

PHOENIX: Pauli-based High-level Optimization Engine for Instruction Execution on NISQ Devices

Abstract—Variational quantum algorithms (VQA) based on Hamiltonian simulation represent a specialized class of quantum programs well-suited for near-term quantum computing applications due to its modest resource requirements in terms of qubits and circuit depth. Unlike the conventional single-qubit (1Q) and two-qubit (2Q) gate sequence representation, Hamiltonian simulation programs are essentially composed of disciplined subroutines known as Pauli exponentiations (Pauli strings with coefficients) that are variably arranged. To capitalize on these distinct program features, this study introduces PHOENIX, a highly effective compilation framework that primarily operates at the high-level Pauli-based intermediate representation (IR) for generic Hamiltonian simulation programs. PHOENIX exploits global program optimization opportunities to the greatest extent, compared to existing SOTA methods despite some of them also utilizing similar IRs. PHOENIX employs the binary symplectic form (BSF) to formally describe Pauli strings and reformulates IR synthesis as reducing the column weights of BSF by appropriate Clifford transformations. It comes with a heuristic BSF simplification algorithm that searches for the most appropriate 2Q Clifford operators in sequence to maximally simplify the BSF at each step, until the BSF can be directly synthesized by basic 1Q and 2Q gates. PHOENIX further performs a global ordering strategy in a Tetris-like fashion for these simplified IR groups, carefully balancing optimization opportunities for gate cancellation, minimizing circuit depth, and managing qubit routing overhead. Experimental results demonstrate that PHOENIX outperforms SOTA VQA compilers across diverse program categories, backend ISAs, and hardware topologies.

I. INTRODUCTION

Quantum computing offers the potential to revolutionize various fields, driving decades of efforts to develop the required physical hardware. For instance, quantum algorithms can achieve exponential speedups in tasks such as integer factorization [1], solving linear equations [2], and quantum system simulation [3]. In the noisy intermediate-scale quantum (NISQ) era where we have access to dozens or hundreds of qubits susceptible to noise (for example, qubit decoherence, gate imperfections, environmental coupling with qubits) [4], variational quantum algorithms (VQA) is a leading class of algorithms proposed to achieve the quantum advantage (e.g., VQE for chemistry simulation and condensed-matter physical system simulation [5], QAOA for combinatorial optimization [6]) due to its modest resource requirements in terms of qubit number and circuit depth as well as its noise-resilience. [7].

The construction of a VQA ansatz circuit is to simulate (approximate) a desired unitary evolution under the system Hamiltonian H and the evolution duration t , through Trotterizing [7] the evolution $U(t)$ given H represented by a linear combination of Pauli strings:

$$U(t) = e^{-iHt} \simeq (S_k(\tau))^r, \tau = \frac{t}{r}, \quad (1)$$

$$H = \sum_{j=1}^L h_j P_j = \sum_{j=1}^L h_j \sigma_0^{(j)} \otimes \cdots \otimes \sigma_{n-1}^{(j)}, \quad (2)$$

where k and r indicate the Trotter orders and steps, respectively. More fine-grained Trotterization procedures results in lower approximation

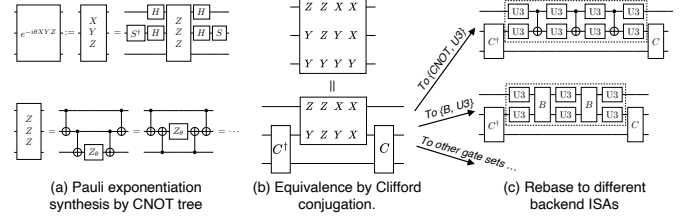


Fig. 1. Conventional Pauli exponentiation synthesis v.s. Simultaneous simplification by Clifford conjugation. (a) Pauli exponentiations are synthesized by gate set $\{H, S, S^\dagger, Z(\theta), \text{CNOT}\}$, through variable CNOT-tree unrolling schemes. (b) Multiple 3Q Pauli exponentiations can be simultaneously simplified into 2Q Pauli exponentiations, through a 2Q Clifford conjugation, where $C = (H \otimes S)\text{CNOT}(H \otimes S^\dagger)$ in this example. (c) The simplified Pauli exponentiations can be rebased to versatile quantum gate sets, such as CNOT-based and B-based ISAs, with significantly reduced 2Q gate count.

errors. For example, 1st order and 2nd order Trotterization procedures are given by

$$S_1 = \prod_{j=1}^L \exp^{-ih_j \tau P_j}, S_2 = \prod_{j=1}^L \exp^{-ih_j \frac{\tau}{2} P_j} \prod_{j=L}^1 \exp^{-ih_j \frac{\tau}{2} P_j},$$

respectively. Following Trotterization, each term S_k is expressed as a product of individual Pauli exponentiations, which can conventionally be easily synthesized using basic 1Q and 2Q gates. The arrangement of these Pauli exponentiations within each Trotter step can be freely chosen without affecting the upper bound of Trotterization (approximation) error [7], which is usually not comparable to physical noise. However, different arrangements offer varying opportunities for optimizing quantum circuits, thereby dominating the accuracy of Hamiltonian simulations on noisy hardware. Consequently, the primary challenge of compiling VQA ansatz circuits lies in synthesizing these Pauli exponentiations from a global perspective, dubbed *Pauli-based intermediate representations (IR)* throughout this paper.

Conventionally, a Pauli exponentiation is synthesized into a 1Q rotation $Z(\theta)$ sandwiched by a pair of symmetric CNOT trees, subsequently conjugated by some H and S gates, as exemplified in Fig. 1. Existing state-of-the-art (SOTA) compilers primarily exploit gate cancellation opportunities exposed by the already synthesized subcircuits, whether by means of the abstract ZX diagram [8], [9], [10] representation or the variants of CNOT trees [11], [12]. Although utilize the Pauli-based IR, the optimization process they formulate is limited to subcircuits and local IR patterns. Furthermore, it assumes the conventional CNOT-based quantum instruction set architecture (ISA). We present instead that a set of Pauli exponentiations *can be simultaneously simplified* through approximate Clifford transformations. For example, Fig. 1 (b) shows that the list of weight-3 Pauli strings $[ZYY; ZZY; XYY; XXY]$ can be simplified into a weight-2 Pauli string list through the conjugation of a 2Q Clifford operator $C = (H \otimes S)\text{CNOT}(H \otimes S^\dagger)$. This opens a door to greater optimization opportunities *totally at the level of Pauli-based IR*. It

also does not make any assumptions on the quantum ISA being used.

In this work, we propose PHOENIX (*Pauli-based High-level Optimization ENgine for Instruction eXecution*)—a highly effective compilation framework for generic Hamiltonian simulation programs on near-term quantum devices. PHOENIX follows the

“IR grouping → group-wise simplification → IR group ordering”

pipeline to compile real-world VQA programs into basic quantum gates. It is ISA-independent, routing-aware, and tailored to generic VQA programs such as molecular simulation involving heterogeneous-weight Pauli strings and QAOA including only weight-2 Pauli strings. Differing from existing SOTA methods, PHOENIX utilizes an alternative formal description for Pauli-based IR and integrates heuristic optimization strategies to achieve global optimization primarily at the high-level semantic layer. We evaluate PHOENIX across diverse VQA programs, backend ISAs, and hardware topologies, demonstrating its superior performance over existing SOTA compilers. Our main contributions are as follows:

- 1) We utilize the *binary symplectic form* (BSF) as the formal description of Pauli-based IR and reformulate the IR synthesis process as simultaneous simplification on BSF by appropriate Clifford conjugations that simultaneously reduce the BSF’s column weights, enabling global optimization to the largest extent while operating at the high-level IR.
- 2) We propose a heuristic *BSF simplification* algorithm as the core optimization pass in PHOENIX. It iteratively searches for the most appropriate 2Q Clifford operators to quickly lower the weight of the BSF, until the BSF can be directly synthesized by basic 1Q and 2Q gates.
- 3) We quantify the circuit depth overhead induced by assembling (ordering) simplified IR groups. Using this, we devise a uniform cost function that comprehensively accounts for circuit depth overhead, gate cancellation, and qubit routing overhead. Under the guidance of that, PHOENIX integrates a *Tetris-like IR group ordering* procedure to robustly achieve a good ordering scheme.

Overall, PHOENIX outperforms the best-known compilers (e.g., TKET [13], PAULIHEDRAL [11], TETRIS [12]), achieving significant reductions in gate count and circuit depth. For example, for logical-level compilation, PHOENIX results in 80.44% reduction in CNOT gate count and 82.71% reduction in 2Q circuit depth on average, compared to the original logical circuits. For hardware-aware compilation with heavy-hex topology, PHOENIX reduces by 35.9% (22.3%) in CNOT gate count and 43.6% (27.8%) in 2Q circuit depth on average, compared to PAULIHEDRAL (TETRIS). The reduction effect in 2Q gate count and circuit depth becomes even more impressive when targeting the newly introduced SU(4) ISA (the representative continuous ISA containing all 2Q gates) [14].

II. RELATED WORKS

Based on ZX diagram representations of Pauli gadgets. Compilers such as TKET [13], [8], PAULIHEDRAL [9], and PCOAST [10] resynthesize quantum circuits into its ZX diagram representation. ZX diagrams are used to visually represent and simplify the commutation relations and phase interactions between Pauli operators, facilitating the reduction of circuit depth and gate count through algebraic and graphical transformations. Although it can efficiently resynthesize Pauli gadgets, the commutation rules it leverages occur locally and it is hard to operate in a hardware-aware manner.

Based on synthesis variants of Pauli-based IR. Dedicated compilers like PAULIHEDRAL [11] and TETRIS [12] leverage Pauli-based IR to identify gate cancellation opportunities between nearest-

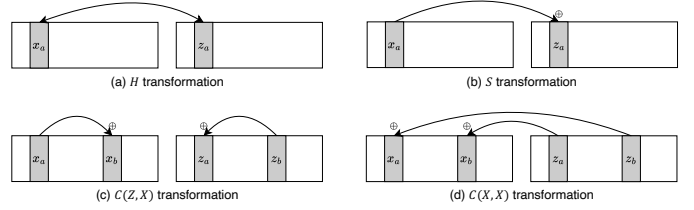


Fig. 2. Examples of Clifford transformations on BSF. Columns a and b , corresponding to qubit a and b , for X and Z blocks are indicated by x_a, x_b, z_a, z_b , respectively. (a) H gate acting on qubit a will exchange x_a and z_a . (b) S gate acting on qubit a results in $z_a \leftarrow z_a \oplus x_a$. (c) $C(Z, X)_{a,b}$, i.e., the CNOT gate, results in $x_b \leftarrow x_b \oplus x_a$ and $z_a \leftarrow z_a \oplus z_b$. (d) $C(X, X)$ transformation is equivalent with successively applying H on a , $C(Z, X)_{a,b}$, and H on a , resulting in $x_a \leftarrow x_a \oplus z_b$ and $x_b \leftarrow x_b \oplus z_a$.

neighbor IRs, exposed by the variants of their synthesis schemes based on CNOT-tree unrolling. They proposed sophisticated co-optimization techniques to minimize CNOT gate for both logical-level synthesis and SWAP-based routing, achieving good performance especially on limited-topology NISQ devices. However, their optimization scope is confined to a finite set of local subcircuit patterns, and they rely solely on the CNOT-based ISA.

III. BSF AND CLIFFORD FORMALISM

We establish our synthesis scheme by representing Pauli strings in the binary symplectic form (BSF) [15]. In the BSF, each n -qubit Pauli string is represented as a row in a tableau, with columns divided into two sections: $[X | Z]$. Here, $[X_{i,j} | Z_{i,j}]$ denotes the j -th component of the i -th Pauli operator. The encoding maps X to $[1 | 0]$, Z to $[0 | 1]$, Y to $[1 | 1]$, and I to $[0 | 0]$. For instance, the Pauli string simplification shown in Fig. 1 (b) follows the formulation

$$\left[\begin{array}{cc|cc} 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \end{array} \right] \xrightarrow{C(X,Y)_{1,2}} \left[\begin{array}{cc|cc} 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{array} \right],$$

where $C(X, Y)_{1,2}$ is the Clifford gate in Fig. 1 (b) while written in the *universal controlled gate* representation

$$C(\sigma_0, \sigma_1) = \frac{1}{2}((I + \sigma_0) \otimes I + (I - \sigma_0) \otimes \sigma_1), \sigma_j \in \{X, Y, Z\}.$$

Any universal controlled gate is equivalent to CNOT up to local H and S conjugations. For example, $C(Z, X) = \text{CNOT}$, and $C(X, Y) = (H \otimes S)\text{CNOT}(H \otimes S^\dagger)$.

By definition, any operator C from the Clifford group is unitary and has the property that CPC^\dagger is also a Pauli operator for any Pauli operator P . The update rules of some Clifford operators in terms of the BSF are demonstrated in Fig. 2, in which only 2Q Clifford operators (Clifford2Q) potentially have nontrivial effects of reducing the Pauli strings’ weights. For any $C(\sigma_0, \sigma_1)$, its tableau update rule is a combinatorial sequence of H , S and $C(Z, X)$ (or $C(Z, Z)$) transformations. For example, $C(X, Y)_{a,b}$ exhibits the rule

$$[x_a, x_b | z_a, z_b] \rightarrow [x_a \oplus x_b \oplus z_b, z_a \oplus z_b | z_a, z_a \oplus z_b].$$

Therefore, the basic approach of our synthesis scheme is to find an appropriate sequence of Clifford2Q operators to simplify the BSF¹ until it can be directly synthesized by basic 1Q and 2Q gates, i.e., the quantity

$$\text{total_weight} := \|\vee_i (r_x^{(i)} \vee r_z^{(i)})\| \quad (3)$$

¹The BSF that needs to simplify does not contain any weight-1 Pauli string.

Algorithm 1: Pauli Strings Simplification in BSF

Input : Pauli strings list pls
Output: Reconfigured circuit components list cfg

```

1  $cfg \leftarrow \emptyset$ ;  $bsf \leftarrow BSF(pls)$ ;  $cliffs\_with\_locals \leftarrow \emptyset$ ;
2 while  $bsf.TOTALWEIGHT() > 2$  do
3    $local\_bsf \leftarrow bsf.POPLOCALPAULIS()$ ;
4    $C \leftarrow \emptyset$ ; // Clifford2Q candidates
5    $B \leftarrow \emptyset$ ; // Each element of B results from applying each Clifford2Q candidate on  $bsf$ 
6    $costs \leftarrow \emptyset$ ; // Cost functions calculated on each element of B
7   for  $cg$  in CLIFFORD_2Q_SET do
8     for  $i, j$  in COMBINATIONS(RANGE( $n$ ), 2) do
9        $cliff \leftarrow cg.ON(i, j)$ ; // qubits acted on
10       $bsf' \leftarrow bsf.APPLYCLIFFORD2Q(cliff)$ ;
11       $cost \leftarrow CALCULATEBSFCOST(bsf')$ ;
12       $C.APPEND(cliff)$ ;
13       $B.APPEND(bsf')$ ;
14       $costs.APPEND(cost)$ ;
15    end
16  end
17   $bsf \leftarrow BSFWITHMINCOST(B, costs)$ ;
18   $cliff \leftarrow CLIFFORDWITHMINCOST(C, costs)$ ;
19   $cliffs\_with\_locals.APPEND((cliff, local\_bsf))$ ;
20 end
21  $cfg.APPEND(bsf)$ ;
22 for  $cliff, local\_bsf$  in  $cliffs\_with\_locals$  do
23   // Clifford2Q operators are added as conjugations, with local Pauli strings peeled before each epoch
24    $cfg.PREPEND(cliff)$ ;
25    $cfg.APPEND(local\_bsf)$ ;
26 end

```

is no more than 2, where $r_{x/z}$ is the r -th row of X or Z block of the BSF, and the norm is the sum of all binary entries. The key lies in how to find the Clifford2Q that can reduce the weights of as many Pauli strings as possible within the BSF.

IV. OUR PROPOSAL: PHOENIX

A. Overview

In the proposed framework PHOENIX, Pauli-based IRs (exponentiations) are first grouped according to the set of qubit indices with non-zero weight. Then, the *BSF simplification* algorithm is applied to the BSF representation of each IR group, generating the simplified subcircuit composed of CNOT-equivalent Clifford2Q operators and Pauli exponentiations with weights no more than 2. PHOENIX further selectively assemble (order) these unarranged simplified IR groups in a Tetris-like style that minimizes a uniform subcircuit assembling cost function that incorporates gate cancellation, circuit depth increase, and qubit routing overhead. By defining appropriate metric functions and heuristic algorithms, PHOENIX achieves superior optimization (less 2Q gate count and circuit depth) compared to SOTA methods, efficiently handling compilation for large-scale VQA programs.

B. BSF simplification for each IR group

To search for appropriate Clifford2Q operators in BSF simplification, it suffices to just focus on a set of Clifford2Q generators of the 2Q Clifford group. We choose the six universal controlled gates

$$\{C(X, X), C(Y, Y), C(Z, Z), C(X, Y), C(Y, Z), C(Z, X)\} \quad (4)$$

that are independent of each other and span the entire 2Q Clifford group [16]. It is a natural choice of sets of Clifford2Q generators, as

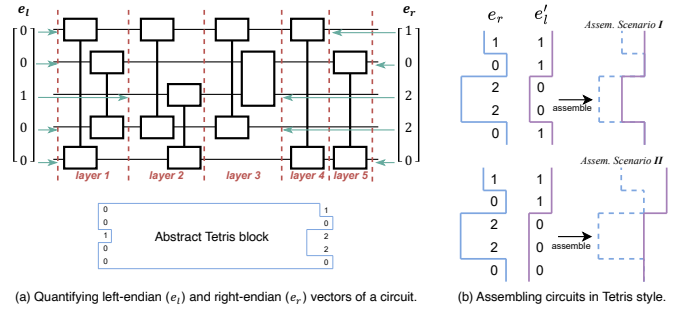


Fig. 3. Tetris-like circuits assembling. (a) Example of e_l and e_r vectors of a generic circuit. (b) In Scenario I, $cost_{depth} = \text{SUM}(e_r + e'_l)$; In Scenario II, $cost_{depth} = \text{SUM}(e_r + e'_l - 1)$.

each of them is Hermitian and equivalent to CNOT, and its corresponding BSF's tableau update rule can be defined as a combination of the update rules for H , S , and CNOT.

To determine which Clifford2Q operator to select from Eq. (4), we define a heuristic cost function

$$cost_{bsf} := n_{nonlocal}^2 * \left(\sum_{\langle i, j \rangle} \|r_x^{(i)} \vee r_z^{(i)} \vee r_x^{(j)} \vee r_z^{(j)}\| + \frac{1}{2} \sum_{\langle i, j \rangle} (\|r_x^{(i)} \vee r_x^{(j)}\| + \|r_z^{(i)} \vee r_z^{(j)}\|) \right) \quad (5)$$

to quantify the disparity of the current BSF from a desired BSF that requires no further simplification ($total_weight \leq 2$). $cost_{bsf}$ is the combined overlap of both X -part and Z -part among each pair of Pauli strings of a BSF, with a factor considering the impact of the number of nonlocal (weight larger than 1) Pauli strings.

Algorithm 1 illustrates details of the BSF simplification procedure. Its input is a set of Pauli strings. The output is a configuration sequence cfg , in which each component is either a Clifford2Q generator from Eq. (4) or a BSF with $total_weight$ no more than 2. Specifically, Algorithm 1 identifies the Clifford 2Q generator from Eq. (4) and qubit pair to act upon that leads to the greatest reduction in the cost function defined in Eq. (5), and the BSF is updated accordingly. This greedy process continues iteratively until the BSF's $total_weight$ is no more than 2. Before each Clifford2Q search epoch, local Pauli strings are peeled from the BSF, as they represent 1Q Pauli rotations and do not induce synthesis overhead. The output cfg represents a simplified IR group, while still in high-level semantics (Clifford2Q, 1Q Pauli-based IRs, and successive 2Q Pauli-based IRs), independent of any specific quantum ISA.

In theory, simplifying a BSF until $total_weight \leq 2$ is always achievable, for example, we can always reduce an individual Pauli string's weight to 1, peel it off, then iterate. The efficacy of Algorithm 1 lies in its simultaneous simplification mechanism guided by the carefully defined global cost function in Eq. (5).

C. Ordering IR groups in a Tetris-like style

With simplified IR groups compiled via BSF simplification, PHOENIX further performs a *Tetris-like IR group ordering* procedure that aims to minimize a *uniform assembling cost metric*. This is unlike [11] and [12] where the ordering is primarily determined by gate cancellation opportunities between IR groups. We can abstract the circuit structure exhibited by each simplified IR group into something resembling a Tetris block. Then, we choose the ordering based on the cost function of assembling Tetris blocks, with gate cancellation

opportunities and qubit routing overhead also taken into account. We primarily have three ingredients:

- 1) *Endian vectors of circuits and the depth cost function.* As shown in Fig. 3 (a), we define a pair of vectors e_l and e_r for each subcircuit (which would correspond to a simplified IR group in our case). The i -th entry of e_l (e_r) refers to how many layers one has to traverse starting from the left (right) side before qubit i is acted upon. The layers are defined by grouping neighboring 2Q gates that act on different qubits. See Fig. 3 (a) for an example for a generic (not necessarily simplified IR group) circuit. With this definition, the depth overhead for assembling two subcircuits – the preceding subcircuit with left-endian vectors e_r and the succeeding subcircuit with left-endian vector e'_l is given by

$$\text{cost}_{\text{depth}} := \begin{cases} \text{SUM}(e_r + e'_l), & \text{if } \text{ALL}(e_r[e'_l == 0] > 0) \\ & \text{and } \text{ALL}(e'_l[e_r == 0] > 0) \\ \text{SUM}(e_r + e'_l - 1), & \text{otherwise} \end{cases}$$

See Fig. 3 (b) for an example.

- 2) *Clifford2Q cancellation.* The simplified IR group usually exposes Hermitian Clifford2Q operators at the two ends of the subcircuit. Therefore some Clifford2Q gate cancellation opportunities could be exploited, which may or may not decrease overall circuit depth, as depicted in Fig. 4 (a). We can add this consideration into $\text{cost}_{\text{depth}}$: if m pairs Clifford2Q cancelled, while (a) without decreasing subcircuit depth, $\text{cost}_{\text{depth}} \leftarrow \text{cost}_{\text{depth}} - 2m$; (b) with one side's subcircuit depth decreases, $\text{cost}_{\text{depth}} \leftarrow \text{cost}_{\text{depth}} - 2m - n$; (c) with both sides' subcircuit depths decrease, $\text{cost}_{\text{depth}} \leftarrow \text{cost}_{\text{depth}} - 2m - 2n$.

- 3) *Consideration of qubit routing overhead.* We try to mitigate the routing overhead for hardware-aware compilation by comparing the qubit interaction graphs of subcircuits. This approach is not limited to specific routing schemes and hardware topologies. Intuitively, two subcircuits with more “similar” qubit interaction behaviors require less mapping transition overhead between them, as demonstrated by Fig. 4 (b). Therefore, we introduce a factor characterized by the similarity between the qubit interaction graphs of two subcircuits in the cost function:

$$\text{cost}_{\text{depth}} \rightarrow \frac{1}{s} \cdot \text{cost}_{\text{depth}}, \quad s = \sum_i \frac{\langle D_i, D'_i \rangle}{\|D_i\|_2 \|D'_i\|_2}, \quad (6)$$

where D (D') is the distance matrix of the preceding (succeeding) subcircuit's qubit interaction graph of the tail (head) part. The tail (head) is defined as starting from the right (left) of the subcircuit and incorporating more and more 2Q gates until all qubits are acted upon. $D_{i,j}$ represents the shortest path length between qubit i and qubit j . D_i is the i -th row of D .

In practice, these IR groups are first pre-arranged in descending order of their weight (subcircuit width). PHOENIX does lookahead for a certain number of subcircuits to find the one with the minimum $\text{cost}_{\text{depth}}$ with respect to the last assembled subcircuit. This process iterates until all subcircuits are assembled.

V. EVALUATION

We evaluate the effectiveness of PHOENIX across diverse Hamiltonian simulation programs, quantum ISAs, and device topologies. Although PHOENIX is implemented in Python, it compiles VQA programs of thousands of Pauli strings and more than ten qubits (approximately corresponding to the program size with 10^4 - 10^6 CNOT gates in conventional synthesis) in dozens of seconds. All experiments are executed on a laptop (Apple M3 Max, 36GB memory).

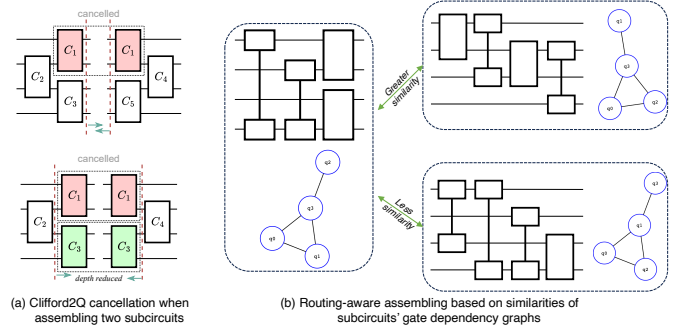


Fig. 4. Gate cancellation opportunities and routing-aware assembling. (a) Clifford2Q cancellation between the preceding and succeeding subcircuits may or may not induce circuit depth decrease. (b) The subcircuit (upper right) whose qubit interaction graph is more similar to that of the already assembled subcircuit (left) is preferred over the other (lower right).

TABLE I
UCCSD BENCHMARK SUITE.

Benchmark	#Qubit	#Pauli	w_{max}	#Gate	#CNOT	Depth	Depth-2Q
CH2_cmplt_BK	14	1488	10	37780	19574	23568	19399
CH2_cmplt_JW	14	1488	14	34280	21072	23700	19749
CH2_frz_BK	12	828	10	19880	10228	12559	10174
CH2_frz_JW	12	828	12	17658	10344	11914	9706
H2O_cmplt_BK	14	1000	10	25238	13108	15797	12976
H2O_cmplt_JW	14	1000	14	23210	14360	16264	13576
H2O_frz_BK	12	640	10	15624	8004	9691	7934
H2O_frz_JW	12	640	12	13704	8064	9332	7613
LiH_cmplt_BK	12	640	10	16762	8680	10509	8637
LiH_cmplt_JW	12	640	12	13700	8064	9342	7616
LiH_frz_BK	10	144	9	2890	1442	1868	1438
LiH_frz_JW	10	144	10	2850	1616	1985	1576
NH_cmplt_BK	12	640	10	15624	8004	9691	7934
NH_cmplt_JW	12	640	12	13704	8064	9332	7613
NH_frz_BK	10	360	9	8303	4178	5214	4160
NH_frz_JW	10	360	10	7046	3896	4640	3674

A. Experimental settings

Metrics. We evaluate PHOENIX using the following metrics: 2Q gate count, 2Q circuit depth, and the algorithmic error for VQA synthesis. Algorithmic error refers to the deviation between the synthesized circuit's unitary matrix and the ideal evolution under the original Hamiltonian, as measured by the infidelity between unitary matrices in our evaluation: $\text{infid} = 1 - \frac{1}{N} |\text{Tr}(U^\dagger V)|$. Notably, we exclude 1Q gates and their count in circuit depth, as 1Q gates are generally considered free resources due to their significantly lower error rates. Additionally, CNOT is not a native operation on most NISQ platforms, requiring extra 1Q drives before and/or after native 2Q gates (e.g., Cross-Resonance [17], Mølmer-Sørensen [18]), making 1Q gate inclusion in metrics potentially misleading.

Baselines. TKET [13], PAULIHEDRAL [11], and TETRIS [12] are primary baselines to be compared with our method. For TKET, the PauliSimp and FullPeepholeOptimise passes are adopted for logical circuit optimization. This is similar to TKET's O2 compilation pass in which the PauliSimp pass is particularly effective at optimizing Pauli gadgets. For PAULIHEDRAL, the QISKIT O2 pass is associated by default because the numerous gate cancellation opportunities exposed by PAULIHEDRAL necessitate inverse cancellation and commutative cancellation operations. For hardware-aware compilation, all baselines and PHOENIX are followed by a QISKIT O3 pass with SABRE qubit mapping [19]. For QAOA benchmarking, 2QAN [20] is used as the SOTA baseline.

Benchmarks. We select two representative VQA benchmarks—

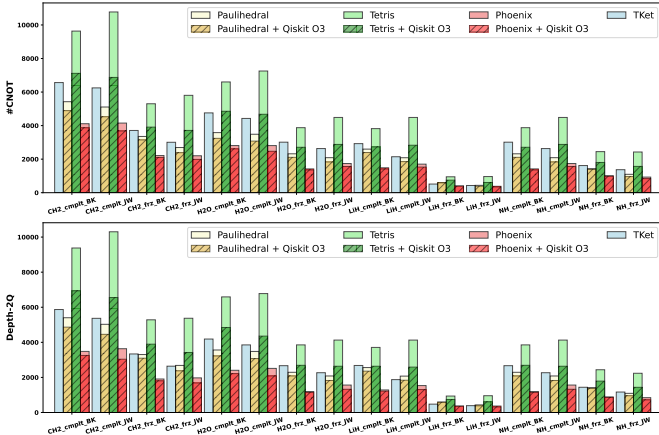


Fig. 5. Logical-level compilation (all-to-all topology).

TABLE II
AVERAGE (GEOMETRIC-MEAN) OPTIMIZATION RATES ON UCCSD.

Compiler	#CNOT opt.	Depth-2Q opt.
TKET	33.07%	30.14%
PAULIHEDRAL	28.41%	29.07%
PAULIHEDRAL + O3	25.72% (-8.54% v.s. no O3)	26.3% (-8.60% v.s. no O3)
TETRIS	53.66%	53.26%
TETRIS + O3	36.73% (-30.94% v.s. no O3)	36.37% (-31.08% v.s. no O3)
PHOENIX	21.15%	19.28%
PHOENIX + O3	19.56% (-6.58% v.s. no O3)	17.29% (-8.42% v.s. no O3)

(1) *UCCSD*: A set of UCCSD ansatzes, including CH₂, H₂O, LiH, and NH; 4 categories of molecule simulation programs. Each category is generated with STO-3G orbitals [21], encoded by Jordan-Wigner (JW) [22] and Bravyi-Kitaev (BK) [23] transformations, in turn approximated by complete or frozen-core orbitals. Details are shown in Tab. I. (2) *QAOA*: A set of 2-local Hamiltonian simulation programs corresponding to random graphs and regular graphs, for which the description and evaluation results are shown in Tab. IV.

B. Main results

Fig. 5 and Tab. II illustrates the main benchmarking results regarding logical program synthesis:

- 1) PHOENIX significantly outperforms baselines across all benchmarks, with an average (geometric mean) 21.15% and 17.29% optimization rate in #CNOT and Depth-2Q, respectively, relative to original circuits.² That is mostly attributed to the group-wise BSF simplification mechanism, as PHOENIX adopts the same IR grouping method as PAULIHEDRAL and TETRIS.
- 2) For the task of logical-level synthesis, TETRIS performs the worst, falling far behind TKET PAULIHEDRAL, and PHOENIX. This is because TETRIS focuses primarily on co-optimization techniques to reduce SWAP gates during qubit routing.
- 3) We also compare PAULIHEDRAL/TETRIS/PHOENIX with and without QISKIT optimization, to evaluate their proposed high-level optimization capabilities. The improvement from using QISKIT O3 for PAULIHEDRAL and TETRIS is more pronounced compared to that of PHOENIX. Therefore, PHOENIX’s high-

²For example, the #CNOT optimization rate is defined as $\frac{\#CNOT_{after}}{\#CNOT_{before}}$.

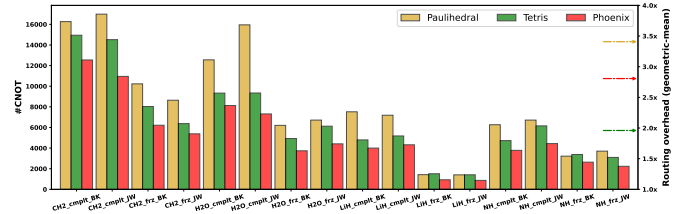


Fig. 6. Hardware-aware compilation on heavy-hex topology. Three dashed lines represent the average multiples of #CNOT within circuits after mapping relative to those after logical optimization, for PAULIHEDRAL (gold), TETRIS (green), and PHOENIX (coral), respectively.

level optimization strategy is more impressive and QISKIT O3 cannot improve much upon it.

C. Hardware-aware compilation

We use the heavy-hex topology, specifically a 64-qubit coupling graph from IBM’s Manhattan processor [24], as the limited-topology backend for hardware-aware compilation. Results are depicted in Fig. 6, in which TKET’s results are excluded due to significantly worse performance compared to PAULIHEDRAL/TETRIS/PHOENIX. Despite focusing primarily on high-level logical program optimization with limited hardware-aware co-optimization, PHOENIX still outperforms baselines, reducing CNOT gate count by 35.9% (22.3%) and 2Q circuit depth by 43.6% (27.8%) on average, compared to PAULIHEDRAL (TETRIS). With the qubit mapping transition overhead taken into account within its IR group ordering process, PHOENIX induces 2.8x #CNOT on average after hardware mapping. It is better than PAULIHEDRAL while worse than TETRIS, since TETRIS specializes in #CNOT cancellation for SWAP-based routing [12]. Consequently, even on limited-topology devices, PHOENIX effectively manages routing overhead and surpasses co-design local optimization strategies.

D. Comparison in diverse ISAs

Quantum ISA, or the native gate set, serves as an interface between software and hardware implementation. Quantum ISA design plays a crucial role in the performance of NISQ devices, in which the 2Q gate dominates the accuracy and difficulty of hardware implementation, as well as the theoretical circuit synthesis abilities. Although the conventional gate set includes 1Q gates and CNOT-based 2Q gates, recently some works propose integrating complex and even continuous 2Q gates into the ISA design, such as the XY(θ) gate family [25], the fractional gates [26], and the AshN gate scheme [14] which considers all possible 2Q gates within the SU(4) group as the quantum ISA. Therefore, we further compare PHOENIX with baselines in different quantum ISAs to showcase its ISA-independent compilation advantage.

This ISA-independent advantage is best shown by choosing the most expressive SU(4) ISA. We evaluate it with both all-to-all and heavy-hex topologies, as the same as the evaluation above for CNOT ISA. For logical-level compilation, PHOENIX directly generates SU(4)-based circuits via its BSF simplification algorithm, while baselines (PAULIHEDRAL and TETRIS equipped with QISKIT O3 by default) require an additional “rebase” (or “transpile”) step to convert CNOT-based circuits to SU(4)-based ones. For hardware-aware compilation, all compilers include a rebase step, following the QISKIT O3 hardware-aware compilation pass. Detailed outcomes are summarized in Tab. III, highlighting the geometric-mean relative optimization rates of PHOENIX compared to the baselines’ results.

TABLE III
COMPARISON FOR DIVERSE ISAS WITH ALL-TO-ALL AND LIMITED TOPOLOGY.

	CNOT ISA (all-to-all)		SU(4) ISA (all-to-all)		CNOT ISA (heavy-hex)		SU(4) ISA (heavy-hex)	
PHOENIX's opt. rate	#CNOT	Depth-2Q	#SU(4)	Depth-2Q	#CNOT	Depth-2Q	#SU(4)	Depth-2Q
PHOENIX v.s. TKET	63.93%	63.96%	56.12%	54.21%	40.79%	48.51%	44.44%	50.65%
PHOENIX v.s. PAULIHEDRAL	82.21%	73.29%	75.69%	65.18%	62.63%	54.92%	39.97%	35.02%
PHOENIX v.s. TETRIS	57.58%	53.01%	56.63%	50.54%	76.27%	71.47%	62.44%	58.66%

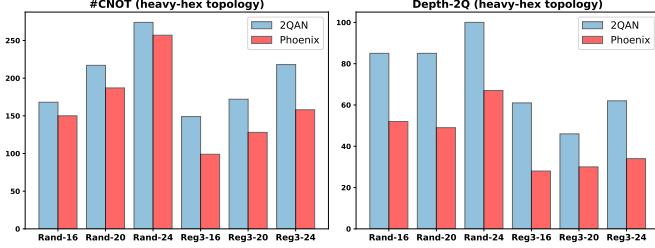


Fig. 7. QAOA benchmarking

TABLE IV
QAOA BENCHMARKING VERSUS 2QAN.

QAOA		#CNOT		Depth-2Q		#SWAP		Routing overhead	
Bench.	#Pauli	2QAN	Phoenix	2QAN	Phoenix	2QAN	Phoenix	2QAN	Phoenix
Rand-16	32	168	150	85	52	37	29	2.62x	2.34x
Rand-20	40	217	187	85	49	47	39	2.71x	2.34x
Rand-24	48	274	257	100	67	63	56	2.85x	2.68x
Reg3-16	24	149	99	61	28	44	17	3.10x	2.06x
Reg3-20	30	172	128	46	30	46	23	2.87x	2.13x
Reg3-24	36	218	158	62	34	62	30	3.03x	2.19x
Avg. improv.		-16.7%		-40.8%		-29.41%		-16.59%	

Again, PHOENIX significantly outperforms baselines when targeting SU(4) ISA. The optimization rates relative to baselines are more impressive than those in CNOT ISA, despite the baselines incorporating sophisticated optimization techniques specifically designed for the CNOT ISA. For instance, the multiple of PHOENIX's #2Q relative to PAULIHEDRAL's in CNOT ISA is 82.21% (62.63%), whereas this value decreases to 75.69% (39.97%) in SU(4) ISA for hardware-agnostic (hardware-aware) compilation. One exception is the hardware-aware compilation comparison with TKET, as TKET's hardware-aware compilation in the CNOT ISA generates circuits with much larger 2Q gate count and circuit depth than other compilers, and there are numerous 2Q subcircuit fusing opportunities such that the rebased circuits involves much fewer SU(4) gates.

E. QAOA benchmarking

For QAOA benchmarking, we focus on the performance in hardware-aware compilation, since the 2Q gate count cannot be reduced and minimizing circuit depth is easy in logical-level compilation. Both 2QAN and PHOENIX can generate depth-optimal QAOA circuits at the logical level in our field test. Fig. 7 and Tab. IV illustrate compilation results on the heavy-hex topology, across six QAOA programs corresponding to both random graphs (each node with degree 4) and regular (each node with degree 3) graphs, with qubit sizes of 16, 20, and 24. PHOENIX outperforms 2QAN across all benchmarks in all metrics (such as #CNOT, #SWAP), especially in Depth-2Q, with an average 40.8% reduction compared to 2QAN. These results further demonstrate the effectiveness of the routing-aware IR group ordering method in PHOENIX.

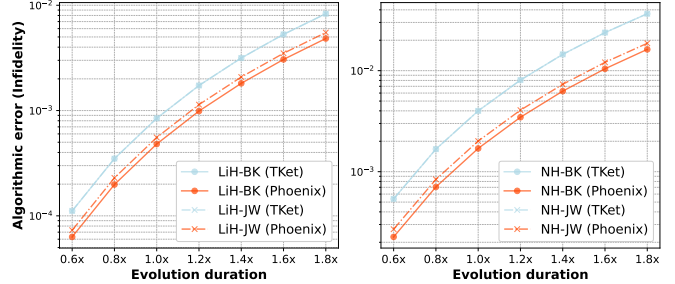


Fig. 8. Algorithmic error comparison of LiH and NH simulation.

F. Algorithmic error analysis

We further highlight PHOENIX's advantage in reducing the algorithmic error. We select UCCSD benchmarks with qubits no more than 10 for evaluation, within the matrix computation capabilities of standard PCs. We rescale the coefficients of Pauli strings to control their algorithmic errors within 5×10^{-5} to 10^{-2} , which corresponds to different evolution durations in molecular simulation, as suggested in Fig. 8. In contrast to TKET, PHOENIX typically leads to lower algorithmic errors for both JW and BK encoding schemes. Although this improvement is program-specific, it is more significant for the Pauli string patterns of BK than those of JW, with 57% (42.7%) and 49.5% (34.1%) for NH (LiH) simulation, respectively. As they adopt the same Pauli string blocking approach, we expect the algorithmic errors resulting from PAULIHEDRAL and TETRIS to be comparable to PHOENIX, and so are not shown in Fig. 8. As a result, the impressive algorithmic error reduction effect of PHOENIX brings us closer to a possible quantum advantage on computational chemistry problems.

VI. CONCLUSION AND OUTLOOK

Mainstream circuit synthesis approaches rely on rewriting rules or patterns, often restricted to small-scale, local optimizations. In contrast, we present PHOENIX, a framework that leverages a high-level Pauli-based IR for Hamiltonian simulation, one of the most prominent and widely studied applications in quantum computing. PHOENIX outperforms all SOTA VQA compilers across diverse program types and target hardware platforms, showcasing its unmatched performance and versatility. This work not only bridges the gap between impactful quantum applications and physically implementable solutions but also prompts a re-evaluation of quantum compiler pipeline design.

Beyond producing significantly optimized circuits, high-level IRs, as a refined abstraction layer on top of the traditional quantum ISA, create more opportunities for efficient synthesis and effective hardware mapping. Furthermore, they can act as intermediate building blocks, guiding the design of quantum processors and corresponding control schemes—potentially incorporating multi-qubit control—to implement these high-level IRs more efficiently.

REFERENCES

- [1] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 1994, pp. 124–134.
- [2] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for linear systems of equations," *Physical review letters*, vol. 103, no. 15, p. 150502, 2009.
- [3] S. Lloyd, "Universal quantum simulators," *Science*, vol. 273, no. 5278, pp. 1073–1078, 1996.
- [4] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018.
- [5] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'Brien, "A variational eigenvalue solver on a photonic quantum processor," *Nature communications*, vol. 5, no. 1, p. 4213, 2014.
- [6] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014.
- [7] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, and P. J. Coles, "Variational quantum algorithms," *Nature Reviews Physics*, vol. 3, no. 9, pp. 625–644, 2021.
- [8] A. Cowtan, S. Dilkes, R. Duncan, W. Simmons, and S. Sivarajah, "Phase gadget synthesis for shallow circuits," *arXiv preprint arXiv:1906.01734*, 2019.
- [9] A. M. van de Griend, "Towards a generic compilation approach for quantum circuits through resynthesis," *arXiv preprint arXiv:2304.08814*, 2023.
- [10] J. Paykin, A. T. Schmitz, M. Ibrahim, X.-C. Wu, and A. Y. Matsuura, "Pcoast: a pauli-based quantum circuit optimization framework," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 1. IEEE, 2023, pp. 715–726.
- [11] G. Li, A. Wu, Y. Shi, A. Javadi-Abhari, Y. Ding, and Y. Xie, "Pauli-hedral: a generalized block-wise compiler optimization framework for quantum simulation kernels," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 554–569.
- [12] Y. Jin, Z. Li, F. Hua, T. Hao, H. Zhou, Y. Huang, and E. Z. Zhang, "Tetris: A compilation framework for vqa applications in quantum computing," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 277–292.
- [13] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, "*t[ket]*: a retargetable compiler for nisq devices," *Quantum Science and Technology*, vol. 6, no. 1, p. 014003, 2020.
- [14] J. Chen, D. Ding, W. Gong, C. Huang, and Q. Ye, "One gate scheme to rule them all: Introducing a complex yet reduced instruction set for quantum computing," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. La Jolla, CA, USA: ACM, 2024, pp. 779–796.
- [15] E. Van Den Berg and K. Temme, "Circuit optimization of hamiltonian simulation by simultaneous diagonalization of pauli clusters," *Quantum*, vol. 4, p. 322, 2020.
- [16] D. Grier and L. Schaeffer, "The classification of clifford gates over qubits," *Quantum*, vol. 6, p. 734, 2022.
- [17] C. Rigetti and M. Devoret, "Fully microwave-tunable universal gates in superconducting qubits with linear couplings and fixed transition frequencies," *Physical Review B—Condensed Matter and Materials Physics*, vol. 81, no. 13, p. 134507, 2010.
- [18] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, "Trapped-ion quantum computing: Progress and challenges," *Applied Physics Reviews*, vol. 6, no. 2, 2019.
- [19] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*, 2019, pp. 1001–1014.
- [20] L. Lao and D. E. Browne, "2qan: A quantum compiler for 2-local qubit hamiltonian simulation algorithms," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 351–365.
- [21] W. J. Hehre, R. F. Stewart, and J. A. Pople, "Self-consistent molecular-orbital methods. i. use of gaussian expansions of slater-type atomic orbitals," *The Journal of Chemical Physics*, vol. 51, no. 6, pp. 2657–2664, 1969.
- [22] P. Jordan and E. Wigner, "über das paulische äquivalenzverbot. z phys 47: 631," 1928.
- [23] S. B. Bravyi and A. Y. Kitaev, "Fermionic quantum computation," *Annals of Physics*, vol. 298, no. 1, pp. 210–226, 2002.
- [24] G. J. Mooney, G. A. White, C. D. Hill, and L. C. Hollenberg, "Whole-device entanglement in a 65-qubit superconducting quantum computer," *Advanced Quantum Technologies*, vol. 4, no. 10, p. 2100061, 2021.
- [25] D. M. Abrams, N. Didier, B. R. Johnson, M. P. d. Silva, and C. A. Ryan, "Implementation of xy entangling gates with a single calibrated pulse," *Nature Electronics*, vol. 3, no. 12, pp. 744–750, 2020.
- [26] I. Quantum. (2024) New fractional gates reduce circuit depth for utility-scale workloads. Accessed: Nov. 18, 2024. [Online]. Available: <https://www.ibm.com/quantum/blog/fractional-gates>