MQT Qudits: A Software Framework for Mixed-Dimensional Quantum Computing

Kevin Mato,[1,*] Martin Ringbauer,[2,†] Lukas Burgholzer,[3,‡] and Robert Wille[3,4,§]

[1] *Chair for Design Automation, Technical University of Munich, Munich, Germany*
[2] *Institut für Experimentalphysik, University of Innsbruck, Innsbruck, Austria*
[3] *Chair for Design Automation, Technical University of Munich, Munich, Germany*
[4] *Software Competence Center Hagenberg (SCCH) GmbH, Hagenberg, Austria*

Quantum computing holds great promise for surpassing the limits of classical devices in many fields. Despite impressive developments, however, current research is primarily focused on qubits. At the same time, quantum hardware based on multi-level, qudit, systems offers a range of advantages, including expanded gate sets, higher information density, and improved computational efficiency, which might play a key role in overcoming not only the limitations of classical machines but also of current qubit-based quantum devices. However, working with qudits faces challenges not only in experimental control but particularly in algorithm development and quantum software. In this work, we introduce *MQT Qudits*, an open-source tool, which, as part of the *Munich Quantum Toolkit* (MQT), is built to assist in designing and implementing applications for mixed-dimensional qudit devices. We specify a standardized language for mixed-dimension systems and discuss circuit specification, compilation to hardware gate sets, efficient circuit simulation, and open challenges. MQT Qudits is available at `github.com/cda-tum/mqt-qudits` and on pypi at `pypi.org/project/mqt.qudits/`.

I. INTRODUCTION

Quantum computing is an emerging information processing paradigm that promises to solve certain industrial and scientific problems up to exponentially faster than classical devices. This potential as spurred impressive developments in the field of quantum computing over the past decades. State-of-the-art devices now host hundreds of noisy *quantum bits* (qubits) and support a limited number of logical operations on these qubits. Such devices are referred to as *Noisy Intermediate-Scale Quantum* (NISQ) devices [1] and have been realized in a number of technology platforms, including superconducting circuits [2], trapped ions [3], neutral atoms [4], and single photons [5]. Notably, these devices almost exclusively work with two-level qubits, while the underlying hardware almost always natively supports encoding multi-valued logic in high-dimensional *quantum digits* (qudits) [6–8].

Research on qudit design and computation has a long history, with efforts having primarily focused on proof-of-principle experimental control, conceptual studies of primitives [9] and comparisons of algorithms for idealized qudits and qubits [10]. Fundamentally, a qudit can not only store and process more information per quantum particle, but also feature a richer Hilbert space structure [11, 12] and, thus, a larger set of logical operations that make processing more efficient [13]. Proof-of-principle experimental demonstrations [9, 14–16] have shown that these qudit advantages translate into improvements in circuit complexity and algorithmic efficiency for a wide range of problems. More recently, efforts have intensified with the demonstration of universal qudit quantum processors with competitive performance [6–8]. Such devices are already being used in quantum simulation applications, where a mixed-dimensional approach is much more naturally suited [16–18]. Examples include the simulation of high-spin systems [17, 19, 20], quantum chemistry [18], and lattice gauge theories [16, 21–23]. Combining the improved efficiency from native, rather than qubit-transpiled, implementations with the improved gate complexity enabled by the temporary expansion of the Hilbert space, has the potential to enable significantly shorter quantum circuits in mixed-dimensional hardware. Yet optimizing the use of mixed-dimensional resources is a highly non-trivial challenge.

In this work, we introduce MQT Qudits, an open-source framework for mixed-dimensional quantum computing, which is part of the *Munich Quantum Toolkit* (MQT, [24]). MQT Qudits provides efficient, automated, and accessible methods for tackling challenging problems in the design of quantum computing applications. In the following, we will provide a brief overview of MQT Qudits from both a user's perspective (showcasing how to utilize the tool for describing problems and algorithms in the form of quantum circuits, and how to simulate and compile them) and a technical perspective (covering the underlying methods, the choices made during development, and how to contribute to the framework). More precisely, we will:

- first, review the background and walk through the advantages, challenges, and opportunities of mixed-dimensional quantum computing (Section II),
- illustrate how to get started with MQT Qudits

* kevin.mato@tum.de
† martin.ringbauer@uibk.ac.at
‡ lukas.burgholzer@tum.de
§ robert.wille@tum.de

(Section III),

- introduce the languages and interfaces used by the tool (Section IV), and
- present the corresponding methods for quantum circuit simulation (Section V) and quantum circuit compilation (Section VI).

II. BACKGROUND

This section gives a brief overview of the principles of qudit quantum computing in current hardware before going into MQT Qudits. It should be mentioned that a thorough examination of the vast topic of quantum computation is not possible within this brief study. The interested reader is referred to in-depth literature, such as Ref. [10].

A. Quantum Information Processing

In the following, we will consider the qudit (quantum digit) as the fundamental unit of information. In contrast to the binary counterpart, the state of a qudit is represented by a vector in a d-dimensional Hilbert space, $\mathcal{H}_d$. This state can be expressed as a linear superposition:

$$|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle + \cdots + \alpha_{d-1}|d-1\rangle, \tag{1}$$

where the set $\{|i\rangle\}$ is referred to as the computational basis, and $|\alpha_i|^2$ is the probability of observing the state $|i\rangle$ upon measurement, with $\sum_{i=0}^{d-1} |\alpha_i|^2 = 1$. Qudits offer a richer state space, enabling more complex quantum information processing (QIP) than their binary counterpart, which are typically obtained by restricting naturally multi-level systems to just two levels. As a result, the core properties of quantum systems, superposition and entanglement, become much more interesting.

Example 1 *Consider a single qudit with three states (also referred to as* qutrit*). The quantum state* $|\psi\rangle = \sqrt{\frac{1}{3}}|0\rangle + \sqrt{\frac{1}{3}}|1\rangle + \sqrt{\frac{1}{3}}|2\rangle$ *corresponds to an equal superposition of the three basis states. This state can equivalently be represented as the vector* $\sqrt{\frac{1}{3}}\left[1,1,1\right]^T$.

Similarly, consider a composite system of one qutrit and one qubit in the (mixed-dimensional) entangled state $|\psi'\rangle = \sqrt{\frac{1}{3}}|0\rangle_3|0\rangle_2 + \sqrt{\frac{1}{3}}|1\rangle_3|1\rangle_2 + \sqrt{\frac{1}{3}}|2\rangle_3|0\rangle_2$ *— equivalently represented by the vector* $\sqrt{\frac{1}{3}}\left[1,0,0,1,1,0\right]^T$.

The manipulation of qudits is achieved through unitary operations, represented by $d \times d$ matrices $U \in \mathrm{SU}(d)$. In mixed-dimensional architectures, comprising qudits of varying dimensions, the unitary matrix governing a multi-qudit circuit scales as $\prod_i d_i \times \prod_i d_i$, where d_i denotes the dimension of each qudit.

Example 2 *A simple example of qudit operations are the generalizations of the Pauli X and Z [25]. In the case of a qutrit, they are described as*

$$X = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \omega & 0 \\ 0 & 0 & \omega^2 \end{bmatrix}, \tag{2}$$

with $\omega = e^{\frac{2\pi i}{d}}$ *and* d *being the dimension of the single qudit.*

Single-qudit operations can be tailored to act on specific subspaces or the entire Hilbert space of the qudit. Subspace operations could, for example, involve rotations built from Gell-Mann matrices [6], which physically describe two-level couplings in subspaces of the qudit. Operations on the full Hilbert space frequently utilize rotations built from the Pauli generalizations [25], called also clock and shift operators. Entangling operations can be engineered by composing Hamiltonians using these fundamental operators. The power of quantum algorithms lies in the efficient manipulation of superpositions of quantum states many-body interference effects and entanglement. Through careful sequencing of quantum operations, we can construct circuits that exploit this interference, and potentially offer computational advantages over classical algorithms for certain problems.

B. Advantages, Challenges and Opportunities

After discussing the fundamental aspects of quantum computing and its implementation using mixed-dimensional quantum systems, we now turn our attention to the advantages, challenges, and opportunities presented by mixed-dimensional quantum computing.

Current qubit-based quantum computers offer potential advantages in solving highly complex problems by reducing the amount of computational resources, yet scalability remains challenging. Qudit-based quantum computers could mitigate these limitations to some extent by increasing the information density and computational efficiency for a given number of quantum information carriers.

For instance, qudits enable native encoding of mixed-dimensional problems, such as those often found in quantum simulation tasks. Rather than requiring $\lceil \log(d) \rceil$ qubits to encode a d-dimensional object, thereby turning simple two-body into complicated many-body interactions, the mixed-dimensional approach enables us to allocate only the required resources and maintain simple circuit structures. In some cases, this may even allow for the flexible truncation of certain quantities to seamlessly enable the computation at different levels of accuracy [16]. However, these varying degrees of freedom

become part of the problem and have to be tracked. Being able to describe the problems easily and efficiently is the first step towards the exploitation of emerging mixed-dimensional quantum processors.

The introduction of qudits in quantum computing architectures creates a lot more freedom for design choices and problem-tailored solutions. This includes not only the choice and evolution of the single-qudit encoding but also the wide range of possible local and entangling operations. While for qubits the controlled-NOT (CNOT) gate is the central resource, qudit systems can build on an ever-increasing collection of non-equivalent entangling gates with varying degrees of entangling power [26] to enable efficient circuit design. The flip side of this greatly increased freedom and flexibility is the increased complexity of designing short and efficient, noise-resilient quantum circuits for qudit systems. Finding a way through this landscape of advantages, challenges, and opportunities presented by mixed-dimensional quantum systems necessitates a comprehensive set of methodologies and automated software tools for effective exploration and utilization, such as classical simulators and compilers.

To this end, the MQT Qudits framework offers an integrated advanced language for quantum circuit representations, simulation, and compilation methods, and software that facilitates the utilization of mixed-dimensional quantum systems. This framework enables researchers to systematically exploit the potential for quantum information processing tasks across diverse technology platforms and to access the full potential of mixed-dimensional qudit systems.

The remainder of the paper will introduce each component of the framework from two different perspectives: one focusing on the functionality for users, and the other on the technical aspects for motivated scientists and developers. We will always begin with the former.

III. GETTING STARTED

MQT Qudits empowers users to efficiently create, simulate, compile, and enhance mixed-dimensional quantum circuits across diverse quantum computing platforms. In the next sections, we provide a concise overview of how to set up and begin using MQT Qudits. The library is primarily written in Python, with components such as simulation engines written in C++ for performance reasons and then interfaced with Python. The installation process is designed to be user-friendly and does not require explicit compilation of the source. MQT Qudits supports Windows, Linux, and Mac for all versions of Python from 3.9 to 3.12. Installing MQT Qudits simply requires the execution of the following lines:

```
$ python3 -m venv .venv
$ source .venv/bin/activate
(.venv) $ pip install mqt.qudits
```

The code above demonstrates how to create a new environment and subsequently install the library directly from the terminal. After setting up the working environment, it is possible to write your first program.

IV. LANGUAGES AND INTERFACES

The first step towards simulating and compiling mixed-dimensional quantum circuits, which could potentially solve complex problems, is to define a language that can represent them. None of the existing languages provide appropriate means for this purpose.

A. The User's Perspective

MQT Qudits introduces DITQASM, an adaptation of OpenQASM [27] for describing mixed-dimensional quantum circuits and architectures. QASM languages serve as a bridge between high-level quantum algorithms and their physical implementation on hardware, and in this case, DITQASM introduces new fundamental features, as illustrated in Figure 1. Users can use DITQASM to translate abstract quantum algorithms into executable forms, analyze circuit complexity, and study the impact of noise and error correction in real quantum systems. Key features include:

- Register allocation: descriptions of the quantum and classical registers involved and their dimensionality.
- Gate-level description: the precise specification of quantum gates and their application to qudits of different dimensions, with a new way of controlling gates on arbitrary levels.
- Hardware-agnostic representation: a standardized way to express quantum circuits across different quantum computing platforms.
- Customizability and extendibility.

Example 3 *Figure 1 shows the DITQASM implementation of a quantum program using two qudit registers: one containing two qudits with dimensions 2 and 3, and another containing two qudits with dimensions 4 and 7 (lines 2-3). Additionally, a classical register is declared with 4 units (line 4). The circuit is then constructed, beginning with a Hadamard gate ("h") applied on the first qudit of register 2 and controlled by the state of the two qudits in the other register. This control is described using the "ctl" syntax (line 5), a new feature introduced in DITQASM. This is followed by a controlled-sum gate "csum" [28] (line 6) and two subspace rotations "rxy" [6]. The definitions of the gates do not require any knowledge of the technology platform implementing the multi-level quantum systems, and the dimensionality is not required at the application of the gates, but the focus is only on*

```
DITQASM 2.0;
qreg reg_1 [2][2, 3];
qreg reg_2 [2][4, 7];
creg meas[4];
h reg_2[0] ctl reg_1[0] reg_1[1] [0,0];
csum reg_2[0], reg_1[0];
rxy (0, 2, pi, pi/2) reg_1[1];
rxy (0, 1, pi, pi/2) reg_2[1];
measure reg_1[0] -> meas[0];
measure reg_1[1] -> meas[1];
measure reg_2[0] -> meas[2];
measure reg_2[1] -> meas[3];
```

FIG. 1. Example of DITQASM representing a program run on a quantum architecture made of two registers, with qudits of dimensions 2, 3, 4, and 7.

```
from mqt.qudits.quantum_circuit import QuantumCircuit
from mqt.qudits.quantum_circuit import QuantumRegister,\
                                       ClassicRegister

circuit = QuantumCircuit()
reg_1 = QuantumRegister("reg_1", 2, [2, 3])
reg_2 = QuantumRegister("reg_2", 2, [4, 7])
meas = ClassicRegister("meas", 4)
circuit.append(reg_qudit)
circuit.append(qubit_reg)
circuit.append_classic(meas)
circuit.h(reg_2[0]).control([reg_1[0],\
                                reg_1[1]], [0, 0])
circuit.csum([reg_2[0], reg_1[0]])
circuit.r(reg_1[1], [0, 2, np.pi, np.pi/2])
circuit.r(reg_2[1], [0, 1, np.pi, np.pi/2])
```

FIG. 2. A simple quantum program written for a mixed-dimensional quantum computer, in Python, through MQT Qudits.

the semantic information of the rotations. The measurement is performed according to the default options (lines 9-12).

Alternatively, circuits can be described by a Python interface, as shown in Figure 2. Here, the Python program starts with the import of the two main components of a quantum circuit: quantum registers, a quantum circuit, and, optionally, classical registers (lines 1-3). After creating a quantum circuit instance, the next two lines allocate two qudit registers, and a classic register (lines 5-11): "`reg_1`" and another one named "`reg_2`." The ability to name and allocate different registers facilitates the direct translation of interactions, such as those represented on a lattice or a graph, into an algorithm. Subsequently, the quantum operations are applied: a Hadamard gate on the "`reg_2`" register and a controlled-sum gate, which generalizes the CNOT gate (lines 12-14). The interpreter tracks the dimensionality of the qudits and applies the corresponding operations. The program concludes with a measurement of the qudits.

The Python interface and DITQASM also support various gates, including hardware-specific ones like the Molmer-Sorenson (MS) gate from [6] and the generalized light-shift (LS), a genuine qudit entangling gate [29, 30], as well as custom unitary evolutions on one, two, or multiple qudits and qubits.

B. The Technical Perspective

The section gives an intuition of the functioning of the DITQASM interpreter and pointers for further extensions and improvements. The distinguishing feature of DITQASM is its capacity to define arbitrary Hilbert space dimensions for individual elements within a quantum register, a capability previously unavailable. The expanded gate set, equipped with automatic dimensionality matching, makes the language hardware-agnostic. Furthermore, the automatic generalization of single-qudit and entangling gates to higher dimensions abstracts away the internal structure and properties of individual qudits, facilitating a focus on algorithm development. The language supports fine-grained control over multi-level systems, allowing for the specification of arbitrary control levels within the range $[0, d_i - 1]$ for the control qudit in controlled operations. Classical registers are implemented as integer wrappers, maintaining consistency with the multi-level quantum paradigm.

The current Python 3-based DITQASM interpreter is in its nascent stages. The current version (0.1.0) employs pattern matching during the parsing of the text of the quantum program; the instructions will be collected, and all the information and metadata will be tracked in the software. This data is then used to generate all the corresponding components and eventually compose them in the correct fashion. This approach ensures extensibility, allowing for easy incorporation of additional gates and features as the field evolves.

The Python interface and DITQASM are in direct relation to each other. This is sufficient for a local prototyping environment but will not suffice for large-scale deployments or for using MQT Qudits in a quantum computer-centric HPC environment. Possible solutions would involve the development of a suitable *Quantum Intermediate Representation* (QIR) for mixed-dimensional systems. Prospective developments include the integration of classical control flow syntax and mid-circuit measurements, crucial for implementing error correction schemes in qudit systems. These advancements are particularly pertinent given the growing relevance of qudit-based approaches across various domains of quantum computing research.

For a more extensive overview of all available options, techniques, and advanced features, please refer to MQT Qudits documentation at `mqt.readthedocs.io/projects/qudits/en/latest/`. This documentation provides in-depth information on working with mixed-dimensional quantum circuits, including detailed API references, tutorials and examples, as well as ad-

vanced configuration options.

V. QUANTUM CIRCUIT SIMULATION

Quantum circuit simulators are essential in assessing the quality of quantum circuits and quickly iterating new design choices for the algorithms to run. They enable the exploration of alternative qudit-based approaches, circuit optimizations, and the development or error mitigation schemes without having to rely on physical hardware. Different types of quantum circuits and applications require specialized simulators for each task. To this end, MQT Qudits offers two simulation options, each one with its merits and limitations. Again, we will examine both from the user and technical perspectives.

A. The User's Perspective

MQT Qudits offers a user-friendly interface that hides the complexity of simulating quantum circuits. In Figure 3, a simple demonstration of how to use the framework for quantum circuit simulations is shown. There are two levels of complexity for performing a simulation depending on the required level of customization. Once a quantum circuit is defined, you can call the simulate method of the circuit instance, which will return a job object upon completion of the execution. It is possible to retrieve the quantum state from the job, by calling "`get_state_vector()`". Alternatively, it is possible to choose a backend, thanks to the method "`get_backend(backend_name)`", between a tensor network simulator ("tnsim"), a mixed-dimensional decision diagram simulator ("misim", [31]), and a suite of simple emulators representing physical devices, again by using "`get_backend(fake_backend_name)`". These latter, prefixed with "fake", facilitate compilation and noisy simulation of quantum circuits with device-specific properties. Upon backend selection, the environment is ready to run the simulation of the circuit.

A particularly relevant feature of a quantum circuit simulator is noise-aware simulation to allow for estimating the quality and feasibility of execution on a given quantum hardware. Figure 4 shows how to construct a noise model by specifying probabilities for generalized Pauli errors, analogous to X and Z flips in the qubit case, extended to the qudit Hilbert space and subspaces thereof. The framework allows for fine-grained control over noise channels, supporting noise on local, as well as, multi-qudit operations with different effects on target and control qudits, and various other noise channels as detailed in the documentation. This approach enables the simulation of realistic noise processes in higher-dimensional quantum systems, crucial for assessing the performance of qudit-based quantum algorithms and error correction schemes. Different types of noise instances can be combined in a noise model and input as parameters of a simulation. The knowledge about the noise of a system can be easily integrated by adding it as an option to the simulation or by choosing a "fake" backend that will possess this knowledge by default.

```python
# import components and circuit
from mqt.qudits.simulation import MQTQuditProvider

# program written before
state = circuit.simulate()

# alternatively
provider = MQTQuditProvider()
backend = provider.get_backend("tnsim")
job = backend.run(circuit)
result = job.result()
state = result.get_state_vector()
```

FIG. 3. Simulation of a qudit circuit with MQT Qudits' TnSim.

```python
# import components and circuit
from mqt.qudits.simulation import MQTQuditProvider
from mqt.qudits.simulation.noise_tools.noise import \
    Noise, NoiseModel

local_error = Noise(0.01, 0.001)
local_error_rz = Noise(0.018, 0.002)

noise_model = NoiseModel()
noise_model.add_quantum_error_locally(local_error,\
                        ["h", "rxy", "s", "x", "z"])
noise_model.add_quantum_error_locally(local_error_rz,\
                        ["rz"])

provider = MQTQuditProvider()
backend = provider.get_backend("tnsim")
job = backend.run(circuit, noise_model=noise_model)
result = job.result()
counts = result.get_counts()
```

FIG. 4. Simulation of a qudit circuit with a tailored noise model through MQT Qudits.

B. The Technical Perspective

The simplicity of use of the simulation components of MQT Qudits is due to the implementation of a modular architecture. The entry point is a provider object that serves as an abstraction layer for various backend implementations. For brevity, this section will cover only the most novel backends of the library: MiSiM [31] and TnSim. MiSiM, a C++ implementation, extends edge-weighted *Decision Diagrams* (DDs, [32, 33]), or QMDDs [34], to efficiently simulate mixed-dimensional qudit systems. By dynamically capturing qudit dimensionality, MiSiM significantly reduces memory requirements, enabling the simulation of complex circuits, e.g., exceeding, in some cases, 100 qutrits. This DD-based approach exploits redundancies and symmetries in the rep-

resentation of quantum states and operations, allowing direct manipulation of operations on the diagram structure.

Additionally, DDs can efficiently represent sparse data, becoming a powerful tool for simulating circuits with non-trivial quantum correlations by representing states that would otherwise require exponential resources. TnSim, conversely, utilizes *tensor networks* representations via Google's TensorNetwork API [35] in Python. The tensor network simulator contained in MQT Qudit takes care of the construction of the network and the dynamical allocation of nodes in it and leverages optimized tensor contraction algorithms, offering performance advantages for circuits with specific entanglement structures. Tensor networks efficiently represent quantum many-body states with limited entanglement, particularly those obeying area-law scaling [36]. They decompose high-dimensional states into contracted lower-dimensional tensors, reducing computational complexity from exponential to polynomial for many physical systems.

Both backends support noise-aware simulation of quantum circuits through a stochastic approach [37–39]. This method appends qudit Pauli X or Z gates after each operation with a specific probability described in the noise instance (Figure 4). For gates operating in subspaces, such as local subspace rotations [6], the noise is applied as subspace X or Z operations. The noise model in Fig. 4 encodes the logic determining whether each gate has a noise operation appended, and for entangling operations, whether noise is applied to the target, control(s), or a subset of qudits involved. A particularly simple, yet highly relevant, extension is the integration of numerical distortions of the gate parameters in the circuit associated with a specific noise channel, or the inclusion of subspace error operations for all types of gates.

This stochastic noise simulation method efficiently models decoherence and gate errors in quantum systems without the need for full density-matrix calculations. However, it requires multiple simulation runs to create a statistical distribution of results, which fluctuates due to operation-specific noise. Parallel processing could conceivably mitigate this limitation by executing runs simultaneously on several processes.

In summary, the library provides a suite of complementary simulators that leverage two data structures and different implementations. These can provide solutions with varying performance depending on the use case, the complexity of the noise to simulate, and the user's preference. Users can choose the most suitable simulator based on their specific requirements and computational constraints.

VI. QUANTUM CIRCUIT COMPILATION

Qudit compilation transforms abstract quantum algorithms into executable sequences for specific hardware, bridging theory, and implementation. The MQT Qudits compiler integrates quantum control theory, optimization, and custom heuristics to address coherence and fidelity challenges. It encompasses gate decomposition, mapping, and noise adaptation, crucial for realizing quantum algorithmic potential in near-term qudit processors. The compiler aims to be accessible to a wide range of users, from novices exploring mixed-dimensional quantum computing to well-resourced laboratories, while offering flexible, modular, and computationally efficient software that balances user-friendly operation with advanced technical capabilities. This section illustrates the use of the compiler.

A. The User Perspective

Figure 5 outlines how the user can exploit MQT Qudits for compilation tasks. The code shows how to set the initial state of the quantum circuit as a first operation, starting from the numpy array [40] of the state. After setting the initial state (line 8), the user can define the rest of the quantum circuit and directly compile the circuit for a target device (lines 9-11), just by stating the name of the device, prefixed by "fake". The compilation, in this case, chooses some default flags and returns the compiled circuit compatible with the device to be executed on. Although highly flexible, the compiler is constantly evolving. Multi-body gate compilation will be supported in future releases. For the user who is interested in benchmarking and tuning the compilation process (lines 14-20), the compilation of a quantum circuit requires minimal code and only three steps, namely

1. instantiation of the qudit compiler,
2. selection of compilation steps, and
3. choice of a target backend.

In this case, the compiler is returning the compiled circuit, which can then be either simulated or passed to the API of the actual device, after using the correct backend, e.g., "innsbruck01".

B. The Technical Perspective

Shielding users from the complexity of the compilation process necessitates the development of sophisticated methods and implementations. The compiler is designed as a modular, pass-based tool that prioritizes algorithmic efficiency, despite being entirely written in Python. In the following, we briefly review the main methods currently implemented in the MQT Qudits compiler, namely a state preparation routine, a single-qudit operations compiler, and a two-qudit operations compiler. The state preparation routine takes a quantum state in the form of a numpy array and outputs a sequence of local and controlled two-level rotation, with

```python
# import components and circuit
from mqt.qudits.simulation import MQTQuditProvider
from mqt.qudits.compiler import QuditCompiler
provider = MQTQuditProvider()

# optionally the user can set the initial state
state = np.array(...)
circuit.set_initial_state(state)
# rest of the program
# simple interface
new_circuit = circuit.compile("faketraps2six")

# alternatively
qudit_compiler = QuditCompiler()
# choice of the passes
passes = ["PhyLocQRPass", "PhyEntQRPass"]

device = provider.get_backend("faketraps2six")
new_circuit = qudit_compiler.compile(device, \
                        circuit, passes)
```

FIG. 5. A prototypical compilation of a quantum circuit on MQT Qudits.

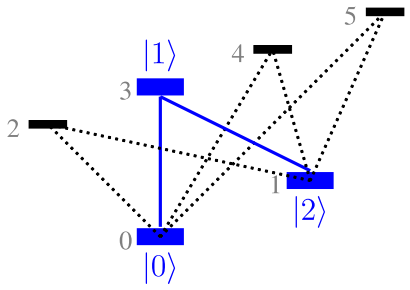

FIG. 6. Energy level graph of a qutrit, 3-level qudit.

an increasing number of controls. The state preparation makes use of mixed-dimensional decision diagrams [31] implemented in Python for efficiently representing the state. A synthesis algorithm reads the DD structure, possibly approximates it if required by the user, and returns a sequence of quantum operations to generate the state. The execution of the multi-controlled operations is allowed by an implicit decomposition in qudit "CNOT" by using ancilla levels in the qudits [16]. We refer to the paper [41] for further details on the synthesis.

The compilation steps enabled in the process depend on the user's choice of passes. Depending on the selected passes, the compilation can target either single-qudit or two-qudit unitaries. The pass nomenclature indicates whether the compilation produces a physically executable sequence compatible with the backend ("Phys") or a logical sequence ("Log"). "Log" denotes single-qudit unitary compilation, while "Ent" signifies two-qudit unitary compilation.

Single-qudit unitaries are compiled using sequences of 2-level subspace rotations. This reconstruction method aligns with the energy level graph of the individual qudit, as shown schematically in Figure 6. The energy level graph [42] is a data structure that represents the internal structure of the qudit, storing information about the quality of rotations between two levels and tracking various properties of the decomposition on the physical system. This graph structure corresponds to the coupling map between the various qudit levels. The use of the energy level graph and graph optimization heuristics has led to improvements in the compilation of single-qudit unitaries, particularly in reducing the number of rotations required, especially phase rotations [42].

MQT Qudits compiles two-qudit unitaries of arbitrary dimensionality into sequences of controlled rotations (crot) and partial swap operations (pswap). These operations are then either directly implemented in hardware or further decomposed into a sequence of local operations and controlled-exchange gates, which is a qudit-embedded 2-level CNOT gate. This methodology [28] provides a computationally efficient means of compiling arbitrary interactions into standardized entangling gates, independent of the underlying hardware architecture. The compiler allows for variational compilation of the standardized entangling gates into the hardware-native gate set, tailoring the circuit to the specific hardware. Further extensions should include tools for compressing the generated circuits to further increase the efficiency.

An additional standalone tool is the qubit compression mapper [43], a utility that recommends dimensionality adjustments in mixed-dimensional architectures to efficiently encode qubit circuits into qudit circuits. This tool facilitates the translation between different quantum information encoding schemes, potentially enabling more efficient use of available quantum resources.

VII. CONCLUSION

MQT Qudits provides an open-source framework for mixed-dimensional quantum computing, including flexible automated tools for the design, specification, simulation, and compilation of mixed-dimensional quantum circuits. Considering the fundamental differences between qubit and qudit QIP, the framework is designed to be easily extendable to new kinds of gate operations that are expected to be developed as the field evolves. Similarly, the development and inclusion of more hardware-specific noise models for the various types of qudit quantum computing hardware is a point of great interest to enable accurate circuit simulation.

A particularly interesting topic is the dynamic evolution of the data structures during the simulation with the goal of tracking the usage of ancilla levels in the qudit. Finally, the compiler framework enables flexible and adaptive compilation, yet does not always achieve optimal circuits. Future efforts should prioritize automated multi-qudit gate compilation, and optimizing two-qudit gate compilations to minimize the resulting cir-

cuit complexity. Efforts should also include resynthesis procedures to reduce the final circuit depth. In order to facilitate these developments, MQT Qudits will, in the future, include scalable verification methods for mixed-dimensional systems.

ACKNOWLEDGMENTS

This research was funded by the European Union under the Horizon Europe Programme as part of the project NeQST (Grant Agreement 101080086) as well as by the European Research Council projects DA QC (Grant Agreement 101001318) and QUDITS (Grant Agreement 101080086). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. The work is part of the Munich Quantum Valley, which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus and was partially supported by the BMK, BMDW, and the State of Upper Austria in the frame of the COMET program (managed by the FFG).

[1] J. Preskill, Quantum computing in the NISQ era and beyond, Quantum **2**, 79 (2018).

[2] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, A. Fowler, C. Gidney, M. Giustina, R. Graff, K. Guerin, S. Habegger, M. P. Harrigan, M. J. Hartmann, A. Ho, M. Hoffmann, T. Huang, T. S. Humble, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi, J. Kelly, P. V. Klimov, S. Knysh, A. Korotkov, F. Kostritsa, D. Landhuis, M. Lindmark, E. Lucero, D. Lyakh, S. Mandrà, J. R. McClean, M. McEwen, A. Megrant, X. Mi, K. Michielsen, M. Mohseni, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Y. Niu, E. Ostby, A. Petukhov, J. C. Platt, C. Quintana, E. G. Rieffel, P. Roushan, N. C. Rubin, D. Sank, K. J. Satzinger, V. Smelyanskiy, K. J. Sung, M. D. Trevithick, A. Vainsencher, B. Villalonga, T. White, Z. J. Yao, P. Yeh, A. Zalcman, H. Neven, and J. M. Martinis, Quantum supremacy using a programmable superconducting processor, Nature **574**, 505 (2019).

[3] I. Pogorelov, T. Feldker, C. D. Marciniak, L. Postler, G. Jacob, O. Krieglsteiner, V. Podlesnic, M. Meth, V. Negnevitsky, M. Stadler, B. Höfer, C. Wächter, K. Lakhmanskiy, R. Blatt, P. Schindler, and T. Monz, Compact Ion-Trap Quantum Computing Demonstrator, PRX Quantum **2**, 020343 (2021).

[4] D. Bluvstein, S. J. Evered, A. A. Geim, S. H. Li, H. Zhou, T. Manovitz, S. Ebadi, M. Cain, M. Kalinowski, D. Hangleiter, J. P. Bonilla Ataides, N. Maskara, I. Cong, X. Gao, P. Sales Rodriguez, T. Karolyshyn, G. Semeghini, M. J. Gullans, M. Greiner, V. Vuletić, and M. D. Lukin, Logical quantum processor based on reconfigurable atom arrays, Nature **626**, 58–65 (2023).

[5] F. Flamini, N. Spagnolo, and F. Sciarrino, Photonic quantum information processing: A review, Reports Prog. Phys. **82**, 016001 (2019).

[6] M. Ringbauer, M. Meth, L. Postler, R. Stricker, R. Blatt, P. Schindler, and T. Monz, A universal qudit quantum processor with trapped ions, (2021), arXiv:2109.06903.

[7] A. Morvan, V. V. Ramasesh, M. S. Blok, J. M. Kreikebaum, K. O'Brien, L. Chen, B. K. Mitchell, R. K. Naik, D. I. Santiago, and I. Siddiqi, Qutrit Randomized Benchmarking, Phys. Rev. Lett. **126**, 210504 (2021).

[8] Y. Chi, J. Huang, Z. Zhang, J. Mao, Z. Zhou, X. Chen, C. Zhai, J. Bao, T. Dai, H. Yuan, M. Zhang, D. Dai, B. Tang, Y. Yang, Z. Li, Y. Ding, L. K. Oxenløwe, M. G. Thompson, J. L. O'Brien, Y. Li, Q. Gong, and J. Wang, A programmable qudit-based quantum processor, Nat. Commun. **13**, 1166 (2022).

[9] B. P. Lanyon, M. Barbieri, M. P. Almeida, T. Jennewein, T. C. Ralph, K. J. Resch, G. J. Pryde, J. L. O'Brien, A. Gilchrist, and A. G. White, Simplifying quantum logic using higher-dimensional Hilbert spaces, Nat. Phys. **5**, 134 (2008).

[10] Y. Wang, Z. Hu, B. C. Sanders, and S. Kais, Qudits and high-dimensional quantum computing, Frontiers in Physics **8**, 10.3389/fphy.2020.589504 (2020).

[11] M. Ringbauer, T. R. Bromley, M. Cianciaruso, L. Lami, W. Y. S. Lau, G. Adesso, A. G. White, A. Fedrizzi, and M. Piani, Certification and Quantification of Multilevel Quantum Coherence, Phys. Rev. X **8**, 041007 (2018).

[12] T. Kraft, C. Ritz, N. Brunner, M. Huber, and O. Gühne, Characterizing Genuine Multilevel Entanglement, Phys. Rev. Lett. **120**, 060502 (2018).

[13] X. Gao, P. Appel, N. Friis, M. Ringbauer, and M. Huber, On the role of entanglement in qudit-based circuit compression, Quantum **7**, 1141 (2023).

[14] Z. Gedik, I. A. Silva, B. Çakmak, G. Karpat, E. L. G. Vidoto, D. O. Soares-Pinto, E. R. DeAzevedo, and F. F. Fanchini, Computational speed-up with a single qudit, Sci. Rep. **5**, 14671 (2015).

[15] X. Zhan, J. Li, H. Qin, Z. hao Bian, and P. Xue, Linear optical demonstration of quantum speed-up with a single qudit, Optics Express **23**, 18422 (2015).

[16] M. Meth, J. F. Haase, J. Zhang, C. Edmunds, L. Postler, A. Steiner, A. J. Jena, L. Dellantonio, R. Blatt, P. Zoller, T. Monz, P. Schindler, C. Muschik, and M. Ringbauer, Simulating 2D lattice gauge theories on a qudit quantum computer (2024), arXiv:2310.12110 [quant-ph].

[17] C. L. Edmunds, E. Rico, I. Arrazola, G. K. Brennen, M. Meth, R. Blatt, and M. Ringbauer, Constructing the spin-1 Haldane phase on a qudit quantum processor (2024).

[18] T. Navickas, R. J. MacDonell, C. H. Valahu, V. C. Olaya-Agudelo, F. Scuccimarra, M. J. Millican, V. G. Matsos, H. L. Nourse, A. D. Rao, M. J. Biercuk, C. Hempel, I. Kassal, and T. R. Tan, Experimental Quantum Simulation of Chemical Dynamics (2024).

[19] F. D. M. Haldane, Nonlinear Field Theory of Large-Spin Heisenberg Antiferromagnets: Semiclassically Quan-

tized Solitons of the One-Dimensional Easy-Axis Néel State, Physical Review Letters **50**, 10.1103/PhysRevLett.50.1153 (1983).

[20] C. Senko, P. Richerme, J. Smith, A. Lee, I. Cohen, A. Retzker, and C. Monroe, Realization of a Quantum Integer-Spin Chain with Controllable Interactions, Phys. Rev. X **5**, 021026 (2015).

[21] D. González-Cuadra, T. V. Zache, J. Carrasco, B. Kraus, and P. Zoller, Hardware Efficient Quantum Simulation of Non-Abelian Gauge Theories with Qudits on Rydberg Platforms, Phys. Rev. Lett. **129**, 160501 (2022).

[22] G. Calajò, G. Magnifico, C. Edmunds, M. Ringbauer, S. Montangero, and P. Silvi, Digital Quantum Simulation of a (1+1)D SU(2) Lattice Gauge Theory with Ion Qudits (2024).

[23] P. P. Popov, V. Kasper, M. Lewenstein, E. Zohar, P. Stornati, and P. Hauke, Non-perturbative signatures of fractons in the twisted multi-flavor Schwinger Model, Preprint at arXiv:2405.00745 (2024).

[24] R. Wille, L. Berent, T. Forster, J. Kunasaikaran, K. Mato, T. Peham, N. Quetschlich, D. Rovara, A. Sander, L. Schmid, D. Schönberger, Y. Stade, and L. Burgholzer, The MQT Handbook: A Summary of Design Automation Tools and Software for Quantum Computing (2024), arXiv:2405.17543 [quant-ph].

[25] S. Clark, Valence bond solid formalism ford-level one-way quantum computation, Journal of Physics A: Mathematical and General **39**, 2701–2721 (2006).

[26] M. Huber and J. I. de Vicente, Structure of multidimensional entanglement in multipartite systems, Phys. Rev. Lett. **110**, 030501 (2013).

[27] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, Open quantum assembly language (2017), arXiv:1707.03429 [quant-ph].

[28] K. Mato, M. Ringbauer, S. Hillmich, and R. Wille, Compilation of entangling gates for high-dimensional quantum systems, in *Asia and South Pacific Design Automation Conf.*, ASPDAC '23 (2023) p. 202–208.

[29] B. C. Sawyer and K. R. Brown, Wavelength-insensitive, multispecies entangling gate for group-2 atomic ions, **103**, 022427 (2021).

[30] P. Hrmo, B. Wilhelm, L. Gerster, M. W. van Mourik, M. Huber, R. Blatt, P. Schindler, T. Monz, and M. Ringbauer, Native qudit entanglement in a trapped ion quantum processor 10.48550/ARXIV.2206.04104 (2022), arXiv:2206.04104.

[31] K. Mato, S. Hillmich, and R. Wille, Mixed-dimensional quantum circuit simulation with decision diagrams, in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Vol. 01 (2023) pp. 978–989.

[32] A. Zulehner and R. Wille, Advanced simulation of quantum computations, **38**, 848 (2019).

[33] A. Zulehner, S. Hillmich, and R. Wille, How to efficiently handle complex values? Implementing decision diagrams for quantum computing (ACM, 2019) pp. 1–7.

[34] P. Niemann, R. Wille, D. M. Miller, M. A. Thornton, and R. Drechsler, QMDDs: Efficient quantum function representation and manipulation, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **35**, 86 (2016).

[35] C. Roberts, A. Milsted, M. Ganahl, A. Zalcman, B. Fontaine, Y. Zou, J. Hidary, G. Vidal, and S. Leichenauer, Tensornetwork: A library for physics and machine learning (2019), arXiv:1905.01330 [physics.comp-ph].

[36] R. Orús, A practical introduction to tensor networks: Matrix product states and projected entangled pair states, Annals of Physics **349**, 117–158 (2014).

[37] W. Berquist, D. Lykov, M. Liu, and Y. Alexeev, Stochastic approach for simulating quantum noise using tensor networks, in *Third International Workshop on Quantum Computing Software (QCS)* (2022) pp. 107–113.

[38] T. Grurl, R. Kueng, J. Fuß, and R. Wille, Stochastic quantum circuit simulation using decision diagrams, in *Design, Automation & Test in Europe Conference*, pp. 194–199.

[39] T. Grurl, J. Fuß, and R. Wille, Noise-aware quantum circuit simulation with decision diagrams, Transactions on Computer-Aided Design of Integrated Circuits and Systems **42**, 860 ().

[40] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, Array programming with NumPy, Nature **585**, 357 (2020).

[41] K. Mato, S. Hillmich, and R. Wille, Mixed-dimensional qudit state preparation using edge-weighted decision diagrams, *Design Automation Conf.*, (2024).

[42] K. Mato, M. Ringbauer, S. Hillmich, and R. Wille, Adaptive compilation of multi-level quantum operations, in *International Conference on Quantum Computing and Engineering (QCE)* (2022) pp. 484–491.

[43] K. Mato, S. Hillmich, and R. Wille, Compression of qubit circuits: Mapping to mixed-dimensional quantum systems, in *International Conference on Quantum Software (QSW)* (2023) pp. 155–161.