MQSF 25

Beyond Circuits

Compiling the next generation of quantum programs with the MQT

Lukas Burgholzer
lukas@munichquantum.software
@burgholzer

https://munichquantum.software/

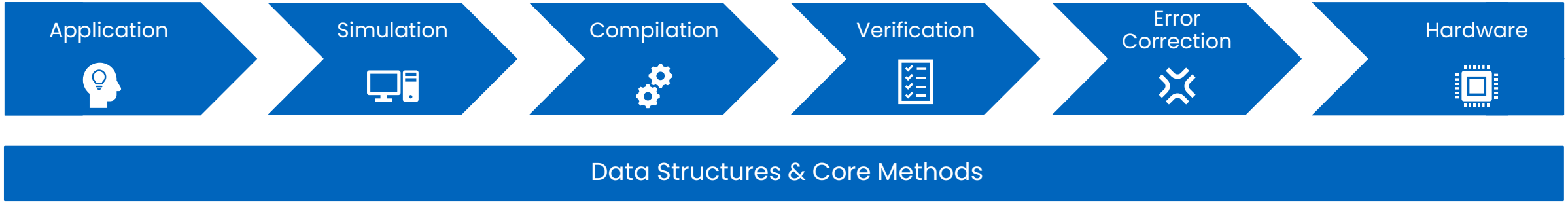
MUNICH QUANTUM SOFTWARE COMPANY

CHAIR FOR DESIGN AUTOMATION

Munich Quantum Valley

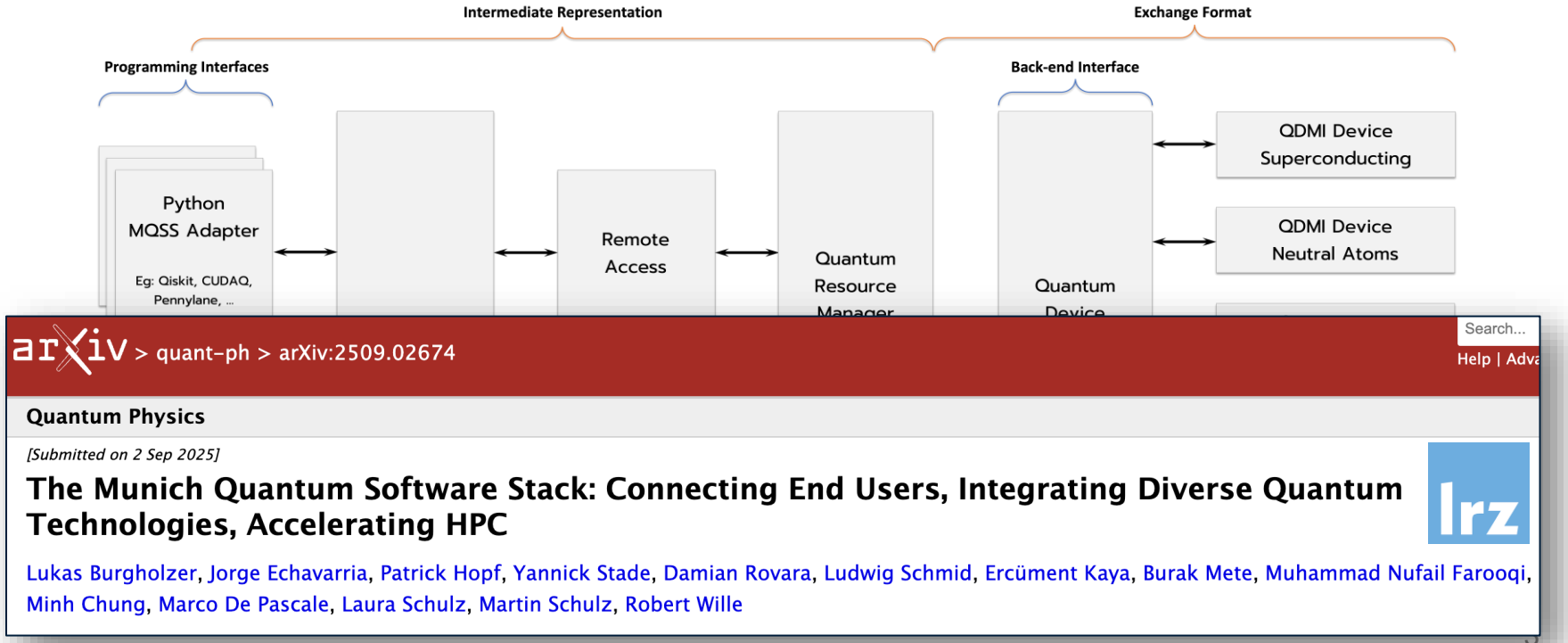
Software for Quantum Computing

Needed across the whole spectrum…



MQSS
Munich Quantum Software Stack

Overview Paper
arxiv.org/abs/2509.02674



The Munich Quantum Toolkit (MQT)

MUNICH QUANTUM TOOLKIT

MUNICH QUANTUM SOFTWARE COMPANY

Leave a ★

All tools are available as open-source repositories on GitHub under the MIT license

MQT ProblemSolver — Application

A Tool for Solving Problems Using Quantum Computing

munich-quantum-toolkit/problemsolver

PyPI

MQT Bench — Application

Benchmarking Software and Tools for Quantum Computing

mqt-bench.app

munich-quantum-toolkit/bench

PyPI

MQT QuSAT — Core Methods

A Tool for Encoding Quantum Computing using Satisfiability Testing (SAT) Techniques

$F \wedge (x_1 \wedge \neg x_2)$

$F \wedge (x_3 \wedge x_2)$

munich-quantum-toolkit/qusat

PyPI

MQT YAQS — Simulation

A Tool for Simulating Open Quantum Systems, Noisy Quantum Circuits, and Realistic Quantum Hardware

munich-quantum-toolkit/yaqs

PyPI

MQT DDSIM — Simulation

A Tool for Classical Quantum Circuit Simulation based on Decision Diagrams

munich-quantum-toolkit/ddsim

PyPI

MQT Predictor — Compilation

A Tool for Determining Good Quantum Circuit Compilation Options

munich-quantum-toolkit /predictor

PyPI

MQT IonShuttler — Compilation

A Tool for Generating Shuttling Schedules for QCCD Architectures

munich-quantum-toolkit/ionshuttler

PyPI

MQT Qudits — Compilation

A Tool for Compiling High-Dimensional Quantum Systems

munich-quantum-toolkit/qudits

PyPI

MQT SyReC — Compilation

A Tool for the Synthesis of Reversible Circuits/Quantum Computing Oracles

munich-quantum-toolkit/syrec

PyPI

MQT QMAP — Compilation

A Tool for Quantum Circuit Mapping And Clifford Circuit Optimization/Synthesis

munich-quantum-toolkit/qmap

PyPI

MQT QCEC — Verification

A Tool for Quantum Circuit Equivalence Checking

munich-quantum-toolkit/qcec

PyPI

MQT Debugger — Verification

A semi-automated tool for debugging quantum programs

munich-quantum-toolkit/debugger

PyPI

MQT DDVis — Data Structures

A Web-Application visualizing Decision Diagrams for Quantum Computing

www.cda.cit.tum.de/app/ddvis

MQT Core — Data Structures

The Backbone of the MQT Intermediate Representation (IR) Decision Diagram and ZX Package

munich-quantum-toolkit/core

PyPI

MQT NAViz — Core Methods

A visualization software for neutral atom quantum computers.

munich-quantum-toolkit/naviz

PyPI

MQT QECC — QECC

A Tool for Quantum Error Correcting Codes

munich-quantum-toolkit/qecc

PyPI

https://mqt.readthedocs.io

https://github.com/munich-quantum-toolkit

Recent Updates from the MQT

📦 12 new releases

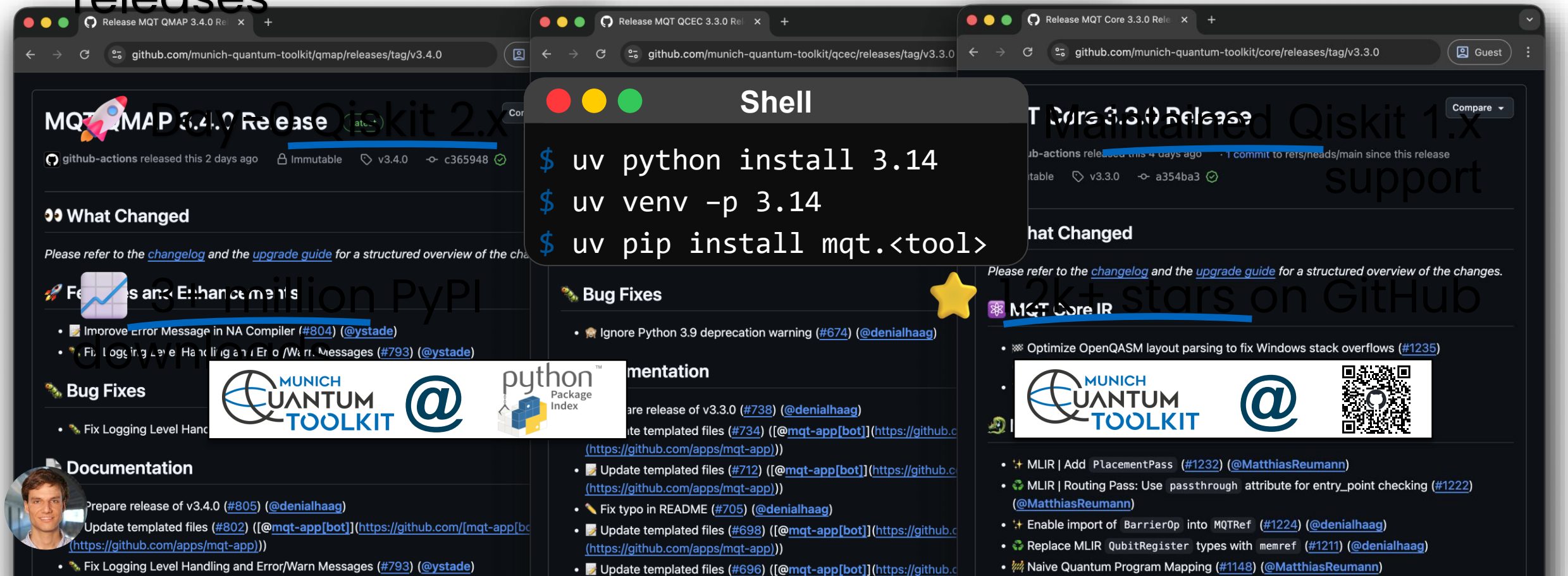
Day-0 Python 3.14 support

Day-0 Qiskit 2.x support

Maintained Qiskit 1.x support

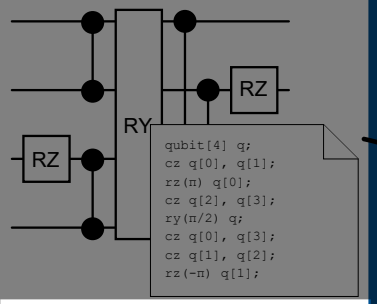
8+ million PyPI downloads

12k+ stars on GitHub

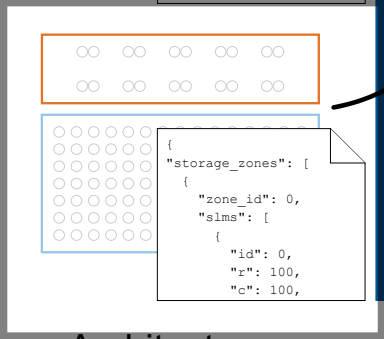


Compiling for Next Generation Hardware

Quantum Circuit




```
qubit[4] q;
cz q[0], q[1];
rz(π) q[0];
cz q[2], q[3];
ry(π/2) q;
cz q[0], q[3];
cz q[1], q[2];
rz(-π) q[1];
```

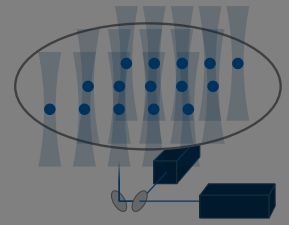


```
{
"storage_zones": [
  {
    "zone_id": 0,
    "slms": [
      {
        "id": 0,
        "r": 100,
        "c": 100,
```

Architecture Specification

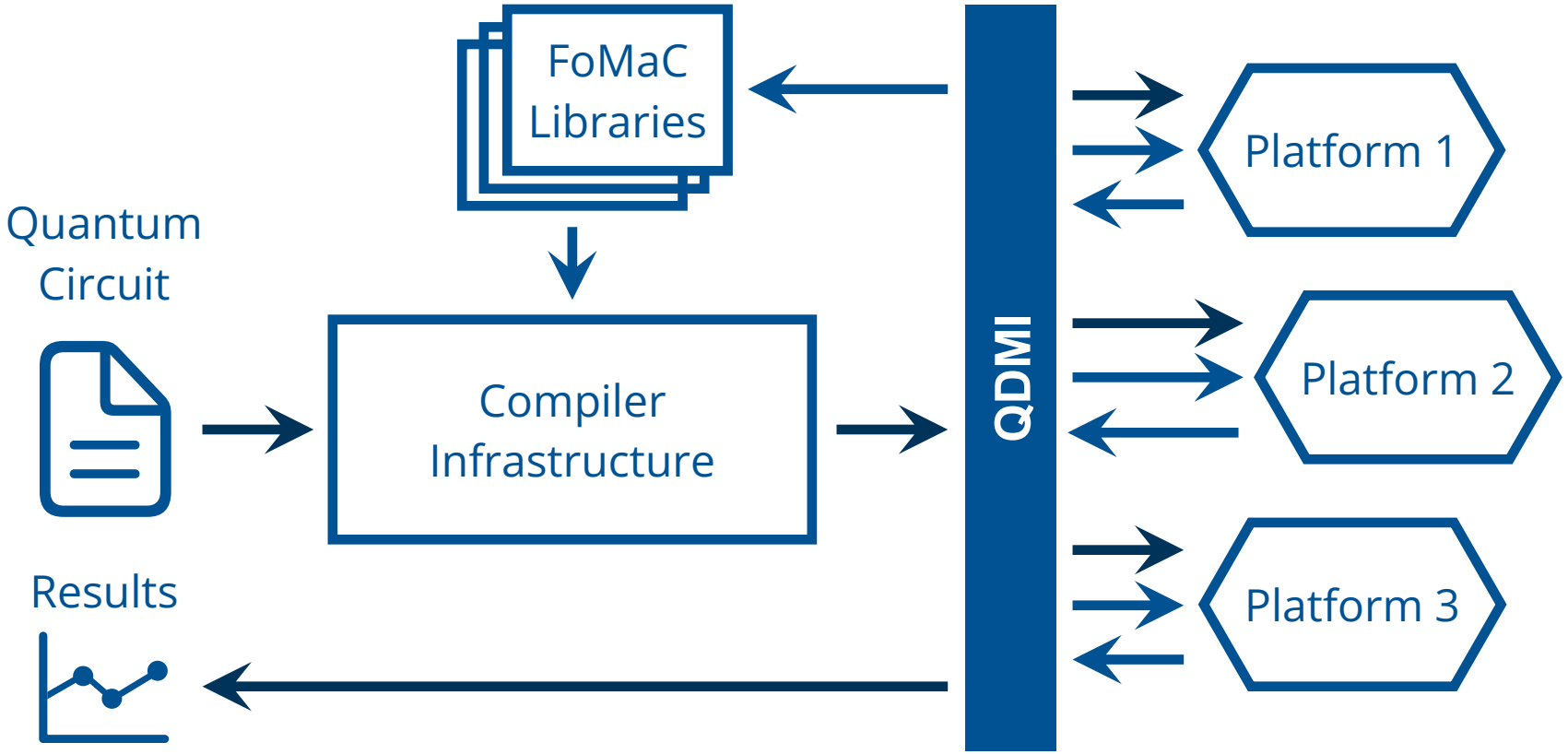
Compiler

Neutral Atom Quantum Computer



```
atom [
  (0.0, 18.0) atom0
  (3.0, 18.0) atom1
  (6.0, 18.0) atom2
  (9.0, 18.0) atom3
]
@+ load [
  atom0
  atom1
```

QDMI The Quantum Device Management Interface



QDMI Documentation

munich-quantum-software-stack.github.io/QDMI

QDMI GitHub

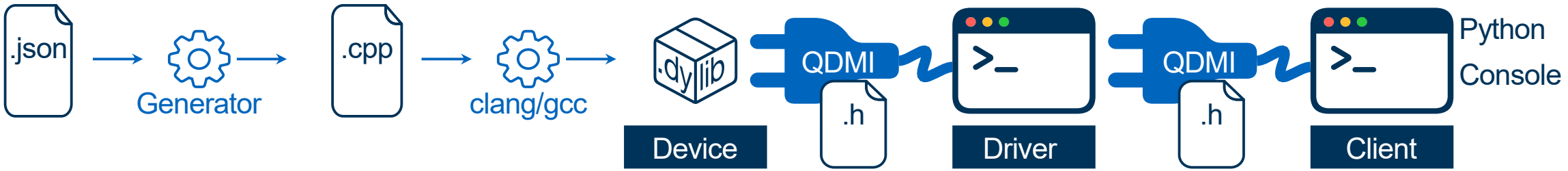
github.com/munich-quantum-software-stack/QDMI

Compilation & Execution Flow

Information Flow

open-source, openly-developed, multi-modality, HPC-compatible

Extending QDMI Device for Neutral Atoms



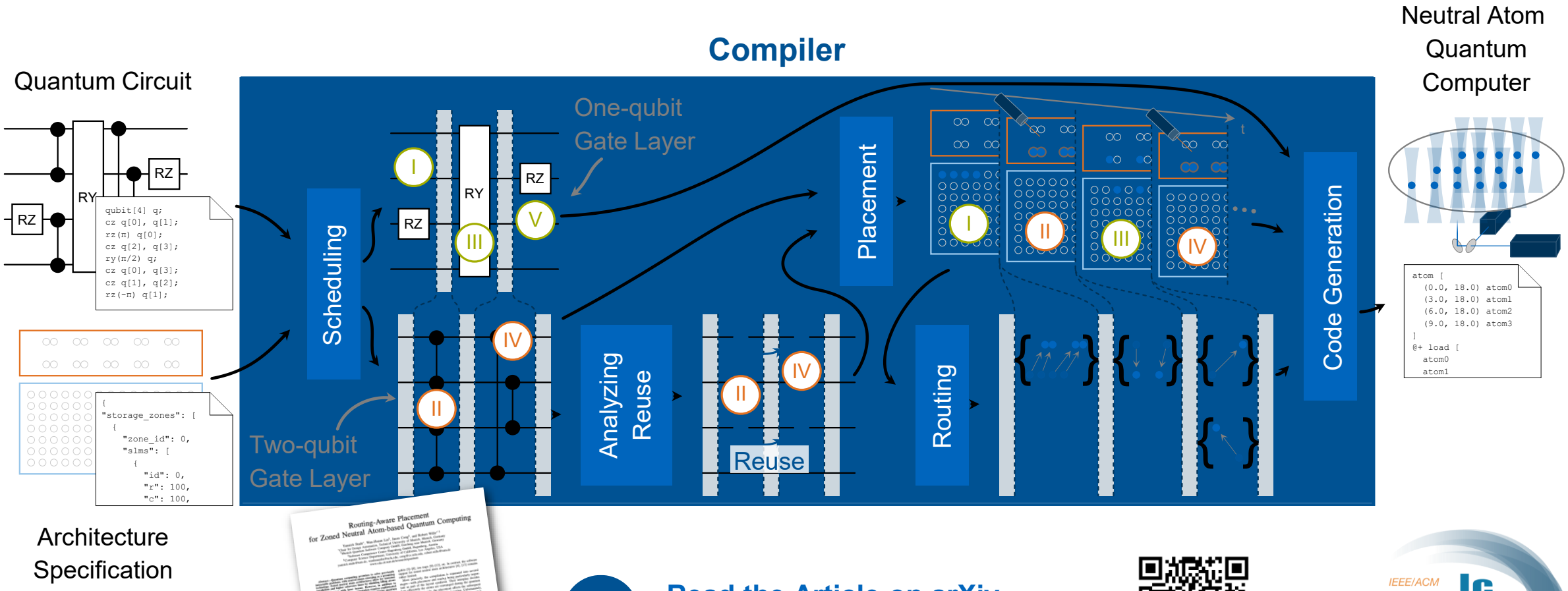

```
int MQ
int MQ
int MQ
```

94

Python

```
>>> from mqt.core.fomac import *
>>> device = devices()[0]
>>> device.name()
'MQT NA Default QDMI Device'
>>> site = device.sites()[24]
>>> site.x_coordinate(), site.y_coordinate()
(1, 2)
>>> [op.name() for op in device.operations()]
['ry', 'cz', 'rz', 'cz', 'load<0>', 'move<0>', 'store<0>']
```

Compiling for Next Generation Hardware



Read the Article on arXiv
arxiv.org/abs/2505.22715

IEEE/ACM ICCAD 2025 INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN 44th Edition

Try It Out

Python

```
>>> from mqt.qmap.na.zoned import *
>>> arch = ZonedNeutralAtomArchitecture.from_json_file(\
"arch.json")
>>> compiler = RoutingAwareCompiler(arch)
>>> from mqt.core import load
>>> circ = load(your_qiskit_circuit)
>>> code = compiler.compile(circ)
>>> print(code)
atom (0.000, 57.000) atom0
atom (3.000, 57.000) atom1
...
```

MUNICH QUANTUM TOOLKIT

MQT QMAP

124 ⭐ 517k

Shell

```
$ (uv) pip install mqt.qmap
```

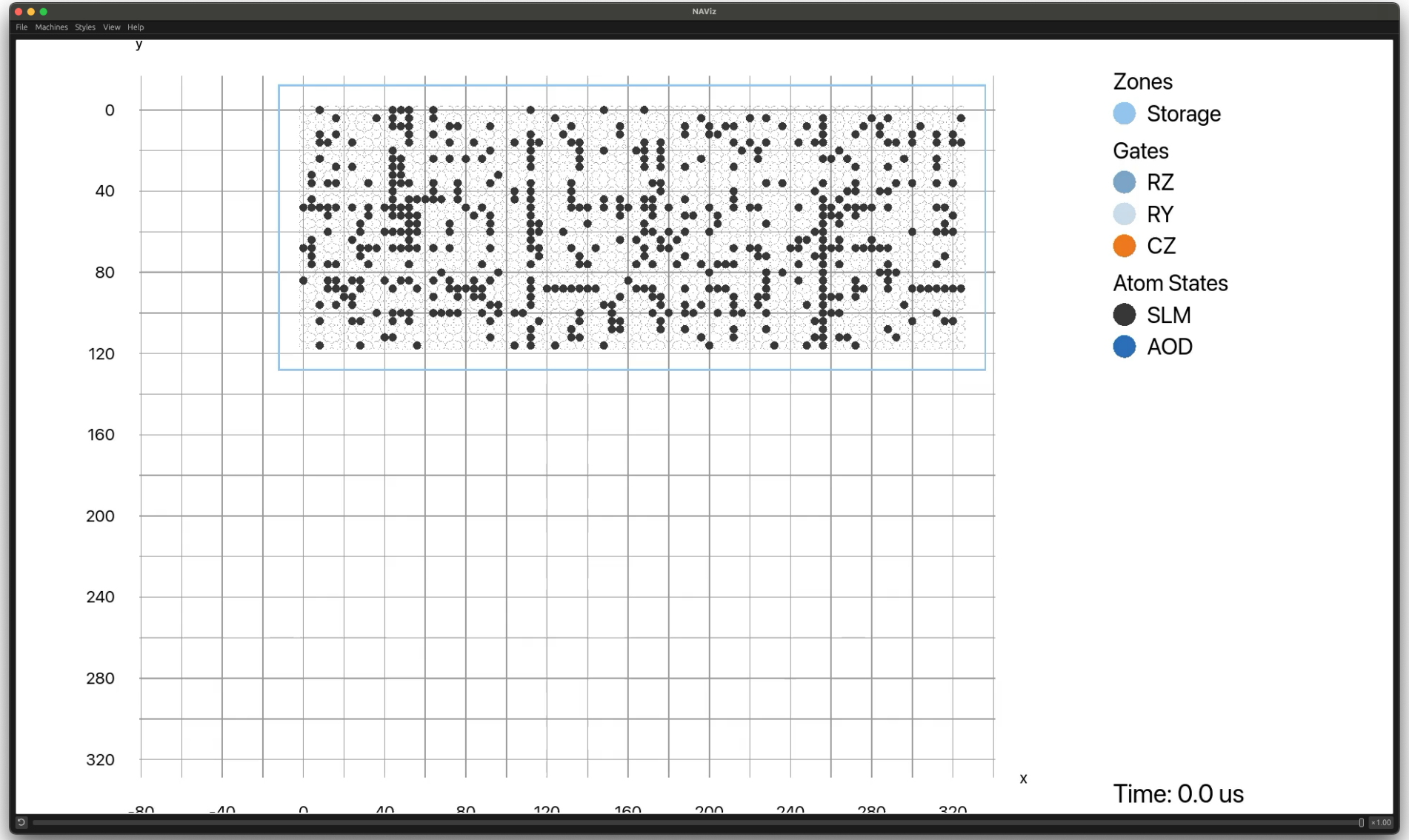
Qiskit Ecosystem

PLANQK

MQT QMAP Documentation with Example
mqt.readthedocs.io/projects/qmap/en/latest/

MQT QMAP GitHub
github.com/munich-quantum-toolkit/qmap

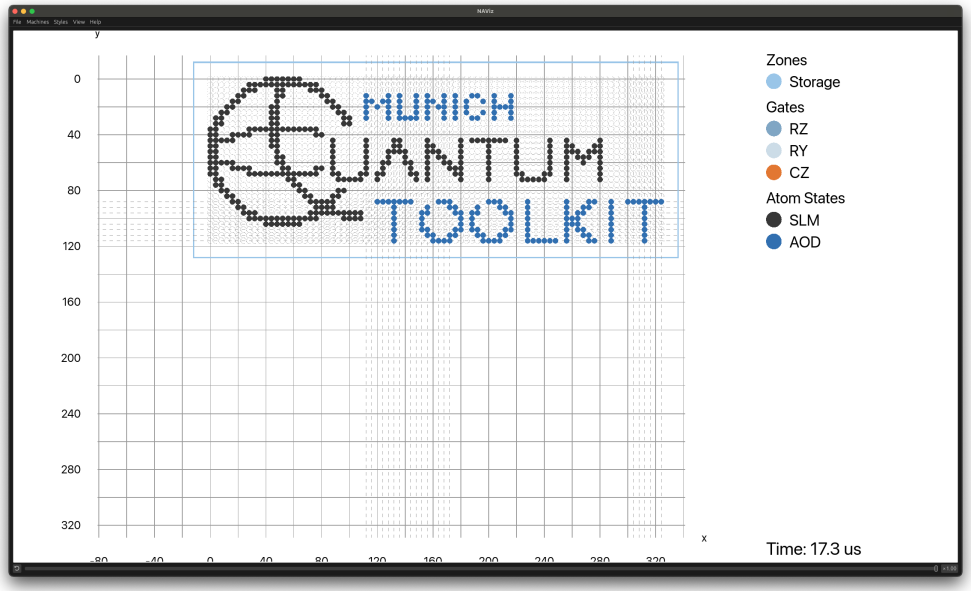
Animate the Output



MQT NAViz

Just released

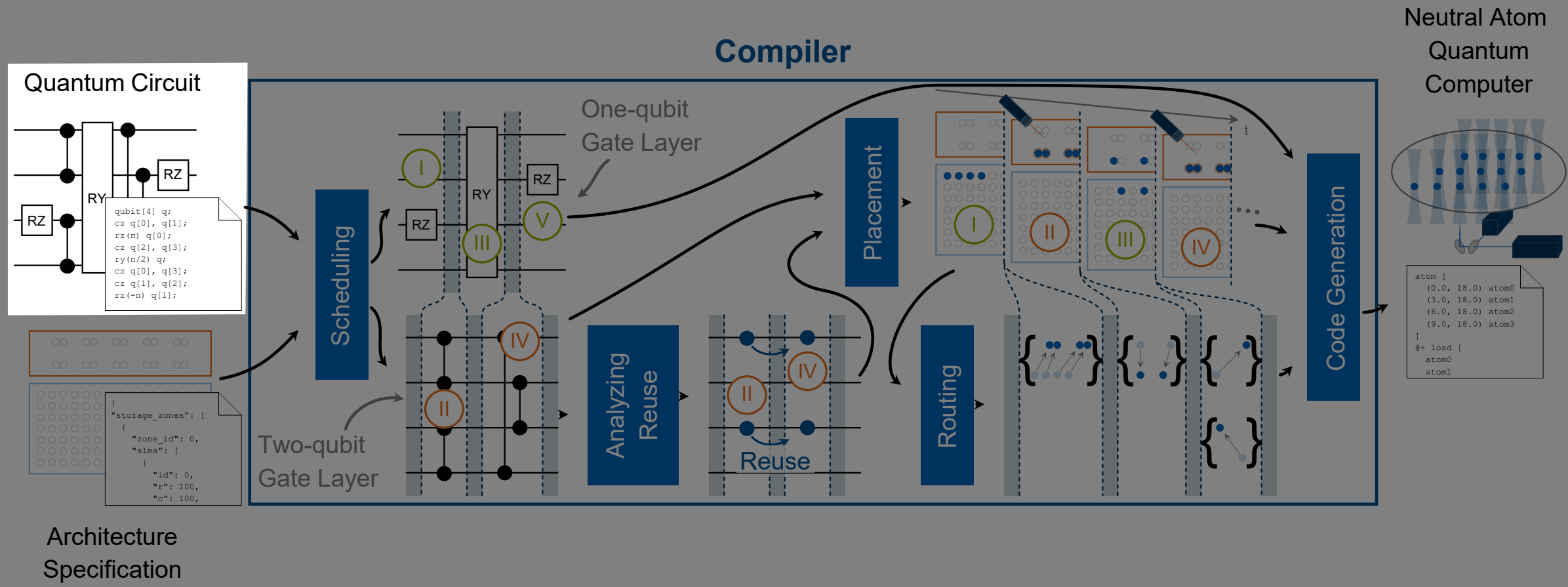
- **Instant Playback**
- **Video Export**
- **Scrubbable Timeline**



Check out the Code
github.com/munich-quantum-toolkit/naviz

Read the Docs
mqt.readthedocs.io/projects/naviz/en/latest/

Compiling for next generation hardware



Intermediate Representations and Compiler Infrastructure

“quantum-first”

OpenQASM 2

OpenQASM 3

Qiskit

MUNICH QUANTUM TOOLKIT

“classical-first”

LLVM COMPILER INFRASTRUCTURE

QIR

Kirin

Bloqade

PENNYLANE CATALYST

NVIDIA.

Building Bridges

PENNYLANE

```python
import pennylane as qml
from pennlylane.tape import QuantumTape

# Define GHZ circuit
def ghz_circuit():
  qml.Hadarmard(wires=0)
  qml.CNOT(wires=[0, 1])
  qml.CNOT(wires=[1, 2])

# Convert to OpenQASM
with QuantumTape() as tape:
  ghz_circuit()
qasm = tape.to_openqasm()

# Map circuit with QMAP
mapped_qasm = map_circuit(qasm)

# Convert back to PennyLane
mapped_ghz = qml.from_Qasm(mapped_qasm)
```

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[3];
creg c[3];
h q[0];
cx q[0], q[1];
cx q[1], q[2];
measure q -> c;
```

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[3];
creg c[3];
h q[1];
cx q[1], q[0];
cx q[1], q[2];
measure q -> c;
```

MUNICH QUANTUM TOOLKIT

```python
from mqt.core import QuantumComputation
import mqt.qmap as qmap

def map_circuit(qasm):
  # Convert OpenQASM to MQT Core IR
  mqt_qc = \
QuantumComputation.from_qasm_str(qasm)

  # Specify the architecture to map to
  n_qubits = 3
  cm = set([(0,1),(1,0),(1,2),(2,1)])
  arc = qmap.Architecture(n_qubits, cm)
  default_config = qmap.Configuration()

  # Map circuit with QMAP
  mapped_qc, _ = \
qmap.map(mqt_qc, arc, default_config)

  # Convert to OpenQASM
  return \
QuantumComputation.qasm2_str(mapped_qc)
```

Building Bridges

PENNYLANE

```python
import pennylane as qml
from pennlylane.tape import QuantumTape

# Define GHZ circuit
def ghz_circuit():
  qml.Hadarmard(wires=0)
  qml.CNOT(wires=[0, 1])
  qml.CNOT(wires=[1, 2])

# Convert to OpenQASM
with QuantumTape() as tape:
  ghz_circuit()
qasm = tape.to_openqasm()

# Map circuit with QMAP
mapped_qasm = map_circuit(qasm)

# Convert back to PennyLane
mapped_ghz = qml.from_Qasm(mapped_qasm)
```

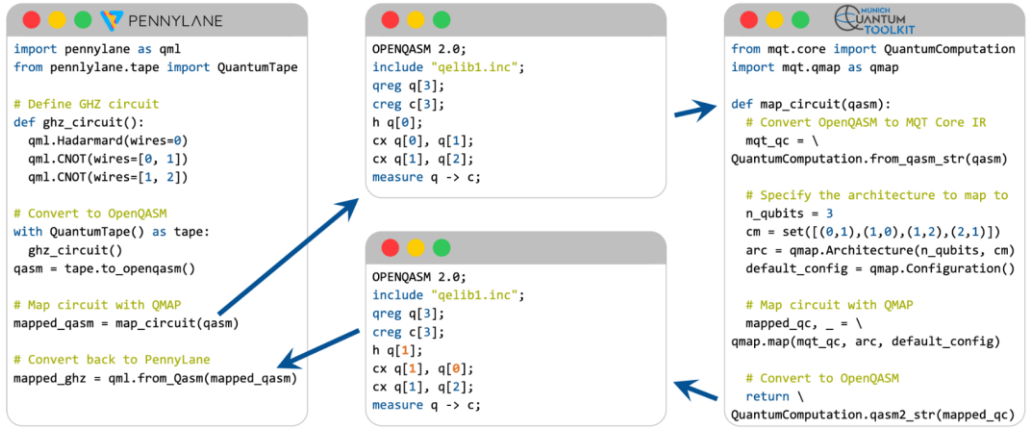
```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[3];
creg c[3];
h q[0];
cx q[0], q[1];
cx q[1], q[2];
measure q -> c;
```

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[3];
creg c[3];
h q[1];
cx q[1], q[0];
cx q[1], q[2];
measure q -> c;
```

MUNICH QUANTUM TOOLKIT

```python
from mqt.core import QuantumComputation
import mqt.qmap as qmap

def map_circuit(qasm):
  # Convert OpenQASM to MQT Core IR
  mqt_qc = \
QuantumComputation.from_qasm_str(qasm)

  # Specify the architecture to map to
  n_qubits = 3
  cm = set([(0,1),(1,0),(1,2),(2,1)])
  arc = qmap.Architecture(n_qubits, cm)
  default_config = qmap.Configuration()

  # Map circuit with QMAP
  mapped_qc, _ = \
qmap.map(mqt_qc, arc, default_config)

  # Convert to OpenQASM
  return \
QuantumComputation.qasm2_str(mapped_qc)
```

Building Bridges



- Over 250 commits just on the plugin itself
  github.com/munich-quantum-toolkit/core/pull/881
- Over 55 closed Issues and PRs
  github.com/munich-quantum-toolkit/core/milestone/8

PENNYLANE

```python
import pennylane as qml
from catalyst import measure
from catalyst.passes import apply_pass

@apply_pass(mqt.qmap{"cMap": [(0,1),(1,0),(1,2),(2,1)]})
@qml.qjit(target="mlir")
@qml.qnode(qml.device("lightning.qubit", wires=3))
def ghz_circuit():
  qml.Hadamard(wires=[0])
  qml.CNOT(wires=[0, 1])
  qml.CNOT(wires=[1, 2])
  return [measure(i) for i in range(3)]
```

MLIR Support in MUNICH QUANTUM TOOLKIT

MQT Core

94 ⭐

469k 📈

MQT Core
github.com/munich-quantum-toolkit/core

MQT Core Documentation
mqt.readthedocs.io/projects/core/en/latest/

Behind the Scenes

PENNYLANE

```python
import pennylane as qml
from catalyst import measure
from catalyst.passes import apply_pass

@apply_pass(mqt.qmap{"cMap": [(0,1),(1,0),(1,2),(2,1)]})
@qml.qjit(target="mlir")
@qml.qnode(qml.device("lightning.qubit", wires=3))
def ghz_circuit():
  qml.Hadamard(wires=[0])
  qml.CNOT(wires=[0, 1])
  qml.CNOT(wires=[1, 2])
  return [measure(i) for i in
```

MQTRef

```
func.func @bellState() {
  %qreg = "mqtref.allocQubitRegister"() <{size_attr = 3 : i64}> : () -> !mqtref.QubitRegister
  %q0 = "mqtref.extractQubit"(%qreg) <{index_attr = 0 : i64}> : (!mqtref.QubitRegister) -> !mqtref.Qubit
  %q1 = "mqtref.extractQubit"(%qreg) <{index_attr = 1 : i64}> : (!mqtref.QubitRegister) -> !mqtref.Qubit
  %q2 = "mqtref.extractQubit"(%qreg) <{index_attr = 2 : i64}> : (!mqtref.QubitRegister) -> !mqtref.Qubit
  mqtref.h() %q0
  mqtref.x() %q1 ctrl %q0
  mqtref.x() %q2 ctrl %q1
  %m0 = mqtref.measure %q0
  %m1 = mqtref.measure %q1
  %m2 = mqtref.measure %q2
  "mqtref.deallocQubitRegister"(%qreg) : (!mqtref.Qubi
  return
}
```

Reference Semantics

⇒ Similar to OpenQASM, QIR, ...

⇒ Good fit for In/Output

Behind the Scenes

MQTOpt

```
func.func @ghzState() {
  %reg_0 = "mqtopt.allocQubitRegister"() <{size_attr = 3 : i64}> : () -> !mqtopt.QubitRegister
  %reg_1, %q0_0 = "mqtopt.extractQubit"(%reg_0) <{index_attr = 0 : i64}> : (!mqtopt.QubitRegister) -> (!mqtopt.QubitRegister, !mqtopt.Qubit)
  %reg_2, %q1_0 = "mqtopt.extractQubit"(%reg_1) <{index_attr = 1 : i64}> : (!mqtopt.QubitRegister) -> (!mqtopt.QubitRegister, !mqtopt.Qubit)
  %reg_3, %q2_0 = "mqtopt.extractQubit"(%reg_2) <{index_attr = 2 : i64}> : (!mqtopt.QubitRegister) -> (!mqtopt.QubitRegister, !mqtopt.Qubit)
  %q0_1 = mqtopt.h() %q0_0 : !mqtopt.Qubit

  %q1_1, %q0_2 = mqtopt.x() %q1_0 ctrl %q0_1 : !mqtopt.Qubit ctrl !mqtopt.Qubit

  %q2_1, %q1_2 = mqtopt.x() %q2_0 ctrl %q1_1 : !mqtopt.Qubit ctrl !mqtopt.Qubit

  %q0_3, %m0_0 = "mqtopt.measure"(%q0_2) : (!mqtopt.Qubit) -> (!mqtopt.Qubit, i1)

  %q1_3, %m1_0 = "mqtopt.measure"(%q1_2) : (!mqtopt.Qubit) -> (!mqtopt.Qubit, i1)

  %q2_2, %m1_0 = "mqtopt.measure"(%q2_1) : (!mqtopt.Qubit) -> (!mqtopt.Qubit, i1)
  %reg_4 = "mqtopt.insertQubit"(%reg_3, %q0_3) <{index_attr = 0 : i64}> : (!mqtopt.QubitRegister, !mqtopt.Qubit) -> !mqtopt.QubitRegister
  %reg_5 = "mqtopt.insertQubit"(%reg_4, %q1_3) <{index_attr = 1 : i64}> : (!mqtopt.QubitRegister, !mqtopt.Qubit) -> !mqtopt.QubitRegister
  %reg_6 = "mqtopt.insertQubit"(%reg_5, %q2_2) <{index_attr = 2 : i64}> : (!mqtopt.QubitRegister, !mqtopt.Qubit) -> !mqtopt.QubitRegister
 "mqtopt.deallocQubitRegister"(%reg_6) : (!mqtopt.QubitRegister) -> ()
  return
}
```

Value Semantics

⇒ Makes Dataflow apparent

⇒ Good for Optimizations

Key Challenges and Obstacles

Steep Learning Curve
Complex MLIR concepts
C++

Version Compatibility
Can we align LLVM versions?

Packaging & Distribution
How to ship all of this?

Structured Programs
Support beyond the base profile of QIR

Uniform Exchange Format
Can we agree on some common format?

JEFF BRIDGES COMPILATION

github.com/unitaryfoundation/jeff

Benchmarking

Software Tool Evaluation

Measure compiler performance, validate optimization passes, stress-test simulators, and more.

Hardware Performance

Benchmark real quantum devices on diverse applications—compare noise resilience, fidelity and execution speed.

AI Training Datasets

Access a library of scalable quantum circuits to power machine learning models and data-driven optimizations.

MQT Bench GitHub
github.com/
munich-quantum-toolkit/bench

Shell

```
$ (uv) pip install mqt.bench
```

100 ⭐

102k

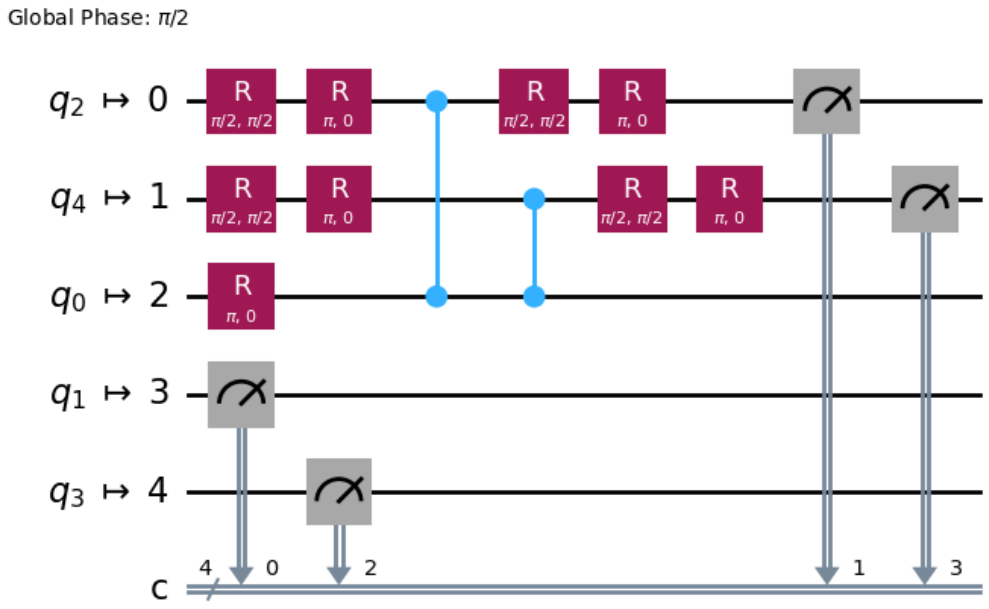
Qiskit Ecosystem

PLANQK 21

Benchmarking: Software Tool Evaluation

optimizer_eval.py

```python
from mqt.bench import BenchmarkLevel, get_benchmark
from mqt.bench.targets import get_device
from your.package import awesome_optimizer

# Bernstein-Vazirani circuit on the algorithmic level
circuit = get_benchmark(benchmark="bv", level=BenchmarkLevel.ALG, circuit_size=5)
circuit.draw()

# Optimize using your developed algorithm
optimized_circuit = awesome_optimizer(circuit)
optimized_circuit.draw()

# Map to IQM Crystal (5 Qubits) to evaluate
mapped_circuit = get_benchmark(
    benchmark=optimized_circuit,
    level=BenchmarkLevel.MAPPED,
    circuit_size=5,
    target=get_device("iqm_crystal_5"),
)
mapped_circuit.draw()
```



MQT Bench GitHub
github.com/
munich-quantum-toolkit/bench

Shell

```
$ (uv) pip install mqt.bench
```

100 ⭐

102k

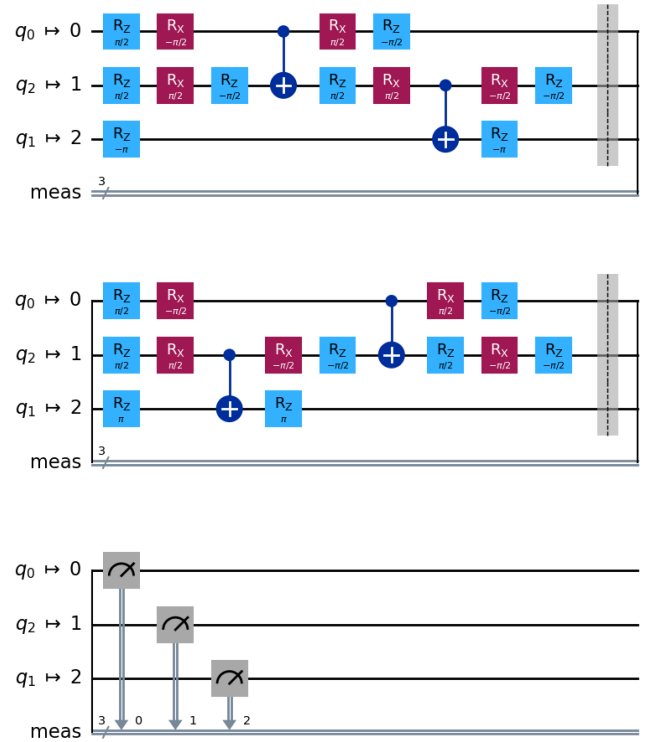
Qiskit Ecosystem

PLANQK 22

Benchmarking: Hardware Performance

hardware_performance.py

```python
from mqt.bench import BenchmarkLevel, get_benchmark
from qiskit.transpiler import InstructionProperties, Target
from qiskit.circuit import Parameter
from qiskit.circuit.library import RXGate, RZGate

# Custom target
custom_target = Target(num_qubits=3, description="custom_target")
# Single qubit properties
single_qubit_props = InstructionProperties(duration=1e-3, error=1e-4)
properties = {(0,): single_qubit_props, (1,): single_qubit_props,
              (2,): single_qubit_props}
custom_target.add_instruction(RXGate(Parameter("alpha")), properties=properties)
custom_target.add_instruction(RZGate(Parameter("beta")), properties=properties)
# Add two qubit properties, measurement etc.
...

# Map Deutsch-Jozsa to custom target and append mirrored version
mapped_circuit = get_benchmark("dj", BenchmarkLevel.MAPPED, circuit_size=3,
    opt_level=3, target=custom_target, generate_mirror_circuit=True)
mapped_circuit.draw()
```



MQT Bench GitHub
github.com/
munich-quantum-toolkit/bench

Shell

```
$ (uv) pip install mqt.bench
```

100 ⭐

102k

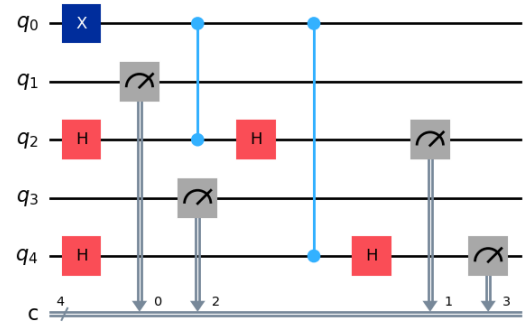
Qiskit Ecosystem

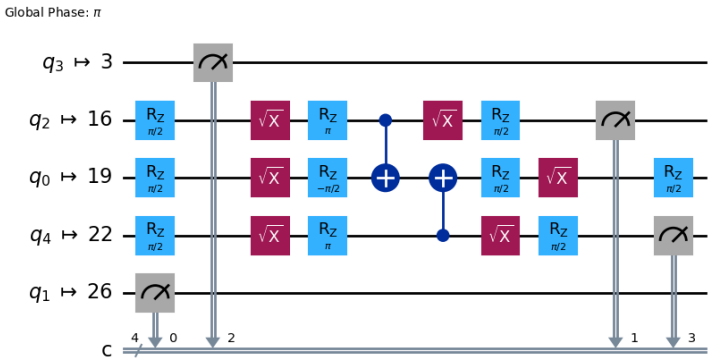
PLANQK

Benchmarking: AI Training Datasets

ai_training.py

```python
from mqt.bench import BenchmarkLevel, get_benchmark
from mqt.predictor import qcompile

# Get uncompiled Bernstein-Vazirani circuit
qc = get_benchmark("bv", level=BenchmarkLevel.INDEP, circuit_size=5)
qc.draw()

# Compile using MQT Predictor (which has been trained using circuits from MQT Bench)
qc_compiled, compilation_information, quantum_device = qcompile(qc)

# Predicted device
print(quantum_device)    # IBM Falcon (27 Qubits)

# Print compiled circuit
qc_compiled.draw()
```





MQT Bench GitHub
github.com/
munich-quantum-toolkit/bench

Shell

```
$ (uv) pip install mqt.bench
```

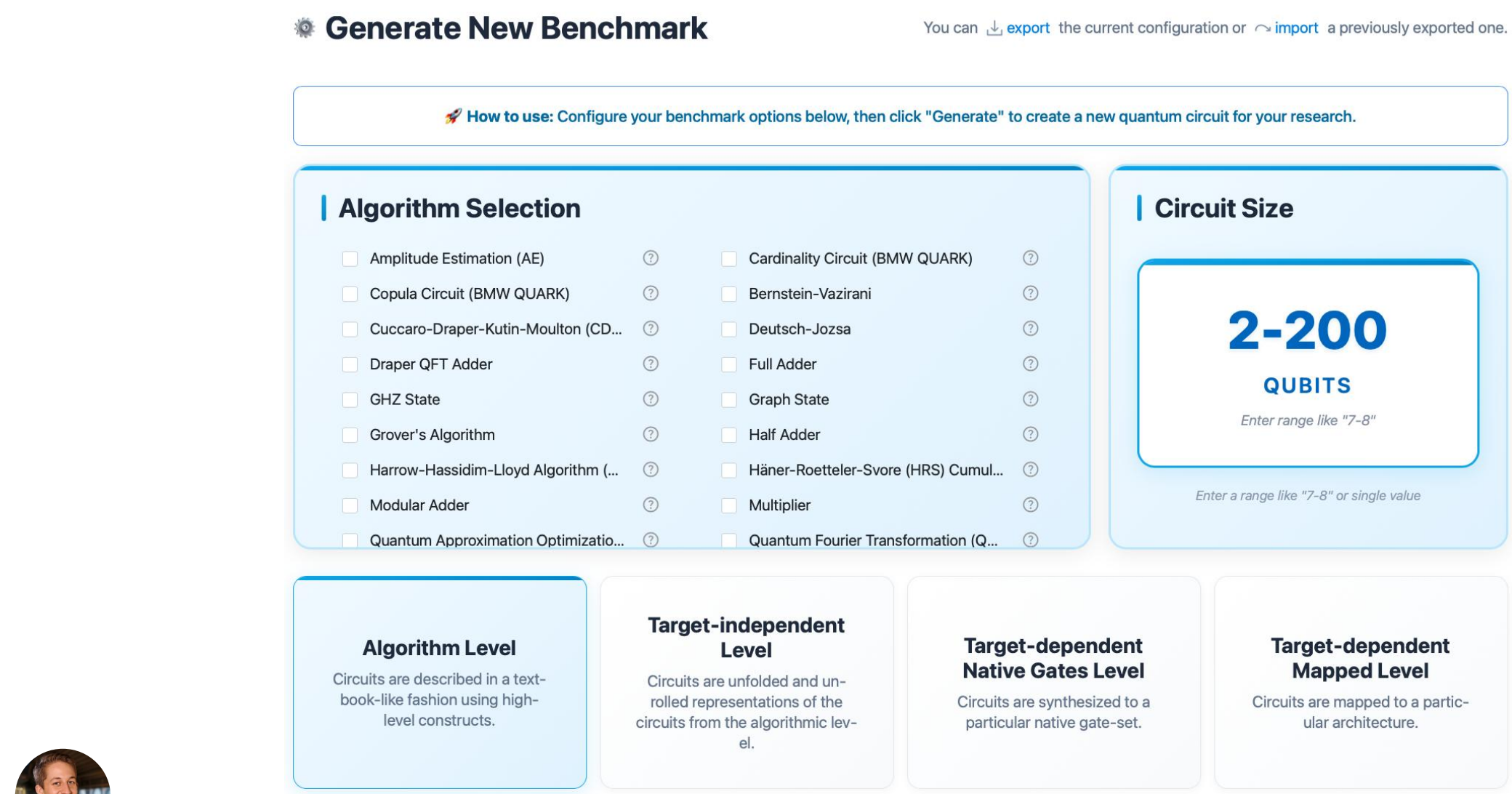
100 ⭐

102k

Qiskit Ecosystem

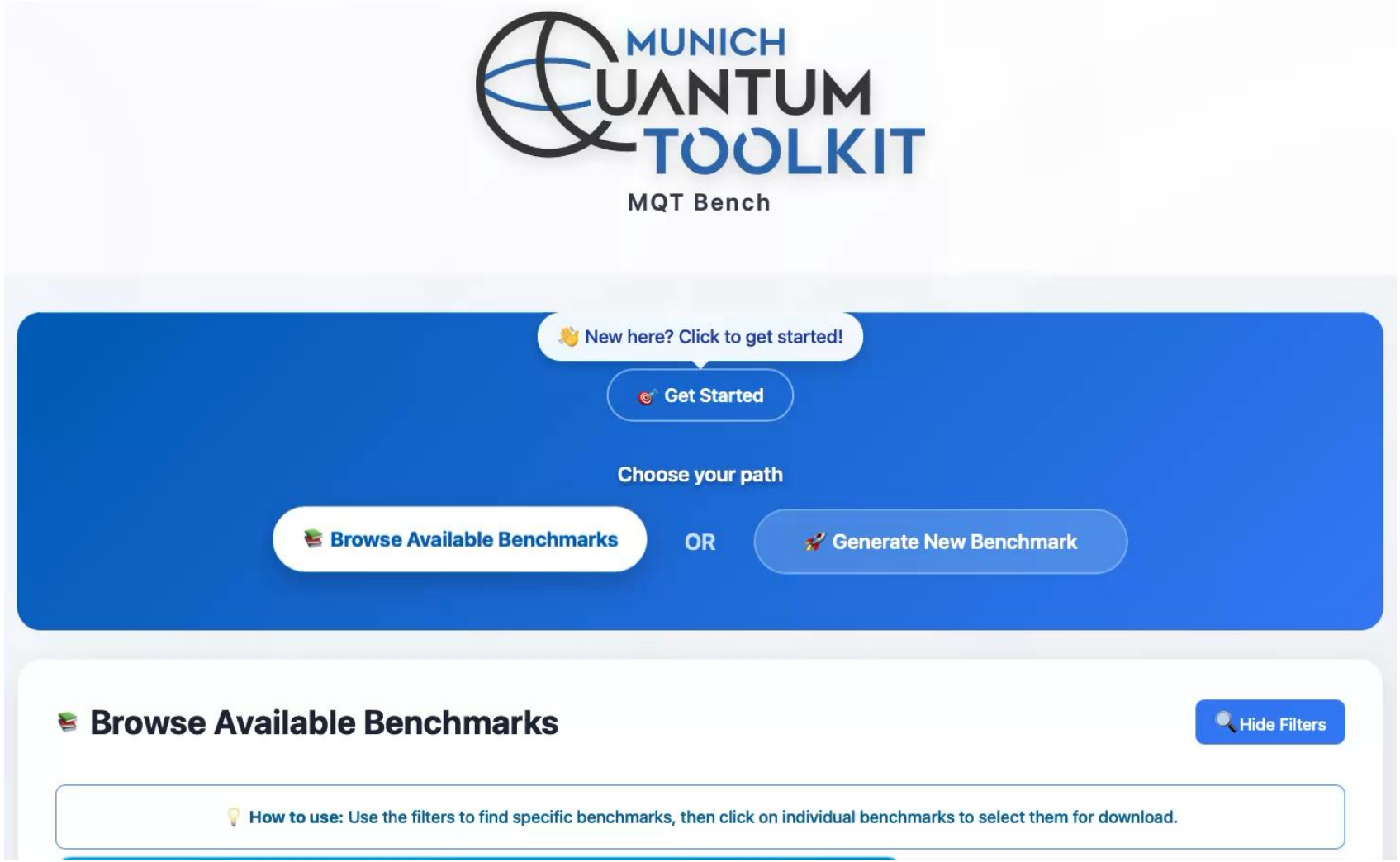
PLANQK

MQT Bench: Frontend Update



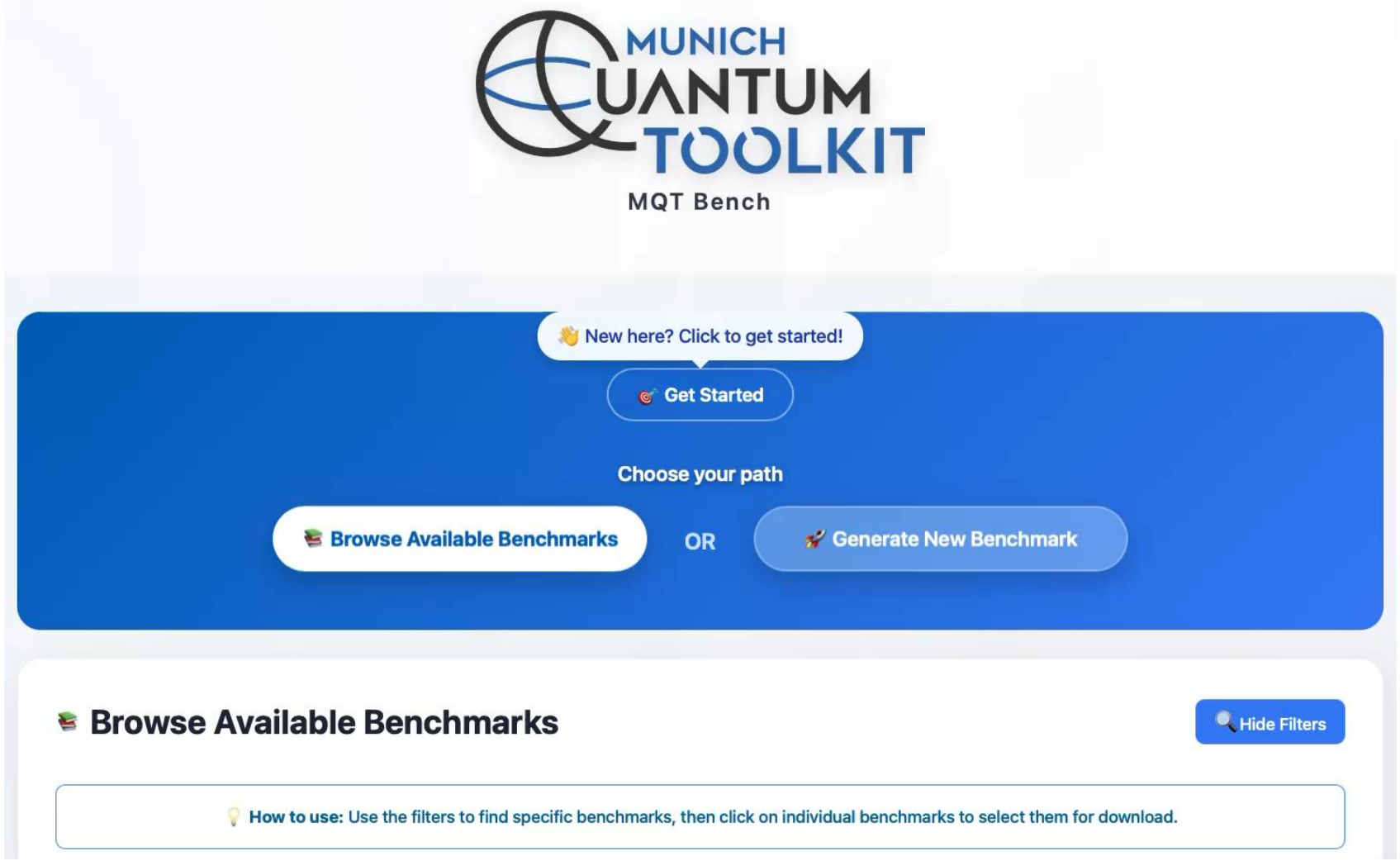
MQT-BENCH.APP

MQT Bench: Benchmark Browser

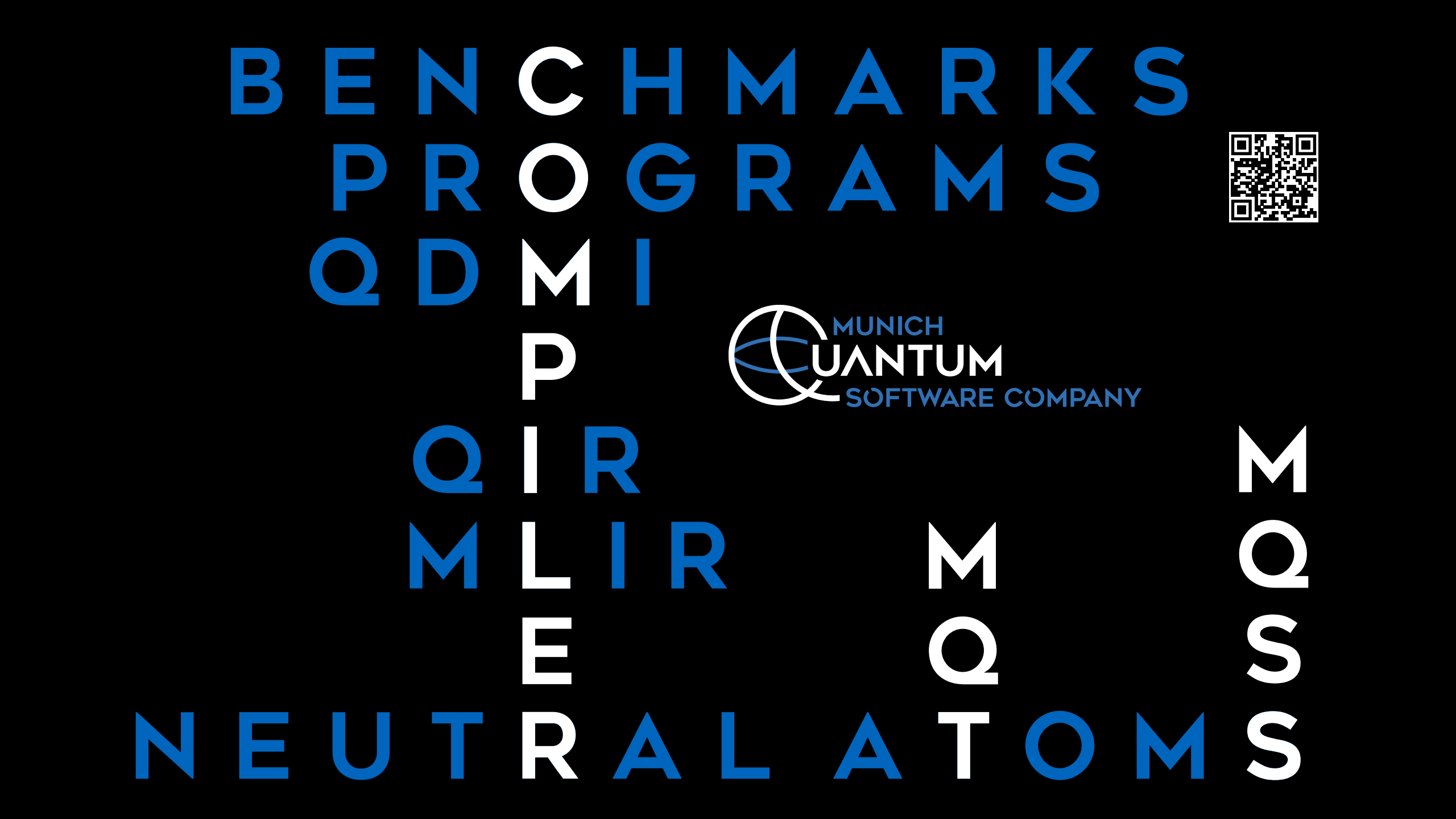


MQT-BENCH.APP

MQT Bench: Benchmark Generation



MQT-BENCH.APP


BENCHMARKS
PROGRAMS
QDMI
QIR
MLIR
NEUTRAL ATOMS
MQSS
MUNICH QUANTUM SOFTWARE COMPANY