



MQT Qudits

A Framework For Mixed-Dimensional Qudit Quantum Computing

Version 0.1.1.dev3

Chair for Design Automation
Technical University of Munich, Germany
quantum.cda@xcit.tum.de

May 13, 2024

MQT Qudits is an open-source C++17 and Python framework for mixed-dimensional qudit quantum computing developed as part of the [Munich Quantum Toolkit \(MQT\)](#) by the [Chair for Design Automation](#) at the [Technical University of Munich](#).

Note: The tool is in an experimental stage, which is subject to frequent changes, and has limited documentation. We are working on improving that. In the meantime, users can explore how to use the framework via the [Tutorial](#), showcasing its main functionality.

Furthermore, [this video](#) briefly illustrates some of the functionalities of MQT Qudits.

We appreciate any feedback and contributions to the project. If you have any questions, feel free to create a [discussion](#) or an [issue](#) on GitHub.

I MQT Qudits Tutorial

Discover a New Dimension in Quantum Computing. Embark on a journey with MQT Qudits, a framework for Mixed-Dimensional Quantum Computing.

Delve into the realm of mixed-dimensional quantum computing with NeQST—a project funded by the European Union and developed as part of the [Munich Quantum Toolkit \(MQT\)](#) by the [Chair for Design Automation](#) at the [Technical University of Munich](#). Our team is focused on creating design automation methods and software for quantum computing. The following tutorial will guide you through the initial tools and contributions we have made to advance Quantum Information Processing for Science and Technology.

I-A Installation Steps:

```
(.venv) $ pip install mqt.qudits
```

For those seeking hands-on customization, simply clone the corresponding repository and perform a local installation.

```
$ git clone https://github.com/cda-tum/mqt-qudits.git
$ cd mqt-qudits
$ python3 -m venv .venv
$ source .venv/bin/activate
(.venv) $ pip install -ve .
+++

```{note}
This requires a C++17 compiler, a minimum CMake version of 3.19, and Python 3.8+.
```

## I-B User Inputs

```
1 import numpy as np
2
3 from mqt.qudits.quantum_circuit import QuantumCircuit
```

### New QASM Extension:

Dive into a language meticulously designed to express quantum algorithms and circuits. MQT Qudits extends the OpenQASM 2.0 grammar, effortlessly adapting to mixed-dimensional registers. In the following, a **DITQASM** program is explicitly written, although several methods for importing programs from files are present in the library.

```
1 qasm = """
2 DITQASM 2.0;
3
4 qreg field [7][5,5,5,5,5,5,5];
5 qreg matter [2];
6
7 creg meas_matter[7];
8 creg meas_fields[3];
9
10 h matter[0] ctl field[0] field[1] [0,0];
11 cx field[2], matter[0];
12 cx field[2], matter[1];
13 rxy (0, 1, pi, pi/2) field[3];
14
15 measure q[0] -> meas[0];
16 measure q[1] -> meas[1];
17 measure q[2] -> meas[2];
18 """
```

A new feature is the **control syntax**:

```
operation __ctl__ _quditline_ [list of qudit control levels]
```

We can import the DITQASM program and construct a quantum circuit.

```
1 circuit = QuantumCircuit()
2 circuit.from_qasm(qasm)
3
4 print(f"Number of operations: {len(circuit.instructions)}")
5 print(f"Number of qudits in the circuit: {circuit.num_qudits}")
6 print(f"Dimensions: {circuit.dimensions}")
```

```
Number of operations: 4
Number of qudits in the circuit: 9
Dimensions: [5, 5, 5, 5, 5, 5, 5, 2, 2]
```

## Python Interface

Constructing and manipulating quantum programs becomes a breeze with Python. You have the flexibility to:

1. **Initialize Quantum Circuits:** Start by creating your quantum circuits effortlessly.
2. **Create Quantum Registers:** Build dedicated quantum registers tailored to your needs.
3. **Compose Circuits:** Seamlessly bring together your quantum registers, forming a unified and powerful circuit.
4. **Apply Operations:** Easily apply a variety of qudit operations, without worrying about the right representation.

Let's construct a quantum circuit from scratch, with the python interface.

```
1 from mqt.qudits.quantum_circuit import QuantumRegister
2
3 circuit = QuantumCircuit()
4
5 field_reg = QuantumRegister("fields", 1, [7])
6 matter_reg = QuantumRegister("matter", 1, [2])
7
8 circuit.append(field_reg)
9 circuit.append(matter_reg)
10
11 print(f"Number of operations: {len(circuit.instructions)}")
12 print(f"Number of qudits in the circuit: {circuit.num_qudits}")
13 print(f"Gate set: {circuit.gate_set}")
```

```
Number of operations: 0
Number of qudits in the circuit: 2
csum
cu_one
cu_two
cu_multi
cx
gellmann
h
ls
ms
pm
r
rh
randu
rz
virtrz
s
x
z
Gate set: None
```

No operations were inserted yet, let's take a look at how operations can be applied!

The size of every line is detected automatically and the right operations are applied to the right qudits

```
1 circuit.h(field_reg[0])
2 circuit.csum([field_reg[0], matter_reg[0]])
3
4 print(f"Number of operations: {len(circuit.instructions)}")
5 print(f"Number of qudits in the circuit: {circuit.num_qudits}")
```

```
Number of operations: 2
Number of qudits in the circuit: 2
```

It is possible to export the code as well and share your program in a QASM file.

```
1 print(circuit.to_qasm())
```

```
DITQASM 2.0;
qreg fields [1][7];
qreg matter [1][2];
creg meas[2];
h fields[0];
csum fields[0], matter[0];
measure fields[0] -> meas[0];
measure matter[0] -> meas[1];
```

Let's save the circuit to a file

```
1 file = circuit.save_to_file("my_circuit", ".")
```

and load it back

```
1 circuit.load_from_file(file)
2
3 print("Program:", circuit.to_qasm(), sep="\n")
4 print("Dimensions: ", circuit.dimensions)
```

```
Program:
DITQASM 2.0;
qreg fields [1][7];
qreg matter [1][2];
creg meas[2];
h fields[0];
csum fields[0], matter[0];
measure fields[0] -> meas[0];
measure matter[0] -> meas[1];

Dimensions: [7, 2]
```

Custom gates can be added to the circuit as well.

```
1 n = 5
2 random_matrix = np.random.randn(n, n) + 1j * np.random.randn(n, n)
3
4 Q, R = np.linalg.qr(random_matrix)
5
6 unitary_matrix = Q
7 cu = circuit.cu_one(field_reg[0], unitary_matrix)
```

Gates follow the order:

- target qudit/s : list or single number
- parameters list with order lower level, upper level, control level, theta, phi
- control data

A simple qudit gate can be added as follows:

```
1 r = circuit.r(field_reg[0], [0, 1, np.pi / 5, np.pi / 7])
```

Operations can also be controlled by other qudits, as shown below:

```
1 from mqt.qudits.quantum_circuit.gate import ControlData
2
3 r_c1 = circuit.r(field_reg[0], [0, 1, np.pi / 5, np.pi / 7], ControlData([matter_reg[0]], [1]))
```

or as

```
1 r_c2 = circuit.r(field_reg[0], [0, 1, np.pi / 5, np.pi / 7]).control([matter_reg[0]], [1])
```

The representation of the matrix corresponding to a gate is dynamic:

- 0: no identities
- 1: identities in between long-range gates are introduced
- 2: full circuit unitary

```
1 print(f"Gate matrix for {r._name}:", r.to_matrix(0), sep="\n")
```

Gate matrix for R7:

```
[[0.95105652+0.j -0.13407745-0.27841469j 0. +0.j
 0. +0.j 0. +0.j 0. +0.j
 0. +0.j]
 [0.13407745-0.27841469j 0.95105652+0.j 0. +0.j
 0. +0.j 0. +0.j 0. +0.j
 0. +0.j]
 [0. +0.j 0. +0.j 1. +0.j
 0. +0.j 0. +0.j 0. +0.j
 0. +0.j]
 [0. +0.j 0. +0.j 0. +0.j
 1. +0.j 0. +0.j 0. +0.j
 0. +0.j]
 [0. +0.j 0. +0.j 0. +0.j
 0. +0.j 1. +0.j 0. +0.j
 0. +0.j]
 [0. +0.j 0. +0.j 0. +0.j
 0. +0.j 0. +0.j 1. +0.j
 0. +0.j]
 [0. +0.j 0. +0.j 0. +0.j
 0. +0.j 0. +0.j 0. +0.j
 1. +0.j]]
```

The inverse of any gate can easily be obtained.

```
1 rd = r.dag()
2 print(f"Inverse gate matrix for {r._name}:", rd.to_matrix(0), sep="\n")
```

Inverse gate matrix for R7\_dag:

```
[[0.95105652-0.j 0.13407745+0.27841469j 0. -0.j
 0. -0.j 0. -0.j 0. -0.j
 0. -0.j]
 [-0.13407745+0.27841469j 0.95105652-0.j 0. -0.j
 0. -0.j 0. -0.j 0. -0.j
 0. -0.j]
 [0. -0.j 0. -0.j 1. -0.j
 0. -0.j 0. -0.j 0. -0.j
 0. -0.j]
 [0. -0.j 0. -0.j 0. -0.j
 1. -0.j 0. -0.j 0. -0.j
 0. -0.j]
 [0. -0.j 0. -0.j 0. -0.j
 0. -0.j 1. -0.j 0. -0.j
 0. -0.j]
 [0. -0.j 0. -0.j 0. -0.j
 0. -0.j 0. -0.j 1. -0.j
 0. -0.j]
 [0. -0.j 0. -0.j 0. -0.j
 0. -0.j 0. -0.j 0. -0.j
 1. -0.j]]
```

The control information can be accessed as well.

```
1 r_c1.control_info
```

```
{'target': 0,
'dimensions_slice': 7,
'params': [0, 1, 0.6283185307179586, 0.4487989505128276],
'controls': ControlData(indices=[1], ctrl_states=[1])}
```

Two- and multi-qudit gates follow the rule:

- two : target\_qudits first is control, second is target
- multi: all are controls, except last one is target

```
r_cl.reference_lines
```

```
[1, 0]
```

## I-C Simulation

After crafting your quantum circuit with precision, take it for a spin using two distinct engines, each flaunting its unique set of data structures.

- **External Tensor-Network Simulator:** Delve into the quantum realm with a robust external tensor-network simulator. Can simulate all the gate-set.
- **MiSiM (C++-Powered):** Unleash the power of decision-diagram-based simulation with MiSiM, seamlessly interfaced with Python for a fluid and efficient experience. Can only simulate the machine following machine gate set:
  - csum
  - cx
  - h
  - rxy
  - rz
  - rh
  - virtz
  - s
  - x
  - z

## Basic Simulation

```
1 circuit = QuantumCircuit()
2
3 field_reg = QuantumRegister("fields", 1, [3])
4 matter_reg = QuantumRegister("matter", 1, [3])
5
6 circuit.append(field_reg)
7 circuit.append(matter_reg)
8
9 h = circuit.h(field_reg[0])
10 csum = circuit.csum([field_reg[0], matter_reg[0]])
11
12 print(f"Number of operations: {len(circuit.instructions)}")
13 print(f"Number of qudits in the circuit: {circuit.num_qudits}")
```

```
Number of operations: 2
Number of qudits in the circuit: 2
```

```
1 from mqt.qudits.simulation import MQTQuditProvider
2
3 provider = MQTQuditProvider()
4 provider.backends("sim")
```

```
['tnsim', 'misim']
```

```
1 from mqt.qudits.visualisation import plot_counts, plot_state
2
3 backend = provider.get_backend("tnsim")
4
5 job = backend.run(circuit)
6 result = job.result()
7
8 state_vector = result.get_state_vector()
9
10 plot_state(state_vector, circuit)
```

```
1 backend = provider.get_backend("misim")
2
3 job = backend.run(circuit)
4 result = job.result()
5
6 state_vector = result.get_state_vector()
7
8 plot_state(state_vector, circuit)
```

## Extending Engines with Noise Model and Properties for FakeBackend

Enhance your quantum simulation experience by extending the engines with a noise model and incorporating various properties. By combining a noise model and carefully tuned properties, you can craft a FakeBackend that closely emulates the performance of the best quantum machines in experimental laboratories. This allows for more realistic and insightful quantum simulations.

Experiment, iterate, and simulate quantum circuits with the sophistication of real-world conditions, all within the controlled environment of your simulation.

```
1 from mqt.qudits.simulation.noise_tools.noise import Noise, NoiseModel
2
3 # Depolarizing quantum errors
4 local_error = Noise(probability_depolarizing=0.001, probability_dephasing=0.001)
5 local_error_rz = Noise(probability_depolarizing=0.03, probability_dephasing=0.03)
6
7 entangling_error = Noise(probability_depolarizing=0.1, probability_dephasing=0.001)
8 entangling_error_extra = Noise(probability_depolarizing=0.1, probability_dephasing=0.1)
9
10 entangling_error_on_target = Noise(probability_depolarizing=0.1, probability_dephasing=0.0)
11 entangling_error_on_control = Noise(probability_depolarizing=0.01, probability_dephasing=0.0)
12
13 # Add errors to noise_tools model
14
15 noise_model = NoiseModel() # We know that the architecture is only two qudits
16 # Very noisy gate
17 noise_model.add_all_qudit_quantum_error(local_error, ["csum"])
18 noise_model.add_recurrent_quantum_error_locally(local_error, ["csum"], [0])
19 # Entangling gates
20 noise_model.add_nonlocal_quantum_error(entangling_error, ["cx", "ls", "ms"])
21 noise_model.add_nonlocal_quantum_error_on_target(entangling_error_on_target, ["cx", "ls", ↵
↵ "ms"])
22 noise_model.add_nonlocal_quantum_error_on_control(entangling_error_on_control, ["csum", "cx", ↵
↵ "ls", "ms"])
23 # Super noisy Entangling gates
24 noise_model.add_nonlocal_quantum_error(entangling_error_extra, ["csum"])
25 # Local Gates
26 noise_model.add_quantum_error_locally(local_error, ["h", "rxy", "s", "x", "z"])
27 noise_model.add_quantum_error_locally(local_error_rz, ["rz", "virt rz"])
28
29 print(noise_model.quantum_errors)
```

```
{'csum': {'all': Noise(probability_depolarizing=0.001, probability_dephasing=0.001), (0,):
↳ Noise(probability_depolarizing=0.001, probability_dephasing=0.001), 'control':
↳ Noise(probability_depolarizing=0.01, probability_dephasing=0.0), 'nonlocal':
↳ Noise(probability_depolarizing=0.1, probability_dephasing=0.1)}, 'cx': {'nonlocal':
↳ Noise(probability_depolarizing=0.1, probability_dephasing=0.001), 'target':
↳ Noise(probability_depolarizing=0.1, probability_dephasing=0.0), 'control': Noise(probability_
↳ depolarizing=0.01, probability_dephasing=0.0)}, 'ls': {'nonlocal': Noise(probability_
↳ depolarizing=0.1, probability_dephasing=0.001), 'target': Noise(probability_depolarizing=0.1,
↳ probability_dephasing=0.0), 'control': Noise(probability_depolarizing=0.01, probability_
↳ dephasing=0.0)}, 'ms': {'nonlocal': Noise(probability_depolarizing=0.1, probability_
↳ dephasing=0.001), 'target': Noise(probability_depolarizing=0.1, probability_dephasing=0.0),
↳ 'control': Noise(probability_depolarizing=0.01, probability_dephasing=0.0)}, 'h': {'local':
↳ Noise(probability_depolarizing=0.001, probability_dephasing=0.001)}, 'rxy': {'local':
↳ Noise(probability_depolarizing=0.001, probability_dephasing=0.001)}, 's': {'local':
↳ Noise(probability_depolarizing=0.001, probability_dephasing=0.001)}, 'x': {'local':
↳ Noise(probability_depolarizing=0.001, probability_dephasing=0.001)}, 'z': {'local':
↳ Noise(probability_depolarizing=0.001, probability_dephasing=0.001)}, 'rz': {'local':
↳ Noise(probability_depolarizing=0.03, probability_dephasing=0.03)}, 'virtz': {'local':
↳ Noise(probability_depolarizing=0.03, probability_dephasing=0.03)}}
```

We can set the noise model for the simulation, but also set several other flags:

- `shots`: number of shots for the stochastic simulation
- `memory`: flag for saving shots (True/False)
- `full_state_memory`: save the full noisy states
- `file_path`: file path of the h5 database storing the data
- `file_name`: name of the file

```
1 backend = provider.get_backend("tnsim")
2
3 job = backend.run(circuit, noise_model=noise_model)
4
5 result = job.result()
6 counts = result.get_counts()
7
8 plot_counts(counts, circuit)
```

```
{'00': 332,
 '01': 0,
 '02': 3,
 '10': 4,
 '11': 329,
 '12': 0,
 '20': 0,
 '21': 7,
 '22': 325}
```

You can also invoke a fake backend and retrieve a few relevant properties, that are already embedded in them

```
1 provider = MQTQuditProvider()
2 provider.backends("fake")
```

```
['faketrap2trits', 'faketrap2six']
```

```
1 backend_ion = provider.get_backend("faketrap2trits", shots=1000)
```



```

1 import matplotlib.pyplot as plt
2 import networkx as nx
3
4 mapping = backend_ion.energy_level_graphs
5
6 pos = nx.circular_layout(mapping[0])
7 nx.draw(mapping[0], pos, with_labels=True, node_size=2000, node_color="lightblue", font_
8 size=12, font_weight="bold")
9 plt.show()

```

```

1 job = backend_ion.run(circuit)
2 result = job.result()
3 counts = result.get_counts()
4
5 plot_counts(counts, circuit)

```

```

{'00': 328,
 '01': 0,
 '02': 1,
 '10': 2,
 '11': 322,
 '12': 0,
 '20': 0,
 '21': 2,
 '22': 345}

```

## I-D Compilation

Tailor your quantum compilation process to achieve optimal performance and emulate the intricacies of experimental setups.

### Compiler Customization with Modern Passes

1. **Optimization Strategies:** Implement specific optimization strategies based on your quantum algorithm's characteristics. Fine-tune compilation for better resource utilization and reduced gate counts.
2. **Gate Decomposition:** Customize gate decomposition techniques to match the capabilities of experimental quantum hardware. Aligning with the native gate set enhances the efficiency of your compiled circuits.

### Experimental-Inspired Compilation

Emulate the features of the best experimental laboratories in your compilation process. Leverage modern compiler passes to customize optimization, gate decomposition, and noise-aware strategies, creating compiled circuits that closely resemble the challenges and advantages of cutting-edge quantum hardware.

Customize, compile, and push the boundaries of quantum algorithms with a tailored approach to quantum compilation.

```

1 from mqt.qudits.compiler import QuditCompiler

```

```

1 qudit_compiler = QuditCompiler()
2
3 passes = ["PhyLocQRPass"]

```

```

1 compiled_circuit_qr = qudit_compiler.compile(backend_ion, circuit, passes)
2
3 print(f"Number of operations: {len(compiled_circuit_qr.instructions)}")
4 print(f"Number of qudits in the circuit: {compiled_circuit_qr.num_qudits}")

```

```

Number of operations: 10
Number of qudits in the circuit: 2

```

```

1 job = backend_ion.run(compiled_circuit_qr)
2
3 result = job.result()
4 counts = result.get_counts()
5
6 plot_counts(counts, compiled_circuit_qr)

```

```

{'00': 326,
 '01': 0,
 '02': 4,
 '10': 0,
 '11': 326,
 '12': 0,
 '20': 0,
 '21': 2,
 '22': 342}

```

```

1 passes = ["PhyLocAdaPass", "ZPropagationPass", "ZRemovalPass"]
2
3 compiled_circuit_ada = qudit_compiler.compile(backend_ion, circuit, passes)
4
5 print(f"Number of operations: {len(compiled_circuit_ada.instructions)}")
6 print(f"Number of qudits in the circuit: {compiled_circuit_ada.num_qudits}")

```

```

Number of operations: 5
Number of qudits in the circuit: 2

```

```

1 job = backend_ion.run(compiled_circuit_ada)
2
3 result = job.result()
4 counts = result.get_counts()
5
6 plot_counts(counts, compiled_circuit_ada)

```

```

{'00': 321,
 '01': 0,
 '02': 4,
 '10': 2,
 '11': 328,
 '12': 0,
 '20': 0,
 '21': 3,
 '22': 342}

```

```

1 from mqt.qudits.visualisation import draw_qudit_local
2
3 draw_qudit_local(compiled_circuit_ada)

```

```

|0>-----[R01(1.57,-2.09)]----[R02(1.23,-2.62)]----[R02(3.14,-2.62)]----[R01(1.57,-0.52)]----MG-
↔-----=||
|0>-----MG----MG----MG----MG----MG-----=||

```

## II Publications

MQT Qudits is academic software. Thus, many of its built-in algorithms have been published as scientific papers [1, 2, 3, 4, 5].

If you use *MQT Qudits* in your work, we would appreciate if you cited the respective paper(s).

## III mqt.qudits

MQT Qudits - A framework for mixed-dimensional qudit quantum computing.

### III-A Subpackages

**mqt.qudits.compiler**

#### Subpackages

**mqt.qudits.compiler.compilation\_minertools** Common utilities for compilation.

Submodules

**mqt.qudits.compiler.compilation\_minertools.local\_compilation\_minertools**

Module Contents

**swap\_elements**(*list\_nodes*, *i*, *j*)

**new\_mod**(*a*, *b*=2 \* *np.pi*)

**pi\_mod**(*a*)

**regulate\_theta**(*angle*)

**phi\_cost**(*theta*)

**theta\_cost**(*theta*)

**rotation\_cost\_calc**(*gate*, *placement*)

**mqt.qudits.compiler.compilation\_minertools.naive\_unitary\_verifier**

Module Contents

**class UnitaryVerifier**(*sequence*, *target*, *dimensions*, *nodes*=None, *initial\_map*=None, *final\_map*=None)

Verifies unitary matrices. *sequence* is a list of numpy arrays *target* is a numpy array *dimensions* is list of ints, equals to the dimensions of the qudits involved in the target operation *initial\_map* is a list representing the mapping of the logic states to the physical ones at the beginning of the computation *final\_map* is a list representing the mapping of the logic states to the physical ones at the end of the computation

**get\_perm\_matrix**(*nodes*, *mapping*)

**verify**()

**mqt.qudits.compiler.compilation\_minertools.numerical\_ansatz\_utils**

Module Contents

**on1**(*gate*, *other\_size*)

**on0**(*gate*, *other\_size*)

**gate\_expand\_to\_circuit**(*gate*, *circuits\_size*, *target*, *dims*=None)

**apply\_gate\_to\_tlines**(*gate\_matrix*, *circuits\_size*=2, *targets*=None, *dims*=None)

Package Contents

```

new_mod(a, b=2 * np.pi)

phi_cost(theta)

pi_mod(a)

regulate_theta(angle)

rotation_cost_calc(gate, placement)

swap_elements(list_nodes, i, j)

theta_cost(theta)

class UnitaryVerifier(sequence, target, dimensions, nodes=None, initial_map=None, final_map=None)
 Verifies unitary matrices. sequence is a list of numpy arrays target is a numpy array dimensions is list of
 ints, equals to the dimensions of the qudits involved in the target operation initial_map is a list representing
 the mapping of the logic states to the physical ones at the beginning of the computation final_map is a
 list representing the mapping of the logic states to the physical ones at the end of the computation

 get_perm_matrix(nodes, mapping)

 verify()

apply_gate_to_tlines(gate_matrix, circuits_size=2, targets=None, dims=None)

gate_expand_to_circuit(gate, circuits_size, target, dims=None)

on0(gate, other_size)

on1(gate, other_size)

mqt.qudits.compiler.onedit
Subpackages
mqt.qudits.compiler.onedit.local_operation_swap
Submodules
mqt.qudits.compiler.onedit.local_operation_swap.swap_routine
Module Contents
find_logic_from_phys(lev_a, lev_b, graph)

graph_rule_update(gate, graph) → None

graph_rule_ongate(gate, graph) → R

gate_chain_condition(previous_gates, current)

route_states2rotate_basic(gate, orig_placement)

cost_calculator(gate, placement, non_zeros)

Package Contents
cost_calculator(gate, placement, non_zeros)

gate_chain_condition(previous_gates, current)

graph_rule_ongate(gate, graph) → R

graph_rule_update(gate, graph) → None

```

**route\_states2rotate\_basic**(*gate, orig\_placement*)

`mqt.qudits.compiler.onedit.local_phases_transpilation`

Submodules

`mqt.qudits.compiler.onedit.local_phases_transpilation.propagate_virtrz`

Module Contents

**class** **ZPropagationPass**(*backend, back=True*)

Bases: `mqt.qudits.compiler.CompilerPass`

Helper class that provides a standard way to create an ABC using inheritance.

**transpile**(*circuit*)

**propagate\_z**(*circuit, line, back*)

**find\_intervals\_with\_same\_target\_qudits**(*instructions*)

**remove\_z**(*original\_circuit, back=True*)

`mqt.qudits.compiler.onedit.local_phases_transpilation.remove_phase_rotations`

Module Contents

**class** **ZRemovalPass**(*backend*)

Bases: `mqt.qudits.compiler.CompilerPass`

Helper class that provides a standard way to create an ABC using inheritance.

**transpile**(*circuit*)

**remove\_rz\_gates**(*original\_circuit, reverse=False*)

**remove\_initial\_rz**(*original\_circuit*)

**remove\_trailing\_rz\_sequence**(*original\_circuit*)

Package Contents

**class** **ZPropagationPass**(*backend, back=True*)

Bases: `mqt.qudits.compiler.CompilerPass`

Helper class that provides a standard way to create an ABC using inheritance.

**transpile**(*circuit*)

**propagate\_z**(*circuit, line, back*)

**find\_intervals\_with\_same\_target\_qudits**(*instructions*)

**remove\_z**(*original\_circuit, back=True*)

**class** **ZRemovalPass**(*backend*)

Bases: `mqt.qudits.compiler.CompilerPass`

Helper class that provides a standard way to create an ABC using inheritance.

**transpile**(*circuit*)

**remove\_rz\_gates**(*original\_circuit, reverse=False*)

**remove\_initial\_rz**(*original\_circuit*)

**remove\_trailing\_rz\_sequence**(*original\_circuit*)

`mqt.qudits.compiler.onedit.mapping_aware_transpilation`

Submodules

`mqt.qudits.compiler.onedit.mapping_aware_transpilation.phy_local_adaptive_decomp`

Module Contents

**class** `PhyLocAdaPass`(*backend*)

Bases: `mqt.qudits.compiler.CompilerPass`

Helper class that provides a standard way to create an ABC using inheritance.

**transpile**(*circuit*)

**class** `PhyAdaptiveDecomposition`(*gate, graph\_orig, cost\_limit=(0, 0), dimension=-1, Z\_prop=False*)

**execute**()

**Z\_extraction**(*decomposition, placement, phase\_propagation*)

**DFS**(*current\_root, level=0*) → `None`

`mqt.qudits.compiler.onedit.mapping_aware_transpilation.phy_local_qr_decomp`

Module Contents

**class** `PhyLocQRPass`(*backend*)

Bases: `mqt.qudits.compiler.CompilerPass`

Helper class that provides a standard way to create an ABC using inheritance.

**transpile**(*circuit*)

**class** `PhyQrDecomp`(*gate, graph\_orig, Z\_prop=False, not\_stand\_alone=True*)

**execute**()

Package Contents

**class** `PhyAdaptiveDecomposition`(*gate, graph\_orig, cost\_limit=(0, 0), dimension=-1, Z\_prop=False*)

**execute**()

**Z\_extraction**(*decomposition, placement, phase\_propagation*)

**DFS**(*current\_root, level=0*) → `None`

**class** `PhyLocAdaPass`(*backend*)

Bases: `mqt.qudits.compiler.CompilerPass`

Helper class that provides a standard way to create an ABC using inheritance.

**transpile**(*circuit*)

**class** `PhyLocQRPass`(*backend*)

Bases: `mqt.qudits.compiler.CompilerPass`

Helper class that provides a standard way to create an ABC using inheritance.

**transpile**(*circuit*)

**class** `PhyQrDecomp`(*gate, graph\_orig, Z\_prop=False, not\_stand\_alone=True*)

**execute()**

`mqt.qudits.compiler.onedit.mapping_un_aware_transpilation`

Submodules

`mqt.qudits.compiler.onedit.mapping_un_aware_transpilation.log_local_adaptive_decomp`

Module Contents

**class LogLocAdaPass**(*backend*)

Bases: `mqt.qudits.compiler.CompilerPass`

Helper class that provides a standard way to create an ABC using inheritance.

**transpile**(*circuit*)

**class LogAdaptiveDecomposition**(*gate, graph\_orig, cost\_limit=(0, 0), dimension=-1, Z\_prop=False*)

**execute()**

**z\_extraction**(*decomposition, placement, phase\_propagation*)

**DFS**(*current\_root, level=0*) → `None`

`mqt.qudits.compiler.onedit.mapping_un_aware_transpilation.log_local_qr_decomp`

Module Contents

**class LogLocQRPass**(*backend*)

Bases: `mqt.qudits.compiler.CompilerPass`

Helper class that provides a standard way to create an ABC using inheritance.

**transpile**(*circuit*)

**class QrDecomp**(*gate, graph\_orig, Z\_prop=False, not\_stand\_alone=True*)

**execute()**

Package Contents

**class LogAdaptiveDecomposition**(*gate, graph\_orig, cost\_limit=(0, 0), dimension=-1, Z\_prop=False*)

**execute()**

**z\_extraction**(*decomposition, placement, phase\_propagation*)

**DFS**(*current\_root, level=0*) → `None`

**class LogLocAdaPass**(*backend*)

Bases: `mqt.qudits.compiler.CompilerPass`

Helper class that provides a standard way to create an ABC using inheritance.

**transpile**(*circuit*)

**class LogLocQRPass**(*backend*)

Bases: `mqt.qudits.compiler.CompilerPass`

Helper class that provides a standard way to create an ABC using inheritance.

**transpile**(*circuit*)

Package Contents

```

class ZPropagationPass(backend, back=True)
 Bases: mqt.qudits.compiler.CompilerPass
 Helper class that provides a standard way to create an ABC using inheritance.
 transpile(circuit)

 propagate_z(circuit, line, back)

 find_intervals_with_same_target_qudits(instructions)

 remove_z(original_circuit, back=True)

class ZRemovalPass(backend)
 Bases: mqt.qudits.compiler.CompilerPass
 Helper class that provides a standard way to create an ABC using inheritance.
 transpile(circuit)

 remove_rz_gates(original_circuit, reverse=False)

 remove_initial_rz(original_circuit)

 remove_trailing_rz_sequence(original_circuit)

class PhyLocAdaPass(backend)
 Bases: mqt.qudits.compiler.CompilerPass
 Helper class that provides a standard way to create an ABC using inheritance.
 transpile(circuit)

class PhyLocQRPass(backend)
 Bases: mqt.qudits.compiler.CompilerPass
 Helper class that provides a standard way to create an ABC using inheritance.
 transpile(circuit)

class LogLocAdaPass(backend)
 Bases: mqt.qudits.compiler.CompilerPass
 Helper class that provides a standard way to create an ABC using inheritance.
 transpile(circuit)

class LogLocQRPass(backend)
 Bases: mqt.qudits.compiler.CompilerPass
 Helper class that provides a standard way to create an ABC using inheritance.
 transpile(circuit)

mqt.qudits.compiler.twodit
Subpackages
mqt.qudits.compiler.twodit.entanglement_qr
Submodules
mqt.qudits.compiler.twodit.entanglement_qr.crot
Module Contents
CEX_SEQUENCE

```



```
class CRotGen(circuit, indices)

 crot_101_as_list(theta, phi)

 permute_crot_101_as_list(i, theta, phase)

 permute_doubled_crot_101_as_list(i, theta, phase)

 z_from_crot_101_list(i, phase)

mqt.qudits.compiler.twodit.entanglement_qr.log_ent_qr_cex_decomp
Module Contents
```

```
class LogEntQRCEXPass(backend)
 Bases: mqt.qudits.compiler.compiler_pass.CompilerPass
 Helper class that provides a standard way to create an ABC using inheritance.
 transpile(circuit)
```

```
class EntangledQRCEX(gate, graph_orig_c, graph_orig_t)

 execute()
```

```
mqt.qudits.compiler.twodit.entanglement_qr.pswap
Module Contents
```

```
class PSwapGen(circuit, indices)

 pswap_101_as_list(teta, phi)

 permute_pswap_101_as_list(pos, theta, phase)

 permute_quad_pswap_101_as_list(pos, theta, phase)

 z_pswap_101_as_list(i, phase, dimension_single)
```

Package Contents

```
class CRotGen(circuit, indices)

 crot_101_as_list(theta, phi)

 permute_crot_101_as_list(i, theta, phase)

 permute_doubled_crot_101_as_list(i, theta, phase)

 z_from_crot_101_list(i, phase)

class EntangledQRCEX(gate, graph_orig_c, graph_orig_t)

 execute()

class LogEntQRCEXPass(backend)
 Bases: mqt.qudits.compiler.compiler_pass.CompilerPass
 Helper class that provides a standard way to create an ABC using inheritance.
 transpile(circuit)

class PSwapGen(circuit, indices)

 pswap_101_as_list(teta, phi)

 permute_pswap_101_as_list(pos, theta, phase)
```

```

 permute_quad_pswap_101_as_list(pos, theta, phase)

 z_pswap_101_as_list(i, phase, dimension_single)

mqt.qudits.compiler.twodit.variational_twodit_compilation
Subpackages
mqt.qudits.compiler.twodit.variational_twodit_compilation.ansatz
Submodules
mqt.qudits.compiler.twodit.variational_twodit_compilation.ansatz.ansatz_gen
Module Contents
prepare_ansatz(u, params, dims)

cu_ansatz(P, dims)

ms_ansatz(P, dims)

ls_ansatz(P, dims)

mqt.qudits.compiler.twodit.variational_twodit_compilation.ansatz.instantiate
Module Contents
ansatz_decompose(u, params, dims)

create_cu_instance(P, dims)

create_ms_instance(P, dims)

create_ls_instance(P, dims)

mqt.qudits.compiler.twodit.variational_twodit_compilation.ansatz.parametrize
Module Contents
CUSTOM_PRIMITIVE

params_splitter(params, dims)

reindex(ir, jc, num_col)

bound_1

bound_2

bound_3

generic_sud(params, dimension) → ndarray

Package Contents
cu_ansatz(P, dims)

ls_ansatz(P, dims)

ms_ansatz(P, dims)

create_cu_instance(P, dims)

create_ls_instance(P, dims)

create_ms_instance(P, dims)

```

**reindex**(*ir, jc, num\_col*)

mqt.qudits.compiler.twodit.variational\_twodit\_compilation.opt

Submodules

mqt.qudits.compiler.twodit.variational\_twodit\_compilation.opt.distance\_measures FROM

An alternative quantum fidelity for mixed states of qudits Xiaoguang Wang, 1, 2, \* Chang-Shui Yu, 3 and x. x. Yi 3

Module Contents

**size\_check**(*a: ndarray, b: ndarray*) → bool

**fidelity\_on\_operator**(*a: ndarray, b: ndarray*) → float

**fidelity\_on\_unitares**(*a: ndarray, b: ndarray*) → float

**fidelity\_on\_density\_operator**(*a: ndarray, b: ndarray*) → float

**density\_operator**(*state\_vector*) → ndarray

**frobenius\_dist**(*x, y*)

mqt.qudits.compiler.twodit.variational\_twodit\_compilation.opt.optimizer

Module Contents

**class Optimizer**

**OBJ\_FIDELITY** = 0.0001

**SINGLE\_DIM\_0**

**SINGLE\_DIM\_1**

**TARGET\_GATE**

**MAX\_NUM\_LAYERS**

**X\_SOLUTION** = []

**FUN\_SOLUTION** = []

**timer\_var** = False

**static bounds\_assigner**(*b1, b2, b3, num\_params\_single, d*)

**classmethod obj\_fun\_core**(*ansatz, lambdas*)

**classmethod objective\_fnc\_ms**(*lambdas*)

**classmethod objective\_fnc\_ls**(*lambdas*)

**classmethod objective\_fnc\_cu**(*lambdas*)

**classmethod solve\_anneal**(*bounds, ansatz\_type, result\_queue*) → None

Package Contents

**density\_operator**(*state\_vector*) → ndarray

**fidelity\_on\_density\_operator**(*a: ndarray, b: ndarray*) → float

**fidelity\_on\_operator**(*a: ndarray, b: ndarray*) → float

**fidelity\_on\_unitares**(*a: ndarray, b: ndarray*) → float

**frobenius\_dist**(*x*, *y*)

**size\_check**(*a*: ndarray, *b*: ndarray) → bool

**class Optimizer**

**OBJ\_FIDELITY** = 0.0001

**SINGLE\_DIM\_0**

**SINGLE\_DIM\_1**

**TARGET\_GATE**

**MAX\_NUM\_LAYERS**

**X\_SOLUTION** = []

**FUN\_SOLUTION** = []

**timer\_var** = False

**static bounds\_assigner**(*b1*, *b2*, *b3*, *num\_params\_single*, *d*)

**classmethod obj\_fun\_core**(*ansatz*, *lambdas*)

**classmethod objective\_fnc\_ms**(*lambdas*)

**classmethod objective\_fnc\_ls**(*lambdas*)

**classmethod objective\_fnc\_cu**(*lambdas*)

**classmethod solve\_anneal**(*bounds*, *ansatz\_type*, *result\_queue*) → None

Submodules

mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz\_solve\_n\_search

Module Contents

**interrupt\_function**() → None

**binary\_search\_compile**(*max\_num\_layer*, *ansatz\_type*)

**run**(*num\_layer*, *ansatz\_type*)

Package Contents

**class LogEntQRCEXPass**(*backend*)

    Bases: [mqt.qudits.compiler.compiler\\_pass.CompilerPass](#)

    Helper class that provides a standard way to create an ABC using inheritance.

**transpile**(*circuit*)

**Submodules**

mqt.qudits.compiler.compiler\_pass

Module Contents

**class CompilerPass**(*backend*, *\*\*kwargs*)

    Bases: [abc.ABC](#)

    Helper class that provides a standard way to create an ABC using inheritance.

```

 abstract transpile(circuit)

mqt.qudits.compiler.dit_manager
Module Contents
class QuditCompiler

 passes_enabled

 compile(backend, circuit, passes_names)

```

## Package Contents

```

class CompilerPass(backend, **kwargs)
 Bases: abc.ABC

 Helper class that provides a standard way to create an ABC using inheritance.

 abstract transpile(circuit)

```

## mqt.qudits.core

Core structure used in the package.

## Submodules

mqt.qudits.core.dfs\_tree

Module Contents

```

class Node(key, rotation, U_of_level, graph_current, current_cost, current_decomp_cost, max_cost,
 pi_pulses, parent_key, children=None)

 add(new_key, rotation, U_of_level, graph_current, current_cost, current_decomp_cost, max_cost,
 pi_pulses) → None

 __str__() → str
 Return str(self).

```

## class NAryTree

```

 property total_size

 add(new_key, rotation, U_of_level, graph_current, current_cost, current_decomp_cost, max_cost,
 pi_pulses, parent_key=None) → None

 find_node(node, key)

 depth(key)

 max_depth(node)

 size_refresh(node)

 found_checker(node)

 min_cost_decomp(node)

 retrieve_decomposition(node)

 is_empty()

 print_tree(node, str_aux)

```

`__str__()` → str

Return str(self).

`mqt.qudits.core.level_graph`

Module Contents

**class LevelGraph**(edges, nodes, nodes\_physical\_mapping=None, initialization\_nodes=None, qudit\_index=None, og\_circuit=None)

Bases: `networkx.Graph`

Base class for undirected graphs.

A Graph stores nodes and edges with optional data, or attributes.

Graphs hold undirected edges. Self loops are allowed but multiple (parallel) edges are not.

Nodes can be arbitrary (hashable) Python objects with optional key/value attributes, except that *None* is not allowed as a node.

Edges are represented as links between nodes with optional key/value attributes.

Parameters

`incoming_graph` [graph] (optional, default: None) Data to initialize graph. If None (default) an empty graph is created. The data can be any format that is supported by the `to_networkx_graph()` function, currently including edge list, dict of dicts, dict of lists, NetworkX graph, 2D NumPy array, SciPy sparse matrix, or PyGraphviz graph.  
`attr` [keyword arguments, optional (default= no attributes)] Attributes to add to graph as key=value pairs.

See Also `DiGraph` `MultiGraph` `MultiDiGraph`

Examples Create an empty graph structure (a “null graph”) with no nodes and no edges.

```
>>> G = nx.Graph()
```

G can be grown in several ways.

**Nodes:**

Add one node at a time:

```
>>> G.add_node(1)
```

Add the nodes from any container (a list, dict, set or even the lines from a file or the nodes from another graph).

```
>>> G.add_nodes_from([2, 3])
>>> G.add_nodes_from(range(100, 110))
>>> H = nx.path_graph(10)
>>> G.add_nodes_from(H)
```

In addition to strings and integers any hashable Python object (except None) can represent a node, e.g. a customized node object, or even another Graph.

```
>>> G.add_node(H)
```

**Edges:**

G can also be grown by adding edges.

Add one edge,

```
>>> G.add_edge(1, 2)
```

a list of edges,

```
>>> G.add_edges_from([(1, 2), (1, 3)])
```

or a collection of edges,

```
>>> G.add_edges_from(H.edges)
```

If some edges connect nodes not yet in the graph, the nodes are added automatically. There are no errors when adding nodes or edges that already exist.

### Attributes:

Each graph, node, and edge can hold key/value attribute pairs in an associated attribute dictionary (the keys must be hashable). By default these are empty, but can be added or changed using `add_edge`, `add_node` or direct manipulation of the attribute dictionaries named `graph`, `node` and `edge` respectively.

```
>>> G = nx.Graph(day="Friday")
>>> G.graph
{'day': 'Friday'}
```

Add node attributes using `add_node()`, `add_nodes_from()` or `G.nodes`

```
>>> G.add_node(1, time="5pm")
>>> G.add_nodes_from([3], time="2pm")
>>> G.nodes[1]
{'time': '5pm'}
>>> G.nodes[1]["room"] = 714 # node must exist already to use G.nodes
>>> del G.nodes[1]["room"] # remove attribute
>>> list(G.nodes(data=True))
[(1, {'time': '5pm'}), (3, {'time': '2pm'})]
```

Add edge attributes using `add_edge()`, `add_edges_from()`, subscript notation, or `G.edges`.

```
>>> G.add_edge(1, 2, weight=4.7)
>>> G.add_edges_from([(3, 4), (4, 5)], color="red")
>>> G.add_edges_from([(1, 2, {"color": "blue"}), (2, 3, {"weight": 8})])
>>> G[1][2]["weight"] = 4.7
>>> G.edges[1, 2]["weight"] = 4
```

Warning: we protect the graph data structure by making `G.edges` a read-only dict-like structure. However, you can assign to attributes in e.g. `G.edges[1, 2]`. Thus, use 2 sets of brackets to add/change data attributes: `G.edges[1, 2]['weight'] = 4` (For multigraphs: `MG.edges[u, v, key][name] = value`).

### Shortcuts:

Many common graph features allow python syntax to speed reporting.

```
>>> 1 in G # check if node in graph
True
>>> [n for n in G if n < 3] # iterate through nodes
[1, 2]
>>> len(G) # number of nodes in graph
5
```

Often the best way to traverse all edges of a graph is via the neighbors. The neighbors are reported as an adjacency-dict `G.adj` or `G.adjacency()`

```
>>> for n, nbrsdict in G.adjacency():
... for nbr, eattr in nbrsdict.items():
... if "weight" in eattr:
... # Do something useful with the edges
... pass
```

But the `edges()` method is often more convenient:

```
>>> for u, v, weight in G.edges.data("weight"):
... if weight is not None:
... # Do something useful with the edges
... pass
```

## Reporting:

Simple graph information is obtained using object-attributes and methods. Reporting typically provides views instead of containers to reduce memory usage. The views update as the graph is updated similarly to dict-views. The objects *nodes*, *edges* and *adj* provide access to data attributes via lookup (e.g. *nodes[n]*, *edges[u, v]*, *adj[u][v]*) and iteration (e.g. *nodes.items()*, *nodes.data('color')*, *nodes.data('color', default='blue')*) and similarly for *edges*) Views exist for *nodes*, *edges*, *neighbors()/adj* and *degree*.

For details on these and other miscellaneous methods, see below.

## Subclasses (Advanced):

The Graph class uses a dict-of-dict-of-dict data structure. The outer dict (node\_dict) holds adjacency information keyed by node. The next dict (adjlist\_dict) represents the adjacency information and holds edge data keyed by neighbor. The inner dict (edge\_attr\_dict) represents the edge data and holds edge attribute values keyed by attribute names.

Each of these three dicts can be replaced in a subclass by a user defined dict-like object. In general, the dict-like features should be maintained but extra features can be added. To replace one of the dicts create a new graph class by changing the class(!) variable holding the factory for that dict-like structure.

node\_dict[~~factory~~function, (default: dict)] Factory function to be used to create the dict containing node attributes, keyed by node id. It should require no arguments and return a dict-like object

node\_attr\_~~dict~~function (default: dict) Factory function to be used to create the node attribute dict which holds attribute values keyed by attribute name. It should require no arguments and return a dict-like object

adjlist\_out[~~dict~~function, (default: dict)] Factory function to be used to create the outer-most dict in the data structure that holds adjacency info keyed by node. It should require no arguments and return a dict-like object.

adjlist\_in[~~dict~~function, (default: dict)] Factory function to be used to create the adjacency list dict which holds edge data keyed by neighbor. It should require no arguments and return a dict-like object

edge\_attr\_~~dict~~function, (default: dict)] Factory function to be used to create the edge attribute dict which holds attribute values keyed by attribute name. It should require no arguments and return a dict-like object.

graph\_attr[~~dict~~function, (default: dict)] Factory function to be used to create the graph attribute dict which holds attribute values keyed by attribute name. It should require no arguments and return a dict-like object.

Typically, if your extension doesn't impact the data structure all methods will inherit without issue except: *to\_directed/to\_undirected*. By default these methods create a DiGraph/Graph class and you probably want them to create your extension of a DiGraph/Graph. To facilitate this we define two class variables that you can set in your subclass.

to\_directed[~~class~~class, (default: DiGraph or MultiDiGraph)] Class to create a new graph structure in the *to\_directed* method. If *None*, a NetworkX class (DiGraph or MultiDiGraph) is used.

to\_undirected[~~class~~class, (default: Graph or MultiGraph)] Class to create a new graph structure in the *to\_undirected* method. If *None*, a NetworkX class (Graph or MultiGraph) is used.

## Subclassing Example

Create a low memory graph class that effectively disallows edge attributes by using a single attribute dict for all edges. This reduces the memory used, but you lose edge attributes.



```

>>> class ThinGraph(nx.Graph):
... all_edge_dict = {"weight": 1}
...
... def single_edge_dict(self):
... return self.all_edge_dict
...
... edge_attr_dict_factory = single_edge_dict
>>> G = ThinGraph()
>>> G.add_edge(2, 1)
>>> G[2][1]
{'weight': 1}
>>> G.add_edge(2, 2)
>>> G[2][1] is G[2][2]
True

```

**property** `log_phy_map`

**phase\_storing\_setup()** → `None`

**distance\_nodes**(*source*, *target*)

**distance\_nodes\_pi\_pulses\_fixed\_ancilla**(*source*, *target*)

**logic\_physical\_map**(*physical\_nodes*) → `None`

**define\_\_states**(*initialization\_nodes*, *inreach\_nodes*) → `None`

**update\_list**(*lst*\_, *num\_a*, *num\_b*)

**deep\_copy\_func**(*l\_n*)

**index**(*lev\_graph*, *node*)

**swap\_node\_attributes**(*node\_a*, *node\_b*)

**swap\_node\_attr\_simple**(*node\_a*, *node\_b*) → `None`

**swap\_nodes**(*node\_a*, *node\_b*)

**get\_VRz\_gates**()

**get\_node\_sensitivity\_cost**(*node*)

**get\_edge\_sensitivity**(*node\_a*, *node\_b*)

**is\_irnode**(*node*)

**is\_Inode**(*node*)

**\_\_str\_\_**() → `str`

Returns a short summary of the graph.

Returns

**info** [string] Graph information including the graph name (if any), graph type, and the number of nodes and edges.

Examples

```

>>> G = nx.Graph(name="foo")
>>> str(G)
"Graph named 'foo' with 0 nodes and 0 edges"

```

```

>>> G = nx.path_graph(3)
>>> str(G)
'Graph with 3 nodes and 2 edges'

```

**set\_circuit**(circuit: *mqt.qudits.circuit.QuantumCircuit*) → *None*

**set\_qudits\_index**(index: *int*) → *None*

## Package Contents

**class** *NaryTree*

**property** *total\_size*

**add**(new\_key, rotation, *U\_of\_level*, graph\_current, current\_cost, current\_decomp\_cost, max\_cost, pi\_pulses, parent\_key=*None*) → *None*

**find\_node**(node, key)

**depth**(key)

**max\_depth**(node)

**size\_refresh**(node)

**found\_checker**(node)

**min\_cost\_decomp**(node)

**retrieve\_decomposition**(node)

**is\_empty**()

**print\_tree**(node, str\_aux)

**\_\_str\_\_**() → *str*

Return str(self).

**class** *Node*(key, rotation, *U\_of\_level*, graph\_current, current\_cost, current\_decomp\_cost, max\_cost, pi\_pulses, parent\_key, children=*None*)

**add**(new\_key, rotation, *U\_of\_level*, graph\_current, current\_cost, current\_decomp\_cost, max\_cost, pi\_pulses) → *None*

**\_\_str\_\_**() → *str*

Return str(self).

**class** *LevelGraph*(edges, nodes, nodes\_physical\_mapping=*None*, initialization\_nodes=*None*, qudit\_index=*None*, og\_circuit=*None*)

Bases: *networkx.Graph*

Base class for undirected graphs.

A Graph stores nodes and edges with optional data, or attributes.

Graphs hold undirected edges. Self loops are allowed but multiple (parallel) edges are not.

Nodes can be arbitrary (hashable) Python objects with optional key/value attributes, except that *None* is not allowed as a node.

Edges are represented as links between nodes with optional key/value attributes.

Parameters

incoming *graph* [optional, default: *None*]] Data to initialize graph. If *None* (default) an empty graph is created. The data can be any format that is supported by the *to\_networkx\_graph()* function, currently including edge list, dict of dicts, dict of lists, NetworkX graph, 2D NumPy array, SciPy sparse matrix, or PyGraphviz graph.

attr [keyword arguments, optional (default= no attributes)] Attributes to add to graph as key=value pairs.

See Also [DiGraph](#) [MultiGraph](#) [MultiDiGraph](#)

Examples Create an empty graph structure (a “null graph”) with no nodes and no edges.

```
>>> G = nx.Graph()
```

G can be grown in several ways.

#### Nodes:

Add one node at a time:

```
>>> G.add_node(1)
```

Add the nodes from any container (a list, dict, set or even the lines from a file or the nodes from another graph).

```
>>> G.add_nodes_from([2, 3])
>>> G.add_nodes_from(range(100, 110))
>>> H = nx.path_graph(10)
>>> G.add_nodes_from(H)
```

In addition to strings and integers any hashable Python object (except None) can represent a node, e.g. a customized node object, or even another Graph.

```
>>> G.add_node(H)
```

#### Edges:

G can also be grown by adding edges.

Add one edge,

```
>>> G.add_edge(1, 2)
```

a list of edges,

```
>>> G.add_edges_from([(1, 2), (1, 3)])
```

or a collection of edges,

```
>>> G.add_edges_from(H.edges)
```

If some edges connect nodes not yet in the graph, the nodes are added automatically. There are no errors when adding nodes or edges that already exist.

#### Attributes:

Each graph, node, and edge can hold key/value attribute pairs in an associated attribute dictionary (the keys must be hashable). By default these are empty, but can be added or changed using `add_edge`, `add_node` or direct manipulation of the attribute dictionaries named `graph`, `node` and `edge` respectively.

```
>>> G = nx.Graph(day="Friday")
>>> G.graph
{'day': 'Friday'}
```

Add node attributes using `add_node()`, `add_nodes_from()` or `G.nodes`

```
>>> G.add_node(1, time="5pm")
>>> G.add_nodes_from([3], time="2pm")
>>> G.nodes[1]
{'time': '5pm'}
>>> G.nodes[1]["room"] = 714 # node must exist already to use G.nodes
>>> del G.nodes[1]["room"] # remove attribute
>>> list(G.nodes(data=True))
[(1, {'time': '5pm'}), (3, {'time': '2pm'})]
```

Add edge attributes using `add_edge()`, `add_edges_from()`, subscript notation, or `G.edges`.

```
>>> G.add_edge(1, 2, weight=4.7)
>>> G.add_edges_from([(3, 4), (4, 5)], color="red")
>>> G.add_edges_from([(1, 2, {"color": "blue"}), (2, 3, {"weight": 8})])
>>> G[1][2]["weight"] = 4.7
>>> G.edges[1, 2]["weight"] = 4
```

Warning: we protect the graph data structure by making `G.edges` a read-only dict-like structure. However, you can assign to attributes in e.g. `G.edges[1, 2]`. Thus, use 2 sets of brackets to add/change data attributes: `G.edges[1, 2]['weight'] = 4` (For multigraphs: `MG.edges[u, v, key][name] = value`).

### Shortcuts:

Many common graph features allow python syntax to speed reporting.

```
>>> 1 in G # check if node in graph
True
>>> [n for n in G if n < 3] # iterate through nodes
[1, 2]
>>> len(G) # number of nodes in graph
5
```

Often the best way to traverse all edges of a graph is via the neighbors. The neighbors are reported as an adjacency-dict `G.adj` or `G.adjacency()`

```
>>> for n, nbrsdict in G.adjacency():
... for nbr, eattr in nbrsdict.items():
... if "weight" in eattr:
... # Do something useful with the edges
... pass
```

But the `edges()` method is often more convenient:

```
>>> for u, v, weight in G.edges.data("weight"):
... if weight is not None:
... # Do something useful with the edges
... pass
```

### Reporting:

Simple graph information is obtained using object-attributes and methods. Reporting typically provides views instead of containers to reduce memory usage. The views update as the graph is updated similarly to dict-views. The objects `nodes`, `edges` and `adj` provide access to data attributes via lookup (e.g. `nodes[n]`, `edges[u, v]`, `adj[u][v]`) and iteration (e.g. `nodes.items()`, `nodes.data('color')`, `nodes.data('color', default='blue')`) and similarly for `edges`). Views exist for `nodes`, `edges`, `neighbors()`/`adj` and `degree`.

For details on these and other miscellaneous methods, see below.

### Subclasses (Advanced):

The Graph class uses a dict-of-dict-of-dict data structure. The outer dict (`node_dict`) holds adjacency information keyed by node. The next dict (`adjlist_dict`) represents the adjacency information and holds edge data keyed by neighbor. The inner dict (`edge_attr_dict`) represents the edge data and holds edge attribute values keyed by attribute names.

Each of these three dicts can be replaced in a subclass by a user defined dict-like object. In general, the dict-like features should be maintained but extra features can be added. To replace one of the dicts create a new graph class by changing the `class()` variable holding the factory for that dict-like structure.

`node_dict` ~~[factory, (default: dict)]~~ Factory function to be used to create the dict containing node attributes, keyed by node id. It should require no arguments and return a dict-like object  
`node_attr` ~~dict~~ ~~factory~~ ~~function~~ ~~to be used to create~~ ~~the node attribute dict which holds attribute values keyed by attribute name. It should require no arguments and return a dict-like object~~

**adjlist\_out\_dict\_factory**(default: dict)] Factory function to be used to create the outer-most dict in the data structure that holds adjacency info keyed by node. It should require no arguments and return a dict-like object.

**adjlist\_in\_dict\_factory**(default: dict)] Factory function to be used to create the adjacency list dict which holds edge data keyed by neighbor. It should require no arguments and return a dict-like object

**edge\_attr\_dict\_factory**(default: dict)] Factory function to be used to create the edge attribute dict which holds attribute values keyed by attribute name. It should require no arguments and return a dict-like object.

**graph\_attr\_dict\_factory**(default: dict)] Factory function to be used to create the graph attribute dict which holds attribute values keyed by attribute name. It should require no arguments and return a dict-like object.

Typically, if your extension doesn't impact the data structure all methods will inherit without issue except: *to\_directed/to\_undirected*. By default these methods create a DiGraph/Graph class and you probably want them to create your extension of a DiGraph/Graph. To facilitate this we define two class variables that you can set in your subclass.

**to\_directed\_class**, (default: DiGraph or MultiDiGraph)] Class to create a new graph structure in the *to\_directed* method. If *None*, a NetworkX class (DiGraph or MultiDiGraph) is used.

**to\_undirected\_class**, (default: Graph or MultiGraph)] Class to create a new graph structure in the *to\_undirected* method. If *None*, a NetworkX class (Graph or MultiGraph) is used.

### Subclassing Example

Create a low memory graph class that effectively disallows edge attributes by using a single attribute dict for all edges. This reduces the memory used, but you lose edge attributes.

```
>>> class ThinGraph(nx.Graph):
... all_edge_dict = {"weight": 1}
...
... def single_edge_dict(self):
... return self.all_edge_dict
...
... edge_attr_dict_factory = single_edge_dict
>>> G = ThinGraph()
>>> G.add_edge(2, 1)
>>> G[2][1]
{'weight': 1}
>>> G.add_edge(2, 2)
>>> G[2][1] is G[2][2]
True
```

**property** log\_phy\_map

**phase\_storing\_setup**() → None

**distance\_nodes**(source, target)

**distance\_nodes\_pi\_pulses\_fixed\_ancilla**(source, target)

**logic\_physical\_map**(physical\_nodes) → None

**define\_\_states**(initialization\_nodes, inreach\_nodes) → None

**update\_list**(lst\_, num\_a, num\_b)

**deep\_copy\_func**(l\_n)

**index**(lev\_graph, node)

**swap\_node\_attributes**(node\_a, node\_b)

**swap\_node\_attr\_simple**(node\_a, node\_b) → None

`swap_nodes(node_a, node_b)`

`get_VRz_gates()`

`get_node_sensitivity_cost(node)`

`get_edge_sensitivity(node_a, node_b)`

`is_irnode(node)`

`is_Inode(node)`

`__str__() → str`

Returns a short summary of the graph.

Returns

info [string] Graph information including the graph name (if any), graph type, and the number of nodes and edges.

Examples

```
>>> G = nx.Graph(name="foo")
>>> str(G)
"Graph named 'foo' with 0 nodes and 0 edges"
```

```
>>> G = nx.path_graph(3)
>>> str(G)
'Graph with 3 nodes and 2 edges'
```

`set_circuit(circuit: mqt.qudits.circuit.QuantumCircuit) → None`

`set_qudits_index(index: int) → None`

## **mqt.qudits.exceptions**

Exceptions module.

### **Submodules**

`mqt.qudits.exceptions.backendserror`

Module Contents

**exception BackendNotFoundError**(*message: str*)

Bases: `Exception`

Common base class for all non-exit exceptions.

`mqt.qudits.exceptions.circuiterror`

Module Contents

**exception CircuitError**(*message: str*)

Bases: `Exception`

Common base class for all non-exit exceptions.

`mqt.qudits.exceptions.compilerexception`

Module Contents

**exception NodeNotFoundException**(*value*)

Bases: `Exception`

Common base class for all non-exit exceptions.

`__str__() → str`

Return `str(self)`.

**exception** `SequenceFoundException(node_key: int = -1)`

Bases: `Exception`

Common base class for all non-exit exceptions.

`__str__() → str`

Return `str(self)`.

**exception** `RoutingException`

Bases: `Exception`

Common base class for all non-exit exceptions.

`__str__() → str`

Return `str(self)`.

**exception** `FidelityReachException(message: str = "")`

Bases: `Exception`

Common base class for all non-exit exceptions.

`mqt.qudits.exceptions.joberror`

Module Contents

**exception** `JobError(message: str)`

Bases: `Exception`

Common base class for all non-exit exceptions.

**class** `JobTimeoutError(message: str)`

## Package Contents

**exception** `BackendNotFoundError(message: str)`

Bases: `Exception`

Common base class for all non-exit exceptions.

**exception** `CircuitError(message: str)`

Bases: `Exception`

Common base class for all non-exit exceptions.

**exception** `FidelityReachException(message: str = "")`

Bases: `Exception`

Common base class for all non-exit exceptions.

**exception** `NodeNotFoundException(value)`

Bases: `Exception`

Common base class for all non-exit exceptions.

`__str__() → str`

Return `str(self)`.

**exception** `RoutingException`

Bases: `Exception`

Common base class for all non-exit exceptions.

`__str__() → str`

Return `str(self)`.

**exception SequenceFoundException**(*node\_key: int = -1*)

Bases: [Exception](#)

Common base class for all non-exit exceptions.

**\_\_str\_\_**() → *str*

Return str(self).

**exception JobError**(*message: str*)

Bases: [Exception](#)

Common base class for all non-exit exceptions.

**class JobTimeoutError**(*message: str*)

**mqt.qudits.quantum\_circuit**

Qudit Quantum Circuit Module.

### Subpackages

[mqt.qudits.quantum\\_circuit.components](#)

Subpackages

[mqt.qudits.quantum\\_circuit.components.extensions](#)

Submodules

[mqt.qudits.quantum\\_circuit.components.extensions.controls](#)

Module Contents

**class ControlData**

**indices:** [list\[int\]](#) | *int*

**ctrl\_states:** [list\[int\]](#) | *int*

[mqt.qudits.quantum\\_circuit.components.extensions.gate\\_types](#)

Module Contents

**class GateTypes**(\*args, \*\*kws)

Bases: [enum.Enum](#)

Enumeration for job status.

**SINGLE** = 'Single Qudit Gate'

**TWO** = 'Two Qudit Gate'

**MULTI** = 'Multi Qudit Gate'

**CORE\_GATE\_TYPES** = ()

[mqt.qudits.quantum\\_circuit.components.extensions.matrix\\_factory](#)

Module Contents

**class MatrixFactory**(*gate, identities\_flag*)

**generate\_matrix**()

**classmethod apply\_identities\_and\_controls**(*matrix, qudits\_applied, dimensions, ref\_lines, controls=None, controls\_levels=None*)

**classmethod wrap\_in\_identities**(*matrix, indices, sizes*)



**from\_dirac\_to\_basis**(*vec, d*)

**calculate\_q0\_q1**(*lev, dim*)

**insert\_at**(*big\_arr, pos, to\_insert\_arr*)

Quite a forceful way of embedding a parameters into big\_arr.

Submodules

mqt.qudits.quantum\_circuit.components.classic\_register

Module Contents

**class ClassicRegister**(*name, size*)

**classmethod from\_map**(*sitemap: dict*) → list[*ClassicRegister*]

**\_\_qasm\_\_**()

**\_\_getitem\_\_**(*key*)

mqt.qudits.quantum\_circuit.components.quantum\_register

Module Contents

**class QuantumRegister**(*name, size, dims=None*)

**classmethod from\_map**(*sitemap: dict*) → list[*QuantumRegister*]

**\_\_qasm\_\_**()

**\_\_getitem\_\_**(*key*)

Package Contents

**class ClassicRegister**(*name, size*)

**classmethod from\_map**(*sitemap: dict*) → list[*ClassicRegister*]

**\_\_qasm\_\_**()

**\_\_getitem\_\_**(*key*)

**class QuantumRegister**(*name, size, dims=None*)

**classmethod from\_map**(*sitemap: dict*) → list[*QuantumRegister*]

**\_\_qasm\_\_**()

**\_\_getitem\_\_**(*key*)

mqt.qudits.quantum\_circuit.gates Instructions module.

Submodules

mqt.qudits.quantum\_circuit.gates.csum

Module Contents

**class CSum**(*circuit: QuantumCircuit, name: str, target\_qudits: list[int] | int, dimensions: list[int] | int, controls: ControlData | None = None*)

Bases: *mqt.qudits.quantum\_circuit.gate.Gate*

Unitary gate\_matrix.

**\_\_array\_\_**() → ndarray

**validate\_parameter**(*parameter=None*) → bool

`__str__()` → str

Return str(self).

`mqt.qudits.quantum_circuit.gates.custom_multi`

Module Contents

**class CustomMulti**(circuit: QuantumCircuit, name: str, target\_qudits: list[int] | int, parameters: ndarray, dimensions: list[int] | int, controls: ControlData | None = None)

Bases: `mqt.qudits.quantum_circuit.gate.Gate`

Multi body custom gate

`__array__()` → ndarray

**validate\_parameter**(parameter=None) → bool

`__str__()` → str

Return str(self).

`mqt.qudits.quantum_circuit.gates.custom_one`

Module Contents

**class CustomOne**(circuit: QuantumCircuit, name: str, target\_qudits: list[int] | int, parameters: ndarray, dimensions: list[int] | int, controls: ControlData | None = None)

Bases: `mqt.qudits.quantum_circuit.gate.Gate`

One body custom gate

`__array__()` → ndarray

**validate\_parameter**(parameter=None) → bool

`__str__()` → str

Return str(self).

`mqt.qudits.quantum_circuit.gates.custom_two`

Module Contents

**class CustomTwo**(circuit: QuantumCircuit, name: str, target\_qudits: list[int] | int, parameters: ndarray, dimensions: list[int] | int, controls: ControlData | None = None)

Bases: `mqt.qudits.quantum_circuit.gate.Gate`

Two body custom gate

`__array__()` → ndarray

**validate\_parameter**(parameter=None) → bool

`__str__()` → str

Return str(self).

`mqt.qudits.quantum_circuit.gates.cx`

Module Contents

**class CEx**(circuit: QuantumCircuit, name: str, target\_qudits: list[int] | int, parameters: list | None, dimensions: list[int] | int, controls: ControlData | None = None)

Bases: `mqt.qudits.quantum_circuit.gate.Gate`

Unitary gate\_matrix.

`__array__()` → ndarray

**validate\_parameter**(parameter) → bool

`__str__()` → str

Return str(self).

`mqt.qudits.quantum_circuit.gates.gellmann`

Module Contents

**class** `GellMann`(*circuit*: `QuantumCircuit`, *name*: str, *target\_qudits*: list[int] | int, *parameters*: list, *dimensions*: list[int] | int, *controls*: `ControlData` | None = None)

Bases: `mqt.qudits.quantum_circuit.gate.Gate`

Gate used as generator for Givens rotations.

`__array__()` → ndarray

**validate\_parameter**(*parameter*) → bool

`__str__()` → str

Return str(self).

`mqt.qudits.quantum_circuit.gates.h`

Module Contents

**class** `H`(*circuit*: `QuantumCircuit`, *name*: str, *target\_qudits*: list[int] | int, *dimensions*: list[int] | int, *controls*: `ControlData` | None = None)

Bases: `mqt.qudits.quantum_circuit.gate.Gate`

Unitary gate\_matrix.

`__array__()` → ndarray

**validate\_parameter**(*parameter*=None) → bool

`__str__()` → str

Return str(self).

`mqt.qudits.quantum_circuit.gates.ls`

Module Contents

**class** `LS`(*circuit*: `QuantumCircuit`, *name*: str, *target\_qudits*: list[int] | int, *parameters*: list | None, *dimensions*: list[int] | int, *controls*: `ControlData` | None = None)

Bases: `mqt.qudits.quantum_circuit.gate.Gate`

Unitary gate\_matrix.

`__array__()` → ndarray

**validate\_parameter**(*parameter*) → bool

`__str__()` → str

Return str(self).

`mqt.qudits.quantum_circuit.gates.ms`

Module Contents

**class** `MS`(*circuit*: `QuantumCircuit`, *name*: str, *target\_qudits*: list[int] | int, *parameters*: list | None, *dimensions*: list[int] | int, *controls*: `ControlData` | None = None)

Bases: `mqt.qudits.quantum_circuit.gate.Gate`

Unitary gate\_matrix.

`__array__()` → ndarray

**validate\_parameter**(*parameter*) → bool

`__str__()` → str

Return str(self).

`mqt.qudits.quantum_circuit.gates.perm`

Module Contents

**class Perm**(circuit: QuantumCircuit, name: str, target\_qubits: list[int] | int, parameters: list, dimensions: list[int] | int, controls: ControlData | None = None)

Bases: `mqt.qudits.quantum_circuit.gate.Gate`

Unitary gate\_matrix.

`__array__()` → ndarray

**validate\_parameter**(parameter) → bool

Verify that the input is a list of indices

`__str__()` → str

Return str(self).

`mqt.qudits.quantum_circuit.gates.r`

Module Contents

**class R**(circuit: QuantumCircuit, name: str, target\_qubits: list[int] | int, parameters: list | None, dimensions: list[int] | int, controls: ControlData | None = None)

Bases: `mqt.qudits.quantum_circuit.gate.Gate`

Unitary gate\_matrix.

**property cost**

`__array__()` → ndarray

**levels\_setter**(la, lb)

**validate\_parameter**(parameter) → bool

`__str__()` → str

Return str(self).

`mqt.qudits.quantum_circuit.gates.randu`

Module Contents

**class RandU**(circuit: QuantumCircuit, name: str, target\_qubits: list[int] | int, dimensions: list[int] | int, controls: ControlData | None = None)

Bases: `mqt.qudits.quantum_circuit.gate.Gate`

Unitary gate\_matrix.

`__array__()` → ndarray

**validate\_parameter**() → bool

`__str__()` → str

Return str(self).

`mqt.qudits.quantum_circuit.gates.rh`

Module Contents

**class Rh**(circuit: QuantumCircuit, name: str, target\_qubits: list[int] | int, parameters: list | None, dimensions: list[int] | int, controls: ControlData | None = None)

Bases: `mqt.qudits.quantum_circuit.gate.Gate`

SU2 Hadamard

```

__array__() → ndarray

levels_setter(la, lb)

validate_parameter(parameter) → bool

__str__() → str
 Return str(self).

mqt.qudits.quantum_circuit.gates.rz
Module Contents

class Rz(circuit: QuantumCircuit, name: str, target_qubits: list[int] | int, parameters: list | None,
 dimensions: list[int] | int, controls: ControlData | None = None)
 Bases: mqt.qudits.quantum_circuit.gate.Gate
 Unitary gate_matrix.
 property cost

__array__() → ndarray

levels_setter(la, lb)

validate_parameter(parameter) → bool

__str__() → str
 Return str(self).

mqt.qudits.quantum_circuit.gates.s
Module Contents

class S(circuit: QuantumCircuit, name: str, target_qubits: list[int] | int, dimensions: list[int] | int, controls:
 ControlData | None = None)
 Bases: mqt.qudits.quantum_circuit.gate.Gate
 Unitary gate_matrix.

__array__() → ndarray

validate_parameter(parameter=None) → bool

__str__() → str
 Return str(self).

mqt.qudits.quantum_circuit.gates.virt_rz
Module Contents

class VirtRz(circuit: QuantumCircuit, name: str, target_qubits: list[int] | int, parameters: list | None,
 dimensions: list[int] | int, controls: ControlData | None = None)
 Bases: mqt.qudits.quantum_circuit.gate.Gate
 Unitary gate_matrix.
 property cost

__array__() → ndarray

validate_parameter(parameter) → bool

```

```

 __str__() → str
 Return str(self).
mqt.qudits.quantum_circuit.gates.x
Module Contents
class X(circuit: QuantumCircuit, name: str, target_qudits: list[int] | int, dimensions: list[int] | int, controls:
 ControlData | None = None)
 Bases: mqt.qudits.quantum_circuit.gate.Gate
 Unitary gate_matrix.
 __array__() → ndarray
 validate_parameter(parameter=None) → bool
 __str__() → str
 Return str(self).
mqt.qudits.quantum_circuit.gates.z
Module Contents
class Z(circuit: QuantumCircuit, name: str, target_qudits: list[int] | int, dimensions: list[int] | int, controls:
 ControlData | None = None)
 Bases: mqt.qudits.quantum_circuit.gate.Gate
 Unitary gate_matrix.
 __array__() → ndarray
 validate_parameter(parameter=None) → bool
 __str__() → str
 Return str(self).
Package Contents
class ControlData
 indices: list[int] | int
 ctrl_states: list[int] | int
class GateTypes(*args, **kws)
 Bases: enum.Enum
 Enumeration for job status.
 SINGLE = 'Single Qudit Gate'
 TWO = 'Two Qudit Gate'
 MULTI = 'Multi Qudit Gate'
class CSum(circuit: QuantumCircuit, name: str, target_qudits: list[int] | int, dimensions: list[int] | int,
 controls: ControlData | None = None)
 Bases: mqt.qudits.quantum_circuit.gate.Gate
 Unitary gate_matrix.
 __array__() → ndarray
 validate_parameter(parameter=None) → bool

```

```

 __str__() → str
 Return str(self).

class CustomMulti(circuit: QuantumCircuit, name: str, target_qubits: list[int] | int, parameters: ndarray,
 dimensions: list[int] | int, controls: ControlData | None = None)

 Bases: mqt.qudits.quantum_circuit.gate.Gate

 Multi body custom gate

 __array__() → ndarray

 validate_parameter(parameter=None) → bool

 __str__() → str
 Return str(self).

class CustomOne(circuit: QuantumCircuit, name: str, target_qubits: list[int] | int, parameters: ndarray,
 dimensions: list[int] | int, controls: ControlData | None = None)

 Bases: mqt.qudits.quantum_circuit.gate.Gate

 One body custom gate

 __array__() → ndarray

 validate_parameter(parameter=None) → bool

 __str__() → str
 Return str(self).

class CustomTwo(circuit: QuantumCircuit, name: str, target_qubits: list[int] | int, parameters: ndarray,
 dimensions: list[int] | int, controls: ControlData | None = None)

 Bases: mqt.qudits.quantum_circuit.gate.Gate

 Two body custom gate

 __array__() → ndarray

 validate_parameter(parameter=None) → bool

 __str__() → str
 Return str(self).

class CEx(circuit: QuantumCircuit, name: str, target_qubits: list[int] | int, parameters: list | None,
 dimensions: list[int] | int, controls: ControlData | None = None)

 Bases: mqt.qudits.quantum_circuit.gate.Gate

 Unitary gate_matrix.

 __array__() → ndarray

 validate_parameter(parameter) → bool

 __str__() → str
 Return str(self).

class GellMann(circuit: QuantumCircuit, name: str, target_qubits: list[int] | int, parameters: list,
 dimensions: list[int] | int, controls: ControlData | None = None)

 Bases: mqt.qudits.quantum_circuit.gate.Gate

 Gate used as generator for Givens rotations.

 __array__() → ndarray

 validate_parameter(parameter) → bool

```

```

 __str__() → str
 Return str(self).

class H(circuit: QuantumCircuit, name: str, target_qubits: list[int] | int, dimensions: list[int] | int, controls:
 ControlData | None = None)
 Bases: mqt.qudits.quantum_circuit.gate.Gate
 Unitary gate_matrix.
 __array__() → ndarray
 validate_parameter(parameter=None) → bool

 __str__() → str
 Return str(self).

class LS(circuit: QuantumCircuit, name: str, target_qubits: list[int] | int, parameters: list | None,
 dimensions: list[int] | int, controls: ControlData | None = None)
 Bases: mqt.qudits.quantum_circuit.gate.Gate
 Unitary gate_matrix.
 __array__() → ndarray
 validate_parameter(parameter) → bool

 __str__() → str
 Return str(self).

class MS(circuit: QuantumCircuit, name: str, target_qubits: list[int] | int, parameters: list | None,
 dimensions: list[int] | int, controls: ControlData | None = None)
 Bases: mqt.qudits.quantum_circuit.gate.Gate
 Unitary gate_matrix.
 __array__() → ndarray
 validate_parameter(parameter) → bool

 __str__() → str
 Return str(self).

class Perm(circuit: QuantumCircuit, name: str, target_qubits: list[int] | int, parameters: list, dimensions:
 list[int] | int, controls: ControlData | None = None)
 Bases: mqt.qudits.quantum_circuit.gate.Gate
 Unitary gate_matrix.
 __array__() → ndarray
 validate_parameter(parameter) → bool
 Verify that the input is a list of indices
 __str__() → str
 Return str(self).

class R(circuit: QuantumCircuit, name: str, target_qubits: list[int] | int, parameters: list | None, dimensions:
 list[int] | int, controls: ControlData | None = None)
 Bases: mqt.qudits.quantum_circuit.gate.Gate
 Unitary gate_matrix.
 property cost

```



```

__array__() → ndarray

levels_setter(la, lb)

validate_parameter(parameter) → bool

__str__() → str
 Return str(self).

class RandU(circuit: QuantumCircuit, name: str, target_qubits: list[int] | int, dimensions: list[int] | int,
 controls: ControlData | None = None)
 Bases: mqt.qudits.quantum_circuit.gate.Gate
 Unitary gate_matrix.
 __array__() → ndarray
 validate_parameter() → bool
 __str__() → str
 Return str(self).

class Rh(circuit: QuantumCircuit, name: str, target_qubits: list[int] | int, parameters: list | None,
 dimensions: list[int] | int, controls: ControlData | None = None)
 Bases: mqt.qudits.quantum_circuit.gate.Gate
 SU2 Hadamard
 __array__() → ndarray
 levels_setter(la, lb)
 validate_parameter(parameter) → bool
 __str__() → str
 Return str(self).

class Rz(circuit: QuantumCircuit, name: str, target_qubits: list[int] | int, parameters: list | None,
 dimensions: list[int] | int, controls: ControlData | None = None)
 Bases: mqt.qudits.quantum_circuit.gate.Gate
 Unitary gate_matrix.
 property cost
 __array__() → ndarray
 levels_setter(la, lb)
 validate_parameter(parameter) → bool
 __str__() → str
 Return str(self).

class S(circuit: QuantumCircuit, name: str, target_qubits: list[int] | int, dimensions: list[int] | int, controls:
 ControlData | None = None)
 Bases: mqt.qudits.quantum_circuit.gate.Gate
 Unitary gate_matrix.
 __array__() → ndarray
 validate_parameter(parameter=None) → bool

```

```

 __str__() → str
 Return str(self).

class VirtRz(circuit: QuantumCircuit, name: str, target_qubits: list[int] | int, parameters: list | None,
 dimensions: list[int] | int, controls: ControlData | None = None)

 Bases: mqt.qudits.quantum_circuit.gate.Gate

 Unitary gate_matrix.

 property cost

 __array__() → ndarray

 validate_parameter(parameter) → bool

 __str__() → str
 Return str(self).

class X(circuit: QuantumCircuit, name: str, target_qubits: list[int] | int, dimensions: list[int] | int, controls:
 ControlData | None = None)

 Bases: mqt.qudits.quantum_circuit.gate.Gate

 Unitary gate_matrix.

 __array__() → ndarray

 validate_parameter(parameter=None) → bool

 __str__() → str
 Return str(self).

class Z(circuit: QuantumCircuit, name: str, target_qubits: list[int] | int, dimensions: list[int] | int, controls:
 ControlData | None = None)

 Bases: mqt.qudits.quantum_circuit.gate.Gate

 Unitary gate_matrix.

 __array__() → ndarray

 validate_parameter(parameter=None) → bool

 __str__() → str
 Return str(self).

```

## Submodules

mqt.qudits.quantum\_circuit.circuit

Module Contents

**is\_not\_none\_or\_empty**(variable)

**add\_gate\_decorator**(func)

**class QuantumCircuit**(\*args)

property num\_qubits

property dimensions

property gate\_set: None

qasm\_to\_gate\_set\_dict

classmethod get\_qasm\_set()

**reset()** → None  
**copy()**  
**append**(qreg: QuantumRegister) → None  
**append\_classic**(creg: ClassicRegister) → None  
**csum**(qudits: list[int])  
**cu\_one**(qudits: int, parameters: ndarray, controls: ControlData | None = None)  
**cu\_two**(qudits: list[int], parameters: ndarray, controls: ControlData | None = None)  
**cu\_multi**(qudits: list[int], parameters: ndarray, controls: ControlData | None = None)  
**cx**(qudits: list[int], parameters: list | None = None)  
**gellmann**(qudit: int, parameters: list | None = None, controls: ControlData | None = None)  
**h**(qudit: int, controls: ControlData | None = None)  
**rh**(qudit: int, parameters: list, controls: ControlData | None = None)  
**ls**(qudits: list[int], parameters: list | None = None)  
**ms**(qudits: list[int], parameters: list | None = None)  
**pm**(qudits: list[int], parameters: list)  
**r**(qudit: int, parameters: list, controls: ControlData | None = None)  
**randu**(qudits: list[int])  
**rz**(qudit: int, parameters: list, controls: ControlData | None = None)  
**virtrz**(qudit: int, parameters: list, controls: ControlData | None = None)  
**s**(qudit: int, controls: ControlData | None = None)  
**x**(qudit: int, controls: ControlData | None = None)  
**z**(qudit: int, controls: ControlData | None = None)  
**replace\_gate**(gate\_index: int, sequence: list[Gate]) → None  
**set\_instructions**(sequence: list[Gate])  
**from\_qasm**(qasm\_prog) → None  
 Create a circuit from qasm text  
**to\_qasm**()  
**save\_to\_file**(file\_name: str, file\_path: str = '.') → str  
 Save qasm into a file with the specified name and path.  
 Parameters
 

- **text** (str) – The text to be saved into the file.
- **file\_name** (str) – The name of the file.
- **file\_path** (str, optional) – The path where the file will be saved. Defaults to “.” (current directory).

 Returns The full path of the saved file.  
 Return type

**load\_from\_file**(*file\_path: str*) → None

Load text from a file.

Parameter **file\_path** (*str*) – The path of the file to load.

Returns The text loaded from the file.

Return type

**draw**() → None

mqt.qudits.quantum\_circuit.gate

Module Contents

**class Instruction**(*name: str*)

Bases: [abc.ABC](#)

Helper class that provides a standard way to create an ABC using inheritance.

**class Gate**(*circuit: QuantumCircuit, name: str, gate\_type: enum, target\_qudits: list[int] | int, dimensions: list[int] | int, params: list | ndarray | None = None, control\_set=None, label: str | None = None, duration=None, unit='dt')*)

Bases: [Instruction](#)

Unitary gate\_matrix.

**property reference\_lines**

**property get\_control\_lines**

**property control\_info**

**abstract \_\_array\_\_**() → ndarray

**dag**()

**to\_matrix**(*identities=0*) → ndarray

Return a np.ndarray for the gate\_matrix unitary parameters.

Returns if the Gate subclass has a parameters definition.

Return type ndarray

Raises [CircuitError](#) – If a Gate subclass does not implement this method an exception will be raised when this base class method is called.

**control**(*indices: list[int] | int, ctrl\_states: list[int] | int*)

**abstract validate\_parameter**(*parameter*)

**\_\_qasm\_\_**() → str

Generate QASM for Gate export

**abstract \_\_str\_\_**() → str

Return str(self).

**check\_long\_range**()

**set\_gate\_type\_single**() → None

**set\_gate\_type\_two**() → None

**set\_gate\_type\_multi**() → None

**return\_custom\_data**() → str

mqt.qudits.quantum\_circuit.qasm

Module Contents

## class QASM

Class that manages the parsing of QASM programs

**parse\_nonspecial\_lines**(*line*, *rgxs*, *in\_comment\_flag*)

**parse\_qreg**(*line*, *rgxs*, *sitemap*) → bool

**parse\_creg**(*line*, *rgxs*, *sitemap\_classic*) → bool

**safe\_eval\_math\_expression**(*expression*)

**parse\_gate**(*line*, *rgxs*, *sitemap*, *gates*) → bool

**parse\_ignore**(*line*, *rgxs*, *warned*) → bool

**parse\_ditqasm2\_str**(*contents*)

Parse the string contents of an OpenQASM 2.0 file. This parser only supports basic gate\_matrix definitions, and is not guaranteed to check the full openqasm grammar.

**parse\_ditqasm2\_file**(*fname*)

Parse an OpenQASM 2.0 file.

## Package Contents

### class QuantumCircuit(\*args)

property num\_qubits

property dimensions

property gate\_set: None

qasm\_to\_gate\_set\_dict

classmethod get\_qasm\_set()

reset() → None

copy()

append(*qreg*: QuantumRegister) → None

append\_classic(*creg*: ClassicRegister) → None

csum(*qubits*: list[int])

cu\_one(*qubits*: int, *parameters*: ndarray, *controls*: ControlData | None = None)

cu\_two(*qubits*: list[int], *parameters*: ndarray, *controls*: ControlData | None = None)

cu\_multi(*qubits*: list[int], *parameters*: ndarray, *controls*: ControlData | None = None)

cx(*qubits*: list[int], *parameters*: list | None = None)

gellmann(*qubit*: int, *parameters*: list | None = None, *controls*: ControlData | None = None)

h(*qubit*: int, *controls*: ControlData | None = None)

rh(*qubit*: int, *parameters*: list, *controls*: ControlData | None = None)

ls(*qubits*: list[int], *parameters*: list | None = None)

ms(*qubits*: list[int], *parameters*: list | None = None)

pm(*qubits*: list[int], *parameters*: list)

```

r(qudit: int, parameters: list, controls: ControlData | None = None)

randu(qudits: list[int])

rz(qudit: int, parameters: list, controls: ControlData | None = None)

virtrz(qudit: int, parameters: list, controls: ControlData | None = None)

s(qudit: int, controls: ControlData | None = None)

x(qudit: int, controls: ControlData | None = None)

z(qudit: int, controls: ControlData | None = None)

replace_gate(gate_index: int, sequence: list[Gate]) → None

set_instructions(sequence: list[Gate])

from_qasm(qasm_prog) → None
 Create a circuit from qasm text

to_qasm()

save_to_file(file_name: str, file_path: str = '.') → str
 Save qasm into a file with the specified name and path.

 Parameters
 • text (str) – The text to be saved into the file.
 • file_name (str) – The name of the file.
 • file_path (str, optional) – The path where the file will be saved. Defaults to “.”
 (current directory).

 Returns The full path of the saved file.
 Return type

load_from_file(file_path: str) → None
 Load text from a file.

 Parameter file_path (str) – The path of the file to load.
 Returns The text loaded from the file.
 Return type

draw() → None

class QuantumRegister(name, size, dims=None)

 classmethod from_map(sitemap: dict) → list[QuantumRegister]

 __qasm__()

 __getitem__(key)

class QASM
 Class that manages the parsing of QASM programs

 parse_nonspecial_lines(line, rgxs, in_comment_flag)

 parse_qreg(line, rgxs, sitemap) → bool

 parse_creg(line, rgxs, sitemap_classic) → bool

 safe_eval_math_expression(expression)

 parse_gate(line, rgxs, sitemap, gates) → bool

```

**parse\_ignore**(*line*, *rgxs*, *warned*) → bool

**parse\_ditqasm2\_str**(*contents*)

Parse the string contents of an OpenQASM 2.0 file. This parser only supports basic gate\_matrix definitions, and is not guaranteed to check the full openqasm grammar.

**parse\_ditqasm2\_file**(*fname*)

Parse an OpenQASM 2.0 file.

**mqt.qudits.simulation**

**Subpackages**

mqt.qudits.simulation.backends

Subpackages

mqt.qudits.simulation.backends.fake\_backends

Submodules

mqt.qudits.simulation.backends.fake\_backends.fake\_traps2six

Module Contents

**class FakeIonTraps2Six**(*provider*: mqt.qudits.simulation.qudit\_provider.QuditProvider | None = None, *name*: str | None = None, *description*: str | None = None, *online\_date*: datetime | None = None, *backend\_version*: str | None = None, *\*\*fields*)

Bases: mqt.qudits.simulation.backends.tnsim.TNSim

Helper class that provides a standard way to create an ABC using inheritance.

**property version**: int

**property energy\_level\_graphs**: list[LevelGraph, LevelGraph]

mqt.qudits.simulation.backends.fake\_backends.fake\_traps2three

Module Contents

**class FakeIonTraps2Trits**(*provider*: mqt.qudits.simulation.qudit\_provider.QuditProvider | None = None, *name*: str | None = None, *description*: str | None = None, *online\_date*: datetime | None = None, *backend\_version*: str | None = None, *\*\*fields*)

Bases: mqt.qudits.simulation.backends.tnsim.TNSim

Helper class that provides a standard way to create an ABC using inheritance.

**property version**: int

**property energy\_level\_graphs**: list[LevelGraph, LevelGraph]

Package Contents

**class FakeIonTraps2Six**(*provider*: mqt.qudits.simulation.qudit\_provider.QuditProvider | None = None, *name*: str | None = None, *description*: str | None = None, *online\_date*: datetime | None = None, *backend\_version*: str | None = None, *\*\*fields*)

Bases: mqt.qudits.simulation.backends.tnsim.TNSim

Helper class that provides a standard way to create an ABC using inheritance.

**property version**: int

**property energy\_level\_graphs**: list[LevelGraph, LevelGraph]

```
class FakeIonTraps2Trits(provider: mqt.qudits.simulation.qudit_provider.QuditProvider | None = None,
 name: str | None = None, description: str | None = None, online_date:
 datetime | None = None, backend_version: str | None = None, **fields)
```

Bases: `mqt.qudits.simulation.backends.tnsim.TNSim`

Helper class that provides a standard way to create an ABC using inheritance.

**property** version: `int`

**property** energy\_level\_graphs: `list[LevelGraph, LevelGraph]`

Submodules

`mqt.qudits.simulation.backends.backendv2`

Module Contents

```
class Backend(provider: mqt.qudits.simulation.qudit_provider.QuditProvider | None = None, name: str |
 None = None, description: str | None = None, online_date: datetime | None = None,
 backend_version: str | None = None, **fields: Any)
```

Bases: `abc.ABC`

Helper class that provides a standard way to create an ABC using inheritance.

**property** version: `int`

**property** instructions: `list[tuple[Gate, tuple[int]]]`

**property** operations: `list[Gate]`

**property** operation\_names: `list[str]`

**property** num\_qudits: `int`

**abstract property** energy\_level\_graphs: `list[LevelGraph, LevelGraph]`

**property** options

**property** provider

**target**

**set\_options**(\*\*fields) → `None`

**abstract run**(run\_input, \*\*options) → `Job`

`mqt.qudits.simulation.backends.misim`

Module Contents

```
class MISim(**fields)
```

Bases: `mqt.qudits.simulation.backends.backendv2.Backend`

Helper class that provides a standard way to create an ABC using inheritance.

**run**(circuit: `QuantumCircuit`, \*\*options) → `Job`

**execute**(circuit: `QuantumCircuit`, noise\_model: `NoiseModel` | *None* = *None*) → `ndarray`

`mqt.qudits.simulation.backends.stochastic_sim`

Module Contents

**stochastic\_simulation**(backend: `Backend`, circuit: `QuantumCircuit`)

**stochastic\_execution**(args)



**stochastic\_simulation\_misim**(*backend*: Backend, *circuit*: QuantumCircuit)

**stochastic\_execution\_misim**(*args*)

mqt.qudits.simulation.backends.tnsim

Module Contents

**class** TNSim(\*\**fields*)

Bases: [mqt.qudits.simulation.backends.backendv2.Backend](#)

Helper class that provides a standard way to create an ABC using inheritance.

**run**(*circuit*: QuantumCircuit, \*\**options*)

**execute**(*circuit*: QuantumCircuit)

Package Contents

**class** MISim(\*\**fields*)

Bases: [mqt.qudits.simulation.backends.backendv2.Backend](#)

Helper class that provides a standard way to create an ABC using inheritance.

**run**(*circuit*: QuantumCircuit, \*\**options*) → Job

**execute**(*circuit*: QuantumCircuit, *noise\_model*: NoiseModel | None = None) → ndarray

**class** TNSim(\*\**fields*)

Bases: [mqt.qudits.simulation.backends.backendv2.Backend](#)

Helper class that provides a standard way to create an ABC using inheritance.

**run**(*circuit*: QuantumCircuit, \*\**options*)

**execute**(*circuit*: QuantumCircuit)

mqt.qudits.simulation.jobs

Submodules

mqt.qudits.simulation.jobs.job

Module Contents

**class** Job(*backend*: Backend | None, *job\_id*: str = 'auto', \*\**kwargs*)

Class to handle jobs

This first version of the Backend abstract class is written to be mostly backwards compatible with the legacy providers interface. This was done to ease the transition for users and provider maintainers to the new versioned providers. Expect future versions of this abstract class to change the data model and interface.

**version** = 1

**job\_id**() → str

Return a unique id identifying the job.

**backend**() → Backend

Return the backend where this job was executed.

**done**() → bool

Return whether the job has successfully run.

**running**() → bool

Return whether the job is actively running.

**cancelled()** → bool

Return whether the job has been cancelled.

**in\_final\_state()** → bool

Return whether the job is in a final job state such as DONE or ERROR.

**wait\_for\_final\_state**(*timeout: float | None = None, wait: float = 5, callback: Callable | None = None*) → None

Poll the job status until it progresses to a final state such as DONE or ERROR.

Parameters

- **timeout** – Seconds to wait for the job. If None, wait indefinitely.
- **wait** – Seconds between queries.
- **callback** – Callback function invoked after each query.

Raises **JobTimeoutError** – If the job does not reach a final state before the specified timeout.

**abstract submit()** → NoReturn

Submit the job to the backend for execution.

**result()**

Return the results of the job.

**set\_result**(*result*) → None

**abstract cancel()** → NoReturn

Attempt to cancel the job.

**abstract status()** → str

Return the status of the job, among the values of BackendStatus.

mqt.qudits.simulation.jobs.job\_result

Module Contents

**class JobResult**(*state\_vector: list[numpy.complex128], counts: list[int]*)

**get\_counts()** → list[int]

**get\_state\_vector()** → list[numpy.complex128]

mqt.qudits.simulation.jobs.jobstatus

Module Contents

**class JobStatus**(\*args, \*\*kwargs)

Bases: **enum.Enum**

Enumeration for job status.

**INITIALIZING** = 'Initializing: Job is being initialized'

**QUEUED** = 'Queued: Job is waiting in the queue'

**VALIDATING** = 'Validating: Job is being validated'

**RUNNING** = 'Running: Job is actively running'

**CANCELLED** = 'Cancelled: Job has been cancelled'

**DONE** = 'Done: Job has successfully run'

**ERROR** = 'Error: Job incurred an error'

**JOB\_FINAL\_STATES** = ()

Package Contents

```
class Job(backend: Backend | None, job_id: str = 'auto', **kwargs)
```

Class to handle jobs

This first version of the Backend abstract class is written to be mostly backwards compatible with the legacy providers interface. This was done to ease the transition for users and provider maintainers to the new versioned providers. Expect future versions of this abstract class to change the data model and interface.

```
version = 1
```

```
job_id() → str
```

Return a unique id identifying the job.

```
backend() → Backend
```

Return the backend where this job was executed.

```
done() → bool
```

Return whether the job has successfully run.

```
running() → bool
```

Return whether the job is actively running.

```
cancelled() → bool
```

Return whether the job has been cancelled.

```
in_final_state() → bool
```

Return whether the job is in a final job state such as DONE or ERROR.

```
wait_for_final_state(timeout: float | None = None, wait: float = 5, callback: Callable | None = None) → None
```

Poll the job status until it progresses to a final state such as DONE or ERROR.

Parameters

- **timeout** – Seconds to wait for the job. If None, wait indefinitely.
- **wait** – Seconds between queries.
- **callback** – Callback function invoked after each query.

Raises **JobTimeoutError** – If the job does not reach a final state before the specified timeout.

```
abstract submit() → NoReturn
```

Submit the job to the backend for execution.

```
result()
```

Return the results of the job.

```
set_result(result) → None
```

```
abstract cancel() → NoReturn
```

Attempt to cancel the job.

```
abstract status() → str
```

Return the status of the job, among the values of BackendStatus.

```
class JobResult(state_vector: list[numpy.complex128], counts: list[int])
```

```
get_counts() → list[int]
```

```
get_state_vector() → list[numpy.complex128]
```

```
class JobStatus(*args, **kws)
```

Bases: `enum.Enum`

Enumeration for job status.

```
INITIALIZING = 'Initializing: Job is being initialized'
```

```

QUEUED = 'Queued: Job is waiting in the queue'
VALIDATING = 'Validating: Job is being validated'
RUNNING = 'Running: Job is actively running'
CANCELLED = 'Cancelled: Job has been cancelled'
DONE = 'Done: Job has successfully run'
ERROR = 'Error: Job incurred an error'

mqt.qudits.simulation.noise_tools
Submodules
mqt.qudits.simulation.noise_tools.noise
Module Contents
class Noise
 probability_depolarizing: float
 probability_dephasing: float
class NoiseModel
 property basis_gates
 add_recurrent_quantum_error_locally(noise, gates, qudits) → None
 add_quantum_error_locally(noise, gates) → None
 add_all_qudit_quantum_error(noise, gates) → None
 add_nonlocal_quantum_error(noise, gates) → None
 add_nonlocal_quantum_error_on_target(noise, gates) → None
 add_nonlocal_quantum_error_on_control(noise, gates) → None
 __str__() → str
 Return str(self).
mqt.qudits.simulation.noise_tools.noisy_circuit_factory
Module Contents
class NoisyCircuitFactory(noise_model: NoiseModel, circuit: QuantumCircuit)
 generate_circuit()
Package Contents
class Noise
 probability_depolarizing: float
 probability_dephasing: float
class NoiseModel
 property basis_gates
 add_recurrent_quantum_error_locally(noise, gates, qudits) → None

```

```

add_quantum_error_locally(noise, gates) → None
add_all_qudit_quantum_error(noise, gates) → None
add_nonlocal_quantum_error(noise, gates) → None
add_nonlocal_quantum_error_on_target(noise, gates) → None
add_nonlocal_quantum_error_on_control(noise, gates) → None

__str__() → str
 Return str(self).

```

```

class NoisyCircuitFactory(noise_model: NoiseModel, circuit: QuantumCircuit)

 generate_circuit()

```

## Submodules

mqt.qudits.simulation.qudit\_provider

Module Contents

```

class MQTQuditProvider

 property version: int

 get_backend(name: str | None = None, **kwargs) → Backend
 Return a single backend matching the specified filtering.

 backends(name: str | None = None, **kwargs) → list[Backend]
 Return a list of backends matching the specified filtering.

 __eq__(other: object) → bool
 Return self==value.

 __hash__() → int
 Return hash(self).

```

mqt.qudits.simulation.save\_info

Module Contents

```

save_full_states(list_of_vectors, file_path=None, file_name=None) → None

save_shots(shots, file_path=None, file_name=None) → None

```

## Package Contents

```

class MQTQuditProvider

 property version: int

 get_backend(name: str | None = None, **kwargs) → Backend
 Return a single backend matching the specified filtering.

 backends(name: str | None = None, **kwargs) → list[Backend]
 Return a list of backends matching the specified filtering.

 __eq__(other: object) → bool
 Return self==value.

 __hash__() → int
 Return hash(self).

```

## mqt.qudits.visualisation

### Submodules

mqt.qudits.visualisation.drawing\_routines

Module Contents

**draw\_qudit\_local**(circuit: QuantumCircuit) → None

mqt.qudits.visualisation.mini\_quantum\_information

Module Contents

**get\_density\_matrix\_from\_counts**(results, circuit)

**partial\_trace**(rho, qudits2keep, dims, optimize=False)

Calculate the partial trace

p\_a = Tr\_b(p)

Parameters

p [2D array] Matrix to trace

qudits2keep [array] An array of indices of the spaces to keep after being traced. For instance, if the space is A x B x C x D and we want to trace out B and D, keep = [0,2]

dims [array] An array of the dimensions of each space. For instance, if the space is A x B x C x D, dims = [dim\_A, dim\_B, dim\_C, dim\_D]

Returns

p\_a [2D array] Traced matrix

mqt.qudits.visualisation.plot\_information

Module Contents

**class HistogramWithErrors**(labels, counts, errors, title='Simulation')

**generate\_histogram**() → None

**save\_to\_png**(filename) → None

**state\_labels**(circuit)

**plot\_state**(result: ndarray, circuit: QuantumCircuit, errors=None) → None

**plot\_counts**(result, circuit: QuantumCircuit) → None

### Package Contents

**draw\_qudit\_local**(circuit: QuantumCircuit) → None

**get\_density\_matrix\_from\_counts**(results, circuit)

**partial\_trace**(rho, qudits2keep, dims, optimize=False)

Calculate the partial trace

p\_a = Tr\_b(p)

Parameters

p [2D array] Matrix to trace

qudits2keep [array] An array of indices of the spaces to keep after being traced. For instance, if the space is A x B x C x D and we want to trace out B and D, keep = [0,2]

dims [array] An array of the dimensions of each space. For instance, if the space is A x B x C x D, dims = [dim\_A, dim\_B, dim\_C, dim\_D]

Returns

p\_a [2D array] Traced matrix

plot\_counts(result, circuit: QuantumCircuit) → None

plot\_state(result: ndarray, circuit: QuantumCircuit, errors=None) → None

## III-B Package Contents

\_\_version\_\_: str

version\_info: tuple[int, int, int, str, str] | tuple[int, int, int]

## References

- [1] Kevin Mato, Stefan Hillmich, and Robert Wille. Mixed-dimensional qudit state preparation using edge-weighted decision diagrams. In *Design Automation Conference (DAC)*. 2024.
- [2] Kevin Mato, Stefan Hillmich, and Robert Wille. Mixed-dimensional quantum circuit simulation with decision diagrams. In *International Conference on Quantum Computing and Engineering (QCE)*. 2023. doi:10.1109/QCE57702.2023.00112.
- [3] Kevin Mato, Stefan Hillmich, and Robert Wille. Compression of qubit circuits: Mapping to mixed-dimensional quantum systems. In *International Conference on Quantum Software (QSW)*. 2023. doi:10.1109/QSW59989.2023.00027.
- [4] Kevin Mato, Martin Ringbauer, Stefan Hillmich, and Robert Wille. Compilation of entangling gates for high-dimensional quantum systems. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2023. doi:10.1145/3566097.3567930.
- [5] Kevin Mato, Martin Ringbauer, Stefan Hillmich, and Robert Wille. Adaptive compilation of multi-level quantum operations. In *International Conference on Quantum Computing and Engineering (QCE)*. 2022. doi:10.1109/QCE53715.2022.00070.

## Index

### Symbols

`__array__()` (*CEx method*), 34, 39  
`__array__()` (*CSum method*), 33, 38  
`__array__()` (*CustomMulti method*), 34, 39  
`__array__()` (*CustomOne method*), 34, 39  
`__array__()` (*CustomTwo method*), 34, 39  
`__array__()` (*Gate method*), 44  
`__array__()` (*GellMann method*), 35, 39  
`__array__()` (*H method*), 35, 40  
`__array__()` (*LS method*), 35, 40  
`__array__()` (*MS method*), 35, 40  
`__array__()` (*Perm method*), 36, 40  
`__array__()` (*R method*), 36, 40  
`__array__()` (*RandU method*), 36, 41  
`__array__()` (*Rh method*), 36, 41  
`__array__()` (*Rz method*), 37, 41  
`__array__()` (*S method*), 37, 41  
`__array__()` (*VirtRz method*), 37, 42  
`__array__()` (*X method*), 38, 42  
`__array__()` (*Z method*), 38, 42  
`__eq__()` (*MQTQuditProvider method*), 53  
`__getitem__()` (*ClassicRegister method*), 33  
`__getitem__()` (*QuantumRegister method*), 33, 46  
`__hash__()` (*MQTQuditProvider method*), 53  
`__qasm__()` (*ClassicRegister method*), 33  
`__qasm__()` (*Gate method*), 44  
`__qasm__()` (*QuantumRegister method*), 33, 46  
`__str__()` (*CEx method*), 34, 39  
`__str__()` (*CSum method*), 33, 38  
`__str__()` (*CustomMulti method*), 34, 39  
`__str__()` (*CustomOne method*), 34, 39  
`__str__()` (*CustomTwo method*), 34, 39  
`__str__()` (*Gate method*), 44  
`__str__()` (*GellMann method*), 35, 39  
`__str__()` (*H method*), 35, 40  
`__str__()` (*LS method*), 35, 40  
`__str__()` (*LevelGraph method*), 25, 30  
`__str__()` (*MS method*), 35, 40  
`__str__()` (*NAryTree method*), 21, 26  
`__str__()` (*Node method*), 21, 26  
`__str__()` (*NodeNotFoundException method*), 30, 31  
`__str__()` (*NoiseModel method*), 52, 53  
`__str__()` (*Perm method*), 36, 40  
`__str__()` (*R method*), 36, 41  
`__str__()` (*RandU method*), 36, 41  
`__str__()` (*Rh method*), 37, 41  
`__str__()` (*RoutingException method*), 31  
`__str__()` (*Rz method*), 37, 41  
`__str__()` (*S method*), 37, 41  
`__str__()` (*SequenceFoundException method*), 31, 32  
`__str__()` (*VirtRz method*), 37, 42  
`__str__()` (*X method*), 38, 42  
`__str__()` (*Z method*), 38, 42  
`__version__` (in module *mqt.qudits*), 55

### A

`add()` (*NAryTree method*), 21, 26  
`add()` (*Node method*), 21, 26  
`add_all_qudit_quantum_error()` (*NoiseModel method*), 52, 53  
`add_gate_decorator()` (in module *mqt.qudits.quantum\_circuit.circuit*), 42  
`add_nonlocal_quantum_error()` (*NoiseModel method*), 52, 53  
`add_nonlocal_quantum_error_on_control()` (*NoiseModel method*), 52, 53  
`add_nonlocal_quantum_error_on_target()` (*NoiseModel method*), 52, 53  
`add_quantum_error_locally()` (*NoiseModel method*), 52  
`add_recurrent_quantum_error_locally()` (*NoiseModel method*), 52  
`ansatz_decompose()` (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz\_instance*), 18

`append()` (*QuantumCircuit method*), 43, 45  
`append_classic()` (*QuantumCircuit method*), 43, 45  
`apply_gate_to_tlines()` (in module *mqt.qudits.compiler.compilation\_minitools*), 12  
`apply_gate_to_tlines()` (in module *mqt.qudits.compiler.compilation\_minitools.numerical\_ansatz\_utils*), 11  
`apply_identities_and_controls()` (*MatrixFactory class method*), 32

### B

`Backend` (class in *mqt.qudits.simulation.backends.backendv2*), 48  
`backend()` (*Job method*), 49, 51  
`BackendNotFoundError`, 30, 31  
`backends()` (*MQTQuditProvider method*), 53  
`basis_gates` (*NoiseModel property*), 52  
`binary_search_compile()` (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz\_solve\_n\_sea*), 20  
`bound_1` (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz\_parametrize*), 18  
`bound_2` (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz\_parametrize*), 18  
`bound_3` (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz\_parametrize*), 18  
`bounds_assigner()` (*Optimizer static method*), 19, 20

### C

`calculate_q0_q1()` (in module *mqt.qudits.quantum\_circuit.components.extensions.matrix\_factory*), 33  
`cancel()` (*Job method*), 50, 51  
`CANCELLED` (*JobStatus attribute*), 50, 52  
`canceled()` (*Job method*), 49, 51  
`CEx` (class in *mqt.qudits.quantum\_circuit.gates*), 39  
`CEx` (class in *mqt.qudits.quantum\_circuit.gates.cx*), 34  
`CEX_SEQUENCE` (in module *mqt.qudits.compiler.twodit.entanglement\_qr.crot*), 16  
`check_long_range()` (*Gate method*), 44  
`CircuitError`, 30, 31  
`ClassicRegister` (class in *mqt.qudits.quantum\_circuit.components*), 33  
`ClassicRegister` (class in *mqt.qudits.quantum\_circuit.components.classic\_register*), 33  
`compile()` (*QuditCompiler method*), 21  
`CompilerPass` (class in *mqt.qudits.compiler*), 21  
`CompilerPass` (class in *mqt.qudits.compiler.compiler\_pass*), 20  
`control()` (*Gate method*), 44  
`control_info` (*Gate property*), 44  
`ControlData` (class in *mqt.qudits.quantum\_circuit.components.extensions.controls*), 32  
`ControlData` (class in *mqt.qudits.quantum\_circuit.gates*), 38  
`copy()` (*QuantumCircuit method*), 43, 45  
`CORE_GATE_TYPES` (in module *mqt.qudits.quantum\_circuit.components.extensions.gate\_types*), 32  
`cost` (*R property*), 36, 40  
`cost` (*Rz property*), 37, 41  
`cost` (*VirtRz property*), 37, 42  
`cost_calculator()` (in module *mqt.qudits.compiler.onedit.local\_operation\_swap*), 12  
`cost_calculator()` (in module *mqt.qudits.compiler.onedit.local\_operation\_swap.swap\_routine*), 12  
`create_cu_instance()` (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz*), 18



**create\_cu\_instance()** (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz.instantiate*), 18  
**create\_ls\_instance()** (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz.instantiate*), 18  
**create\_ls\_instance()** (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz.instantiate*), 18  
**create\_ms\_instance()** (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz.instantiate*), 18  
**create\_ms\_instance()** (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz.instantiate*), 18  
**crot\_101\_as\_list()** (CRotGen method), 17  
**CRotGen** (class in *mqt.qudits.compiler.twodit.entanglement\_qr*), 17  
**CRotGen** (class in *mqt.qudits.compiler.twodit.entanglement\_qr.crot*), 16  
**Csum** (class in *mqt.qudits.quantum\_circuit.gates*), 38  
**Csum** (class in *mqt.qudits.quantum\_circuit.gates.csum*), 33  
**csum()** (QuantumCircuit method), 43, 45  
**ctrl\_states** (ControlData attribute), 32, 38  
**cu\_ansatz()** (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz*), 18  
**cu\_ansatz()** (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz.ansatz*), 18  
**cu\_multi()** (QuantumCircuit method), 43, 45  
**cu\_one()** (QuantumCircuit method), 43, 45  
**cu\_two()** (QuantumCircuit method), 43, 45  
**CUSTOM\_PRIMITIVE** (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz.prim*), 18  
**CustomMulti** (class in *mqt.qudits.quantum\_circuit.gates*), 39  
**CustomMulti** (class in *mqt.qudits.quantum\_circuit.gates.custom\_multi*), 34  
**CustomOne** (class in *mqt.qudits.quantum\_circuit.gates*), 39  
**CustomOne** (class in *mqt.qudits.quantum\_circuit.gates.custom\_one*), 34  
**CustomTwo** (class in *mqt.qudits.quantum\_circuit.gates*), 39  
**CustomTwo** (class in *mqt.qudits.quantum\_circuit.gates.custom\_two*), 34  
**cx()** (QuantumCircuit method), 43, 45

## D

**dag()** (Gate method), 44  
**deep\_copy\_func()** (LevelGraph method), 25, 29  
**define\_states()** (LevelGraph method), 25, 29  
**density\_operator()** (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.opt*), 19  
**density\_operator()** (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.opt.distance\_measures*), 19  
**depth()** (NaryTree method), 21, 26  
**DFS()** (LogAdaptiveDecomposition method), 15  
**DFS()** (PhyAdaptiveDecomposition method), 14  
**dimensions** (QuantumCircuit property), 42, 45  
**distance\_nodes()** (LevelGraph method), 25, 29  
**distance\_nodes\_pi\_pulses\_fixed\_ancilla()** (LevelGraph method), 25, 29  
**DONE** (JobStatus attribute), 50, 52  
**done()** (Job method), 49, 51  
**draw()** (QuantumCircuit method), 44, 46  
**draw\_qudit\_local()** (in module *mqt.qudits.visualisation*), 54  
**draw\_qudit\_local()** (in module *mqt.qudits.visualisation.drawing\_routines*), 54

## E

**energy\_level\_graphs** (Backend property), 48  
**energy\_level\_graphs** (FakeIonTraps2Six property), 47  
**energy\_level\_graphs** (FakeIonTraps2Trits property), 47, 48  
**EntangledQRCEX** (class in *mqt.qudits.compiler.twodit.entanglement\_qr*), 17  
**EntangledQRCEX** (class in *mqt.qudits.compiler.twodit.entanglement\_qr.log\_ent\_qr\_cex\_decomp*), 17  
**ERROR** (JobStatus attribute), 50, 52  
**execute()** (EntangledQRCEX method), 17  
**execute()** (LogAdaptiveDecomposition method), 15  
**execute()** (MISim method), 48, 49  
**execute()** (PhyAdaptiveDecomposition method), 14  
**execute()** (PhyQrDecomp method), 14  
**execute()** (QrDecomp method), 15  
**execute()** (TNSim method), 49

## F

**FakeIonTraps2Six** (class in *mqt.qudits.simulation.backends.fake\_backends*), 47  
**FakeIonTraps2Six** (class in *mqt.qudits.simulation.backends.fake\_backends.fake\_traps2six*), 47  
**FakeIonTraps2Trits** (class in *mqt.qudits.simulation.backends.fake\_backends*), 47  
**FakeIonTraps2Trits** (class in *mqt.qudits.simulation.backends.fake\_backends.fake\_traps2three*), 47  
**fidelity\_on\_density\_operator()** (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.opt*), 19  
**fidelity\_on\_density\_operator()** (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.opt.distance\_measures*), 19  
**fidelity\_on\_operator()** (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.opt*), 19  
**fidelity\_on\_operator()** (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.opt.distance\_measures*), 19  
**fidelity\_on\_unitares()** (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.opt*), 19  
**fidelity\_on\_unitares()** (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.opt.distance\_measures*), 19  
**FidelityReachException**, 31  
**find\_intervals\_with\_same\_target\_qudits()** (ZPropagationPass method), 13, 16  
**find\_logic\_from\_phys()** (in module *mqt.qudits.compiler.onedit.local\_operation\_swap.swap\_routine*), 12  
**find\_node()** (NaryTree method), 21, 26  
**found\_checker()** (NaryTree method), 21, 26  
**frobenius\_dist()** (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.opt*), 19  
**frobenius\_dist()** (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.opt.distance\_measures*), 19  
**from\_dirac\_to\_basis()** (in module *mqt.qudits.quantum\_circuit.components.extensions.matrix\_factory*), 32  
**from\_map()** (ClassicRegister class method), 33  
**from\_map()** (QuantumRegister class method), 33, 46  
**from\_qasm()** (QuantumCircuit method), 43, 46  
**FUN\_SOLUTION** (Optimizer attribute), 19, 20

## G

**Gate** (class in *mqt.qudits.quantum\_circuit.gate*), 44  
**gate\_chain\_condition()** (in module *mqt.qudits.compiler.onedit.local\_operation\_swap*), 12  
**gate\_chain\_condition()** (in module *mqt.qudits.compiler.onedit.local\_operation\_swap.swap\_routine*), 12  
**gate\_expand\_to\_circuit()** (in module *mqt.qudits.compiler.compilation\_minitools*), 12  
**gate\_expand\_to\_circuit()** (in module *mqt.qudits.compiler.compilation\_minitools.numerical\_ansatz\_utils*), 11  
**gate\_set** (QuantumCircuit property), 42, 45

GateTypes (class in *mqt.qudits.quantum\_circuit.components.extensions.gate\_types*), 32  
 JobTimeoutError (class in *mqt.qudits.exceptions*), 32  
 JobTimeoutError (class in *mqt.qudits.exceptions.joberror*), 31

GateTypes (class in *mqt.qudits.quantum\_circuit.gates*), 38  
 GellMann (class in *mqt.qudits.quantum\_circuit.gates*), 39  
 GellMann (class in *mqt.qudits.quantum\_circuit.gates.gellmann*), 35  
 gellmann() (*QuantumCircuit* method), 43, 45  
 generate\_circuit() (*NoisyCircuitFactory* method), 52, 53  
 generate\_histogram() (*HistogramWithErrors* method), 54  
 generate\_matrix() (*MatrixFactory* method), 32  
 generic\_sud() (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz.parameters*), 18

get\_backend() (*MQTQuditProvider* method), 53  
 get\_control\_lines (Gate property), 44  
 get\_counts() (*JobResult* method), 50, 51  
 get\_density\_matrix\_from\_counts() (in module *mqt.qudits.visualisation*), 54  
 get\_density\_matrix\_from\_counts() (in module *mqt.qudits.visualisation.mini\_quantum\_information*), 54  
 get\_edge\_sensitivity() (*LevelGraph* method), 25, 30  
 get\_node\_sensitivity\_cost() (*LevelGraph* method), 25, 30  
 get\_perm\_matrix() (*UnitaryVerifier* method), 11, 12  
 get\_qasm\_set() (*QuantumCircuit* class method), 42, 45  
 get\_state\_vector() (*JobResult* method), 50, 51  
 get\_VRz\_gates() (*LevelGraph* method), 25, 30  
 graph\_rule\_ongate() (in module *mqt.qudits.compiler.onedit.local\_operation\_swap*), 12  
 graph\_rule\_ongate() (in module *mqt.qudits.compiler.onedit.local\_operation\_swap.swap\_routine*), 12  
 graph\_rule\_update() (in module *mqt.qudits.compiler.onedit.local\_operation\_swap*), 12  
 graph\_rule\_update() (in module *mqt.qudits.compiler.onedit.local\_operation\_swap.swap\_routine*), 12

## H

H (class in *mqt.qudits.quantum\_circuit.gates*), 40  
 H (class in *mqt.qudits.quantum\_circuit.gates.h*), 35  
 h() (*QuantumCircuit* method), 43, 45  
 HistogramWithErrors (class in *mqt.qudits.visualisation.plot\_information*), 54

## I

in\_final\_state() (*Job* method), 50, 51  
 index() (*LevelGraph* method), 25, 29  
 indices (*ControlData* attribute), 32, 38  
 INITIALIZING (*JobStatus* attribute), 50, 51  
 insert\_at() (in module *mqt.qudits.quantum\_circuit.components.extensions.matrix\_factory*), 33  
 Instruction (class in *mqt.qudits.quantum\_circuit.gate*), 44  
 instructions (Backend property), 48  
 interrupt\_function() (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz.parameters*), 20  
 is\_empty() (*NaryTree* method), 21, 26  
 is\_Inode() (*LevelGraph* method), 25, 30  
 is\_irnode() (*LevelGraph* method), 25, 30  
 is\_not\_none\_or\_empty() (in module *mqt.qudits.quantum\_circuit.circuit*), 42

## J

Job (class in *mqt.qudits.simulation.jobs*), 50  
 Job (class in *mqt.qudits.simulation.jobs.job*), 49  
 JOB\_FINAL\_STATES (in module *mqt.qudits.simulation.jobs.jobstatus*), 50  
 job\_id() (*Job* method), 49, 51  
 JobError, 31, 32  
 JobResult (class in *mqt.qudits.simulation.jobs*), 51  
 JobResult (class in *mqt.qudits.simulation.jobs.job\_result*), 50  
 JobStatus (class in *mqt.qudits.simulation.jobs*), 51  
 JobStatus (class in *mqt.qudits.simulation.jobs.jobstatus*), 50

## L

LevelGraph (class in *mqt.qudits.core*), 26  
 LevelGraph (class in *mqt.qudits.core.level\_graph*), 22  
 levels\_setter() (*R* method), 36, 41  
 levels\_setter() (*Rh* method), 37, 41  
 levels\_setter() (*Rz* method), 37, 41  
 load\_from\_file() (*QuantumCircuit* method), 43, 46  
 log\_phy\_map (LevelGraph property), 25, 29  
 LogAdaptiveDecomposition (class in *mqt.qudits.compiler.onedit.mapping\_un\_aware\_transpilation*), 15  
 LogAdaptiveDecomposition (class in *mqt.qudits.compiler.onedit.mapping\_un\_aware\_transpilation.log\_local\_adaptation*), 15  
 LogEntQRCEXPASS (class in *mqt.qudits.compiler.twodit*), 20  
 LogEntQRCEXPASS (class in *mqt.qudits.compiler.twodit.entanglement\_qr*), 17  
 LogEntQRCEXPASS (class in *mqt.qudits.compiler.twodit.entanglement\_qr.log\_ent\_qr\_cex\_decomp*), 17  
 logic\_physical\_map() (*LevelGraph* method), 25, 29  
 LogLocAdaPass (class in *mqt.qudits.compiler.onedit*), 16  
 LogLocAdaPass (class in *mqt.qudits.compiler.onedit.mapping\_un\_aware\_transpilation*), 15  
 LogLocAdaPass (class in *mqt.qudits.compiler.onedit.mapping\_un\_aware\_transpilation.log\_local\_adaptation*), 15  
 LogLocQRPASS (class in *mqt.qudits.compiler.onedit*), 16  
 LogLocQRPASS (class in *mqt.qudits.compiler.onedit.mapping\_un\_aware\_transpilation*), 15  
 LogLocQRPASS (class in *mqt.qudits.compiler.onedit.mapping\_un\_aware\_transpilation.log\_local\_qr\_decomp*), 15  
 LS (class in *mqt.qudits.quantum\_circuit.gates*), 40  
 LS (class in *mqt.qudits.quantum\_circuit.gates.ls*), 35  
 ls() (*QuantumCircuit* method), 43, 45  
 ls\_ansatz() (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz*), 18  
 ls\_ansatz() (in module *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz.ansatz\_gen*), 18

## M

MatrixFactory (class in *mqt.qudits.quantum\_circuit.components.extensions.matrix\_factory*), 32  
 max\_depth() (*NaryTree* method), 21, 26  
 MAX\_NUM\_LAYERS (*Optimizer* attribute), 19, 20  
 min\_cost\_decomp() (*NaryTree* method), 21, 26  
 MISim (class in *mqt.qudits.simulation.backends*), 49  
 MISim (class in *mqt.qudits.simulation.backends.misim*), 48  
 module  
   *mqt.qudits*, 11  
   *mqt.qudits.compiler*, 11  
   *mqt.qudits.compiler.compilation\_minertools*, 11  
   *mqt.qudits.compiler.compilation\_minertools.local\_compilation\_minertools*, 11  
   *mqt.qudits.compiler.compilation\_minertools.naive\_unitary\_verifier*, 11  
   *mqt.qudits.compiler.compilation\_minertools.numerical\_ansatz\_utils*, 11  
   *mqt.qudits.compiler.compiler\_pass*, 20  
   *mqt.qudits.compiler.dit\_manager*, 21  
   *mqt.qudits.compiler.onedit*, 12  
   *mqt.qudits.compiler.onedit.local\_operation\_swap*, 12  
   *mqt.qudits.compiler.onedit.local\_operation\_swap.swap\_routine*, 12  
   *mqt.qudits.compiler.onedit.local\_phases\_transpilation*, 13

mqt.qudits.compiler.onedit.local\_phases\_transpilation.remove\_phase\_rotations, 36  
mqt.qudits.compiler.onedit.local\_phases\_transpilation.remove\_phase\_rotations.circuit.gates.randu, 36  
mqt.qudits.compiler.onedit.mapping\_aware\_transpilation.mqt.qudits.quantum\_circuit.gates.rh, 36  
mqt.qudits.compiler.onedit.mapping\_aware\_transpilation.mqt.qudits.quantum\_circuit.gates.rz, 37  
mqt.qudits.compiler.onedit.mapping\_aware\_transpilation.mqt.qudits.quantum\_circuit.gates.s, 37  
mqt.qudits.compiler.onedit.mapping\_aware\_transpilation.phy\_local\_adaptive\_decomp.circuit.gates.virt\_rz, 37  
mqt.qudits.compiler.onedit.mapping\_aware\_transpilation.phy\_local\_adaptive\_decomp.mqt.qudits.quantum\_circuit.gates.x, 38  
mqt.qudits.compiler.onedit.mapping\_aware\_transpilation.phy\_local\_qr\_decomp.circuit.gates.z, 38  
mqt.qudits.compiler.onedit.mapping\_un\_aware\_transpilation.mqt.qudits.quantum\_circuit.qasm, 44  
mqt.qudits.compiler.onedit.mapping\_un\_aware\_transpilation.mqt.qudits.simulation, 47  
mqt.qudits.compiler.onedit.mapping\_un\_aware\_transpilation.mqt.qudits.simulation.backends, 47  
mqt.qudits.compiler.onedit.mapping\_un\_aware\_transpilation.mqt.qudits.simulation.backends.backends.fake\_backendv2, 48  
mqt.qudits.compiler.onedit.mapping\_un\_aware\_transpilation.mqt.qudits.simulation.backends.fake\_backends, 47  
mqt.qudits.compiler.onedit.mapping\_un\_aware\_transpilation.mqt.qudits.simulation.backends.fake\_backends.fake\_traps2six, 47  
mqt.qudits.compiler.twodit, 16  
mqt.qudits.compiler.twodit.entanglement\_qr, 16  
mqt.qudits.compiler.twodit.entanglement\_qr.crot, 16  
mqt.qudits.compiler.twodit.entanglement\_qr.log\_ent\_qr\_cex\_decomp, 17  
mqt.qudits.compiler.twodit.entanglement\_qr.pswap, 17  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.simulation.backends.fake\_backends.fake\_traps2three, 47  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.simulation.backends.misim, 48  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.simulation.backends.stochastic\_sim, 48  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.simulation.backends.tnsim, 49  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.simulation.jobs, 49  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.simulation.jobs.job, 49  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.simulation.jobs.job\_result, 50  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.simulation.jobs.jobstatus, 50  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.simulation.noise\_tools, 52  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.simulation.noise\_tools.noise, 52  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.simulation.noise\_tools.noisy\_circuit\_factory, 52  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.simulation.provider, 53  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.simulation.save\_info, 53  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.simulation.serialization, 54  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.visualisation.drawing\_routines, 54  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.visualization.mini\_quantum\_information, 54  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.visualization.plot\_information, 54  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits, 19  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.distance\_measures, 19  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.compiler, 19  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.optimizer, 19  
mqt.qudits.compiler.twodit.variational\_twodit\_compilation.mqt.qudits.compiler.compilation\_minertools, 11  
mqt.qudits.core, 21  
mqt.qudits.core dfs\_tree, 21  
mqt.qudits.core.level\_graph, 22  
mqt.qudits.exceptions, 30  
mqt.qudits.exceptions.backenderror, 30  
mqt.qudits.exceptions.circuiterror, 30  
mqt.qudits.exceptions.compilerexception, 30  
mqt.qudits.exceptions.joberror, 31  
mqt.qudits.quantum\_circuit, 32  
mqt.qudits.quantum\_circuit.circuit, 42  
mqt.qudits.quantum\_circuit.components, 32  
mqt.qudits.quantum\_circuit.components.classic\_register, 33  
mqt.qudits.quantum\_circuit.components.extensions, 32  
mqt.qudits.quantum\_circuit.components.extensions.composite, 32  
mqt.qudits.quantum\_circuit.components.extensions.gate\_types, 32  
mqt.qudits.quantum\_circuit.components.extensions.map\_to\_compiler.onedit.local\_phases\_transpilation.propagate\_virtro, 32  
mqt.qudits.quantum\_circuit.components.quantum\_register, 33  
mqt.qudits.quantum\_circuit.gate, 44  
mqt.qudits.quantum\_circuit.gates, 33  
mqt.qudits.quantum\_circuit.gates.csum, 33  
mqt.qudits.quantum\_circuit.gates.custom\_multi, 34  
mqt.qudits.quantum\_circuit.gates.custom\_one, 34  
mqt.qudits.quantum\_circuit.gates.custom\_two, 34  
mqt.qudits.quantum\_circuit.gates.cx, 34  
mqt.qudits.quantum\_circuit.gates.gellmann, 35  
mqt.qudits.quantum\_circuit.gates.h, 35  
mqt.qudits.quantum\_circuit.gates.ls, 35  
mqt.qudits.quantum\_circuit.gates.ms, 35

```

mqt.qudits.compiler.twodit
 module, 16
mqt.qudits.compiler.twodit.entanglement_qr
 module, 16
mqt.qudits.compiler.twodit.entanglement_qr.crot
 module, 16
mqt.qudits.compiler.twodit.entanglement_qr.log_ent_qr_cex_decomp
 module, 17
mqt.qudits.compiler.twodit.entanglement_qr.pswap
 module, 17
mqt.qudits.compiler.twodit.variational_twodit_compilation
 module, 18
mqt.qudits.compiler.twodit.variational_twodit_compilation.answer
 module, 18
mqt.qudits.compiler.twodit.variational_twodit_compilation.answer Ansatz_gen
 module, 18
mqt.qudits.compiler.twodit.variational_twodit_compilation.answer instantiate
 module, 18
mqt.qudits.compiler.twodit.variational_twodit_compilation.answer parameterize
 module, 18
mqt.qudits.compiler.twodit.variational_twodit_compilation.answer solve_n_search
 module, 20
mqt.qudits.compiler.twodit.variational_twodit_compilation.optimize
 module, 19
mqt.qudits.compiler.twodit.variational_twodit_compilation.optimize n_e_measures
 module, 19
mqt.qudits.compiler.twodit.variational_twodit_compilation.optimize n_minimizer
 module, 19
mqt.qudits.core
 module, 21
mqt.qudits.core.dfs_tree
 module, 21
mqt.qudits.core.level_graph
 module, 22
mqt.qudits.exceptions
 module, 30
mqt.qudits.exceptions.backenderror
 module, 30
mqt.qudits.exceptions.circuiterror
 module, 30
mqt.qudits.exceptions.compilerexception
 module, 30
mqt.qudits.exceptions.joberror
 module, 31
mqt.qudits.quantum_circuit
 module, 32
mqt.qudits.quantum_circuit.circuit
 module, 42
mqt.qudits.quantum_circuit.components
 module, 32
mqt.qudits.quantum_circuit.components.classic_register
 module, 33
mqt.qudits.quantum_circuit.components.extensions
 module, 32
mqt.qudits.quantum_circuit.components.extensions.controls
 module, 32
mqt.qudits.quantum_circuit.components.extensions.gate_types
 module, 32
mqt.qudits.quantum_circuit.components.extensions.matrix_factorization
 module, 32
mqt.qudits.quantum_circuit.components.quantum_register
 module, 33
mqt.qudits.quantum_circuit.gate
 module, 44
mqt.qudits.quantum_circuit.gates
 module, 33
mqt.qudits.quantum_circuit.gates.csum
 module, 33
mqt.qudits.quantum_circuit.gates.custom_multi
 module, 34
mqt.qudits.quantum_circuit.gates.custom_one
 module, 34
mqt.qudits.quantum_circuit.gates.custom_two
 module, 34
mqt.qudits.quantum_circuit.gates.cx
 module, 34
mqt.qudits.quantum_circuit.gates.gellmann
 module, 35
mqt.qudits.quantum_circuit.gates.h
 module, 35
mqt.qudits.quantum_circuit.gates.ls
 module, 35
mqt.qudits.quantum_circuit.gates.ms
 module, 35
mqt.qudits.quantum_circuit.gates.perm
 module, 36
mqt.qudits.quantum_circuit.gates.r
 module, 36
mqt.qudits.quantum_circuit.gates.randu
 module, 36
mqt.qudits.quantum_circuit.gates.rh
 module, 36
mqt.qudits.quantum_circuit.gates.rz
 module, 36
mqt.qudits.quantum_circuit.gates.s
 module, 36
mqt.qudits.quantum_circuit.gates.virt_rz
 module, 36
mqt.qudits.quantum_circuit.gates.x
 module, 38
mqt.qudits.quantum_circuit.gates.z
 module, 38
mqt.qudits.quantum_circuit.qasm
 module, 38
mqt.qudits.simulation
 module, 47
mqt.qudits.simulation.backends
 module, 47
mqt.qudits.simulation.backends.backendv2
 module, 48
mqt.qudits.simulation.backends.fake_backends
 module, 47
mqt.qudits.simulation.backends.fake_backends.fake_traps2six
 module, 47
mqt.qudits.simulation.backends.fake_backends.fake_traps2three
 module, 47
mqt.qudits.simulation.backends.misim
 module, 48
mqt.qudits.simulation.backends.stochastic_sim
 module, 48
mqt.qudits.simulation.backends.tnsim
 module, 49
mqt.qudits.simulation.jobs
 module, 49
mqt.qudits.simulation.jobs.job
 module, 49
mqt.qudits.simulation.jobs.job_result
 module, 50
mqt.qudits.simulation.jobs.jobstatus
 module, 50
mqt.qudits.simulation.noise_tools
 module, 52
mqt.qudits.simulation.noise_tools.noise
 module, 52
mqt.qudits.simulation.noise_tools.noisy_circuit_factory
 module, 52
mqt.qudits.simulation.qudit_provider
 module, 53
mqt.qudits.simulation.save_info
 module, 53
mqt.qudits.visualisation
 module, 54
mqt.qudits.visualisation.drawing_routines
 module, 54
mqt.qudits.visualisation.mini_quantum_information
 module, 54
mqt.qudits.visualisation.plot_information
 module, 54
MQTQuditProvider (class in mqt.qudits.simulation), 53
MQTQuditProvider (class in
 mqt.qudits.simulation.qudit_provider), 53
MS (class in mqt.qudits.quantum_circuit.gates), 40
MS (class in mqt.qudits.quantum_circuit.gates.ms), 35
ms() (QuantumCircuit method), 43, 45

```



`ms_ansatz()` (in module `mqt.qudits.compiler.twodit.variational_twodit_compilation.ansatz.ansatz_gen`), 18  
`ms_ansatz()` (in module `mqt.qudits.compiler.twodit.variational_twodit_compilation.ansatz.ansatz_gen`), 18  
**MULTI** (*GateTypes* attribute), 32, 38

## N

**N**ArYTree (class in `mqt.qudits.core`), 26  
**N**ArYTree (class in `mqt.qudits.core.dfs_tree`), 21  
`new_mod()` (in module `mqt.qudits.compiler.compilation_minitools`), 11  
`new_mod()` (in module `mqt.qudits.compiler.compilation_minitools.local_compilation_minitools`), 11  
**N**ode (class in `mqt.qudits.core`), 26  
**N**ode (class in `mqt.qudits.core.dfs_tree`), 21  
**N**odeNotFoundException, 30, 31  
**N**oise (class in `mqt.qudits.simulation.noise_tools`), 52  
**N**oise (class in `mqt.qudits.simulation.noise_tools.noise`), 52  
**N**oiseModel (class in `mqt.qudits.simulation.noise_tools`), 52  
**N**oiseModel (class in `mqt.qudits.simulation.noise_tools.noise`), 52  
**N**oisyCircuitFactory (class in `mqt.qudits.simulation.noise_tools`), 53  
**N**oisyCircuitFactory (class in `mqt.qudits.simulation.noise_tools.noisy_circuit_factory`), 52  
`num_qudits` (*Backend* property), 48  
`num_qudits` (*QuantumCircuit* property), 42, 45

## O

**O**BJ\_FIDELITY (*Optimizer* attribute), 19, 20  
`obj_fun_core()` (*Optimizer* class method), 19, 20  
`objective_fnc_cu()` (*Optimizer* class method), 19, 20  
`objective_fnc_ls()` (*Optimizer* class method), 19, 20  
`objective_fnc_ms()` (*Optimizer* class method), 19, 20  
`on0()` (in module `mqt.qudits.compiler.compilation_minitools`), 12  
`on0()` (in module `mqt.qudits.compiler.compilation_minitools.numerical_ansatz_utils`), 11  
`on1()` (in module `mqt.qudits.compiler.compilation_minitools`), 12  
`on1()` (in module `mqt.qudits.compiler.compilation_minitools.numerical_ansatz_utils`), 11  
`operation_names` (*Backend* property), 48  
`operations` (*Backend* property), 48  
**O**ptimizer (class in `mqt.qudits.compiler.twodit.variational_twodit_compilation.optimizer`), 20  
**O**ptimizer (class in `mqt.qudits.compiler.twodit.variational_twodit_compilation.optimizer`), 19  
`options` (*Backend* property), 48

## P

`params_splitter()` (in module `mqt.qudits.compiler.twodit.variational_twodit_compilation.ansatz.ansatz_gen`), 18  
`parse_creg()` (*QASM* method), 45, 46  
`parse_ditqasm2_file()` (*QASM* method), 45, 47  
`parse_ditqasm2_str()` (*QASM* method), 45, 47  
`parse_gate()` (*QASM* method), 45, 46  
`parse_ignore()` (*QASM* method), 45, 46  
`parse_nonspecial_lines()` (*QASM* method), 45, 46  
`parse_qreg()` (*QASM* method), 45, 46  
`partial_trace()` (in module `mqt.qudits.visualisation`), 54  
`partial_trace()` (in module `mqt.qudits.visualisation.mini_quantum_information`), 54  
`passes_enabled` (*QuditCompiler* attribute), 21  
**P**erm (class in `mqt.qudits.quantum_circuit.gates`), 40  
**P**erm (class in `mqt.qudits.quantum_circuit.gates.perm`), 36  
`permute_crot_101_as_list()` (*CRotGen* method), 17  
`permute_doubled_crot_101_as_list()` (*CRotGen* method), 17  
`permute_pswap_101_as_list()` (*PSwapGen* method), 17  
`permute_quad_pswap_101_as_list()` (*PSwapGen* method), 17  
`phase_storing_setup()` (*LevelGraph* method), 25, 29  
`phi_cost()` (in module `mqt.qudits.compiler.compilation_minitools`), 12  
`phi_cost()` (in module `mqt.qudits.compiler.compilation_minitools.local_compilation_minitools`), 11  
**P**hyAdaptiveDecomposition (class in `mqt.qudits.compiler.onedit.mapping_aware_transpilation`), 14  
**P**hyAdaptiveDecomposition (class in `mqt.qudits.compiler.onedit.mapping_aware_transpilation.phy_local_adaptive_decomposition`), 14  
**P**hyLocAdaPass (class in `mqt.qudits.compiler.onedit`), 16  
**P**hyLocAdaPass (class in `mqt.qudits.compiler.onedit.mapping_aware_transpilation`), 14  
**P**hyLocAdaPass (class in `mqt.qudits.compiler.onedit.mapping_aware_transpilation.phy_local_adaptive_decomposition`), 14  
**P**hyLocQRPass (class in `mqt.qudits.compiler.onedit`), 16  
**P**hyLocQRPass (class in `mqt.qudits.compiler.onedit.mapping_aware_transpilation`), 14  
**P**hyLocQRPass (class in `mqt.qudits.compiler.onedit.mapping_aware_transpilation.phy_local_qr_decomposition`), 14  
**P**hyQrDecomp (class in `mqt.qudits.compiler.onedit.mapping_aware_transpilation`), 14  
**P**hyQrDecomp (class in `mqt.qudits.compiler.onedit.mapping_aware_transpilation.phy_local_qr_decomposition`), 14  
`pi_mod()` (in module `mqt.qudits.compiler.compilation_minitools`), 12  
`pi_mod()` (in module `mqt.qudits.compiler.compilation_minitools.local_compilation_minitools`), 11  
`plot_counts()` (in module `mqt.qudits.visualisation`), 55  
`plot_counts()` (in module `mqt.qudits.visualisation.plot_information`), 54  
`plot_state()` (in module `mqt.qudits.visualisation`), 55  
`plot_state()` (in module `mqt.qudits.visualisation.plot_information`), 54  
`pm()` (*QuantumCircuit* method), 43, 45  
`prepare_ansatz()` (in module `mqt.qudits.compiler.twodit.variational_twodit_compilation.ansatz.ansatz_gen`), 18  
`print_tree()` (*N*ArYTree method), 21, 26  
`probability_dephasing` (*Noise* attribute), 52  
`probability_depolarizing` (*Noise* attribute), 52  
`propagate_z()` (*ZPropagationPass* method), 13, 16  
`provider` (*Backend* property), 48  
`pswap_101_as_list()` (*PSwapGen* method), 17  
**P**SwapGen (class in `mqt.qudits.compiler.twodit.entanglement_qr`), 17  
**P**SwapGen (class in `mqt.qudits.compiler.twodit.entanglement_qr.pswap`), 17  
`parametrize()` (in module `mqt.qudits.compiler.twodit.entanglement_qr.pswap`), 17

## Q

**Q**ASM (class in `mqt.qudits.quantum_circuit`), 46  
**Q**ASM (class in `mqt.qudits.quantum_circuit.qasm`), 44  
`qasm_to_gate_set_dict` (*QuantumCircuit* attribute), 42, 45  
**Q**rDecomp (class in `mqt.qudits.compiler.onedit.mapping_unaware_transpilation.log_local_qr_decomposition`), 15  
**Q**uantumCircuit (class in `mqt.qudits.quantum_circuit`), 45  
**Q**uantumCircuit (class in `mqt.qudits.quantum_circuit.circuit`), 42  
**Q**uantumRegister (class in `mqt.qudits.quantum_circuit`), 46  
**Q**uantumRegister (class in `mqt.qudits.quantum_circuit.components`), 33  
**Q**uantumRegister (class in `mqt.qudits.quantum_circuit.components.quantum_register`), 33  
**Q**uditCompiler (class in `mqt.qudits.compiler.dit_manager`), 21

QUEUED (*JobStatus* attribute), 50, 51

## R

R (class in *mqt.qudits.quantum\_circuit.gates*), 40  
R (class in *mqt.qudits.quantum\_circuit.gates.r*), 36  
r() (*QuantumCircuit* method), 43, 45  
RandU (class in *mqt.qudits.quantum\_circuit.gates*), 41  
RandU (class in *mqt.qudits.quantum\_circuit.gates.randu*), 36  
randu() (*QuantumCircuit* method), 43, 46  
reference\_lines (*Gate* property), 44  
regulate\_theta() (in module  
    *mqt.qudits.compiler.compilation\_minitools*), 12  
regulate\_theta() (in module  
    *mqt.qudits.compiler.compilation\_minitools.local\_compilation\_minitools*), 11  
reindex() (in module  
    *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz*), 18  
reindex() (in module  
    *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz\_parameters*), 18  
remove\_initial\_rz() (*ZRemovalPass* method), 13, 16  
remove\_rz\_gates() (*ZRemovalPass* method), 13, 16  
remove\_trailing\_rz\_sequence() (*ZRemovalPass* method), 13, 16  
remove\_z() (*ZPropagationPass* method), 13, 16  
replace\_gate() (*QuantumCircuit* method), 43, 46  
reset() (*QuantumCircuit* method), 42, 45  
result() (*Job* method), 50, 51  
retrieve\_decomposition() (*NAryTree* method), 21, 26  
return\_custom\_data() (*Gate* method), 44  
Rh (class in *mqt.qudits.quantum\_circuit.gates*), 41  
Rh (class in *mqt.qudits.quantum\_circuit.gates.rh*), 36  
rh() (*QuantumCircuit* method), 43, 45  
rotation\_cost\_calc() (in module  
    *mqt.qudits.compiler.compilation\_minitools*), 12  
rotation\_cost\_calc() (in module  
    *mqt.qudits.compiler.compilation\_minitools.local\_compilation\_minitools*), 11  
route\_states2rotate\_basic() (in module  
    *mqt.qudits.compiler.onedit.local\_operation\_swap*), 12  
route\_states2rotate\_basic() (in module  
    *mqt.qudits.compiler.onedit.local\_operation\_swap.swap\_routes*), 12  
RoutingException, 31  
run() (*Backend* method), 48  
run() (in module  
    *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.ansatz\_parameters*), 20  
run() (*MISim* method), 48, 49  
run() (*TNSim* method), 49  
RUNNING (*JobStatus* attribute), 50, 52  
running() (*Job* method), 49, 51  
Rz (class in *mqt.qudits.quantum\_circuit.gates*), 41  
Rz (class in *mqt.qudits.quantum\_circuit.gates.rz*), 37  
rz() (*QuantumCircuit* method), 43, 46

## S

S (class in *mqt.qudits.quantum\_circuit.gates*), 41  
S (class in *mqt.qudits.quantum\_circuit.gates.s*), 37  
s() (*QuantumCircuit* method), 43, 46  
safe\_eval\_math\_expression() (*QASM* method), 45, 46  
save\_full\_states() (in module  
    *mqt.qudits.simulation.save\_info*), 53  
save\_shots() (in module *mqt.qudits.simulation.save\_info*), 53  
save\_to\_file() (*QuantumCircuit* method), 43, 46  
save\_to\_png() (*HistogramWithErrors* method), 54  
SequenceFoundException, 31  
set\_circuit() (*LevelGraph* method), 25, 30  
set\_gate\_type\_multi() (*Gate* method), 44  
set\_gate\_type\_single() (*Gate* method), 44  
set\_gate\_type\_two() (*Gate* method), 44  
set\_instructions() (*QuantumCircuit* method), 43, 46  
set\_options() (*Backend* method), 48  
set\_qudits\_index() (*LevelGraph* method), 26, 30  
set\_result() (*Job* method), 50, 51

SINGLE (*GateTypes* attribute), 32, 38

SINGLE\_DIM\_0 (*Optimizer* attribute), 19, 20

SINGLE\_DIM\_1 (*Optimizer* attribute), 19, 20

size\_check() (in module  
    *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.opt*), 20  
size\_check() (in module  
    *mqt.qudits.compiler.twodit.variational\_twodit\_compilation.opt.distance\_measurement*), 19  
size\_refresh() (*NAryTree* method), 21, 26  
solve\_anneal() (*Optimizer* class method), 19, 20  
state\_labels() (in module  
    *mqt.qudits.visualisation.plot\_information*), 54  
status() (*Job* method), 50, 51  
stop\_execution() (in module  
    *mqt.qudits.simulation.backends.stochastic\_sim*), 48  
stochastic\_execution\_misim() (in module  
    *mqt.qudits.simulation.backends.stochastic\_sim*), 49  
stochastic\_simulation() (in module  
    *mqt.qudits.simulation.backends.stochastic\_sim*), 48  
stochastic\_simulation\_misim() (in module  
    *mqt.qudits.simulation.backends.stochastic\_sim*), 48  
submit() (*Job* method), 50, 51  
swap\_elements() (in module  
    *mqt.qudits.compiler.compilation\_minitools*), 12  
swap\_elements() (in module  
    *mqt.qudits.compiler.compilation\_minitools.local\_compilation\_minitools*), 11  
swap\_node\_attr\_simple() (*LevelGraph* method), 25, 29  
swap\_node\_attributes() (*LevelGraph* method), 25, 29  
swap\_nodes() (*LevelGraph* method), 25, 29

## T

target (*Backend* attribute), 48

TARGET\_GATE (*Optimizer* attribute), 19, 20

theta\_cost() (in module  
    *mqt.qudits.compiler.compilation\_minitools*), 12

theta\_cost() (in module  
    *mqt.qudits.compiler.compilation\_minitools.local\_compilation\_minitools*), 11  
timer\_var (*Optimizer* attribute), 19, 20

TNSim (class in *mqt.qudits.simulation.backends*), 49

TNSim (class in *mqt.qudits.simulation.backends.tnsim*), 49

to\_matrix() (*Gate* method), 44

to\_qasm() (*QuantumCircuit* method), 43, 46

total\_size (*NAryTree* property), 21, 26

transpile() (*CompilerPass* method), 20, 21

transpile() (*LogLocQRCPass* method), 17, 20

transpile() (*LogLocAdaPass* method), 15, 16

transpile() (*LogLocQRPass* method), 15, 16

transpile() (*PhyLocAdaPass* method), 14, 16

transpile() (*PhyLocQRPass* method), 14, 16

transpile() (*ZPropagationPass* method), 13, 16

transpile() (*ZRemovalPass* method), 13, 16

TWO (*GateTypes* attribute), 32, 38

## U

UnitaryVerifier (class in  
    *mqt.qudits.compiler.compilation\_minitools*), 12

UnitaryVerifier (class in  
    *mqt.qudits.compiler.compilation\_minitools.naive\_unitary\_verifier*), 11

update\_list() (*LevelGraph* method), 25, 29

## V

validate\_parameter() (*CEx* method), 34, 39

validate\_parameter() (*Csum* method), 33, 38

validate\_parameter() (*CustomMulti* method), 34, 39

validate\_parameter() (*CustomOne* method), 34, 39

validate\_parameter() (*CustomTwo* method), 34, 39

validate\_parameter() (*Gate* method), 44

validate\_parameter() (*GellMann* method), 35, 39

validate\_parameter() (*H* method), 35, 40

validate\_parameter() (*LS* method), 35, 40

validate\_parameter() (*MS* method), 35, 40

validate\_parameter() (*Perm method*), 36, 40  
 validate\_parameter() (*R method*), 36, 41  
 validate\_parameter() (*RandU method*), 36, 41  
 validate\_parameter() (*Rh method*), 37, 41  
 validate\_parameter() (*Rz method*), 37, 41  
 validate\_parameter() (*S method*), 37, 41  
 validate\_parameter() (*VirtRz method*), 37, 42  
 validate\_parameter() (*X method*), 38, 42  
 validate\_parameter() (*Z method*), 38, 42  
 VALIDATING (*JobStatus attribute*), 50, 52  
 verify() (*UnitaryVerifier method*), 11, 12  
 version (*Backend property*), 48  
 version (*FakeIonTraps2Six property*), 47  
 version (*FakeIonTraps2Trits property*), 47, 48  
 version (*Job attribute*), 49, 51  
 version (*MQTQuditProvider property*), 53  
 version\_info (*in module mqt.qudits*), 55  
 VirtRz (*class in mqt.qudits.quantum\_circuit.gates*), 42  
 VirtRz (*class in mqt.qudits.quantum\_circuit.gates.virt\_rz*), 37  
 virtrz() (*QuantumCircuit method*), 43, 46

## W

wait\_for\_final\_state() (*Job method*), 50, 51  
 wrap\_in\_identities() (*MatrixFactory class method*), 32

## X

X (*class in mqt.qudits.quantum\_circuit.gates*), 42  
 X (*class in mqt.qudits.quantum\_circuit.gates.x*), 38  
 x() (*QuantumCircuit method*), 43, 46  
 X\_SOLUTION (*Optimizer attribute*), 19, 20

## Z

Z (*class in mqt.qudits.quantum\_circuit.gates*), 42  
 Z (*class in mqt.qudits.quantum\_circuit.gates.z*), 38  
 z() (*QuantumCircuit method*), 43, 46  
 z\_extraction() (*LogAdaptiveDecomposition method*), 15  
 Z\_extraction() (*PhyAdaptiveDecomposition method*), 14  
 z\_from\_crot\_101\_list() (*CRotGen method*), 17  
 z\_pswap\_101\_as\_list() (*PSwapGen method*), 17, 18  
 ZPropagationPass (*class in mqt.qudits.compiler.onedit*), 15  
 ZPropagationPass (*class in*  
     *mqt.qudits.compiler.onedit.local\_phases\_transpilation*),  
     13  
 ZPropagationPass (*class in*  
     *mqt.qudits.compiler.onedit.local\_phases\_transpilation.propagate\_virtz*),  
     13  
 ZRemovalPass (*class in mqt.qudits.compiler.onedit*), 16  
 ZRemovalPass (*class in*  
     *mqt.qudits.compiler.onedit.local\_phases\_transpilation*),  
     13  
 ZRemovalPass (*class in*  
     *mqt.qudits.compiler.onedit.local\_phases\_transpilation.remove\_phase\_rotations*),  
     13