Breaking the Treewidth Barrier in Quantum Circuit Simulation with Decision Diagrams

Bin Cheng[*1], Ziyuan Wang[*2,3], Ruixuan Deng[2], Jianxin Chen[†2], and Zhengfeng Ji[†2,3]

[1] *Centre for Quantum Technologies, National University of Singapore*
[2] *Department of Computer Science and Technology, Tsinghua University*
[3] *Zhongguancun Laboratory*

October 9, 2025

Abstract

Classical simulation of quantum circuits is a critical tool for validating quantum hardware and probing the boundary between classical and quantum computational power. Existing state-of-the-art methods, notably tensor network approaches, have computational costs governed by the treewidth of the underlying circuit graph, making circuits with large treewidth intractable. This work rigorously analyzes FeynmanDD, a decision diagram-based simulation method proposed in CAV 2025 by a subset of the authors, and shows that the size of the multi-terminal decision diagram used in FeynmanDD is exponential in the *linear rank-width* of the circuit graph. As linear rank-width can be substantially smaller than treewidth and is at most larger than the treewidth by a logarithmic factor, our analysis demonstrates that FeynmanDD outperforms all tensor network-based methods for certain circuit families. We also show that the method remains efficient if we use the Solovay-Kitaev algorithm to expand arbitrary single-qubit gates to sequences of Hadamard and T gates, essentially removing the gate-set restriction posed by the method.

1 Introduction

Classical simulation of quantum circuits plays a central role in advancing our understanding of quantum computation. On one hand, the limitation of classical methods for simulating universal quantum circuits that are revealed during the study reinforces the belief that quantum computers possess computational capabilities beyond those of classical systems [Got98, Val02]. On the other hand, classical simulation techniques provide indispensable tools for validating experimental claims of quantum computation (see e.g. [KLR+08, MGE11, AAB+19]), particularly in the context of recent quantum supremacy demonstrations [AAB+19]. Identifying the precise point

[*]First authors.
[†]Correspondence authors. Email: {chenjianxin,jizhengfeng}@tsinghua.edu.cn.

at which quantum computation outperform classical ones is therefore a question of both practical and theoretical significance.

Over the past decades, a rich set of classical simulation techniques has emerged. Prominent among these are the efficient simulation of Clifford circuits enabled by the celebrated Gottesman-Knill theorem [Got98], algorithms exploiting low T-count structures [BG16], matchgate simulation techniques [Val02], and, most notably, tensor network-based approaches [MS08, HZN+21, PGVWC07, O'G19]. Among these, tensor networks have emerged as particularly powerful tools. They are widely used in many-body physics as the numerical computation engine to study many-body phenomena [Vid04, Vid08, Orú14] and have become the method of choice for simulating general quantum circuits, especially those lacking special algebraic structures like in Clifford or matchgate circuits.

The performance of tensor network-based algorithms is governed by an important graph parameter of the underlying circuit: the treewidth of the associated tensor network [MS08]. The treewidth measures how close a graph is to a tree and is, roughly speaking, a measure of the complexity of entangling structure of the circuit. Both the time and space complexity of tensor network-based simulation algorithms are bounded by $2^{O(w)}\,\mathrm{poly}(m)$, where w is the treewidth and m is the number of gates in the circuit. There are refinements and many variants of classical simulation methods that can be characterized by other graph parameters, such as the path-width [ALM07] the vertex congestion and modified cut width [O'G19], but these are within linearly or polylogarithmically factors of the treewidth and thus are only of practical importance for implementation and do not affect the theoretical bounds of the related algorithms. From a parameterized complexity perspective [FG06], these tensor network-based methods are classical fixed-parameter tractable (FPT) algorithms using treewidth as the parameter.

For generic quantum circuits with little inherent structure, tensor network algorithms with complexity exponential in the treewidth have represented the state of the art for decades. This naturally raises the question:

Is the treewidth of the tensor network an inherent barrier to the classical simulation of general quantum circuits, or can it be overcome using fundamentally different simulation techniques?

In this work, we answer this question in the affirmative. We show that the recently introduced FeynmanDD method [WCYJ25] admits a rigorous complexity analysis in terms of the so called *linear rank-width* of the same circuit graph. The approach departs entirely from the tensor network paradigm, relying instead on *binary decision diagrams* (BDDs) [Bry86, Bry95, Weg00, Knu09] as the core computational data structure. Importantly, there exist families of circuits where the linear rank-width is significantly smaller than the treewidth, suggesting the potential for substantial speedups in classical simulation. Thus, the FeynmanDD method opens a new regime for classical simulation beyond the treewidth barrier, paving the way for its use in validating, benchmarking, and advancing our understanding of quantum computation.

At the core of the FeynmanDD algorithm lies a Feynman path integral type of simulation powered by an underlying decision diagram data structure. For a circuit C using many supported discrete gate sets, including $\mathcal{Z} = \{\mathrm{H}, \mathrm{CCZ}\}$, $\mathcal{T} = \{\mathrm{H}, \mathrm{T}, \mathrm{CZ}\}$,

and the iSWAP Google supremacy gate set $\mathcal{G} = \{\sqrt{\mathrm{X}}, \sqrt{\mathrm{Y}}, \sqrt{\mathrm{W}}, \mathrm{iSWAP}\}$[1], many simulation tasks such as computing the amplitude, estimating the acceptance probability, and sampling the outcome of a computational basis measurement can be formulated as the computation of an exponential sum known as a sum-of-power (SOP) form: $\frac{1}{R}\sum_x \omega_r^{f_C(x)}$, where r is an integer modulus determined by the gate set and is a fixed small constant, R is a normalization factor, and f_C is a low-degree polynomial over $\{0,1\}$ taking integer values modulo r. The computation of the exponential sum in the SOP can be reduced to a constant number of counting problems that compute the size of $\{x \mid f_C(x) = j\}$. The key insight of FeynmanDD is that if the function $f_C(x)$ has a succinct r-terminal decision diagram, the counting can be performed efficiently. Even though the method borrows a lot from the classical decision diagram literature, the heavy use of the efficient counting algorithm of BDDs to do circuit analysis is, to the best of our knowledge, unique to the FeynmanDD method.

The decision diagrams employed in FeynmanDD have long been a standard tool in classical computer science for representing and manipulating Boolean circuits and functions efficiently. Introduced in the 1980s [Bry86], BDDs have since become central to applications such as circuit synthesis, formal verification, and model checking. Their impact has been widely recognized, and many textbooks are dedicated to this topic [Weg00, Knu09, MMBS04]. In *The Art of Computer Programming*, Knuth dedicated an entire section to the theory and applications of BDDs [Knu09].

BDDs possess several important features and properties, and the key result we rely on to prove the linear rank-width bound is an explicit bound on the number of nodes at any level of the diagram—that is, the number of distinct functions obtained by assigning values to the preceding variables. Specifically, for a function $f(x_1, x_2, \ldots, x_n)$ and a variable order $x_1, x_2, \ldots, x_n$, the number of nodes at the i-th level is given by the number of distinct functions that essentially depend on x_i and result from assigning values to $x_1, x_2, \ldots, x_{i-1}$ in f. Although this number can be as large as 2^{i-1}, it can be significantly smaller for certain functions and can be linked to the linear rank-width of the underlying graph. Our approach combines an explicit bound on the number of nodes at each level of the Feynman diagram with a tailored, level-by-level BDD construction algorithm. By integrating these techniques, we demonstrate that the space and time complexity of our FeynmanDD framework is exponential in the linear rank-width of the circuit graph, particularly for two-qubit gate sets such as $\mathcal{T}$ and $\mathcal{G}$.

Using this rank-width characterization, we also formally prove that diagonal single-qubit gates such as T have limited impact on the complexity and that FeynmanDD is efficient even for circuits with arbitrary single-qubit gates. Specifically, we show that: first, the number of nodes at each level increases by at most a factor of r when diagonal gates like T gates are introduced; and second, adding a sequence of H and T gates to the circuit increases the linear rank-width of the circuit graph by at most 2. Together with the Solovay-Kitaev theorem [Kit97, NC00], this result significantly mitigates the restriction of the method, which originally applied only to discrete gate sets. Thus, in essence, the complexity of the method is determined by the entangling backbone of the circuit, consisting of CZ and H gates. Note that the backbone of CZ and H gates are Clifford gates, but they are not necessarily efficient to simulate using

[1] In [WCYJ25], gate sets containing fSim were also supported. However, it is technically complicated to handle using the methods in this paper, so we do not consider it here.

FeynmanDD; however, if they are, then one can actually simulate the circuit when an arbitrary number of T gates are added to the backbone with tolerable overhead.

As an interesting remark, we point out that for a circuit of n qubits and m gates, it is well-known that Schrödinger-type simulation has space and time complexities of $2^{O(n)}$ and $m2^{O(n)}$, respectively. The Feynman path integral method is space-efficient, requiring only polynomial space, but its time complexity is $2^{\Omega(m)}$, which is exponential in the number of gates and typically far worse than the Schrödinger method. Interestingly, however, decision diagrams can somehow automatically organize the paths to avoid this drawback of Feynman-style simulation and can even outperform tensor-network based methods for certain circuit families, as our analysis indicates.

This work leaves open many interesting problems.

- First, FeynmanDD is characterized by linear rank-width, which uses a special caterpillar tree in the definition. It is interesting to ask whether one can further improve the algorithm so that its complexity is characterized by the rank-width, rather than the linear rank-width, of a circuit. This could further improve efficiency and relate the simulation of quantum circuits to more graph parameters.

- Second, the use of decision diagrams may be an overkill for the simple task we face and the simple functions f_C we work with in our problem. Is it possible to optimize the algorithm so that we do not need to maintain the full decision data structure in memory? We used this idea to prove that the construction of the FeynmanDD can be done in time exponential in the linear-rank width in Section 3.3 but it may motivate further implementation optimizations.

- Third, given the success of tensor-network methods in many-body physics, it is hopeful that the alternative FeynmanDD method, which sometimes offers advantages, may also find applications in this field. This is an interesting direction to explore.

- Finally, tensor-network methods have a practical implementation advantage, as there have been many optimizations and even GPU accelerations. Can we perform similar optimizations for decision diagram-based methods?

2 Preliminaries

2.1 Sum-of-powers representation

Here, we review the sum-of-powers (SOP) representation of quantum circuits, which is derived from the Feynman path integral formalism [WCYJ25]. Roughly speaking, an SOP is a tensor that has an alternative representation as a sum of powers of the r-th root of unity $\omega_r = e^{2\pi \mathrm{i}/r}$ for some fixed r. That is, it takes the form

$$\frac{1}{R}\sum_{\mathbf{y}} \omega_r^{f(\mathbf{x},\mathbf{y})}, \tag{1}$$

where R is a normalization coefficient, and $f(\mathbf{x},\mathbf{y})$ is a function of Boolean variables $\mathbf{x},\mathbf{y}$ taking integer values modulo r. It can be viewed as a tensor with $n = \ell(\mathbf{x})$ open legs of dimension 2 where $\ell(\mathbf{x})$ is the number of Boolean variables in $\mathbf{x}$.

Many quantities of a quantum circuit can be represented as an SOP if the gate set has certain properties, which we now describe. We use the gate set $\mathcal{T} = \{\mathrm{H}, \mathrm{T}, \mathrm{CZ}\}$ as an example to illustrate the idea. A quantum circuit C specified using $\mathcal{T}$ can be written as $C = U_m \dots U_2 U_1$, where each $U_i \in \mathcal{T}$ acts on one or two qubits of the circuit. Inserting identity operators between the gates, we can write the tensor of C as

$$\langle \mathbf{x}_1 | C | \mathbf{x}_0 \rangle = \sum_{\mathbf{y}_1, \dots, \mathbf{y}_{m-1}} \langle \mathbf{x}_1 | U_m | \mathbf{y}_{m-1} \rangle \langle \mathbf{y}_{m-1} | U_{m-1} \cdots U_2 | \mathbf{y}_1 \rangle \langle \mathbf{y}_1 | U_1 | \mathbf{x}_0 \rangle.$$

For simplicity, we identify $\mathbf{x}_0$ with $\mathbf{y}_0$ and $\mathbf{x}_1$ with $\mathbf{y}_m$. The above equation can be written as

$$\langle \mathbf{x}_1 | C | \mathbf{x}_0 \rangle = \sum_{\mathbf{y}_1, \dots, \mathbf{y}_{m-1}} \prod_{j=1}^{m} \langle \mathbf{y}_j | U_j | \mathbf{y}_{j-1} \rangle. \tag{2}$$

This summation is essentially the Feynman path integral formalism. If U_j is a H gate acting on the s-th qubit, we have

$$\langle \mathbf{y}_j | U_j | \mathbf{y}_{j-1} \rangle = \frac{1}{\sqrt{2}} (-1)^{\mathbf{y}_{j,s} \mathbf{y}_{j-1,s}} = \frac{1}{\sqrt{2}} \omega_8^{4 \mathbf{y}_{j,s} \mathbf{y}_{j-1,s}},$$

which is already in the required power form. If U_j is a CZ gate acting on the s-th and t-th qubits, we have

$$\langle \mathbf{y}_j | U_j | \mathbf{y}_{j-1} \rangle = {\omega_8}^{4 \mathbf{y}_{j,s} \mathbf{y}_{j,t}} \delta_{\mathbf{y}_{j,s}, \mathbf{y}_{j-1,s}} \delta_{\mathbf{y}_{j,t}, \mathbf{y}_{j-1,t}},$$

which is a power form if we identify the variable $\mathbf{y}_{j,s}$ with $\mathbf{y}_{j-1,s}$, and $\mathbf{y}_{j,t}$ with $\mathbf{y}_{j-1,t}$. The T gate is also a diagonal gate and can be treated similarly. When U_j is a T gate on the s-th qubit, we have

$$\langle \mathbf{y}_j | U_j | \mathbf{y}_{j-1} \rangle = {\omega_8}^{\mathbf{y}_{j,s}} \delta_{\mathbf{y}_{j,s}, \mathbf{y}_{j-1,s}}.$$

It is also a power of ω_8 if we identify $\mathbf{y}_{j,s}$ with $\mathbf{y}_{j-1,s}$. This explains how circuits in gate set $\mathcal{T}$ can be written as an SOP tensor of modulus 8. The polynomial f in this case is a degree-2 polynomial over $\mathbb{Z}_8$ and the normalization factor $R = \sqrt{2^h}$, where h is the number of Hadamard gates in the circuit. As an example, consider the circuit in Fig. 1 (a). The polynomial f_C in its SOP representation is $4x_1x_1' + 4x_1'x_3 + 4x_2x_5 + 4x_3x_5 + 4x_3x_4 + 4x_5x_6 + x_1' + 2x_5$.

The SOP representation is powerful and flexible. SOP representations exist for other gate sets such as $\mathcal{Z} = \{\mathrm{H}, \mathrm{CCZ}\}$ with modulus 2 and the Google supremacy gate set $\mathcal{G}$ [AAB+19] with modulus 24. Apart from computing the amplitude of the quantum circuit, other computational tasks such as estimating the acceptance probability or sampling the output distribution can also be reduced to computing the SOP form. Usually, these quantities can also be expressed in the SOP form $\frac{1}{\sqrt{R}} \sum_{\mathbf{x}} \omega^{f(\mathbf{x})}$, but with a different polynomial f and a different normalization factor R. We refer to a more in-depth discussion on representing circuit-related quantities using SOP in [WCYJ25]. We call the variables that are not summed over in the final representation external variables, and the rest internal variables.

In this work, we mainly focus on the gate set $\mathcal{T} = \{\mathrm{H}, \mathrm{T}, \mathrm{CZ}\}$ as it is the most commonly used two-qubit discrete gate set. As the above discussion indicates, every

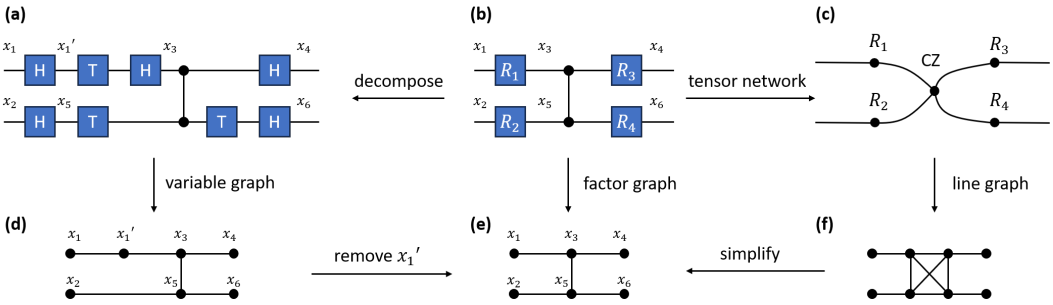

Figure 1: Quantum circuits and graphs. (a) A quantum circuit C' constructed from the gate set $\mathcal{T}$. (b) A quantum circuit C constructed from the gate set $\mathcal{R}$, which has the same unitary as the circuit C'. Here, we let $R_1 = \mathrm{HTH}$, $R_2 = \mathrm{TH}$, $R_3 = \mathrm{H}$, and $R_4 = \mathrm{HT}$. (c) The tensor network of the circuit C. (d) The variable graph (and the factor graph) of the circuit C'. (e) The factor graph of the circuit C. (f) The line graph of the tensor network in (c).

H gate relates two different variables of the qubit before and after the application of the gate and can be seen as a time-like correlation gate. The CZ gate uses two input variables identified with two output variables and can be seen as a space-like correlation gate. These two gates have the same degree-2 form in the polynomial function in the exponent. The T gates are represented as linear terms in the exponent.

In this work, we also consider another gate set $\mathcal{R}$ consisting of all single-qubit rotations and the CZ gate. For continuous rotation gates, one can first apply the Solovay-Kitaev algorithm to decompose the continuous rotation into a sequence of discrete gates from the finite universal gate set $\mathcal{T}$, and then express the tensor in the SOP form. This incurs only a logarithmic overhead in the number of gates [NC00].

2.2 Decision diagrams and Feynman decision diagrams

A binary decision diagram (BDD) is a rooted, directed acyclic graph (DAG) used to represent Boolean functions. It consists of decision nodes and terminal nodes. Each decision node is labeled by a variable and has two outgoing edges: one for the variable being 0 and the other for the variable being 1. The terminal nodes are labeled either 0 or 1. A path from the root to a terminal node corresponds to evaluating the Boolean function for a given variable assignment. In the literature, a special type of BDD called a reduced ordered BDD is the most important, and it is customary to refer to these simply as BDDs. Here, *ordered* means that the variable ordering along any path from the root to a terminal node is consistent, and *reduced* means that there are no isomorphic subgraphs and no node has identical children. The most important feature of a reduced ordered BDD is that it provides a canonical representation of a Boolean function. In our case, we will need a variant of BDDs where the number of terminal nodes is r, not necessarily 2. These are called multi-terminal BDDs (MTBDDs) [BFG+93].

As explicitly listed in [Knu09], BDDs possess many desirable virtues and properties. We summarize some of these that will be used in this work; readers are referred to [Knu09, Weg00] for more details. Note that the following properties also hold for MTBDDs with little or no modification. For this reason, we may simply refer

to MTBDDs as BDDs in the rest of this work. First, and most important for our work, is their counting ability. Specifically, given a function $f : \{0,1\}^n \to \mathbb{Z}_r$ whose BDD representation has size $\mathrm{B}(f)$, counting the number of solutions to $f(\mathbf{x}) = j$ for $j = 0, 1, \ldots, r-1$ can be done in time $\mathcal{O}(n\mathrm{B}(f))$ using BDDs [Weg00, Knu09]. This serves as the computational engine of the FeynmanDD method. Second, BDDs can be composed without an explosion in size. For example, if two functions f and g have bounded BDD sizes $\mathrm{B}(f)$ and $\mathrm{B}(g)$ respectively, then any combination of them, such as $f \wedge g$ or $f + g \pmod 2$, has size at most $\mathcal{O}(\mathrm{B}(f)\mathrm{B}(g))$. Third, there is a simple way to bound the number of nodes at each level of a BDD. Each BDD node can be associated with a subfunction of the original function. For a node at the i-th level, the path from the root to this node activates a partial assignment of the first $i-1$ variables. In the notation of [WCYJ25], this can be written as $f[a_1/x_1, \ldots, a_{i-1}/x_{i-1}]$, corresponding to setting $x_j = a_j$ for $j = 1, \ldots, i-1$. The resulting subfunction depends only on the remaining variables $x_i, \ldots, x_n$, and the node represents this subfunction. Since the BDD is reduced, the number of nodes at the i-th level equals the number of distinct subfunctions $f[a_1/x_1, \ldots, a_{i-1}/x_{i-1}]$ for all assignments $a_1, \ldots, a_{i-1} \in \{0,1\}$ that essentially depend on $x_i, \ldots, x_n$.

As shown in [WCYJ25], many quantities of quantum circuits can be reduced to the evaluation of a SOP with no external variables. The evaluation of the SOP can be written as

$$\frac{1}{R}\sum_{\mathbf{x}} \omega_r^{f(\mathbf{x})} = \sum_{j=0}^{r-1} \frac{N_j}{R}\omega_r^j,$$

where $N_j = |\{x \mid f_C(x) = j\}|$. The FeynmanDD method consists of three steps: (a) identifying the polynomial $f : \{0,1\}^n \to \mathbb{Z}_r$ in the SOP; (b) constructing the r-terminal decision diagram for f; and (c) utilizing the counting ability of BDDs to compute N_j for each j and adding r complex numbers to get an exact numerical value of the SOP. The resulting multi-terminal decision diagram is called a FeynmanDD to highlight its origin from the Feynman path integral formalism [WCYJ25]. Numerical studies in [WCYJ25] and Section 4.3 of this paper reveal that it is a competitive method of quantum circuit simulation for various families of circuits.

It is well-known that the variable ordering will significantly impact the size of BDD and that finding an optimal variable ordering is NP-hard [Weg00]. In [WCYJ25], several heuristic methods for choosing the variable order were discussed, but no general theory or analysis is provided for minimizing the BDD size. This is one of the key questions that this paper aims to address.

2.3 Rank-width and linear rank-width

Here, we introduce the concept of rank-width and its linearized variant, which are width parameters defined by Oum and Seymour [OS06]. Previously, rank-width and linear rank-width have found applications in characterizing the entanglement structure and the complexity of graph states [HEB04, VdNDVB07, DW18, AMSDS20]. While related to our approach, these methods are not capable of handling non-Clifford gates.

Let $G = (V, E)$ be a graph with adjacency matrix $\mathbf{A} \in \mathbb{F}_2^{n\times n}$, and let $X \subset V$ be a subset of its vertices. Let $\mathbf{A}[X, V\backslash X] = (a_{ij} : i \in X, j \in V\backslash X)$ be a submatrix of $\mathbf{A}$ of size $|X| \times |V\backslash X|$, formed by the rows corresponding to the vertices in X and the

columns corresponding to the vertices not in X. Define a rank function ρ_G on vertex subsets of G as $\rho_G(X) := \text{rank}(\mathbf{A}[X, V \setminus X])$. We call ρ_G the cut-rank function of G.

A rank decomposition of a graph G is a pair (T, μ) consisting of a subcubic tree T and a bijection $\mu : \text{leaf}(T) \to V(G)$. Here, a subcubic tree is a tree with maximum degree at most 3. For every edge e of T, removing e from T splits T into two connected components, which in turn induces a bipartition of the leaves of T. This bipartition of leaves corresponds to a bipartition of the vertices of G, denoted by X_e and $V(G) \setminus X_e$. The width of an edge e in T is defined as $\rho_G(X_e)$, and the width of a rank decomposition (T, μ) is defined as $\max_{e \in E(T)} \rho_G(X_e)$. The rank-width of a graph G, denoted by $\text{rw}(G)$, is the minimum width over all rank decompositions of G, i.e.,

$$\text{rw}(G) := \min_{(T,\mu)} \max_{e \in E(T)} \rho_G(X_e).$$

For linear rank-width, we restrict the subcubic tree T to be a caterpillar, which is a tree that becomes a path when all its leaves are removed. However, for our purposes, we use the following equivalent definition. Let $\pi_1, \ldots, \pi_n$ be a linear ordering of $V(G)$, which is a permutation of the vertices. The width of this linear ordering is defined as $\max_{1 \le i \le n-1} \rho_G(\{\pi_1, \ldots, \pi_i\})$. The linear rank-width of G, denoted by $\text{lrw}(G)$, is the minimum width over all linear orderings of $V(G)$, i.e.,

$$\text{lrw}(G) := \min \left\{ \max_{1 \le i \le n-1} \rho_G(\{\pi_1, \ldots, \pi_i\}) \mid \pi_1, \ldots, \pi_n \text{ is a linear ordering of } V(G) \right\}.$$

Computing the linear rank-width of a graph is NP-hard in general [Oum17]. Nevertheless, analogous to the treewidth, the problem is fixed-parameter tractable (FPT). Recall that a problem is fixed-parameter tractable if it can be solved by an algorithm with runtime $\mathcal{O}(f(k) \cdot n^c)$, where n is the input size, k is a parameter of the input, f is a computable function, and c is a constant independent of k. The function f is generally exponential or grows even faster. Previously, it was shown that the linear rank-width of a graph is linearly related to the path-width of a corresponding matroid [Oum05, HO08, Hli18]. In addition, Hliněný constructed an FPT algorithm with runtime $\mathcal{O}(f(k) \cdot n^3)$ to decide, for a fixed k, whether the path-width of a matroid is at most k, and if so, output a corresponding path decomposition [Hli18]. Altogether, we have the following proposition.

Proposition 2.1. *Let k be a positive parameter. Given an n-vertex graph G, we can output a linear ordering of G of width at most k or confirm that the linear rank-width of G is larger than k in time $\mathcal{O}(f(k) \cdot n^3)$, where f is a computable function.*

For the tensor network simulation algorithm, it is well-known that its time and space complexity is exponential in the treewidth of the underlying graph of the tensor network [MS08]. The treewidth of a graph G, denoted as $\text{tw}(G)$, is a parameter measuring how far G is from being a tree. The linearized version of treewidth is called path-width, denoted as $\text{pw}(G)$. There are some known relations between these graph parameters. First, the linear rank-width is upper bounded by the path-width [AK15]: $\text{lrw}(G) \le \text{pw}(G)$. Second, the path-width and the treewidth are in some sense equivalent [KS93]: $\text{pw}(G) \le \mathcal{O}(\log n \cdot \text{tw}(G))$. Altogether, we have the following relation:

$$\text{lrw}(G) \le \mathcal{O}(\log n \cdot \text{tw}(G)). \tag{3}$$

Taking the exponential on both sides, we find that FeynmanDD has at most a polynomial slowdown compared to tensor network methods. However, in some cases, the linear rank-width can be much smaller than the treewidth. For example, for the complete graph K_n, its treewidth is $n-1$ while its linear rank-width is only 1. That is, for some circuits, the FeynmanDD method can be exponentially more efficient than tensor network methods. Based on this idea we give a family of random circuits that has bounded linear rank-width but high treewidth in Section 4.2.

3 Time complexity of FeynmanDD

In this section, we analyze the time complexity of the FeynmanDD method for simulating quantum circuits, establishing a connection to the linear rank-width of the underlying circuit graph. The overall complexity of FeynmanDD is determined by two main steps: constructing the BDD for the SOP polynomial f and then using this BDD to count the number of solutions.

Our analysis proceeds in two parts. First, we address the counting step, whose time complexity is proportional to the size of the BDD. We derive a bound on the BDD size, initially for discrete gate sets

$$\mathcal{T} = \{\mathrm{H}, \mathrm{T}, \mathrm{CZ}\},$$
$$\mathcal{G} = \{\sqrt{\mathrm{X}}, \sqrt{\mathrm{Y}}, \sqrt{\mathrm{W}}, \mathrm{iSWAP}\},$$

and then extend the analysis to the continuous gate set $\mathcal{R}$. While we focus on computing the zero-to-zero amplitude $\langle 0^n|C|0^n\rangle$, the proof generalizes to other simulation tasks like computing acceptance probabilities or sampling.

Second, we analyze the complexity of constructing the BDD in Section 3.3. Although this step can be a bottleneck, with a time complexity potentially much larger than the final BDD size, we show that for the gate sets under consideration, the construction time is also bounded by the BDD size.

3.1 Gate set $\mathcal{T}$ and $\mathcal{G}$

We first consider the gate set $\mathcal{T}$, where the SOP polynomial is a degree-2 polynomial over $\mathbb{Z}_8$ of the form $f(\mathbf{x}) = 4\sum_{i<j} a_{ij}x_ix_j + \sum_i b_ix_i$, with $a_{ij} \in \{0,1\}$ and $b_j \in \mathbb{Z}_8$. This polynomial can be written more compactly as $f(\mathbf{x}) = 4\mathbf{x}^T\mathbf{A}_{\mathsf{up}}\mathbf{x} + \mathbf{b}^T\mathbf{x}$, where $\mathbf{b} = (b_i)$ is a vector, and the matrix $\mathbf{A}_{\mathsf{up}}$ is defined as $(\mathbf{A}_{\mathsf{up}})_{ij} = a_{ij}$ for $i < j$ and 0 otherwise. We can then define a symmetric matrix $\mathbf{A} = \mathbf{A}_{\mathsf{up}} + \mathbf{A}_{\mathsf{up}}^T$ with a zero diagonal, which serves as the adjacency matrix of a graph. We call this the *variable graph* of the quantum circuit C, denoted by G_C. An example of a variable graph is shown in Fig. 1 (d). Note that the variable graph G_C is associated with a specific SOP representation and thus with a particular gate set. Moreover, the representation $\mathbf{A}$ depends on the ordering of the variables $x_1, \ldots, x_n$. Hence, when writing a concrete matrix $\mathbf{A}$, an implicit variable ordering is assumed. However, the variable graph G_C itself is independent of the variable ordering; different orderings correspond to isomorphic variable graphs.

After constructing the MTBDD for the SOP polynomial f, what remains for quantum circuit simulation is to count the number of solutions to $f(\mathbf{x}) = j$ for

$0 \leq j \leq r-1$. Recall that the time complexity of counting using BDDs is $\mathcal{O}(n\mathrm{B}(f))$, where $\mathrm{B}(f)$ is the size of the BDD for f. Below, our focus is on deriving a bound for the BDD size of a degree-2 polynomial, by counting the number of nodes in each level of the BDD. Specifically, we prove that $\mathrm{B}(f)$ is exponential in the linear rank-width of the variable graph associated with the quantum circuit C.

Theorem 3.1. *Suppose we are given a quantum circuit C constructed from the gate set $\mathcal{T}$ with an SOP polynomial $f(\mathbf{x}) = 4\mathbf{x}^T\mathbf{A}_{\mathsf{up}}\mathbf{x} + \mathbf{b}^T\mathbf{x}$ of n variables. The number of nodes in the i-th level of the BDD is at most 2^{r_i+3}, where $r_i := \mathrm{rank}(\mathbf{A}[[i-1],[n]\backslash[i-1]])$. In particular, this implies that the BDD size under the variable ordering $x_1,\ldots,x_n$ is at most $n \cdot 2^{w+3}$, where w is the width of the corresponding linear ordering of the variable graph G_C.*

Proof. For the i-th level, partition f into two parts $f = f_1 + f_2$, where f_2 is a function of $x_i,\ldots,x_n$ only and f_1 consists of the remaining terms. Explicitly,

$$f_1(\mathbf{x}) = 4\sum_{j=1}^{i-1} x_j(a_{j,j+1}x_{j+1} + \cdots + a_{j,n}x_n) + \sum_{j=1}^{i-1} b_j x_j, \tag{4}$$

$$f_2(\mathbf{x}) = 4\sum_{j=i}^{n-1} x_j(a_{j,j+1}x_{j+1} + \cdots + a_{j,n}x_n) + \sum_{j=i}^{n} b_j x_j. \tag{5}$$

In the substitution $f[a_1/x_1,\ldots,a_{i-1}/x_{i-1}]$, since f_2 is independent of $x_1,\ldots,x_{i-1}$, the number of different functions generated is determined by the linear function $f_1[a_1/x_1,\ldots,a_{i-1}/x_{i-1}]$. The terms in f_1 that depend only on $\{x_1,\ldots,x_{i-1}\}$ will jointly contribute an additive constant from $\mathbb{Z}_8$ in the substitution. Up to this additive constant, we have

$$f_1[a_1/x_1,\ldots,a_{i-1}/x_{i-1}] = 4\mathbf{a}^T\mathbf{A}_{[i-1]}\mathbf{x}_{[n]\backslash[i-1]},$$

where $\mathbf{a} = (a_1,\ldots,a_{i-1})^T$, $\mathbf{A}_{[i-1]} := \mathbf{A}[[i-1],[n]\backslash[i-1]]$, and $\mathbf{x}_{[n]\backslash[i-1]} = (x_i,\ldots,x_n)^T$. Therefore, when taking the additive constant into account, the number of different functions is equal to eight times the number of vectors in the row space of $\mathbf{A}_{[i-1]}$, which is 2^{r_i+3}.

The variable ordering $x_1,\ldots,x_n$ induces a linear ordering of the variable graph G_C. By definition, the width of this linear ordering is $w = \max_{1\leq i\leq n-1} r_i$. Thus, the BDD size is at most $n \cdot 2^{w+3}$. □

From this theorem, it follows that given a quantum circuit C and its variable graph G_C, if a linear ordering of G_C with small width can be found, then the corresponding FeynmanDD will also be of small size. The optimal variable ordering with the smallest BDD size corresponds to the linear ordering of G_C with width equal to the linear rank-width $\mathrm{lrw}(G_C)$.

Although finding an optimal variable ordering is NP-hard, the FPT algorithm from Proposition 2.1 yields an optimal ordering in time $\mathcal{O}(f(\mathrm{lrw}(G_C)) \cdot n^3)$, where f is some computable function.

Theorem 3.2. *Given a quantum circuit C constructed from the gate set $\mathcal{T}$, let $f_C : \{0,1\}^n \to \mathbb{Z}_8$ be its sum-of-powers representation. Let G_C be the variable graph*

Gate	Input	Output	Representation	Factor
$\sqrt{\mathrm{X}}$	x_0	x_1	$\omega_{24}^{18x_0+18x_1+12x_0x_1}$	$1/\sqrt{2}$
$\sqrt{\mathrm{Y}}$	x_0	x_1	$\omega_{24}^{12x_0+12x_0x_1}$	$1/\sqrt{2}$
$\sqrt{\mathrm{W}}$	x_0	x_1	$\omega_{24}^{15x_0+21x_1+12x_0x_1}$	$1/\sqrt{2}$
iSWAP	x_0, x_1	x_1, x_0	$\omega_{24}^{18x_0+18x_1+12x_0x_1}$	1

Table 1: Sum-of-powers representation for the Google supremacy gate set.

of C and let $w = \mathrm{lrw}(G_C)$ be its linear rank-width. Then, in time $\mathcal{O}(f(w) \cdot n^3)$, where f is a computable function, we can output an optimal variable ordering of f_C with BDD size upper bounded by $n \cdot 2^{w+3}$.

Proof. Using the FPT algorithm in Proposition 2.1, we can find a linear ordering of G_C with width at most k or confirm that the linear rank-width of G_C is larger than k in time $\mathcal{O}(f(k) \cdot n^3)$. Iterate this FPT algorithm for $k = 1, 2$ and so on until a linear ordering is output, which happens when $k = w$. This will find an optimal linear ordering of width w of G_C in time $\mathcal{O}(f(w) \cdot n^3)$. Such an optimal linear ordering of G_C gives a variable ordering of f_C with BDD size at most $n \cdot 2^{w+3}$ by Theorem 3.1. □

The above results also apply to the gate set $\mathcal{G}$. The matrix elements of the gates in $\mathcal{G}$ can be expressed as powers of ω_{24} (see Table 1). All gates in $\mathcal{G}$ are non-diagonal. From this table, it can be seen that the SOP polynomial for a quantum circuit constructed from $\mathcal{G}$ takes the form $f(\mathbf{x}) = 12\mathbf{x}^T\mathbf{A}_{\mathsf{up}}\mathbf{x} + \mathbf{b}^T\mathbf{x}$, where $\mathbf{A} \in \mathbb{F}_2^{n\times n}$ and $\mathbf{b} \in \mathbb{Z}_{24}^n$. The variable graph of a quantum circuit constructed from $\mathcal{G}$ can be similarly defined with $\mathbf{A}$ as its adjacency matrix. Thus, following the same proof as in Theorem 3.1, we find that the BDD size under a fixed variable ordering is at most $24n2^w$, where w is the linear rank-width of the corresponding linear ordering of the variable graph.

3.2 Gate set $\mathcal{R}$

Here, we aim to demonstrate that FeynmanDD can also handle the continuous gate set $\mathcal{R}$ with acceptable overhead. For a quantum circuit C constructed from $\mathcal{R}$, we first introduce the *factor graph*, which is a complex undirected graphical model [BISN18] and a variant of the tensor network representation proposed in [MS08]. We then show that a simplified version of the factor graph, commonly used in the literature, is equivalent to the variable graph defined above. Finally, we prove that compiling C into a quantum circuit C' with gate set $\mathcal{T}$ increases the linear rank-width of the variable graph by at most 2.

3.2.1 Factor graph, tensor network and variable graph

We first recall the undirected graphical model in [BISN18], where the underlying graphical model is the factor graph of a quantum circuit. The factor graph of C arises from the Feynman path integral formalism. In Eq. (2), each gate will contribute a factor defined by $\langle \mathbf{y}_j | U_j | \mathbf{y}_{j-1} \rangle$. For a single-qubit gate U, the factor is given by $\phi_U(x, y) = \langle x|U|y\rangle$, where we use $x, y \in \{0, 1\}$ to denote the input and output

variables of U. For a two-qubit gate U, the factor is given by $\phi_U(x_1, x_2, y_1, y_2) = \langle y_1, y_2|U|x_1, x_2\rangle$, where x_1, x_2 and y_1, y_2 are the input and output variables of U, respectively. Then, the factor graph of C is defined such that each binary variable corresponds to a vertex in the factor graph and each factor corresponds to a clique. The summation over the binary variables in Eq. (2) corresponds to eliminating the vertices and connecting the neighbors of the eliminated vertices in the factor graph.

Next, we show that the factor graph and tensor network are, in some sense, equivalent. The tensor network representation can also be derived from the Feynman path integral formalism. Each factor can be viewed as a tensor. A single-qubit gate is represented by a tensor with two legs, and a two-qubit gate is represented by a tensor with four legs. Summation over the binary variables corresponds to connecting the legs of the tensors, known as tensor contraction. In Fig. 1, subfigure (c) represents the tensor network of the quantum circuit shown in (b).

Given a tensor network G, its *line graph* G^* is constructed with a precise vertex-edge correspondence. The vertex set of G^* is in one-to-one correspondence with the edge set of G. Two vertices e_1 and e_2 in G^* are connected if there exists a vertex v in G to which both edges are incident. The line graph for the tensor network in Fig. 1 (c) is shown in (f).

From this definition, one can observe that the factor graph defined above is exactly the line graph of the tensor network representation of C. Moreover, the summation in the Feynman path integral corresponds to contracting the tensor network or eliminating the vertices in the factor graph, whose complexity is exponential in the treewidth of the underlying graph [MS08, BISN18]. Since the treewidth of the line graph is linearly related to the treewidth of the original graph, the undirected graphical model method in [BISN18] is in some sense equivalent to the tensor network method.

In [BISN18] however, a simplified version of the factor graph is actually used, where diagonal two-qubit gates such as CZ are represented by factors with two variables, and diagonal single-qubit gates like T are represented by factors with one variable [BISN18]. In this simplification, the factor for CZ is identical to that of the Hadamard gate, up to a multiplicative factor. In the factor graph, a diagonal single-qubit gate corresponds to a single node, and a diagonal two-qubit gate corresponds to an edge (see Fig. 1 (e)). One can observe that if the quantum circuit is constructed from $\mathcal{T}$, the factor graph is exactly the same as the variable graph. For this reason, we also denote the factor graph of a quantum circuit C as G_C and simply refer to it as the variable graph of C. However, as we will see later, despite this connection, the FeynmanDD method is fundamentally different from the tensor network method or the undirected graphical model method.

3.2.2 Compiling continuous gates into discrete gates

Below, we prove that for a quantum circuit with gate set $\mathcal{R}$, compiling it into a sequence of gates from $\mathcal{T}$ only increases the linear rank-width of the variable graph by at most 2.

Theorem 3.3. *Given a quantum circuit C with gate set $\mathcal{R}$ and its variable graph G_C. Apply the Solovay-Kitaev algorithm to decompose it into a quantum circuit C' with gate set $\mathcal{T}$. Let $G_{C'}$ be the variable graph of C'. Then,* $\mathrm{lrw}(G_{C'}) \leq \mathrm{lrw}(G_C) + 2$.

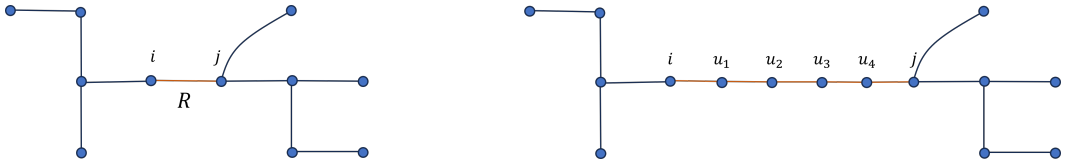

Figure 2: Let C be a quantum circuit constructed from gate set $\mathcal{R}$, with variable graph G_C (left panel). Let C' be a quantum circuit compiled from C using the Solovay-Kitaev algorithm, with variable graph $G_{C'}$ (right panel).

Proof. Suppose $\pi_1, \ldots, \pi_n$ is a linear ordering of G_C with width $w = \mathrm{lrw}(G_C)$. Consider a single-qubit rotation gate R in C with input and output variables x_i and x_j, respectively. Under the variable ordering π, x_i is in the π_i-th position and x_j is in the π_j-th position. Without loss of generality, assume $\pi_i < \pi_j$. After applying the Solovay-Kitaev algorithm, R is decomposed into a sequence of gates from $\mathcal{T}$, $R = \mathrm{HTHTTH} \cdots \mathrm{TH}$. Suppose this introduces $k-1$ new variables $u_1, \ldots, u_{k-1}$, where k is the number of Hadamard gates. See Fig. 2 for an illustration. Let the adjacency matrix of G_C under this variable ordering be $\mathbf{A}$,

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_1 & \mathbf{A}_2 \\ \mathbf{A}_2^T & \mathbf{A}_3 \end{pmatrix} , \tag{6}$$

where $\mathbf{A}_1$ is a $\pi_i \times \pi_i$ matrix and the π_i-th row and column correspond to the variable x_i. Consider a linear ordering of $G_{C'}$ as

$$\pi_1, \ldots, \pi_i, u_1, \ldots, u_{k-1}, \pi_{i+1}, \ldots, \pi_j, \ldots, \pi_n, \tag{7}$$

where the new variables are inserted after π_i. Let $\mathbf{A}'$ be the adjacency matrix of $G_{C'}$ under this variable ordering. Then, we have

$$\mathbf{A}' = \begin{pmatrix} \mathbf{A}_1 & \mathbf{B}_1 & \mathbf{A'}_2 \\ \mathbf{B}_1^T & \mathbf{B}_2 & \mathbf{B}_3 \\ \mathbf{A'}_2^T & \mathbf{B}_3^T & \mathbf{A}_3 \end{pmatrix} , \tag{8}$$

where $\mathbf{B}_1$ has a single 1 in the bottom-left entry and zeros elsewhere, $\mathbf{B}_3$ has a single 1 in the last row in the position corresponding to π_j and zeros elsewhere, $\mathbf{A'}_2$ is modified from $\mathbf{A}_2$ by flipping the entry corresponding to π_j in the last row to be zero, and $\mathbf{B}_2$ is a tridiagonal matrix with ones on the superdiagonal and subdiagonal and zeros elsewhere. For the example in the figure, we have

$$\mathbf{B}_2 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} . \tag{9}$$

To determine the linear rank-width of $G_{C'}$ under this ordering, we examine the rank of the submatrices $\mathbf{A}'[[t], [n+k-1] \backslash [t]]$ for $t = 1, \ldots, n+k-2$:

- For $t < \pi_i$, the rank equals $\mathrm{rank}(\mathbf{A}[[t], [n] \backslash [t]]) \leq w$.

- For $\pi_i \leq t < \pi_i + k - 1$, the rank is at most $\text{rank}(\mathbf{A}_2) + 2 \leq w + 2$, since the columns of the new submatrix contribute at most two additional linearly independent vectors.
- For $\pi_i + k - 1 \leq t \leq \pi_j + k - 2$, the rank is at most $\text{rank}(\mathbf{A}[[t-k+1], [n] \backslash [t-k+1]]) + 1 \leq w + 1$, since the columns of the new submatrix contribute at most one additional linearly independent vector.
- For $t > \pi_j + k - 2$, the rank equals $\text{rank}(\mathbf{A}[[t-k+1], [n] \backslash [t-k+1]]) \leq w$.

Thus, the linear rank-width of $G_{C'}$ is at most $w + 2$.

Now, consider the general case where all single-qubit rotation gates are decomposed. One can imagine that $\mathbf{A}'$ will have a similar block structure as in Eq. (8), but with more $\mathbf{B}_1$, $\mathbf{B}_2$ and $\mathbf{B}_3$ blocks inserted. By the same line of reasoning, the rank of the submatrices $\mathbf{A}'[[t], [n+k-1] \backslash [t]]$ will always increase by at most 2. Thus, we conclude that $\text{lrw}(G_{C'}) \leq \text{lrw}(G_C) + 2$. □

3.3 Time complexity of building decision diagrams

In Theorems 3.1 and 3.2, we established an upper bound of $n \cdot 2^{w+3}$ on the size of the decision diagram for a quantum circuit and showed that an optimal variable ordering can be obtained in $\mathcal{O}(f(w) \cdot n^3)$ time. In this section, we analyze the time complexity of building a decision diagram starting from a SOP representation of the circuit. In [WCYJ25], the underlying decision diagram is built using the apply functionality of the decision diagram data structure, and a heuristic method called the *binary synthesis method* is employed to speed up the construction. Although the binary synthesis method supports more gate sets and is practically efficient, as verified by numerical simulations in [WCYJ25], its complexity is difficult to analyze. Here, we exploit the special low-degree polynomial structure of the SOP and provide an alternative method for building the decision diagram in a level-by-level fashion. This direct method allows us to show that the construction of the decision diagram is not the bottleneck of the FeynmanDD simulation method.

Theorem 3.4. *Given a quantum circuit C constructed from the gate set $\mathcal{T}$, its corresponding sum-of-powers representation f_C and a variable ordering such that the linear rank-width of its variable graph is w, the corresponding decision diagram can be constructed in time $\mathcal{O}(n^2 \cdot 2^w)$.*

Proof. We employ a constructive proof derived by analyzing the time complexity of the algorithm used in the FeynmanDD construction process. The overall procedure of the algorithm is outlined in Algorithm 1.

The construction proceeds top-down, building the decision diagram level by level according to the variable ordering $x_1, x_2, \ldots, x_n$. At each step, a non-leaf node is visited, and its connections to its child nodes are computed directly using the special form of the Boolean function f_C. Upon completing the traversal, the desired reduced ordered BDD is obtained. The root node at level 1 represents the entire function f_C. A node at level i for $2 \leq i \leq n+1$ represents a subfunction $f_C[a_1/x_1, \ldots, a_{i-1}/x_{i-1}]$, obtained by assigning values to the first $i-1$ variables. The nodes in the $(n+1)$-th level are the leaf nodes of the BDD, each representing a constant γ modulo r. When the circuit C uses the gate set $\mathcal{T}$, $\gamma \in \mathbb{Z}_8$.

The key to ensuring that the number of nodes in the decision diagram matches the theoretical bound is to avoid generating redundant nodes. To achieve this, we use a unique table [Bry86] to store nodes: when considering a child of a parent node, we first check whether the subfunction represented by the child is already represented by an existing node in the decision diagram. If not, a new node is created; otherwise, the existing node is retrieved and used as the child. The parent's corresponding child pointer is then set accordingly.

To complete the above strategy, an important technical detail is how we represent the functions for each node, which we present below. Recall the discussion from the proof of Theorem 3.1. Let $\mathbf{A}$ be the adjacency matrix of the variable graph of C under the given variable order. For a node at level i, the function f represented by each such node can be decomposed as $f = f_1 + f_2$, where f_2 is a function of $x_i, \ldots, x_n$ only and f_1 consists of the remaining terms as in Eqs. (4) and (5).

For the root node level, $f_1 = 0$ and $f_2 = f_C$. More generally, after considering substitutions $f[a_1/x_1, \ldots, a_{i-1}/x_{i-1}]$, f can essentially be rewritten as

$$4\mathbf{a}^T \mathbf{A}_{[i-1]} \mathbf{x}_{[n]\setminus[i-1]} + f_2(\mathbf{x}) + \gamma_i,$$

where $\mathbf{a} = (a_1, \ldots, a_{i-1})^T$, $\mathbf{A}_{[i-1]} := \mathbf{A}[[i-1], [n]\setminus[i-1]]$, $\mathbf{x}_{[n]\setminus[i-1]} = (x_i, \ldots, x_n)^T$, and the constant γ_i is derived from the term $\sum_{j=1}^{i-1} b_j a_j$ in $f_1(\mathbf{x})$. If the original function f_C contains a global constant, it can also be incorporated into γ_i. As analyzed in Theorem 3.1, there are two places where differences may arise in the functions represented by different nodes at level i: (1) The constant γ_i; (2) The linear terms involving x_i through x_n that are generated within f_1. It is worth emphasizing that f_2 also contains linear terms, but these terms are identical for all nodes at level i. The coefficients of the terms mentioned in (2) can be linearly represented by the basis of the row space of $\mathbf{A}_{[i-1]}$:

$$\mathbf{a}^T \mathbf{A}_{[i-1]} = \sum_{j=1}^{r_i} c_j \mathbf{d}_j, \tag{10}$$

where $r_i := \mathrm{rank}(\mathbf{A}_{[i-1]})$, $\{\mathbf{d}_j\}_{[i-1]}$ is a basis for the row space of $\mathbf{A}_{[i-1]}$, and $c_j \in \{0, 1\}$. For simplicity in decomposition, the basis $\{\mathbf{d}_j\}_{[i-1]}$ is chosen to be the set of nonzero rows in the row-reduced echelon form (RREF) of $\mathbf{A}_{[i-1]}$. It can be noted that we omit the leading factor of 4 in Eq. (10), which allows us to represent c_j using $\{0, 1\}$ instead of $\{0, 4\}$. This simplification relies on the condition that the quadratic coefficients in the gate set must be equal to $r/2$. Clearly, both the gate set $\mathcal{T}$ and the gate set $\mathcal{G}$ satisfy this condition.

Consider the tuple $(\gamma_i, c_1, \ldots, c_{r_i})$, which is uniquely determined by $\mathbf{a}$ and serves to distinguish different functions at level i. Different assignments $\mathbf{a}$ may yield the same tuple; in the context of decision diagrams, this indicates that the corresponding variable assignments lead to the same subfunction $f_C[a_1/x_1, \ldots, a_{i-1}/x_{i-1}]$. We maintain a lookup table for each level to index its nodes. When the construction process is about to add a node at level i, we query the table using the corresponding tuple. If a node with the same tuple already exists, we simply link the parent's zero (or one) child pointer to that node, avoiding redundant creation. Various data structures can be used to implement this functionality, such as a hash table as already

observed in [Bry86], where it is argued that the cost of retrieval of an element from a unique table is $O(1)$ on average. If a worst-case bound is required, one may use an array to pre-allocate all possible nodes indexed by the subfunctions. The number of entries in the table or array for each level is at most 2^{r_i+3}.

Next, we briefly describe how to obtain the ranks and basis vectors for the row spaces of $\mathbf{A}_{[i-1]}$ for $2 \leq i \leq n$. It is well-known that the time complexity for computing the RREF of an $n \times m$ matrix is $\mathcal{O}(nm \min\{n, m\})$. Since we need to compute the RREF for each $\mathbf{A}_{[i-1]}$, we can reuse the results from the previous level. The transition from $\mathbf{A}_{[i-1]}$ to $\mathbf{A}_{[i]}$ involves removing the column corresponding to the variable x_i and adding the row corresponding to the variable x_i. The computation for the initial matrix $\mathbf{A}_{[1]}$ is trivial. Once we have solved for $\mathbf{A}_{[i-1]}$ and obtained the matrix $\mathbf{A}'_{[i-1]}$ consisting of the non-zero rows of its RREF, we can remove the first column of $\mathbf{A}'_{[i-1]}$ and append the new row for x_i. Thus, computing the RREF of $\mathbf{A}_{[i]}$ only requires computing the RREF of an $(n-i) \times (r_i+1)$ matrix, whose time complexity is $\mathcal{O}(nr_i^2)$. Since the rank r_i is bounded by the linear rank-width w, the total time complexity across all levels is $\mathcal{O}(n^2w^2)$.

When decomposing $\mathbf{a}^T\mathbf{A}_{[i-1]}$ using Eq. (10), we define p_j as the pivot position of $\mathbf{d}_j$, i.e., the index where the first 1 occurs. Based on the properties of $\mathbb{Z}_2$ arithmetic and the RREF, the coefficient c_j is given by $c_j = (\mathbf{a}^T\mathbf{A}_{[i-1]})_{p_j}$, meaning each coefficient c_j equals the entry of $\mathbf{a}^T\mathbf{A}_{[i-1]}$ at the pivot position for $\mathbf{d}_j$. Extracting the set $\{c_j\}$ can be completed in $\mathcal{O}(r_i)$ time. In some special cases, the matrix $\mathbf{A}_{[i-1]}$ has rank 0, in which case the tuple contains only γ_i and we do not need to perform the decomposition.

We now describe how to derive the child nodes from the parent node. For clarity, we rewrite the expression $\mathbf{a}^T\mathbf{A}_{[i-1]}\mathbf{x}_{[n]\setminus[i-1]}$ as:

$$g_i(x_i, \ldots, x_n) = \mathbf{a}^T\mathbf{A}_{[i-1]}\mathbf{x}_{[n]\setminus[i-1]} = \sum_{k=i}^{n} \alpha_k x_k.$$

For a node at level i, we maintain $g_i(x_i, \ldots, x_n)$ and γ_i. When generating the child nodes, the terms in f_2 will have an effect. It is easy to see that when $x_i = 0$,

$$g_{i,0}(x_{i+1}, \ldots, x_n) = \sum_{k=i+1}^{n} \alpha_k x_k$$
$$\gamma_{i,0} = \gamma_i$$

and when $x_i = 1$,

$$g_{i,1}(x_{i+1}, \ldots, x_n) = \sum_{k=i+1}^{n} (\alpha_k + a_{i,k}) x_k$$
$$\gamma_{i,1} = \gamma_i + 4\alpha_i + b_i$$

The addition of the coefficients in g is performed modulo 2, while the addition for γ is performed modulo 8.

To directly obtain a *reduced* diagram, we require that, for any non-leaf level i, the function represented by a node in level i must contain at least one term involving the variable x_i. If a node's function does not essentially depend on x_i, this node should

be placed at a lower level, which means that, in Algorithm 1, we must first determine the correct level corresponding to the child node before proceeding with any index lookup or insertion. Taking the one child node of a level i node as an example, we first calculate $g_{i,1}$ and $\gamma_{i,1}$ for it. The condition for the child node to be placed in level $i+1$ is that $(4\alpha_{i+1}+b_{i+1}) \not\equiv 0 \pmod 8$ or that $f_2(\mathbf{x})$ contains a quadratic term involving x_{i+1}. If neither of these conditions is satisfied, the check should continue to the next level. In subsequent checks, we simply ignore the coefficient of x_{i+1} in $g_{i,1}$. This process continues until the leaf node level is reached. For the nodes in the leaf level (i.e., level $n+1$), we directly use the constant γ for indexing.

The two nested for-loops traverse all nodes in the decision diagram. For each node, the computation of child nodes' tuple, tuple lookups, and child node insertions can all be performed in $\mathcal{O}(n)$ time. Thus, the total time complexity for constructing the Decision Diagram is $\mathcal{O}(n^2 \cdot 2^{w+3} + n^2 w^2)$, which asymptotically simplifies to $\mathcal{O}(n^2 \cdot 2^w)$.

□

4 Provable advantage over tensor networks

This section presents concrete quantum circuit constructions where FeynmanDD demonstrates efficient simulation while tensor network contraction complexity remains high. We focus on computing the output amplitude $\langle 0^n|C|0^n\rangle$ for an Instantaneous Quantum Polynomial (IQP) circuit C of the form $C = \mathrm{H}^{\otimes n} D \mathrm{H}^{\otimes n}$, where D is a diagonal circuit [SB09, BJS11, BMS16]. We consider two different gate sets: $\mathcal{Z} = \{\mathrm{H}, \mathrm{Z}, \mathrm{CZ}, \mathrm{CCZ}\}$ and $\mathcal{T}$. For the former, the diagonal gate D is composed of Z, CZ, and CCZ gates; for the latter, D is composed of T and CZ gates. We present two concrete circuit constructions: one based on the linear-network model characterizing the BDD size [Knu09], and the other based on our previous complexity characterization using linear rank-width. The analysis of the first construction does not depend on the linear rank-width characterization at all, while the second is based entirely on this new characterization.

4.1 Linear-network circuits

For IQP circuits with gate set $\mathcal{Z}$, the sum-of-powers representation for $\langle 0^n|C|0^n\rangle$ can be expressed as:

$$\langle 0^n|C|0^n\rangle = \frac{1}{2^n} \sum_{x\in\{0,1\}^n} (-1)^{f(x)}.$$

Here, $f : \{0,1\}^n \to \{0,1\}$ represents a multilinear degree-3 polynomial naturally defined by D, where $\mathrm{Z}, \mathrm{CZ}, \mathrm{CCZ}$ gates correspond to linear, quadratic, and cubic terms, respectively. As previously discussed, decision diagrams can efficiently count solutions to $f(x) \equiv j \pmod r$, thus enabling amplitude computation. The time complexity is $\mathcal{O}(n\mathrm{B}(f))$. Consequently, if $\mathrm{B}(f)$ is polynomial in n, the amplitude can be computed efficiently.

To establish a provable bound for the size of $\mathrm{B}(f)$, we make use of a linear-network construction from [Knu09]. Consider a Boolean function f computed by the linear network depicted in Fig. 3. Each module M_j receives the variable x_j as

Algorithm 1 Construction of the Decision Diagram

Input: SOP representation f_C; Variable order $x_1, x_2 \ldots, x_n$.

Output: The decision diagram of f_C.

if SOP contains only a constant term γ **then**
  return constant node(γ)
end if
Initialize $\mathbf{A}$ as the adjacency matrix of variable graph G_C
for $i = 2$ **to** n **do**
  Compute basis $\{\mathbf{d}_j\}_{[i-1]}$ for the row space of $\mathbf{A}_{[i-1]}$
  Record the corresponding pivot positions $\{p_j\}_{j=1}^{r_i}$
end for
Initialize lookup table[i] for $1 \leq i \leq n-1$
Initialize the root node in level 1.
Set $g_1 = 0, \gamma_1 = 0$ for the root node.
for $i = 1$ **to** n **do** `//Iterate through levels and nodes`
  for each node in level i **do**
    Read the node's corresponding g_i and γ_i.
    `/* Process Zero Child (`$x_i = 0$`) */`
    Compute the zero child node's $g_{i,0}$ and $\gamma_{i,0}$.
    Determine level index s where the zero child node should be inserted.
    if s is $n+1$ **then**
      node.set_zero_child(constant_node ($\gamma_{i,0}$))
    else
      Compute tuple $(\gamma_{i,0}, c_1, \ldots, c_{r_s})$ using the pivots for $\mathbf{A}_{[s-1]}$.
      $m \leftarrow$ table[s].lookup_or_insert$(\gamma_{i,0}, c_s, \ldots, c_{r_s})$
      node.set_zero_child(m)
      Record $g_{i,0}$ and $\gamma_{i,0}$ for m.
    end if
    `/* Process One Child (`$x_i = 1$`) */`
    Compute the one child node's $g_{i,1}$ and $\gamma_{i,1}$.
    Determine level index t where the one child node should be inserted.
    if t is $n+1$ **then**
      node.set_one_child(constant node ($\gamma_{i,1}$))
    else
      Compute tuple $(\gamma_{i,1}, c_1, \ldots, c_{r_t})$ using the pivots for $\mathbf{A}_{[t-1]}$.
      $m \leftarrow$ table[t].lookup_or_insert$(\gamma_{i,1}, c_t, \ldots, c_{r_t})$
      node.set_one_child(m)
      Record $g_{i,1}$ and $\gamma_{i,1}$ for m.
    end if
  end for
end for
return the root node.

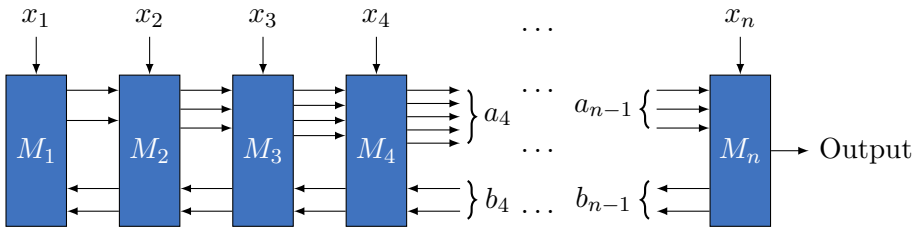

Figure 3: A linear network for computing Boolean functions. Figure adapted from [Knu09].

input. The network includes inter-module wires carrying Boolean signals, with a_j forward-propagating wires from M_j to M_{j+1} and b_j backward-propagating wires from M_{j+1} to M_j, for $1 \leq j \leq n-1$. By defining $a_0 = b_0 = b_n = 0$ and $a_n = 1$, we can specify that module M_j has $a_{j-1} + b_j + 1$ input bits and $a_j + b_{j-1}$ output bits. Theorem 4.1 provides a critical theorem bounding the size of $\mathrm{B}(f)$:

Theorem 4.1 (Theorem M of [Knu09])**.** *If a Boolean function f can be computed by the linear network in Fig. 3, then* $\mathrm{B}(f) \leq \sum_{j=0}^{n} 2^{a_j 2^{b_j}}$.

The theorem reveals that the number of backward signals b_j substantially influences the Binary Decision Diagram (BDD) size. Subsequently, we will construct a family of Boolean functions f (and corresponding IQP circuits) that exhibit a provably small BDD size according to Theorem 4.1, while maintaining a large tensor network contraction complexity.

We define a degree-3 polynomial $f : \{0,1\}^n \to \{0,1\}$ as follows:

$$f(x) = A(x) \sum_{i=1}^{n} x_i + \sum_{i=1}^{n-k+1} C_{i:i+k-1}, \tag{11}$$

where $A(x) := \sum_{i=1}^{n} \alpha_i x_i$, with α_i randomly selected from $\{0,1\}$, $k = \mathcal{O}(\log n)$, and $C_{i:j}$ consists exclusively of degree-3 terms involving variables $x_i, \ldots, x_j$. The design purposefully ensures that the second term $C_{i:i+k-1}$ involves k consecutive variables, guaranteeing that the number of forward signals for each module in the linear network remains bounded by $k+1$. An important reason to include the $C_{i:i+k-1}$ terms in this construction is that the corresponding IQP circuit would otherwise be a Clifford circuit, whose output amplitude can be efficiently computed using the Gottesman-Knill algorithm [Got98, AG04].

Below, we use the linear network model to bound the BDD size of f. For this reason, we refer to the resulting IQP circuits as *linear-network circuits*. This family of circuits has been numerically studied in [WCYJ25] and the authors shows that FeynmanDD outperforms existing DD-based simulators such as DDSIM [ZW19, ZHW19] and WCFLOBDD [SCR23, SCR24]. Meanwhile, Table 3 numerically confirms that FeynmanDD performs better on linear-network circuits than Quimb [Gra18] and TensorCircuit [ZAW+23], two simulation methods based on tensor networks. Note that in the numerical experiment, we used the binary synthesis method which supports three-qubit gates.

Module	Forward Signals	Backward Signal
M_1	$x_1, x_1A(x), A(x)$	
M_2	$x_1, x_2, A(x) \sum_{i=1,2} x_i, A(x)$	A_2
$\vdots$	$\vdots$	$\vdots$
M_{k-1}	$x_1, \ldots, x_{k-1}, A(x) \sum_{i=1}^{k-1} x_i, A(x)$	A_{k-1}
M_k	$x_2, \ldots, x_k, C_{1:k}, A(x) \sum_{i=1}^{k} x_i, A(x)$	A_k
$\vdots$	$\vdots$	$\vdots$
M_{n-1}	$x_{n-k+1}, \ldots, x_{n-1}, \sum_{i=1}^{n-k} C_{i:i+k-1}, A(x) \sum_{i=1}^{n-1} x_i, A(x)$	A_{n-1}
M_n	$f(x)$	A_n

Table 2: Linear-network construction for a function f with small BDD size.

Theorem 4.2. *The size of the Binary Decision Diagram (BDD) for $f(x)$ as defined in Eq. (11) is bounded by $\mathrm{B}(f) = \mathcal{O}\left(n2^k\right)$. Specifically, when $k = \mathcal{O}(\log n)$, we have $\mathrm{B}(f) = \mathcal{O}(\mathrm{poly}(n))$.*

Proof. Let $A_j(x) = \sum_{i=j}^{n} \alpha_i x_i$ represent a partial sum of $A(x)$ from index j to n, with $A_1(x) = A(x)$. We demonstrate that f can be computed using the linear network in Table 2. The computation proceeds through a systematic signal transmission process: For backward signals, module M_j sends A_j to module M_{j-1} when $j > 1$. This enables module M_1 to compute $A(x)$ and $x_1A(x)$. Subsequently, each module M_j receives $A(x)$ and $A(x)\sum_{i=1}^{j-1} x_i$ from M_{j-1}. It then computes and sends $A(x)\sum_{i=1}^{j} x_i$ to M_{j+1}. Through this signal propagation strategy, the first term of $f(x)$ in Eq. (11) can be computed across the linear network.

For the second term, the computation follows a progressive signal transmission strategy: When $j < k$, module M_j sends variables $x_1, \ldots, x_j$ to module M_{j+1}. At $j = k$, module M_k receives variables $x_1, \ldots, x_k$ and computes $C_{1:k}$. Since x_1 becomes unnecessary for subsequent computations, M_k sends $x_2, \ldots, x_k$ and $C_{1:k}$ to M_{k+1}. When $j = k + 1$, module M_{k+1} computes $C_{2:k+1}$. As x_2 is no longer required, M_{k+1} forwards $x_3, \ldots, x_{k+1}$ and the sum $C_{1:k} + C_{2:k+1}$ to M_{k+2}. This process continues, progressively eliminating unnecessary variables and accumulating computational results, ultimately enabling the computation of $f(x)$.

Since each module M_j has at most $k + 2$ forward signals and 1 backward signal, the BDD size of f is bounded by $\mathrm{B}(f) = \mathcal{O}\left(n2^k\right)$ according to Theorem 4.1. □

We will now demonstrate that the tensor network method has an exponentially lower bounded contraction complexity. This result is anticipated, as the following observations suggest: When each $\alpha_i = 1$, the first term of $f(x)$ generates a complete graph in the corresponding tensor network. Even with α_i randomly selected, the subgraph's connectivity remains sufficiently extensive to render tensor network contraction challenging. In the subsequent analysis, we will formally establish a

lower bound for this complexity, formally stated in the following theorem.

Theorem 4.3. *Tensor network simulation has exponential time and space complexity for the IQP circuit corresponding to the function in Eq.* (11).

As a preliminary step, we will first provide a precise definition of contraction complexity.

Definition 4.1 (Contraction Complexity)**.** Given a graph G and a contraction ordering π of its edges, the contraction complexity of π is defined as the maximum degree of the merged vertices formed in the contraction process, denoted as $cc(\pi)$. The contraction complexity of G $\mathrm{cc}(G)$ is defined as the minimum of $cc(\pi)$ over all possible orderings: $\mathrm{cc}(G) := \min_\pi \mathrm{cc}(\pi)$.

Then, we give the following proposition.

Proposition 4.1. *Given a graph $G = (V, E)$, let $G' = (V', E')$ be a subgraph of G, that is, G' is obtained from G by deleting edges or vertices. Then* $\mathrm{cc}(G') \leq \mathrm{cc}(G)$.

Proof. Let π be a contraction ordering of G and let π' be an induced ordering of π by removing $\pi(i) \notin E'$ from π. The degrees of the merged vertices of G' are all smaller than or equal to that of G, which implies $\mathrm{cc}(\pi') \leq \mathrm{cc}(\pi)$ and thus $\mathrm{cc}(G') \leq \mathrm{cc}(G)$. □

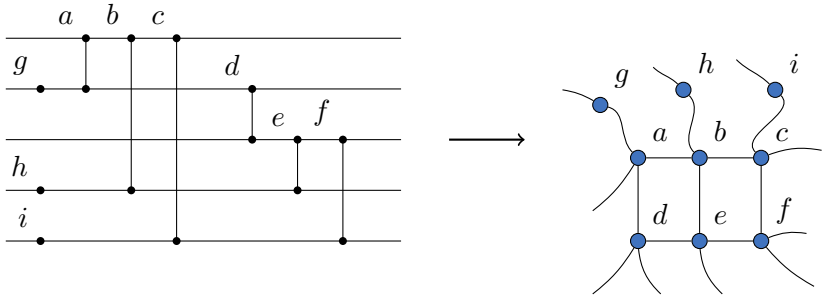

Figure 4: Left: A subcircuit of circuit specified by Eq. (11) with $\boldsymbol{\alpha} = (0, 1, 0, 1, 1)$. Right: The corresponding subgraph forming a lattice.

In Proposition 4.2 of [MS08], a key relationship $\mathrm{cc}(G) = \mathrm{tw}(G^*)$ is established, where G^* is the line graph of G. A complementary result from the same reference provides an important lower bound:

$$\mathrm{tw}(G^*) \geq \frac{\mathrm{tw}(G) - 1}{2}.$$

Our objective is to lower bound the treewidth of the graph G corresponding to our constructed circuits. In conjunction with Proposition 4.1, this task reduces to bounding the treewidth of the subgraph derived from a subcircuit.

Proof of Theorem 4.3. Consider a subcircuit transformed from the following terms of $f(x)$ in Eq. (11),

$$\sum_{i:\alpha_i=1} x_i + \sum_{\substack{i:\alpha_i=1 \\ j:\alpha_j=0}} x_i x_j \ ,$$

which consists of Z gates and CZ gates. Fig. 4 shows an example for $\boldsymbol{\alpha} = (0, 1, 0, 1, 1)$. A tensor with two legs represents a Z gate, and a tensor with four legs represents a CZ gate. It can be seen that the tensor-network representation embeds a lattice as a subgraph. The CZ gates can be partitioned into $n - w$ groups, where $w = |\boldsymbol{\alpha}|$ is the Hamming weight of $\boldsymbol{\alpha}$. For example, there are two groups in Fig. 4, $\{a, b, c\}$ and $\{d, e, f\}$. Each row in the lattice represents one group of CZ gates, with w vertices. Thus, the lattice is $w \times (n - w)$. For such a lattice, the treewidth is $\min(w, n - w)$. Therefore, if $w = n/c$ for some constant $c > 1$, then the time complexity of using tensor-network method is $2^{\Omega(n)}$. □

4.2 Linear rank-width construction

We now turn to the gate set $\mathcal{T}$ and use the results established in Section 3.1 to construct IQP circuits whose variable graphs have small linear rank-width but large treewidth.

Let $\mathbf{A} := a\mathbf{v}_0\mathbf{v}_0^T + \sum_{i=1}^{k} \mathbf{v}_i\mathbf{v}_i^T$ be a symmetric matrix over $\mathbb{F}_2$, where $\mathbf{v}_0$ is the all-ones vector, a is a random bit, $\mathbf{v}_i \in \mathbb{F}_2^n$ are randomly chosen vectors for $i \in [k]$, and $k = \mathcal{O}(\log n)$. Let $\mathbf{b} \in \mathbb{Z}_8^n$ be a random vector. Define a polynomial $f : \{0, 1\}^n \to \mathbb{Z}_8$ as $f(\mathbf{x}) = 4\mathbf{x}^T\mathbf{A}_{\mathsf{up}}\mathbf{x} + \mathbf{b}^T\mathbf{x}$. A corresponding IQP circuit can be constructed as $C = \mathrm{H}^{\otimes n}\, D\, \mathrm{H}^{\otimes n}$ with $D = \prod_{i<j} \mathrm{CZ}_{ij}^{A_{ij}} \prod_i T_i^{b_i}$. The zero-to-zero amplitude of C is given by

$$\langle 0^n|C|0^n\rangle = \frac{1}{2^n} \sum_{\mathbf{x}} \omega_8^{f(\mathbf{x})}.$$

Next, we show that this family of IQP circuits can be efficiently simulated by FeynmanDD by bounding the linear rank-width of the variable graph G_C. Observe that each row of $\mathbf{A}$ is a linear combination of $\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_k$, meaning that $\mathrm{rank}(\mathbf{A}) \leq k + 1$. Since the cut-rank function $\rho_G(X) = \mathrm{rank}(\mathbf{A}[X, V \setminus X])$ returns the rank of a submatrix of $\mathbf{A}$, we have $\rho_G(X) \leq k + 1$ for all $X \subseteq V$. Thus, the linear rank-width of G_C is bounded by $k + 1$ and the FeynmanDD has size $O(n2^k) = O(\mathrm{poly}(n))$.

Then, we analyze the time complexity of the tensor network method for simulating this family of IQP circuits. Denote the quantum circuit corresponding to $a = 0$ as C_0 and the one corresponding to $a = 1$ as C_1. Since $\mathbf{v}_0$ is an all-ones vector, the graph G_{C_1} is the complement of G_{C_0}. The Nordhaus-Gaddum property [JW12] of treewidth states that for any graph G with n vertices,

$$\mathrm{tw}(G) + \mathrm{tw}(\overline{G}) \geq n - 2.$$

Therefore, at least one of G_{C_0} and G_{C_1} has treewidth at least $(n - 2)/2 = \Omega(n)$. This implies the tensor network method has time complexity $2^{\Omega(n)}$ for at least one of these two circuits while FeynmanDD is efficient for both.

4.3 Numerical experiments

We conduct numerical experiments to compare the performance of FeynmanDD with two state-of-the-art tensor network simulators, Quimb [Gra18] and Tensor-Circuit [ZAW+23]. The experiments are performed on a server equipped with two Intel Xeon Platinum 8358P CPUs (each with 32 cores/64 threads) and 512 GiB of memory.

			FeynmanDD		quimb		TensorCircuit	
n	k	Gates	Time	Mem	Time	Mem	Time	Mem
20	5	162.0	0.002	12.88	0.0033/1.7682	202.93	1.81/34.78	473.47
20	7	163.0	0.003	12.88	0.0031/2.1038	204.79	1.83/42.73	477.11
30	5	319.8	0.003	12.90	0.0090/3.8352	213.92	3.31/86.50	539.16
30	7	321.1	0.003	12.83	0.0154/4.2740	220.81	3.42/95.95	558.22
40	5	527.3	0.004	12.87	0.7052/7.9125	932.70	5.80/127.72	821.91
40	7	531.7	0.005	12.88	0.9697/11.2940	1433.34	6.20/129.52	1293.28

Table 3: Quantum circuit simulation benchmarks on linear-network circuits using three backends: FeynmanDD, quimb, and TensorCircuit. In the table, n represents the number of qubits, and rank k is a constant measuring gate locality. For each pair of (n, k) parameters, 10 circuit instances are generated for testing. Time is measured in seconds (s), and memory usage is measured in MB. The timing data of tensor network simulators is presented as contraction time/total time.

		FeynmanDD		quimb		TensorCircuit	
n	Gates	Time	Mem	Time	Mem	Time	Mem
				CZ v2 d10			
16	115	0.0069	5.73	0.0015/0.82200	196.75	0.46/13.53	418.82
20	145	0.0123	7.13	0.0012/11.1465	204.42	0.59/23.39	427.34
25	184	0.0235	9.66	0.0013/0.49820	200.52	0.68/30.87	434.68
				iS v1 d10			
16	115	0.0186	8.41	0.0012/20.9540	204.62	0.46/16.92	425.38
20	145	0.0485	15.80	0.0012/11.1722	201.10	0.57/21.13	432.32
25	184	0.1024	28.03	0.0015/0.49450	199.61	0.66/28.70	440.32

Table 4: Quantum circuit simulation benchmarks on Google quantum circuits (CZ v2 d10 and iS v1 d10) using three backends: FeynmanDD, quimb, and TensorCircuit. In the table, n represents the number of qubits. For each n, 10 circuit instances are generated for testing. Time is measured in seconds (s), and peak memory usage is measured in MB. The timing data of tensor network simulators is presented as contraction time/total time.

			FeynmanDD		quimb		TensorCircuit	
n	k	Gates	Time	Mem	Time	Mem	Time	Mem
20	5	163.3	0.0050	5.04	0.0034/2.0102	200.68	1.39/ 78.30	455.77
20	7	160.0	0.0104	6.34	0.0034/2.2925	201.47	1.31/ 58.13	453.52
30	5	324.8	0.0102	6.25	0.1457/6.0830(1)	389.12(1)	3.18/110.55	647.22
30	7	321.2	0.0317	10.99	0.1387/5.7132	330.47	3.24/110.97	703.99
40	5	537.0	0.0146	7.50	4.8298/50.317	3960.82	140.87/268.17	66586.96
40	7	532.4	0.0511	15.79	14.951/258.46	10794.56	186.46/312.41	102544.06

Table 5: Quantum circuit simulation benchmarks on linear rank-width circuits. In the table, n represents the number of qubits, and rank k is a constant measuring gate locality. For each pair of (n, k) parameters, 10 circuit instances are generated for testing. Time is measured in seconds (s), and memory usage is measured in MB. The timing data of tensor network simulators is presented as contraction time/total time. The (x) mark indicates that x tests in the group of ten circuits throws out-of-memory (OOM) and the displayed value is derived from the average of the remaining $10 - x$ test results.

We first benchmark the three simulators on two families of Google supremacy circuits: CZ v2 d10 and iS v1 d10.[2] We then benchmark linear-network circuits, based on the construction in Section 4.1, and a new circuit family with bounded linear rank-width, based on the construction in Section 4.2.

For each family of circuits, we generate 10 random instances per parameter configuration. We measure both runtime and peak memory usage for computing the zero-to-zero amplitude of each circuit. We use quimb's default `amplitude` method for amplitude computation. For TensorCircuit, we employ the built-in `amplitude` method along with the customized contraction path finder recommended in section 6.5.1 of [ZAW+23]. For the two tensor network simulators, quimb and TensorCircuit, the execution time is disaggregated into the duration of the tensor contraction itself and the total runtime, which incorporates the time for path finding and other overheads. FeynmanDD's timing encompasses the entire process from DD construction to counting completion.

The results, summarized in Tables 4 and 5, demonstrate that FeynmanDD significantly outperforms the two tensor network simulators on circuits exhibiting bounded linear rank-width, achieving superior performance in both simulation time and memory usage. For the Google supremacy circuits, the current FeynmanDD implementation is faster than TensorCircuit but slower than Quimb's contraction time. Whether this speed gap is a fundamental limitation of the method or merely an artifact of the implementation remains a subject for future exploration. Given that heuristics for minimizing the linear rank-width are largely unexplored, we did not compare the variable ordering finding cost across the methods, instead adopting the qubit order proposed in [WCYJ25].

References

[AAB+19] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao,

[2]The benchmark circuits are from the repository at https://github.com/sboixo/GRCS.

Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[AG04] Scott Aaronson and Daniel Gottesman. Improved Simulation of Stabilizer Circuits. *Physical Review A*, 70(5):052328, 2004.

[AK15] Isolde Adler and Mamadou Moustapha Kanté. Linear rank-width and linear clique-width of trees. *Theoretical Computer Science*, 589:87–98, 2015.

[ALM07] Dorit Aharonov, Zeph Landau, and Johann Makowsky. The quantum FFT can be classically simulated, 2007.

[AMSDS20] Jeremy C. Adcock, Sam Morley-Short, Axel Dahlberg, and Joshua W. Silverstone. Mapping graph state orbits under local complementation. *Quantum*, 4:305, 2020.

[BFG+93] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, pages 188–191, Santa Clara, CA, USA, 1993. IEEE Comput. Soc. Press.

[BG16] Sergey Bravyi and David Gosset. Improved classical simulation of quantum circuits dominated by Clifford gates. *Physical Review Letters*, 116(25):250501, 2016.

[BISN18] Sergio Boixo, Sergei V. Isakov, Vadim N. Smelyanskiy, and Hartmut Neven. Simulation of low-depth quantum circuits as complex undirected graphical models, 2018.

[BJS11] Michael J. Bremner, Richard Jozsa, and Dan J. Shepherd. Classical simulation of commuting quantum computations implies collapse of the polynomial hierarchy. *Proc. R. Soc. A*, 467(2126):459–472, 2011.

[BMS16] Michael J. Bremner, Ashley Montanaro, and Dan J. Shepherd. Average-Case Complexity Versus Approximate Simulation of Commuting Quantum Computations. *Phys. Rev. Lett.*, 117(8):080501, 2016.

[Bry86] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[Bry95] R.E. Bryant. Binary decision diagrams and beyond: enabling technologies for formal verification. In *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*, pages 236–243, San Jose, CA, USA, 1995. IEEE Comput. Soc. Press.

[DW18] Axel Dahlberg and Stephanie Wehner. Transforming graph states using single-qubit operations. *Phil. Trans. R. Soc. A.*, 376(2123):20170325, 2018.

[FG06] Jörg Flum and M. Grohe. *Parameterized complexity theory*. Texts in theoretical computer science. Springer, Berlin ; New York, 2006.

[Got98] Daniel Gottesman. The Heisenberg Representation of Quantum Computers. *arXiv:quant-ph/9807006*, 1998.

[Gra18] Johnnie Gray. quimb: a python library for quantum information and many-body calculations. *Journal of Open Source Software*, 3(29):819, 2018.

[HEB04] M. Hein, J. Eisert, and H. J. Briegel. Multiparty entanglement in graph states. *Phys. Rev. A*, 69(6):062311, 2004.

[Hli18] Petr Hliněný. A Simpler Self-reduction Algorithm for Matroid Path-Width. *SIAM J. Discrete Math.*, 32(2):1425–1440, 2018.

[HO08] Petr Hliněný and Sang-il Oum. Finding Branch-Decompositions and Rank-Decompositions. *SIAM Journal on Computing*, 38(3):1012–1032, 2008.

[HZN+21] Cupjin Huang, Fang Zhang, Michael Newman, Xiaotong Ni, Dawei Ding, Junjie Cai, Xun Gao, Tenghui Wang, Feng Wu, Gengyan Zhang, Hsiang-Sheng Ku, Zhengxiong Tian, Junyin Wu, Haihong Xu, Huanjun Yu, Bo Yuan, Mario Szegedy, Yaoyun Shi, Hui-Hai Zhao, Chunqing Deng, and Jianxin Chen. Efficient parallelization of tensor network contraction for simulating quantum computation. *Nature Computational Science*, 1(9):578–587, 2021.

[JW12] Gwenaël Joret and David R. Wood. Nordhaus-Gaddum for treewidth. *European Journal of Combinatorics*, 33(4):488–490, 2012.

[Kit97] Alexei Kitaev. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191–1249, 1997.

[KLR+08] E. Knill, D. Leibfried, R. Reichle, J. Britton, R. B. Blakestad, J. D. Jost, C. Langer, R. Ozeri, S. Seidelin, and D. J. Wineland. Randomized benchmarking of quantum gates. *Physical Review A*, 77(1):012307, 2008.

[Knu09] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 1 (Bitwise Tricks & Techniques; Binary Decision Diagrams)*. AddisonWesley Professional, Upper Saddle River, NJ, 1 edition edition, 2009.

[KS93] Ephraim Korach and Nir Solel. Tree-width, path-width, and cutwidth. *Discrete Applied Mathematics*, 43(1):97–101, 1993.

[MGE11] Easwar Magesan, J. M. Gambetta, and Joseph Emerson. Scalable and Robust Randomized Benchmarking of Quantum Processes. *Physical Review Letters*, 106(18):180504, 2011.

[MMBS04] Paul Molitor, Janett Mohnke, Bernd Becker, and Christoph Scholl. *Equivalence Checking of Digital Circuits: Fundamentals, Principles, Methods*. Springer New York, NY, 2004.

[MS08] Igor L. Markov and Yaoyun Shi. Simulating Quantum Computation by Contracting Tensor Networks. *SIAM Journal on Computing*, 38(3):963–981, 2008.

[NC00] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[O'G19] Bryan O'Gorman. Parameterization of tensor network contraction. *LIPIcs, Volume 135, TQC 2019*, 135:10:1–10:19, 2019.

[Orú14] Román Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, 349:117–158, 2014.

[OS06] Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006.

[Oum05] Sang-il Oum. Rank-width and vertex-minors. *Journal of Combinatorial Theory, Series B*, 95(1):79–100, 2005.

[Oum17] Sang-il Oum. Rank-width: Algorithmic and structural results. *Discrete Applied Mathematics*, 231:15–24, 2017.

[PGVWC07] D. Perez-Garcia, F. Verstraete, M. M. Wolf, and J. I. Cirac. Matrix Product State Representations. *arXiv:quant-ph/0608197*, 2007.

[SB09] Dan Shepherd and Michael J. Bremner. Temporally unstructured quantum computation. *Proc. R. Soc. A*, 465(2105):1413–1439, 2009.

[SCR23] Meghana Sistla, Swarat Chaudhuri, and Thomas Reps. Symbolic quantum simulation with quasimodo. In *Computer Aided Verification: 35th International Conference, CAV 2023, Paris, France, July 17–22, 2023, Proceedings, Part III*, pages 213–225, Berlin, Heidelberg, 2023. Springer-Verlag.

[SCR24] Meghana Sistla, Swarat Chaudhuri, and Thomas Reps. Weighted Context-Free-Language Ordered Binary Decision Diagrams. *Weighted CFLOBDDs*, 8(OOPSLA2):320:1390–320:1419, 2024.

[Val02] Leslie G. Valiant. Quantum Circuits That Can Be Simulated Classically in Polynomial Time. *SIAM Journal on Computing*, 31(4):1229–1254, 2002.

[VdNDVB07] M. Van den Nest, W. Dür, G. Vidal, and H. J. Briegel. Classical simulation versus universality in measurement-based quantum computation. *Physical Review A*, 75(1):012337, 2007.

[Vid04] Guifré Vidal. Efficient Simulation of One-Dimensional Quantum Many-Body Systems. *Physical Review Letters*, 93(4):040502, 2004.

[Vid08] G. Vidal. Class of Quantum Many-Body States That Can Be Efficiently Simulated. *Physical Review Letters*, 101(11):110501, 2008.

[WCYJ25] Ziyuan Wang, Bin Cheng, Longxiang Yuan, and Zhengfeng Ji. FeynmanDD: Quantum Circuit Analysis with Classical Decision Diagrams. In *Proceedings of the 37th International Conference on Computer Aided Verification*, Zagreb, Croatia, 2025.

[Weg00] Ingo Wegener. *Branching programs and binary decision diagrams: theory and applications*. SIAM monographs on discrete mathematics and applications. Society for Industrial and Applied Mathematics, Philadelphia, 2000.

[ZAW+23] Shi-Xin Zhang, Jonathan Allcock, Zhou-Quan Wan, Shuo Liu, Jiace Sun, Hao Yu, Xing-Han Yang, Jiezhong Qiu, Zhaofeng Ye, Yu-Qin Chen, Chee-Kong Lee, Yi-Cong Zheng, Shao-Kai Jian, Hong Yao, Chang-Yu Hsieh, and Shengyu Zhang. TensorCircuit: a quantum software framework for the NISQ era. *Quantum*, 7:912, 2023.

[ZHW19] Alwin Zulehner, Stefan Hillmich, and Robert Wille. How to Efficiently Handle Complex Values? Implementing Decision Diagrams for Quantum Computing. *arXiv:1911.12691 [quant-ph]*, 2019.

[ZW19] Alwin Zulehner and Robert Wille. Advanced Simulation of Quantum Computations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(5):848–859, 2019.