), pp. 172–185, https://doi.org/10.1080/00107514.2014.964942.

[60] C. Shao, *Quantum Algorithms to Matrix Multiplication*, preprint, arXiv:1803.01601, 2018.

[61] Y. Shen, K. Klymko, E. Rabani, D. Camps, R. Van Beeumen, and M. Lindsey, *Efficient Quantum Trace Estimation with Reconfigurable Real-Time Circuits*, preprint, arXiv:2401.04176, 2024.

[62] V. V. Shende, S. S. Bullock, and I. L. Markov, *Synthesis of quantum logic circuits*, in Proceedings of the Asia and South Pacific Design Automation Conference, 2005, pp. 272–275.

[63] D. Shepherd, *Computation with Unitaries and One Pure Qubit*, preprint, https://arxiv.org/abs/quant-ph/0608132, 2006.

[64] Y. Subasi, R. D. Somma, and D. Orsucci, *Quantum algorithms for systems of linear equations inspired by adiabatic quantum computing*, Phys. Rev. Lett., 122 (2019), 060504, https://doi.org/10.1103/PhysRevLett.122.060504.

[65] E. Tang, *A quantum-inspired classical algorithm for recommendation systems*, in Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, 2019, pp. 217–228.

[66] L. N. Trefethen, *Approximation Theory and Approximation Practice*, SIAM, Philadelphia, 2019, https://doi.org/10.1137/1.9781611975949 .

[67] S. Ubaru, J. Chen, and Y. Saad, *Fast estimation of* $tr(f(a))$ *via stochastic Lanczos quadrature*, SIAM J. Matrix Anal. Appl., 38 (2017), pp. 1075–1099, https://doi.org/10.1137/16m1104974.

[68] S. Ubaru and Y. Saad, *Applications of trace estimation techniques*, in High Performance Computing in Science and Engineering: Third International Conference, Springer, New York, 2017, pp. 19–33.