

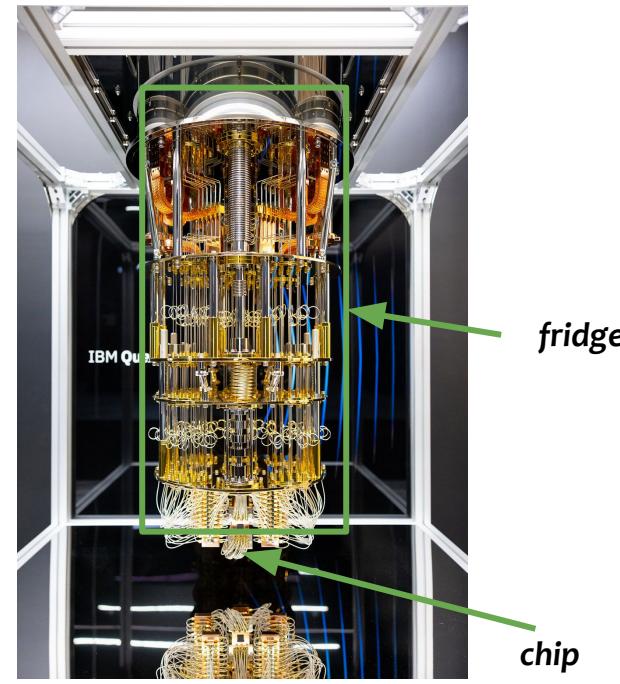
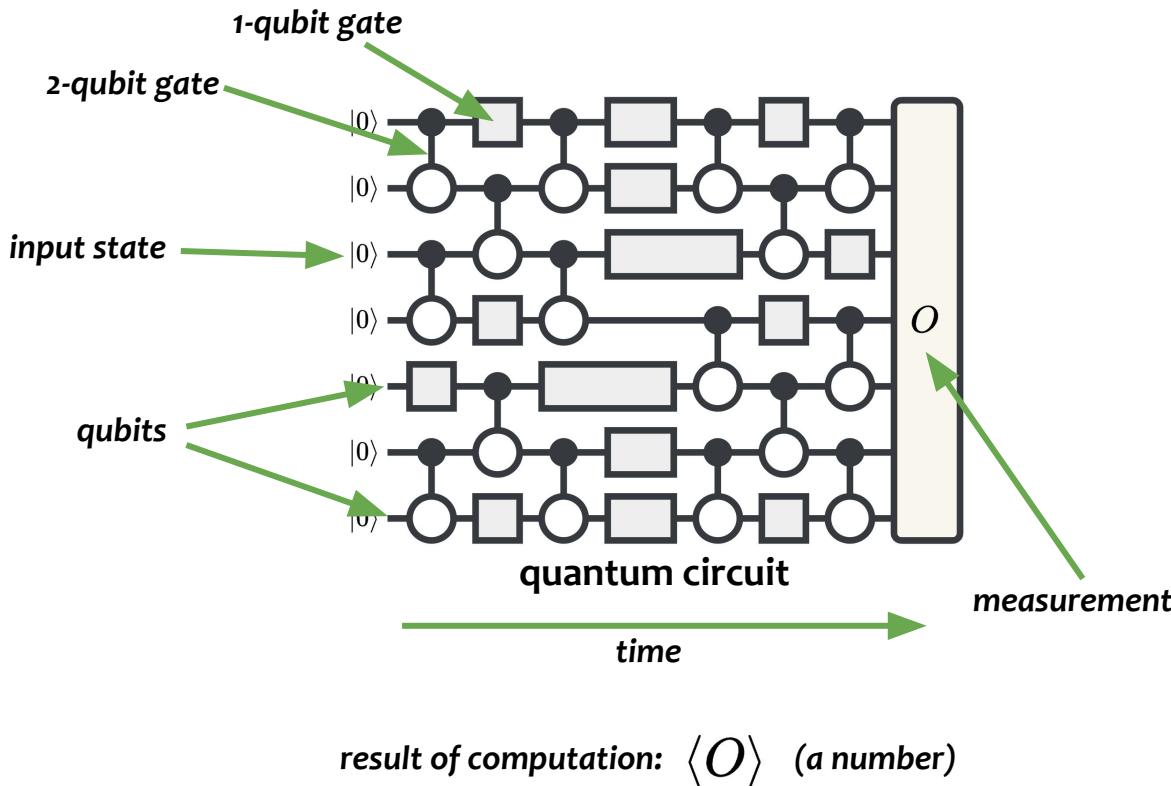
qTPU

Scalable Quantum-Classical Computing via Tensor Networks

Nathaniel Tornow, Christian B. Mendl, Pramod Bhatotia



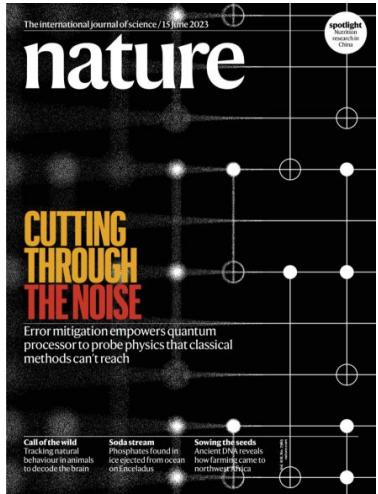
Quantum Circuits 101



QPU (127 qubits)

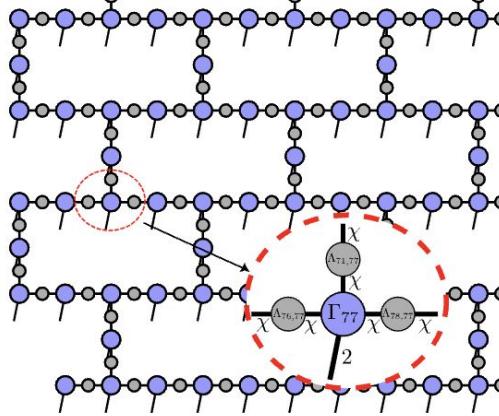
The Race To “Quantum Utility”

Quantum Computing (QC)



vs.

Classical Tensor Network (TN) Methods



QC shows “utility” by running complex experiments on 127-qubit QPUs

Outperformed by **classical tensor network** method on a laptop

Kim, Y., Eddins, A., Anand, S., Wei, K. X., Van Den Berg, E., Rosenblatt, S., ... & Kandala, A. (2023). Evidence for the utility of quantum computing before fault tolerance. *Nature*, 618(7965), 500-505.

Tindall, J., Fishman, M., Stoudenmire, M., & Sels, D. (2023). Efficient tensor network simulation of IBM's kicked Ising experiment. *arXiv preprint arXiv:2306.14887*.

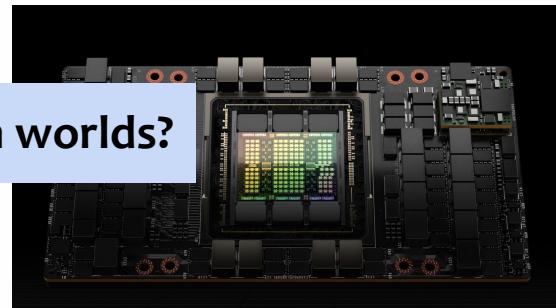
Quantum Computing vs. Classical Methods

Quantum Computing

Classical Simulation (TN)

| | | |
|---------------------|---------------|-------------------------|
| Number of qubits | ✗ | ✓ |
| Fidelity | ✗ | ✓ |
| Scaling/Parallelism | ✗ | ✓ (HPC/ GPU processing) |
| Entanglement | ✓ (unlimited) | ✗ (memory limits) |

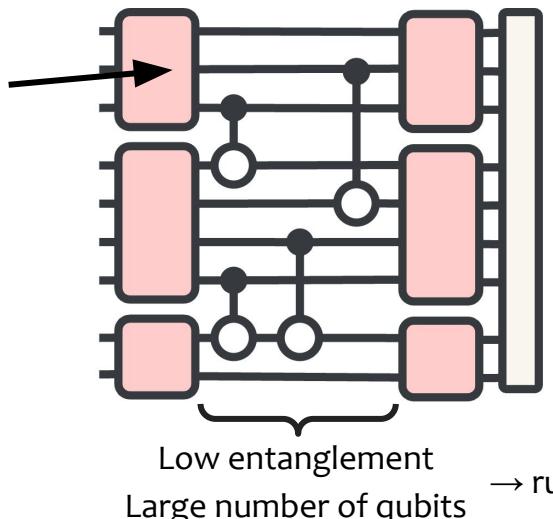
How can we get the benefits of both worlds?



The Goal: Hybrid Processing

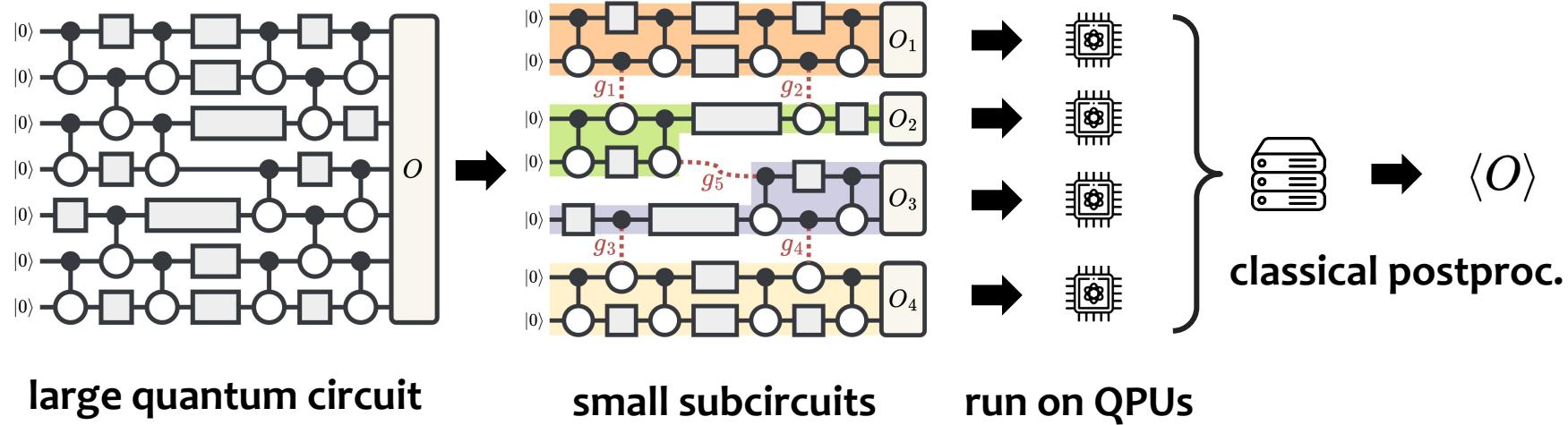
Why not use QPUs and efficient classical TN methods together?

High entanglement
Low number of qubits
→ run on QPU



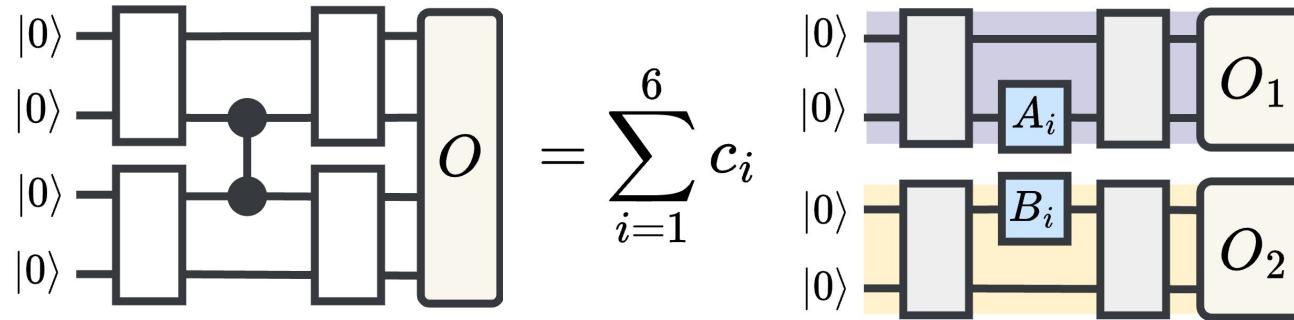
- Number of qubits
- Fidelity
- HPC-Scaling & Parallelism
- Entanglement

A promising Approach: Circuit Knitting



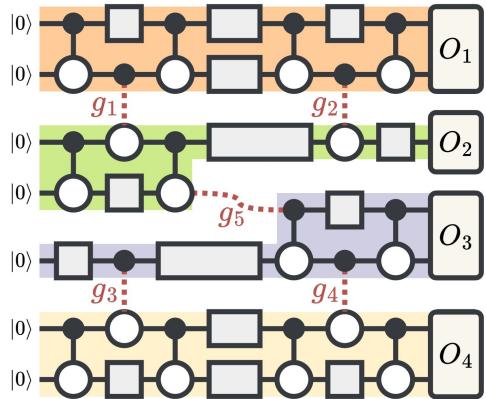
1. **Cut** a large circuit into multiple **smaller subcircuits**
2. Run the subcircuits **independently** on QPUs
3. Use **classical postprocessing** to reconstruct the result

Background: Quasiprobability Decomposition (QPD)



“run two-qubit gate as sum of single-qubit gates”

Background: Circuit Knitting Overheads



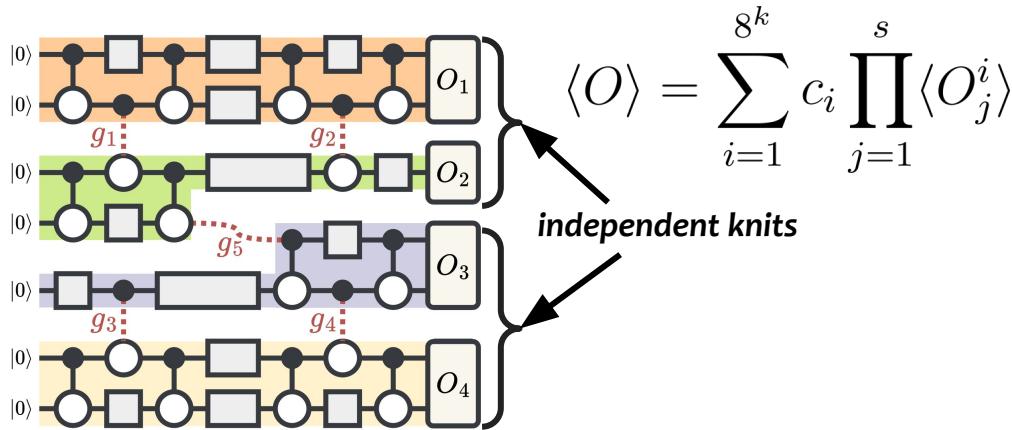
$$\langle O \rangle = \sum_{i=1}^{8^k} c_i \prod_{j=1}^s \langle O_j^i \rangle$$

Postprocessing overhead: $\mathcal{O}(8^k)$ float-operations (FLOPs) (k cuts)

Circuit Knitting exhibits high exponential overheads

Problem: Naive Circuit Knitting

Current circuit knitting use a very naive circuit knitting implementation



Naive implementation:

```

1 res = 0
2 for i in range(8**k):
3     res_i = 1
4     for j in range(s):
5         res_i *= get_subres(i, j)
6     res += res_i

```

Always worst-case postprocessing overhead

Impossible to accelerate using GPUs

Could we use the sparse structure of a circuit to mitigate postprocessing overhead?

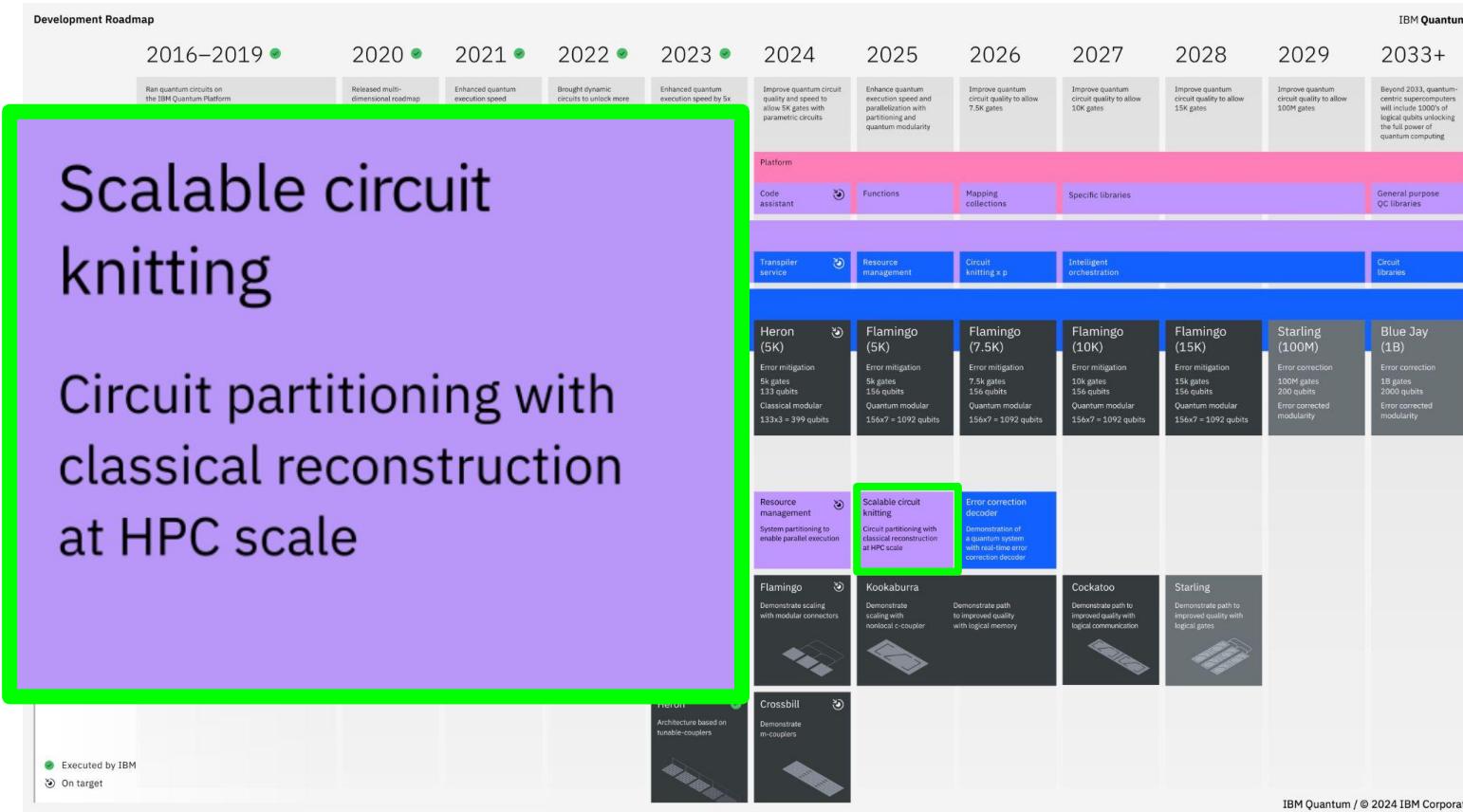
Problem Statement

Recap:

- No one-size-fits-all between QPU-processing and classical TN-based methods
- Possible approach: Circuit knitting
- **But:** prohibitive “brute-force” postprocessing overhead

How can we provide **large-scale** and **GPU-accelerated quantum-classical processing** of quantum circuits using **quantum circuit knitting**?

Relevance: IBM Quantum Roadmap



Outline

- Motivation
- **Challenges and Key Ideas**
- Background: Tensor Networks
- The Key Idea: Hybrid Quantum Circuit Contraction
- The qTPU Compiler
- Implementation and Evaluation

- 1. Enabling efficient quantum-classical processing**

How to provide an efficient quantum-classical processing technique?

- 2. Efficiency and adaptability**

How to transform quantum circuits into an optimized hybrid program?

- 3. Large-scale hybrid processing**

How to scale hybrid processing in using QPUs and classical accelerators (GPUs)

qTPU: Large-scale hybrid quantum-classical processing using tensor networks

Key contributions:

1. Key approach:

Represent circuit knitting as a **hybrid tensor network (h-TN)**

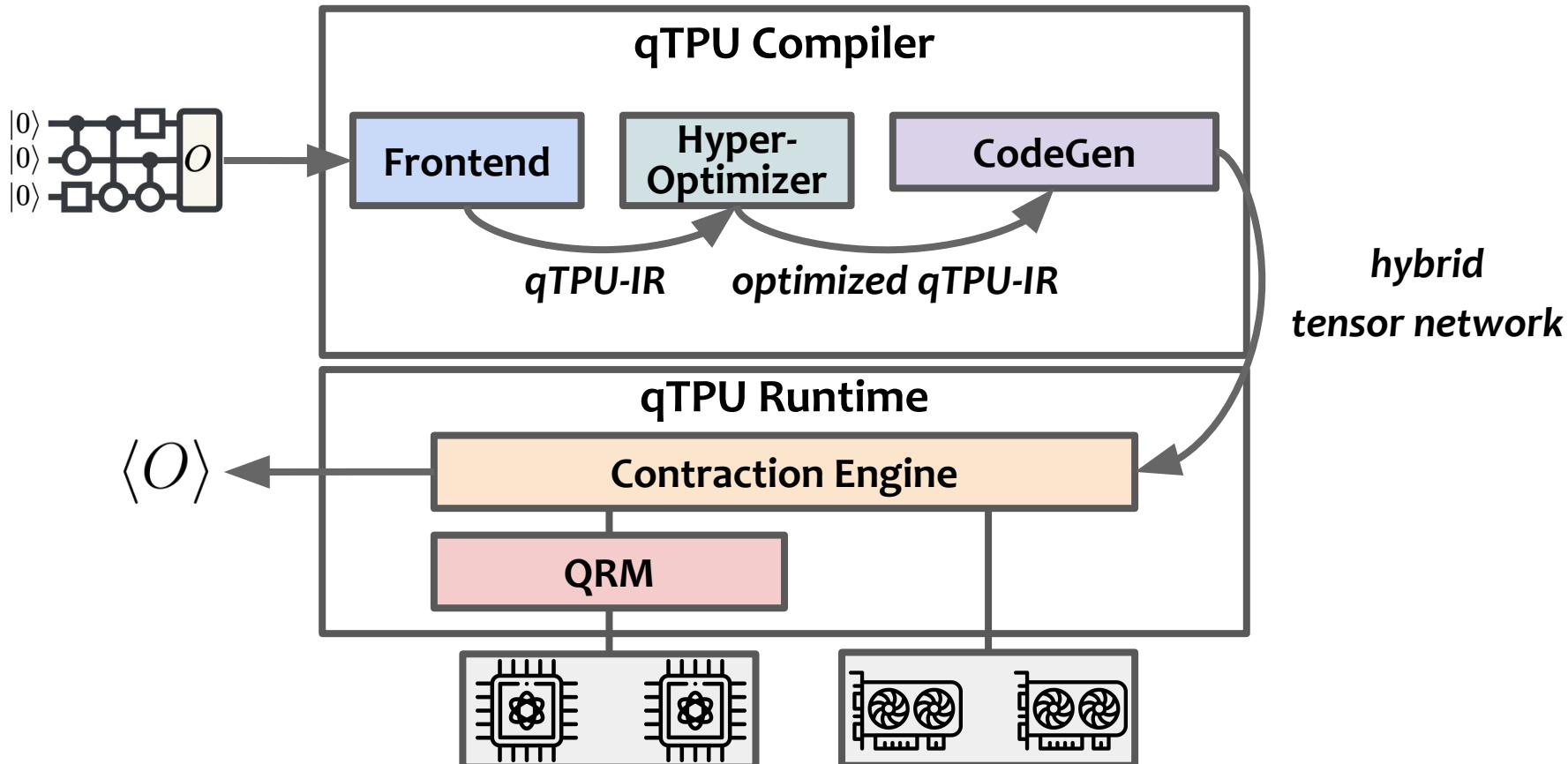
2. Efficiency:

Automatic transformation of circuits to optimized h-TNs (**qTPU Compiler**)

3. Scalability:

Large-scale h-TN contraction using hybrid QPUs and GPUs (**qTPU Runtime**)

qTPU System Overview



Outline

- Motivation
- Challenges and Key Ideas
- **Background: Tensor Networks**
- The Key Idea: Hybrid Tensor Networks
- The qTPU Compiler
- Implementation and Evaluation

Background: Tensor Networks

Tensor = Multidimensional vector



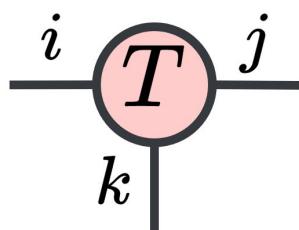
Scalar (0 dimensions)



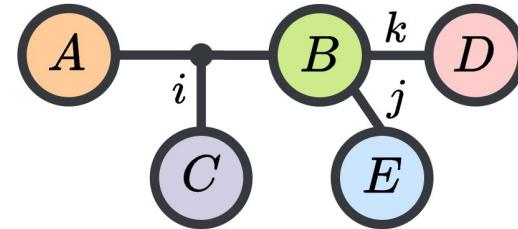
Vector (1 dimension)



Matrix (2 dimensions)



3-dimensional **tensor**

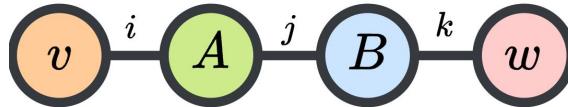


$$= \sum_{ijk} (A)_i (B)_{ijk} (C)_i (D)_k (E)_j$$

tensor network contraction

Tensor Network Contraction Sequence

$$y = v^T \cdot A \cdot B \cdot w \quad v, w \in \mathbb{R}^n \quad A, B \in \mathbb{R}^{n \times n}$$



How to contract this tensor network?

→ Minimize contraction cost (number of float-operations, FLOPS)

Naive (brute-force)

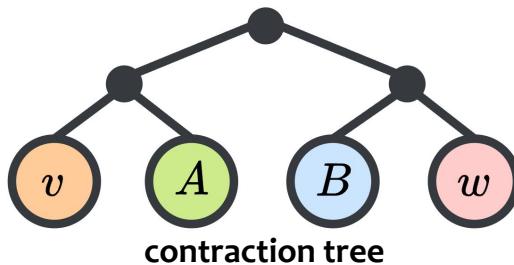
$$y = \sum_{ijk} v_i A_{ij} B_{jk} w_k$$

```

1  y = 0
2  for i, j, k in product(range(n), repeat=3):
3      y += v[i] * A[i, j] * B[j, k] * w[k]

```

Efficient contraction sequence



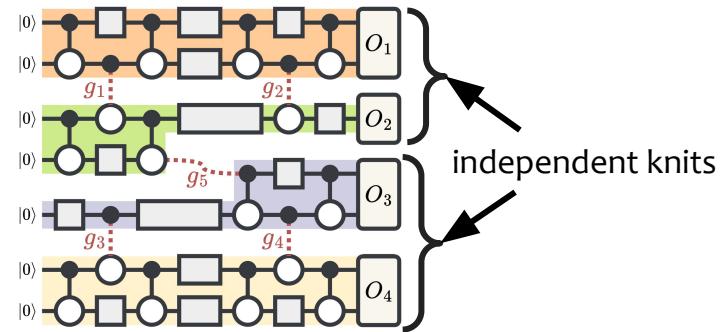
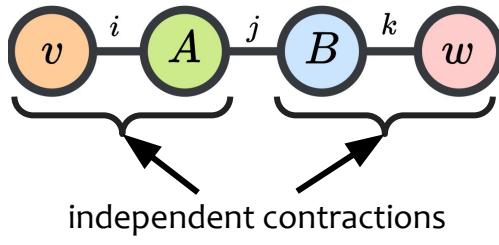
$$\mathcal{O}(n^3)$$

$$\mathcal{O}(n^2)$$

Tensor Network Contraction for Circuit Knitting

Current circuit knitting posprocessing is similar to the **naive contraction** of a TN!

Efficient TN contraction sequences can significantly reduce contraction cost!

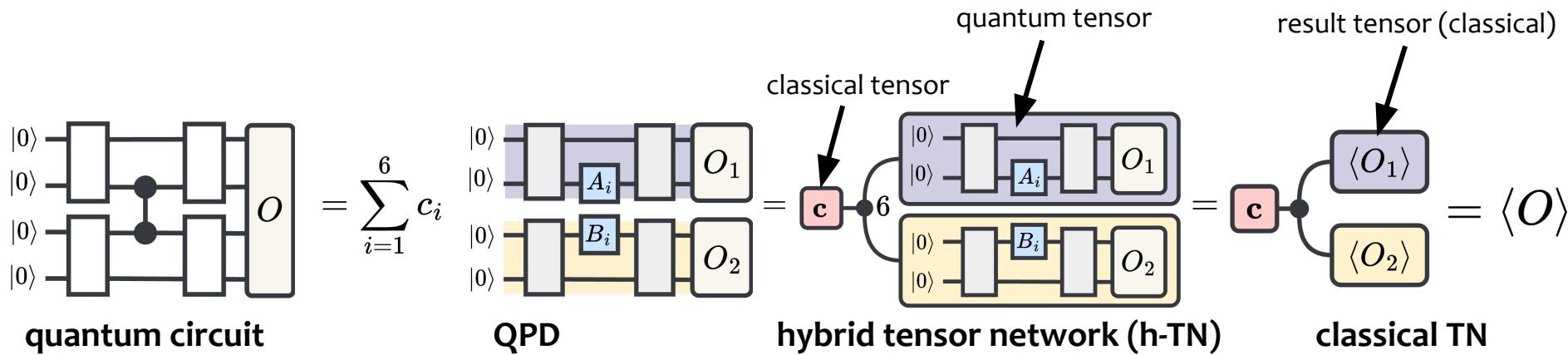


Can we use TN-techniques to significantly reduce postprocessing overhead?

Outline

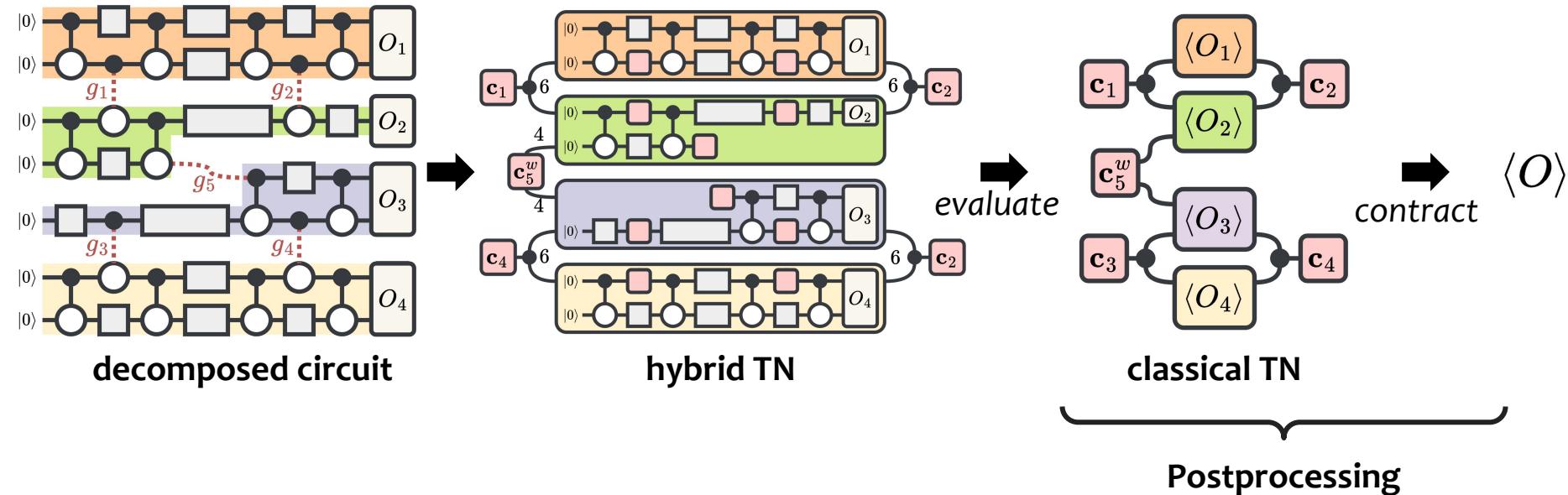
- Motivation
- Challenges and Key Ideas
- Background: Tensor Networks
- **The Key Idea: Hybrid Quantum Circuit Contraction**
- The qTPU Compiler
- Implementation and Evaluation

The Key Idea: Hybrid Circuit Contraction

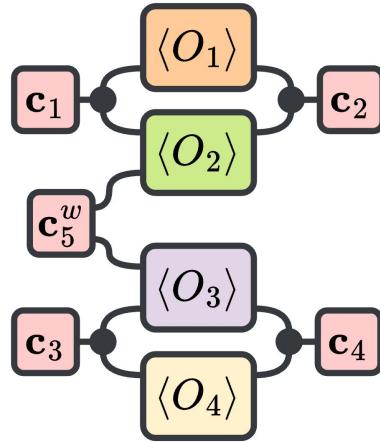


$$\left(\begin{array}{c} |0\rangle \\ |0\rangle \end{array} \right) \xrightarrow{\text{Quantum Tensor}} \left(\begin{array}{c} |0\rangle \\ |0\rangle \end{array} \right) \xrightarrow{\text{Classical Tensor}} \left(\begin{array}{c} |0\rangle \\ |0\rangle \end{array} \right) = \left(\begin{array}{c} |0\rangle \\ |0\rangle \end{array} \right) \xrightarrow{\text{Quantum Tensor}} \left(\begin{array}{c} |0\rangle \\ |0\rangle \end{array} \right) \xrightarrow{\text{Classical Tensor}} \left(\begin{array}{c} |0\rangle \\ |0\rangle \end{array} \right) = \dots$$

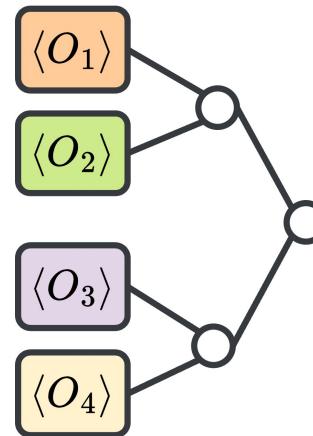
Implication on Postprocessing Overhead



Implication on Postprocessing Overhead (2)



postprocessing TN



optimal contraction tree

TN-based cost: $\mathcal{O}(8^3)$

brute-force cost: $\mathcal{O}(8^5)$

Hybrid circuit contraction reduces postprocessing overhead by orders-of-magnitude!

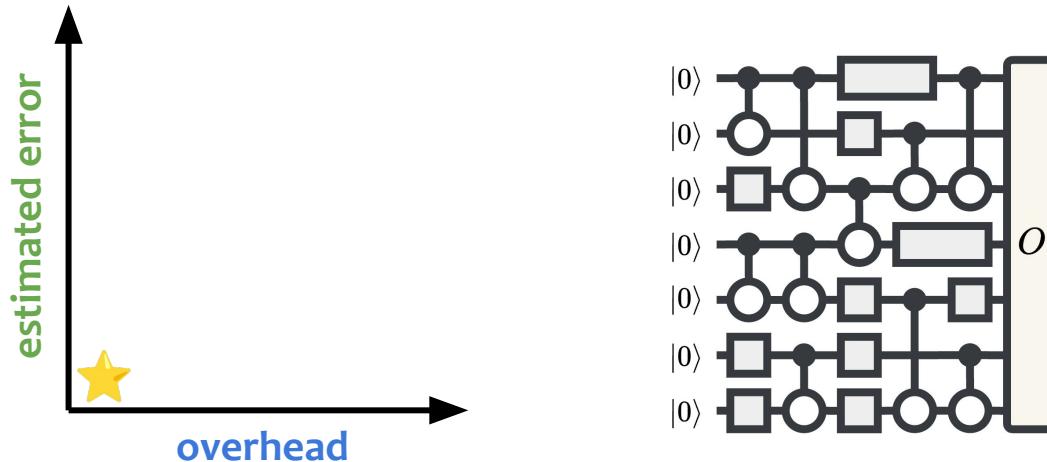
Outline

- Motivation
- Challenges and Key Ideas
- Background: Tensor Networks
- The Key Idea: Hybrid Quantum Circuit Contraction
- **The qTPU Compiler**
- Evaluation

Compiler Design Goals

How can we ensure that we fully utilize the benefits of hybrid circuit contraction?

- Transform quantum circuits into **optimized hybrid TNs!**
- Find an **optimal tradeoff** between **estimated error** and **overhead**

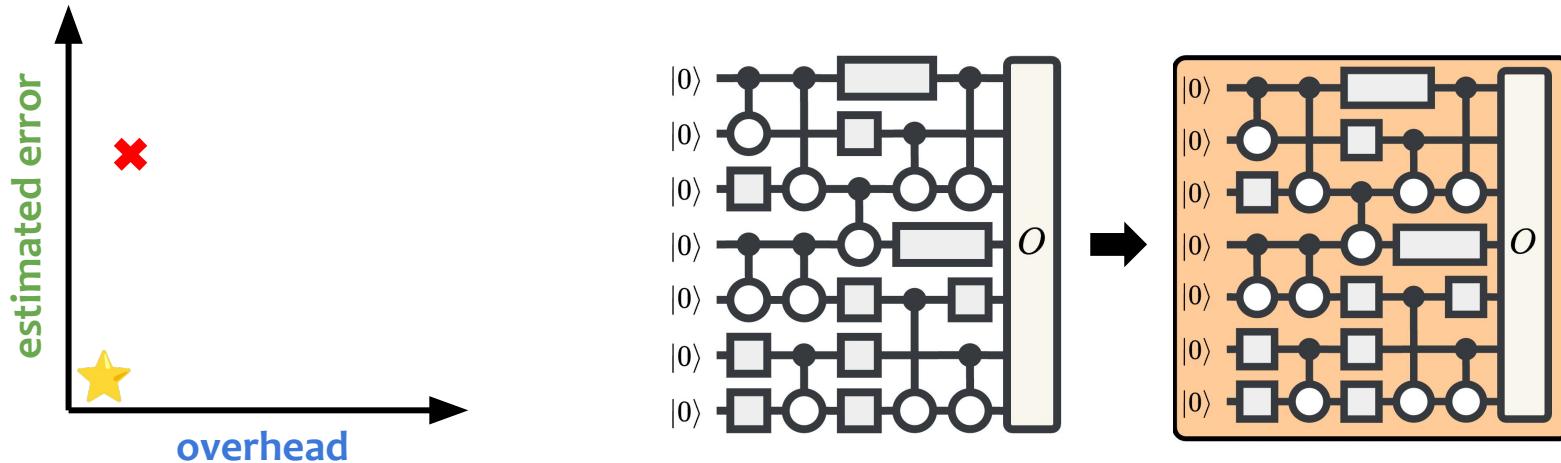


Here: estimated error = number of qubits

Compiler Design Goals

How can we ensure that we fully utilize the benefits of hybrid circuit contraction?

- Transform quantum circuits into **optimized hybrid TNs!**
- Find an **optimal tradeoff** between **estimated error** and **overhead**

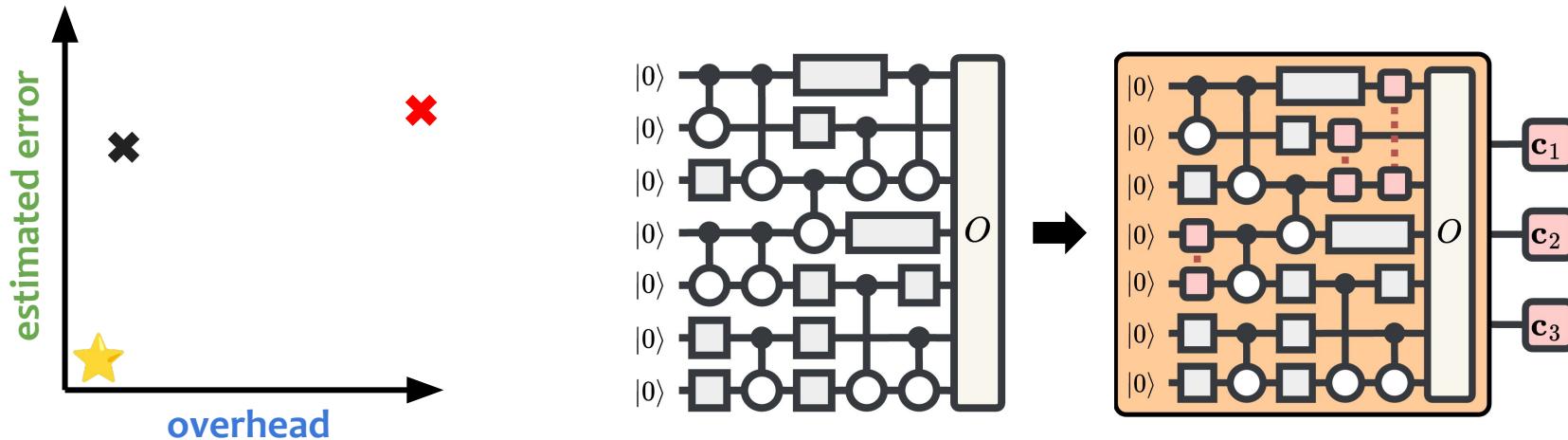


Here: $\text{estimated error} = \text{number of qubits}$

Compiler Design Goals

How can we ensure that we fully utilize the benefits of hybrid circuit contraction?

- Transform quantum circuits into **optimized hybrid TNs!**
- Find an **optimal tradeoff** between **estimated error** and **overhead**

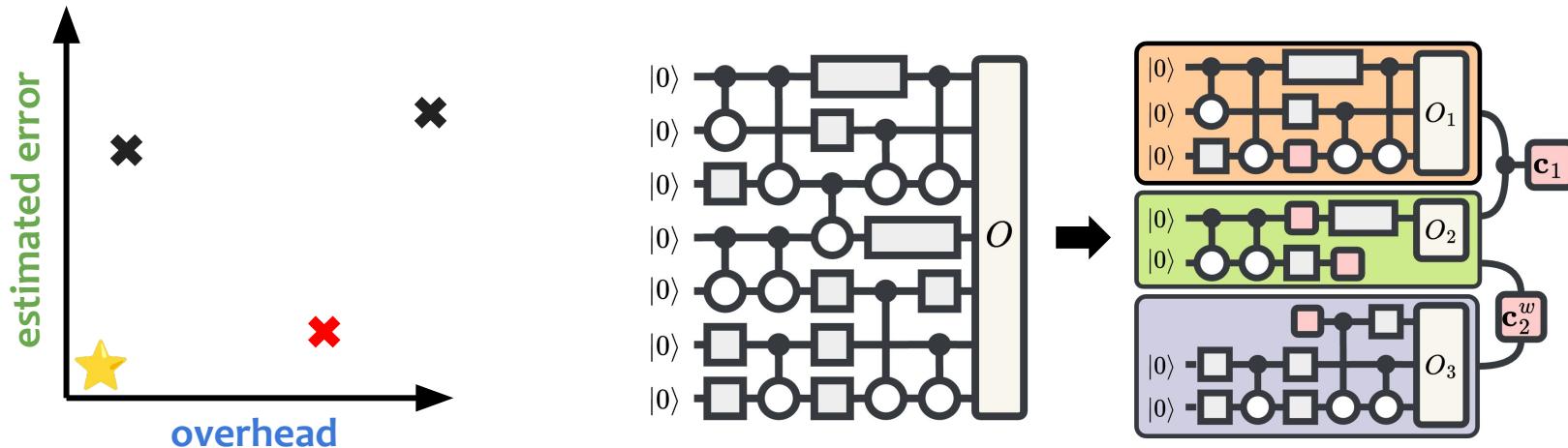


Here: $\text{estimated error} = \text{number of qubits}$

Compiler Design Goals

How can we ensure that we fully utilize the benefits of hybrid circuit contraction?

- Transform quantum circuits into **optimized hybrid TNs!**
- Find an **optimal tradeoff** between **estimated error** and **overhead**

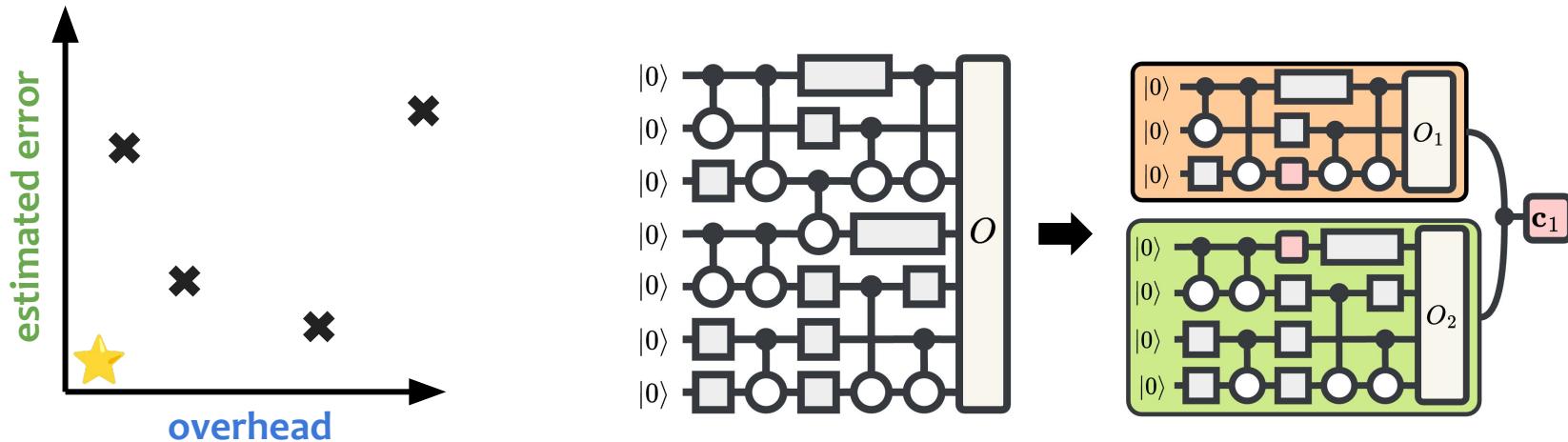


Here: $\text{estimated error} = \text{number of qubits}$

Compiler Design Goals

How can we ensure that we fully utilize the benefits of hybrid circuit contraction?

- Transform quantum circuits into **optimized hybrid TNs!**
- Find an **optimal tradeoff** between **estimated error** and **overhead**

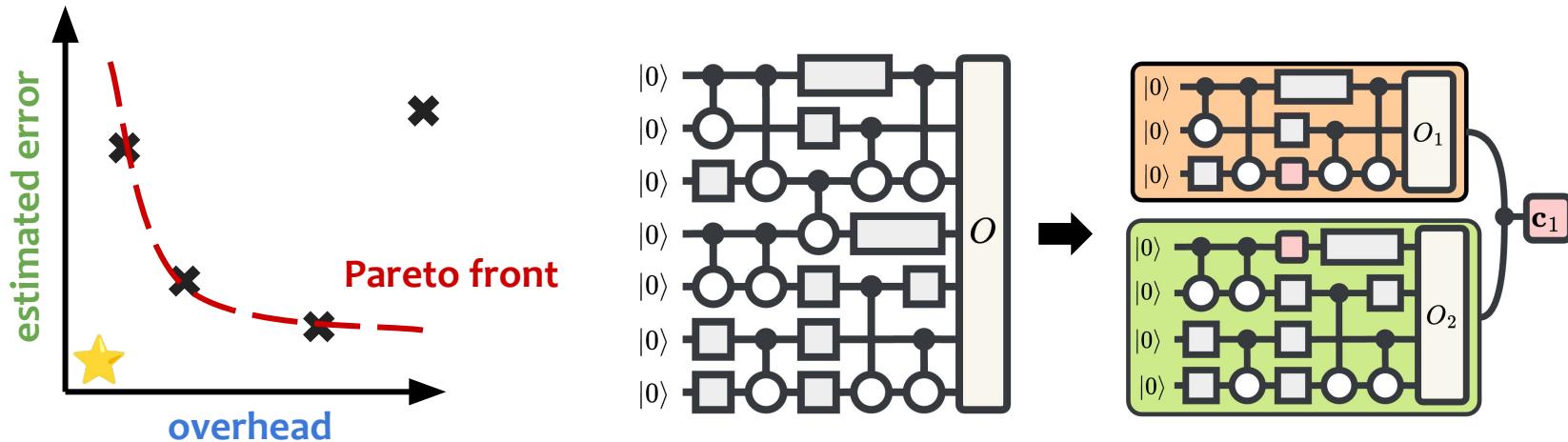


Here: $\text{estimated error} = \text{number of qubits}$

Compiler Design Goals

How can we ensure that we fully utilize the benefits of hybrid circuit contraction?

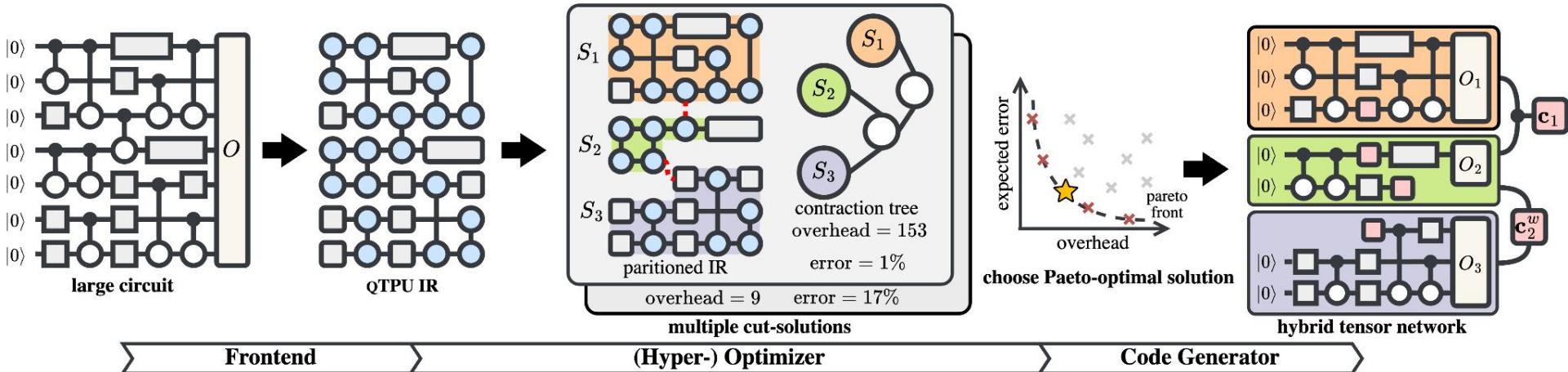
- Transform quantum circuits into **optimized hybrid TNs!**
- Find an **optimal tradeoff** between **estimated error** and **overhead**



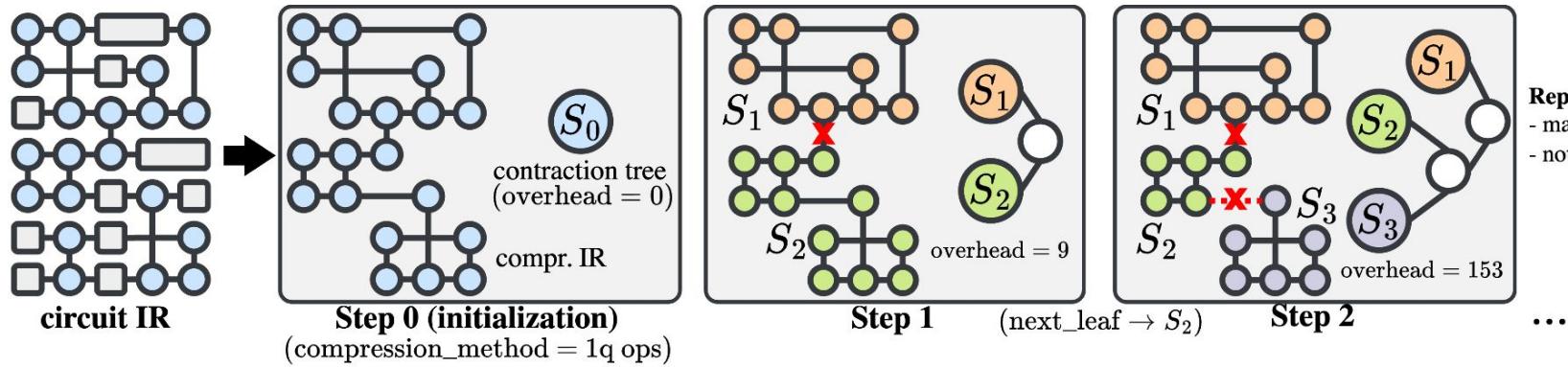
Here: estimated error = number of qubits

qTPU Compiler overview

Approach: Use Hyperparameter Optimization to explore the tradeoff!



Optimizer



Outline

- Motivation
- Challenges and Key Ideas
- Background: Tensor Networks
- The Key Idea: Hybrid Quantum Circuit Contraction
- The qTPU Compiler
- **Implementation and Evaluation**

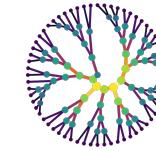
Implementation

Implementation using Python

- **Qiskit** for quantum circuits  **Qiskit**
- **Qiskit-Addon-Cutting** (QAC) for quasiprobability decompositions (QPDs)

Compiler:

- **Cotengra** and **Quimb** for tensor network optimizations
- **Optuna** for hyperparameter optimization



Runtime:

- **NVIDIA CuTensorNet** for GPU-acceleration
- Qiskit for quantum circuit execution



Main research questions (RQs):

RQ1: How does qTPU impact the postprocessing overhead of circuit knitting?

RQ2: How does qTPU's scale in end-to-end runtime?

RQ3: How fast must QPUs be, such that qTPU outperforms purely classical methods?

(More results in the paper!)

Main research questions (RQs):

RQ1: How does qTPU impact the postprocessing overhead of circuit knitting?

RQ2: How does qTPU's scale in end-to-end runtime?

RQ3: How fast must QPUs be, such that qTPU outperforms purely classical methods?

(More results in the paper!)

Baselines:

- **Qiskit-Addon-Cutting** (QAC), the state-of-the-art circuit knitting framework
- Purely classical simulation using tensor networks (**cuTensorNet**)

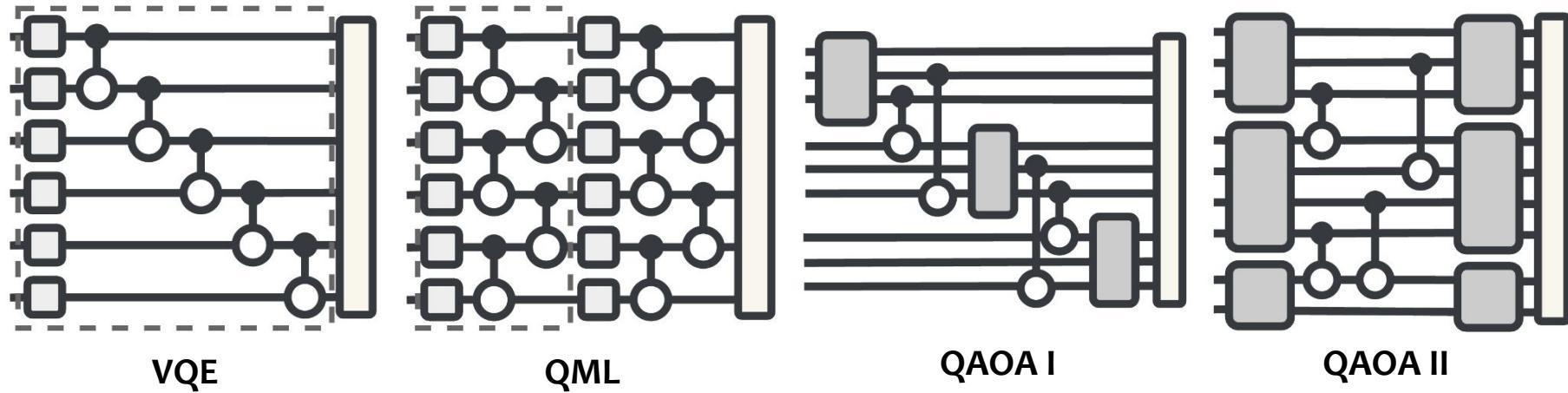
Testbed

- 72 cores (144 HT), 2.40GHz, 362GB RAM
- NVIDIA A100 80GB PCIe GPU
- QPU is simulated by a GPU-based statevector simulator

Metrics

- Postprocessing overhead (FLOPs)
- Runtime (seconds)

Benchmarks



Postprocessing Overhead

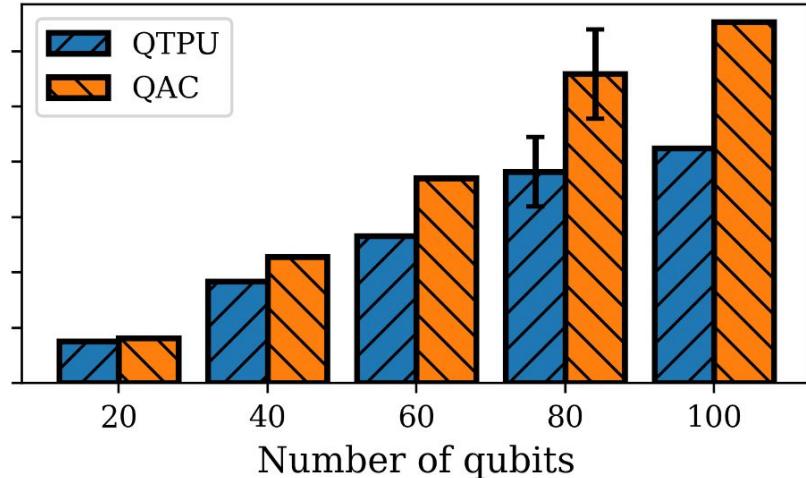
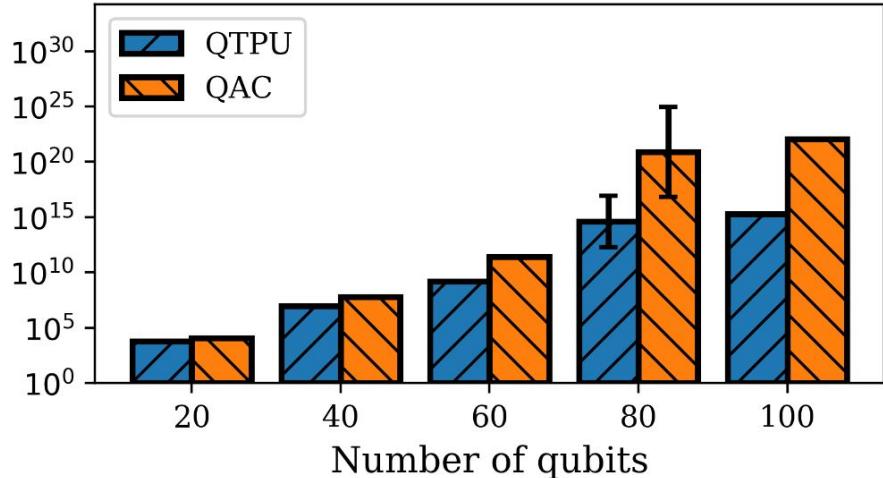
RQ1: How does qTPU impact the postprocessing overhead of circuit knitting?

Postproc. overhead [FLOPs]

a) VQE

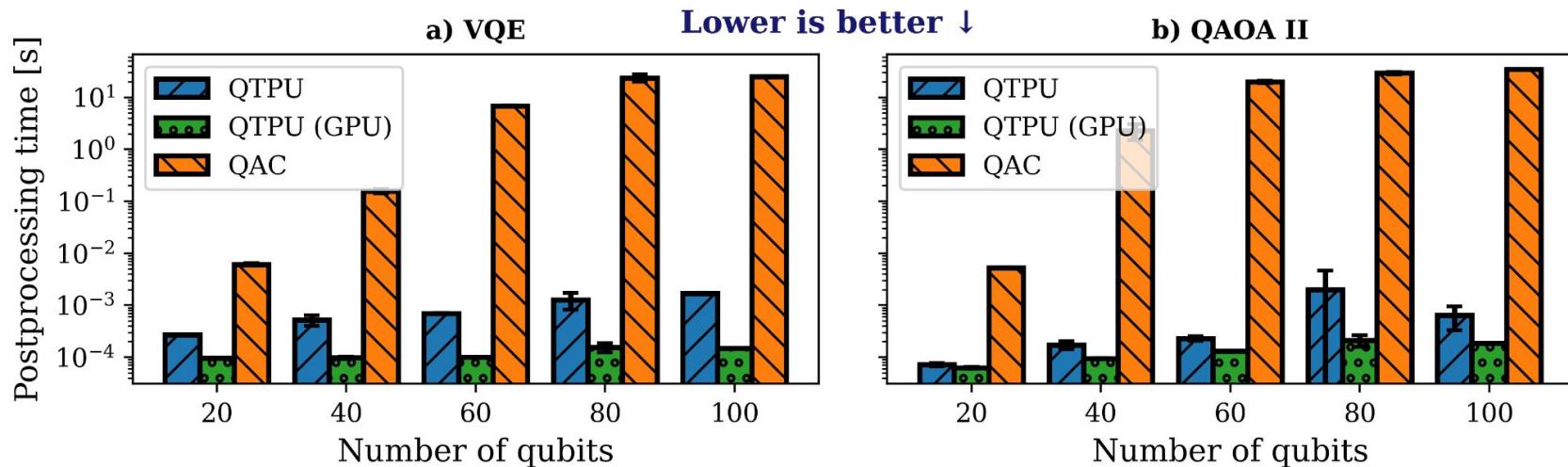
Lower is better ↓

b) QAOA II



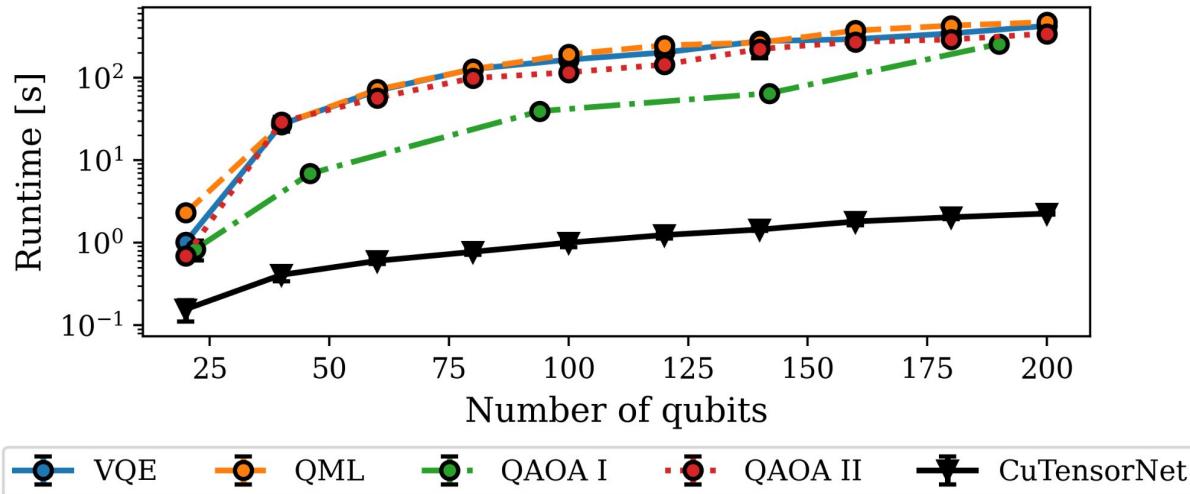
Postprocessing Time

RQ1: How does qTPU impact the postprocessing overhead of circuit knitting?



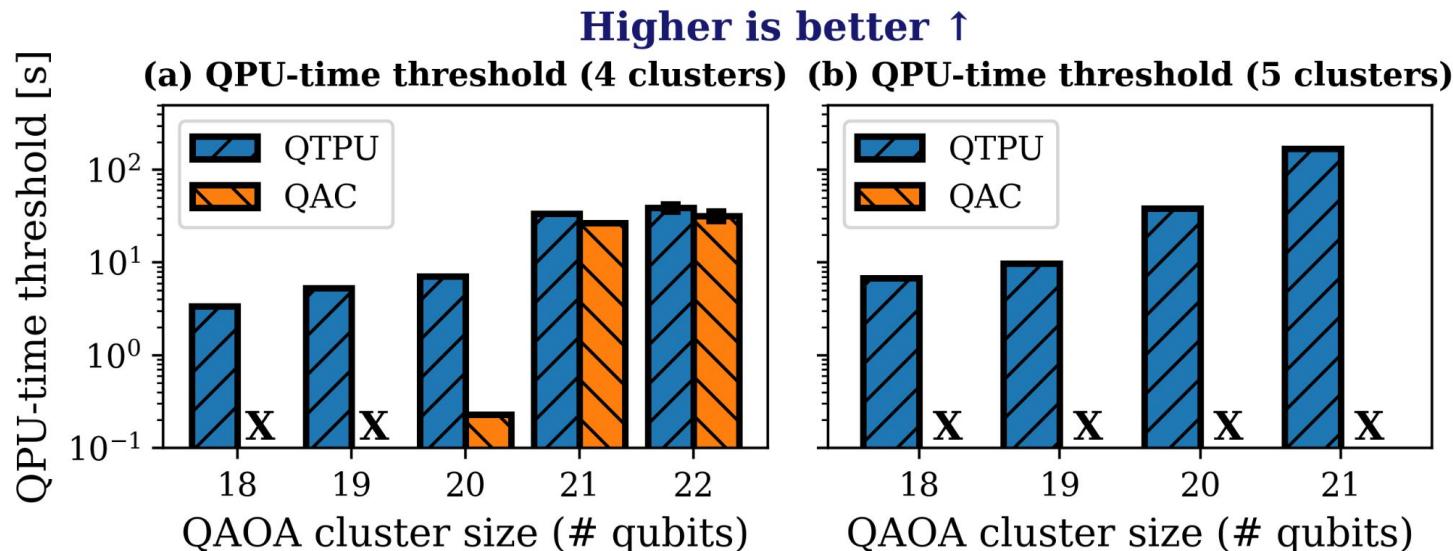
End-to-end Runtime

RQ2: How does qTPU's scale in end-to-end runtime?



Runtime Threshold

RQ3: How fast must QPUs be, such that qTPU outperforms purely classical methods?



Conclusion

- Race: QPUs vs. efficient classical methods
- Why not use both? → Hybrid processing on QPUs and GPUs
- **Circuit knitting** as a promising approach, but high **brute-force** overheads

qTPU: Large-scale hybrid quantum-classical processing using tensor networks

1. **Key approach:** representing circuit knitting as a **hybrid tensor network (h-TN)**
2. **qTPU Compiler:** Automatic transformation of circuits to optimized h-TNs
3. **qTPU Runtime:** Large-scale h-TN contraction using hybrid QPUs and GPUs

→ **Orders-of-magnitude reduction** in postprocessing overhead

→ Chance to outperform purely classical methods!

<https://github.com/nathanieltnow/qtpu>

nathaniel.tornow@tum.de