RUTGERS

NC STATE UNIVERSITY

Strategies and tools to compile hybrid CV-DV quantum circuits

Zihan Chen

Apr 29, 2026

zihan.chen.cs@rutgers.edu

Google slides copy: https://docs.google.com/presentation/d/1YGaR569vz1727BzR713kx_yb7-eVKFufAsbzVGd5trk/edit?slide=id.g3362633b84c_0_168#slide=id.g3362633b84c_0_168

ASPLOS Tutorial

Hybrid Oscillator-Qubit Quantum Processors — Instruction Set Architecture, Abstract Machine Models, and Applications

ASPLOS 2026 Tutorial. Pittsburgh, PA.

Time: 8 AM – Noon, March 22, 2026. **Room**: Ft. Duquesne

- Slides and resources: https://cvdv.ncsu.edu/resources/asplos-tutorial/
- Featured on ACM SIGARCH Computer Architecture Today: https://www.sigarch.org/beyond-qubits-a-systems-view-of-hybrid-cv-dv-quantum-computing/

Foundations

Yuan Liu

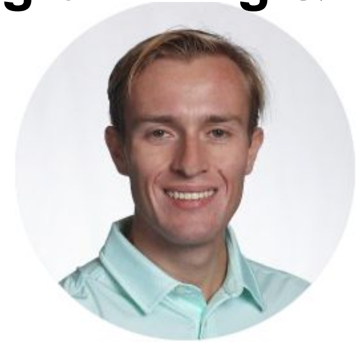
Benchmarking

Shubdeep Mohapatra

Compilation

Zihan Chen

Programming & Demo



Jim Furches

- Slides and resources: https://cvdv.ncsu.edu/resources/asplos-tutorial/
- Featured on ACM SIGARCH Computer Architecture Today: https://www.sigarch.org/beyond-qubits-a-systems-view-of-hybrid-cv-dv-quantum-computing/

Foundations

Yuan Liu

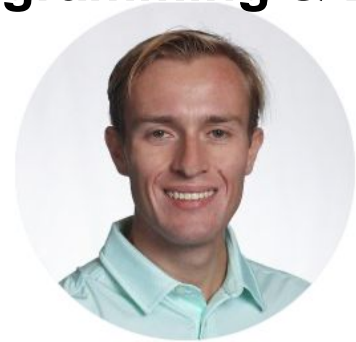
Compilation: Strategies and tools to compile hybrid CV-DV quantum circuits

Zihan Chen

Benchmarking

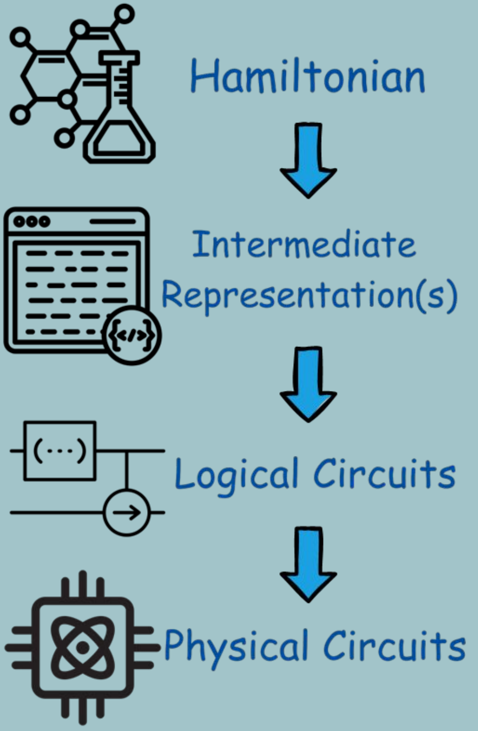
Shubdeep Mohapatra

Programming & Demo



Jim Furches

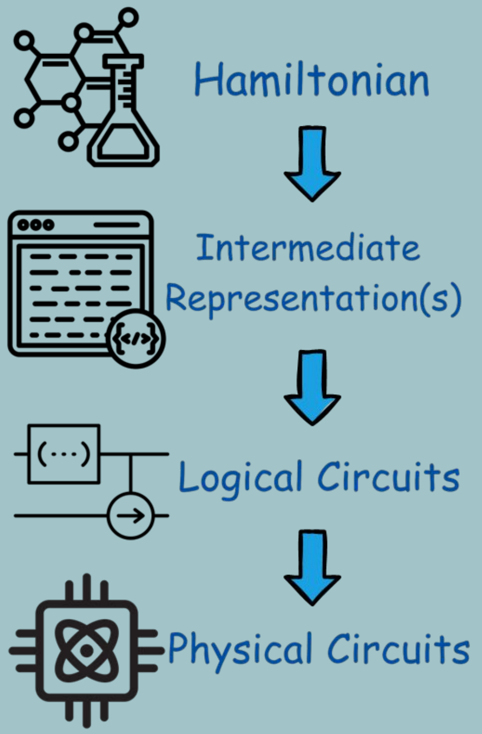
Compilation



RUTGERS NC STATE UNIVERSITY

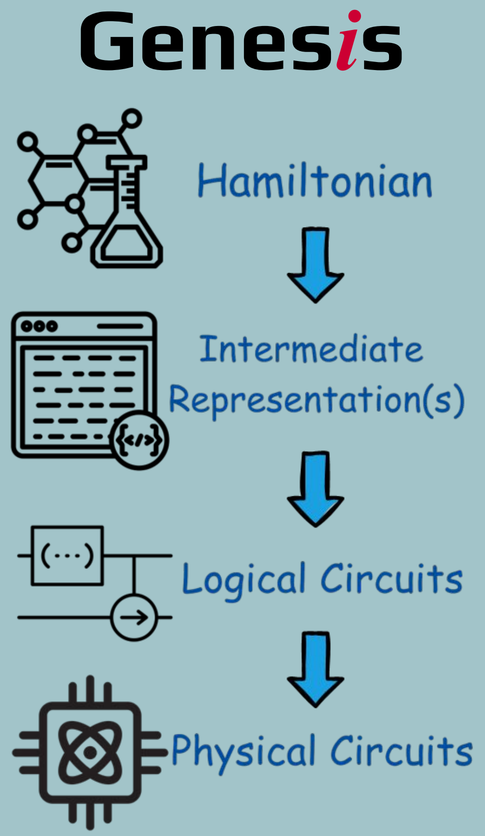
1. Symbolic Compilation for CV-DV Hamiltonians: Current Approaches
2. Challenges and Opportunities Beyond Today's Compilers
3. An End-to-End Compilation Walkthrough with Genesis

Compilation



RUTGERS NC STATE UNIVERSITY

1. Symbolic Compilation for CV-DV Hamiltonians: Current Approaches
2. Challenges and Opportunities Beyond Today’s Compilers
3. An End-to-End Compilation Walkthrough with Genesis

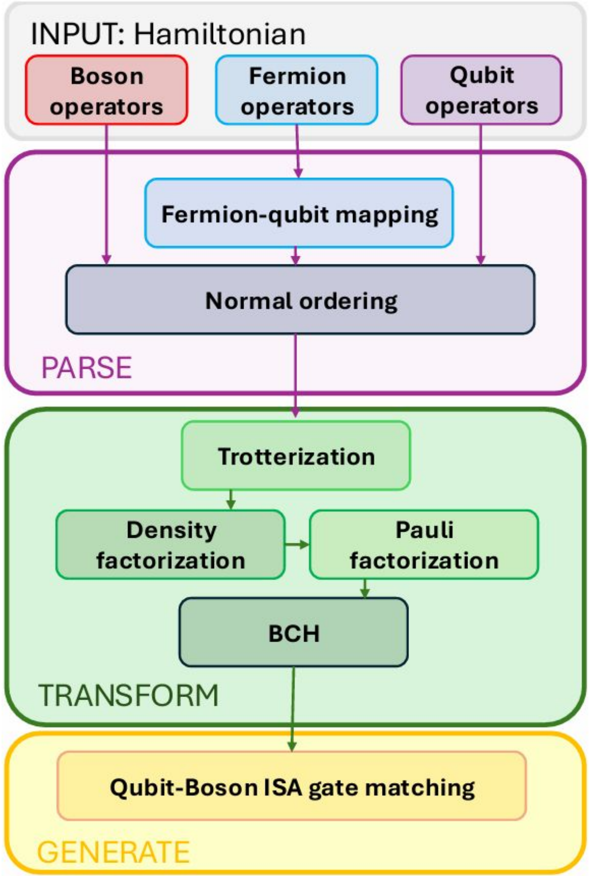


RUTGERS NC STATE UNIVERSITY

Genesis: A Compiler for Hamiltonian Simulation on Hybrid CV-DV Quantum Computers

> Chen, Zihan, Jiakang Li, Minghao Guo, Henry Chen, Zirui Li, Joel Bierman, Yipeng Huang, Huiyang Zhou, Yuan Liu, and Eddy Z. Zhang. "Genesis: A Compiler for Hamiltonian Simulation on Hybrid CV-DV Quantum Computers." In Proceedings of the 52nd Annual International Symposium on Computer Architecture, pp. 1583-1597. 2025.

> arxiv:2505.13683



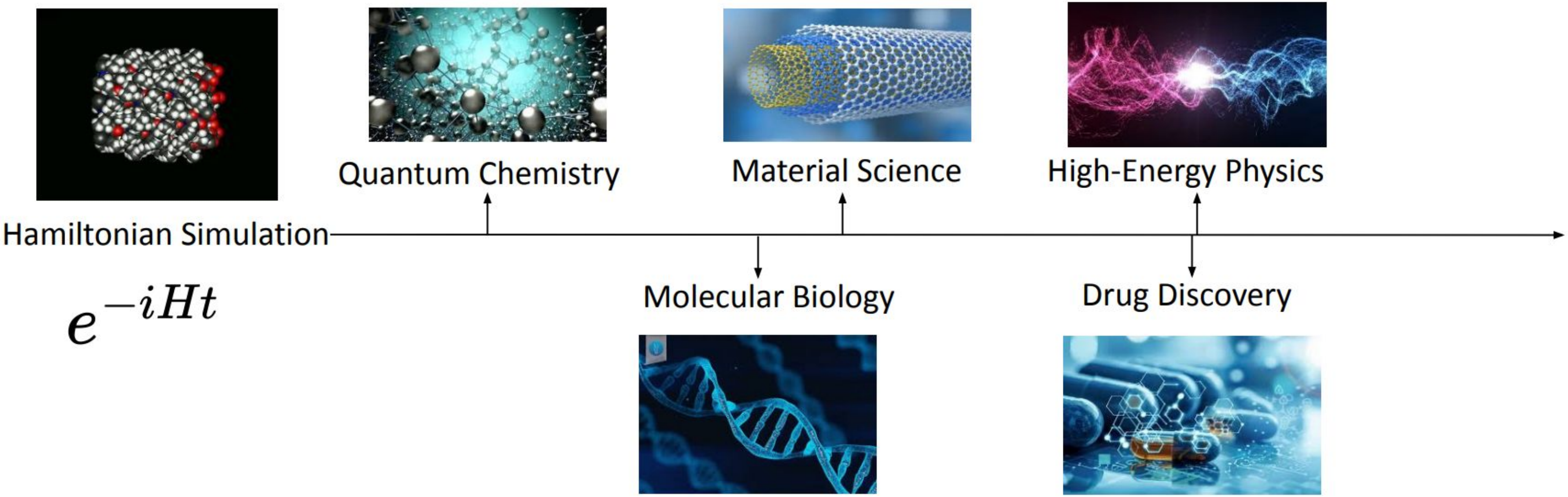
Symbolic Hamiltonian Compiler for Hybrid Qubit-Boson Processors

> Decker, Ethan, Erik Gustafson, Evan McKinney, Alex K. Jones, Lucas Goetz, Ang Li, Alexander Schuckert, Samuel Stein, Gushu Li, and Eleanor Crane. "Symbolic Hamiltonian Compiler for Hybrid Qubit-Boson Processors." In 2025 IEEE International Conference on Quantum Computing and Engineering (QCE), vol. 1, pp. 540-548. IEEE, 2025.

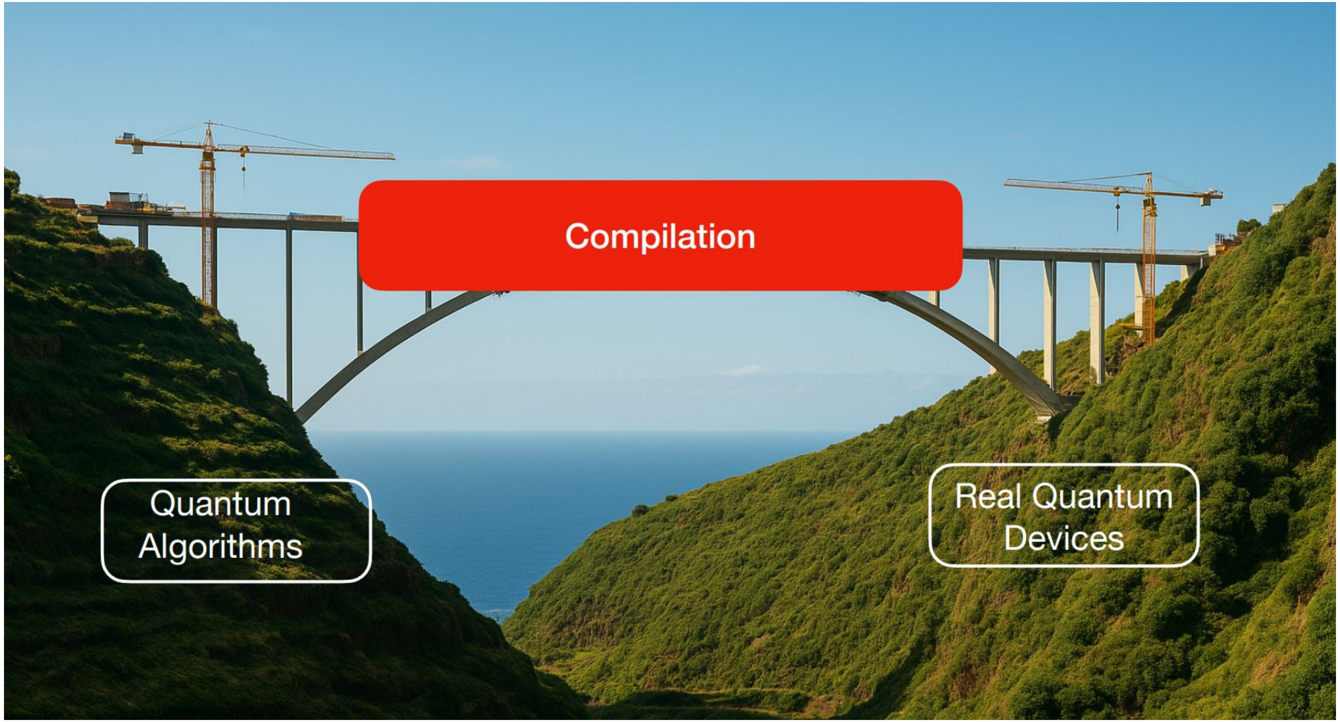
> arXiv:2506.00215

1.1 Why Hamiltonian Simulation matters?

- Hybrid CV-DV systems can **natively** represent bosonic degrees of freedom while retaining **qubit-based** control and interaction structure



1.1 Why Hamiltonian Simulation matters?



1.2 Operator representations

- Continuous-variable (CV) quantum systems encode information in observables with continuous spectra, in contrast to discrete-variable (DV) qubits. A single CV mode (qumode) is modeled as a quantum harmonic oscillator with Hamiltonian:

$$\hat{H} = \frac{1}{2}(\hat{p}^2 + \hat{x}^2) = \hat{a}^\dagger \hat{a} + \frac{1}{2}$$

Source: **"HyQBench: A Benchmark Suite for Hybrid CV-DV Quantum Computing."** Mohapatra et al. arXiv:2603.04398

1.2 Operator representations

- Fock basis:
  - Creation operator

$$\hat{a}^\dagger$$

  - Annihilation operator

$$\hat{a}$$

  - Number operator

$$\hat{n} = \hat{a}^\dagger \hat{a}$$

Current compilers operate on ladder operators (**Fock basis**), because many Hamiltonians are naturally expressed in second-quantized form.

- Position basis:
  - Position operator

$$\hat{x}$$

  - Momentum operator

$$\hat{p}$$

However, recent simulation work* also show the potential of position basis encoding, and offer **alternative compilation pathways**.

* **"Efficient Qubit Simulation of Hybrid Oscillator-Qubit Quantum Computation."** Lu et al. arXiv:2603.09233

1.3 Why matrix-free symbolic compilation?

Hamiltonian example:

$$H = -t\sum_{i,j,\sigma} c^{\dagger}_{i\sigma}c_{j\sigma} + U\sum_{i} b^{\dagger}_{i}b_{i} + g\sum_{i,\sigma}\hat{n}_{i\sigma}(b^{\dagger}_{i} + b_{i})$$

Corresponds Hamiltonian Grammar intermediate representation:

```
Const t = 1;
Const U = 1;
Const g = 1;

Range i = [0, 10, 1];
Range j = [0, 10, 1];
Range sigma = [0, 2, 1];

Result = - t *Sum_over(i, j, sigma){FC[i][sigma]* FA[j][sigma]}
         + U *Sum_over(i){BC[i]* BA[i]}
         + g * Sum_over(i, sigma){TensorProd(FN[i][sigma], BC[i] + BA[i])};
```

RUTGERS NC STATE UNIVERSITY

1.3 Why matrix-free symbolic compilation?

- Limitations of Unitary Synthesis in Hybrid CV-DV Systems
  - CV systems are inherently infinite-dimensional.

$$\dim(\mathcal{H}) = \infty$$

With n qubits and m qumodes under Fock cutoffs $f_1, \ldots, f_m$, the hybrid Hilbert space scales as

$$\dim(\mathcal{H}) = 2^n \prod_{i=1}^{m} f_i \quad (\approx 2^n f^m \text{ for uniform } f),$$

  - High cutoff → better accuracy, but poor scalability
  - Low cutoff → efficient, but large approximation error

Source: **"Hybrid continuous-discrete-variable quantum computing: a guide to utility."** Kemper et al. arXiv:2511.13882

1.3 Why matrix-free symbolic compilation?

- Hamiltonians are naturally expressed in algebraic operator form

$$H = \sum_i c_i O_i$$

  - Enables:
    - **Symbolic rewriting**
    - **Rule-based decomposition**
    - **Structure-preserving transformations**

1.4 How symbolic compilation works

- Direct Hybrid CVDV Gate Synthesis

Type	Gate Name	Definition
Qubit	x, y Rotation	$r_\varphi(\theta) = \exp\left[-i\frac{\theta}{2}(\cos\varphi\sigma_x + \sin\varphi\sigma_y)\right]$
	z Rotation	$r_z(\theta) = \exp\left(-i\frac{\theta}{2}\sigma_z\right)$
Qumode	Phase-Space Rotation	$\mathrm{R}(\theta) = \exp\left[-i\theta a^\dagger a\right]$
	Displacement	$\mathrm{D}(\alpha) = \exp\left[\left(\alpha a^\dagger - \alpha^* a\right)\right]$
	Beam-Splitter	$\mathrm{BS}(\theta, \varphi) = \exp\left[-i\frac{\theta}{2}\left(e^{i\varphi}a^\dagger b + e^{-i\varphi}ab^\dagger\right)\right]$
Hybrid	Conditional Phase-Space Rotation	$\mathrm{CR}(\theta) = \exp\left[-i\frac{\theta}{2}\sigma_z a^\dagger a\right]$
	Conditional Parity	$\mathrm{CP} = \exp\left[-i\frac{\pi}{2}\sigma_z a^\dagger a\right]$
	Conditional Displacement	$\mathrm{CD}(\alpha) = \exp\left[\sigma_z\left(\alpha a^\dagger - \alpha^* a\right)\right]$
	Conditional Beam-Splitter	$\mathrm{CBS}(\theta, \varphi) = \exp\left[-i\frac{\theta}{2}\sigma_z\left(e^{i\varphi}a^\dagger b + e^{-i\varphi}ab^\dagger\right)\right]$
	Rabi Interaction	$\mathrm{RB}(\theta) = \exp\left[-i\sigma_x\left(\theta a^\dagger - \theta^* a\right)\right]$

More resources: **“Hybrid Oscillator-Qubit Quantum Processors: Instruction Set Architectures, Abstract Machine Models, and Applications.”** Liu et al. arXiv:2407.10381; **"Hybridlane: A Software Development Kit for Hybrid Continuous-Discrete Variable Quantum Computing."** Furches et al. arXiv:2603.10919

RUTGERS NC STATE UNIVERSITY

1.4 How symbolic compilation works

- Product formula will be an important building block
  - Trotterization(Trotter-Suzuki formula), error = *O(t^2/k)*

$$e^{-i(M+N)t} \approx \left(e^{-iMt/k} e^{-iNt/k}\right)^k,$$

  - BCH(Baker Campbell Hausdorff formula), error = *O(t^3)*

$$e^{t^2[M,N]} \approx e^{tM} e^{tN} e^{-tM} e^{-tN}.$$

1.4 How symbolic compilation works

- Use block encoding to handle complex operators
  - Block encoding allow us embed an operator A using a larger unitary:

$$U = \begin{bmatrix} A/\alpha & * \\ * & * \end{bmatrix}$$

  - Combine with product formula, it can help us decompose some complex terms to more fine-grained version:

$$[i\tau\mathcal{B}_{-iA}, i\tau\mathcal{B}_{B^\dagger}] = i\tau^2 \begin{bmatrix} 2AB & 0 \\ 0 & -BA-(BA)^\dagger \end{bmatrix}$$

  - Recent work also provide a key primitive for bosonic operator construction:

$$\mathcal{S}_1 = \exp\left(it\begin{bmatrix} 0 & a^\dagger \\ a & 0 \end{bmatrix}\right) \quad \mathcal{S}_1 \approx \mathrm{e}^{\mathrm{i}(\pi/2)a^\dagger a}\mathrm{e}^{\mathrm{i}\left(\alpha\left(a^\dagger+a\right)\right)\otimes\sigma^y}\mathrm{e}^{-\mathrm{i}(\pi/2)a^\dagger a}\mathrm{e}^{\mathrm{i}\left(\alpha\left(a^\dagger+a\right)\right)\otimes\sigma^x}$$

Source: **"Leveraging Hamiltonian Simulation Techniques to Compile Operations on Bosonic Devices."** Kang, Christopher, et al. arXiv:2303.15542

1.4 How symbolic compilation works

- Decomposition Rules Set

Rules	Operator Template	Conditions	Decomposition Output	Reference	Precision
1	$\exp(Mt+Nt) \approx \text{Trotter}(Mt, Nt)$		$(\exp(Mt/k)\exp(Nt/k))^k$	Trotterization	Approx
2	$\exp([Mt, Nt]) \approx \text{BCH}(Mt, Nt)$		$\exp(Mt)\exp(Nt)\exp(-Mt)\exp(-Nt)$	BCH	Approx
3	$\exp(t^2[M,N])$	M, N Hermitian	$\exp([it\sigma_i N, it\sigma_i M])$	[20]	Exact
4	$\exp(-it^2\sigma_i\{M,N\})$	M, N Hermitian	$\exp([it\sigma_j M, it\sigma_k N])$	[20]	Exact
5	$\exp(-it^2\sigma_z[M,N])$		$\exp([(itN, it\sigma_z M])$	This paper	Exact
6	$\exp(t^2\sigma_z((MN-(MN)^\dagger)))$	$[M,N]=0$	$\exp([X \cdot it\mathcal{B}_N \cdot X, it\mathcal{B}_M])$	[20]	Exact
7	$\exp(it^2\sigma_z((MN+(MN)^\dagger)))$	$[M,N]=0$	$\exp([S \cdot it\mathcal{B}_M \cdot S^\dagger, X \cdot it\mathcal{B}_N \cdot X])$	[20]	Exact
8	$\exp\left(-2it\begin{pmatrix} MN & 0 \\ 0 & -MN \end{pmatrix}\right)$	M, N Hermitian	$\exp(-it\sigma_z[M,N] - it\sigma_z\{M,N\})$	This paper	Exact
9	$\exp\left(2it^2\begin{pmatrix} MN & 0 \\ 0 & -MN \end{pmatrix}\right)$	$[M,N]=0$ $MN=(MN)^\dagger$	$\exp([(S \cdot it\mathcal{B}_M \cdot S^\dagger, X \cdot it\mathcal{B}_N \cdot X])$	[20]	Exact
10	$\exp(2it\mathcal{B}_{MN})$	$[M,N]=0$	$X \cdot \exp(t\sigma_y(MN-(MN)^\dagger) + it\sigma_x(MN+(MN)^\dagger)) \cdot X$	[20]	Exact
11	$\exp\left(it\begin{pmatrix} 2MN & 0 \\ 0 & -NM-(NM)^\dagger \end{pmatrix}\right)$	$MN=(MN)^\dagger$	$\exp([S \cdot it\mathcal{B}_M \cdot S^\dagger, X \cdot it\mathcal{B}_N \cdot X])$	[20]	Exact
12	$\mathcal{B}_a = \exp\left(2i\alpha\begin{pmatrix} 0 & a \\ a^\dagger & 0 \end{pmatrix}\right)$	$\alpha=\alpha^*$	$\exp(i(\pi/2)a^\dagger a)\exp(i(\alpha(a^\dagger+a))\otimes\sigma_y)\exp(-i(\pi/2)a^\dagger a)\exp(i(\alpha(a^\dagger+a))\otimes\sigma_x)$	[20]	Approx
13	$\mathcal{B}_{a^\dagger} = \exp\left(2i\alpha\begin{pmatrix} 0 & a^\dagger \\ a & 0 \end{pmatrix}\right)$	$\alpha=\alpha^*$	$\exp(i(\pi/2)a^\dagger a)\exp(i(\alpha(a^\dagger+a))\otimes\sigma_y)\exp(-i(\pi/2)a^\dagger a)\exp(-i(\alpha(a^\dagger+a))\otimes\sigma_x)$	This paper	Approx
14	$e^{(P_1P_2\cdots P_n)(\alpha a_k^\dagger - \alpha^* a_k)}$		Multi-qubit-controlled displacement: Right hand side (RHS) of Equation (11) first line	[28]	Exact
15	$e^{2i\alpha^2 P_1P_2\cdots P_n}$		Multi-Pauli Exponential: Right hand side (RHS) of Equation (9) first line	This Paper	Exact
16	All Native Gates RHS in Table 2		All Native Gates Left Hand Side (LHS) Table 2	[28]	Exact

RUTGERS NC STATE UNIVERSITY

[20] **"Leveraging Hamiltonian Simulation Techniques to Compile Operations on Bosonic Devices."** Kang, Christopher, et al. arXiv:2303.15542

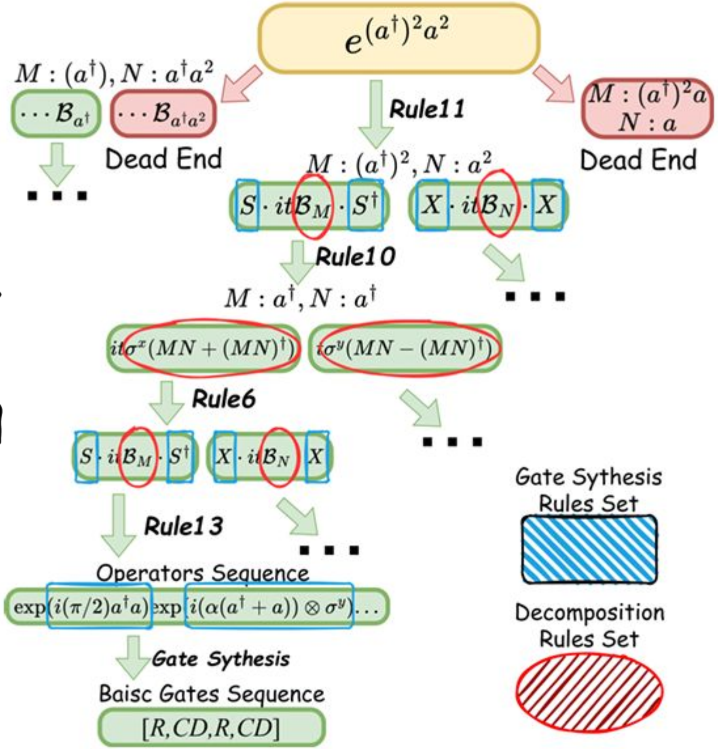
1.4 How symbolic compilation works:
Rule-Based Recursive Template Matching

Basic Gates Set

Type	Gate Name	Definition
Qubit	x, y Rotation	$r_\varphi(\theta) = \exp\left[-i\frac{\theta}{2}(\cos\varphi\sigma_x + \sin\varphi\sigma_y)\right]$
	z Rotation	$r_z(\theta) = \exp\left(-i\frac{\theta}{2}\sigma_z\right)$
Qumode	Phase-Space Rotation	$\mathrm{R}(\theta) = \exp\left[-i\theta a^\dagger a\right]$
	Displacement	$\mathrm{D}(\alpha) = \exp\left[\left(\alpha a^\dagger - \alpha^* a\right)\right]$
	Beam-Splitter	$\mathrm{BS}(\theta,\varphi) = \exp\left[-i\frac{\theta}{2}\left(e^{i\varphi}a^\dagger b + e^{-i\varphi}ab^\dagger\right)\right]$
Hybrid	Conditional Phase-Space Rotation	$\mathrm{CR}(\theta) = \exp\left[-i\frac{\theta}{2}\sigma_z a^\dagger a\right]$
	Conditional Parity	$\mathrm{CP} = \exp\left[-i\frac{\pi}{2}\sigma_z a^\dagger a\right]$
	Conditional Displacement	$\mathrm{CD}(\alpha) = \exp\left[\sigma_z\left(\alpha a^\dagger - \alpha^* a\right)\right]$
	Conditional Beam-Splitter	$\mathrm{CBS}(\theta,\varphi) = \exp\left[-i\frac{\theta}{2}\sigma_z\left(e^{i\varphi}a^\dagger b + e^{-i\varphi}ab^\dagger\right)\right]$
	Rabi Interaction	$\mathrm{RB}(\theta) = \exp\left[-i\sigma_x\left(\theta a^\dagger - \theta^* a\right)\right]$

Decomposition Rules Set

Rules	Operator Template	Conditions	Decomposition Output	Reference	Precision
1	$\exp(Mt+Nt) \approx \mathrm{Trotter}(Mt,Nt)$		$(\exp(Mt/k)\exp(Nt/k))^k$	Trotterization	Approx
2	$\exp([Mt,Nt]) \approx \mathrm{BCH}(Mt,Nt)$		$\exp(Mt)\exp(Nt)\exp(-Mt)\exp(-Nt)$	BCH	Approx
3	$\exp(t^2[M,N])$	M,N Hermitian	$\exp([it\sigma_i N, it\sigma_i M])$	[20]	Exact
4	$\exp(-it^2\sigma_i\{M,N\})$	M,N Hermitian	$\exp([it\sigma_j M, it\sigma_k N])$	[20]	Exact
5	$\exp(-it^2\sigma_z[M,N])$		$\exp([(itN, it\sigma_z M])$	This paper	Exact
6	$\exp(t^2\sigma_z((MN-(MN)^\dagger)))$	$[M,N]=0$	$\exp([X\cdot it\mathcal{B}_N\cdot X, it\mathcal{B}_M])$	[20]	Exact
7	$\exp(it^2\sigma_z((MN+(MN)^\dagger)))$	$[M,N]=0$	$\exp([S\cdot it\mathcal{B}_M\cdot S^\dagger, X\cdot it\mathcal{B}_N\cdot X])$	[20]	Exact
8	$\exp\left(-2it\begin{pmatrix}MN & 0\\ 0 & -MN\end{pmatrix}\right)$	M,N Hermitian	$\exp(-it\sigma_z[M,N]-it\sigma_z\{M,N\})$	This paper	Exact
9	$\exp\left(2it^2\begin{pmatrix}MN & 0\\ 0 & -MN\end{pmatrix}\right)$	$[M,N]=0$ $MN=(MN)^\dagger$	$\exp([(S\cdot it\mathcal{B}_M\cdot S^\dagger, X\cdot it\mathcal{B}_N\cdot X])$	[20]	Exact
10	$\exp(2it\mathcal{B}_{MN})$	$[M,N]=0$	$X\cdot\exp(t\sigma_y(MN-(MN)^\dagger)+it\sigma_x(MN+(MN)^\dagger))\cdot X$	[20]	Exact
11	$\exp\left(it\begin{pmatrix}2MN & 0\\ 0 & -NM-(NM)^\dagger\end{pmatrix}\right)$	$MN=(MN)^\dagger$	$\exp([S\cdot it\mathcal{B}_M\cdot S^\dagger, X\cdot it\mathcal{B}_N\cdot X])$	[20]	Exact
12	$\mathcal{B}_a = \exp\left(2i\alpha\begin{pmatrix}0 & a\\ a^\dagger & 0\end{pmatrix}\right)$	$\alpha=\alpha^*$	$\exp(i(\pi/2)a^\dagger a)\exp(i(\alpha(a^\dagger+a))\otimes\sigma_y)\exp(-i(\pi/2)a^\dagger a)\exp(i(\alpha(a^\dagger+a))\otimes\sigma_x)$	[20]	Approx
13	$\mathcal{B}_{a^\dagger} = \exp\left(2i\alpha\begin{pmatrix}0 & a^\dagger\\ a & 0\end{pmatrix}\right)$	$\alpha=\alpha^*$	$\exp(i(\pi/2)a^\dagger a)\exp(i(\alpha(a^\dagger+a))\otimes\sigma_y)\exp(-i(\pi/2)a^\dagger a)\exp(-i(\alpha(a^\dagger+a))\otimes\sigma_x)$	This paper	Approx
14	$e^{(P_1P_2\cdots P_n)(\alpha a_k^\dagger-\alpha^* a_k)}$		Multi-qubit-controlled displacement: Right hand side (RHS) of Equation (11) first line	[28]	Exact
15	$e^{2i\alpha^2 P_1P_2\cdots P_n}$		Multi-Pauli Exponential: Right hand side (RHS) of Equation (9) first line	This Paper	Exact
16	All Native Gates RHS in Table 2		All Native Gates Left Hand Side (LHS) Table 2	[28]	Exact



$$e^{(a^\dagger)^2a^2}$$

Basic Gates Sequence
(Logical Circuit)

Repeat the rewrite process until it produces **only basis gates**

RUTGERS
NC STATE UNIVERSITY

1.4 How symbolic compilation works

TABLE I: Frequently used qubit-boson gates, from [23].

Name	Operation
Bosonic Operators	
$\mathrm{R}_i(\theta)$	$e^{-i\theta\hat{n}_i}$
$\mathrm{D}_i(\alpha)$	$e^{\alpha\hat{a}_i^\dagger-\alpha^*\hat{a}_i}$
$\mathrm{BS}_{i,j}(\phi,\theta)$	$e^{-i\theta(e^{i\phi}\hat{a}_i^\dagger\hat{a}_j+e^{-i\phi}\hat{a}_i\hat{a}_j^\dagger)}$
Qubit Operators	
$R_j^z(\theta)$	$e^{-i\theta\hat{Z}_j}$
$R_j^y(\theta)$	$e^{-i\theta\hat{Y}_j}$
$R_j^x(\theta)$	$e^{-i\theta\hat{X}_j}$
CNOT	Controlled X
Coupled Bosonic-Qubit Operators	
$\mathrm{CR}_{i,j}(\theta)$	$e^{-i\theta/2\hat{Z}_i\hat{n}_j}$
$\mathrm{C\Pi}_{i,j}$	$e^{-i\pi/2\hat{Z}_i\hat{n}_j}$
$\mathrm{SNAP}_{i,j}(\vec{\theta})$	$e^{-i\hat{Z}_i\sum_n\theta_n\|n\rangle\langle n\|}$
$\mathrm{SQR}_{i,j}(\vec{\theta},\vec{\phi})$	$\sum_n\hat{R}_i^{\phi_n}(\theta_n)\|n\rangle\langle n\|_j$

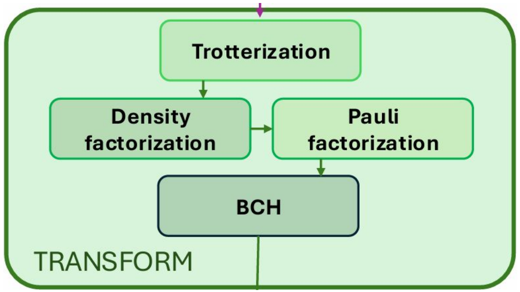
Gate	Expression	Ancilla qubit	Gate Decomposition
Density factorization	$\exp\left(-i(\hat{n}_i\hat{O}_j)\right)$	Required	$\prod_{k=0}^{K-1}\mathrm{SQR}_{\mathrm{anc},i}(\vec{\pi}_k,\vec{0})e^{-i2^{k-1}\hat{O}_j}e^{i2^{k-1}\hat{Z}_{\mathrm{anc}}\hat{O}_j}\mathrm{SQR}_{\mathrm{anc},i}(-\vec{\pi}_k,\vec{0})$
Pauli factorization	$\exp\left(\hat{Z}_i\left(\hat{\Theta}\hat{a}_j^\dagger-\hat{\Theta}^\dagger\hat{a}_j\right)\right)$	None	$\mathrm{C\Pi}_{i,j}\,e^{i\left(\hat{\Theta}\hat{a}_j^\dagger+\hat{\Theta}^\dagger\hat{a}_j\right)}\,\mathrm{C\Pi}_{i,j}^\dagger$
BCH	$\exp\left(-2i(\hat{O}_{\vec{I}}\hat{O}_{\vec{J}})\theta^2\right)$	Required	$\mathrm{CU}_{k,\vec{I}}^X(\theta)\,\mathrm{CU}_{k,\vec{J}}^Y(\theta)\,\mathrm{CU}_{k,\vec{I}}^X(-\theta)\,\mathrm{CU}_{k,\vec{J}}^Y(-\theta)+\mathcal{O}(\theta^3)$
Trotter	$\exp\left(-i\left(\hat{O}_{\vec{I}}+\hat{O}_{\vec{J}}\right)\theta\right)$	Required	$\mathrm{CU}_{i,\vec{I}}^Z(\theta)\,\mathrm{CU}_{j,\vec{J}}^Z(\theta)+\mathcal{O}(\theta^2)$

TABLE II: **Compiler rules**, reproduced with permission from [7]. Row 1) $\hat{\Theta}$ is any operator that commutes with $\hat{a}_j$ and $\hat{Z}_j$. Row 2) $K=\lceil\log_2(n_{\max}+1)\rceil$, with $n_{\max}$ is the maximum photon-number cutoff for mode i. Rows 2 & 3) Approximate methods. $\mathrm{CU}_{k,\vec{I}}^Z(\theta)=\exp\left(-i\theta\hat{X}_k\hat{O}_{\vec{I}}\right)$, where $\hat{O}_{\vec{I}}$ refers to an operator acting on bosonic modes with indices listed in $\vec{I}$. The superscripts X and Y in CU denote the qubit axis on which the operator $\hat{U}$ is conditioned. This axis can be controlled by conjugating CU^Z by single qubit rotations.

RUTGERS NC STATE UNIVERSITY

Gates and Rules Sets from Decker2025QCE.

1.4 How symbolic compilation works

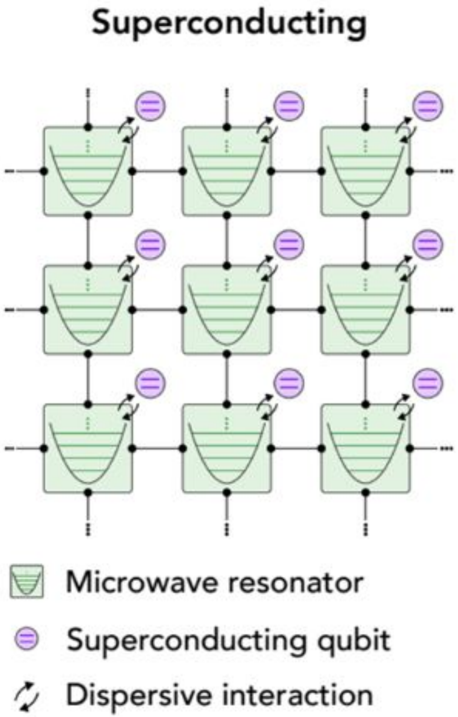


Gate	Expression	Ancilla qubit	Gate Decomposition
Density factorization	$\exp\left(-i(\hat{n}_i \hat{O}_j)\right)$	Required	$\prod_{k=0}^{K-1} \mathrm{SQR}_{\mathrm{anc},i}(\vec{\pi}_k, \vec{0}) e^{-i2^{k-1}\hat{O}_j} e^{i2^{k-1}\hat{Z}_{\mathrm{anc}}\hat{O}_j} \mathrm{SQR}_{\mathrm{anc},i}(-\vec{\pi}_k, \vec{0})$
Pauli factorization	$\exp\left(\hat{Z}_i \left(\hat{\Theta}\hat{a}_j^\dagger - \hat{\Theta}^\dagger \hat{a}_j\right)\right)$	None	$\mathrm{C\Pi}_{i,j}\, e^{i\left(\hat{\Theta}\hat{a}_j^\dagger + \hat{\Theta}^\dagger \hat{a}_j\right)}\, \mathrm{C\Pi}_{i,j}^\dagger$
BCH	$\exp\left(-2i(\hat{O}_{\vec{I}}\hat{O}_{\vec{J}})\theta^2\right)$	Required	$\mathrm{CU}_{k,\vec{I}}^X(\theta)\,\mathrm{CU}_{k,\vec{J}}^Y(\theta)\,\mathrm{CU}_{k,\vec{I}}^X(-\theta)\,\mathrm{CU}_{k,\vec{J}}^Y(-\theta) + \mathcal{O}(\theta^3)$
Trotter	$\exp\left(-i\left(\hat{O}_{\vec{I}} + \hat{O}_{\vec{J}}\right)\theta\right)$	Required	$\mathrm{CU}_{i,\vec{I}}^Z(\theta)\,\mathrm{CU}_{j,\vec{J}}^Z(\theta) + \mathcal{O}(\theta^2)$

TABLE II: **Compiler rules**, reproduced with permission from [7]. Row 1) $\hat{\Theta}$ is any operator that commutes with $\hat{a}_j$ and $\hat{Z}_j$. Row 2) $K = \lceil \log_2(n_{\max} + 1) \rceil$, with $n_{\max}$ is the maximum photon-number cutoff for mode i. Rows 2 & 3) Approximate methods. $\mathrm{CU}_{k,\vec{I}}^Z(\theta) = \exp\left(-i\theta \hat{X}_k \hat{O}_{\vec{I}}\right)$, where $\hat{O}_{\vec{I}}$ refers to an operator acting on bosonic modes with indices listed in $\vec{I}$. The superscripts X and Y in CU denote the qubit axis on which the operator $\hat{U}$ is conditioned. This axis can be controlled by conjugating CU^Z by single qubit rotations.

Rules Sets from Decker2025QCE.

1.5 Limited Hardware Connectivity



- Qumode-qumode Mapping.

Interactions are limited to adjacent qumodes. For non-adjacent qumodes, qumode SWAP gates are used, with routing optimized to minimize such SWAPs.

- Qubit-qumode Mapping.

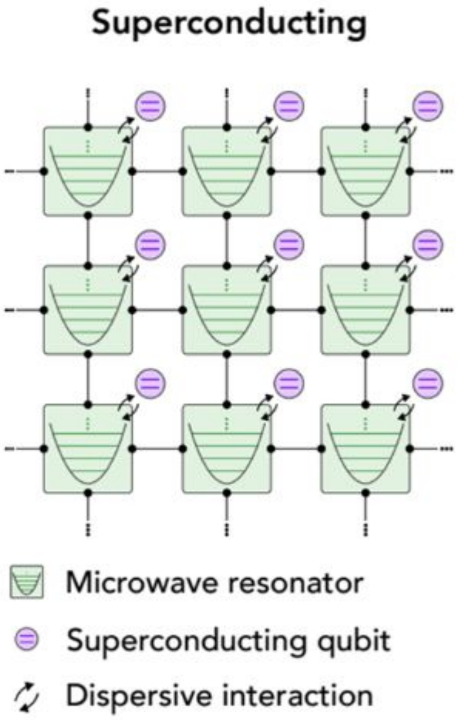
Each qumode interacts only with its associated qubit. For interactions with other qumodes, adjacency is established by moving qumodes, similar to qumode-qumode mapping.

- Qubit-qubit Mapping.

Qubits interact indirectly via an ancilla qumode, which is moved between qubits to mediate interactions and complete gate operations.

Source: **"Hybrid Oscillator-Qubit Quantum Processors: Instruction Set Architectures, Abstract Machine Models, and Applications."** Liu et al. arXiv:2407.10381

1.5 Limited Hardware Connectivity



- Qumode-qumode Mapping.

Interactions are limited to adjacent qumodes. For non-adjacent qumodes, qumode SWAP gates are used, with routing optimized to minimize such SWAPs.

- Qubit-qumode Mapping.

Each qumode interacts only with its associated qubit. For interactions with other qumodes, adjacency is established by moving qumodes, similar to qumode-qumode mapping.

$$SWAP_{i,j} = R_i(-\pi/2)R_j(-\pi/2)BS_{i,j}(\pi, 0)$$

Working Frontier: all unresolved gates whose dependence has been resolved

Using a Qiskit Sabre-like reward function to execute gate from the frontier and update it.

Source: **"Hybrid Oscillator-Qubit Quantum Processors: Instruction Set Architectures, Abstract Machine Models, and Applications."** Liu et al. arXiv:2407.10381

1.5 Limited Hardware Connectivity

Qumode-SWAP

1 Beam-Splitter gate

(20x depth/duration)

Qubit-SWAP (in CVDV System)

12 control displacement gates

12 Qumode-SWAPs

(480x depth/duration)

- Qubit-qubit Mapping

Qubits interact indirectly via an ancilla qumode, which is moved between qubits to mediate interactions and complete gate operations.

- Dynamic Qubit Floating

Alternative hardware platforms?

Optimization of SWAP cost and routing overhead in the future.

Are there specific demands about qubit connectivity and transport mechanisms?

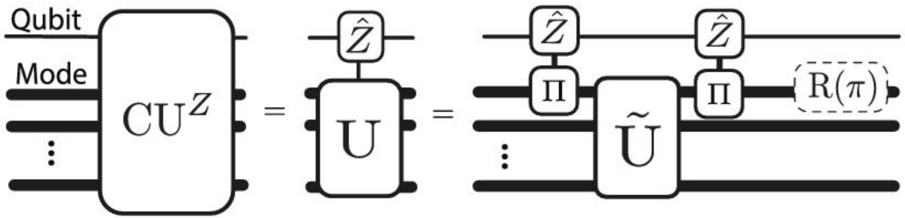
Source: **"Hybrid Oscillator-Qubit Quantum Processors: Instruction Set Architectures, Abstract Machine Models, and Applications."** Liu et al. arXiv:2407.10381

1.6 Multi-qubit controlled unitary

- Qubits are not directly connected with each other, we propose a scheme to synthesize an arbitrary multi-qubit controlled CV unitary on Hybrid CV-DV platforms.

$$\mathrm{CD}^{(k,P_1P_2\cdots P_n)}(\alpha) = U_{\mathrm{seq}}^{\dagger} D(i^n\alpha) U_{\mathrm{seq}}$$
$$= e^{(P_1P_2\cdots P_n)(\alpha a_k^{\dagger} - \alpha^* a_k)}$$

- *k* is the qumode index, *P_i* represents different control qubit index
- *D* can be replaced by any(or most of) CV unitary



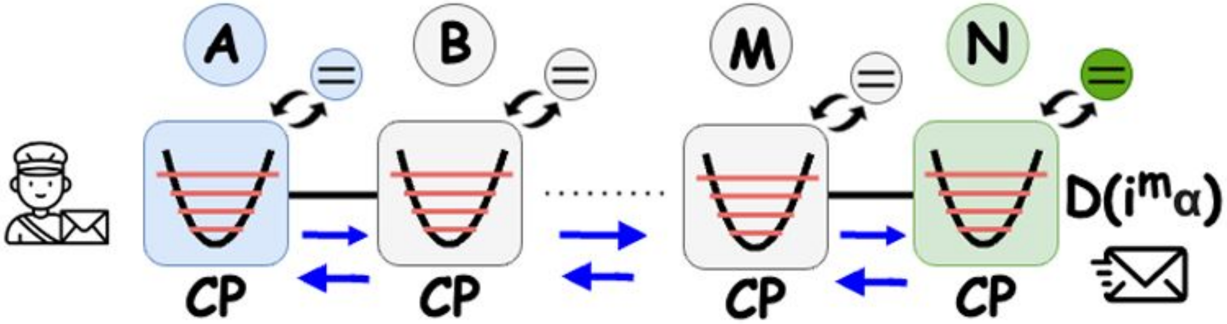
Source: **“Hybrid oscillator-qubit quantum processors: Simulating fermions, bosons, and gauge fields.”** Crane et al. arXiv:2409.03747

RUTGERS NC STATE UNIVERSITY

1.6 Multi-qubit controlled unitary

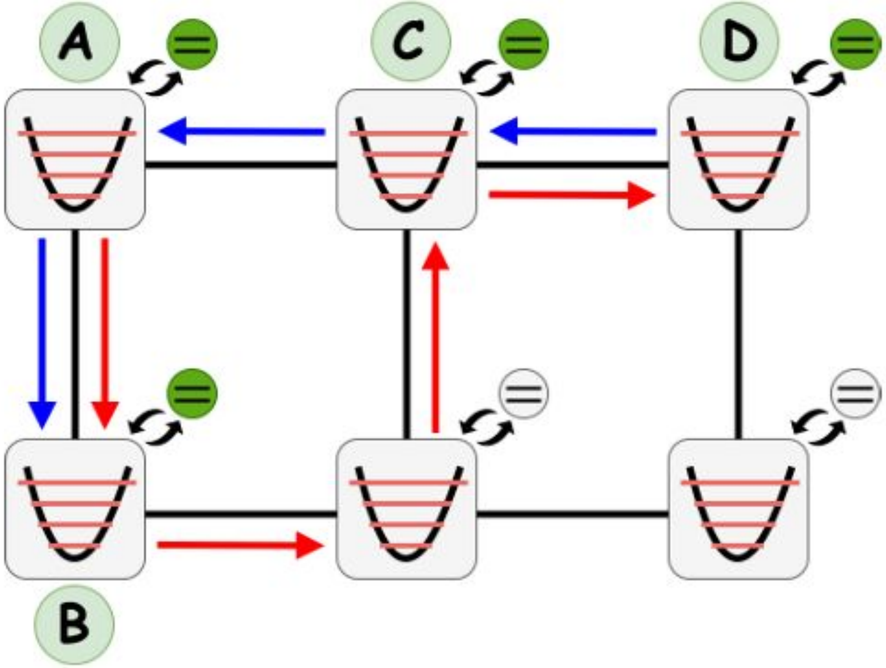
$$\mathrm{CU}^{Z}_{i,j} = e^{\hat{Z}_i(\hat{\Theta}\hat{a}_j^\dagger - \hat{\Theta}^\dagger \hat{a}_j)}$$
$$= \mathrm{C\Pi}_{i,j}\, e^{i(\hat{\Theta}\hat{a}_j^\dagger + \hat{\Theta}^\dagger \hat{a}_j)}\, \mathrm{C\Pi}^\dagger_{i,j}$$

$$\mathrm{CU}^{ZZ}_{i,j,k} = e^{\hat{Z}_i \hat{Z}_j(\hat{\Theta}\hat{a}_k^\dagger - \hat{\Theta}^\dagger \hat{a}_k)}$$



Source: **"Hybrid Oscillator-Qubit Quantum Processors: Instruction Set Architectures, Abstract Machine Models, and Applications."** Liu et al. arXiv 2407.10381v2

1.6 Multi-qubit controlled unitary

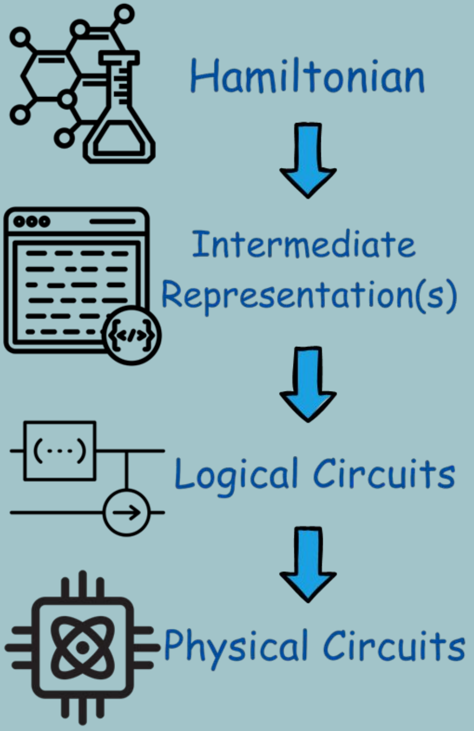


A → B → C → D: 4 BS gates

D → C → A → B: 3 BS gates

The Optimized Ancilla Qumode Routing Problem can be reformulated as a relaxed **Hamiltonian Path Problem**, similar to a modified Traveling Salesman Problem (TSP). Unlike the closed-path TSP, this problem allows revisiting vertices and does not require returning to the starting vertex.

Compilation



RUTGERS NC STATE UNIVERSITY

1. Symbolic Compilation for CV-DV Hamiltonians: Current Approaches
2. Challenges and Opportunities Beyond Today's Compilers
3. An End-to-End Compilation Walkthrough with Genesis

2.1 Programmability: Expanding benchmark coverage

- Current compilers support a limited set of native gates and decomposition rules
- Broader coverage enables compilation of more benchmarks

Metric	Genesis	Decker2025QCE
Gate set size	10	11
Rule set size	15	4
Benchmarks evaluated	6	2

2.1 Programmability: Expanding benchmark coverage

- Multi-qumode gates are limited supported in current work:
  - Only support Beam-Splitter gates

Beamsplitter	$BS(\theta,\varphi) = \exp[-i\frac{\theta}{2}(e^{i\varphi}a^\dagger b + \text{h.c.})]$
Two-mode Squeezing	$TMS(\xi) = \exp[\xi a^\dagger b^\dagger - \text{h.c.}]$
Two-mode Sum	$SUM(\lambda) = \exp[\frac{\lambda}{2}(a + a^\dagger)(b^\dagger - b)]$

Source: **"Hybridlane: A Software Development Kit for Hybrid Continuous-Discrete Variable Quantum Computing."** Furches et al. arXiv:2603.10919

2.1 Programmability: Support multi-qumode interactions

- Multi-qumode interactions are important:
  - More complex hamiltonian simulation: Rossini, Davide, and Rosario Fazio. "Phase diagram of the extended Bose–Hubbard model." New Journal of Physics 14.6 (2012): 065012.

$$\mathcal{H} = -J\sum_j \left(b_j^\dagger b_{j+1} + \text{h.c.}\right) + \frac{U}{2}\sum_j n_j(n_j - 1) + V\sum_j n_j n_{j+1}$$

  - Optimization problem using quantum Hamiltonian descent (QHD) formalism: Eq 22 "Hybrid continuous-discrete-variable quantum computing: a guide to utility." Kemper et al. arXiv:2511.13882

$$\hat{H}(t) = \sum_{k=1}^{N} \frac{\hat{p}_k^2}{2\mu(t)} + \sum_{j,k=0}^{N} V_{jk}\hat{x}_j\hat{x}_k + \sum_{j,k,l,m=0}^{N} W_{jklm}\hat{x}_j\hat{x}_k\hat{x}_l\hat{x}_m$$

  - Bosonic QEC, preparation multi-mode GKP state: Brenner, Lukas, et al. "Complexity of Gottesman-Kitaev-Preskill States." Physical Review X 15.3 (2025): 031073.

RUTGERS NC STATE UNIVERSITY

2.1 Programmability: Beyond ladder operators

- Support position basis encoding for direct mapping, instead using ladder operators for all the compilation tasks:

Cubic Phase $C(r) = \exp[-ir\hat{x}^3]$

$$\hat{H}(t) = \sum_{k=1}^{N} \frac{\hat{p}_k^2}{2\mu(t)} + \sum_{j,k=0}^{N} V_{jk}\hat{x}_j\hat{x}_k + \sum_{j,k,l,m=0}^{N} W_{jklm}\hat{x}_j\hat{x}_k\hat{x}_l\hat{x}_m$$

RUTGERS NC STATE UNIVERSITY

2.1 Programmability: Advanced hybrid gates

- Support more powerful hybrid gate:
  - SQR gate is a cavity-conditioned qubit rotation gate, also known as the photon-number **selective qubit rotation** (SQR) gate, is defined as:

$$\mathrm{SQR}(\vec{\theta}, \vec{\varphi}) = \sum_{n=0}^{N_{\max}} R_{\varphi_n}(\theta_n) \otimes |n\rangle\langle n|$$

  - SNAP gate is a powerful gate that imparts a different phase (chosen by the user) on each Fock state, also known as **photon number selective phase** (SNAP) gate, is defined as:

$$\mathrm{SNAP}(\vec{\varphi}) = e^{-i \sum_n \sigma_z \varphi_n |n\rangle\langle n|}$$

RUTGERS NC STATE UNIVERSITY

2.1 Programmability: Advanced hybrid gates

- Using SQR to allow using one ancilla qubit to enable a qumode conditional CV or DV gate
  - Compute → Control → Uncompute
  - Realize the *i*-conditional *j* operator, *i* and *j* can be multi qumode system

$$\exp\left(-i\, a_i^\dagger a_i\, M_j\right)$$

M_j Hermitian

$j \neq i$, M_i does not act on σ_z

$$\prod_{k=0}^{\lceil \log_2 (n_{\max}+1) \rceil -1} \mathrm{SQR}_i(\vec{\pi}_k, 0)\ \exp\left(-i 2^{k-1} M_j\right)\ \exp\left(i 2^{k-1} \sigma_z M_j\right)\ \mathrm{SQR}_i(-\vec{\pi}_k, 0)$$

$$H = \hat{n}_i \hat{n}_j = a_i^\dagger a_i \boxed{a_j^\dagger a_j}$$

2.2 Resource–Accuracy Tradeoff: Error analysis needed

- Many rules just intermediate transformation, they are exact equivalent rewriting.
- But Product Formulas are not free! Trotterization, BCH, Block encoding implementation are all approximate rewriting.

$$\epsilon_{\mathrm{Trot}} = \left\| e^{-i(M+N)t} - \left(e^{-iMt/k} e^{-iNt/k}\right)^k \right\|,$$

$$\epsilon_{\mathrm{BCH}} = \left\| e^{t^2[M,N]} - e^{tM} e^{tN} e^{-tM} e^{-tN} \right\|.$$

$$\epsilon_{\mathrm{Trot}_p} = O\left(\frac{t^{p+1}}{k^p}\right). \qquad \epsilon_{\mathrm{BCH}_p} = O(t^{2p+1})$$

2.2 Resource–Accuracy Tradeoff: A more detailed cost model needed

- Error analysis → Fidelity
- Gate sequence → Depth
- Some advanced hybrid gates / decomposition rules → Extra qubit ancillas
- Hardware-dependent availability → Not every gate is supported in specific hardware
- A more detailed cost model:

Type	Operation	Short	Operator	Estimated gate time	Estimated fidelity	Ref.
	Rotation	$R(\theta)$	$e^{i\theta\hat{a}^\dagger\hat{a}}$	200 μs[a]	99%[a]	[121]
	Displacement	$D(z)$	$e^{z\hat{a}^\dagger - z^*\hat{a}}$	10 μs	99%	[122]
Qumode gates	Single-mode squeezing	$S(z)$	$e^{(z^*\hat{a}\hat{a} - z\hat{a}^\dagger\hat{a}^\dagger)/2}$	3 μs	98%	[123]
	Red sideband	$\mathrm{RSB}(z)$	$e^{iz\hat{a}\sigma^+ + iz^*\hat{a}^\dagger\sigma^-}$	200 μs	99.9%	[68]
	Blue sideband	$\mathrm{BSB}(z)$	$e^{iz\hat{a}^\dagger\sigma^+ + iz^*\hat{a}\sigma^-}$	200 μs	99.9%	[68]
Hybrid gates	Controlled rotation	$\mathrm{CR}(\theta)$	$e^{i\theta\sigma^z\hat{a}^\dagger\hat{a}}$	200 μs[a]	99%[a]	[13]
	Controlled displacement	$\mathrm{CD}(z)$	$e^{\sigma^z(z\hat{a}^\dagger - z^*\hat{a})}$	800 μs	95%[a]	[125]

Source: **"Hybrid quantum simulations with qubits and qumodes on trapped-ion platforms."** Araz et al. arXiv:2410.07346

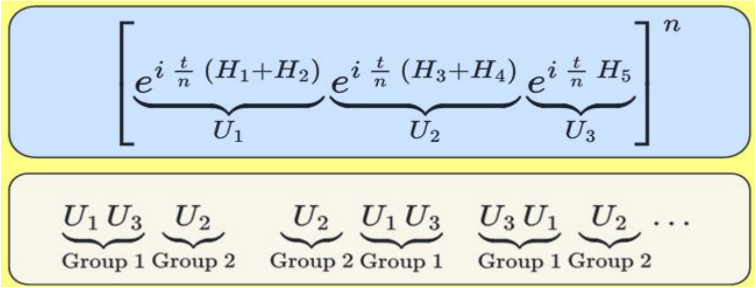
2.3 Flexibility: Commutativity / Associativity

- Commutativity

  A + B = B + A

- Associativity

  A + B + C = (A + B) + C = A + (B + C)



Decker, Ethan, et al. "Kernpiler: Compiler optimization for quantum hamiltonian simulation with partial trotterization." arXiv preprint arXiv:2504.07214 (2025).

2.3 Flexibility: Commutation Rules

- For bosonic operators, the fundamental commutation relation is:

$$[a, a^\dagger] = aa^\dagger - a^\dagger a = 1,$$

$$aa^\dagger = a^\dagger a + 1.$$

- It allows a lot flexibility for higher order of ladder operator terms:

$$a^\dagger a a^\dagger a = a^\dagger (aa^\dagger) a = a^\dagger (a^\dagger a + 1) a$$

$$= a^\dagger a^\dagger a a + a^\dagger a.$$

Capability	Genesis	Decker2025QCE
Commutation-aware rewriting	❌	preprocessing only

RUTGERS NC STATE UNIVERSITY

2.3 Flexibility: BCH commutator construction

- For Cross-Kerr term, we can construct the intermediate which is product formula decomposition friendly, then solve the compilation

$$H = \hat{n}_i \hat{n}_j = a_i^\dagger a_i \, a_j^\dagger a_j.$$

Instead of directly synthesizing this oscillator-only interaction, we first construct a controlled version,

$$U(t) = e^{-i\sigma_z \hat{n}_i \hat{n}_j t} = e^{-i\sigma_z a_i^\dagger a_i a_j^\dagger a_j t},$$

which reduces to the desired oscillator unitary by initializing the auxiliary qubit in the $|0\rangle$ state (the +1 eigenstate of σ_z).

RUTGERS NC STATE UNIVERSITY

2.3 Flexibility: BCH commutator construction

- We can construct a commutator structure using some extra Pauli operators, when we have a commutator structure, it can help us using product formula to compile.

$$e^{-At}e^{-Bt}e^{At}e^{Bt} = \exp\left(t^2[A,B] + O(t^3)\right).$$

Since number operators on different modes commute,

$$[\hat{n}_i, \hat{n}_j] = 0,$$

we inject non-commutativity through Pauli operators by defining

$$A = -i\frac{1}{\sqrt{2}}\sigma_x\hat{n}_i, \qquad B = -i\frac{1}{\sqrt{2}}\sigma_y\hat{n}_j.$$

Because the oscillator and qubit operators act on independent Hilbert spaces, the commutator factorizes:

$$[A,B] = (-i)^2\frac{1}{2}[\sigma_x,\sigma_y]\,\hat{n}_i\hat{n}_j = -\frac{1}{2}(2i\sigma_z)\,\hat{n}_i\hat{n}_j = -i\sigma_z\hat{n}_i\hat{n}_j.$$

2.3 Flexibility: BCH commutator construction

- One hamiltonian, different decomposition strategies, **which we should choose?**
  - Product formula:

$$\exp(-i\,\sigma_z \hat{n}_i \hat{n}_j\, t^2) \approx \exp(At)\exp(Bt)\exp(-At)\exp(-Bt) + O(t^3)$$

$$A = -i\frac{1}{\sqrt{2}}\sigma_x \hat{n}_i, \qquad B = -i\frac{1}{\sqrt{2}}\sigma_y \hat{n}_j.$$

CR **gates**

  - Selective qubit rotation (SQR) gate:

$$\prod_{k=0}^{\lceil \log_2(n_{\max}+1)\rceil-1} \mathrm{SQR}_i(\vec{\pi}_k, 0)\ \exp(-i2^{k-1}M_j)\ \exp(i2^{k-1}\sigma_z M_j)\ \mathrm{SQR}_i(-\vec{\pi}_k, 0)$$

RUTGERS NC STATE UNIVERSITY

2.3 Flexibility: CVDV QSP

- For Kerr nonlinear term, on the right hand is normal ordering, on the left hand is the polynomial of number operator:

$$a^\dagger a a^\dagger a = a^\dagger (a a^\dagger) a = a^\dagger (a^\dagger a + 1) a$$
$$= a^\dagger a^\dagger a a + a^\dagger a.$$

- For all hamiltonian terms with the same number of annihilation and creation operator can be rewritten to the form of normal ordering, for example if the number of each type of ladder operator is *m*:

$$\sum_{k=0}^{m} c_k (a^\dagger)^k a^k$$

RUTGERS NC STATE UNIVERSITY

2.3 Flexibility: CVDV QSP

- For all hamiltonian terms with normal ordering, it can be decomposed to the combination of number operator *n*:

Normal Ordering:

$$a^\dagger a^\dagger \cdots a^\dagger a a \cdots a$$

Falling factorial:

$$(x)_m = x(x-1)(x-2)\cdots(x-m+1) = ff(x,m)$$

For example,

$$(x)_0 = 1 \quad (1)$$
$$(x)_1 = x \quad (2)$$
$$(x)_2 = x(x-1) = x^2 - x \quad (3)$$
$$(x)_3 = x(x-1)(x-2) = x^3 - 3x^2 + 2x \quad (4)$$

Using proof by mathematical induction, we can show that:

$$(a^\dagger)^m a^m = (n)_m = ff(n,m)$$

RUTGERS NC STATE UNIVERSITY

2.3 Flexibility: CVDV QSP

- Then we have a polynomial of number operator, using the m=2 to see the case of Kerr nonlinear term:

$$(x)_0 = 1$$
$$(x)_1 = x$$
$$(x)_2 = x(x-1) = x^2 - x$$
$$(x)_3 = x(x-1)(x-2) = x^3 - 3x^2 + 2x$$

$$(a^\dagger)^2 a^2 = n(n-1) = n^2 - n$$
$$= (a^\dagger a)^2 - a^\dagger a$$

2.3 Flexibility: CVDV QSP

- We can transform all hamiltonian terms with the same number of annihilation and creation operator to *P(n)*.
- QSP can efficiently approximate/synthesis circuit using the gate sequences of signal operator and extra rotation gate, with constant number of qubit qumode requires.
- We can use *CR* gate to work as signal operator in CVDV systems.

$$\mathrm{CR}(\theta) = \exp\left[-i\frac{\theta}{2}\sigma_z a^\dagger a\right]$$

- More resources:

> Martyn, John M., et al. "Grand unification of quantum algorithms." PRX quantum 2.4 (2021): 040203.

> Motlagh, Danial, and Nathan Wiebe. "Generalized quantum signal processing." PRX Quantum 5.2 (2024): 020368.

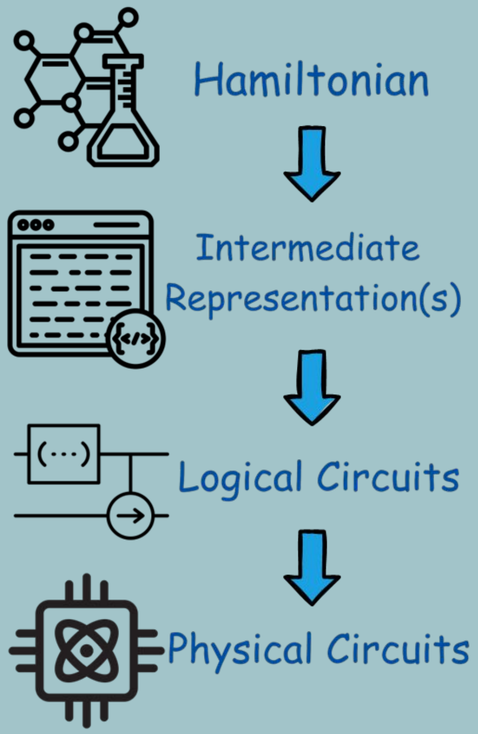
> Liu, Yuan, et al. "Hybrid oscillator-qubit quantum processors: Instruction set architectures, abstract machine models, and applications." PRX Quantum 7.1 (2026): 010201.

2.4 Verification, Simulation and Implementation

- Algorithm implementation and numerical verification
  - Mohapatra, Shubdeep, et al. "HyQBench: A Benchmark Suite for Hybrid CV-DV Quantum Computing." arXiv preprint arXiv:2603.04398 (2026).

- Circuit level implementation and calibration
  - Furches, Jim, Timothy J. Stavenger, and Carlos Ortiz Marrero. "Hybridlane: A Software Development Kit for Hybrid Continuous-Discrete Variable Quantum Computing." arXiv preprint arXiv:2603.10919 (2026).

Compilation

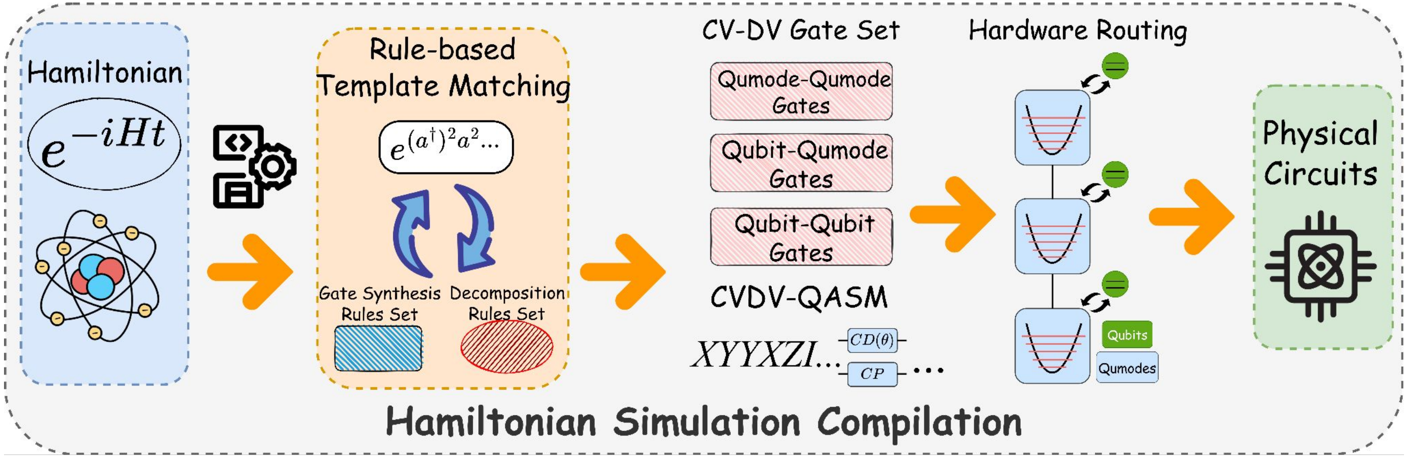


RUTGERS NC STATE UNIVERSITY

1. Symbolic Compilation for CV-DV Hamiltonians: Current Approaches
2. Challenges and Opportunities Beyond Today's Compilers
3. An End-to-End Compilation Walkthrough with Genesis

Genesis CVDV Compiler

https://github.com/ruadapt/Genesis-CVDV-Compiler



ArXiv:

Code:

RUTGERS NC STATE UNIVERSITY

Genesis: Multi-IR system(Python+ANTLR)

1. Hamiltonian Formula

$$H = -t\sum_{i,j,\sigma} c^{\dagger}_{i\sigma}c_{j\sigma} + U\sum_{i} b_i^{\dagger} b_i + g\sum_{i,\sigma} \hat{n}_{i\sigma}(b_i^{\dagger} + b_i)$$

2. Researcher-friendly Hamiltonian Grammar(DSL for formula description)

```
- t *Sum_over(i, j, sigma){FC[i][sigma]* FA[j][sigma]}
+ U *Sum_over(i){BC[i]* BA[i]}
+ g * Sum_over(i, sigma){TensorProd(FN[i][sigma], BC[i] + BA[i])};
```

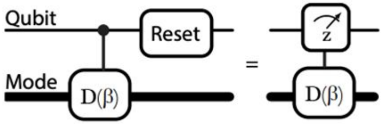
3. Intermediate Representation (Dialect for compilation pipeline)

```
pauli(0.392699075j): IXIZIYII;
pauli(0.392699075j): IIYIXIII;
bosonic: exp(prod((-1j),dagger(b(0)),b(0)));
bosonic: exp(prod((-1j),dagger(b(1)),b(1)));
hybrid: exp(prod((-0.78539815j),sigma(0, 0),sum(dagger(b(0)),b(0))));
hybrid: exp(prod((0.78539815j),sigma(3, 0),sum(dagger(b(0)),b(0))));
```

4. Logical CVDVQASM file(Gate-level, optimization finished)

```
// Pauli String with Parameter
pauli(pi/4) YIYZXXIIIIIIII;
pauli(pi/4) XZYZXYIIIIIIII;
...
// Phase Space Rotation Gate
R(pi/4) qm[1];
R(pi/4) qm[2];
...
// Control Displacement Gate
CD(pi/4) q[2], qm[1];
CD(pi/4) q[3], qm[1];
...
// Displacement Gate
D(pi/4) qm[2];
D(pi/4) qm[2];
...
```

5. Physical CVDVQASM file(Assembly code generated)



Reconfigurable Parser and Grammar

```
/*--------------------------------------
 *  LEXER RULES
 *--------------------------------------

CONST           : 'Const' ;
RANGE           : 'Range' ;
SUM_OVER        : 'Sum_over' ;
PROD_OVER       : 'Prod_over' ;
TENSORPROD      : 'TensorProd' ;
TENSORPROD_OVER : 'TensorProd_over' ;
IMAG            : 'imag' ;

FC              : 'FC' ;
FA              : 'FA' ;
FN              : 'FN' ;
BC              : 'BC' ;
BA              : 'BA' ;
```

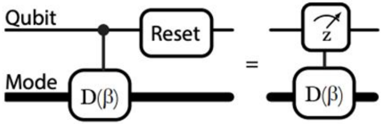
```
class QuantumOpNode(ExpressionNode):
    """
    Represents quantum operators:
      FC[i][sigma], FA[i], FN[i], BC[i], BA[i], Pauli_X[i], etc.
    """
    def __init__(self, op_type: str, indices: List[ExpressionNode], line: Optional[
        super().__init__(line, column)
        self.op_type = op_type
        self.indices = indices
```

Define AST

ANTLR generate a visitor.py to access the AST. Build a interpreter to do further compilation

Multiple IR for multiple compilation/optimization

Physical CVDVQASM file(Assembly code generated)



Logical CVDVQASM file(Gate-level, optimization finished)

```
// Pauli String with Parameter
pauli(pi/4) YIYZXXIIIIIIII;
pauli(pi/4) XZYZXYIIIIIIII;
...
// Phase Space Rotation Gate
R(pi/4) qm[1];
R(pi/4) qm[2];
...
// Control Displacement Gate
CD(pi/4) q[2], qm[1];
CD(pi/4) q[3], qm[1];
...
// Displacement Gate
D(pi/4) qm[2];
D(pi/4) qm[2];
...
```

Intermediate Representation (Dialect for compilation pipeline)

```
pauli(0.392699075j): IXIZIYII;
pauli(0.392699075j): IIYIXIII;
bosonic: exp(prod((-1j),dagger(b(0)),b(0)));
bosonic: exp(prod((-1j),dagger(b(1)),b(1)));
hybrid: exp(prod((-0.785398615j),sigma(0, 0),sum(dagger(b(0)),b(0))));
hybrid: exp(prod((0.785398615j),sigma(3, 0),sum(dagger(b(0)),b(0))));
```

Genesis CVDV Compiler

- benchmark
- doc
- grammar
- qubit_hamiltonians
- src
- tests
- .gitattributes
- .gitignore
- License
- README.md
- requirements.txt

https://github.com/ruadapt/Genesis-CVDV-Compiler/blob/main/doc/grammar.md

Hamiltonian DSL (H-DSL) Specification

This document specifies the **Hamiltonian Domain-Specific Language (H-DSL)** implemented in the `hamiltonianDSL.g4` grammar. It enables compact representation of quantum Hamiltonians with fermionic and bosonic operators.

🔍 Example

```
Const t = 1;
Const U = 1;
Const g = 1;

Range i = [0, 10, 1];
Range j = [0, 10, 1];
Range sigma = [0, 2, 1];

Result = - t * Sum_over(i, j, sigma){FC[i][sigma] * FA[j][sigma]}
       + U * Sum_over(i){BC[i] * BA[i]}
       + g * Sum_over(i, sigma){TensorProd(FN[i][sigma], BC[i] + BA[i])};
```

This corresponds to the Hamiltonian:

$$H = -t\sum_{i,j,\sigma} c_{i\sigma}^\dagger c_{j\sigma} + U\sum_i b_i^\dagger b_i + g\sum_{i,\sigma}\hat{n}_{i\sigma}(b_i^\dagger + b_i)$$

Genesis CVDV Compiler

- benchmark
- doc
- grammar
- qubit_hamiltonians
- src
- tests
- .gitattributes
- .gitignore
- License
- README.md
- requirements.txt

https://github.com/ruadapt/Genesis-CVDV-Compiler/blob/main/tests/demo.ipynb

Code Demo

In [1]:

```
# We need to change the working directory to the parent directory of the project and run the following shell commands.

import os
os.chdir("..")
```

Demonstration Purpose Small Size Demo

Single-file Mode

In [2]:

```
# Default single-file mode
!python3 -m src.main benchmark/electronicVibration_small.ham
# Specify output file
!python3 -m src.main benchmark/electronicVibration_small.ham -o output/electronicVibration_small.cvdvqasm
# Debug mode
!python3 -m src.main benchmark/electronicVibration_small.ham --debug
```

```
Processing job 1 of 1
Parsing input file: benchmark/electronicVibration_small.ham
Found 1 result(s) in benchmark/electronicVibration_small.ham
Writing intermediate result 'Result' to: output/intermediate_electronicVibration_small_Result.cvdvqasm
Writing logical result 'Result' to: output/logical_electronicVibration_small_Result.cvdvqasm
Writing final result 'Result' to: output/electronicVibration_small_Result.cvdvqasm
Mapping output/logical_electronicVibration_small_Result.cvdvqasm_depth_sum_simulated_annealing_tsp_qbstuck
Finish all parsing, decomposition, and mapping for input file: benchmark/electronicVibration_small.ham
Time taken: 14.21971869468689 seconds

Total time taken: 14.219862222671509 seconds
```

RUTGERS

NC STATE UNIVERSITY

Thank You!