First studies of Block encoding

Shi Qiu

Motivation

- The combinatorial track-finding problem can be mapped to solving the system of linear equation:

$$Ax = b \Rightarrow x = A^{-1}b$$

- The linear algebraic structure is compatible with algorithms like HHL, which has runtime scales like

$$O(\log N\, s^2 \kappa^2/\epsilon)$$

Sparsity: $s = \max_i \#\{j: A_{ij} \neq 0\}$; Condition number $\kappa = \lambda_{\max}/\lambda_{\min}$; Target error in the output state: $\epsilon = \||\tilde{x}\rangle - |x^\star\rangle\|$

- QSVT says that if $\boldsymbol{A/\alpha_A}$ (α_A subnormalization factor) is **block encoded** by a (big) oracle O, then one can **block-encode** a scaled version of $\boldsymbol{A^{-1}}$ using about

$$O(\kappa \log(1/\epsilon))$$

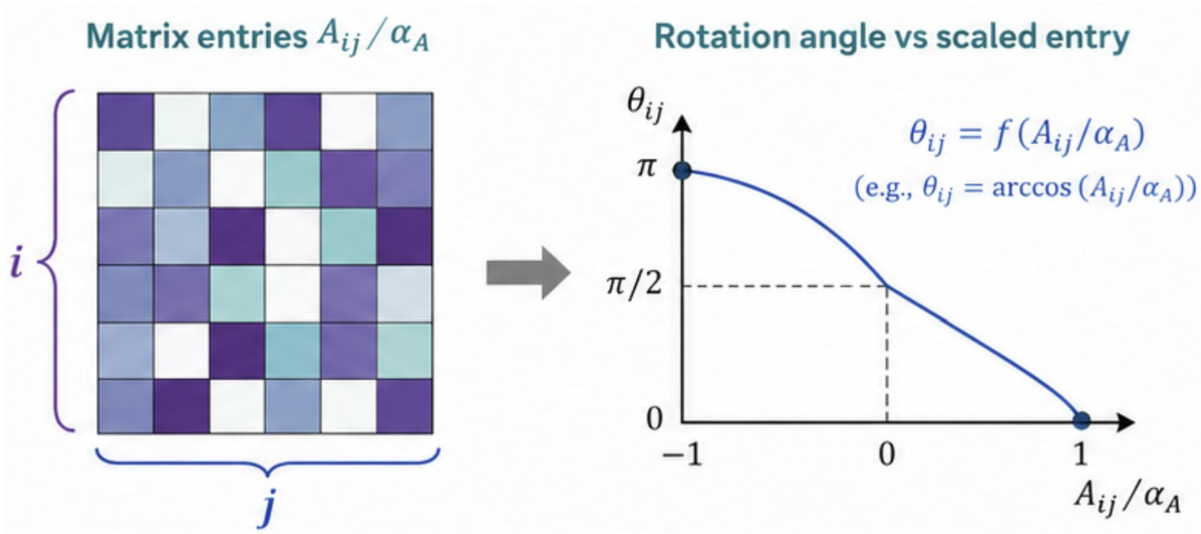
Block encode a general matrix

- Block encoding means that the target matrix A (properly scaled by α_A) is placed in the upper-left corner of the larger unitary.

$$U = \begin{bmatrix} A/\alpha & \cdot \\ \cdot & \cdot \end{bmatrix} \qquad \Longrightarrow \qquad A = \alpha(\langle 0| \otimes I)U(|0\rangle \otimes I)$$

- Most general idea is that you apply the rotation to ancilla qubit(s) where the rotation angle is related to each entry values.

- Three registers:
  - ancilla: 1 qubit, where amplitudes get loaded
  - i register: n qubits (the "work" register representing rows),
  - j register: n qubits (the "data" register representing columns / input basis states)



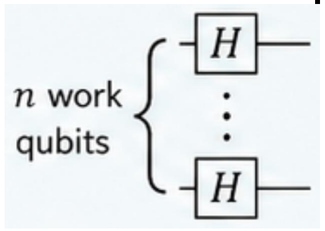
Each (scaled) entry A_{ij}/α_A determines the rotation angle θ_{ij} used on the ancilla.

Block encode a general matrix

- Starting with an arbitrary input $|\Psi\rangle = \sum_{j=0}^{N-1} \psi_j |j\rangle$, where $N = 2^n$ is the matrix size of A

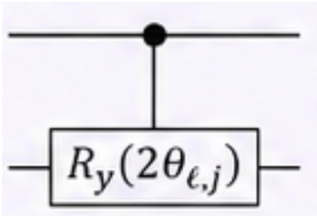
$$|\Psi_0\rangle = |0\rangle_a |0^n\rangle_W \otimes |\Psi\rangle_J$$

- D_s apply Hadamards on $|0^n\rangle_W$:



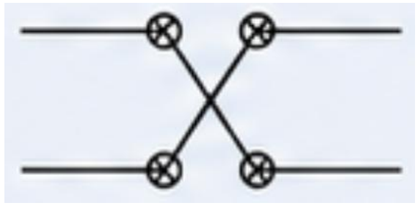
$$|\Psi_0\rangle \xrightarrow{H^{\otimes n}|0^n\rangle_W} \frac{1}{\sqrt{N}} \sum_{\ell,j} \psi_j |0\rangle_a |\ell\rangle_W |j\rangle_J$$

- O_A rotates the amplitude ancilla conditioned on (ℓ, j) so that:



$$|\Psi_1\rangle \xrightarrow{O_A} |\Psi_2\rangle = \frac{1}{\sqrt{N}} \sum_{\ell,j} \psi_j \left(A_{\ell,j} |0\rangle_a + \sqrt{1 - A_{\ell,j}^2} |1\rangle_a \right) |\ell\rangle_W |j\rangle_J$$

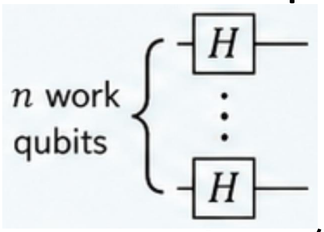
- SWAP the two n-qubit registers and does not touch ancilla qubit



$$|\Psi_2\rangle \xrightarrow{\text{SWAP}} |\Psi_3\rangle = \frac{1}{\sqrt{N}} \sum_{\ell,j} \psi_j \left(A_{\ell,j} |0\rangle_a + \cdots |1\rangle_a \right) |j\rangle_W |\ell\rangle_J$$

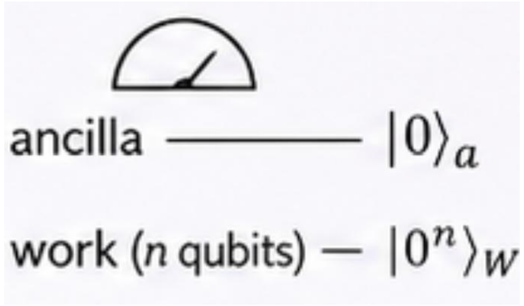
Block encode a general matrix

- $D_s^\dagger$ applys final $H^{\otimes n}$ on the work register ($D_s^\dagger|j\rangle = \frac{1}{\sqrt{N}}\sum_{t=0}(-1)^{t\cdot j}|t\rangle$, where $t \cdot j$ is the bitwise inner product mod 2):



$$|\Psi_3\rangle = \frac{1}{N}\sum_t \sum_{\ell,j} \psi_j (-1)^{t\cdot j}\left(A_{\ell,j}|0\rangle_a + \cdots |1\rangle_a\right)|t\rangle_W|\ell\rangle_J$$

- Inner product $\langle 0|_a\langle 0^n|_W \otimes I_J|\Psi_3\rangle$



$$\langle 0|_a\langle 0^n|_W \otimes I_J|\Psi_3\rangle = \frac{1}{N}\sum_{\ell,j}\psi_j A_{\ell,j}|\ell\rangle_J$$

$$= \frac{1}{N}\sum_{\ell}\left(\sum_j A_{\ell,j}\,\psi_j\right)|\ell\rangle_J = \frac{A}{N}|\Psi\rangle_J$$

- All circuit elements, except for O_A, can be readily written in simple 1- and 2-qubit gates.

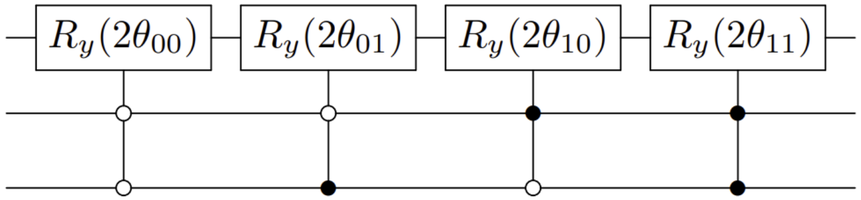
FABLE method to block encode a general matrix

- Goal: Implement O_A in simple 1- and 2-qubit gates for arbitrary matrices
- O_A acts on the $|0\rangle_a$ state as an R_y gate with angle

$$\theta_{ij} = \arccos(A_{ij})$$

- Naïve implementation of O_A will use N^2 multi-controlled Ry gates

  e.g. for a 2×2 matrix



FABLE: Fast Approximate Quantum Circuits for Block-Encodings

https://arxiv.org/pdf/2205.00081

FABLE method to block encode a general matrix

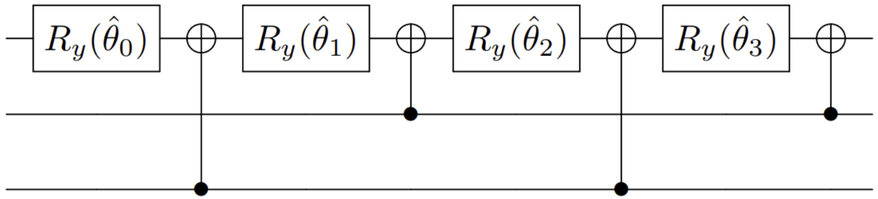
- Goal: Implement O_A in simple 1- and 2-qubit gates for arbitrary matrices
- O_A acts on the $|0\rangle_a$ state as an R_y gate with angle

$$\theta_{ij} = \arccos(A_{ij})$$

- Naïve implementation of O_A will use N^2 multi-controlled Ry gates

  e.g. for a 2×2 matrix

- FABLE method implement O_A following grey code order (00, 01, 11, 10)



	Gray code			
	4	3	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	0	1	0
4	0	1	1	0
5	0	1	1	1
6	0	1	0	1
7	0	1	0	0
8	1	1	0	0
9	1	1	0	1
10	1	1	1	1
11	1	1	1	0
12	1	0	1	0

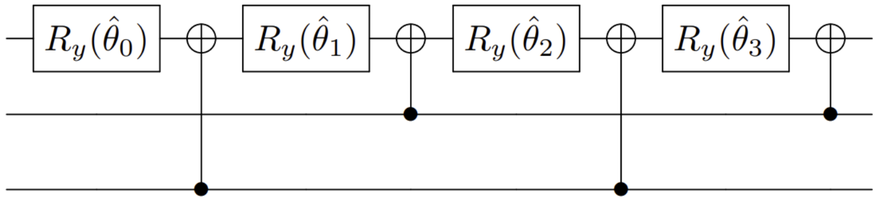
FABLE: Fast Approximate Quantum Circuits for Block-Encodings

https://arxiv.org/pdf/2205.00081

FABLE method to block encode a general matrix

- FABLE method uses the key identity: $XR_y(\theta)X = R_y(-\theta)$
- The state of the first qubit is rotated as:

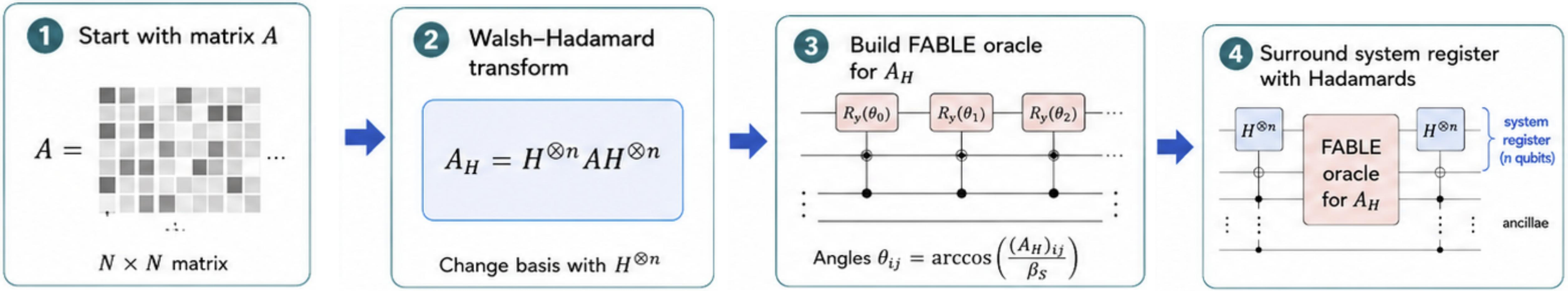
$$
\begin{aligned}
00: \quad & R_y(\hat{\theta}_3)\quad R_y(\hat{\theta}_2)\quad R_y(\hat{\theta}_1)\quad R_y(\hat{\theta}_0) = \\
& R_y(\ \ \hat{\theta}_3 + \hat{\theta}_2 + \hat{\theta}_1 + \hat{\theta}_0), \\
01: \quad & R_y(\hat{\theta}_3)XR_y(\hat{\theta}_2)\quad R_y(\hat{\theta}_1)XR_y(\hat{\theta}_0) = \\
& R_y(\ \ \hat{\theta}_3 - \hat{\theta}_2 - \hat{\theta}_1 + \hat{\theta}_0), \\
10: \quad & XR_y(\hat{\theta}_3)\quad R_y(\hat{\theta}_2)XR_y(\hat{\theta}_1)\quad R_y(\hat{\theta}_0) = \\
& R_y(-\hat{\theta}_3 - \hat{\theta}_2 + \hat{\theta}_1 + \hat{\theta}_0), \\
11: \quad & XR_y(\hat{\theta}_3)XR_y(\hat{\theta}_2)XR_y(\hat{\theta}_1)XR_y(\hat{\theta}_0) = \\
& R_y(-\hat{\theta}_3 + \hat{\theta}_2 - \hat{\theta}_1 + \hat{\theta}_0),
\end{aligned}
$$



- The new angles $\hat{\theta}$ are related to θ as:

$$
\begin{bmatrix}\theta_0\\ \theta_1\\ \theta_2\\ \theta_3\end{bmatrix} = \begin{bmatrix}1 & 1 & 1 & 1\\ 1 & -1 & -1 & 1\\ 1 & 1 & -1 & -1\\ 1 & -1 & 1 & -1\end{bmatrix} \begin{bmatrix}\hat{\theta}_0\\ \hat{\theta}_1\\ \hat{\theta}_2\\ \hat{\theta}_3\end{bmatrix} = \begin{bmatrix}1 & 1 & 1 & 1\\ 1 & -1 & 1 & -1\\ 1 & 1 & -1 & -1\\ 1 & -1 & -1 & 1\end{bmatrix} \begin{bmatrix}1 & & & \\ & 1 & & \\ & & 0 & 1\\ & & 1 & 0\end{bmatrix} \begin{bmatrix}\hat{\theta}_0\\ \hat{\theta}_1\\ \hat{\theta}_2\\ \hat{\theta}_3\end{bmatrix} = (\hat{H}\otimes\hat{H})P_G \begin{bmatrix}\hat{\theta}_0\\ \hat{\theta}_1\\ \hat{\theta}_2\\ \hat{\theta}_3\end{bmatrix}
$$

(S-)FABLE method to block encode a general matrix



- For matrix of size $N = 2^n$, need N^2 CNOT and R_y gates, respectively
- One can apply R_y only for $\widehat{\boldsymbol{\theta}}$ above some threshold to approximately block encode the matrix
- High compressibility generally needs matrices which are sparse in the Walsh-Hadamard domain.
- If A is a sparse matrix, then the matrix $H^{\otimes n} A H^{\otimes n}$ is sparse in the Walsh-Hadamard domain because $H^{\otimes n}\left(H^{\otimes n} A H^{\otimes n}\right) H^{\otimes n} = \mathrm{A}$ is sparse
- S-FABLE method block encodes HAH and then sandwich it with $H(HAH)H$

FABLE: Fast Approximate Quantum Circuits for Block-Encodings

https://arxiv.org/pdf/2205.00081

S-FABLE and LS-FABLE: Fast approximate block-encoding algorithms for unstructured sparse matrices

https://arxiv.org/pdf/2401.04234

Generate tracking matrices

- Tracking matrices are generated with following settings:

  measurement error = 0.02

  collision noise = 0.02

  drop rate = 0

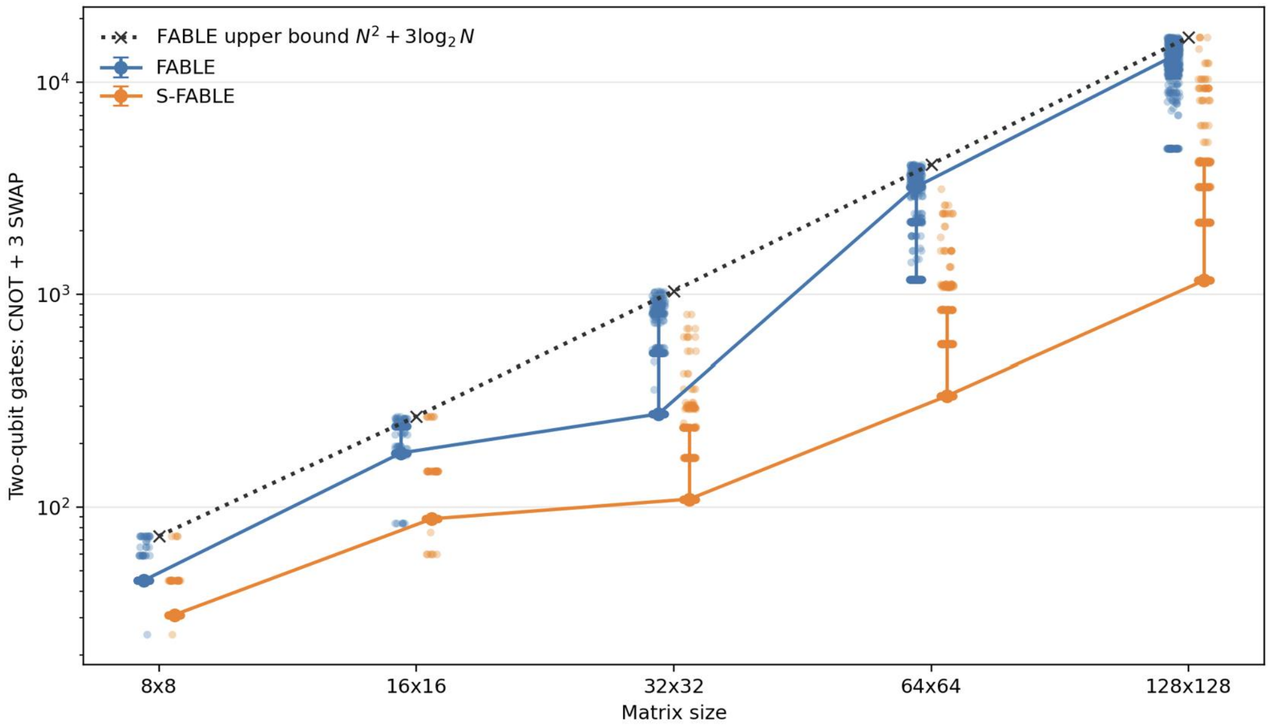
  ghost rate = 0

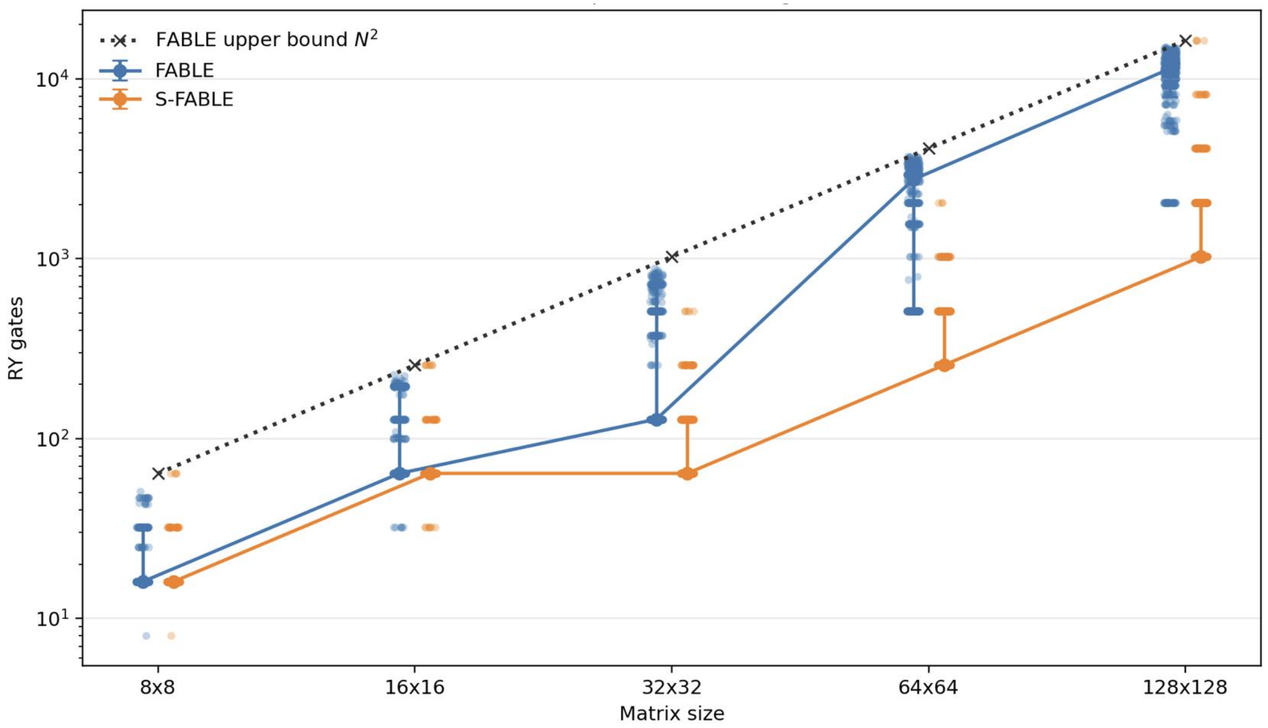
  epsilon = 0.001

- Choose matrix sizes equalling to only $2^n \times 2^n$ $(3 \leq n \leq 7)$. The matrix size is decided by $\left(\mathrm{n}_{\mathrm{layers}} - 1\right) \times n^2_{\mathrm{particles}}$.

Matrix size	Number of layers	Number of particles	Number of generated matrices
8×8	3	2	500
16×16	5	2	500
32×32	3	4	500
64×64	5	4	500
128×128	9	4	500

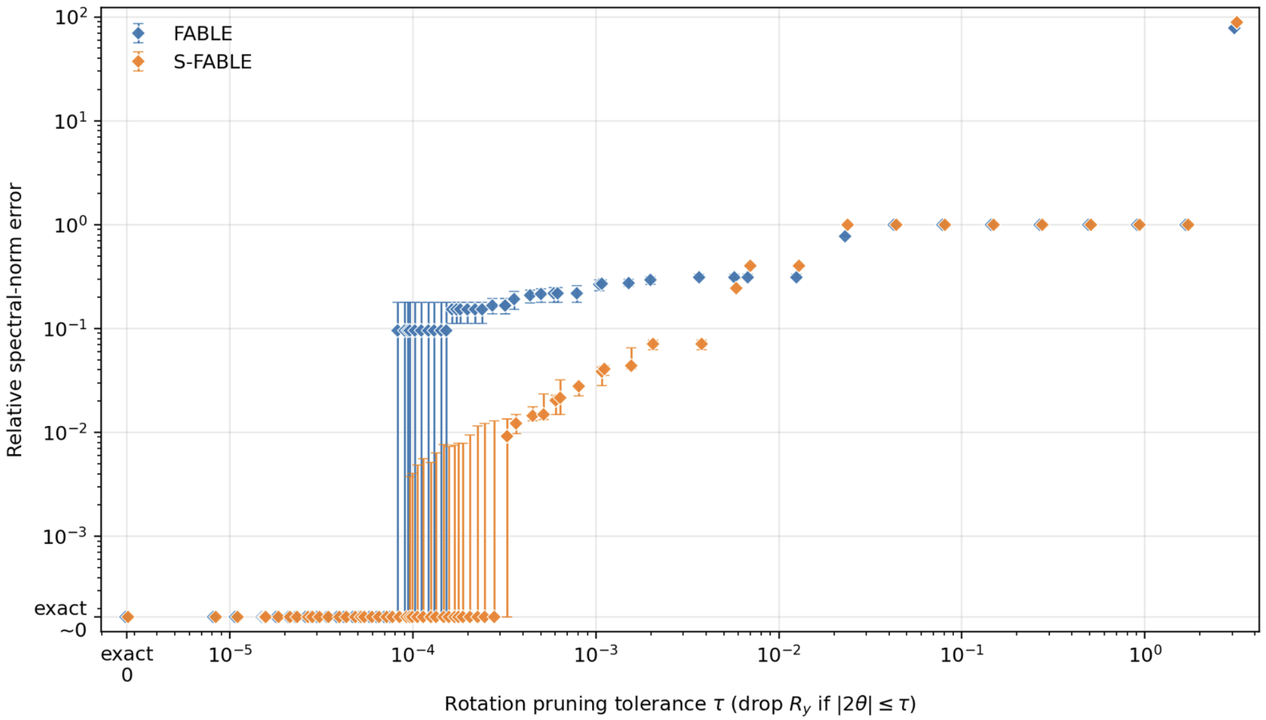
Gate counts for FABLE and S-FABLE

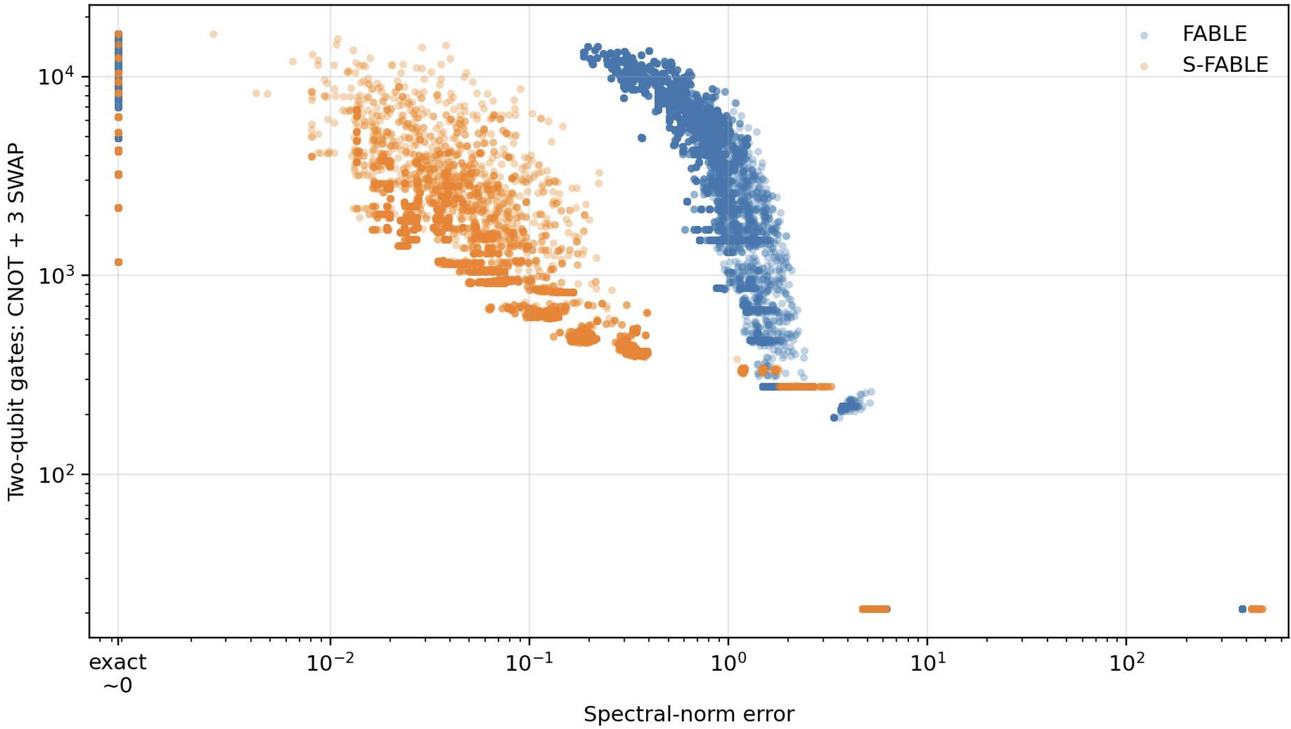




FABLE and S-FABLE with increasing rotation pruning tolerance

128×128





Postselection for FABLE/S-FABLE

- When taking inner product $\langle 0|_a \langle 0^n|_W \otimes I_J |\Psi\rangle$, you "throw away" all bad branches

$$U|0\rangle_a|0^n\rangle_W|\Psi\rangle_J = \frac{1}{N}\underbrace{|0\rangle_a|0^n\rangle_W \tilde{A}|\Psi\rangle_J}_{\substack{a \text{ ancilla qubits,} \\ \text{good branch}}} + \sqrt{1 - \left\|\tilde{A}|\Psi\rangle_J\right\|^2}\ \underbrace{|\sigma_\Psi^\perp\rangle}_{\substack{\text{everything else,} \\ \text{bad branch}}}$$

- The success probability is

$$p_\Psi = \left\|\tilde{A}|\Psi\rangle_J\right\|^2 = \frac{\left\|A|\Psi\rangle_J\right\|^2}{\beta^2 N^2} \underbrace{=}_{A \text{ is Hermitian}} \frac{\langle\Psi|A^2|\Psi\rangle_J}{\alpha^2 N^2}$$

- For our $A = 3I - B$, FABLE uses $\beta_F = 3$ and $\beta_S = \max\left|(A_H)_{ij}\right|$. For example,

$$p_\Psi^{FABLE} = \frac{\langle\Psi|(3I - B)^2|\Psi\rangle_J}{9N^2}$$

Post-selection probability

- The post-selection probability depends on $|\psi\rangle$
- Define

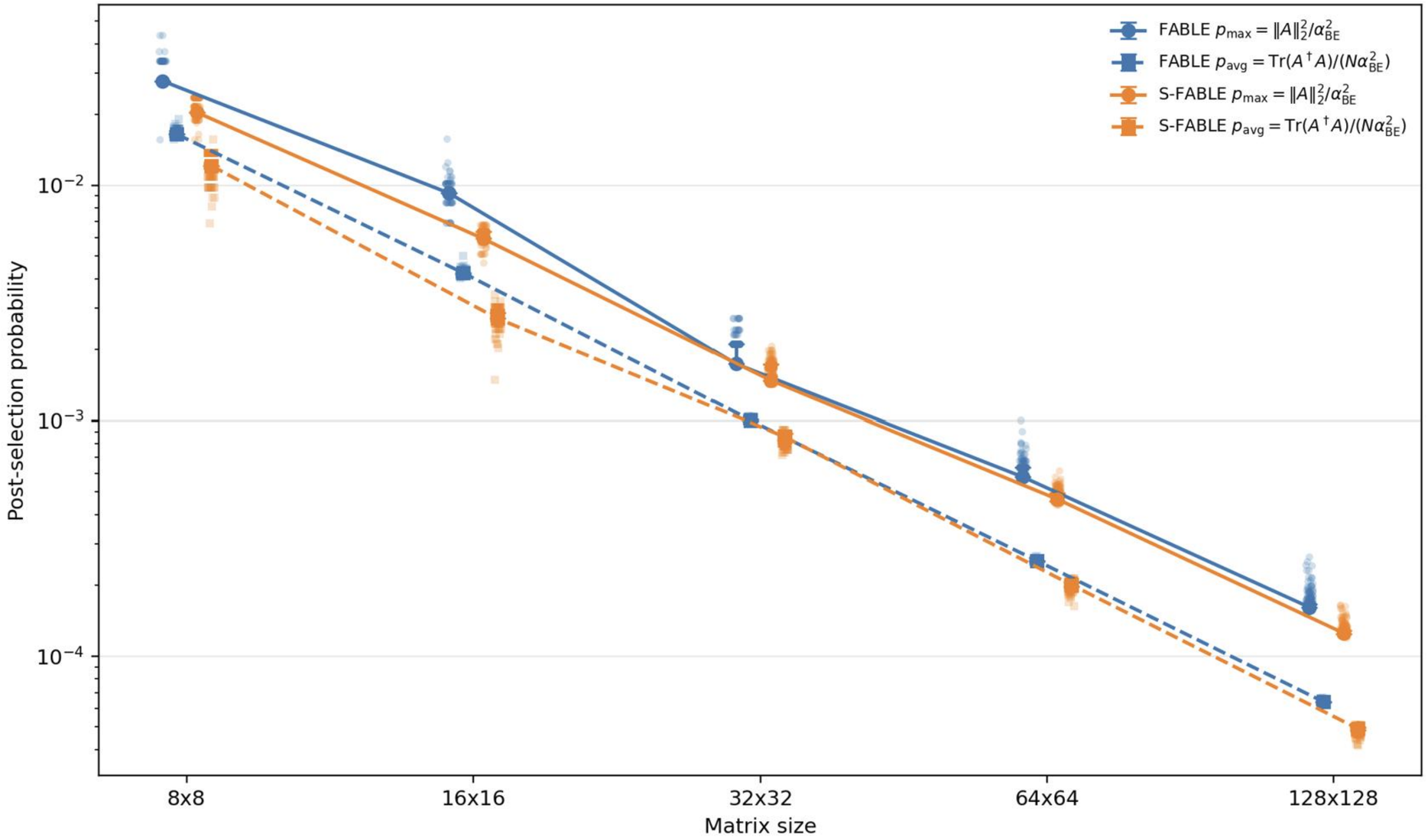
$$p_{max} = \max_{||\psi||=1} \frac{\left\| A|\Psi\rangle_J \right\|^2}{\alpha^2}$$

$$= \frac{\|A\|^2}{\alpha^2}$$

And

$$p_{avg} = \mathbb{E}_{|\psi\rangle}[p_{succ}(|\psi\rangle)]$$

$$= \frac{Tr(A^\dagger A)}{N\alpha^2}$$



Block encoding with LCU

- Liner combination of unitaries (LCU) decomposition of matrix A

$$A = \sum_{k=0}^{N-1} \alpha_k U_k \,, \qquad \text{with } \alpha_k \text{ being real positive coeffs}$$

- Define the prepare operator (which prepares a state whose amplitudes are determined by the coefficients of the LCU):

$$\text{PREP}|0\rangle = \sum_{k=0}^{N-1} \sqrt{\frac{|\alpha_k|}{\lambda}} |k\rangle \,, \qquad \lambda = \sum_k |\alpha_k|$$

- The select operator (which selects which unitary is applied):

$$\text{SEL}|k\rangle|\psi\rangle = |k\rangle U_k |\psi\rangle$$

Block encoding with LCU

- Liner combination of unitaries (LCU) decomposition of matrix A

$$A = \sum_{k=0}^{N-1} \alpha_k U_k \,, \qquad \text{with } \alpha_k \text{ being real positive coeffs}$$

- Define the prepare operator (which prepares a state whose amplitudes are determined by the coefficients of the LCU):

$$\text{PREP}|0\rangle = \sum_{k=0}^{N-1} \sqrt{\frac{|\alpha_k|}{\lambda}} |k\rangle \,, \qquad \lambda = \sum_{k} |\alpha_k|$$

- The select operator (which selects which unitary is applied):

$$\text{SEL}|k\rangle|\psi\rangle = |k\rangle U_k |\psi\rangle$$

- Combine PREP and SEL operator:

$$\langle 0|\mathbf{PREP}^{\dagger}\text{SEL}\,\boldsymbol{PREP}|0\rangle|\psi\rangle = \sum_{k,k'} \left\langle k' \middle| \sqrt{\frac{|\alpha_{k'}|}{\boldsymbol{\lambda}}}\,\text{SEL}\sqrt{\frac{|\alpha_k|}{\boldsymbol{\lambda}}} \middle| k \right\rangle |\psi\rangle$$

Block encoding with LCU

- Liner combination of unitaries (LCU) decomposition of matrix A

$$A = \sum_{k=0}^{N-1} \alpha_k U_k \,, \qquad \text{with } \alpha_k \text{ being real positive coeffs}$$

- Define the prepare operator (which prepares a state whose amplitudes are determined by the coefficients of the LCU):

$$\text{PREP}|0\rangle = \sum_{k=0}^{N-1} \sqrt{\frac{|\alpha_k|}{\lambda}} |k\rangle \,, \qquad \lambda = \sum_k |\alpha_k|$$

- The select operator (which selects which unitary is applied):

$$\text{SEL}|k\rangle|\psi\rangle = |k\rangle U_k |\psi\rangle$$

- Combine PREP and SEL operator:

$$\langle 0|\mathbf{PREP}^\dagger \text{SEL}\, \boldsymbol{PREP}|0\rangle|\psi\rangle = \sum_{k,k'} \left\langle k' \middle| \sqrt{\frac{|\alpha_{k'}|}{\boldsymbol{\lambda}}}\, \text{SEL} \sqrt{\frac{|\alpha_k|}{\boldsymbol{\lambda}}} \middle| k \right\rangle |\psi\rangle = \sum_{k,k'} \left\langle k' \middle| \frac{\sqrt{|\boldsymbol{\alpha_{k'}}||\boldsymbol{\alpha_k}|}}{\boldsymbol{\lambda}}\, \text{SEL} \middle| k \right\rangle |\psi\rangle$$

Block encoding with LCU

- Liner combination of unitaries (LCU) decomposition of matrix A

$$A = \sum_{k=0}^{N-1} \alpha_k U_k \,, \qquad \text{with } \alpha_k \text{ being real positive coeffs}$$

- Define the prepare operator (which prepares a state whose amplitudes are determined by the coefficients of the LCU):

$$\text{PREP}|0\rangle = \sum_{k=0}^{N-1} \sqrt{\frac{|\alpha_k|}{\lambda}} |k\rangle \,, \qquad \lambda = \sum_k |\alpha_k|$$

- The select operator (which selects which unitary is applied):

$$\text{SEL}|k\rangle|\psi\rangle = |k\rangle U_k |\psi\rangle$$

- Combine PREP and SEL operator:

$$\langle 0|\mathbf{PREP}^\dagger \boldsymbol{SEL\ PREP}|0\rangle|\psi\rangle = \sum_{k,k'} \left\langle k' \middle| \sqrt{\frac{|\alpha_{k'}|}{\boldsymbol{\lambda}}}\, \text{SEL} \sqrt{\frac{|\alpha_k|}{\boldsymbol{\lambda}}} \middle| k \right\rangle |\psi\rangle = \sum_{k,k'} \left\langle k' \middle| \frac{\sqrt{|\alpha_{k'}||\alpha_k|}}{\boldsymbol{\lambda}}\, \text{SEL} \middle| k \right\rangle |\psi\rangle$$

$$= \sum_{k,k'} \left\langle k' \middle| \frac{\sqrt{|\alpha_{k'}||\alpha_k|}}{\boldsymbol{\lambda}} \middle| k \right\rangle \boldsymbol{U_k}\, |\psi\rangle$$

Block encoding with LCU

- Liner combination of unitaries (LCU) decomposition of matrix A

$$A = \sum_{k=0}^{N-1} \alpha_k U_k, \qquad \text{with } \alpha_k \text{ being real positive coeffs}$$

- Define the prepare operator (which prepares a state whose amplitudes are determined by the coefficients of the LCU):

$$\text{PREP}|0\rangle = \sum_{k=0}^{N-1} \sqrt{\frac{|\alpha_k|}{\lambda}}|k\rangle, \qquad \lambda = \sum_k |\alpha_k|$$

- The select operator (which selects which unitary is applied):

$$\text{SEL}|k\rangle|\psi\rangle = |k\rangle U_k|\psi\rangle$$

- Combine PREP and SEL operator:

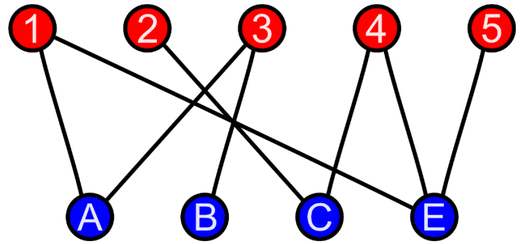
$$\langle 0|\text{PREP}^\dagger \text{SEL PREP}|0\rangle|\psi\rangle = \sum_{k,k'} \left\langle k' \middle| \sqrt{\frac{|\alpha_{k'}|}{\lambda}}\,\text{SEL}\,\sqrt{\frac{|\alpha_k|}{\lambda}} \middle| k \right\rangle |\psi\rangle = \sum_{k,k'} \left\langle k' \middle| \frac{\sqrt{|\alpha_{k'}||\alpha_k|}}{\lambda}\,\text{SEL} \middle| k \right\rangle |\psi\rangle$$

$$= \sum_{k,k'} \left\langle k' \middle| \frac{\sqrt{|\alpha_{k'}||\alpha_k|}}{\lambda} \middle| k \right\rangle U_k |\psi\rangle = \sum_k \frac{\alpha_k U_k}{\lambda}|\psi\rangle = \frac{A}{\lambda}|\psi\rangle$$

Block encoding with LCU

Our matrix represent bipartite graphs $A = 3I - \sum_{c=1}^{\chi} P_c$

where each P_c is a (partial-)permutation-like 0/1 matrix corresponding to one color in our compatibility graph*.



- This means we can decompose our matrix A into colour groups with 1- and 2-cycle transpositions:

$$U_1 = \{(0,2),(1,4),(3,6),(5),(7)\ldots\}, \qquad U_2 = \{(0,5),(2,3),(1),(4),(6),(7)\ldots\},\ldots$$

- The actual permutation that we want to implement is, e.g.

$$P_2 = U_2 - |1\rangle\langle 1| - |4\rangle\langle 4| - |6\rangle\langle 6| - |7\rangle\langle 7| - \cdots$$

- For the SEL operator $\mathrm{SEL}|c\rangle|\psi\rangle = |c\rangle \otimes (-P_c)|\psi\rangle$, we encode the selector labels as:

$$|00\ldots00\rangle_s \to I, \qquad |00\ldots01\rangle_s \to P_1, \qquad |00\ldots10\rangle_s \to P_2, \qquad |\mathrm{REST}\rangle_s \to \text{diagonal } Z - \text{strings/ununsed}$$

* The minimum edge-colouring can be found using Kőnig's theorem in polynomial time

Block encoding with LCU

- Two ways to implement $S_{\text{perm}} = \sum_{\in \text{perm}} |\ell\rangle\langle\ell| \otimes U_\ell$ to avoid explosion of multi-controlled gates

A. Global selector–data permutation

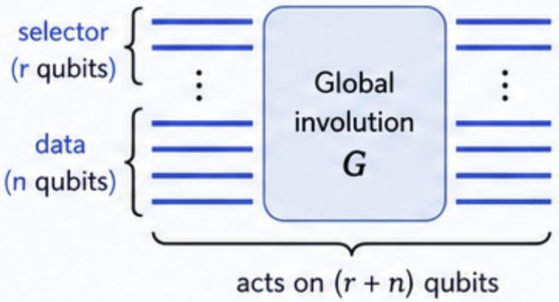
1 Lift each branch swap

$(u \leftrightarrow v)$ in branch ℓ becomes
$((\ell, u) \leftrightarrow (\ell, v))$ on selector + data

2 Compile one global involution

$G\,|\ell\rangle\,|x\rangle = |\ell\rangle\,|U_\ell\,x\rangle$

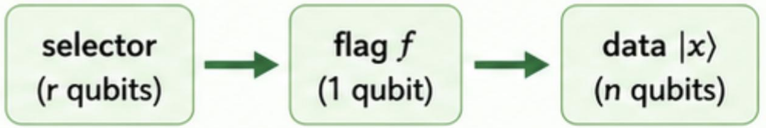
Cycle compiler acts on
$n + r$ qubits.



B. Branch-flag controlled permutation

1 Compute a reusable branch flag

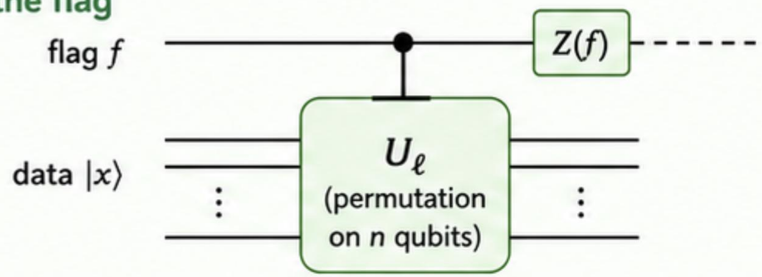
$f \oplus= [\text{selector} = \ell]$
The expensive equality check
is paid once per branch.



2 Apply U_ℓ controlled by the flag

if $f = 1$: $|x\rangle \rightarrow U_\ell\,|x\rangle$

optional $Z(f)$ absorbs
the – sign for
permutation branches.



Global: $\text{Cost}(\text{SELECT}) = \text{Cost}(U_{\text{perm,global}}) + \text{Cost}(U_{\text{sign,perm}}) + \text{Cost}(U_{\text{diag, flag}})$

Branch flag: $\text{Cost}(\text{SELECT}) = \text{Cost}(U_{\text{perm/sign, branch flag}}) + \text{Cost}(U_{\text{diag,flag}})$

Block encoding with LCU

- Canonical state idea:

1. choose a reversible map R that sends difficult pairs **to** easy canonical pairs;
2. synthesize U_{can} from controlled target-bit flips;
3. undo R.

1. Start with two arbitrary transposition pairs

Example involution

$$U = (0011 \leftrightarrow 1111)(0101 \leftrightarrow 1101)$$

1 **Pair 1:** $|0011\rangle \longleftrightarrow |1111\rangle$

2 **Pair 2:** $|0101\rangle \longleftrightarrow |1101\rangle$

All other basis states are unchanged.

Goal: realize this permutation using a standard canonical transposition pattern.

2. Rotate to a canonical pair structure

Choose a reversible relabeling (conjugation) operator R

$$R : |0011\rangle \to |0000\rangle, \quad |1111\rangle \to |0001\rangle$$
$$R : |0101\rangle \to |0010\rangle, \quad |1101\rangle \to |0011\rangle$$

Canonical involution after relabeling

$$U_{\text{can}} = (0000 \leftrightarrow 0001)(0010 \leftrightarrow 0011)$$

After relabeling, each pair differs only in the last bit. The shared prefix becomes the control pattern.

Arbitrary pairs (hard) → R → Canonical pairs (easy)

$$U = R^\dagger U_{\text{can}} R$$

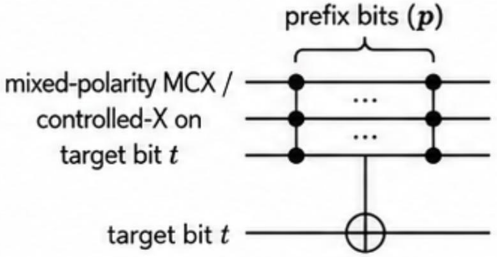
Rotate in, apply the easy canonical permutation, rotate back.

3. How a canonical transposition works

A single canonical pair (abstract)

$|p0\rangle \longleftrightarrow |p1\rangle$ where p is a shared prefix.

A canonical pair differs only in one target bit.



State evolution

$|p0\rangle \to |p1\rangle$

$|p1\rangle \to |p0\rangle$

$|x\rangle \to |x\rangle$ for all x not in the pair.

Our two canonical pairs after relabeling

$|0000\rangle \longleftrightarrow |0001\rangle$ (prefix 000)

$|0010\rangle \longleftrightarrow |0011\rangle$ (prefix 001)

Each canonical pair is implemented by flipping the last bit conditioned on the prefix.

For odd number of 2-cycle transpositions, we need an extra ancilla qubit to make the # of 2-cycle even

Almost-Optimal Computational Basis State Transpositions

Steven Herbert[1,2], Julien Sorci[1], and Yao Tang[1]

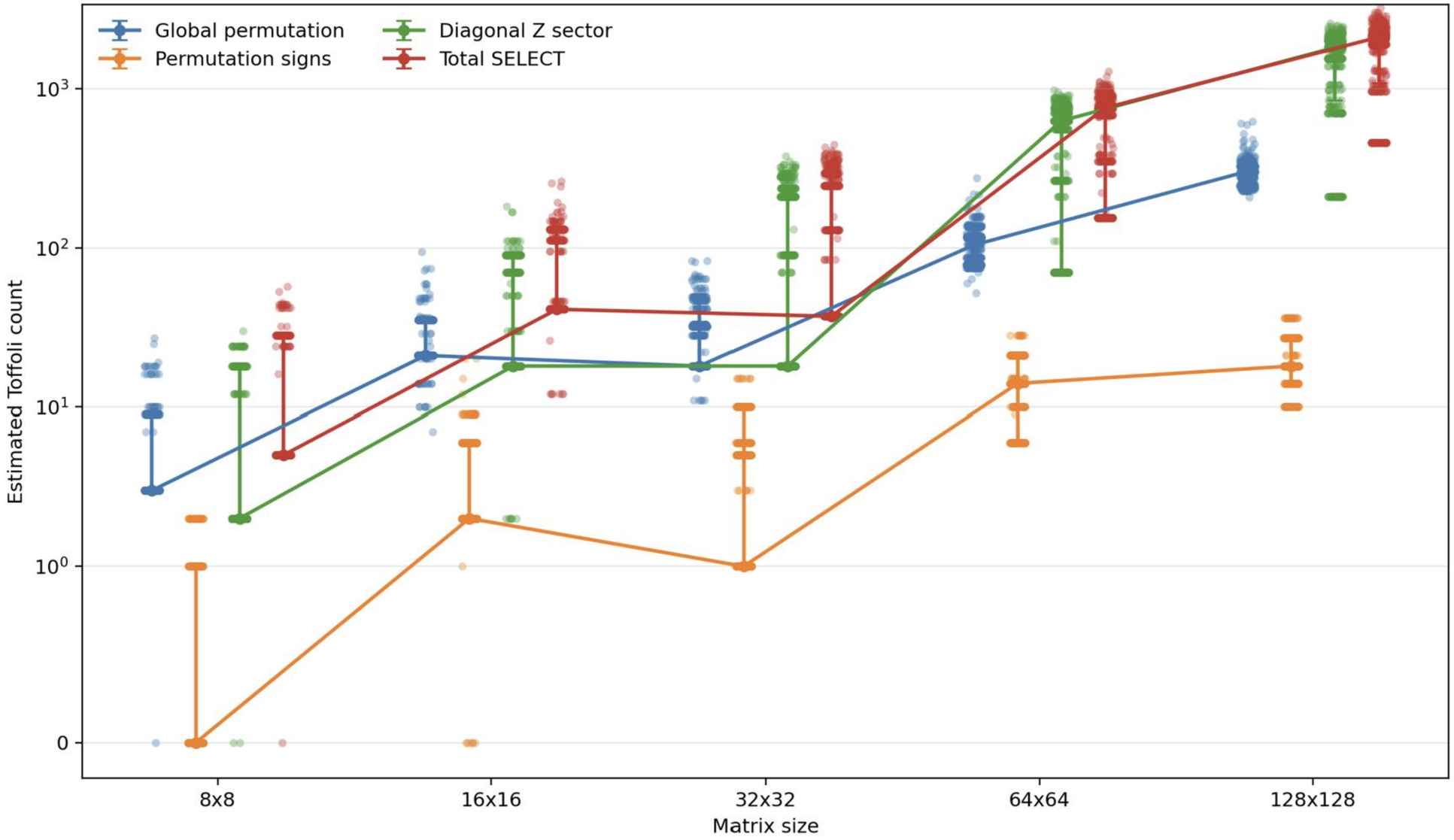
[1] *Quantinuum, Terrington House, 13–15 Hills Road, Cambridge CB2 1NL, United Kingdom*
[2] *Department of Computer Science and Technology, University of Cambridge, United Kingdom*

https://arxiv.org/pdf/2309.12820

Block encoding with LCU

$N_{Toffoli}^{SELECT}$ comes from decomposed multi−controlled SELECT gates)

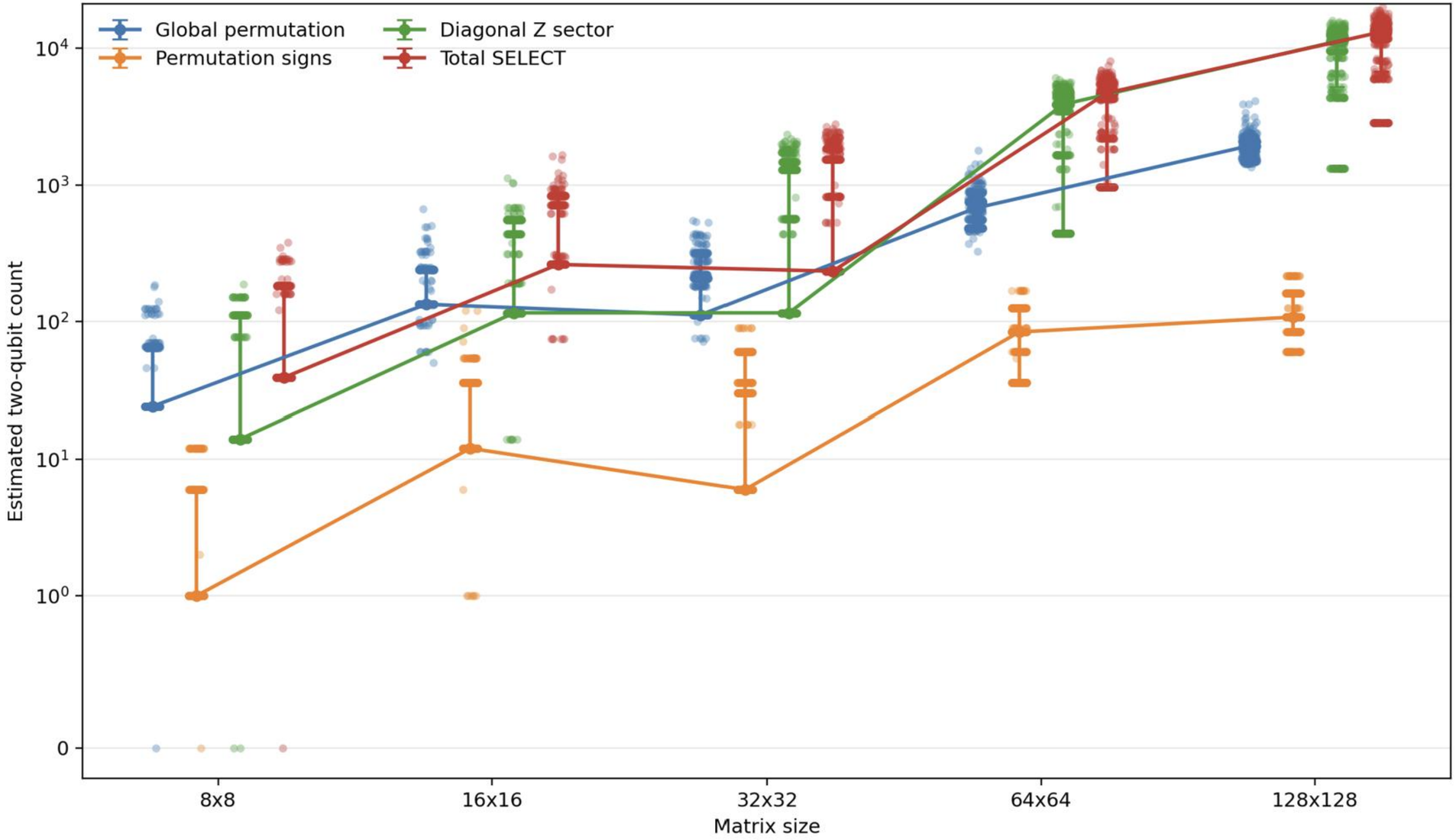
k = 2 controls: 1 Toffoli
k >= 3 controls:
assume that we have enough clean ancillas: 2k - 3 Toffoli



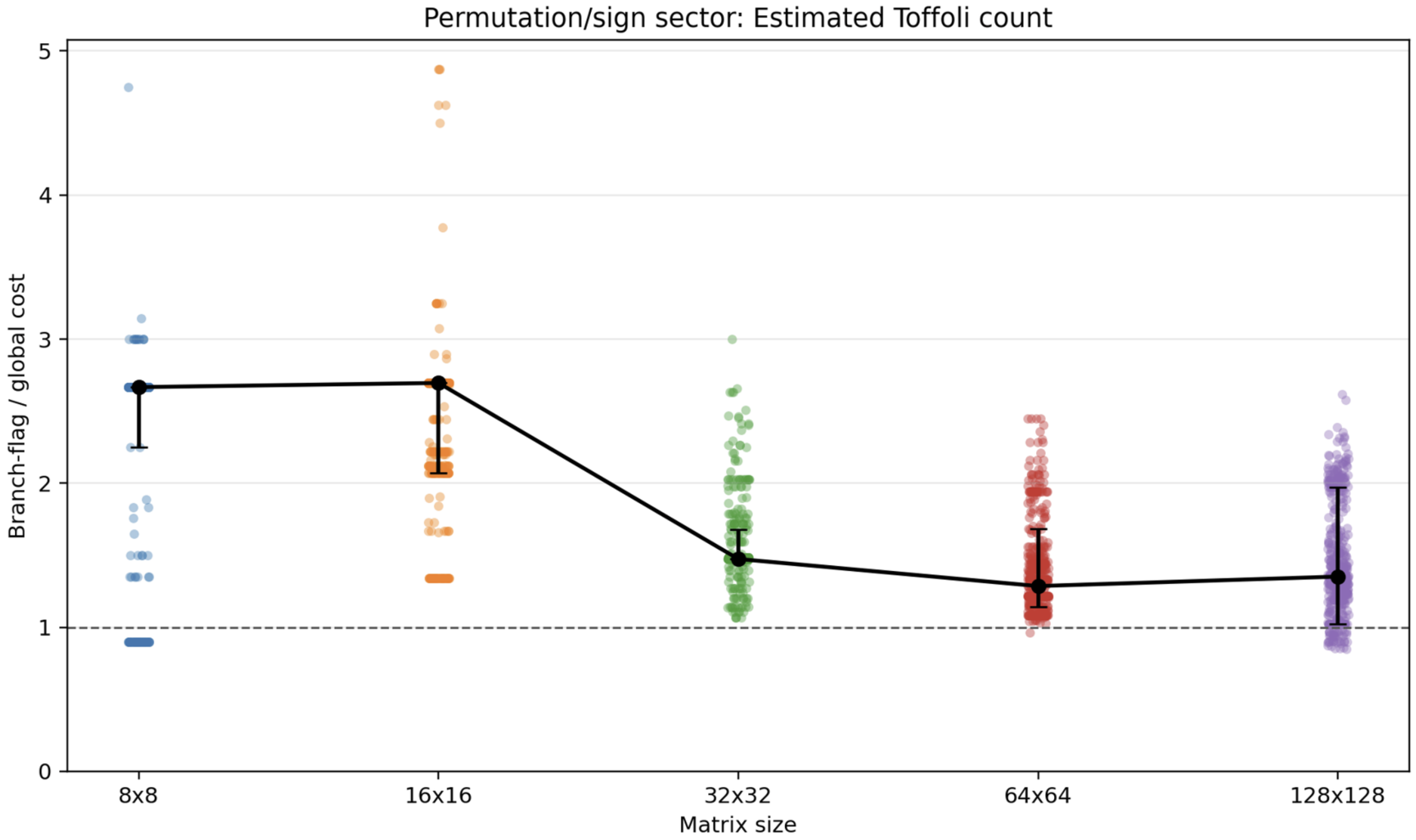
Block encoding with LCU

Two sources of two-qubit gates:

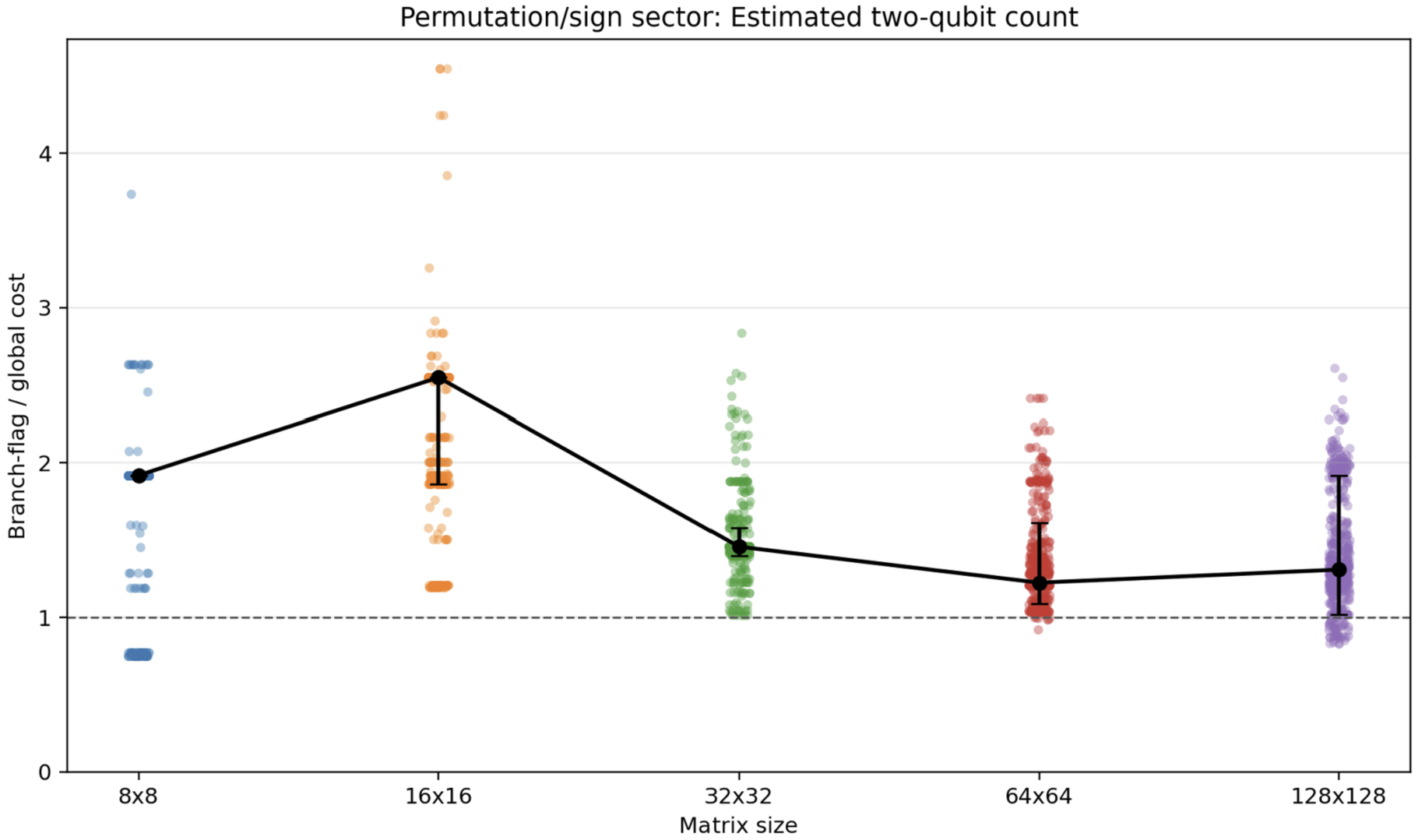
- $N_{CNOT/CZ}^{SELECT}$: extra CNOT from permutation/sign/flag logic and CZ from diagonal Z-strings
- $6N_{Toffoli}^{SELECT}$ (six CNOT-equivalents per standard Toffoli)



Block encoding with LCU



Block encoding with LCU



Comparison between FABLE/S-FABLE and LCU

Resource category	FABLE/S-FABLE	LCU PREP-SELECT-PREP
Two-qubit gates	N_{CNOT} (FABLE Gray-code oracle) $+3N_{SWAP}$ (FABLE register swap $i \leftrightarrow j$, $1SWAP \approx 3CNOT$)	$2N_{CNOT}^{PREP}$ (Möttönen PREP and $\mathrm{PREP}^{\dagger}$) $+N_{CNOT/CZ}^{SELECT}$ (extra CNOT from permutation/sign/flag logic and CZ from diagonal Z-strings) $+6N_{Toffoli}^{SELECT}$ (six CNOT-equivalents per standard Toffoli)

Comparison between FABLE/S-FABLE and LCU

Resource category	FABLE/S-FABLE	LCU PREP-SELECT-PREP
Two-qubit gates	N_{CNOT} (FABLE Gray-code oracle) $+3N_{SWAP}$ (FABLE register swap $i \leftrightarrow j, 1SWAP \approx 3CNOT$)	$2N_{CNOT}^{PREP}$ (Möttönen PREP and $\mathrm{PREP}^{\dagger}$) $+N_{CNOT/CZ}^{SELECT}$ (extra CNOT from permutation/sign/flag logic and CZ from diagonal Z-strings) $+6N_{Toffoli}^{SELECT}$ (six CNOT-equivalents per standard Toffoli)
Arbitrary rotations	N_{RY} (FABLE data-loading oracle)	$2N_{RY}^{PREP}$ (Möttönen PREP and $\mathrm{PREP}^{\dagger}$)

Comparison between FABLE/S-FABLE and LCU

Resource category	FABLE/S-FABLE	LCU PREP-SELECT-PREP
Two-qubit gates	N_{CNOT} (FABLE Gray-code oracle) $+3N_{SWAP}$ (FABLE register swap $i \leftrightarrow j$, $1SWAP \approx 3CNOT$)	$2N_{CNOT}^{PREP}$ (Möttönen PREP and $\mathrm{PREP}^{\dagger}$) $+N_{CNOT/CZ}^{SELECT}$ (extra CNOT from permutation/sign/flag logic and CZ from diagonal Z-strings) $+6N_{Toffoli}^{SELECT}$ (six CNOT-equivalents per standard Toffoli)
Arbitrary rotations	N_{RY} (FABLE data-loading oracle)	$2N_{RY}^{PREP}$ (Möttönen PREP and $\mathrm{PREP}^{\dagger}$)
One-qubit Clifford	N_H (FABLE index H; S-FABLE extra outer H)	N_X (Mixed-polarity controls in SELECT)

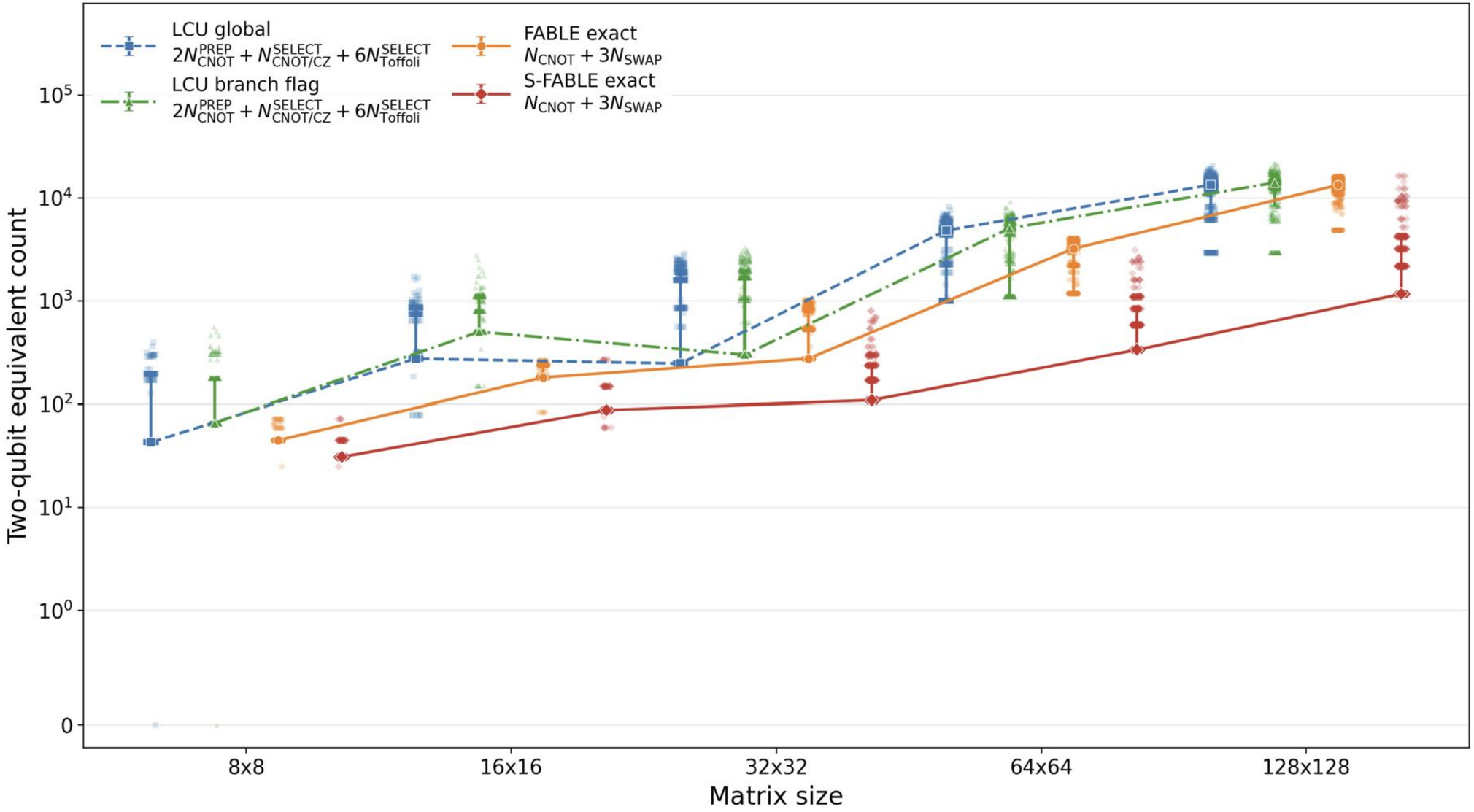
Comparison between FABLE/S-FABLE and LCU

Resource category	FABLE/S-FABLE	LCU PREP-SELECT-PREP
Two-qubit gates	N_{CNOT} (FABLE Gray-code oracle) $+3N_{SWAP}$ (FABLE register swap $i \leftrightarrow j$, $1SWAP \approx 3CNOT$)	$2N_{CNOT}^{PREP}$ (Möttönen PREP and $\mathrm{PREP}^{\dagger}$) $+N_{CNOT/CZ}^{SELECT}$ (extra CNOT from permutation/sign/flag logic and CZ from diagonal Z-strings) $+6N_{Toffoli}^{SELECT}$ (six CNOT-equivalents per standard Toffoli)
Arbitrary rotations	N_{RY} (FABLE data-loading oracle)	$2N_{RY}^{PREP}$ (Möttönen PREP and $\mathrm{PREP}^{\dagger}$)
One-qubit Clifford	N_H (FABLE index H; S-FABLE extra outer H)	N_X (Mixed-polarity controls in SELECT)
Toffoli	none	$N_{Toffoli}^{SELECT}$ (Decomposed multi−controlled SELECT gates) Global: global perm, selector-controlled phase, diagonal selector-flag compute/uncompute Branch-flag: branch-flag permutation/sign, diagonal selector-flag compute/uncompute

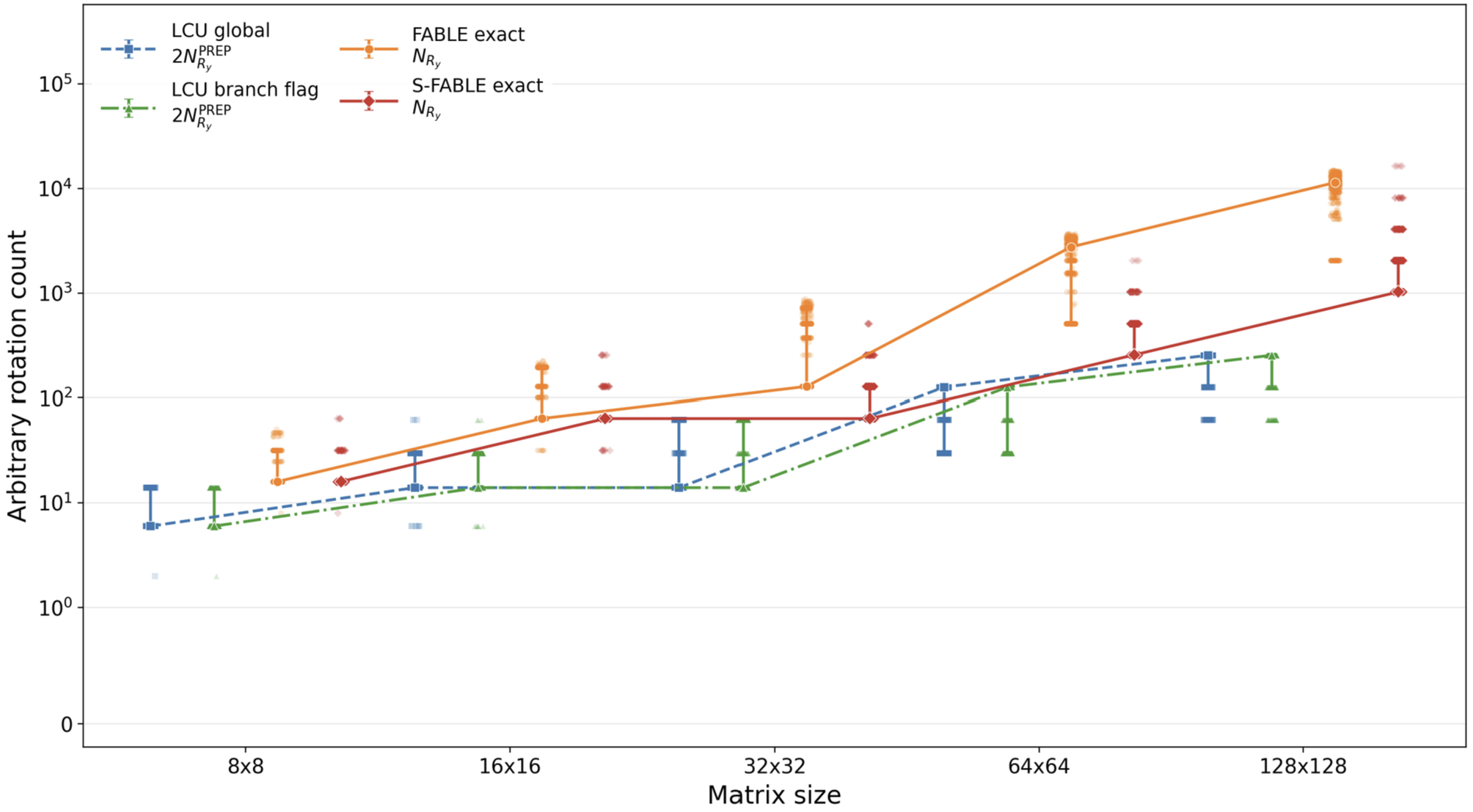
Comparison between FABLE/S-FABLE and LCU

Resource category	FABLE/S-FABLE	LCU PREP-SELECT-PREP
Two-qubit gates	N_{CNOT} (FABLE Gray-code oracle) $+3N_{SWAP}$ (FABLE register swap $i \leftrightarrow j, 1SWAP \approx 3CNOT$)	$2N_{CNOT}^{PREP}$ (Möttönen PREP and $\mathrm{PREP}^{\dagger}$) $+N_{CNOT/CZ}^{SELECT}$ (extra CNOT from permutation/sign/flag logic and CZ from diagonal Z-strings) $+6N_{Toffoli}^{SELECT}$ (six CNOT-equivalents per standard Toffoli)
Arbitrary rotations	N_{RY} (FABLE data-loading oracle)	$2N_{RY}^{PREP}$ (Möttönen PREP and $\mathrm{PREP}^{\dagger}$)
One-qubit Clifford	N_H (FABLE index H; S-FABLE extra outer H)	N_X (Mixed-polarity controls in SELECT)
Toffoli	none	$N_{Toffoli}^{SELECT}$ (Decomposed multi−controlled SELECT gates) Global: global perm, selector-controlled phase, diagonal selector-flag compute/uncompute Branch-flag: branch-flag permutation/sign, diagonal selector-flag compute/uncompute
Post-selection	$\alpha_F = N\beta_F, \alpha_S = N\beta_S$ where $\beta_F = 3, \beta_S = \max\lvert(A_H)_{ij}\rvert$ for our tracking matrix	$\alpha_{LCU} = \sum_k \lvert a_k \rvert$

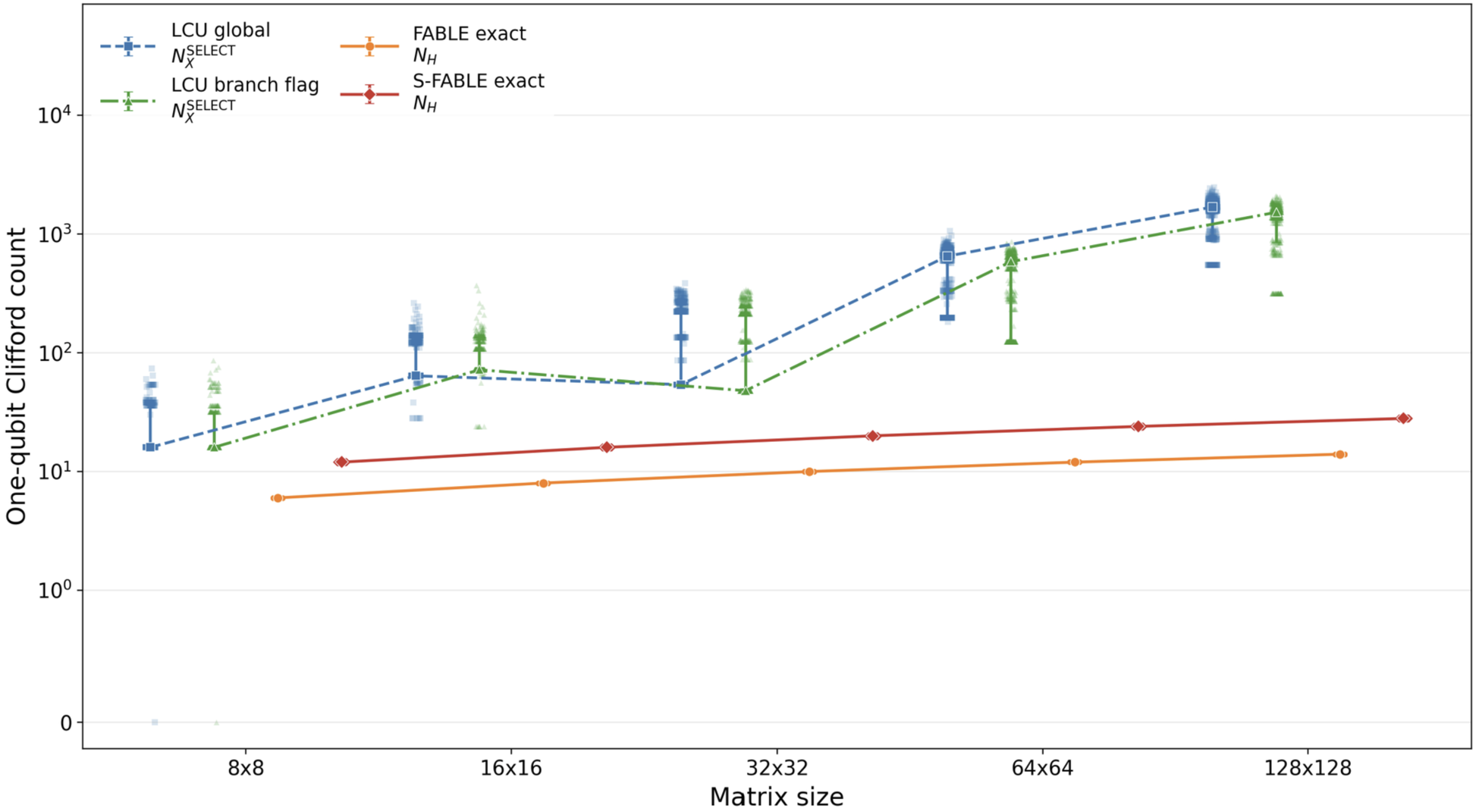
Block encoding with LCU



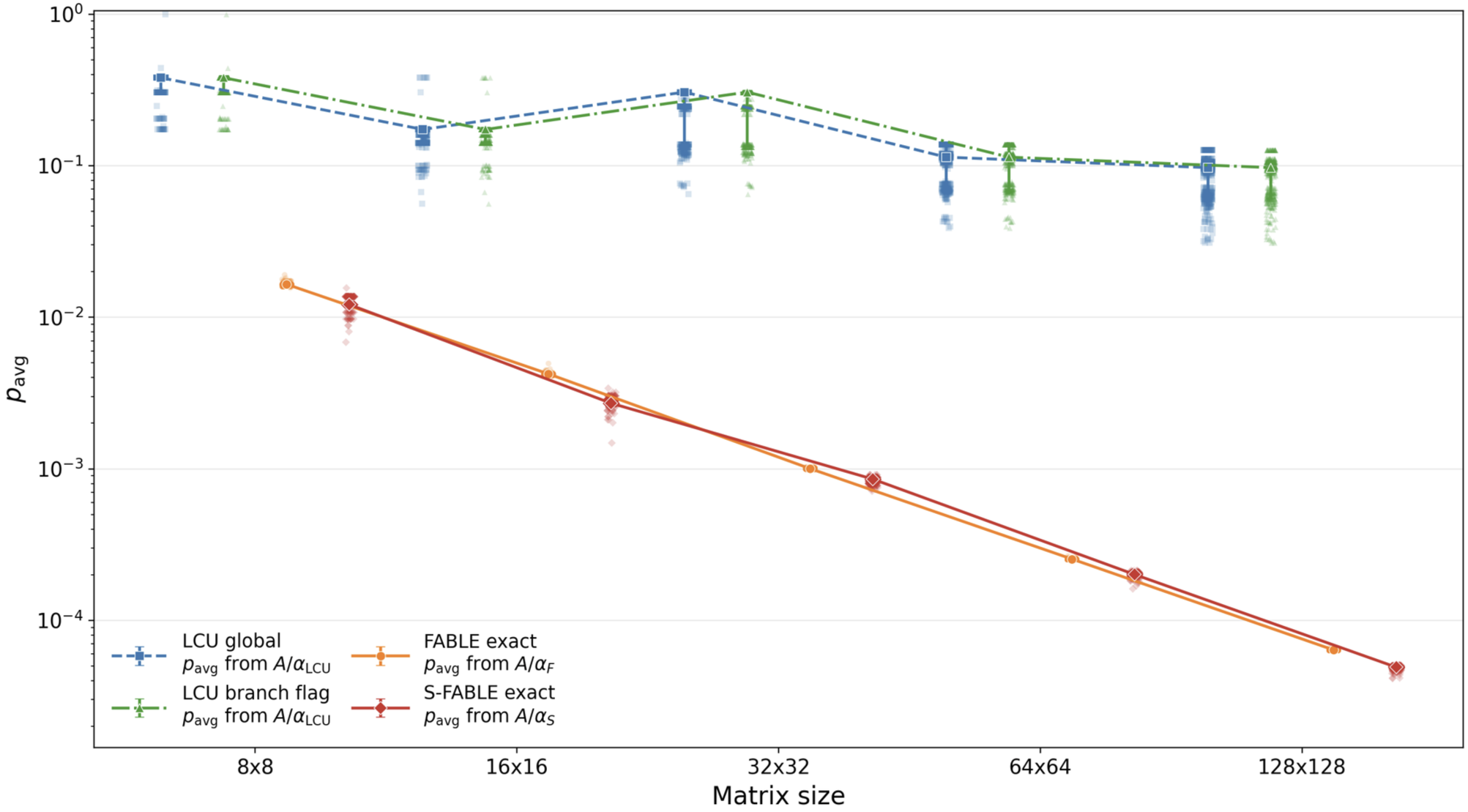
Comparison between FABLE/S-FABLE and LCU



Comparison between FABLE/S-FABLE and LCU



Block encoding with LCU



Conclusions

Method	Gate-count character	Post-selection character	Overall
LCU global permutation	Toffoli/CNOT/CZ-heavy; high SELECT cost	Good $p_{max}/p_{avg} \sim 10^{-1}$ for large matrix size like 128x128	Strong normalization, expensive logic Likely the best LCU implementation here
LCU brahcn-flag	Similar but slightly heavier in permutation/sign sector	Same as global perm	Slightly poorer than global permutation

Conclusions

Method	Gate-count character	Post-selection character	Overall
LCU global permutation	Toffoli/CNOT/CZ-heavy; high SELECT cost	Good $p_{max}/p_{avg} \sim 10^{-1}$ for large matrix size like 128x128	Strong normalization, expensive logic Likely the best LCU implementation here
LCU brahcn-flag	Similar but slightly heavier in permutation/sign sector	Same as global perm	Slightly poorer than global permutation
FABLE exact	Many R_y, many CNOT/SWAP equivalent gates	Poor p_{max}/p_{avg} and scale poorly with matrix size	not attractive versus S-FABLE
S-FABLE exact	Much fewer gates than FABLE	Still poor p_{max}/p_{avg} but slightly better than FABLE	Best FABLE-type circuit, but p_{max}/p_{avg} scales poorly