



MQSF 25

Beyond Circuits

Compiling the next generation of quantum programs with the MQT

Lukas Burgholzer

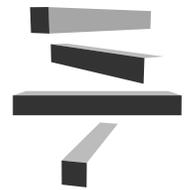
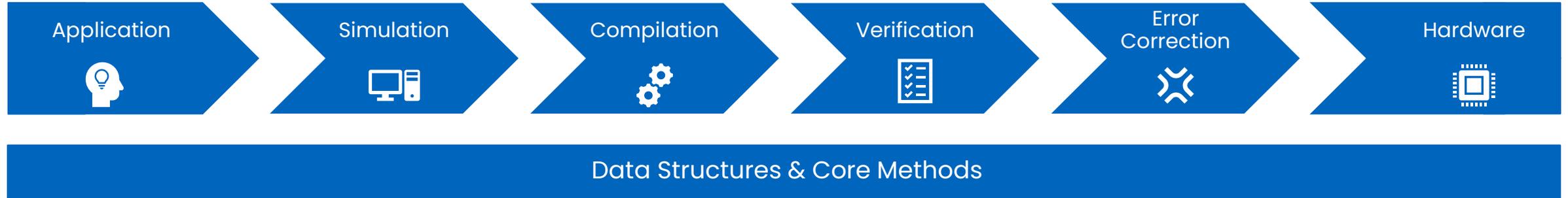
lukas@munichquantum.software





Software for Quantum Computing

Needed across the whole spectrum...

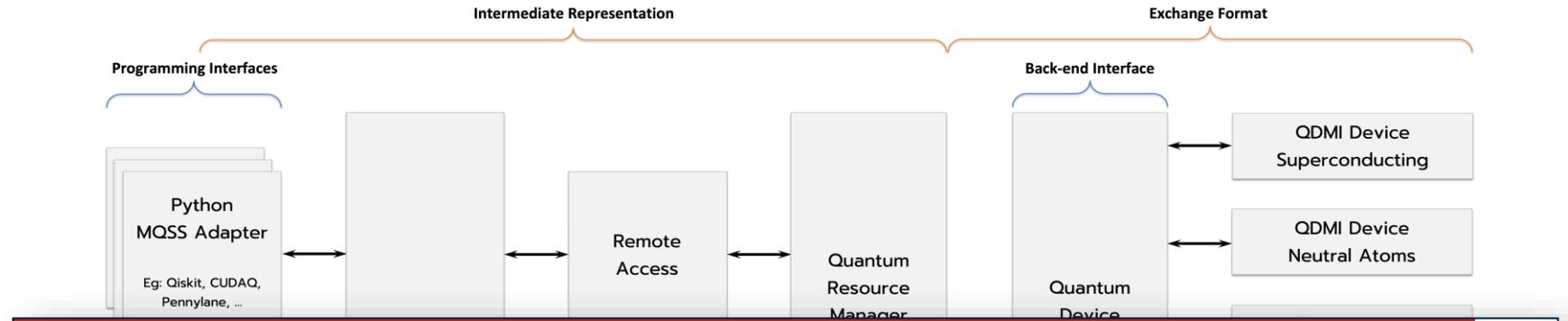


MQSS
Munich Quantum
Software Stack



Overview Paper

arxiv.org/abs/2509.02674



The screenshot shows the arXiv abstract page for the paper "The Munich Quantum Software Stack: Connecting End Users, Integrating Diverse Quantum Technologies, Accelerating HPC". The paper is in the "Quantum Physics" category and was submitted on 2 Sep 2025. The authors listed are Lukas Burgholzer, Jorge Echavarría, Patrick Hopf, Yannick Stade, Damian Rovara, Ludwig Schmid, Ercüment Kaya, Burak Mete, Muhammad Nufail Farooqi, Minh Chung, Marco De Pascale, Laura Schulz, Martin Schulz, and Robert Wille. The logo for the Institute of Science and Technology (IST) is visible in the bottom right corner.

The Munich Quantum Toolkit (MQT)



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 	MQT Bench Application Benchmarking Software and Tools for Quantum Computing mqt-bench.app munich-quantum-toolkit/bench 	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/quosat 	MQT YAQS Simulation A Tool for Simulating Open Quantum Systems, Noisy Quantum Circuits, and Realistic Quantum Hardware munich-quantum-toolkit/yaqs
MQT DDSIM Simulation A Tool for Classical Quantum Circuit Simulation based on Decision Diagrams munich-quantum-toolkit/ddsim 	MQT Predictor Compilation A Tool for Determining Good Quantum Circuit Compilation Options munich-quantum-toolkit/predictor 	MQT IonShuttler Compilation A Tool for Generating Shuttling Schedules for QCCD Architectures munich-quantum-toolkit/ionshuttler 	MQT Qudits Compilation A Tool for Compiling High-Dimensional Quantum Systems munich-quantum-toolkit/qudits
MQT SyReC Compilation A Tool for the Synthesis of Reversible Circuits/Quantum Computing Oracles munich-quantum-toolkit/syrec 	MQT QMAP Compilation A Tool for Quantum Circuit Mapping And Clifford Circuit Optimization/Synthesis munich-quantum-toolkit/qmap 	MQT QCEC Verification A Tool for Quantum Circuit Equivalence Checking munich-quantum-toolkit/qcec 	MQT Debugger Verification A semi-automated tool for debugging quantum programs munich-quantum-toolkit/debugger
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 	MQT NAViz Core Methods A visualization software for neutral atom quantum computers. munich-quantum-toolkit/naviz 	MQT QECC QECC A Tool for Quantum Error Correcting Codes munich-quantum-toolkit/qecc

<https://mqt.readthedocs.io>



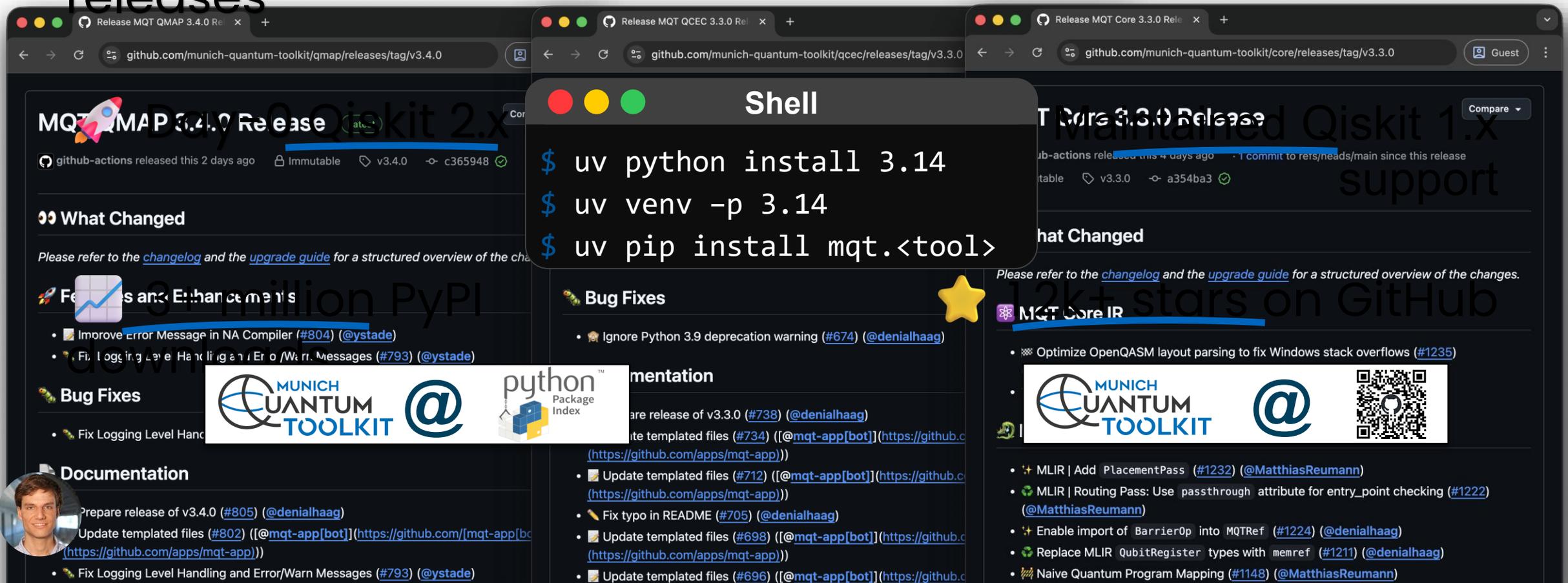
<https://github.com/munich-quantum-toolkit>



Recent Updates from the MQT

 12 new releases

 Day-0 Python 3.14 support



The image shows three overlapping GitHub release pages for MQT QMAP 3.4.0, MQT QCEC 3.3.0, and MQT Core 3.3.0. A central Shell terminal window displays the following commands:

```
$ uv python install 3.14
$ uv venv -p 3.14
$ uv pip install mqt.<tool>
```

Key features and updates from the releases include:

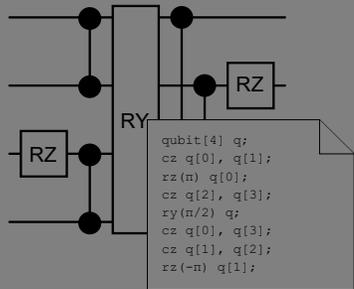
- MQT QMAP 3.4.0 Release:** Improved error message in NA Compiler (#804) (@ystade), Fixed logging level handling and error/warn messages (#793) (@ystade).
- MQT QCEC 3.3.0 Release:** Ignored Python 3.9 deprecation warning (#674) (@denialhaag), Updated templated files (#738) (@denialhaag), Updated templated files (#734) ([@mqt-app[bot]]), Updated templated files (#712) ([@mqt-app[bot]]), Fixed typo in README (#705) (@denialhaag), Updated templated files (#698) ([@mqt-app[bot]]), Updated templated files (#696) ([@mqt-app[bot]]).
- MQT Core 3.3.0 Release:** Optimized OpenQASM layout parsing to fix Windows stack overflows (#1235), Added MLIR | Add PlacementPass (#1232) (@MatthiasReumann), Updated MLIR | Routing Pass: Use passthrough attribute for entry_point checking (#1222) (@MatthiasReumann), Enabled import of BarrierOp into MQTRef (#1224) (@denialhaag), Replaced MLIR QubitRegister types with memref (#1211) (@denialhaag), Added Naive Quantum Program Mapping (#1148) (@MatthiasReumann).

Additional information includes a badge for '12k+ stars on GitHub' and a QR code linking to the Munich Quantum Toolkit.

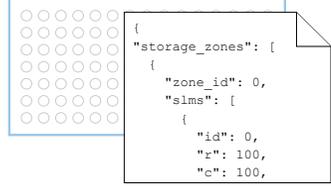
Compiling for Next Generation Hardware

Compiler

Quantum Circuit



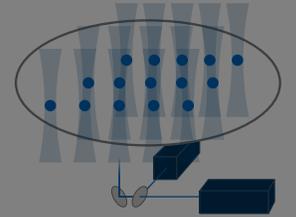
```
qubit[4] q;
cz q[0], q[1];
xz(m) q[0];
cz q[2], q[3];
zy(n/2) q;
cz q[0], q[3];
cz q[1], q[2];
xz(-n) q[1];
```



Architecture Specification



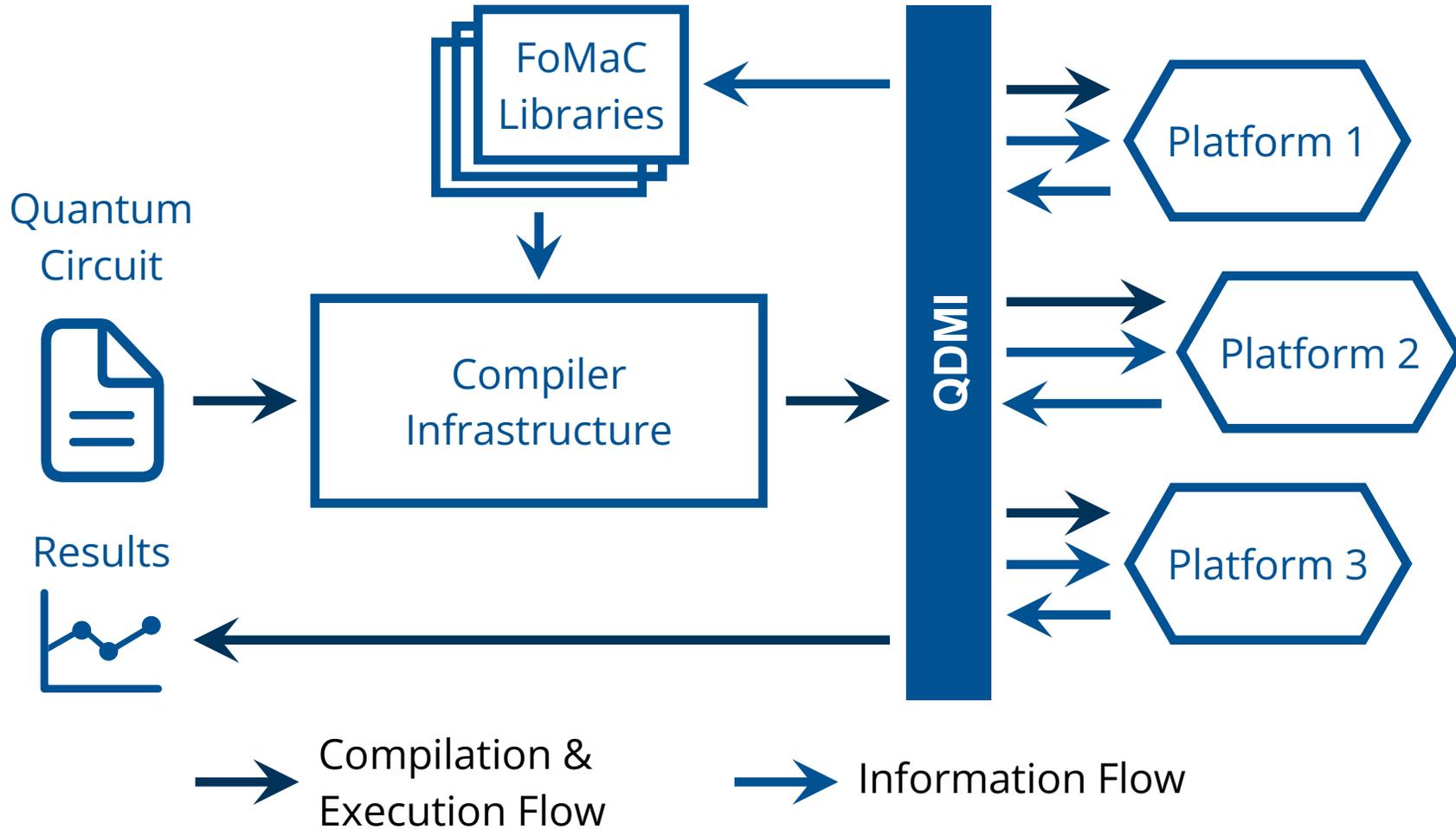
Neutral Atom Quantum Computer



```
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

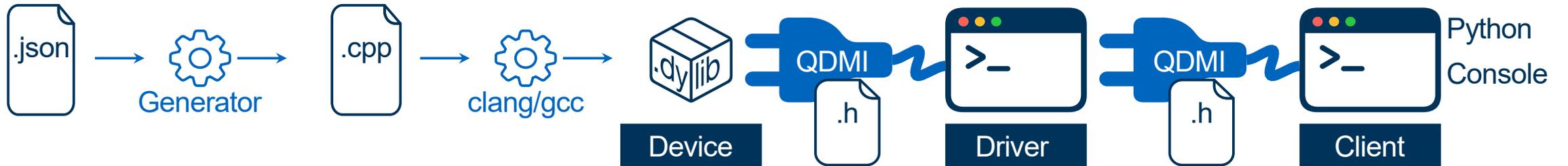
github.com/munich-quantum-software-stack/QDMI



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



Extending QDMI Device for Neutral Atoms



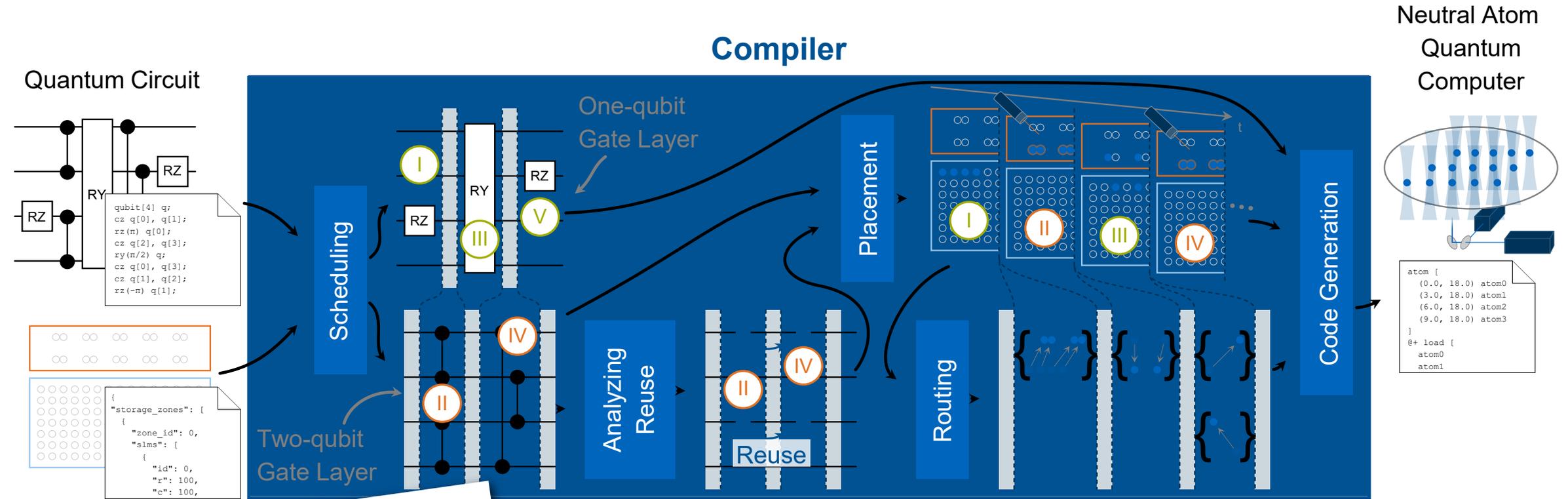
```

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



Architecture Specification



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



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
...

```



MQT QMAP

124 ★

517k

```

Shell
$ (uv) pip install mqt.qmap

```



MQT QMAP GitHub

github.com/munich-quantum-toolkit/qmap

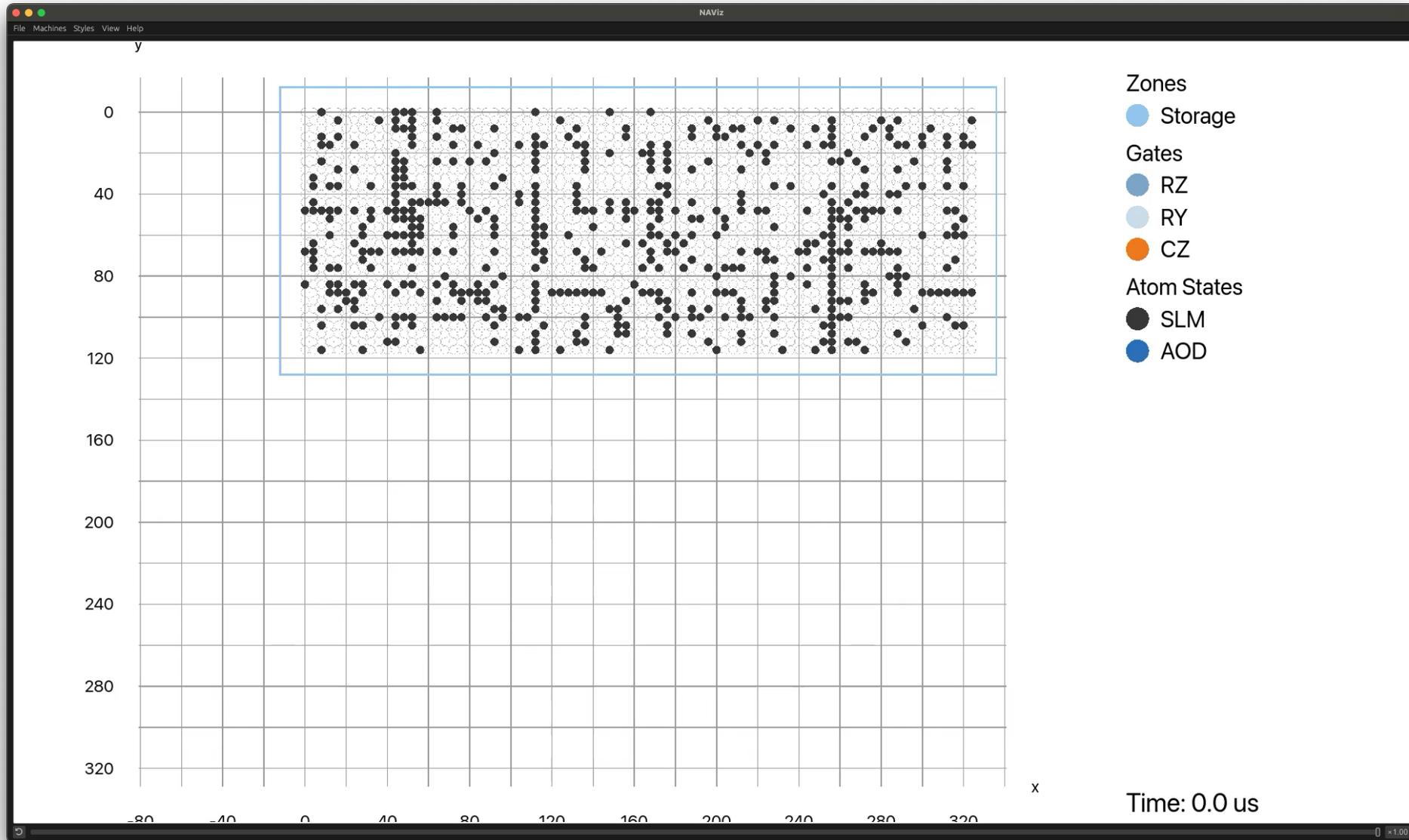


MQT QMAP Documentation with Example

mqt.readthedocs.io/projects/qmap/en/latest/



Animate the Output



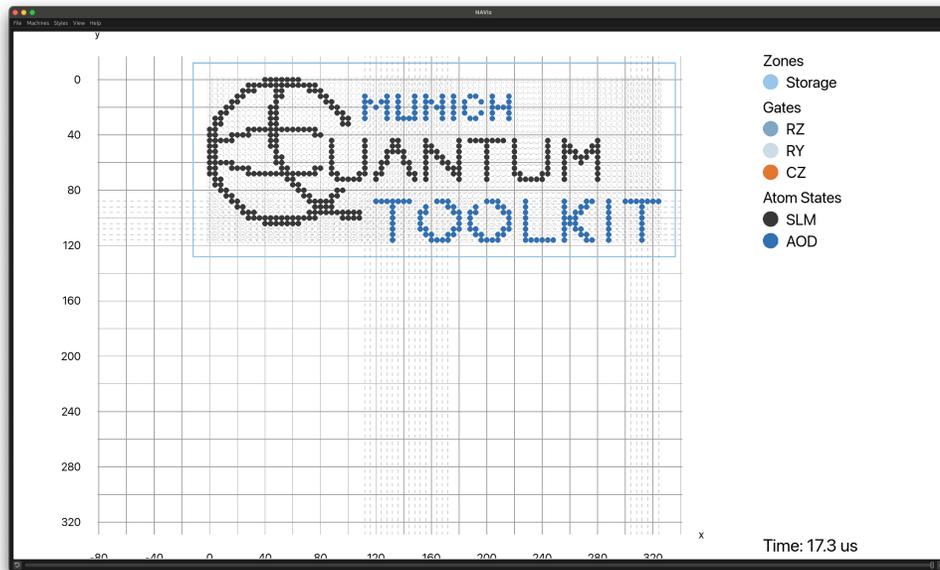
Just released ★

MQT NAViz

 **Instant Playback**

 **Video Export**

 **Scrubbable Timeline**



Check out the Code

github.com/munich-quantum-toolkit/naviz

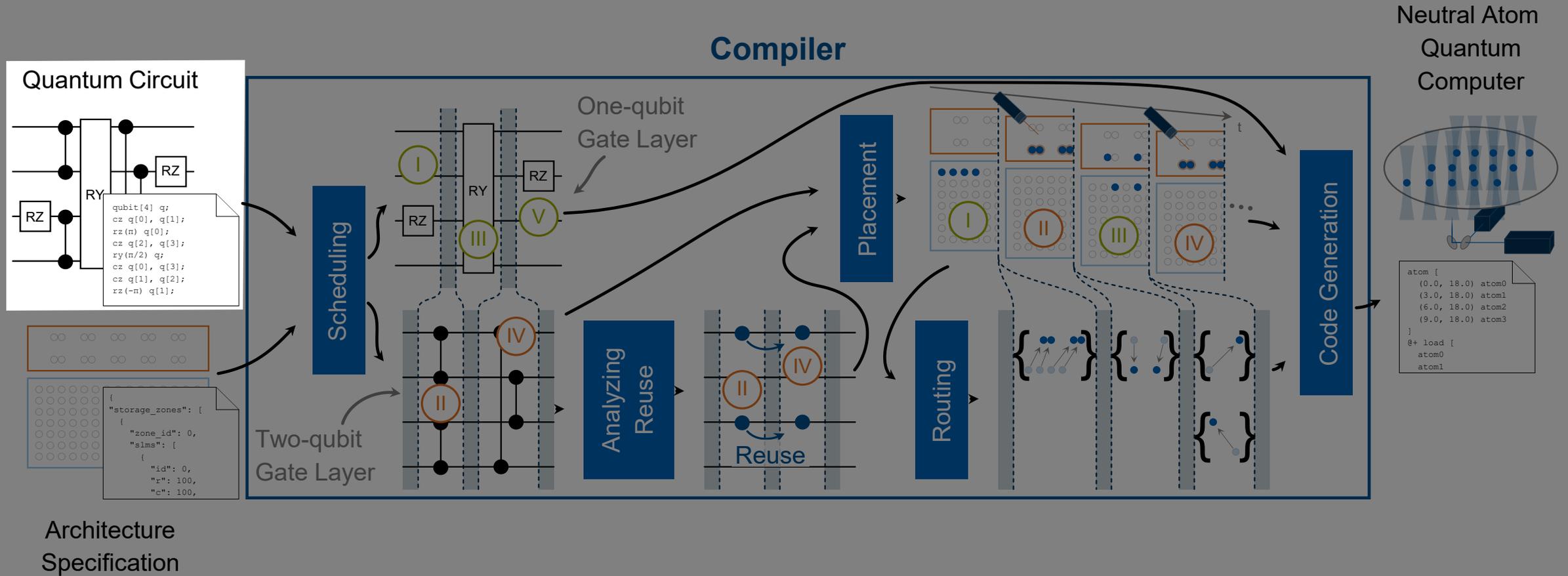


Read the Docs

mqt.readthedocs.io/projects/naviz/en/latest/



Compiling for next generation hardware



Intermediate Representations and Compiler Infrastructure



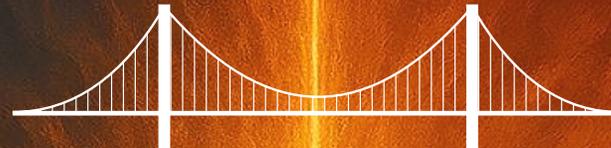
“quantum-first”

OpenQASM 2

OpenQASM 3



“classical-first”



Building Bridges

```

PENNYLANE

import pennylane as qml
from pennylane.tape import QuantumTape

# Define GHZ circuit
def ghz_circuit():
    qml.Hadamard(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

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

```

import pennylane as qml
from pennylane.tape import QuantumTape

# Define GHZ circuit
def ghz_circuit():
    qml.Hadamard(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;

```

```

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()

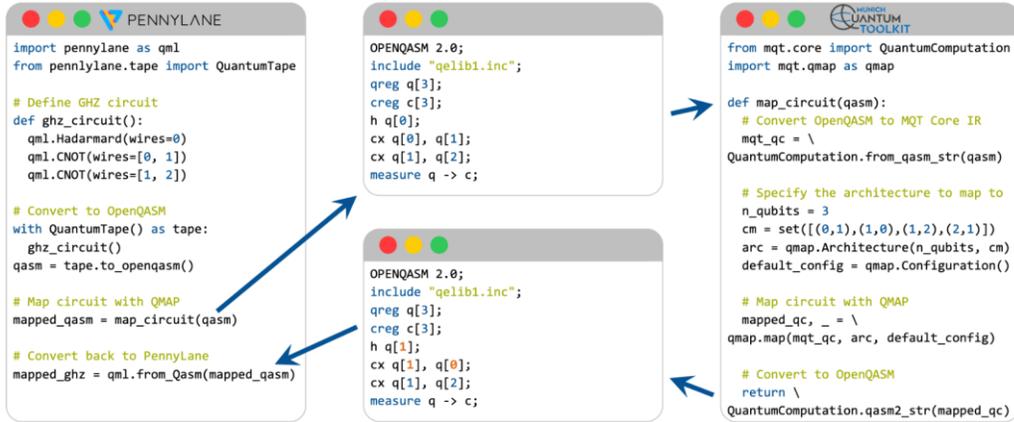
    # Map circuit with QMAP
    mapped_qc, _ = \
    qmap.map(mqt_qc, arc, default_config)

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

```



Building Bridges



```

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)]
    
```

- 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

MLIR Support in



MQT Core

94 ★ 469k



MQT Core

github.com/munich-quantum-toolkit/core



MQT Core Documentation

mqt.readthedocs.io/projects/core/en/latest/



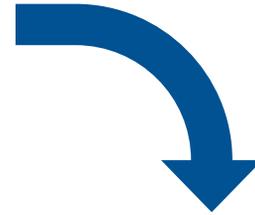
Behind the Scenes

```

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

```



```

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.QubitRegister)
    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](https://github.com/munich-quantum-toolkit/bench)

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

100 ★

102k



Qiskit
Ecosystem

PLAN Q K 21

Benchmarking: Software Tool Evaluation

```

optimizer_eval.py
from mqt.bench import BenchmarkLevel, get_benchmark
from mqt.bench.targets import get_device
from your.package import awesome_optimizer

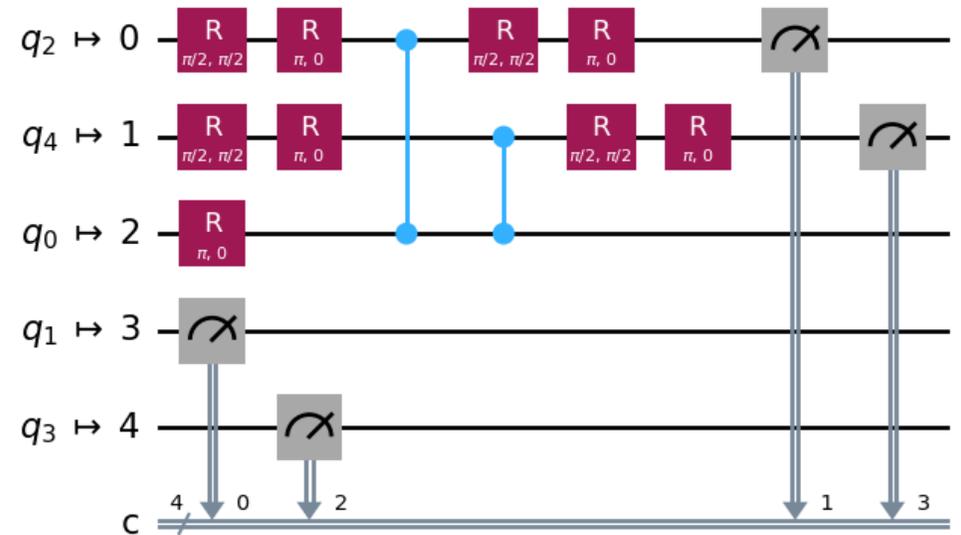
# Bernstein-Vazirani circuit on the algorithmic level
circuit = get_benchmark(benchmark="bv", level=BenchmarkLevel.ALG, circuit_size=5)
circuit.draw()

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

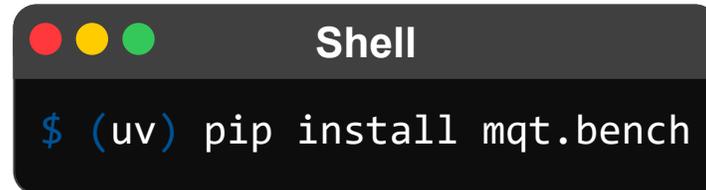
# Map to IQM Crystal (5 Qubits) to evaluate
mapped_circuit = get_benchmark(
    benchmark=optimized_circuit,
    level=BenchmarkLevel.MAPPED,
    circuit_size=5,
    target=get_device("iqm_crystal_5"),
)
mapped_circuit.draw()

```

Global Phase: $\pi/2$



MQT Bench GitHub
[github.com/
munich-quantum-toolkit/bench](https://github.com/munich-quantum-toolkit/bench)



100 ★

102k



Benchmarking: Hardware Performance

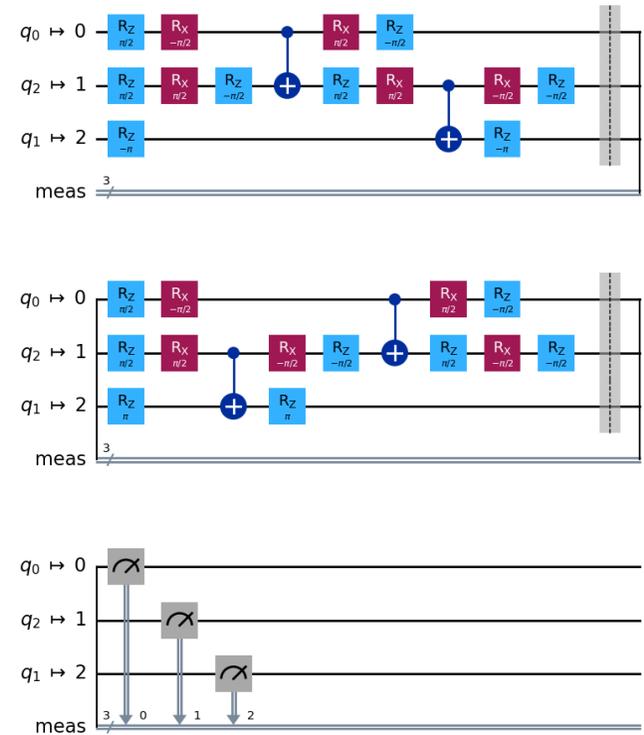
```

hardware_performance.py
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

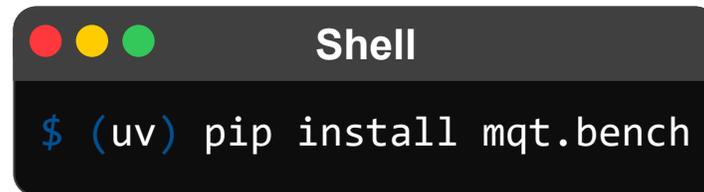
# Custom target
custom_target = Target(num_qubits=3, description="custom_target")
# Single qubit properties
single_qubit_props = InstructionProperties(duration=1e-3, error=1e-4)
properties = {(0,): single_qubit_props, (1,): single_qubit_props,
              (2,): single_qubit_props}
custom_target.add_instruction(RXGate(Parameter("alpha")), 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](https://github.com/munich-quantum-toolkit/bench)



100 ★
102k

Qiskit
Ecosystem
PLAN Q K 23

Benchmarking: AI Training Datasets

```

ai_training.py
from mqt.bench import BenchmarkLevel, get_benchmark
from mqt.predictor import qcompile

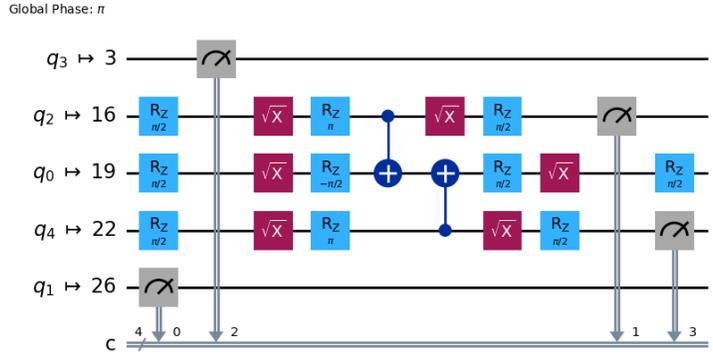
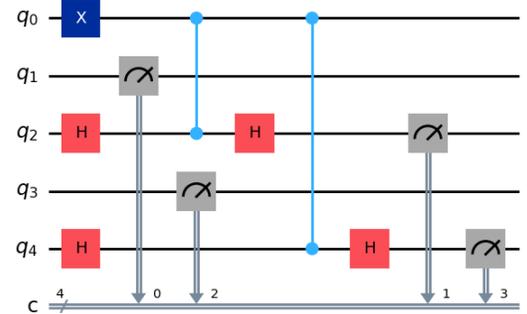
# Get uncompiled Bernstein-Vazirani circuit
qc = get_benchmark("bv", level=BenchmarkLevel.INDEP, circuit_size=5)
qc.draw()

# 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 



MQT Bench: Frontend Update

⚙️ Generate New Benchmark

You can [↓ export](#) the current configuration or [↶ import](#) a previously exported one.

 **How to use:** Configure your benchmark options below, then click "Generate" to create a new quantum circuit for your research.

| Algorithm Selection

- | | |
|--|--|
| <input type="checkbox"/> Amplitude Estimation (AE) ? | <input type="checkbox"/> Cardinality Circuit (BMW QUARK) ? |
| <input type="checkbox"/> Copula Circuit (BMW QUARK) ? | <input type="checkbox"/> Bernstein-Vazirani ? |
| <input type="checkbox"/> Cuccaro-Draper-Kutin-Moulton (CD...) ? | <input type="checkbox"/> Deutsch-Jozsa ? |
| <input type="checkbox"/> Draper QFT Adder ? | <input type="checkbox"/> Full Adder ? |
| <input type="checkbox"/> GHZ State ? | <input type="checkbox"/> Graph State ? |
| <input type="checkbox"/> Grover's Algorithm ? | <input type="checkbox"/> Half Adder ? |
| <input type="checkbox"/> Harrow-Hassidim-Lloyd Algorithm (... ? | <input type="checkbox"/> Häner-Roetteler-Svore (HRS) Cumul... ? |
| <input type="checkbox"/> Modular Adder ? | <input type="checkbox"/> Multiplier ? |
| <input type="checkbox"/> Quantum Approximation Optimizatio... ? | <input type="checkbox"/> Quantum Fourier Transformation (Q... ? |

| Circuit Size

2-200
QUBITS

Enter range like "7-8"

Enter a range like "7-8" or single value

MQT-BENCH.APP



Algorithm Level

Circuits are described in a text-book-like fashion using high-level constructs.

Target-independent Level

Circuits are unfolded and unrolled representations of the circuits from the algorithmic level.

Target-dependent Native Gates Level

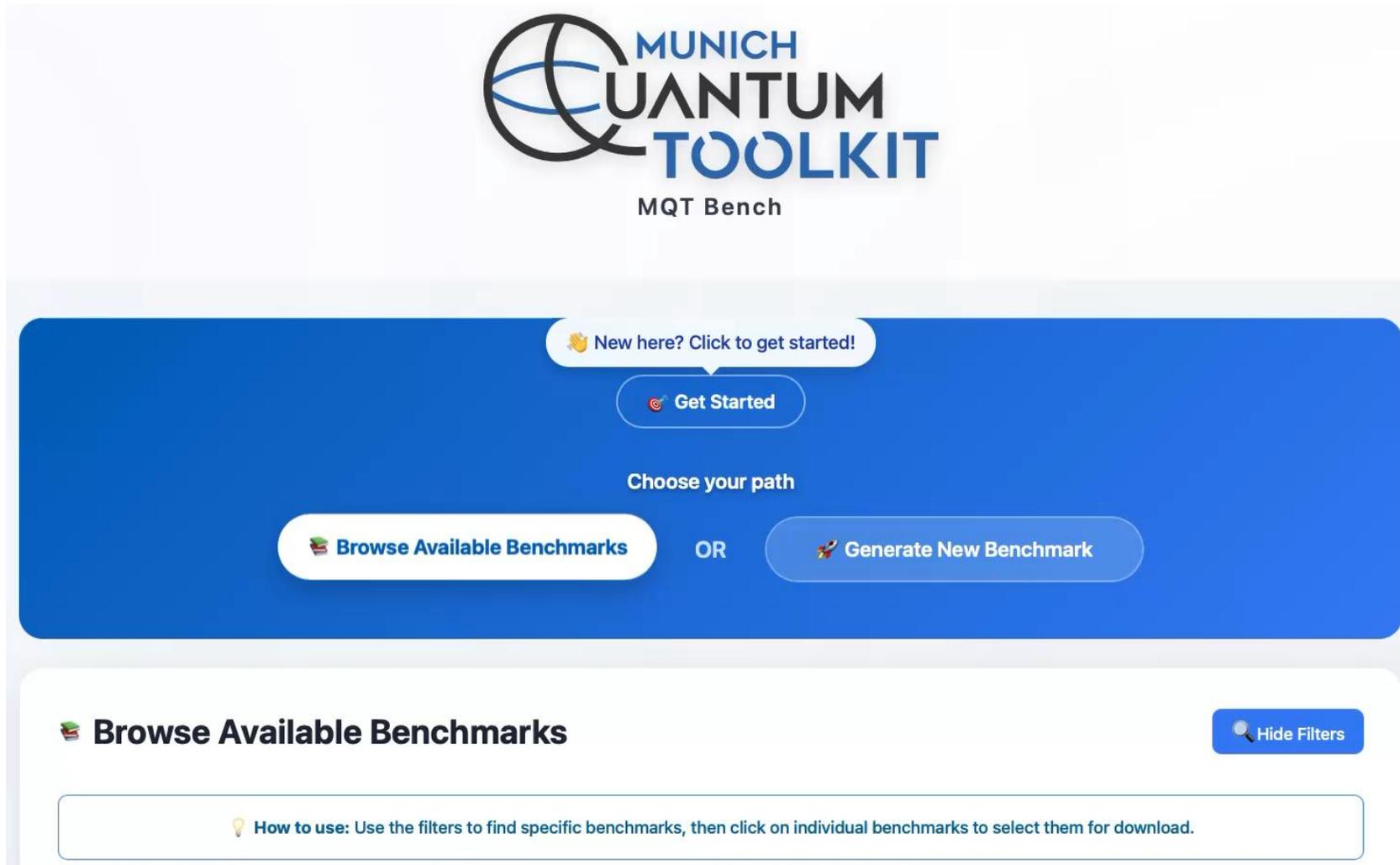
Circuits are synthesized to a particular native gate-set.

Target-dependent Mapped Level

Circuits are mapped to a particular architecture.



MQT Bench: Benchmark Browser

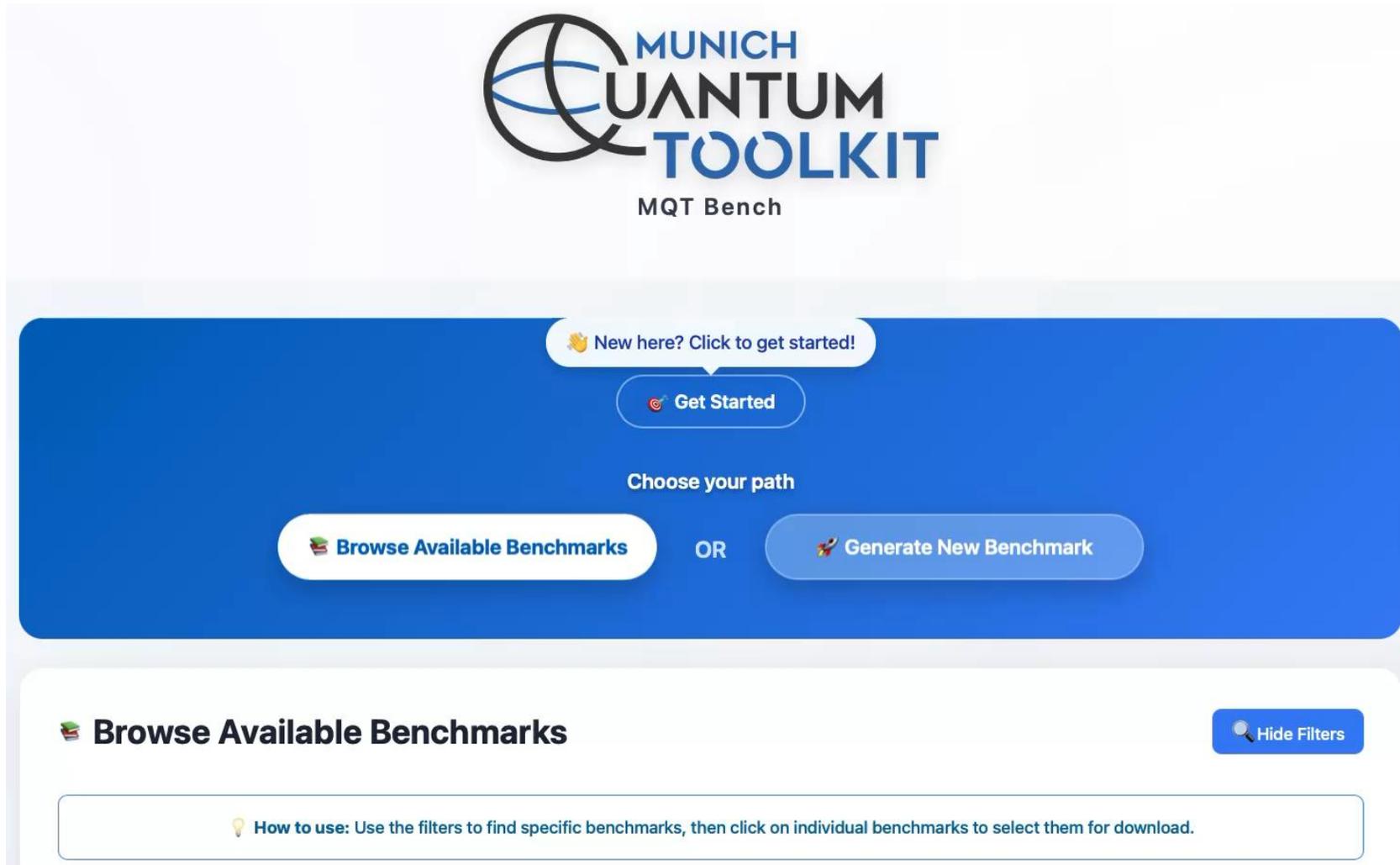


The interface features the Munich Quantum Toolkit logo at the top center, with 'MQT Bench' written below it. A large blue banner contains a call-to-action: 'New here? Click to get started!' with a 'Get Started' button. Below this, it says 'Choose your path' and offers two options: 'Browse Available Benchmarks' and 'Generate New Benchmark', separated by 'OR'. The 'Browse Available Benchmarks' section is active, showing a 'Browse Available Benchmarks' header with a 'Hide Filters' button. A lightbulb icon indicates a tip: 'How to use: Use the filters to find specific benchmarks, then click on individual benchmarks to select them for download.'

MQT-BENCH.APP



MQT Bench: Benchmark Generation



The screenshot shows the MQT Bench web interface. At the top center is the 'MUNICH QUANTUM TOOLKIT' logo with 'MQT Bench' written below it. A blue banner contains a call-to-action: 'New here? Click to get started!' with a 'Get Started' button. Below this, it says 'Choose your path' and offers two options: 'Browse Available Benchmarks' and 'Generate New Benchmark', separated by 'OR'. The 'Browse Available Benchmarks' section is active, showing a 'Browse Available Benchmarks' header with a 'Hide Filters' button. A lightbulb icon indicates a tip: 'How to use: Use the filters to find specific benchmarks, then click on individual benchmarks to select them for download.'

MQT-BENCH.APP



B E N C H M A R K S

P R O G R A M S

Q D M I

P

Q I R

M L I R

E

N E U T R A L A T O M S



M

Q

S

S

M

Q