



# HPCA 2025 Tutorial

## Topic 3. MorphQPV: Exploiting Isomorphism in Quantum Programs to Facilitate Confident Verification



JanusQ  
Cloud

Speaker: Siwei Tan

College of Computer Science and Technology  
Zhejiang University (ZJU)

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



Siwei Tan

[siweitan@zju.edu.cn](mailto:siweitan@zju.edu.cn)

**Siwei Tan** received the Ph.D. degree in computer science from Zhejiang University (ZJU) in 2024 and keep staying at Zhejiang University as a Tenure-track Assistant Professor. He is interested in the quantum software, quantum hardware, and machine learning. He has published more than 20 papers in top international journals and conferences such as ASPLOS, MICRO, HPCA, DAC, ICCAD, FSE, TVCG, et al.

**[FSE 2025] Siwei Tan**, Liqiang Lu, Debin Xiang, et al. “HornBro: Homotopy-like Method for Automated Quantum Program Repair”.

**[ASPLOS 2024] Siwei Tan**, Debing Xiang, Liqiang Lu, et al. “MorphQPV: Exploiting Isomorphism in Quantum Programs to Facilitate Confident Verification”.

**[HPCA 2023] Siwei Tan**, Mingqian Yu, Liqiang Lu, et al. “HyQSAT: A Hybrid Approach for 3-SAT Problems by Integrating Quantum Annealer with CDCL”.

**[MICRO 2023] Siwei Tan**, Congliang Lang, Liang Xiang, et al. “QuCT: A Framework for Analyzing Quantum Circuit by Extracting Contextual and Topological Features”.

# Outline of Presentation

---



- **Background and Challenges**
- Overview of MorphQPV
- Assertion Statement and Validation
- Experiment
- API of MorphQPV

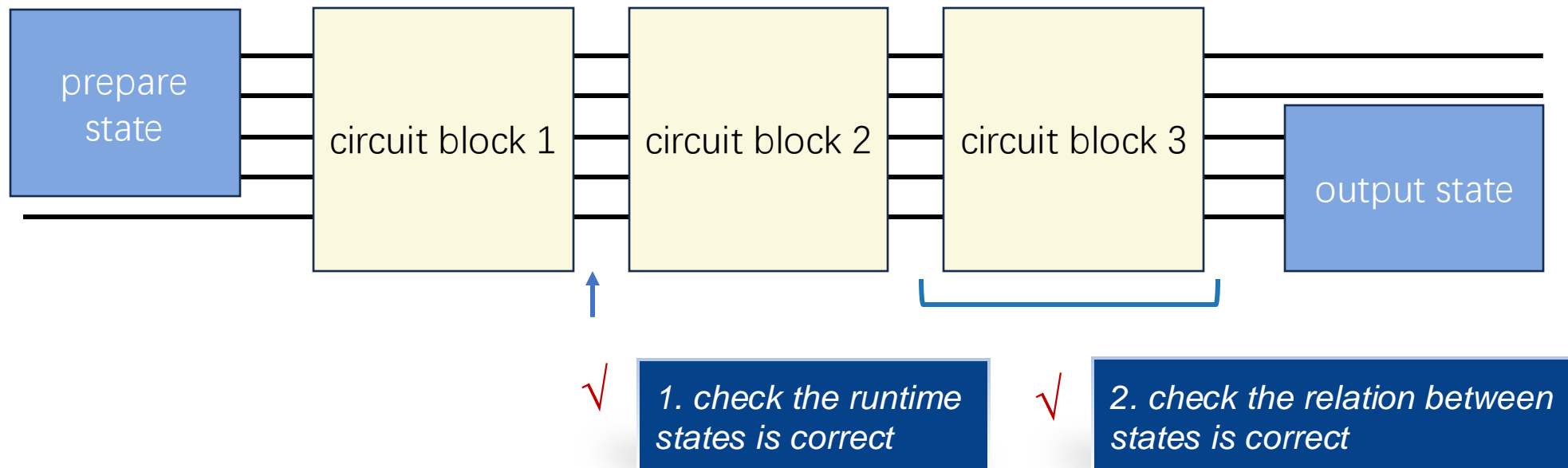
# Ensure Program Is Correct?



Quantum program verification is **a fundamental theoretical method** for reliable quantum computing, which aims to ensure that quantum programs have correct behavior and achieve desired results during execution.

## Process to check the program is correct

1. Check the relationship between the states in each stage
2. There may be mid-measurement or feedback



Described as **an assertion**, which is defined as a predicate. The predicate is expected to be true if there are no bugs.

# Ensure Program Is Correct?



Quantum program verification is **a fundamental theoretical method** for reliable quantum computing, which aims to ensure that quantum programs have correct behavior and achieve desired results during execution.

## Process to check the program is correct

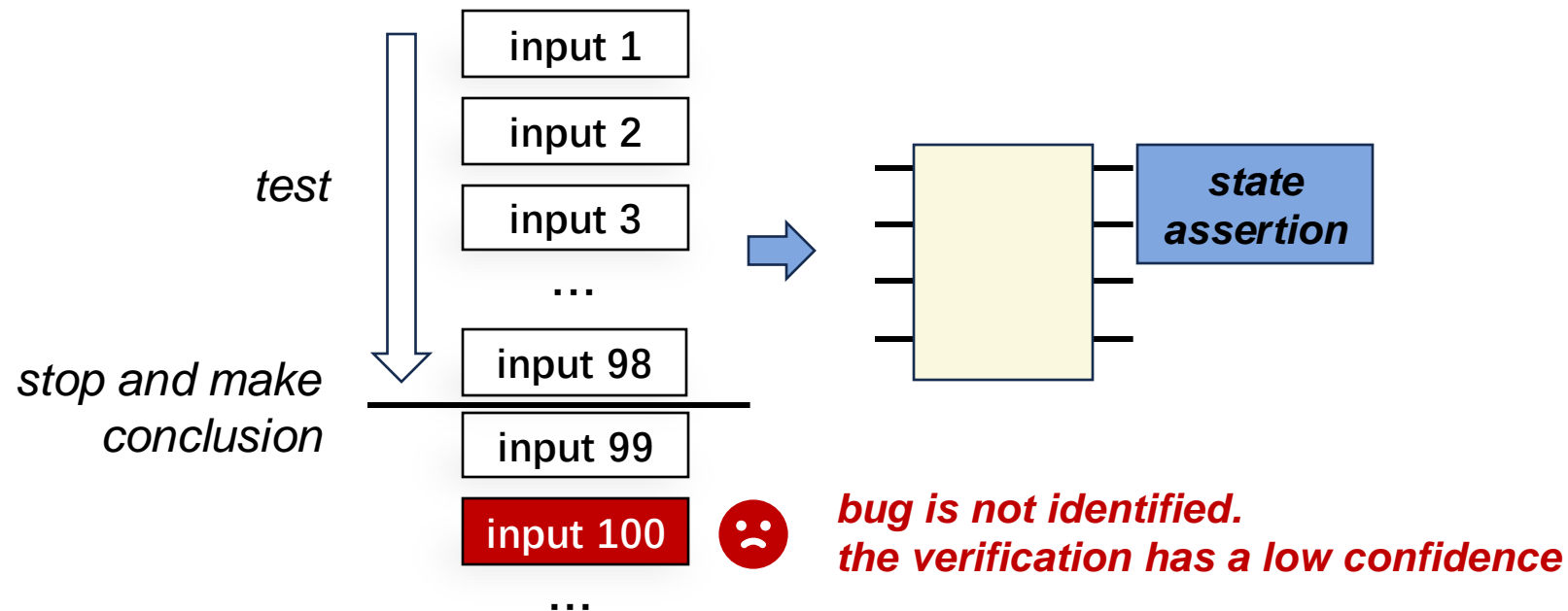
1. Check the relationship between the states in each stage
2. There may be mid-measurement or feedback

$\text{assert } a \geq 0$

$b = \sqrt{a}$

Quantum assertion is similar to the classical assertion, while the verified qubits usually stay in the superposition state.

**Confidence of the verification:** The probability that the correctness holds for all inputs.

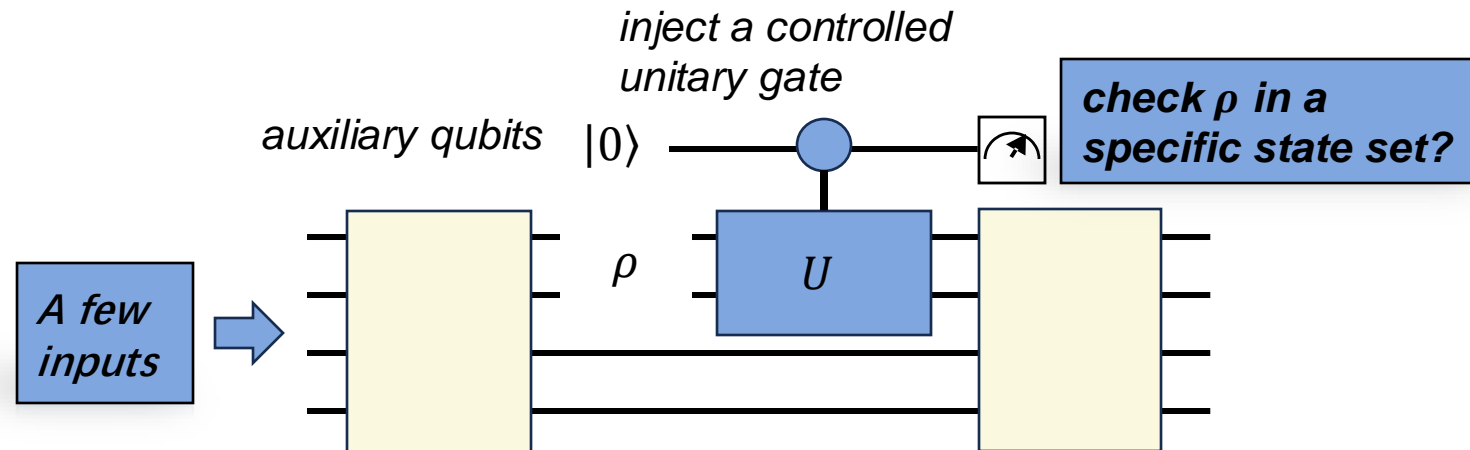


The ability to generalize the test inputs to the whole space is necessary to ensure high confidence

	Statement	Analysis method	Input coverage	Theorem
<b>A good assertion validation method</b>	Specify the relation between states	Efficiently characterize the relation	The whole input space	1. Complexity is up-bounded 2. Theorem guarantee to high confidence

## An example

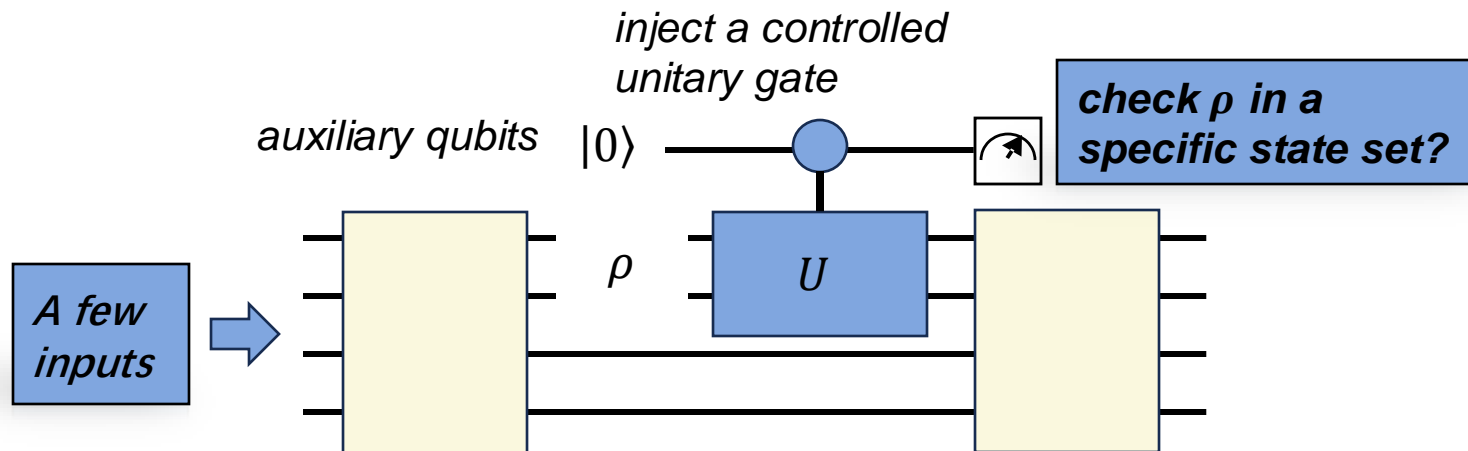
Ji Liu, et al. Quantum circuits for dynamic runtime assertions in quantum computation. ASPLOS, 2020



	Statement	Analysis method	Input coverage	Theorem
<b>A good assertion validation method</b>	Specify the relation between states	Efficiently characterize the relation	The whole input space	1. Complexity is up-bounded 2. Theorem guarantee to high confidence

## An example

Ji Liu, et al. Quantum circuits for dynamic runtime assertions in quantum computation. ASPLOS, 2020



### Expressiveness?

Only support state in state equal operation.

### Efficient?

Numerous gates to synthesize unitary gates.

### High confidence?

Cover a few inputs.  
Lack confidence guarantee.

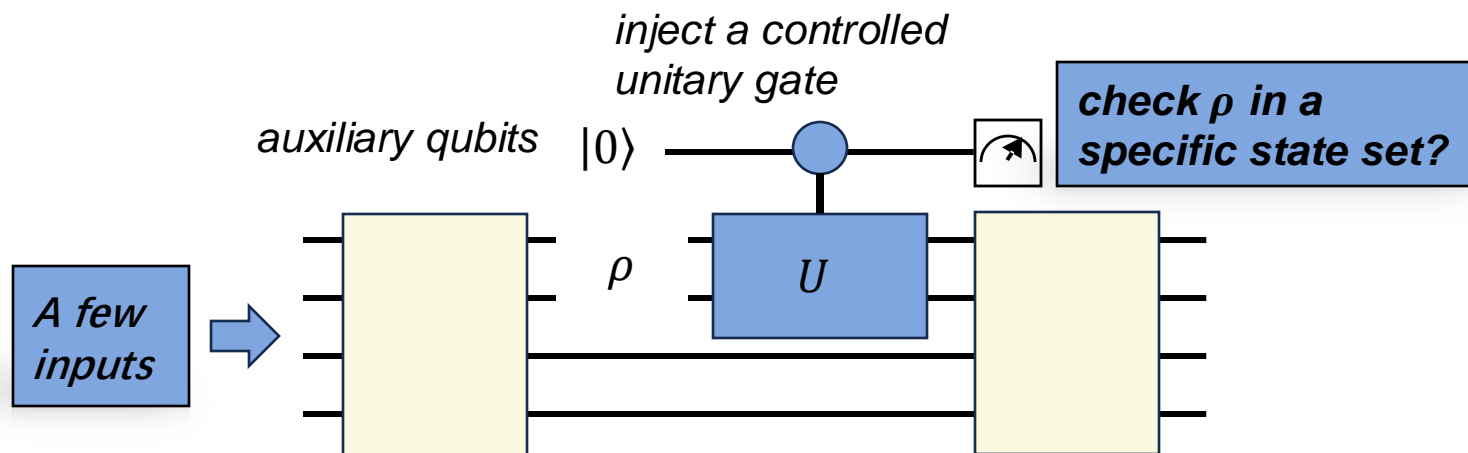


	Statement	Analysis method	Input coverage	Theorem
<b>A good assertion validation method</b>	Specify the relation between states	Efficiently characterize the relation	The whole input space	1. Complexity is up-bounded 2. Theorem guarantee to high confidence

## An example

Ji Liu, et al. Quantum circuits for dynamic runtime assertion

*Similar problems in current works, including Liu, et al. OOPSLA 2020, Huang, et al. ISCA, 2019*



## Expressiveness?

Only support state in state equal operation.

## Efficient?

Numerous gates to synthesize unitary gates.

## High confidence?

Cover a few inputs.  
Lack confidence guarantee.

Quantum collapse leads to loss of information.

~~~~~ ~~~~~ ...

N-qubit state

=

$$\begin{bmatrix} 0.2 & 0.3i & \dots & 0.0 & 0.0 \\ 0.1 & 0.3 & & 0.3 & 0.0 \\ \vdots & & \ddots & \vdots & \\ 0.4 & 0.0 & & 0.1 & 0.5 \\ 0.1 & 0.2 & \dots & 0.0 & 0.0 \end{bmatrix}$$

$2^N \times 2^N$ -size density matrix

*Only diagonal elements are obtained by measurements*

Lack an efficient method to characterize the mapping from input to output

*Necessary to ensure high confidence*

$2^L$ -dimension space that cannot be exhaustively traversed

L qubits

input

circuit

output

M qubits

Process tomography requires  $\mathcal{O}(4^N)$  complexity

N qubits

State tomography requires  $\mathcal{O}(2^N)$  complexity

*Alternative approach but has high complexity ( $N > L, N > M$ )*

- Background and Challenges
- **Overview of MorphQPV**
- Assertion Statement and Validation
- Experiment
- API of MorphQPV

# MorphQPV Overview



write a quantum program

```
input q[0,1];
x q[2,3,4];
cz q[1],q[4];
t1 q[1,2];
x q[2,3,4];
h q[1];
t2 q[0];
```

label states by a  
tracepoint pragma

**Object:** enable input-  
independent assertion

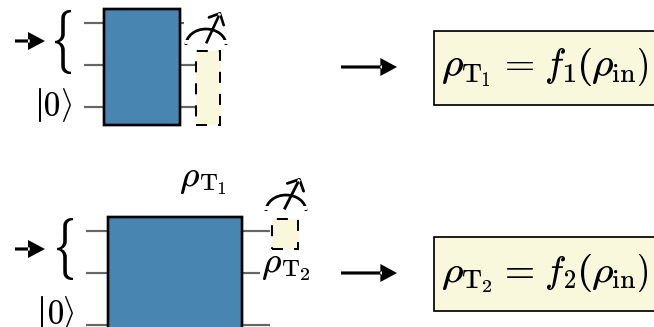
**Object:** minimize the number of  
program execution

**Object:** apply global search to  
counter example

Step 1. assertion statement

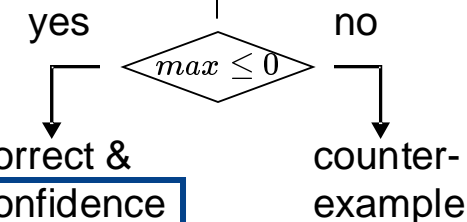
```
assume
  P1(ρT1)
  P2(ρT2)
guarantee
  P3(ρT1, ρT2)
```

Step 2. program characterization



Step 3. assertion validation

```
maximize P3(f1(ρin), f2(ρin)),
subject to P1(f1(ρin)) ≤ 0,
          P2(f2(ρin)) ≤ 0,
```



specify the **expected relation**  
between the tracepoint states

characterize the **natural relation** by  
building approximation functions

compare

**Object:** guide the allocation  
of computation resource

# MorphQPV Overview



*write a quantum program*

```
input q[0,1];
```

```
x q[2,3,4];
```

```
cz q[1],q[4];
```

```
t1 q[1,2];
```

```
x q[2,3,4];
```

```
h q[1];
```

```
t2 q[0];
```

*label states by a  
tracepoint pragma*

# MorphQPV Overview



write a quantum program

```
input q[0,1];  
x q[2,3,4];  
cz q[1],q[4];  
t1 q[1,2];  
x q[2,3,4];  
h q[1];  
t2 q[0];
```

label states by a  
tracepoint pragma

**Object:** enable input-  
independent assertion

Step 1. assertion statement

```
assume  
   $P_1(\rho_{T_1})$   
   $P_2(\rho_{T_2})$   
guarantee  
   $P_3(\rho_{T_1}, \rho_{T_2})$ 
```

specify the **expected relation**  
between the tracepoint states

# MorphQPV Overview



write a quantum program

```
input q[0,1];  
x q[2,3,4];  
cz q[1],q[4];  
t1 q[1,2];  
x q[2,3,4];  
h q[1];  
t2 q[0];
```

label states by a  
tracepoint pragma

**Object:** enable input-  
independent assertion

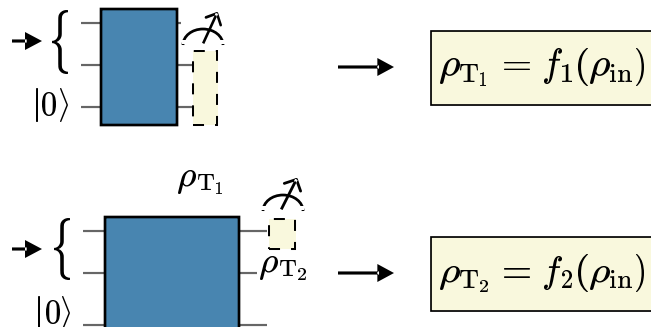
**Object:** minimize the number of  
program execution

Step 1. assertion statement

```
assume  
   $P_1(\rho_{T_1})$   
   $P_2(\rho_{T_2})$   
guarantee  
   $P_3(\rho_{T_1}, \rho_{T_2})$ 
```

specify the **expected relation**  
between the tracepoint states

Step 2. program characterization



characterize the **natural relation** by  
building approximation functions

# MorphQPV Overview



write a quantum program

```
input q[0,1];
x q[2,3,4];
cz q[1],q[4];
t1 q[1,2];
x q[2,3,4];
h q[1];
t2 q[0];
```

label states by a  
tracepoint pragma

**Object:** enable input-  
independent assertion

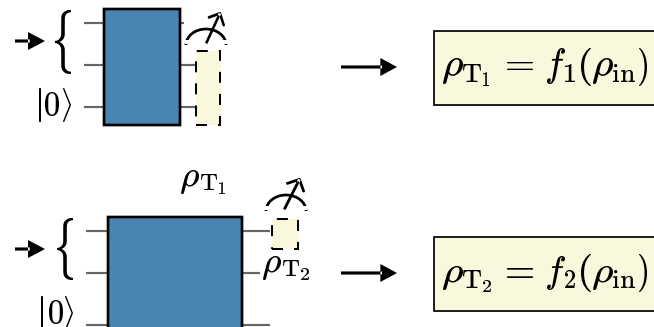
**Object:** minimize the number of  
program execution

**Object:** apply global search to  
counter example

Step 1. assertion statement

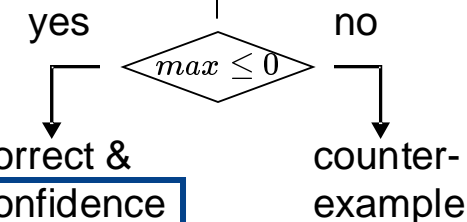
```
assume
  P1(ρT1)
  P2(ρT2)
guarantee
  P3(ρT1, ρT2)
```

Step 2. program characterization



Step 3. assertion validation

```
maximize P3(f1(ρin), f2(ρin)),
subject to P1(f1(ρin)) ≤ 0,
          P2(f2(ρin)) ≤ 0,
```



specify the **expected relation**  
between the tracepoint states

characterize the **natural relation** by  
building approximation functions

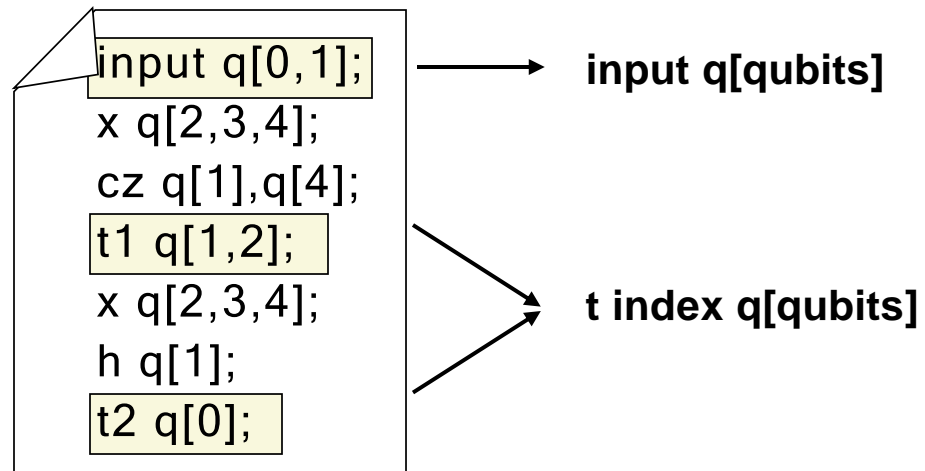
compare

**Object:** guide the allocation  
of computation resource



- Background and Challenges
- Overview of MorphQPV
- **Assertion Statement and Validation**
- Experiment
- API of MorphQPV

## 1. Label the asserted state by tracepoint pragma



## 2. Use assume-guarantee assertion to specify the relation between states

**assume:**

$$P_1(\rho_{T1}), P_2(\rho_{T2})$$

**guarantee:**

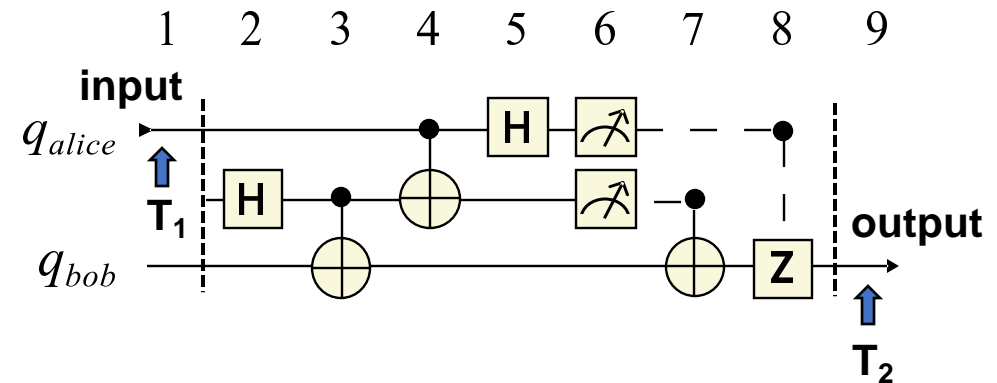
$$P_3(\rho_{T1}, \rho_{T2})$$

If  $P_1 \leq 0$  and  $P_2 \leq 0$   
then  $P_3 \leq 0$   
 $\Rightarrow$  assertion is correct

or

If  $P_1 \leq 0$  and  $P_2 \leq 0$   
then  $P_3 > 0$   
 $\Rightarrow$  assertion fails

## An example: Quantum teleportation



*input state should equal output state*

**assume:**

$$P_1(\rho_{T1}) = \|\rho_{T1}\rho_{T1}^\dagger - \rho_{T1}\|,$$

$$P_2(\rho_{T2}) = \|\rho_{T2}\rho_{T2}^\dagger - \rho_{T2}\|,$$

**guarantee:**

$$P_3(\rho_{T1}, \rho_{T2}) = \|\rho_{T1} - \rho_{T2}\|$$

## Isomorphism

A structure-preserving mapping  $\mathbb{R}_x \rightarrow \mathbb{R}_y$  between two spaces of the same type that can be retraced by an inverse mapping.

### Example of isomorphism

$$\begin{array}{c} x + 1 = y \\ \updownarrow \text{inverse} \\ y - 1 = x \end{array}$$

### Quantum evolution is isomorphism


$$\begin{array}{c} U\rho U^{-1} = \rho' \\ \updownarrow \text{inverse} \\ U^{-1}\rho'U = \rho \end{array}$$

### Feature of isomorphism

additivity:  $f(u + v) = f(u) + f(v)$

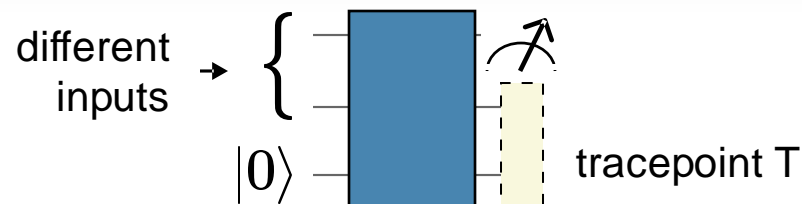
homogeneity:  $f(c u) = c f(u)$

$$f(\sum_i c_i u_i) = \sum_i c_i f(u_i)$$

 also has the feature

*inspire us to generalize the information obtained from individual input into a broader input space*

## Step 1: sample inputs



Inputs are orthogonal and prepared by the Clifford group.

Based on “Sergey Bravyi and Dmitri Maslov. Hadamard-free circuits expose the structure of the clifford group. IEEE Transactions on Information Theory, 2021”.

Record inputs and state at tracepoint as  $\langle \sigma_{\text{input},i}, \sigma_{T,i} \rangle$  pairs.

| Input state                               | Tracepoint state               | Obtained by tomography |
|-------------------------------------------|--------------------------------|------------------------|
| $\sigma_{\text{input},1}$                 | $\sigma_{T,1}$                 |                        |
| $\sigma_{\text{input},2}$                 | $\sigma_{T,2}$                 |                        |
| ...                                       |                                |                        |
| $\sigma_{\text{input},N_{\text{sample}}}$ | $\sigma_{T,N_{\text{sample}}}$ |                        |

## Step 2: construct approximation function

$$f(\rho_{\text{input}}) = \rho_T$$

The function is computed in two steps:

1. For input  $\rho_{\text{input}}$ , it first approximates the  $\rho_{\text{input}}$  to the linear combination of sampled inputs  $\sigma_{\text{input},i}$

$$\rho_{\text{input}} = \sum_i \alpha_i \sigma_{\text{input},i}$$

$\{\alpha_i\}$  is real parameters.

2. It then outputs tracepoint state:

$$\rho_T = \sum_i \alpha_i \sigma_{T,i}$$

Based on the additivity and homogeneity of isomorphism

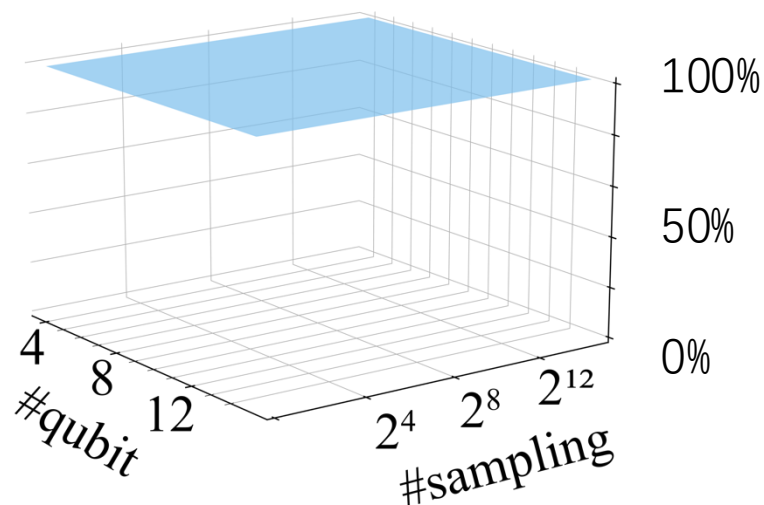
$$f(\sum_i c_i u_i) = \sum_i c_i f(u_i)$$

## Theorem 1 (Approximation Accuracy)

- Case1: For inputs that can be accurately represented by linear combination of sampled inputs, the accuracy is 100%.
- Case2: For inputs with eigenstates that cannot be represented the linear combination of sampled inputs, the average accuracy is  $\frac{N_{\text{sample}}}{2^{N_{\text{input}}}} \times 100\%$

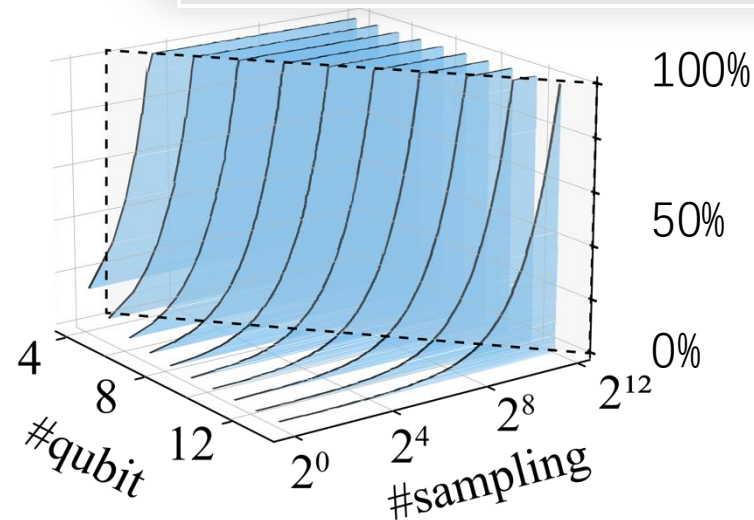
**Example:** Approximation accuracy in the quantum teleportation programs with different number of qubits and sampled inputs.

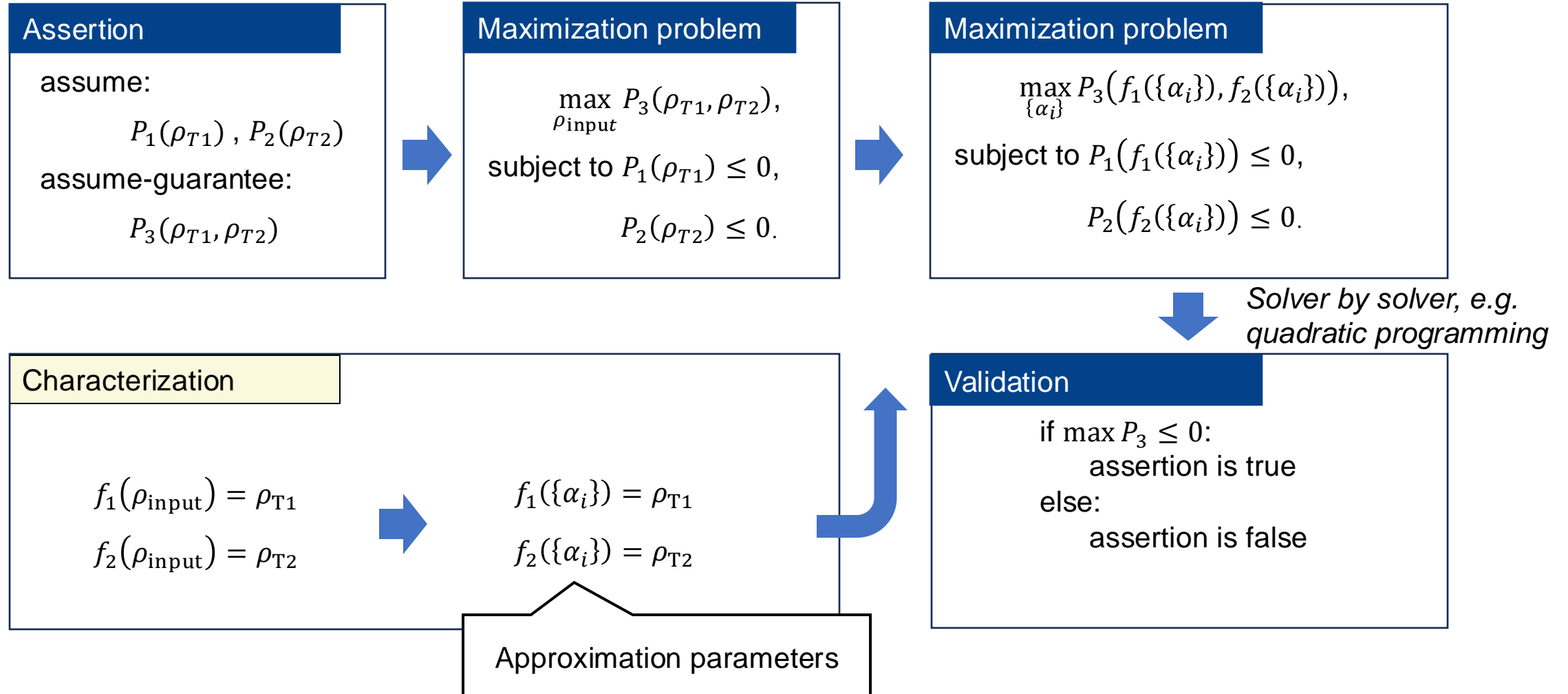
Case 1



Case 2

The approximation result is the same as the tomograph result, when the accuracy is 100%, .





## Assertion

assume:

$$P_1(\rho_{T1}), P_2(\rho_{T2})$$

assume-guarantee:

$$P_3(\rho_{T1}, \rho_{T2})$$

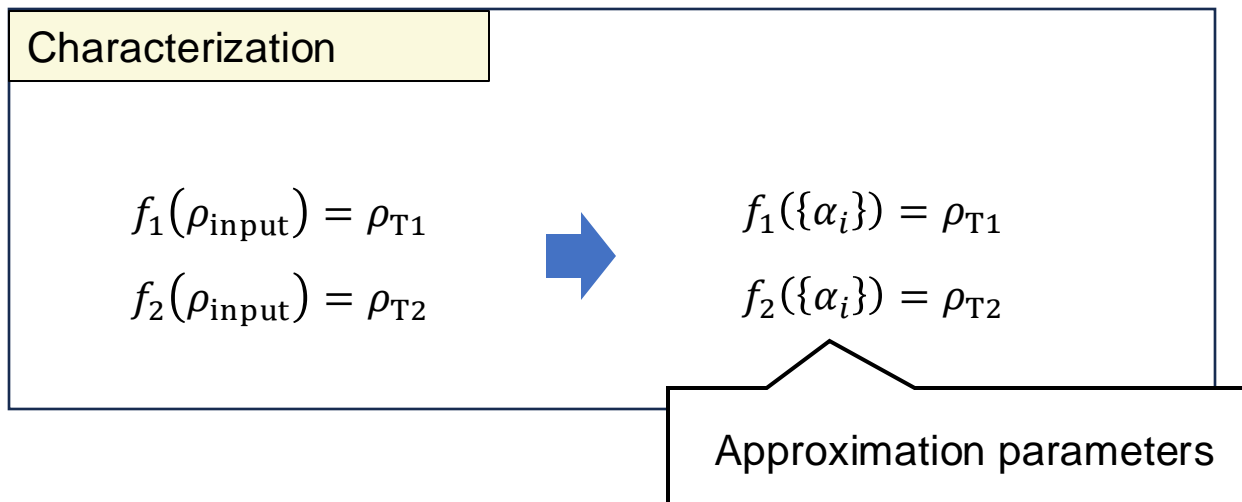
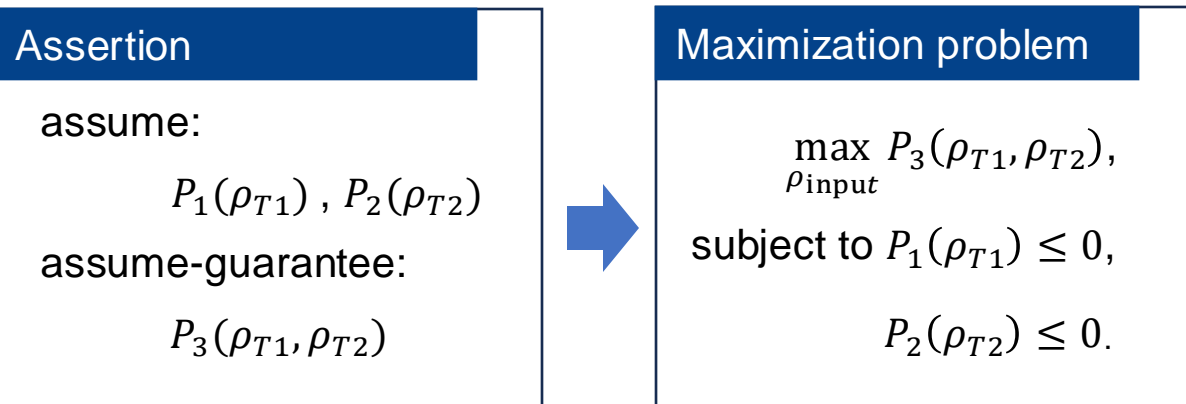


## Maximization problem

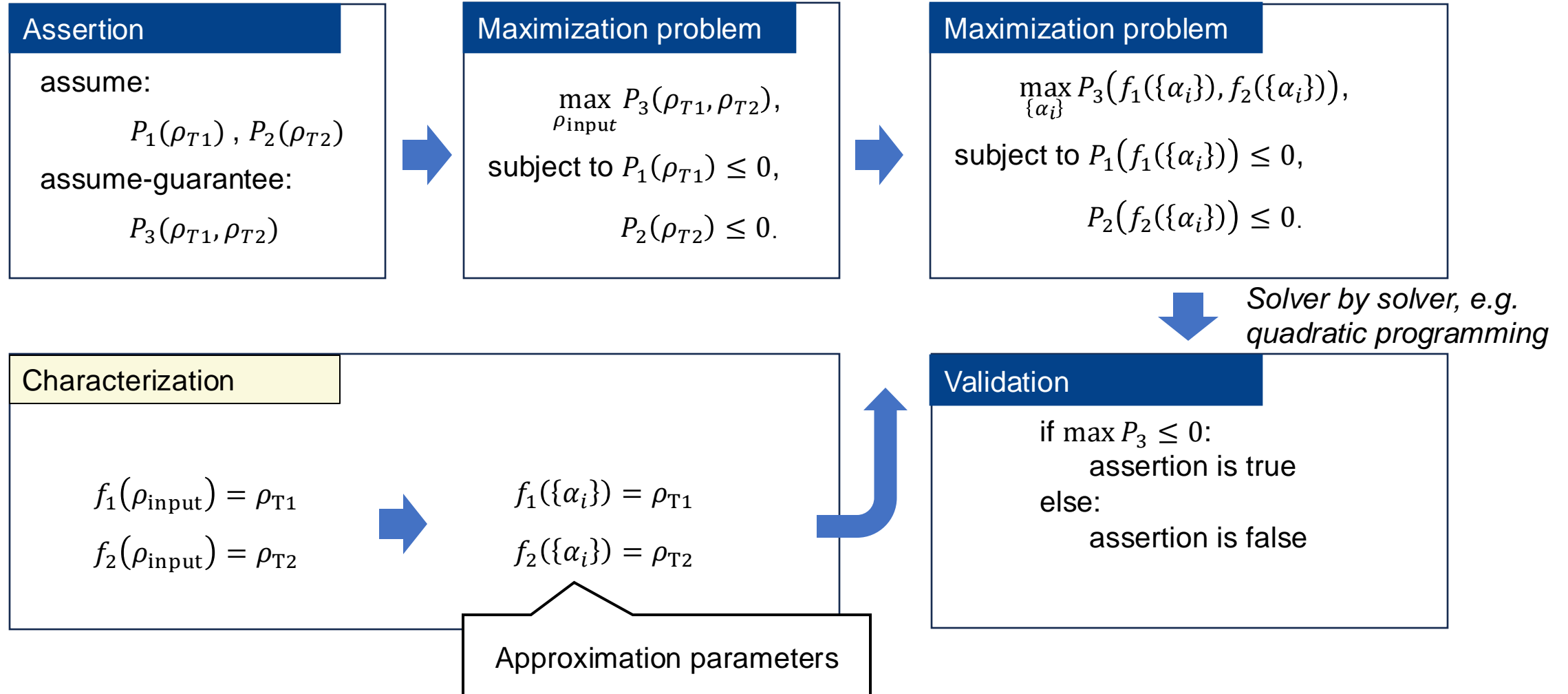
$$\max_{\rho_{\text{input}}} P_3(\rho_{T1}, \rho_{T2}),$$

$$\text{subject to } P_1(\rho_{T1}) \leq 0,$$

$$P_2(\rho_{T2}) \leq 0.$$



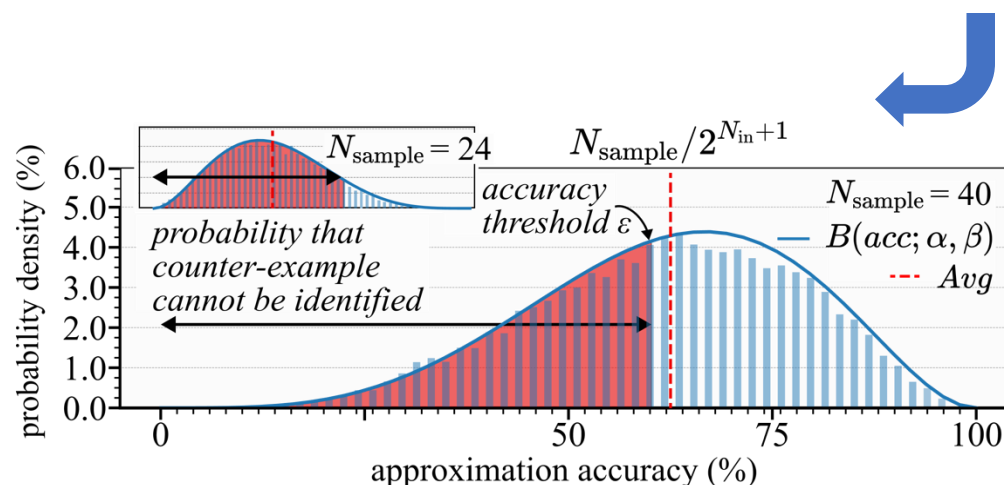




confidence = P(the correctness holds for all inputs) = 1 – P(counter-example exists but is not identified )

P(counter-example exists but is not identified) = P(accuracy of counter – example  $< \epsilon$ )

accuracy threshold to discriminate error



$$P(\text{accuracy} < \epsilon) = \int_0^{\epsilon} B(x; \beta_1, \beta_2)$$

Accuracies follow Beta distribution  $B(\beta_1, \beta_2)$   
 $\beta_1, \beta_2$  can be obtained by fitting some test inputs

## Theorem 2 (Confidence)

When the program only has one counter-example

lower-bound

$$\text{confidence} = 1 - P(\text{accuracy} < \epsilon)$$

When the program only has  $N_{c-e}$  counter-examples

$$\text{confidence} = 1 - P(\text{accuracy} < \epsilon)^{N_{c-e}}$$

Accuracy and confidence linearly increase as the number of sampled inputs grows

# Outline of Presentation

---

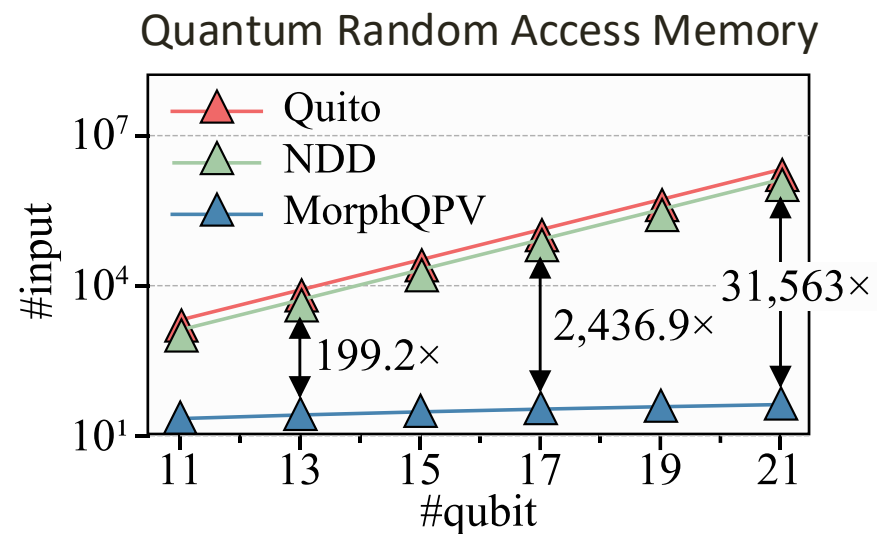
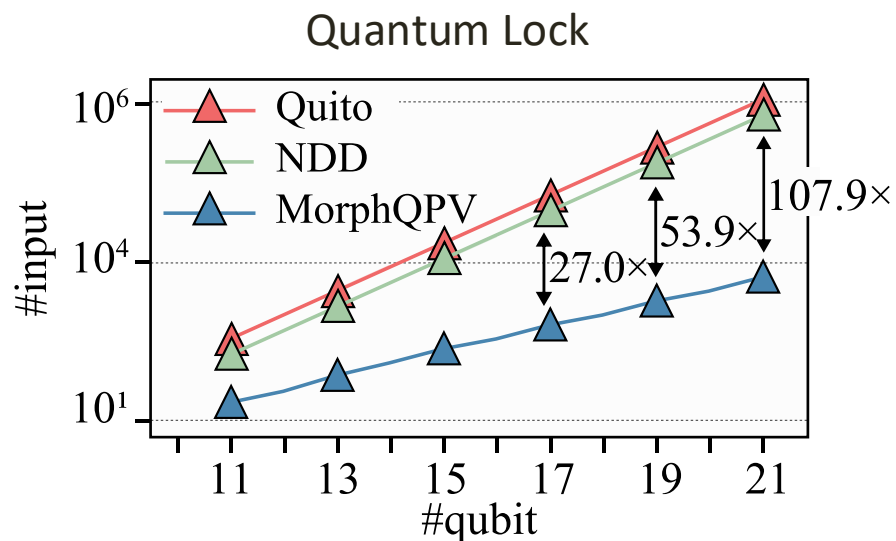


- Background and Challenges
- Overview of MorphQPV
- Assertion Statement and Validation
- **Experiment**
- API of MorphQPV

# Comparison to Prior Works



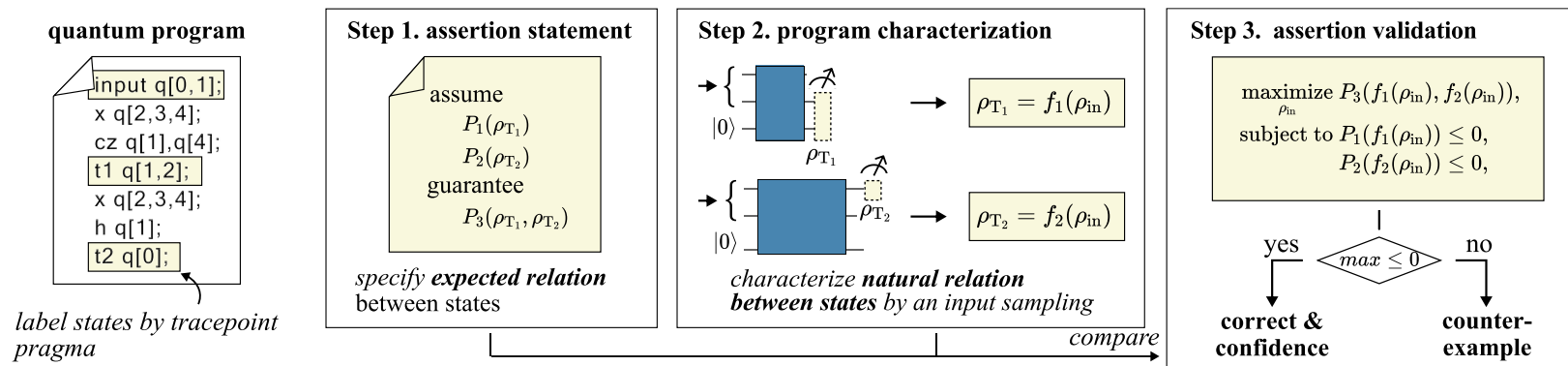
|                  | Huang, et al. ISCA'19    | Li, et al. OOPSLA'20 | Liu, et al. ASPLOS'20 | Feng, et al. ASPLOS'23 | <b>MorphQPV</b>         |
|------------------|--------------------------|----------------------|-----------------------|------------------------|-------------------------|
| Verified Object  | Probability distribution | Mixed state          | Mixed state           | Mixed state            | Mixed state & Evolution |
| Comparison       | Part                     | Equal & In           | Equal & In            | Equal & In             | Full                    |
| Interpretability | Part                     | No                   | No                    | No                     | Full                    |
| Feedback         | No                       | No                   | No                    | No                     | Full                    |



The number of test inputs in debugging the programs with different number of qubits

1. Limitations of prior assertion works: low confidence, low expressiveness, and high overhead
2. Three-step verification of MorphQPV: statement, characterization, and validation
3. Two theorems: upper-bound complexity of verification and lower-bound of confidence
4. Contents that are not mentioned in the presentation:
  - Proof of theorems.
  - Further optimization to minimize the overhead.
  - detailed comparison with prior works.

Please refer to the paper.



# Outline of Presentation

---



- Background and Challenges
- Overview of MorphQPV
- Assertion Statement and Validation
- Experiment
- **API of MorphQPV**

File:

- JanusQ/examples/ipynb/3\_1\_verify\_quantum\_program.ipynb
- [https://janusq.github.io/tutorials/demo/3\\_1\\_verify\\_quantum\\_program](https://janusq.github.io/tutorials/demo/3_1_verify_quantum_program)

configure solver

```
from janusq.verification.morphqpv import MorphQC, Config
from janusq.verification.morphqpv import IsPure, Equal
```

```
myconfig = Config()
myconfig.solver = 'sgd'
```

assertion statement and  
validation in quantum circuit

```
with MorphQC(config=myconfig) as morphQC:
    morphQC.add_tracepoint(0,1)
    morphQC.assume(0, IsPure())
    morphQC.assume(0, Equal(expectation(pauliX@pauliY), 0.4))
    morphQC.x([1,3])
    morphQC.y([0,1,2])
    for i in range(4):
        morphQC.cnot([i, i+1])
    morphQC.s([0,2,4])
    morphQC.add_tracepoint(2,4)
    ...
```



# Thanks for listening

## MorphQPV: Exploiting Isomorphism in Quantum Programs to Facilitate Confident Verification

Siwei Tan\*, Debin Xiang\*, Liqiang Lu†, Junlin Lu, Qiuping Jiang,  
Mingshuai Chen, and Jianwei Yin†