



# HPCA 2025 Tutorial

## Topic 2. QuCT: A Framework for Analyzing Quantum Circuit by Extracting Contextual and Topological Features



JanusQ  
Cloud

Speaker: Tianyao Chu

College of Computer Science and Technology  
Zhejiang University (ZJU)

[https://janusq.github.io/HPCA\\_2025\\_Tutorial/](https://janusq.github.io/HPCA_2025_Tutorial/)



Tianyao Chu

[tianyao\\_chu@zju.edu.cn](mailto:tianyao_chu@zju.edu.cn)

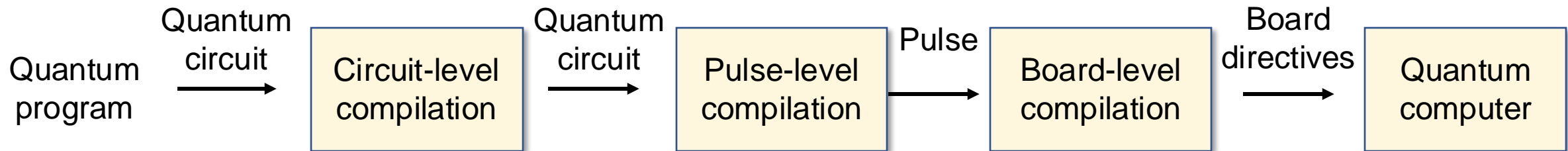
**Tianyao Chu** is a first-year PhD student at the College of Computer Science, Zhejiang University. His research interests include multi-node quantum computing and quantum networking system. He is currently working on the quantum network-on-chip based on enhanced QST.

# Outline of Presentation



- **Background and challenges**
- QuCT overview
- Upstream model: Circuit feature extraction
- Downstream model 1: Circuit fidelity prediction
- Downstream model 2: Unitary decomposition

## Compilation of a quantum program



### Circuit-level compilation:

- **Input:** quantum circuit

**Output:** Quantum circuit that satisfies the constraints

### Pulse-level compilation:

- **Input:** quantum circuit

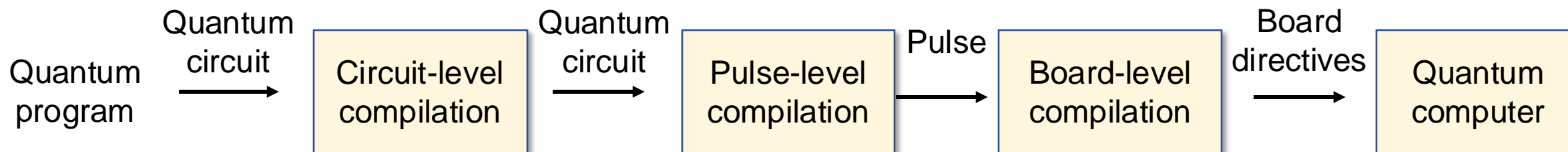
**Output:** Pulses received by qubits

### Board-level compilation:

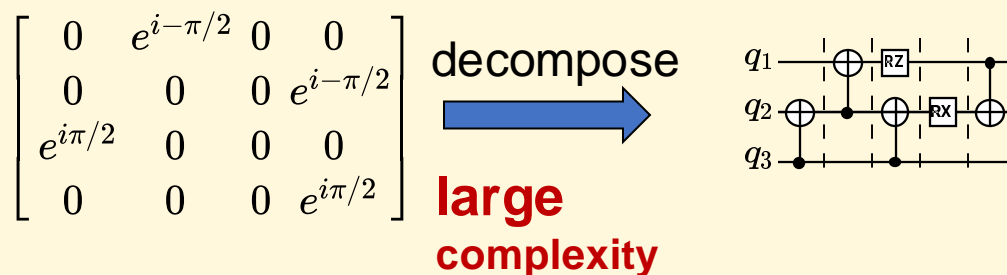
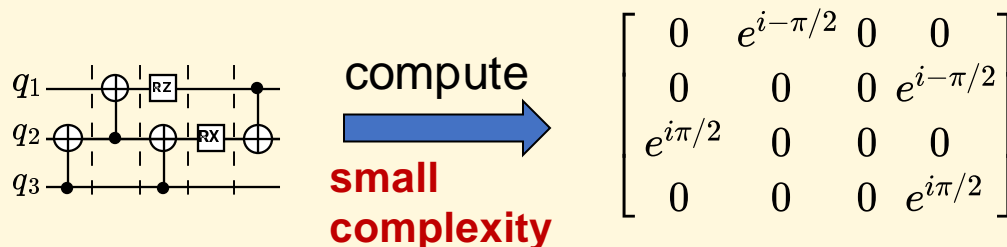
- **Input:** pulses

**Output:** Board directives

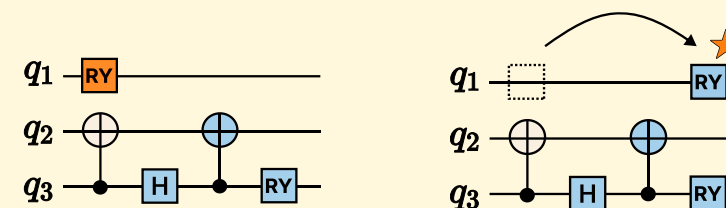
## Key passes of quantum circuit compilation



### Pass 1: Unitary decomposition



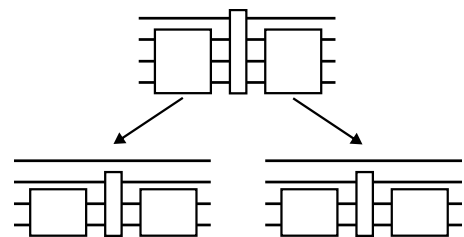
### Pass 2: Fidelity prediction and optimization



Optimize the noise while keeping the equivalence of circuits



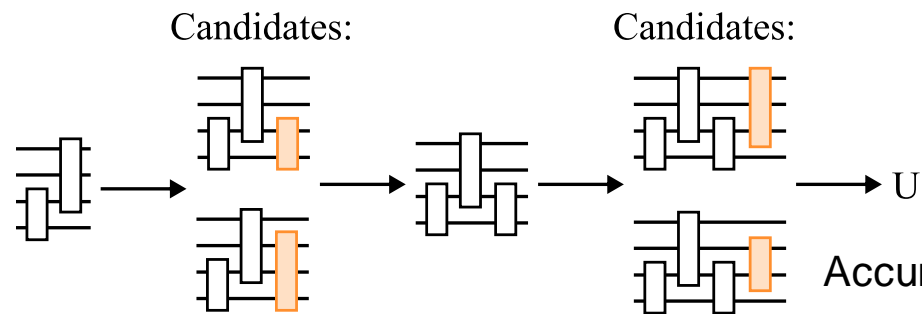
## Unitary decomposition



Template-based method

Fast but

Leads to numerous  
redundant gates



Search-based method

Accurate but

Lacks a heuristic to  
prune candidate space.

Category	Template-based		Search-based	
Method	CCD [1]	QSD [2]	QFAST [3]	Squander [4]
Time	3.6 s	2.1 s	511.2 h	426.2 h
#Gate	3,592	3,817	806	887

$O(4^N)$  #Gate

$O(4^N)$  Time

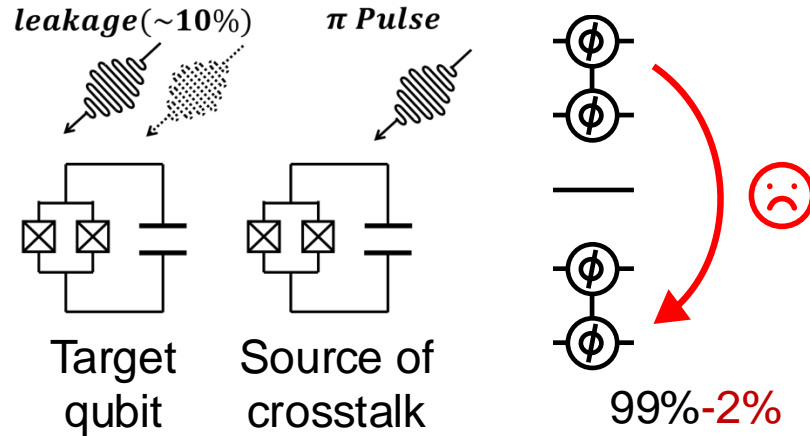
### 5-qubit unitary decomposition

[1] R. Iten, et al. PRA. 2016  
[2] V. Shende, et al. ASP-DAC. 2005  
[3] E. Younis, et al. QCE. 2021.  
[4] P. Rakyta , et al. Quantum, 2022

# Challenges of Fidelity Prediction



## Fidelity prediction



Related to  
the circuit  
structure

Category	RB [5]	XEB [6]	Cycle bench. [7]	Noisy simulat. [8]
Gate-independent error	✓	✓	✓	✓
Crosstalk, Pulse distortion	✗	✗	✓	✓
Inaccuracy (IBMQ Manila )	<b>4-28%</b>	<b>3-36%</b>	<b>2-12%</b>	<b>3-17%</b>

**Fidelity prediction**

**Not one-shot**

[5] E. Knill , et al. D. PRA. 2008.

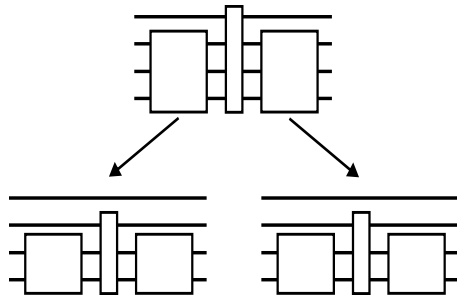
[6] F. Arute, et al. Nature. 2019

[7] A. Erhard, et al. Nature communications. 2019

[8] Isakov, et al ArXiv. 2021.

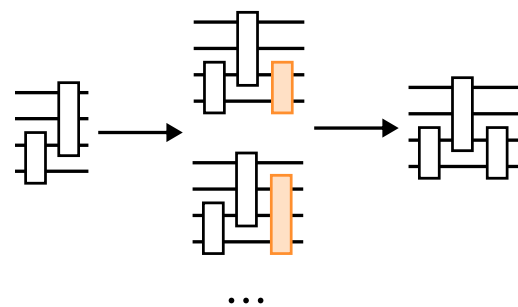
## Unitary decomposition

### Template-based method



**Fast by not accurate:**  
10-qubit unitary ->  
20,000 gate

### Search-based method



**Accurate but slow:**  
10-qubit unitary->  
one year

## Fidelity prediction

Method	Independent noise	Dependent noise	Inaccuracy
RB	✓	✗	4-28%
XEB	✓	✗	3-36%
CB	✓	✓	2-12%
Noisy Simulat.	✓	✓	3-17%

**Fast by inaccurate:**  
cannot model  
dependent noise

**Accurate but slow:**  
require repeated  
executions

**They face a trade-off between the efficiency and accuracy**

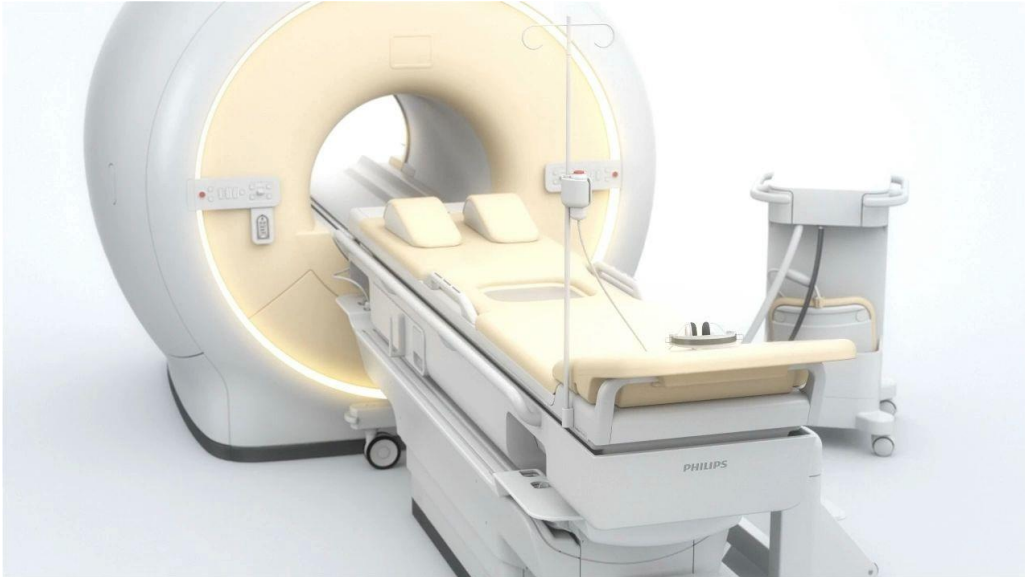


# Outline of Presentation

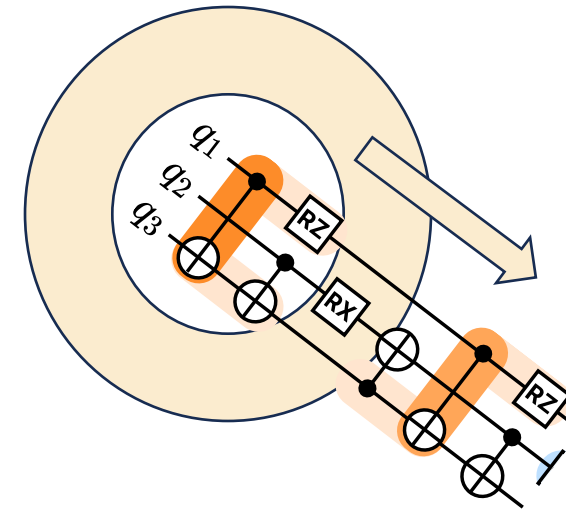


- Background and challenges
- **QuCT overview**
- Upstream model: Circuit feature extraction
- Downstream model 1: Circuit fidelity prediction
- Downstream model 2: Unitary decomposition

## Origin of the name

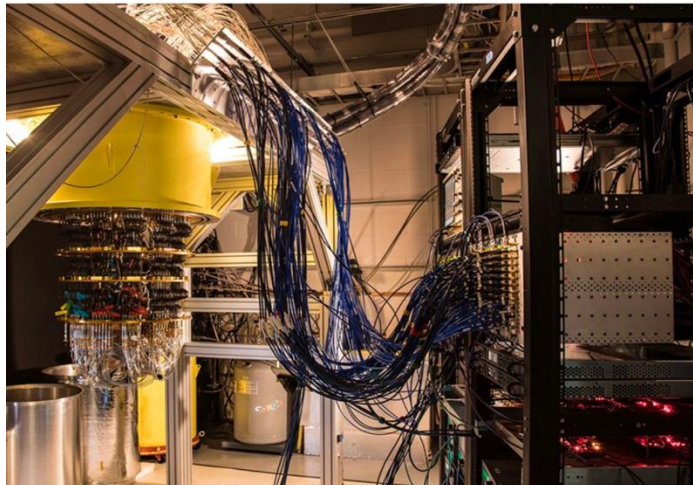


Computerized Tomography

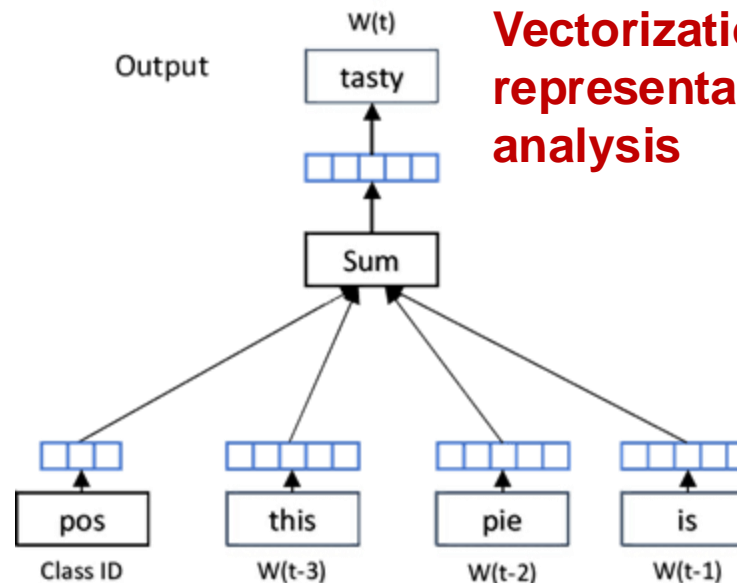


Analyzing Quantum Circuit by  
Contextual and Topological Features

## Solution: Implement circuit topology and context-aware gate vectorization



Quantum circuits are implemented via pulses. There are **interactions between wirings of qubits**.



**Vectorization is a more efficient representation for further analysis**

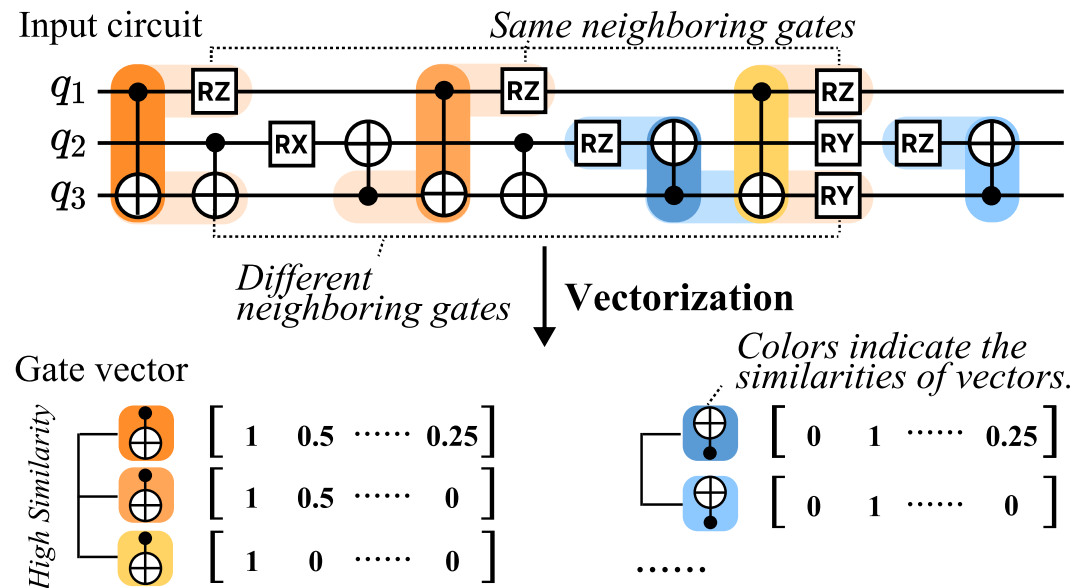
Context extraction is common in **natural language processing (NLP)** and **classical program analysis**

Quantum	NLP
Fidelity prediction	grammar analysis
Circuit generation	Test generation

Quantum program analysis and NLP have similar tasks

Each model is one-shot generated

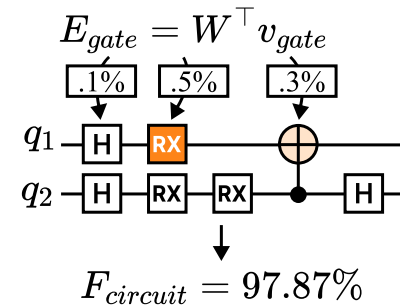
## Upstream Model:



## Downstream Model:

### Circuit Fidelity Prediction

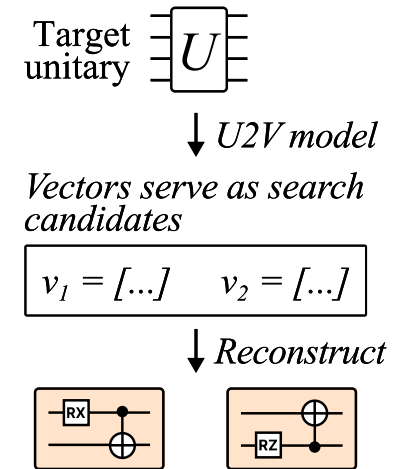
a) Circuit fidelity prediction



b) Compilation- and calibration-level optimizations

More tasks: gate cancellation, bug detection ...

### Unitary Decomposition



Random walk

Vectorization

Fidelity  
prediction

or

Unitary  
decomposition

[illegible]

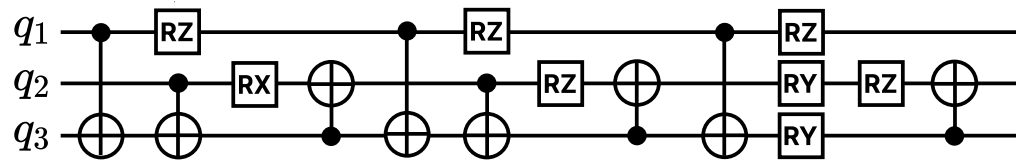
## Upstream Model:

Input circuit

The diagram shows a quantum circuit with three qubits,  $q_1$ ,  $q_2$ , and  $q_3$ . The circuit is composed of the following gates in sequence from left to right:

- A CNOT gate with  $q_1$  as the control and  $q_2$  as the target.
- An  $RZ$  gate on  $q_1$ .
- A CNOT gate with  $q_2$  as the control and  $q_3$  as the target.
- An  $RX$  gate on  $q_2$ .
- A CNOT gate with  $q_1$  as the control and  $q_2$  as the target.
- An  $RZ$  gate on  $q_2$ .
- A CNOT gate with  $q_2$  as the control and  $q_3$  as the target.
- A CNOT gate with  $q_1$  as the control and  $q_2$  as the target.
- Simultaneous  $RY$  gates on  $q_2$  and  $q_3$ .
- An  $RZ$  gate on  $q_1$ .
- A CNOT gate with  $q_2$  as the control and  $q_3$  as the target.
- A final CNOT gate with  $q_1$  as the control and  $q_2$  as the target.

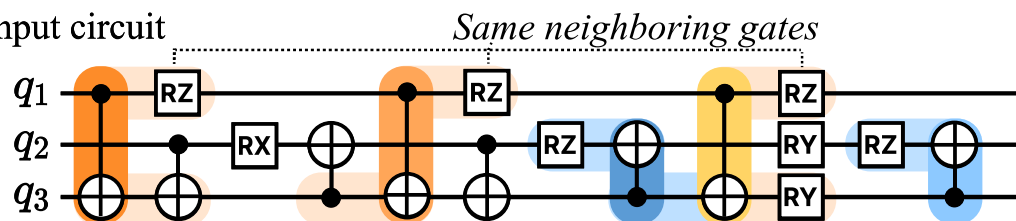
### Input circuit



Random walk

## Upstream Model:

Input circuit



*Different neighboring gates*

**Vectorization**

Gate vector

*High Similarity*

	$\begin{bmatrix} 1 & 0.5 & \cdots & 0.25 \end{bmatrix}$
	$\begin{bmatrix} 1 & 0.5 & \cdots & 0 \end{bmatrix}$
	$\begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}$

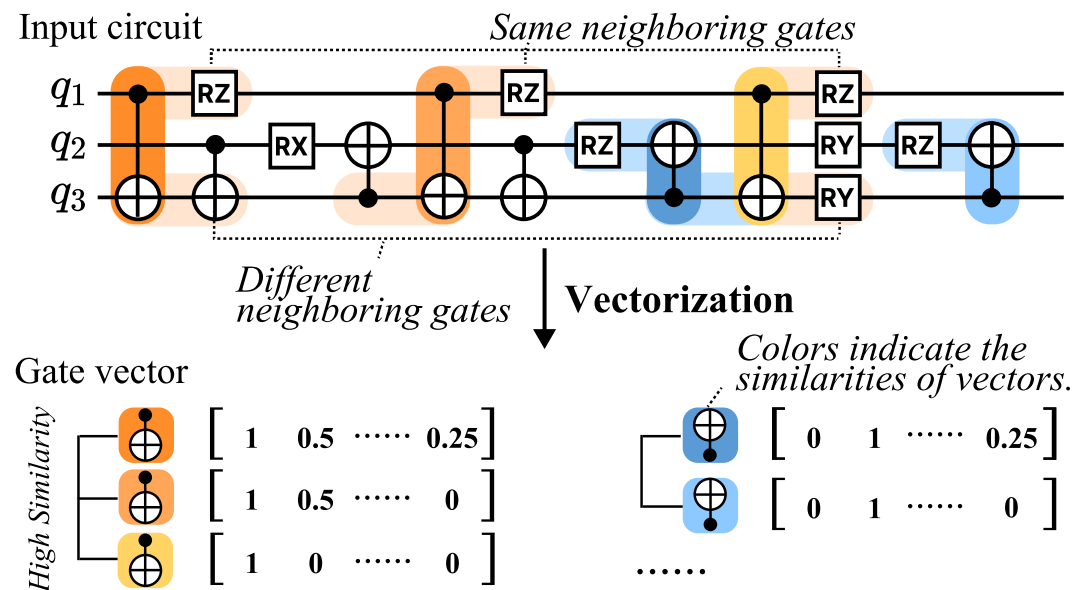
*Colors indicate the similarities of vectors.*

	$\begin{bmatrix} 0 & 1 & \cdots & 0.25 \end{bmatrix}$
	$\begin{bmatrix} 0 & 1 & \cdots & 0 \end{bmatrix}$
.....	

Random walk

Vectorization

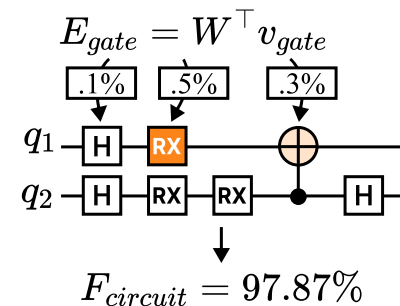
## Upstream Model:



## Downstream Model:

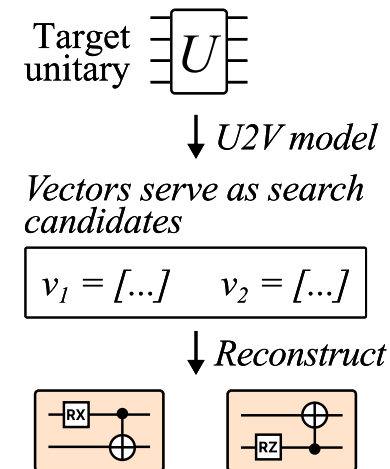
### Circuit Fidelity Prediction

#### a) Circuit fidelity prediction



#### b) Compilation- and calibration-level optimizations

### Unitary Decomposition



More tasks: gate cancellation, bug detection ...

Random walk

Vectorization

Fidelity  
prediction

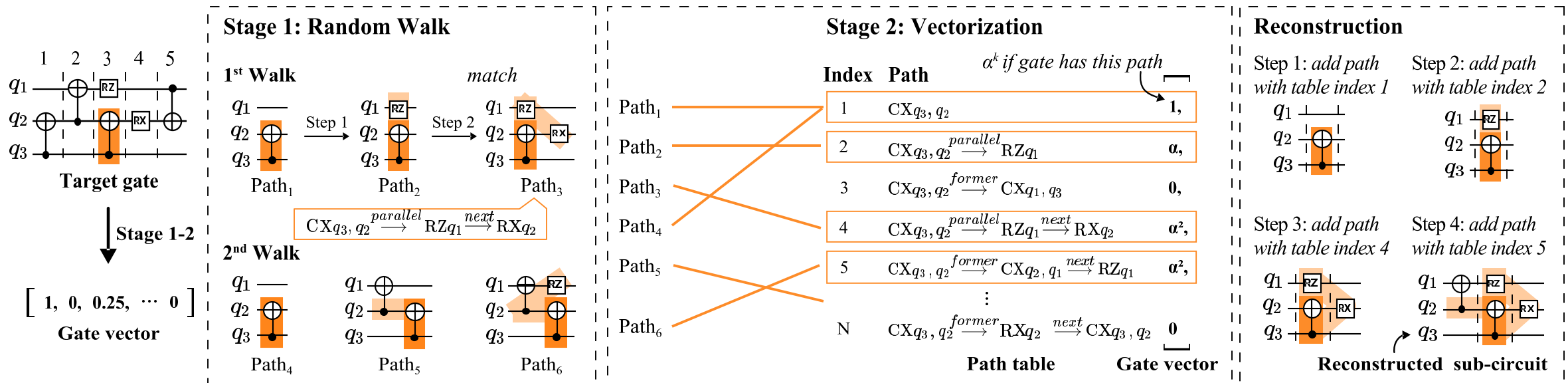
or

Unitary  
decomposition

- Background and challenges
- QuCT overview
- **Upstream model: Circuit feature extraction**
- Downstream model 1: Circuit fidelity prediction
- Downstream model 2: Unitary decomposition
- Experiment

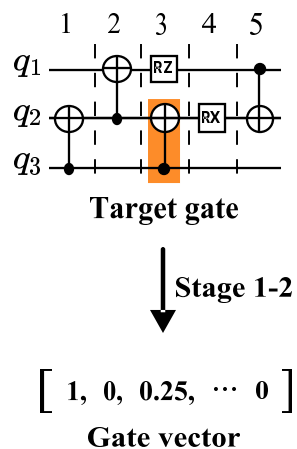


## Two-step vectorization flow

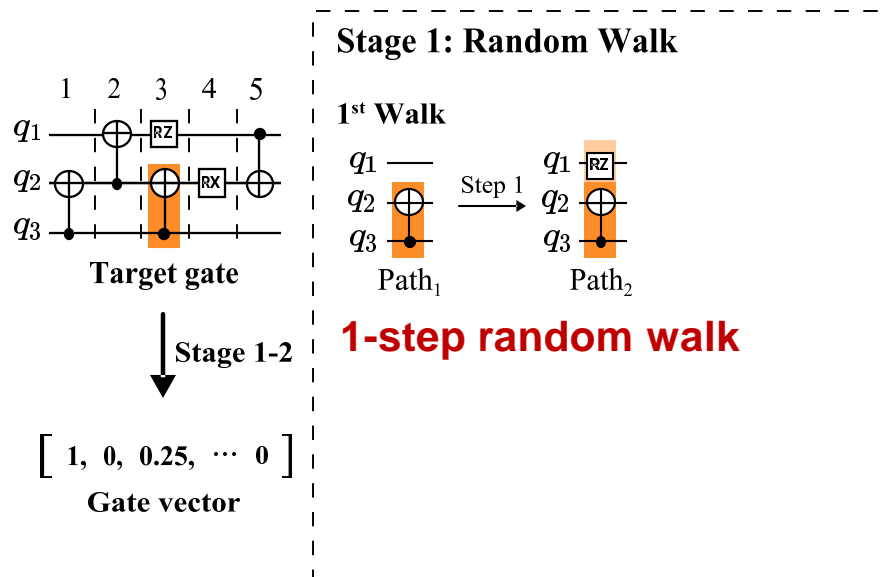


## Two-step vectorization flow

For each gate

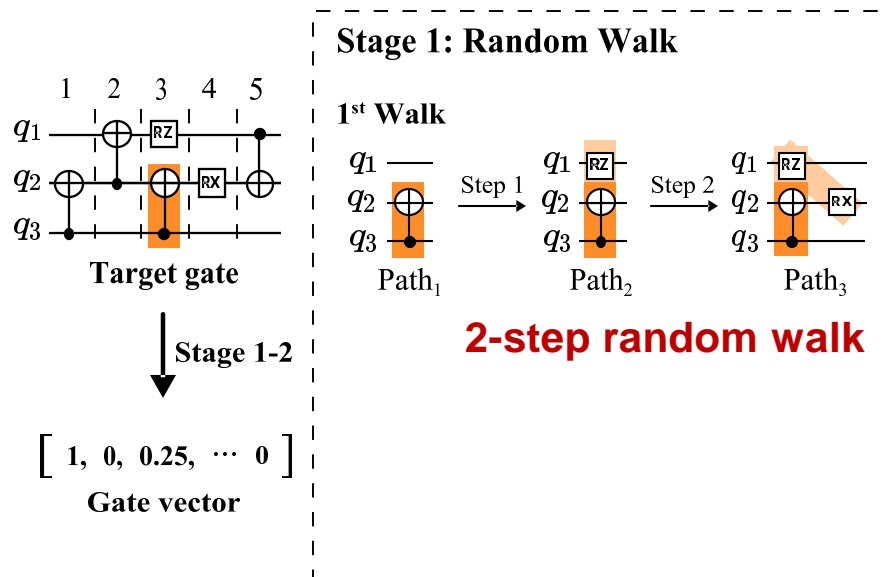


## Two-step vectorization flow



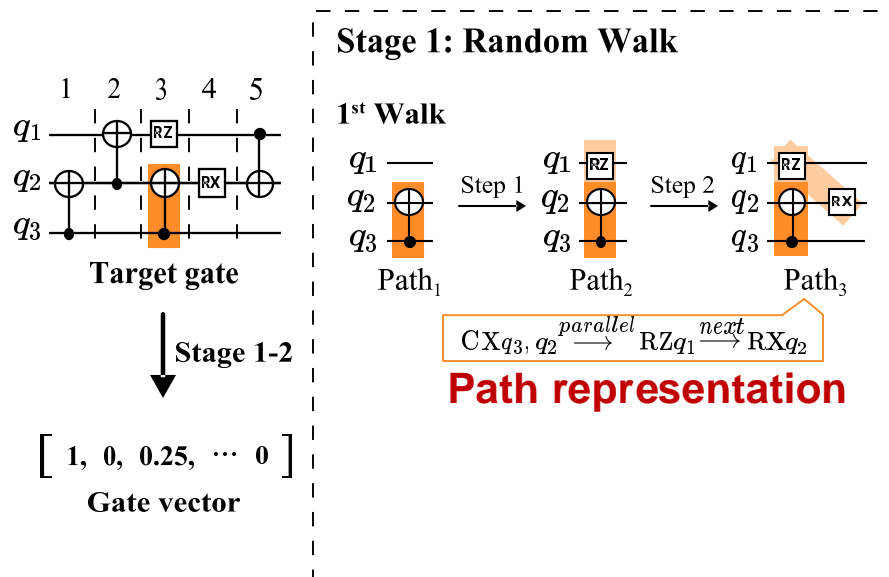
**Step 1: Extract features as paths.**

## Two-step vectorization flow



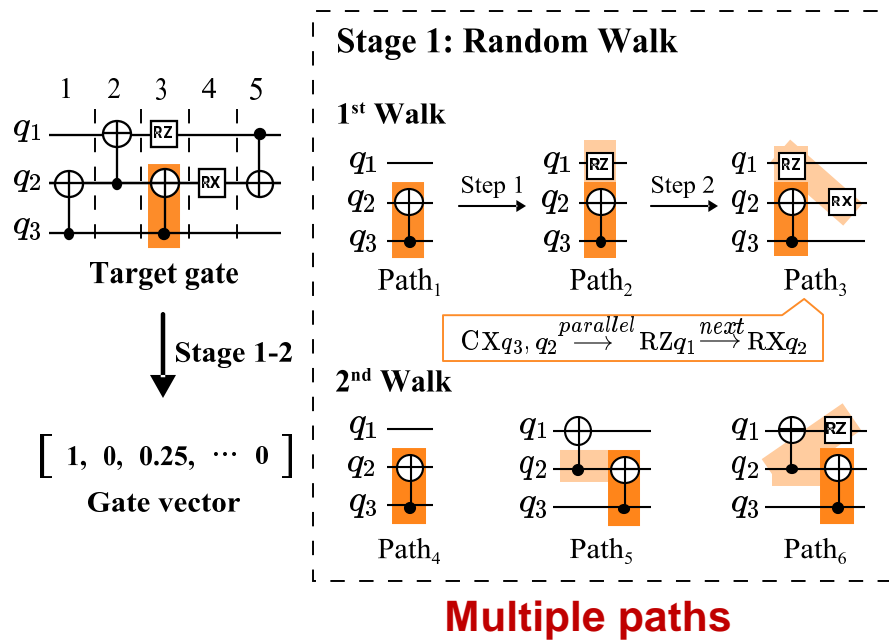
**Step 1: Extract features as paths.**

## Two-step vectorization flow



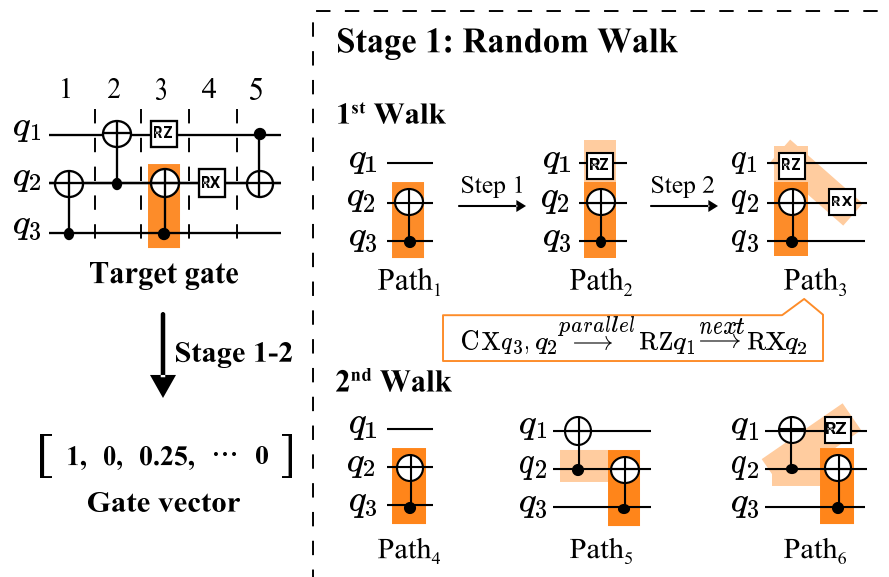
**Step 1: Extract features as paths.**

## Two-step vectorization flow



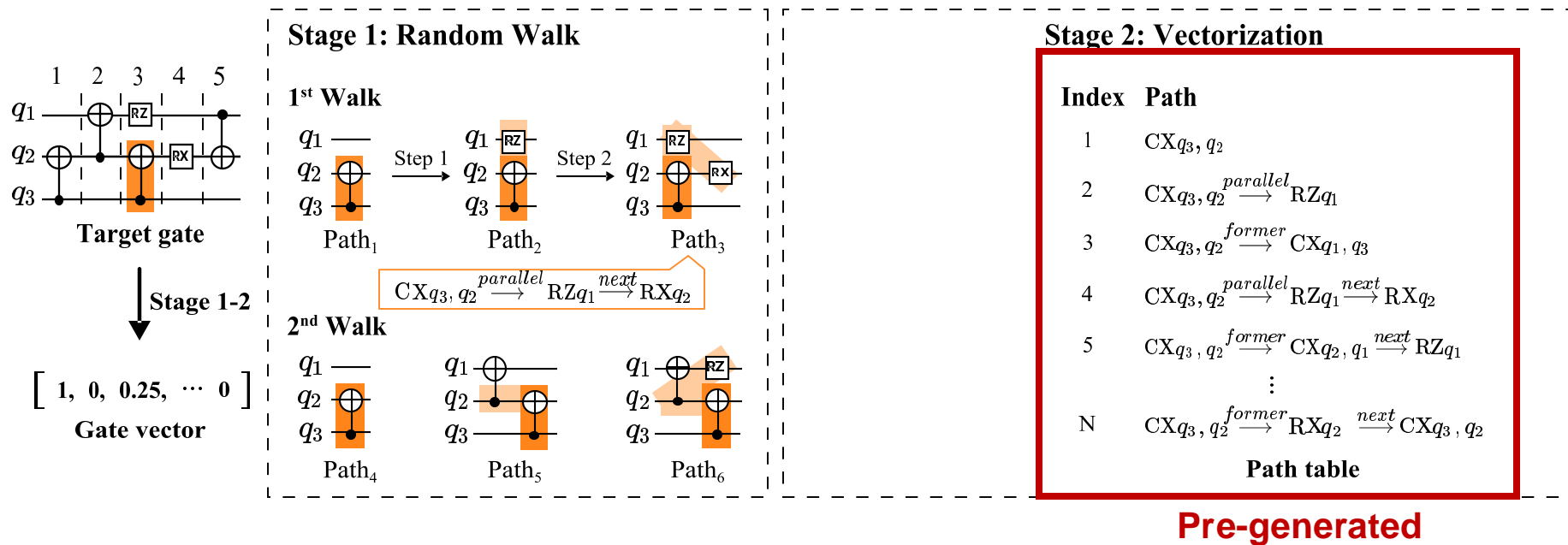
**Step 1: Extract features as paths.**

## Two-step vectorization flow



Obtain vector.

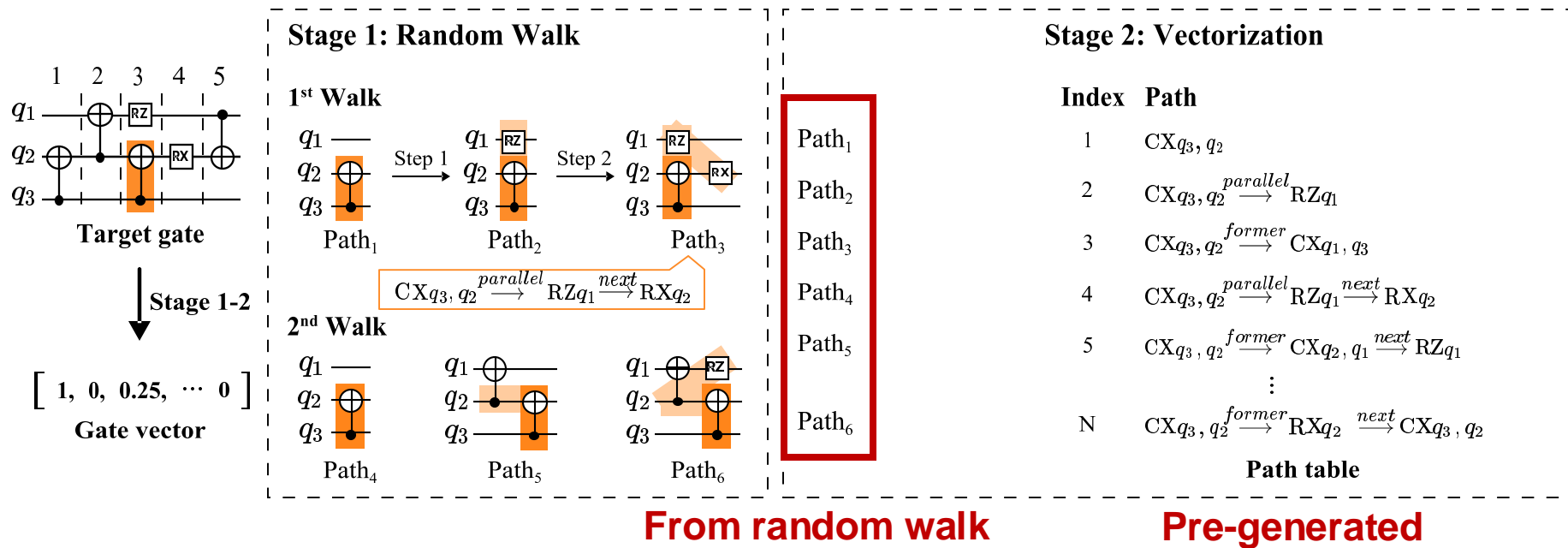
## Two-step vectorization flow



Obtain vector.

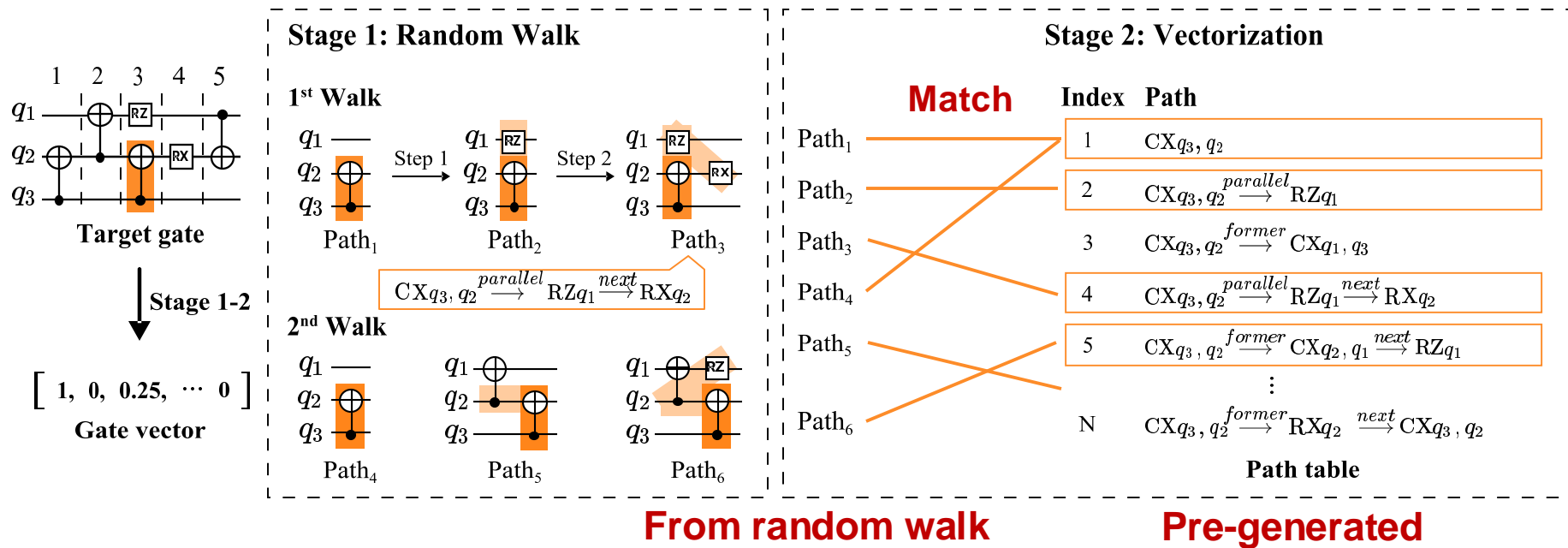


## Two-step vectorization flow



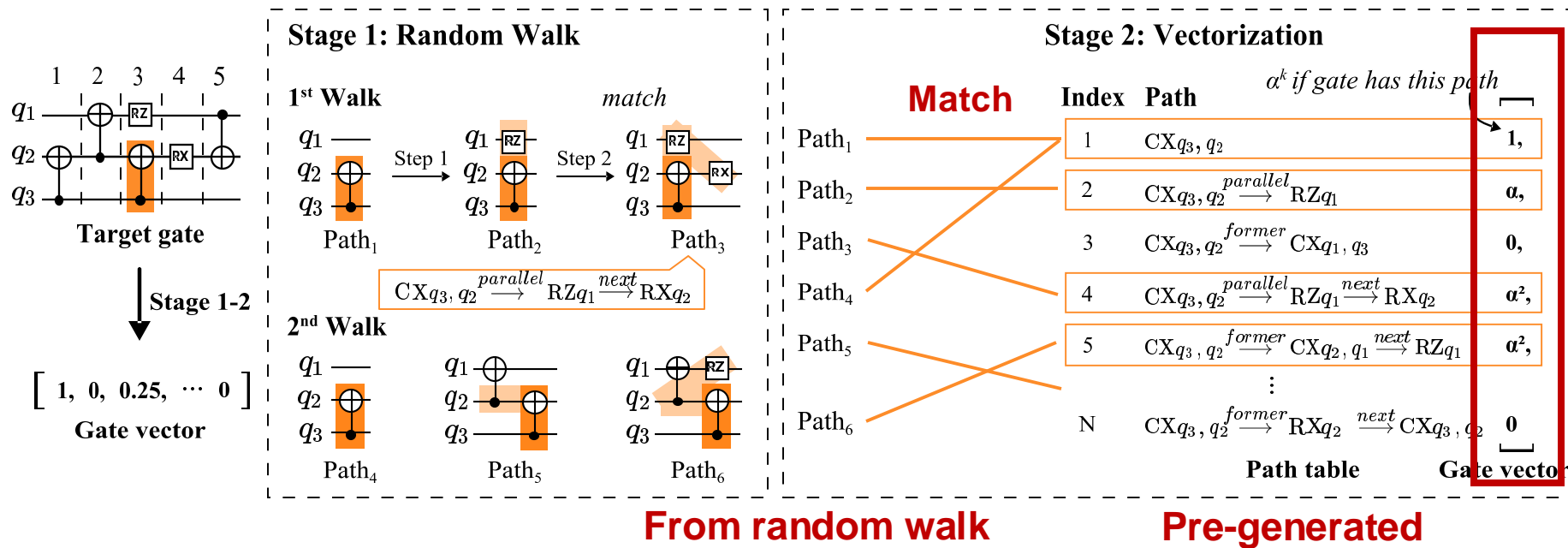
Obtain vector.

## Two-step vectorization flow



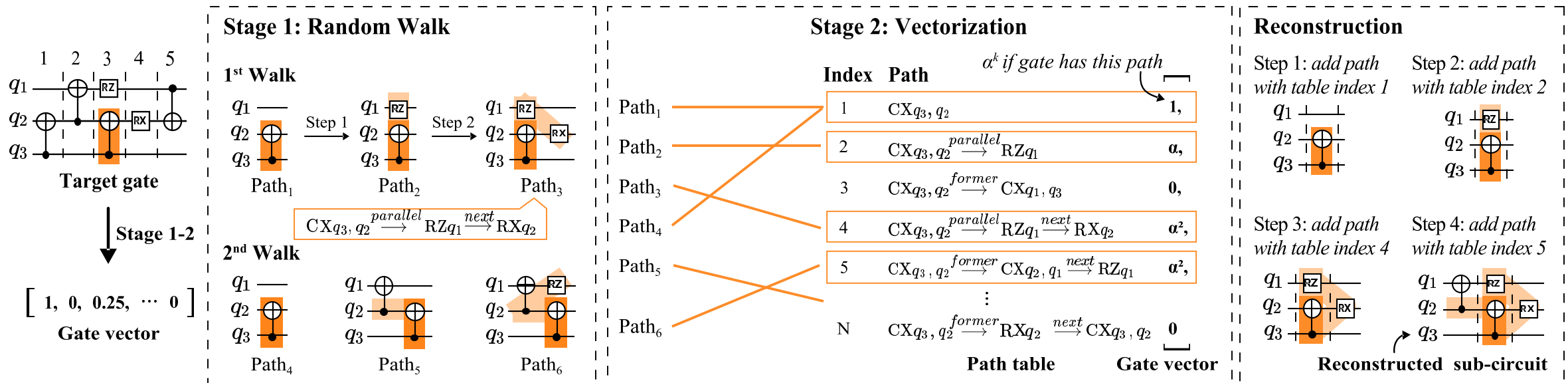
Obtain vector.

## Two-step vectorization flow



Obtain vector.

## Two-step vectorization flow



**Reconstruct circuit.**

# API to Construct Upstream Model



File:

- JanusQ/examples/ipynb/2\_1\_vectorization.ipynb
- [https://janusq.github.io/tutorials/demo/2\\_1\\_vectorization](https://janusq.github.io/tutorials/demo/2_1_vectorization)

```
from janusq.analysis.vectorization import RandomwalkModel
from janusq.objects.backend import GridBackend
from janusq.objects.random_circuit import random_circuits

# define the information of the quantum device
n_qubits = 6
backend = GridBackend(2, 3)

# generate a dataset including varous random circuits
circuit_dataset = random_circuits(backend, n_circuits=100, n_gate_list=[30, 50, 100],
two_qubit_prob_list=[.4], reverse=True)

# apply random work to consturct the vectorization model with a path table
n_steps, n_walks = 1, 100
up_model = RandomwalkModel(n_steps = n_steps, n_walks = n_walks, backend = backend,
decay= 0.5, circuits = circuit_dataset )
up_model.train(circuit_dataset, multi_process=False)
```

define backend {

generate circuit dataset {

construct model using random walk {

# Outline of Presentation



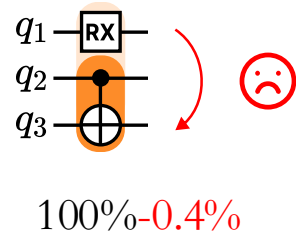
- Background and challenges
- QuCT overview
- Upstream model: Circuit feature extraction
- **Downstream model 1: Circuit fidelity prediction**
- Downstream model 2: Unitary decomposition
- Experiment

# Downstream Model 1: Circuit Fidelity Prediction



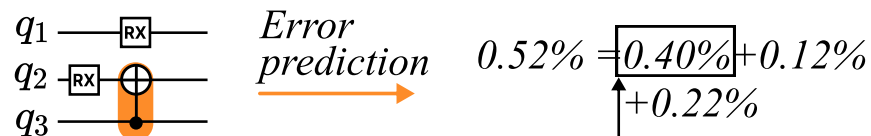
## Gate error prediction

$$E(v_i) = W^T v_i$$



**Model dependent error**

$v_i$  : gate vector.  $W$  : weight vector obtained via training.



Path				...	
$W(\%)$	0.40	0.22	0.12		0

# Downstream Model 1: Circuit Fidelity Prediction



## Gate error prediction

$$E(v_i) = W^T v_i$$

$v_i$  : gate vector.  $W$  : weight vector obtained via training.

## Circuit fidelity prediction

$$F_{circuit} = \prod_{g_i \in G} (1 - E(v_i)) \prod_{q \in Q} MF_q$$

$G$  : gate set,  $Q$  : qubit set,  $MF_q$  : measurement fidelity

***The probability that all gates are correct.***



# Downstream Model 1: Circuit Fidelity Prediction



## Gate error prediction

$$E(v_i) = W^T v_i$$

$v_i$  : gate vector.  $W$  : weight vector obtained via training.

## Circuit fidelity prediction

$$F_{circuit} = \prod_{g_i \in G} (1 - E(v_i)) \prod_{q \in Q} MF_q$$

$G$  : gate set,  $Q$  : qubit set,  $MF_q$  : measurement fidelity

## Training process of weight vector $W$ :

Obtain fidelity dataset  $(circuit, F_{ground-truth}) \dots$ ,  $F_{ground-truth}$  : ground-truth circuit fidelity on the target quantum device.

$$\min_W |F_{circuit} - F_{ground-truth}|$$

**Minimize the distance between the prediction and ground-truth fidelity.**

# API to Construct Fidelity Prediction Model



File:

- JanusQ/examples/ipynb/2\_2\_fidelity\_prediction\_simulator.ipynb
- [https://janusq.github.io/tutorials/demo/2\\_2\\_fidelity\\_prediction\\_simulator](https://janusq.github.io/tutorials/demo/2_2_fidelity_prediction_simulator)

construct upstream model

```
from janusq.dataset import real_qc_5bit
from janusq.objects.backend import FullyConnectedBackend
from janusq.analysis.fidelity_prediction import FidelityModel

circuits, fidelities = real_qc_5bit
backend = FullyConnectedBackend(5)
up_model = RandomwalkModel(n_steps = 1, n_walks = 10, backend = backend,
circuits = circuits)
```

construct fidelity  
prediction model

```
fidelity_model = FidelityModel(up_model)
fidelity_model.train((circuits, fidelities), multi_process = False)
```

predict the fidelity of  
the given circuit

```
circuit = random_circuit(backend, n_gates = 100, two_qubit_prob = 0.5 )
fidelity_model.predict_circuit_fidelity(circuit)
```

# Outline of Presentation

---

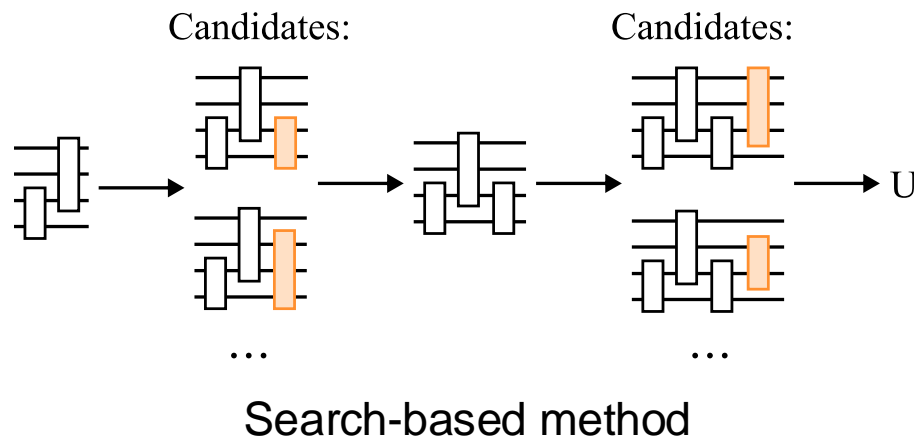


- Background and challenges
- QuCT overview
- Upstream model: Circuit feature extraction
- Downstream model 1: Circuit fidelity prediction
- **Downstream model 2: Unitary decomposition**

# Downstream Model 2: Unitary Decomposition



Improve the current search-based method



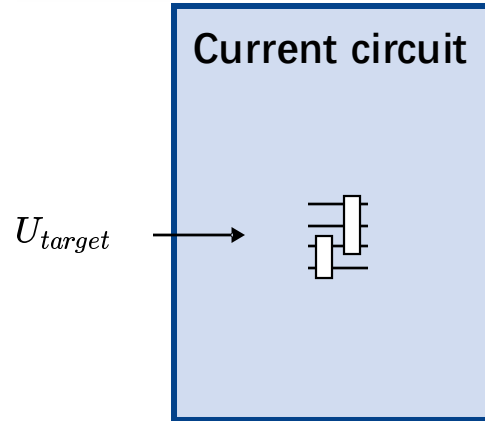
Category	Template-based		Search-based	
Method	CCD [1]	QSD [2]	QFAST [3]	Squander [4]
Time	3.6 s	2.1 s	<b>511.2 h</b>	<b>426.2 h</b>
#Gate	<b>3,592</b>	<b>3,817</b>	806	887

5-qubit unitary decomposition

# Downstream Model 2: Unitary Decomposition



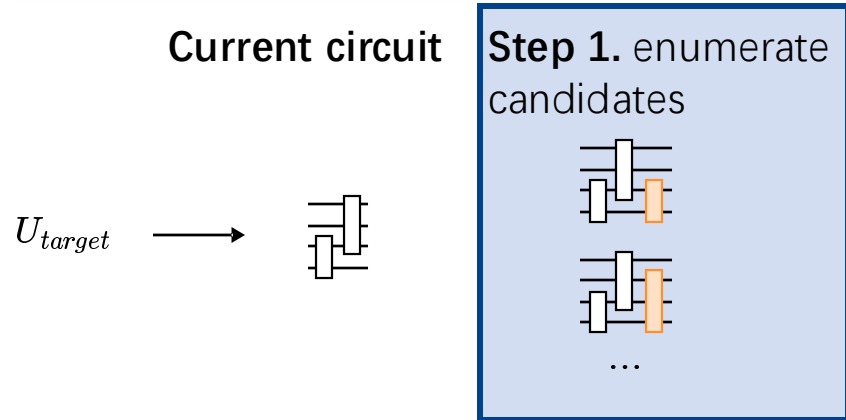
## QFAST workflow



# Downstream Model 2: Unitary Decomposition



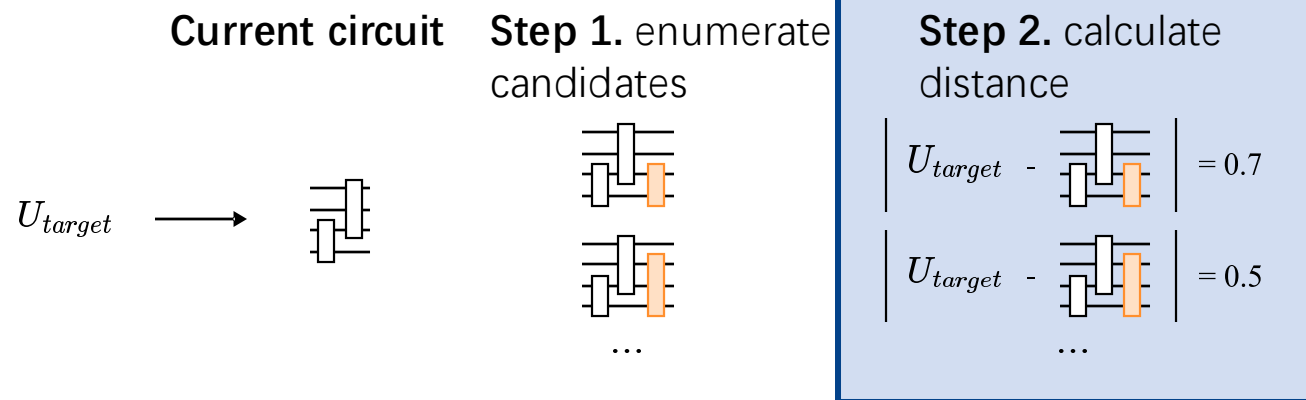
## QFAST workflow



# Downstream Model 2: Unitary Decomposition



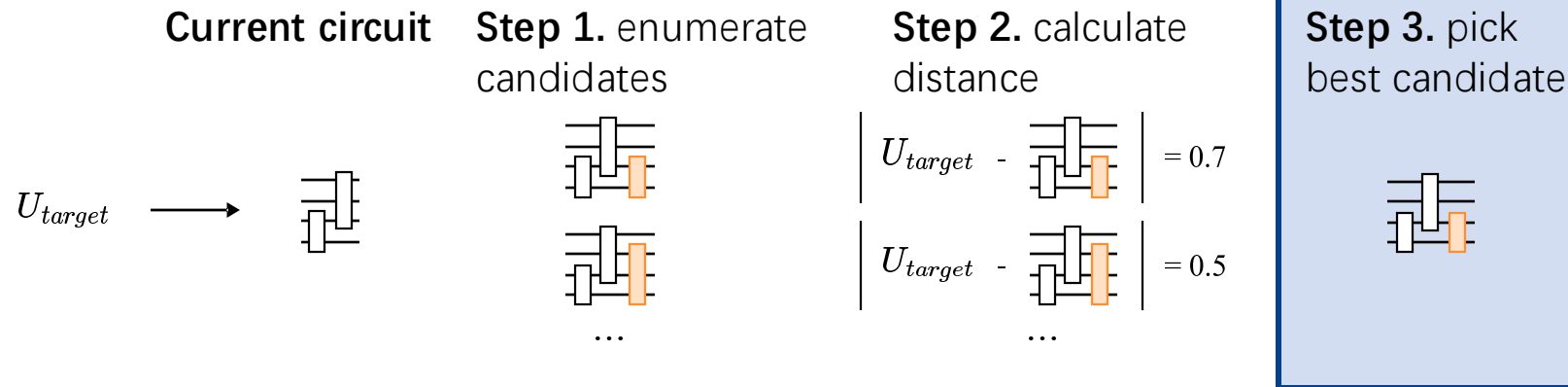
## QFAST workflow



# Downstream Model 2: Unitary Decomposition



## QFAST workflow

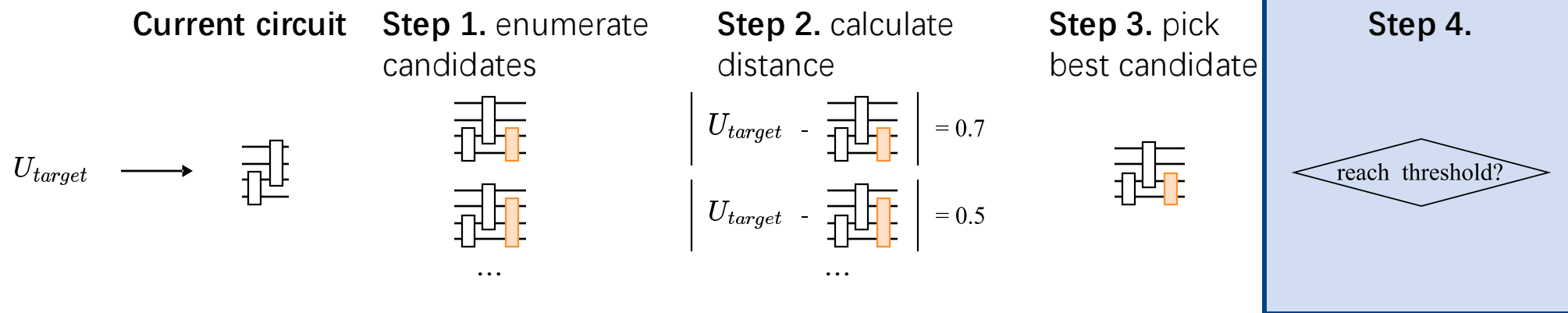




# Downstream Model 2: Unitary Decomposition



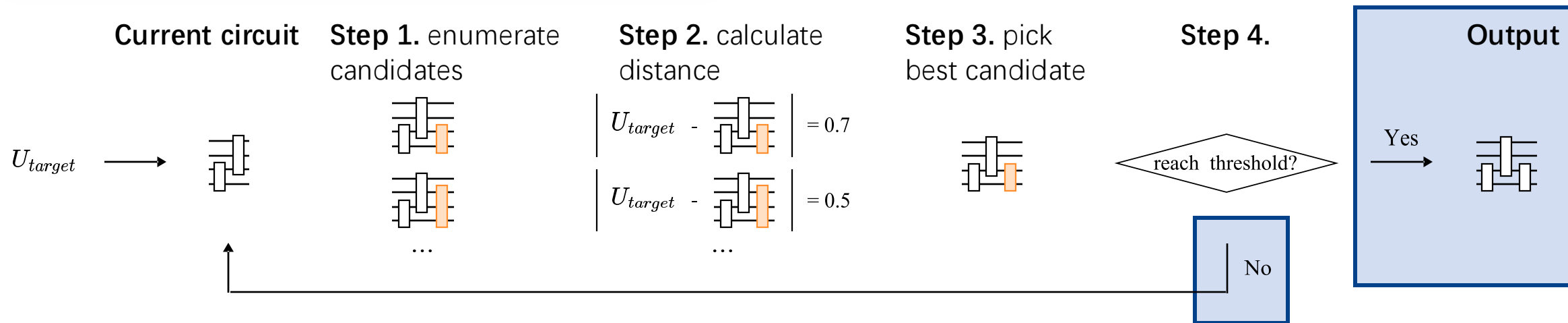
## QFAST workflow



# Downstream Model 2: Unitary Decomposition



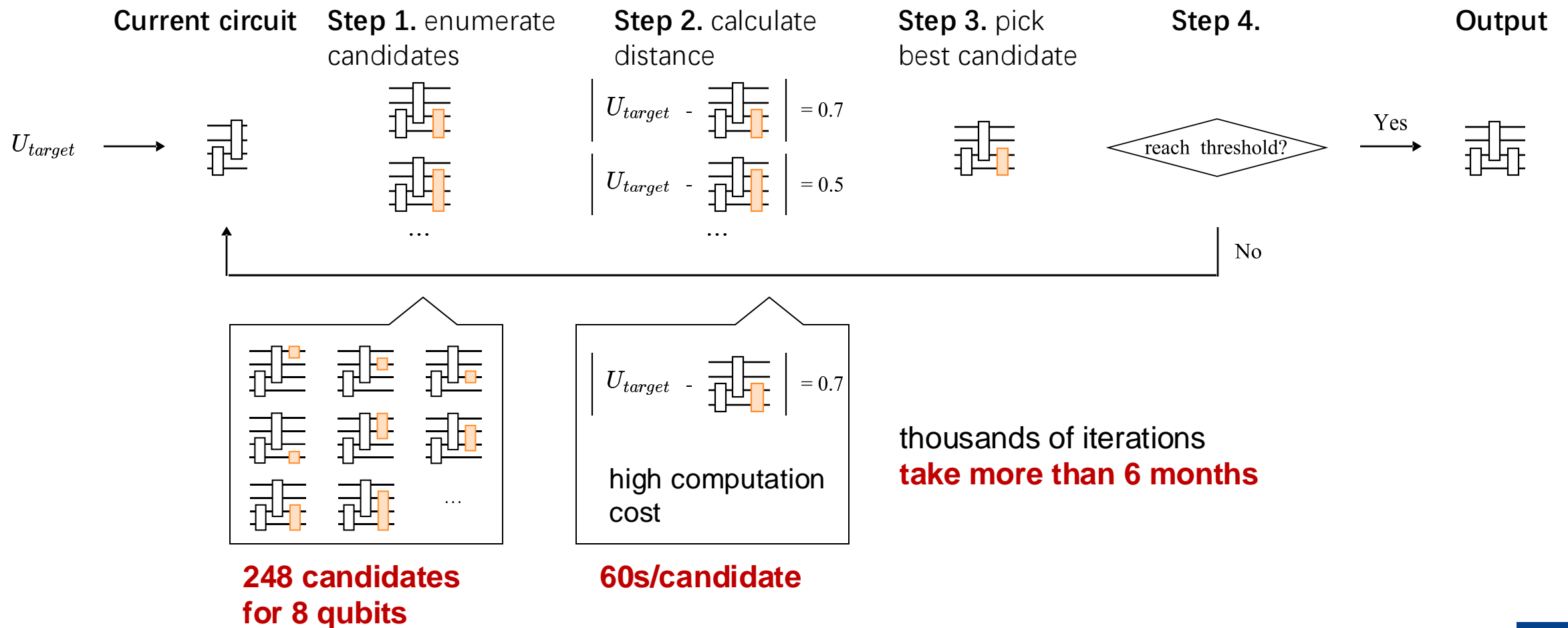
## QFAST workflow



# Downstream Model 2: Unitary Decomposition



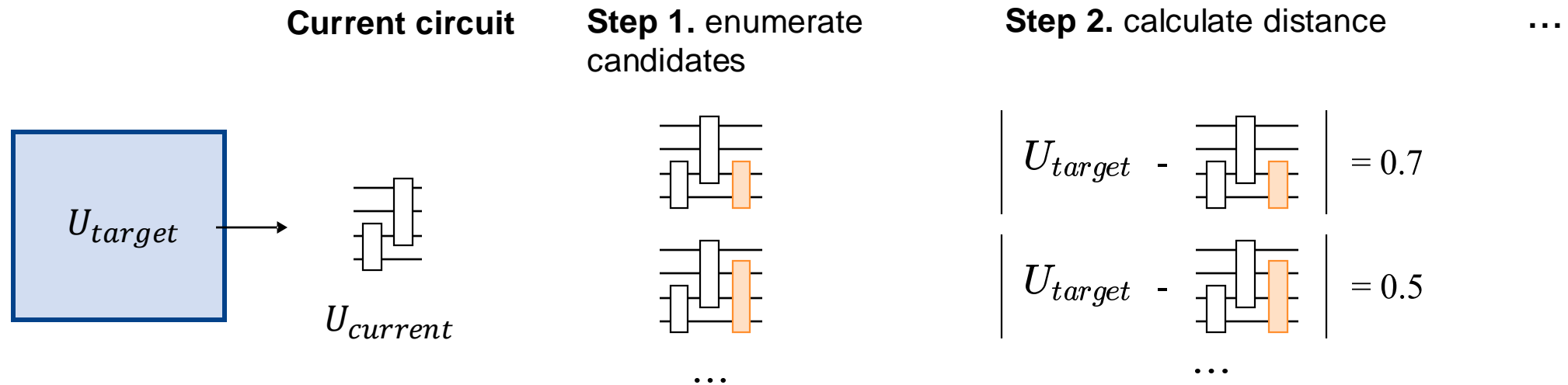
## QFAST workflow





# Downstream Model 2: Unitary Decomposition



## QuCT workflow

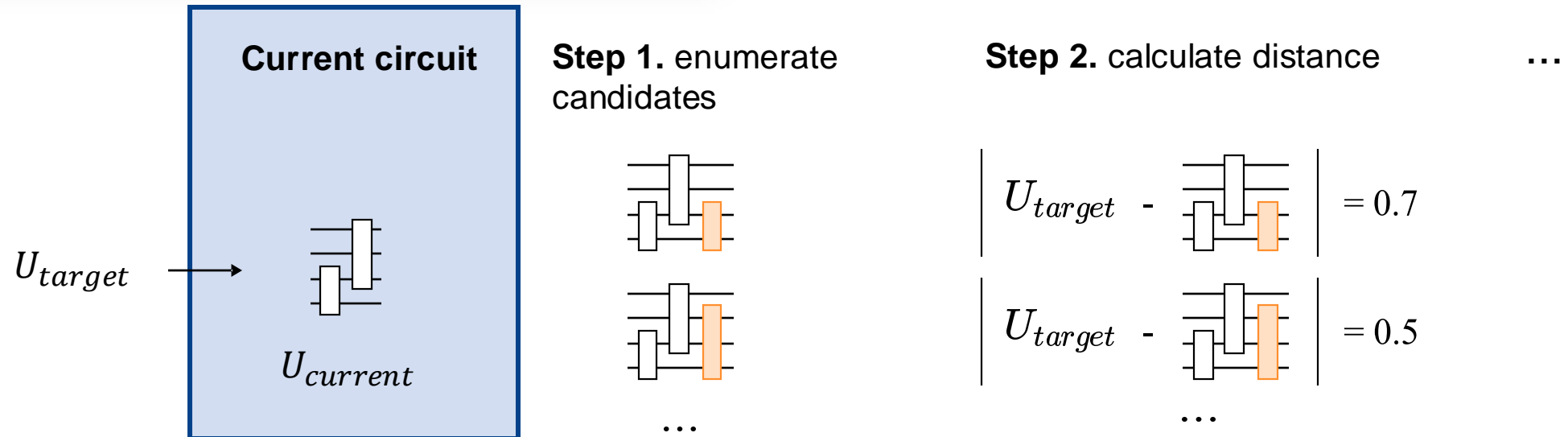




Candidates  and  should equal  $U_{target}U_{current}^{-1}$

# Downstream Model 2: Unitary Decomposition



## QuCT workflow

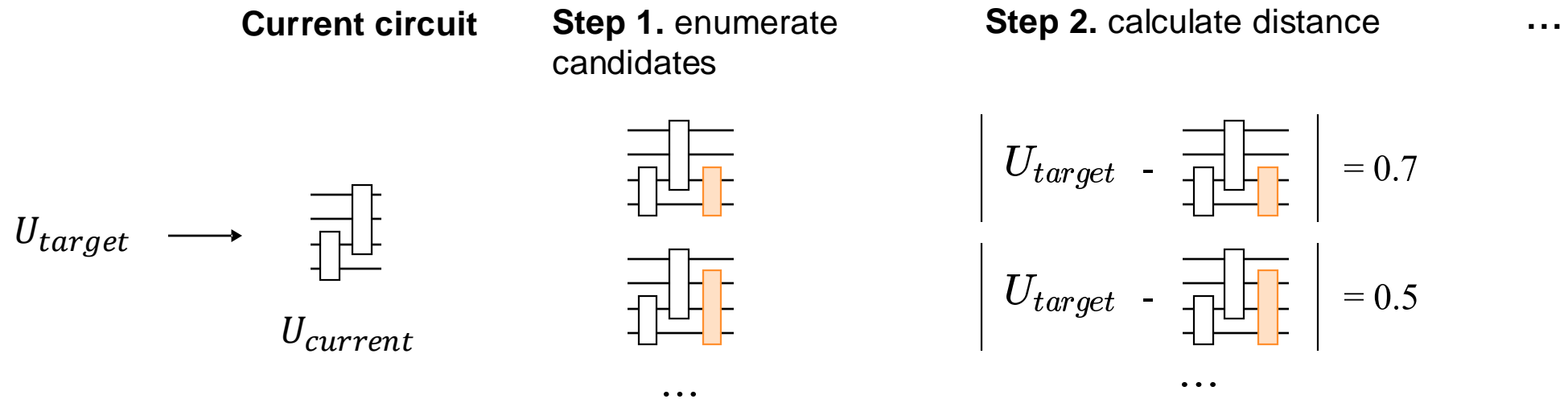




Candidates  and  should equal  $U_{target} U_{current}^{-1}$

# Downstream Model 2: Unitary Decomposition



## QuCT workflow

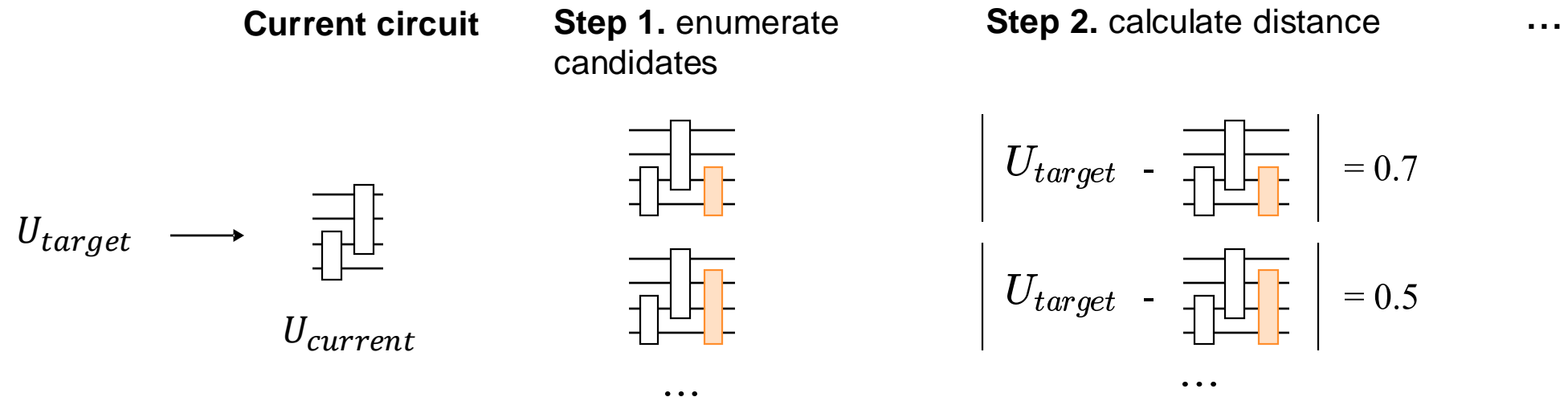




Candidates  and  should equal  $U_{target} U_{current}^{-1}$

# Downstream Model 2: Unitary Decomposition



## QuCT workflow



Candidates  and  should equal  $U_{target}U_{current}^{-1}$

Instead of exhaustive search, QuCT only try the candidates that have high probability of equaling  $U_{target}U_{current}^{-1}$

# Downstream Model 2: Unitary Decomposition



## QuCT workflow

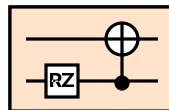
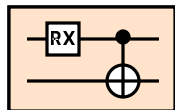
$$U_{target} U_{current}^{-1}$$

↓ U2V model

Gate vectors serve as  
search candidates

$$v_1 = [\dots] \quad v_2 = [\dots]$$

↓ Reconstruct



**U2V model:** a random forest model, trained  
by the pre-generated decomposition results.

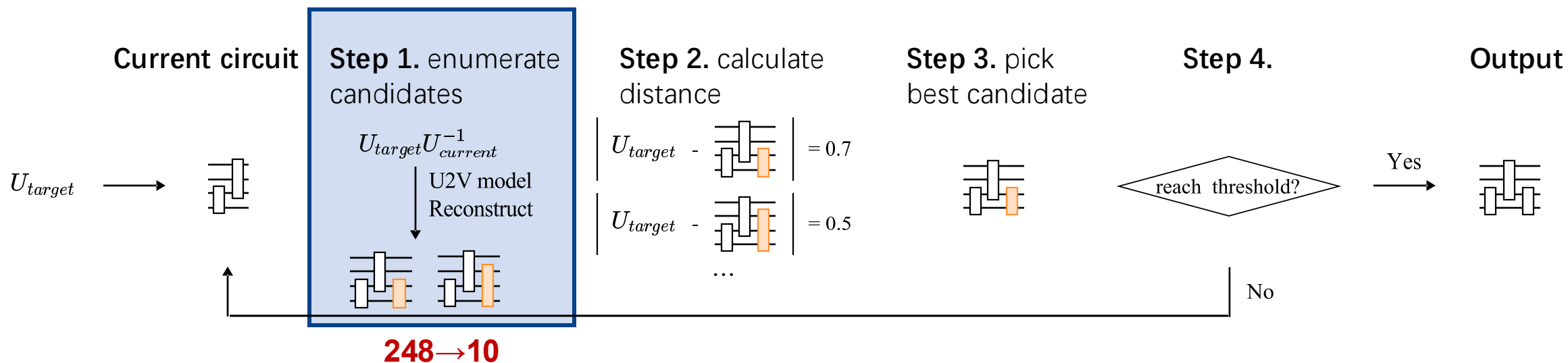
*Use gate vectors to construct good  
candidates.*



# Downstream Model 2: Unitary Decomposition



## QuCT workflow



**Comparison to the template-based method:** template-based approach has smaller design space, as it can only select candidates from a limited-size template library.

$1.8\times$  speedup and  $1.6\times$  gate reduction compared to the template-based method.

# API to Construct Unitary Decomposition Model



File:

- JanusQ/examples/ipynb/2\_5\_unitary\_decomposition.ipynb
- [https://janusq.github.io/tutorials/demo/2\\_5\\_unitary\\_decomposition](https://janusq.github.io/tutorials/demo/2_5_unitary_decomposition)

construct the  
decomposition  
dataset

```
from qiskit.quantum_info import random_unitary
from janusq.objects.backend import FullyConnectedBackend
from janusq.analysis.unitary_decomposition import U2VModel
from janusq.analysis.unitary_decomposition import decompose
```

construct unitary  
decomposition model

```
up_model = RandomwalkModel(n_step, 4 ** n_step, backend, directions=('parallel', 'next'))
u2v_model = U2VModel(up_model)
data = u2v_model.construct_data(dataset, multi_process=False)
u2v_model.train(data, n_qubits)
```

apply decomposition  
to random unitary

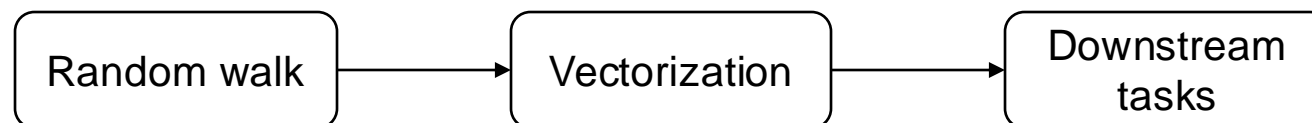
```
unitary = random_unitary(2**n_qubits).data
decompose(unitary, allowed_dist = 0.01, backend = backend, u2v_model = u2v_model ,
multi_process = True)
```

# Extending QuCT To More Downstream Tasks



File:

- JanusQ/examples/ipynb/2\_6\_extend\_framework\_bug\_identification.ipynb
- [https://janusq.github.io/tutorials/demo/2\\_6\\_extend\\_framework\\_bug\\_identification](https://janusq.github.io/tutorials/demo/2_6_extend_framework_bug_identification)



```
from janusq.analysis.vectorization import RandomwalkModel

class DownstreamModel():
    def __init__(self, upstream_mdoel: RandomwalkModel) :
        self.upstream_mdoel = upstream_mdoel
```

Downstream task implementation

# Extending QuCT To More Downstream Tasks



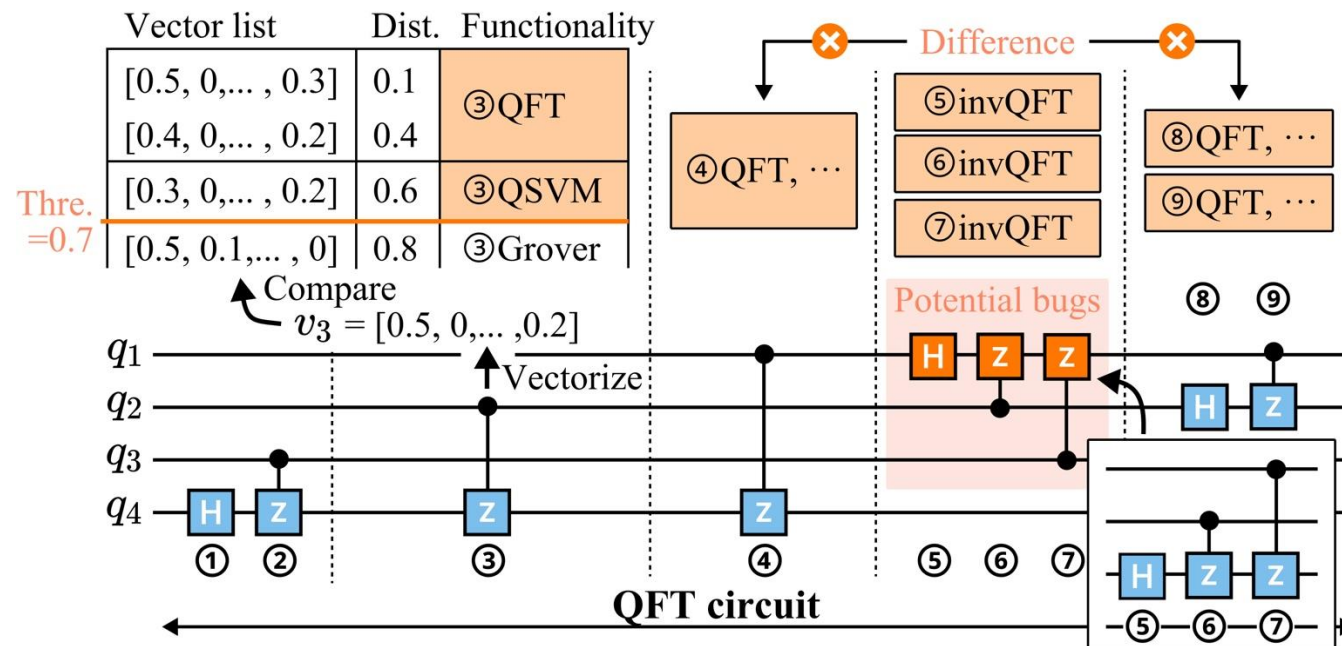
File:

- JanusQ/examples/ipynb/2\_6\_extend\_framework\_bug\_identification.ipynb
- [https://janusq.github.io/tutorials/demo/2\\_6\\_extend\\_framework\\_bug\\_identification](https://janusq.github.io/tutorials/demo/2_6_extend_framework_bug_identification)

For example: Bug Identification

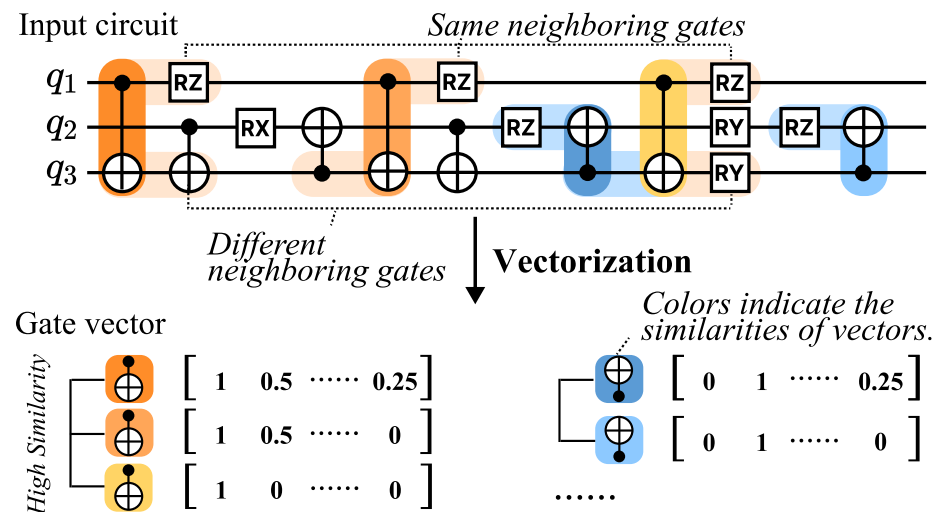
1. Identify the possible functionality

2. Identify the abnormal functionality (different from neighbors)



- Random walk-based method to extract contextual and topological circuit feature.
- Accurate circuit fidelity prediction via modeling gate interactions.
- Fast unitary decomposition via pruning candidate space.

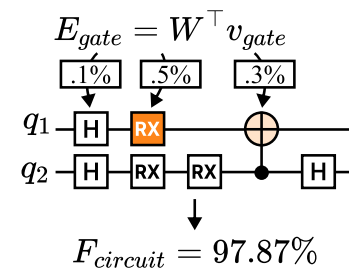
## Upstream Model:



## Downstream Model:

### Circuit Fidelity Prediction

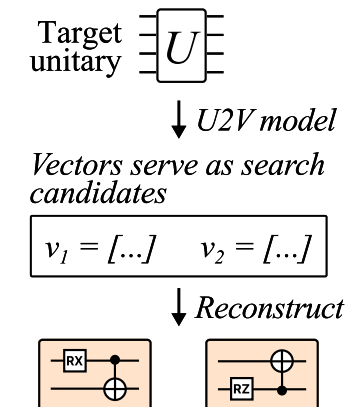
a) Circuit fidelity prediction



b) Compilation- and calibration-level optimizations

More tasks: gate cancellation, bug detection ...

### Unitary Decomposition





# Thanks for listening

## **QuCT: A Framework for Analyzing Quantum Circuit by Extracting Contextual and Topological Features**

Siwei Tan, Congliang Lang, Liang Xiang, Shudi Wang, Xinghui Jia, Ziqi Tan, Tingting Li, Jieming Yin, Yongheng Shang, Andre Python, Liqiang Lu\*, and Jianwei Yin\*