WELCOME TO TUTORIAL

Session 3 JanusQ-CT: A Framework for Analyzing Quantum Circuit by Extracting Contextual and Topological Features

https://janusq.github.io/tutorials/

College of Computer Science and Technology,
Zhejiang University

ASPLOS 2024

Outline of Presentation

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

Background

Compilation of a quantum program

Quantum program $\xrightarrow{\text{Quantum circuit}}$ Circuit-level compilation $\xrightarrow{\text{Quantum circuit}}$ Pulse-level compilation $\xrightarrow{\text{Pulse}}$ Board-level compilation $\xrightarrow{\text{Board directives}}$ Quantum computer

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：

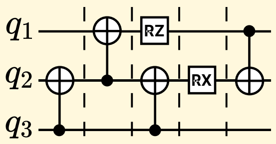
- Input: pulses — Output: Board directives

Background

Key passes of quantum circuit compilation

Quantum program → Quantum circuit → **Circuit-level compilation** → Quantum circuit → **Pulse-level compilation** → Pulse → **Board-level compilation** → Board directives → **Quantum computer**

Pass 1: Unitary decomposition



compute

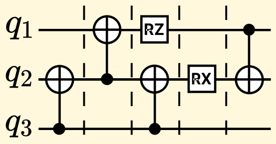
small complexity

$$\begin{bmatrix} 0 & e^{i-\pi/2} & 0 & 0 \\ 0 & 0 & 0 & e^{i-\pi/2} \\ e^{i\pi/2} & 0 & 0 & 0 \\ 0 & 0 & 0 & e^{i\pi/2} \end{bmatrix}$$

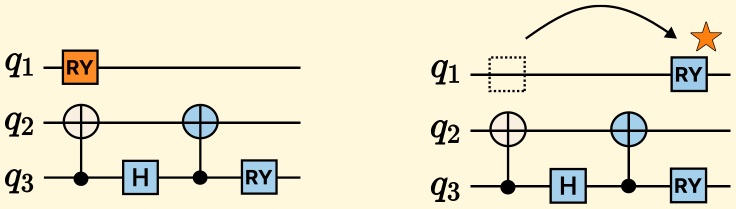
$$\begin{bmatrix} 0 & e^{i-\pi/2} & 0 & 0 \\ 0 & 0 & 0 & e^{i-\pi/2} \\ e^{i\pi/2} & 0 & 0 & 0 \\ 0 & 0 & 0 & e^{i\pi/2} \end{bmatrix}$$

decompose

large complexity



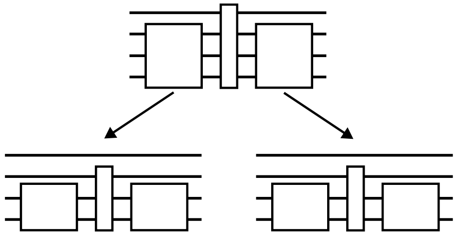
Pass 2: Fidelity prediction and optimization



Optimize the noise while keeping the equivalence of circuits

Challenges of Unitary Decomposition

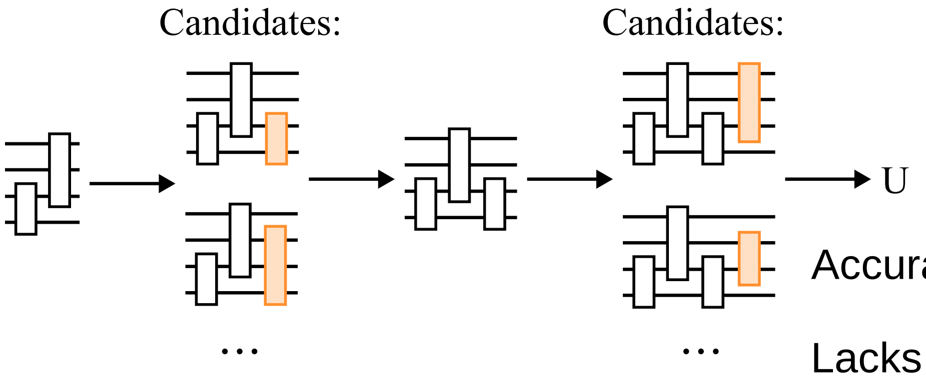
Unitary decomposition



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

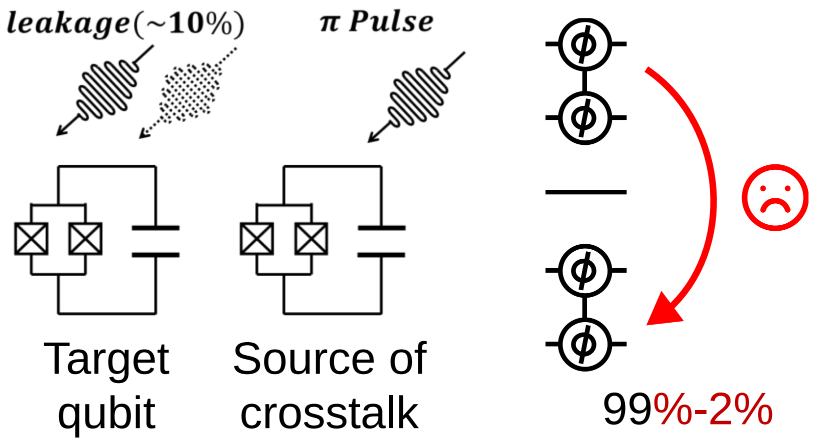
O(4^N) #Gate **O(4^N) Time**

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

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)	**4-28%**	**3-36%**	**2-12%**	**3-17%**

Fidelity prediction

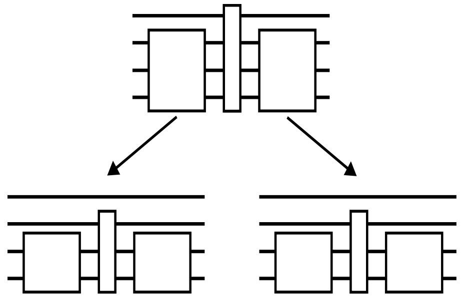
Not one-shot

[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.

Current Compilation Methods

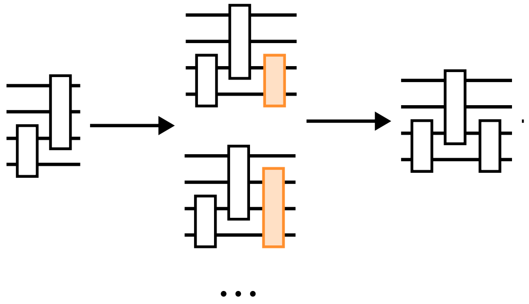
Unitary decomposition

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

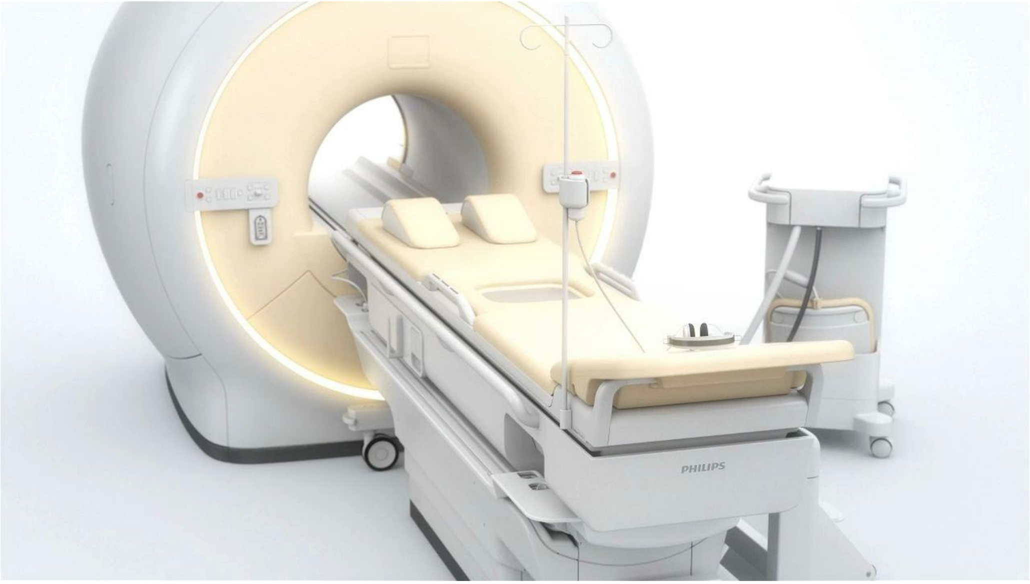
They face a trade-off between the efficiency and accuracy

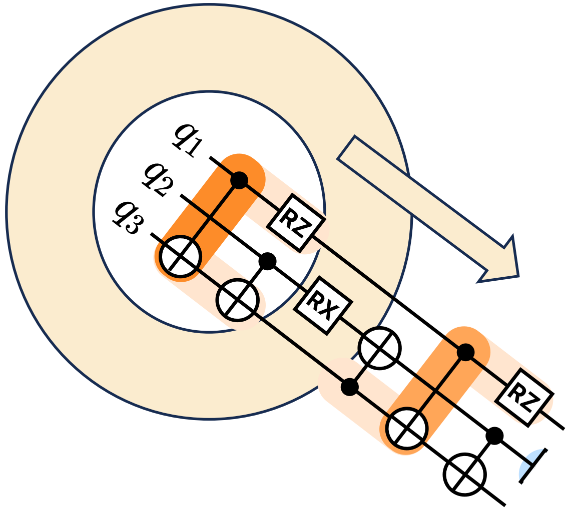
Outline of Presentation

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

Janus-CT Overview

Origin of the name



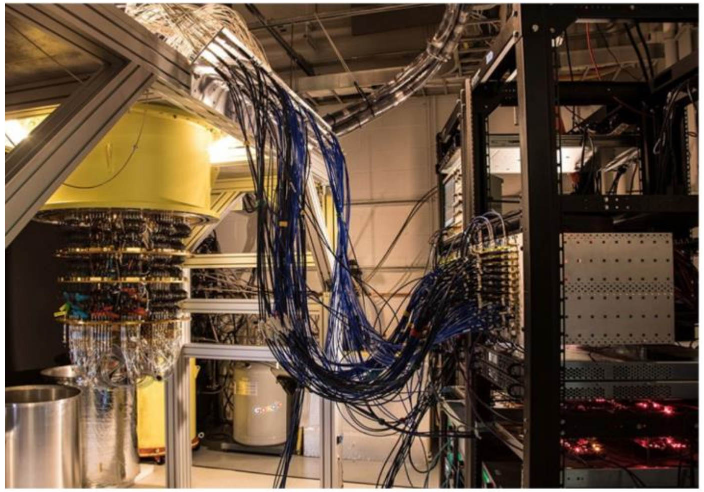


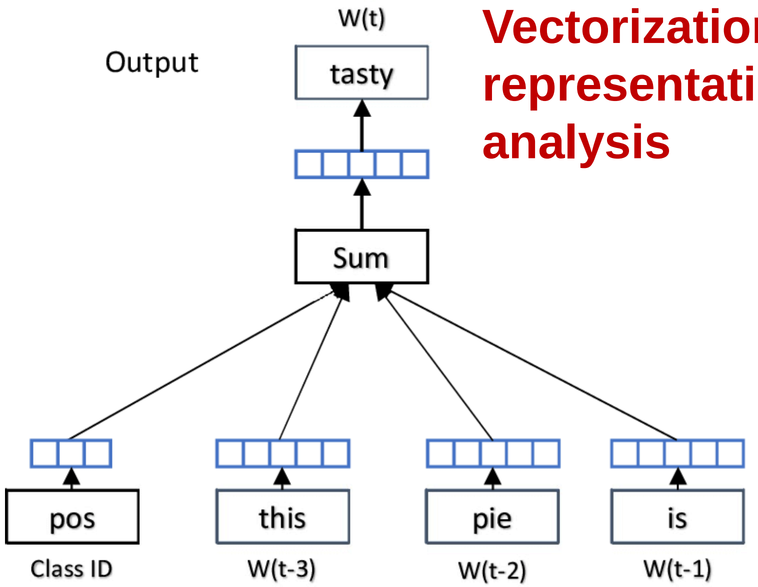
Computerized Tomography

Analyzing Quantum Circuit by Contextual and Topological Features

Janus-CT Insight

Solution: Implement circuit topology and context-aware gate vectorization





Vectorization is a more efficient representation for further analysis

Quantum	NLP
Fidelity prediction	grammar analysis
Circuit generation	Test generation

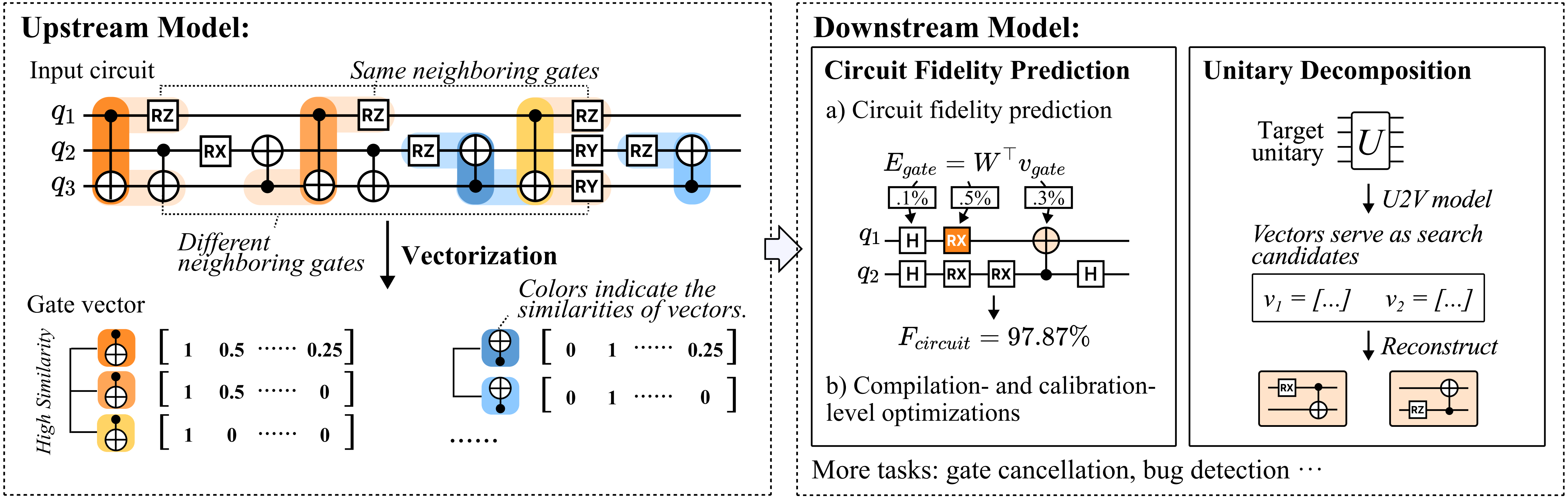
Quantum circuits are implemented via pulses. There are **interactions between wirings of qubits.**

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

Quantum program analysis and NLP have similar tasks

Janus-CT Overview

Each model is one-shot generated

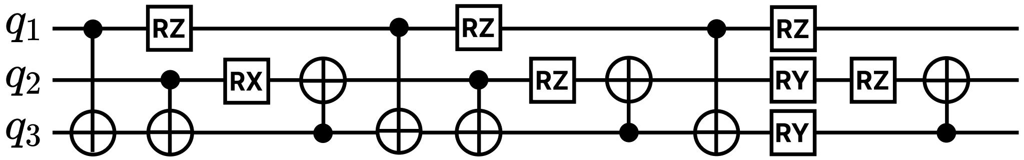


Random walk → Vectorization → Fidelity prediction or Unitary decomposition

Janus-CT Overview

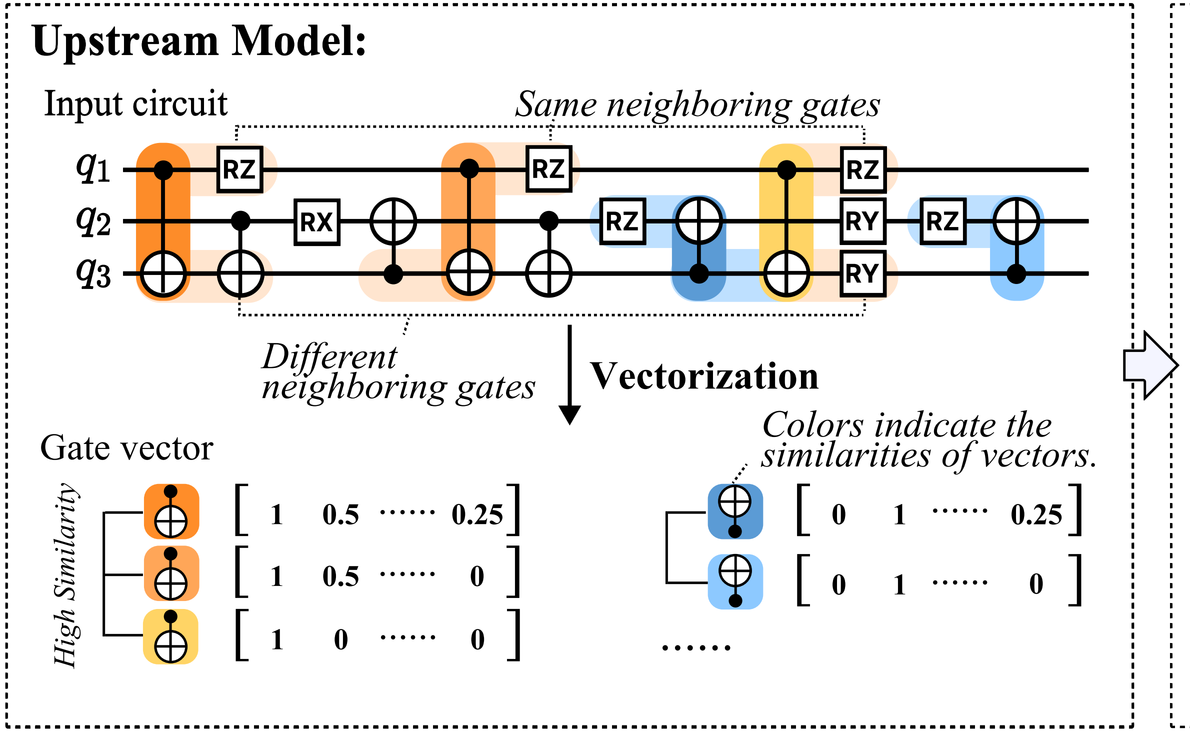
Upstream Model:

Input circuit



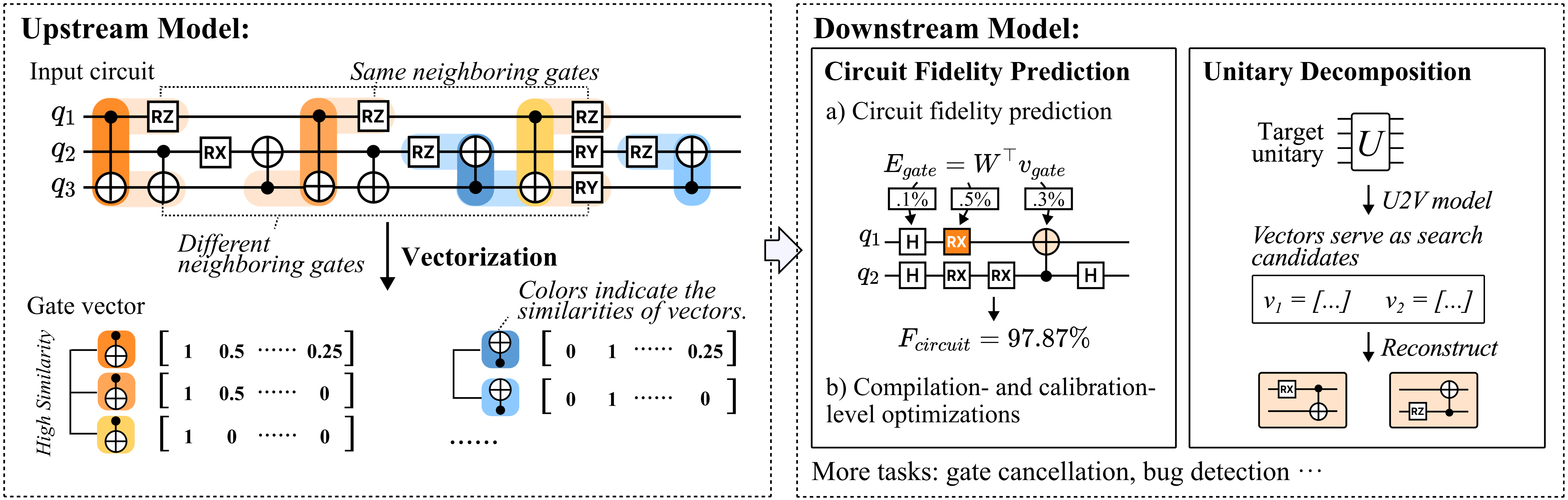
Random walk

Janus-CT Overview



Random walk → Vectorization

Janus-CT Overview



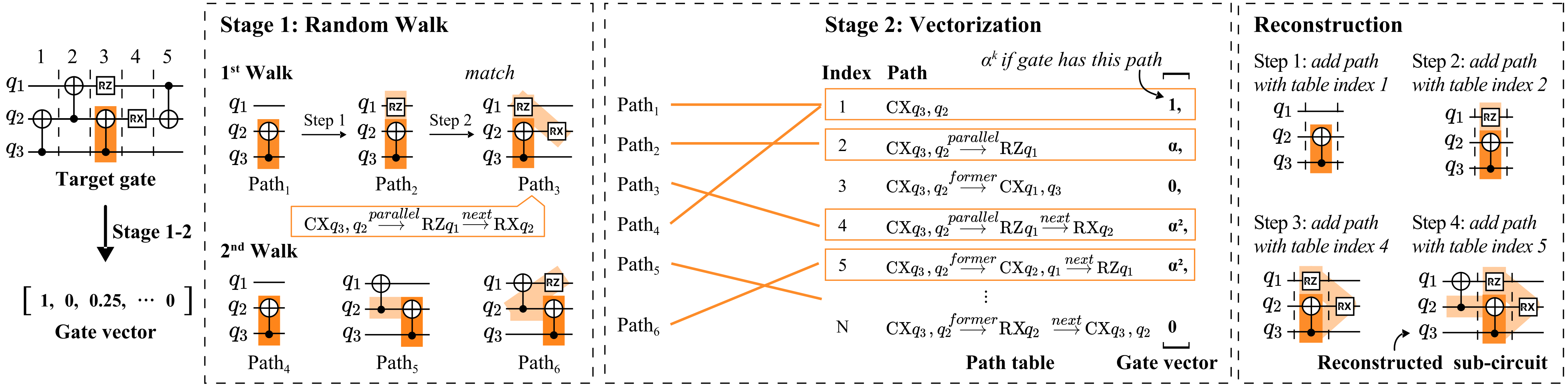
Random walk → Vectorization → Fidelity prediction or Unitary decomposition

Outline of Presentation

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

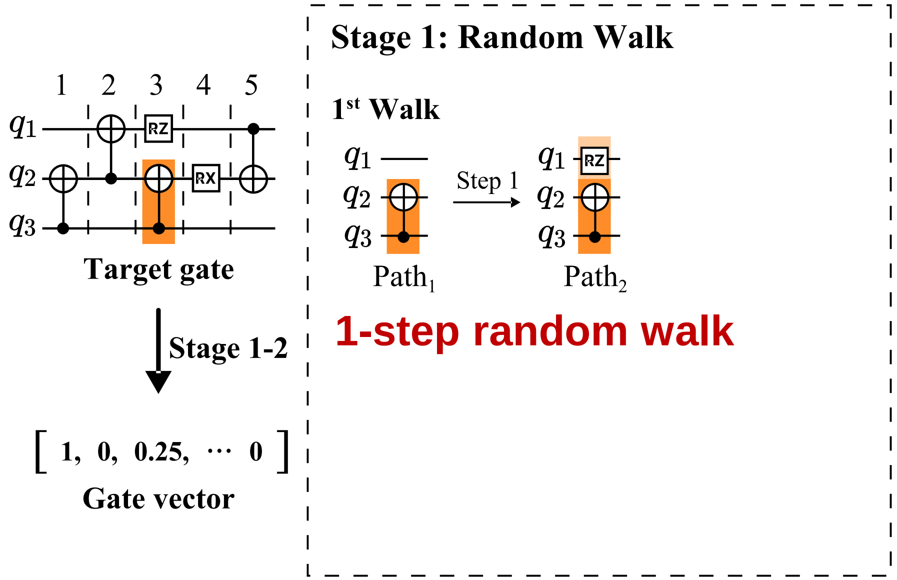
Upstream Model: Circuit Feature Extraction

Two-step vectorization flow



Upstream Model: Circuit Feature Extraction

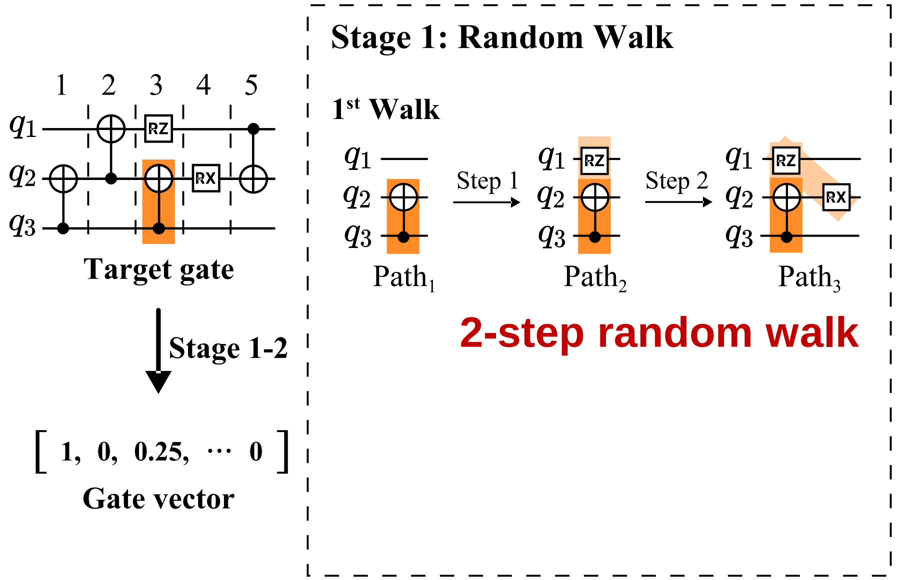
Two-step vectorization flow



Step 1: Extract features as paths.

Upstream Model: Circuit Feature Extraction

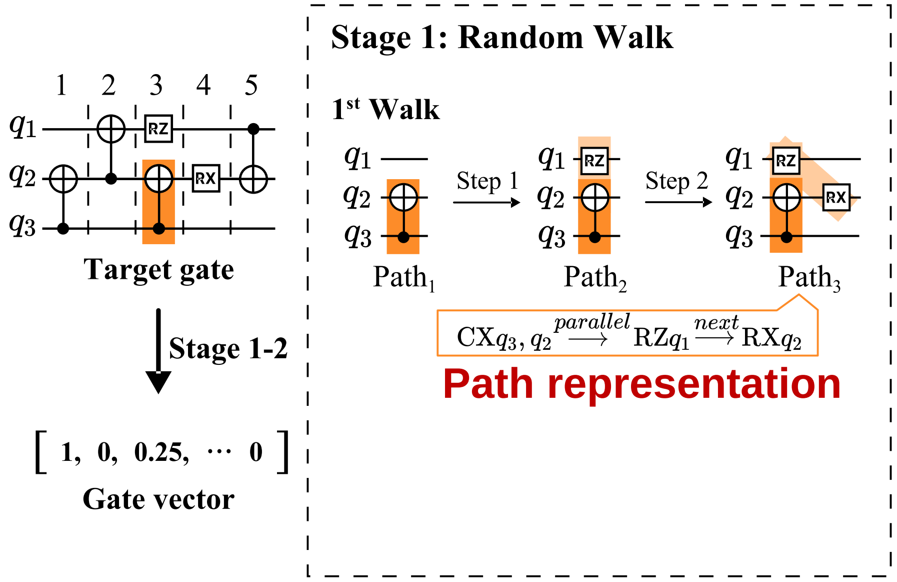
Two-step vectorization flow



Step 1: Extract features as paths.

Upstream Model: Circuit Feature Extraction

Two-step vectorization flow

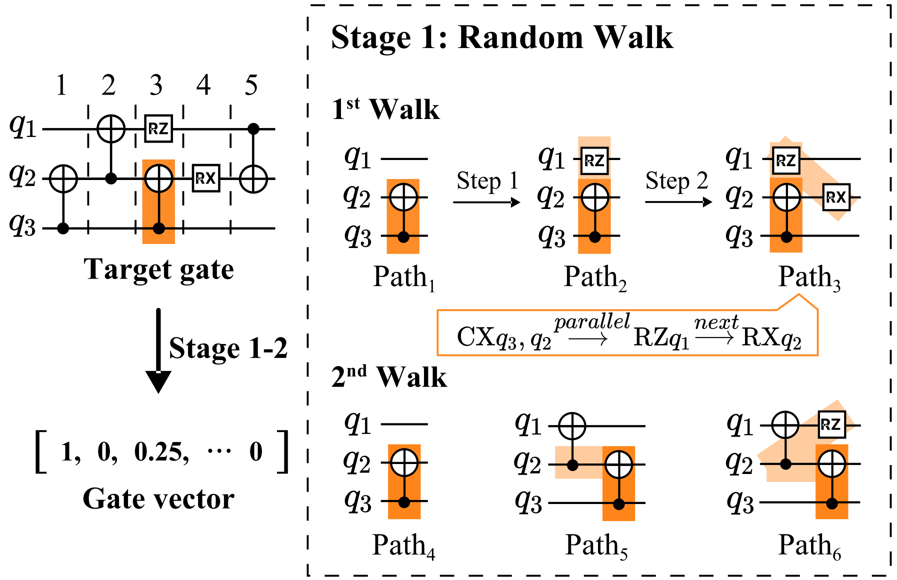


Step 1: Extract features as paths.

Upstream Model: Circuit Feature Extraction

ZHEJIANG UNIVERSITY

Two-step vectorization flow

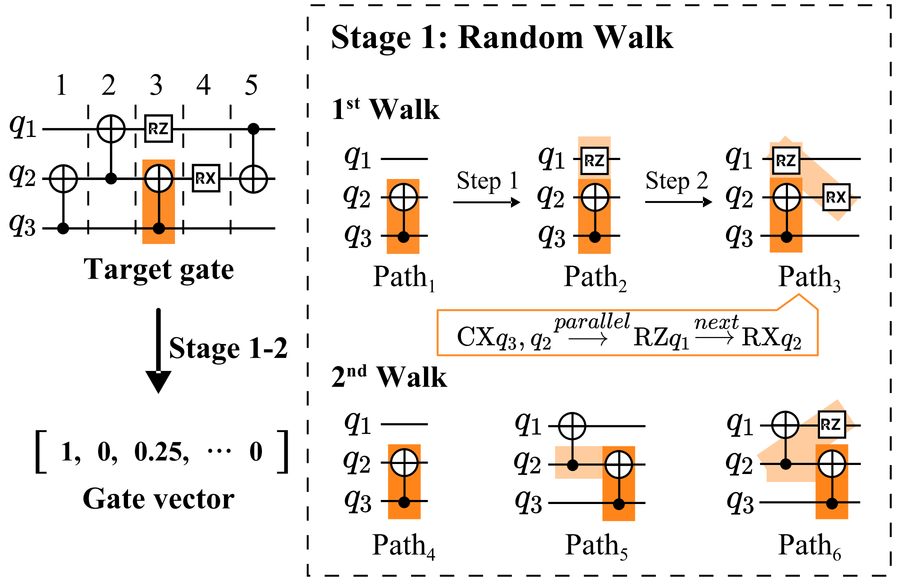


Multiple paths

Step 1: Extract features as paths.

Upstream Model: Circuit Feature Extraction

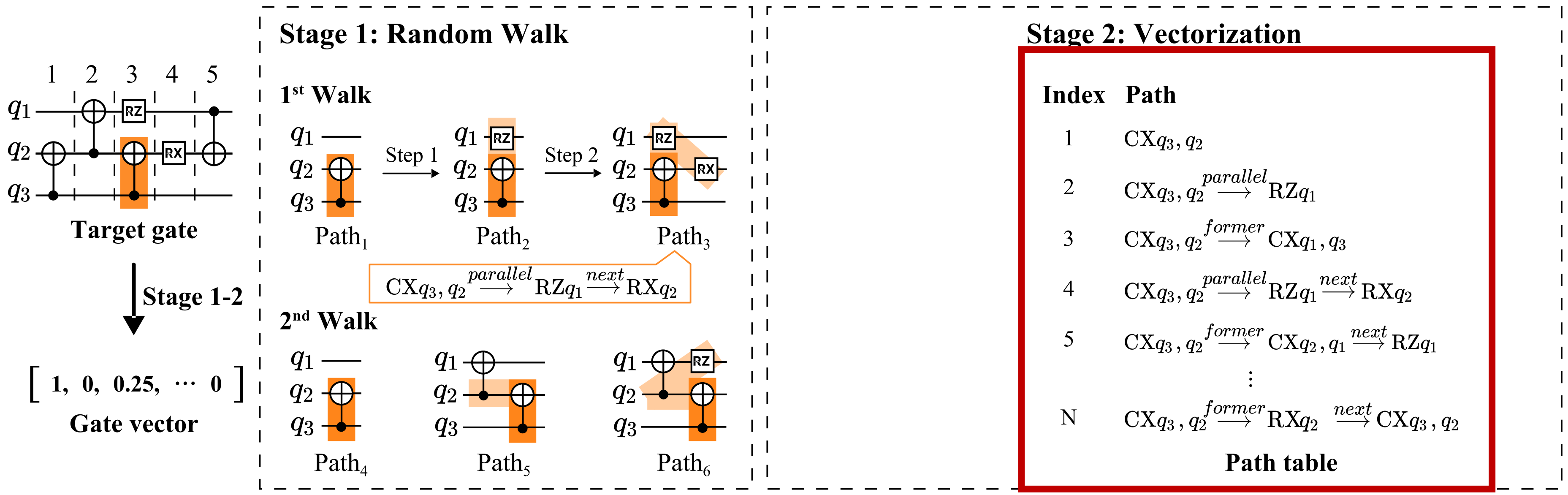
Two-step vectorization flow



Obtain vector.

Upstream Model: Circuit Feature Extraction

Two-step vectorization flow

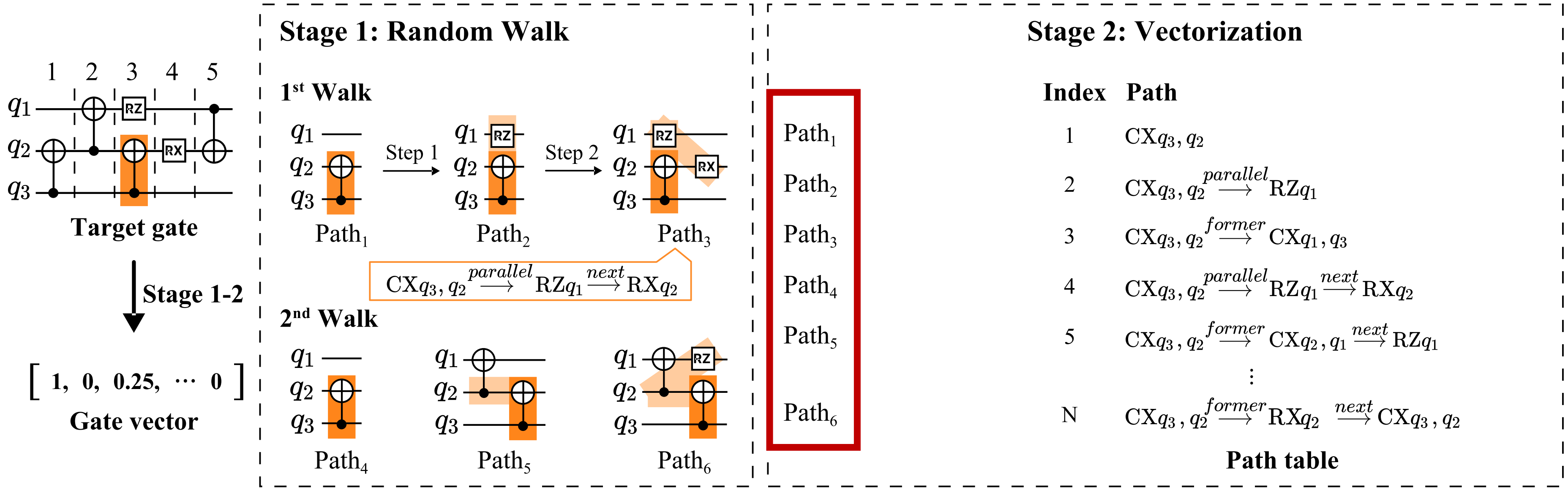


Pre-generated

Obtain vector.

Upstream Model: Circuit Feature Extraction

Two-step vectorization flow

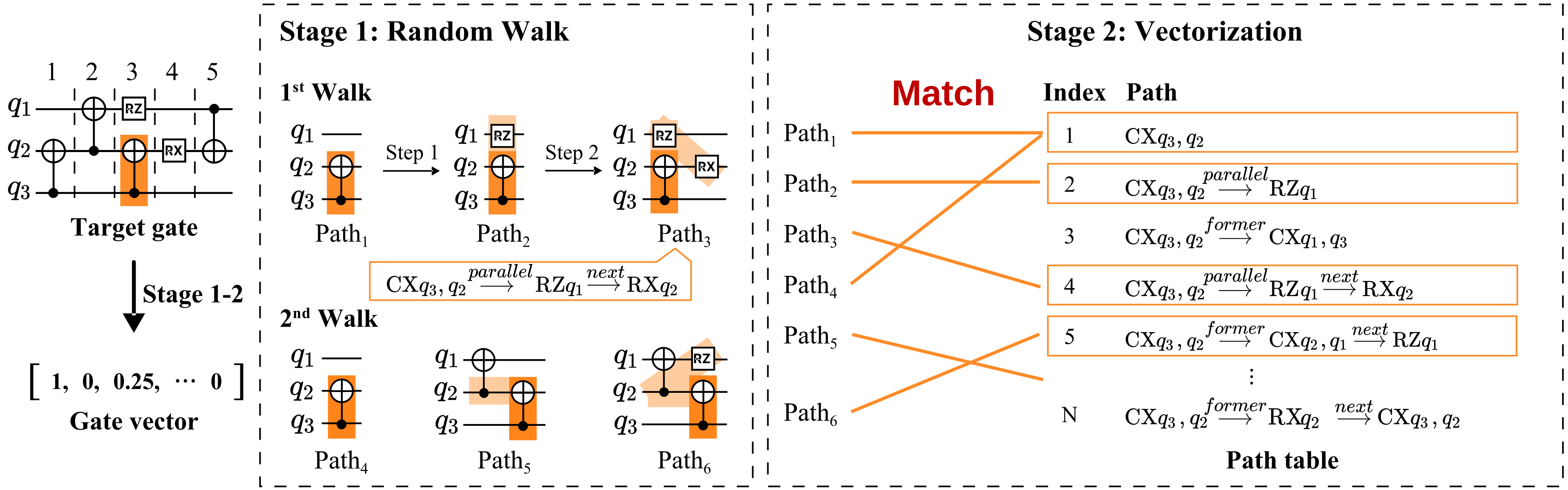


From random walk **Pre-generated**

Obtain vector.

Upstream Model: Circuit Feature Extraction

Two-step vectorization flow



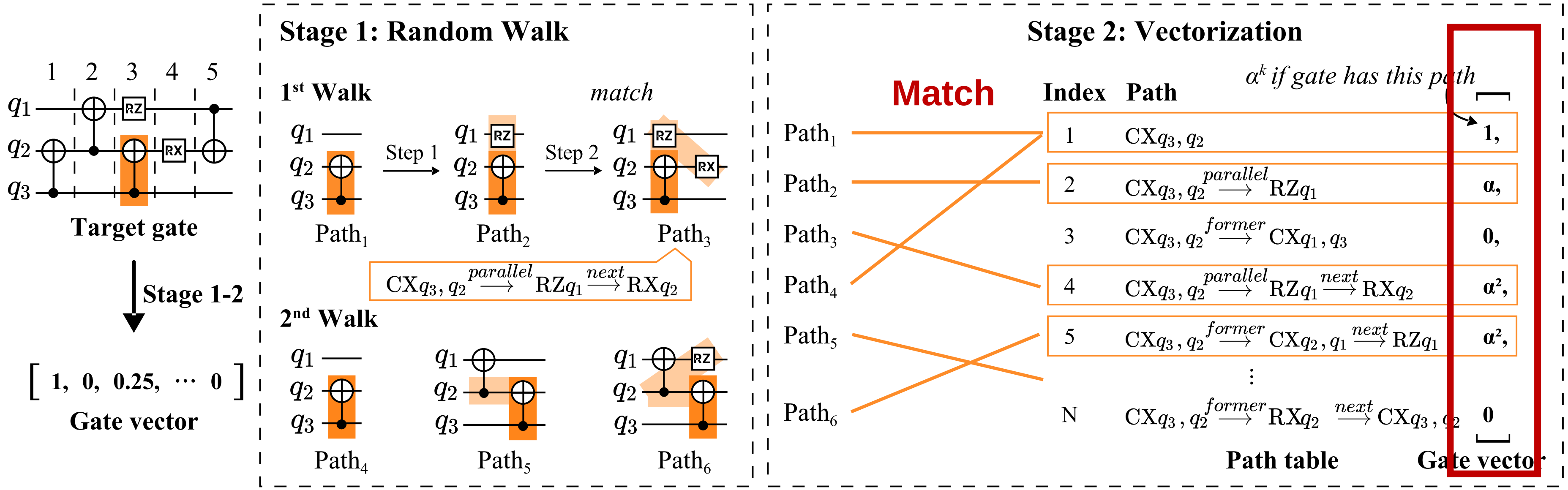
From random walk

Pre-generated

Obtain vector.

Upstream Model: Circuit Feature Extraction

Two-step vectorization flow



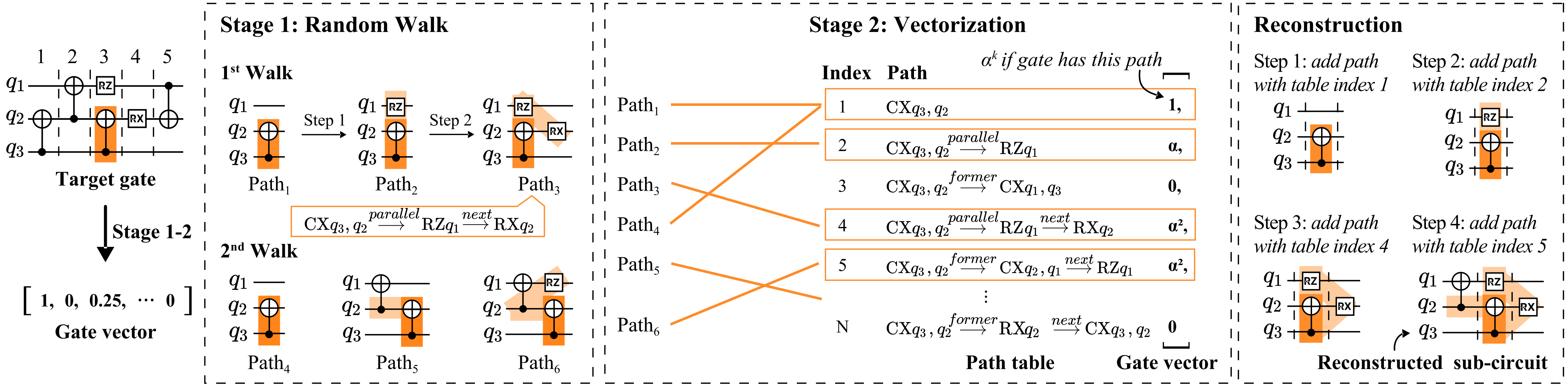
From random walk

Pre-generated

Obtain vector.

Upstream Model: Circuit Feature Extraction

Two-step vectorization flow



Reconstruct circuit.

API to Construct Upstream Model

File:
- examples/3-1.vectorization.ipynb
- https://janusq.github.io/tutorials/Demonstrations/3-1.vectorization.ipynb

define backend

generate fidelity dataset

construct model using random walk

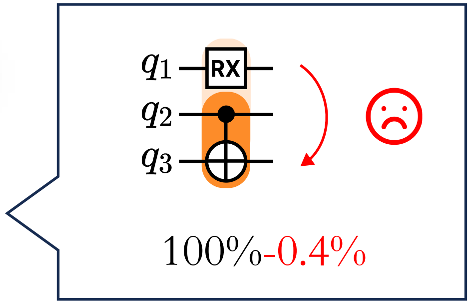
```
from janusq.analysis.vectorization import RandomwalkModel, extract_device
from janusq.data_objects.backend import GridBackend, FullyConnectedBackend, LinearBackend
# define the information of the quantum device
n_qubits = 6
backend = GridBackend(2, 3)

# generate a dataset including varous random circuits
from janusq.data_objects.random_circuit import random_circuits, random_circuit
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)

print('length of the path table is', len(up_model.pathtable))
```

Outline of Presentation

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

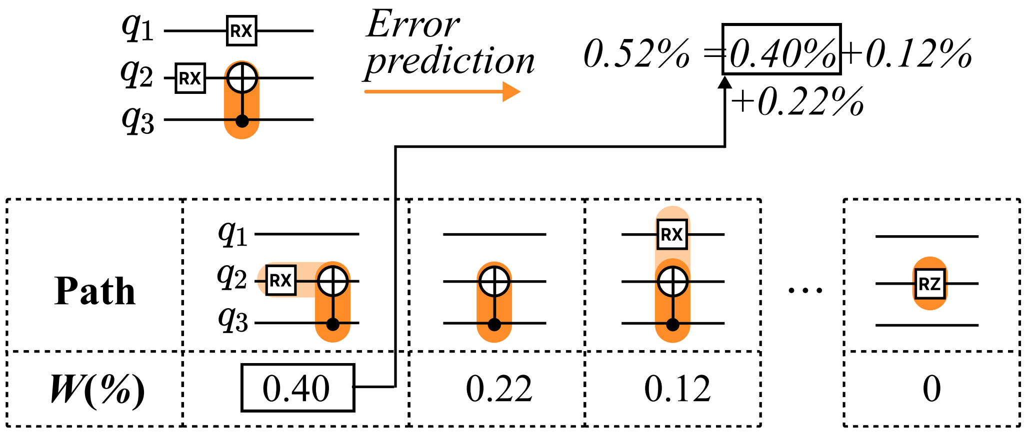
Gate error prediction

$$E(v_i) = W^\intercal v_i$$



Model dependent error

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



Downstream Model 1: Circuit Fidelity Prediction

Gate error prediction

$$E(v_i) = W^\intercal v_i$$

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

$$E(v_i) = W^{\intercal} v_i$$

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,* $F_{ground-truth}$) $\cdots$, $F_{ground-truth}$: 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:
- examples/3-2.fidelity_prediction_simulator.ipynb
- https://janusq.github.io/tutorials/Demonstrations/3-2.fidelity_prediction_simulator.ipynb

Load the fidelity dataset

```
from janusq.dataset import real_qc_5bit
from janusq.data_objects.random_circuit import random_circuit
```

Construct upstream model

```
circuits, fidelities = real_qc_5bit
print(len(circuits))

from janusq.analysis.vectorization import RandomwalkModel
from janusq.data_objects.backend import FullyConnectedBackend
backend = FullyConnectedBackend(5)
up_model = RandomwalkModel(n_steps = 1, n_walks = 10, backend = backend, circuits = circuits)
```

Construct fidelity prediction model

```
from janusq.analysis.fidelity_prediction import FidelityModel
fidelity_model = FidelityModel(up_model)
fidelity_model.train((circuits, fidelities), multi_process = False)
```

Predict the fidelity of the given circuit

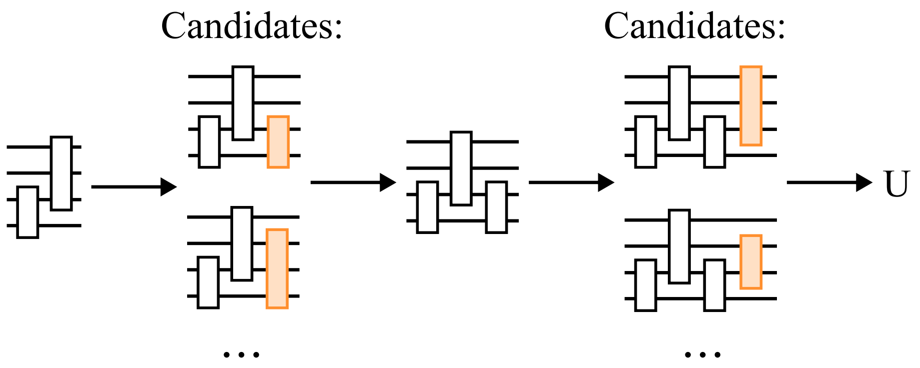
```
circuit = random_circuit(backend, n_gates = 100, two_qubit_prob = 0.5 )
fidelity_model.predict_circuit_fidelity(circuit)
```

Outline of Presentation

- Background and challenges
- Janus-CT 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



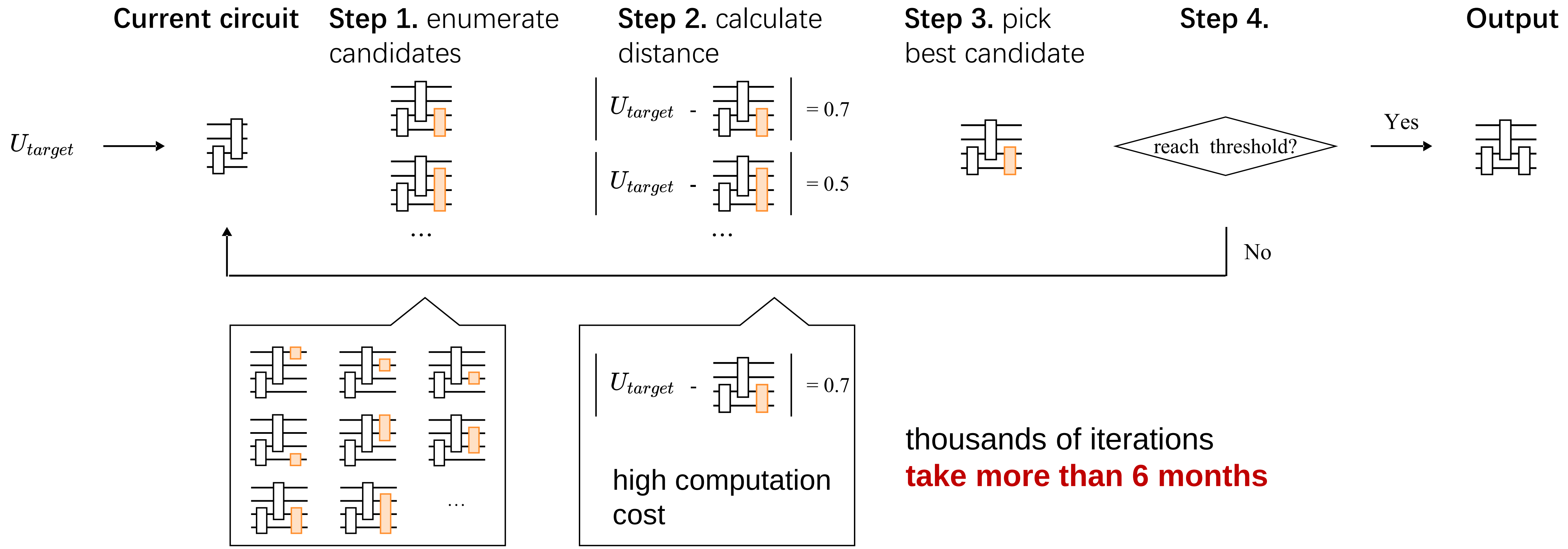
Search-based method

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

5-qubit unitary decomposition

Downstream Model 2: Unitary Decomposition

QFAST workflow



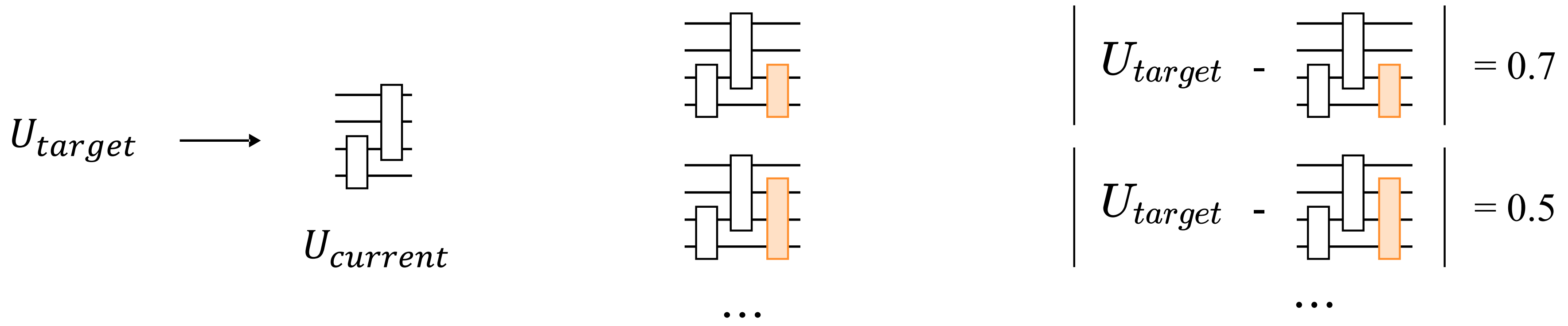
248 candidates for 8 qubits

60s/candidate

Downstream Model 2: Unitary Decomposition

Janus-CT workflow

Current circuit | **Step 1.** enumerate candidates | **Step 2.** calculate distance | …



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

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

Downstream Model 2: Unitary Decomposition

Janus-CT workflow

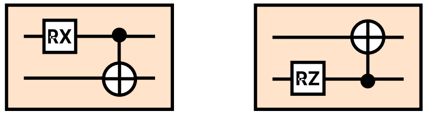
$$U_{target}U_{current}^{-1}$$

$\downarrow$ U2V model

Gate vectors serve as search candidates

$$\boxed{v_1 = [\ldots] \quad v_2 = [\ldots]}$$

$\downarrow$ Reconstruct

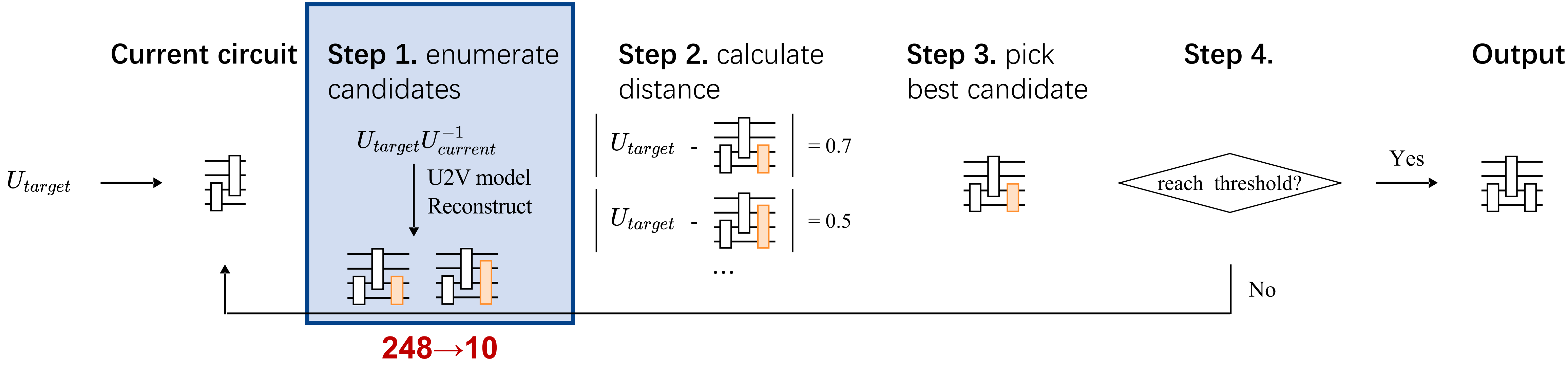


U2V model: a random forest model, trained by the pre-generated decomposition results.

Use gate vectors to construct good candidates.

Downstream Model 2: Unitary Decomposition

Janus-CT workflow



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

1.8× speedup and 1.6 × gate reduction compared to the template-based method.

API to Construct Unitary Decomposition Model

浙江大學 ZHEJIANG UNIVERSITY

File:
- examples/ 3-5.unitary_decomposition.ipynb
- https://janusq.github.io/tutorials/Demonstrations/3-5.unitary_decomposition.ipynb

Construct the decomposition dataset

```
from janusq.data_objects.random_circuit import random_circuits
from janusq.data_objects.backend import FullyConnectedBackend
n_qubits = 5
backend = FullyConnectedBackend(n_qubits)
dataset = random_circuits(backend = backend, n_circuits=50, n_gate_list=[30, 50, 100],
two_qubit_prob_list=[.4], reverse=True)
```

Construct unitary decomposition model

```
from janusq.analysis.unitary_decompostion import U2VModel
from janusq.analysis.vectorization import RandomwalkModel
vec_model = RandomwalkModel(2, 16, backend, directions=('parallel', 'next'), circuits =
dataset)
u2v_model = U2VModel(vec_model)
data = u2v_model.construct_data(dataset, multi_process=False)
u2v_model.train(data, n_qubits)
```

Generate a random unitary

```
from qiskit.quantum_info import random_unitary
unitary = random_unitary(2**n_qubits).data
```

Apply decomposition

```
from janusq.analysis.unitary_decompostion import decompose
decompose(unitary, allowed_dist = 0.01, backend = backend, u2v_model = u2v_model ,
multi_process = True)
```

Extending Janus-CT To More Downstream Tasks

File:
- examples/3-6.extend_framework_bug_identification.ipynb
- https://janusq.github.io/tutorials/Demonstrations/3-6.extend_framework_bug_identification

Random walk → Vectorization → Downstream tasks

```
from janusq.analysis.vectorization import RandomwalkModel

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

Downstream task implementation

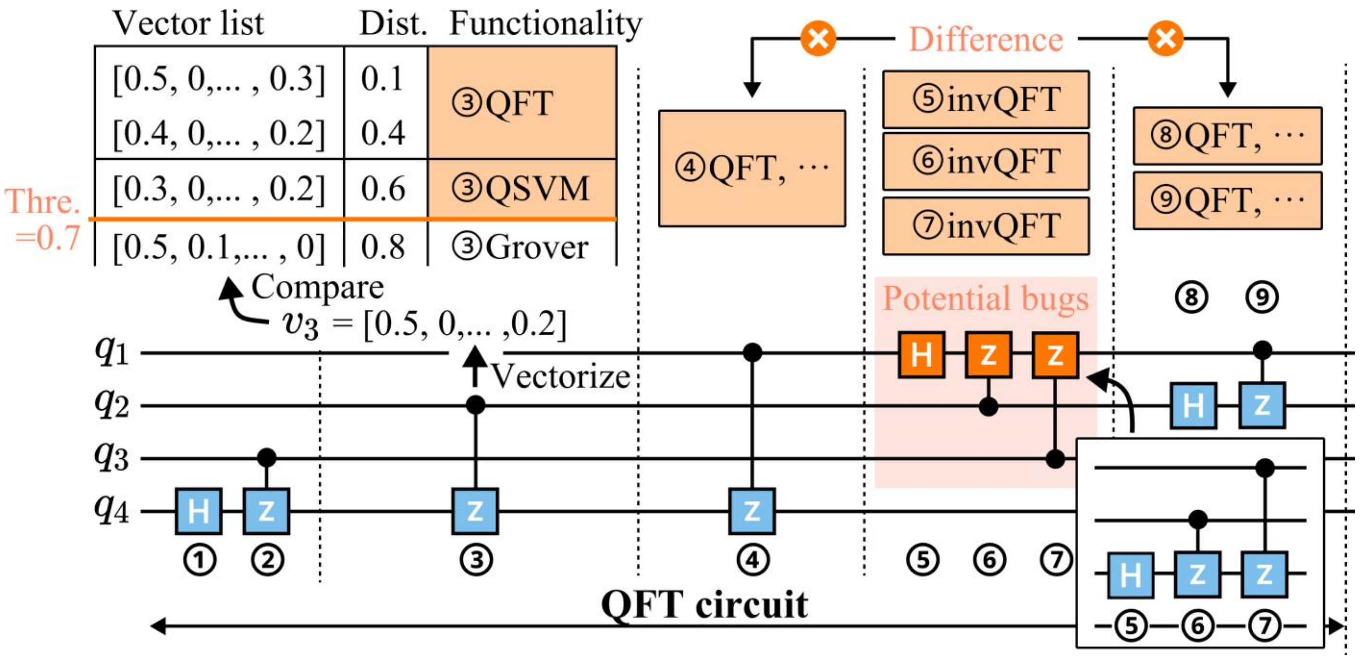
Extending Janus-CT To More Downstream Tasks

File:
- examples/3-6.extend_framework_bug_identification.ipynb
- https://janusq.github.io/tutorials/Demonstrations/3-6.extend_framework_bug_identification

For example: Bug Identification

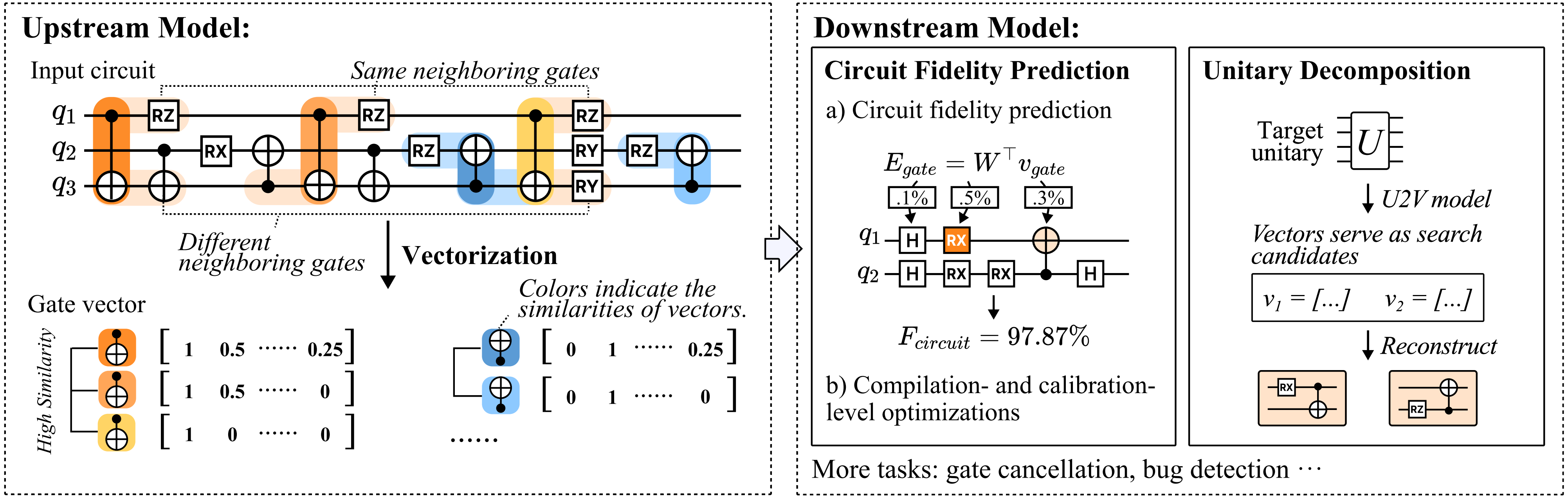
1. Identify the possible functionality

2. Identify the abnormal functionality (different from neighbors)



Conclusion

- 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.



Thanks for listening

Janus-CT: 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*, Jianwei Yin*