QProv: A provenance system for quantum computing

Benjamin Weder  |  Johanna Barzen  |  Frank Leymann  |  Marie Salm  |  Karoline Wild

University of Stuttgart, Institute of Architecture of Application Systems, Universitätsstraße 38, Stuttgart, Germany.

Correspondence
Benjamin Weder, University of Stuttgart, Institute of Architecture of Application Systems, Universitätsstraße 38, Stuttgart, 70569, Germany. Email: benjamin.weder@iaas.uni-stuttgart.de

Funding information
German Research Foundation, Grant/Award Number: EXC 2075-390740016; BMWi project PlanQK, Grant/Award Number: 01MK20005N

Abstract
Quantum computing promises breakthroughs in various application areas, such as machine learning, chemistry, or simulations. However, today’s quantum computers are error-prone and have limited capabilities. This leads to various challenges when developing and executing quantum algorithms, for example, the mitigation of occurring errors or the selection of a suitable quantum computer to execute a certain quantum circuit. To address these challenges, detailed information about the quantum circuit to be executed as well as past executions, and the up-to-date information about the available quantum computers are required. Thus, this data must be continuously collected and stored in the long-term, which is currently not supported. To overcome this problem, a provenance approach is introduced for quantum computing. Therefore, relevant provenance attributes that should be gathered in the area of quantum computing are identified. Furthermore, QProv, a provenance system that automatically collects the identified provenance attributes and provides them in a uniform manner to the user is introduced. Finally, a case study with the collected provenance data and corresponding use cases that can benefit from this provenance data are presented here.

1 | INTRODUCTION

Quantum computing has the potential to enable breakthroughs in different application areas, such as machine learning, chemistry, or scientific simulations [1–3]. By exploiting quantum mechanical principles, such as entanglement and superposition, quantum algorithms can solve many problems more efficiently than it is possible on classical computers [4–6]. In recent years, various quantum hardware providers, such as IBM, Rigetti, or IonQ, developed quantum computers and provided access to them via the cloud [1, 7]. Therefore, quantum computers have become accessible to the public, and use cases from various application areas can now be implemented, tested, and executed on real quantum computers [3, 8, 9].

However, today’s quantum computers are affected by noise from various sources, which can cause errors in computations [1, 5, 8]. Additionally, the number of qubits provided by the available quantum computers is limited and periodic recalibrations change their characteristics over time [10]. For example, the decoherence times of the qubits or the gate fidelities may differ significantly after two calibrations. These limitations lead to several challenges when developing and executing quantum circuits on today’s quantum computers [4, 8]. For example, the selection of a suitable quantum computer to execute a certain quantum circuit is difficult due to their diverse capabilities and continuously changing characteristics [11]. Furthermore, different sources of noise may be the reason for an occurring error. Hence, the analysis of errors to improve the results of the executed quantum circuit is complex. One approach to reduce the impact of noise on the results is to use the so-called error-mitigation techniques [8, 12, 13]. However, to apply such techniques, data about the quantum circuit and the target quantum computer is needed. Therefore, to tackle these challenges, detailed information about the available quantum computers with their current characteristics, the quantum circuits to be executed, as well as their past executions are required.

However, the manual collection of all relevant information is time consuming and requires a lot of knowledge, for example, about quantum computers, quantum circuits, and software tools, such as quantum compilers or SDKs. The
systematic and automated collection of such data, as well as its long-term storage for analysis, is referred to as provenance [14, 15]. Thereby, provenance systems are used in different application areas, for example, to analyse the execution of workflows, make a scientific simulation reproducible, or debug some software component [15–17]. However, there exists currently no provenance approach for quantum computing, identifying relevant provenance attributes, as well as for automatically collecting and storing them in the long-term by a provenance system to enable their analysis [4, 8].

To overcome these challenges, we introduce a provenance approach for quantum computing in this paper. Thus, we (i) identified relevant provenance attributes that should be collected in the area of quantum computing. By gathering these attributes, different use cases can be supported, such as the optimisation of a quantum circuit during compilation based on the current hardware characteristics [18] or the mitigation of readout errors in the results of an execution [12, 13]. Further, we (ii) propose a provenance system to automatically collect and store the identified quantum provenance attributes and offer them in a provider-independent manner. To prove the practical feasibility, a prototypical implementation of a corresponding provenance system is presented. Finally, we (iii) introduce a case study with collected provenance data and corresponding use cases that can benefit from this data. Thereby, we discuss the resulting implications for the development of quantum circuits and tooling support, such as modelling tools or quantum compilers.

The remainder of this paper is structured as follows: Section 2 describes fundamentals about provenance, discusses related work, and presents our problem statement. In Section 3, an overview of the identified quantum provenance attributes is given. Afterward, Section 4 introduces the quantum provenance system, and Section 5 presents our case study. Finally, in Section 6, we discuss the limitations of our approach, and we conclude in Section 7.

2 | FUNDAMENTALS, RELATED WORK, AND PROBLEM STATEMENT

In this section, we introduce fundamentals about provenance and discuss related work, describing use cases where provenance data are collected or used in the area of quantum computing. Finally, we present the problem statement that underlies our work.

2.1 | Provenance

Provenance refers to all data and meta-data describing the history of an object, such as a piece of digital data or a physical object [14, 15]. There are different kinds of provenance, such as workflow provenance [16] or data provenance [20]. For example, workflow provenance approaches try to capture all relevant information about the execution and the results of a workflow. The collection of provenance data has the goal to increase the reproducibility, understandability, and quality of a process or object [15]. Therefore, it is important to systematically record all relevant information as detailed as possible. To implement a provenance approach, it has to be defined which provenance data to gather, how to collect it, and where to store the provenance data in the long-term for later analysis [17]. Furthermore, suited analysis methods for the collected data have to be developed and evaluated to gain valuable insights.

In general, the provenance of an object is represented as a graph consisting of nodes defining, for example, actions that were applied to a data object or earlier version of the data object, and edges describing their relations [15, 21]. However, provenance solutions are implemented in different ways, for example, included in scientific workflow management systems [22, 23], or as standalone provenance systems such as Progger [24]. To enable interoperability between these systems and different analysis services, the W3C introduced the PROV standard [19]. It defines an extensible provenance meta-model with the basic elements required to describe the provenance of an object or process, as well as an XML-based serialisation. Figure 1 shows the three basic elements that are used as nodes in a provenance graph: entities, activities, and agents. Thereby, entities represent physical or virtual objects, such as documents or web pages. Additionally, activities are actions or processes, which can generate new entities or may operate on some existing entities to create a new version. For example, an activity could change a web page. Thus, the corresponding provenance graph consists of an entity describing the old version of the web page, the change activity, and the entity representing the new version. Besides, agents can be associated with an activity to describe that they performed some tasks in the activity. Thereby, an agent may represent a human, an organisation, or some software component. Furthermore, the nodes in a provenance graph are connected by typed edges. For example, edges of type wasGeneratedBy describe which activity generated an entity. Finally, a set of attributes can be defined for a node in a provenance graph to further characterise it, such as the name of an entity representing a human.

2.2 | Related work

To the best of our knowledge, there exists currently no holistic provenance approach for quantum computing that enables the systematic collection and long-term storage of relevant provenance data. Different research works identify and gather
important data, but only specific to a certain use case, or the data is collected once for an analysis and not continuously. In the following, we discuss these research works, as well as potential applications for the collection and analysis of provenance data in the area of quantum computing.

For the mitigation of errors, different techniques have been compared and evaluated by Endo et al. [25]. These techniques rely on the current characteristics of the used quantum computer, and they showed how to estimate these characteristics using quantum gate set tomography. Maciejewski et al. [12] also proposed a readout-error mitigation technique, which is based on classical post-processing. For this, they pointed out how to measure the readout errors of a quantum computer and construct the calibration matrix from the results. This matrix can then be inverted and applied to the results of quantum circuit executions to mitigate readout errors.

Häner et al. [26] introduced a methodology to compile quantum circuits to the machine instructions for the execution on a certain quantum computer. Thereby, they use hardware characteristics, such as the qubit connectivity and the hardware-specific gate set, to determine the qubit allocations and gate mappings. Sivarajah et al. [18] presented a quantum compiler that considers further characteristics of the target quantum computer, such as the gate fidelities or the decoherence times of the qubits, to optimise the resulting machine instructions. The same characteristics have also been considered by Tannu and Qureshi [10], who analysed the problem of allocating qubits during the compilation. Further, they observed that these characteristics change significantly over time and tracked their evolution over 52 days on a quantum computer from IBM. Thus, it is important to provide current data about quantum computer characteristics.

With the limited capabilities of today's quantum computers, the selection of a suitable quantum computer for the execution of a given quantum circuit is difficult. Hence, we presented an approach [11] for the selection based on attributes of the quantum circuit to execute, such as the circuit depth and width. Furthermore, hardware characteristics of the available quantum computers, such as the T1 times or the execution times of the gates, are used to estimate if the quantum circuit can be successfully executed on a quantum computer.

Suchara et al. [27] present the Resource Estimator Toolbox to estimate the number of qubits and gates required to execute a quantum algorithm on the given input data, the probability of success for the computation, and the execution time. This data should be collected, for example, to enable comparing the execution on different quantum computers.

To develop an integrated knowledge base for quantum computing, Martyniuk et al. [28] propose a first set of entities for an ontology to curate knowledge about quantum algorithms and their implementations. Thereby, they also identify relevant provenance attributes, such as input and output data of a quantum algorithm.

In previous works [4, 8], we already emphasised the importance of a holistic provenance approach for quantum computing. Beyond the use cases already mentioned, other discussed application areas are the splitting of problems into quantum and classical parts to execute them as a hybrid application or to increase the reusability, quality, and understandability of quantum circuits.

2.3 | Problem statement

As outlined in the previous section, many use cases can benefit from the systematic collection of provenance data in the area of quantum computing. Hence, a holistic provenance approach for quantum computing is needed, which collects, stores, and analyses all relevant provenance data. This is especially important during the NISQ era with its noisy and error-prone quantum computers and their limited capabilities [4, 5, 8]. The first step towards a provenance approach for quantum computing is to identify relevant provenance attributes about quantum computers, quantum circuits, and their execution. Therefore, our first research question (RQ) is as follows:

**RQ1:** "What provenance attributes about quantum computers, quantum circuits, and their execution are relevant to support the development and execution process of quantum circuits?"

To gain valuable insights from the collected data, it has to be gathered systematically and over long periods. Furthermore, for different use cases, such as the selection of a suitable quantum computer, the provenance data must be collected for quantum computers from various quantum hardware providers. However, the manual collection of the data is time consuming and requires mathematical knowledge about quantum computing, as well as technical knowledge about the available quantum computers or software tools, such as quantum compilers or SDKs. Hence, the collection of the provenance data should be automated and performed provider-independent by a provenance system. Thus, our second research question is as follows:

**RQ2:** "How can the different provenance attributes be retrieved in an automated and provider-independent manner?"

3 | QUANTUM PROVENANCE ATTRIBUTES

In this section, we give an overview of the relevant provenance attributes that should be collected in the course of a provenance approach for quantum computing (RQ1). For this, we
analysed the research works presented in Section 2.2 and extracted the relevant provenance attributes to support the presented use cases, for example, the compilation of quantum circuits or the quantum hardware selection. Furthermore, we evaluated and restricted these attributes to a set that can be directly retrieved over the APIs of different quantum hardware providers and tools or gathered, for example, by executing calibration circuits. Thereby, we focus on the gate-based quantum computing model, and new attributes for other quantum computing models can be added in future work. The identified provenance attributes are divided into four categories, as depicted in Figure 2. In the first category, provenance attributes about a quantum circuit are considered, which can be used to analyse executions of the quantum circuit or to compare different quantum circuits. The second category comprises provenance data about quantum computers and their current characteristics. This provenance data is required to be estimated if a quantum circuit can be successfully executed on a quantum computer. Before executing a quantum circuit, it has to be compiled to the machine instructions, and all provenance data related to this compilation is aggregated in the third category. Finally, the last category contains all information about the execution of a quantum circuit. In the following subsections, the four quantum provenance categories and the corresponding provenance attributes are discussed in detail.

### 3.1 Quantum circuit category

In the first quantum provenance category, provenance data about the quantum circuit to be executed is considered. A quantum circuit consists of a set of gates and measurements, which operate on different qubits [1, 2]. Therefore, the used gates (see Q1 in Figure 2), the used measurements (Q2), and their execution order (Q3) should be gathered [8]. This data increases the reproducibility and understandability of the quantum computation. Additionally, it enables the analysis of the results of an execution based on the structure of the quantum circuit [4]. Furthermore, the circuit width (Q4), that is, the number of used qubits and circuit depth (Q5), that is, the maximum number of gates that are executed sequentially on a qubit, have to be analysed and stored [11]. The circuit width and depth can be used to estimate if the quantum circuit can be executed on a quantum computer based on its current characteristics (see Section 3.2) [4, 29]. Another important provenance attribute is the circuit size (Q6), that is, the total number of gates that are executed in the quantum circuit, as this influences the cumulative gate error that is reflected in the execution results [27, 30]. Finally, the input data of the implemented quantum algorithm, for example, the number to factorise for Shor’s algorithm [31], have to be encoded into the quantum circuit by adding a corresponding initialisation circuit to the beginning of the original circuit [4, 6]. Thereby, different encodings (Q7) exist, such as the amplitude or angle encoding, and the used encoding should be collected as provenance data [32]. The selected encoding influences the final depth and width of the quantum circuit, and thus the error probability of the quantum circuit execution [33]. Based on this data, results from quantum circuits using different encodings can be compared, which can serve as a basis to select a suitable encoding for other quantum circuits.

### 3.2 Quantum computer category

The second category contains all provenance attributes related to quantum computers and their hardware characteristics. The first attribute that should be collected is the number of qubits (QC1) that are provided by a quantum computer. This provenance attribute can be used to select a suitable quantum computer for the execution of a quantum circuit, as the number of provided qubits has to be greater or equal to the circuit width [4, 11]. Additionally, it is also important for a later analysis of the execution results [8]. For example, if the number of provided qubits is significantly greater than the circuit width, the unused qubits could be used for error-correction codes to reduce the influence of noise in the result [34, 35]. Next, the decoherence times (QC2) of the various qubits should be gathered as they limit the maximum executable circuit depth.
This is a composite provenance attribute, which can be further refined and comprises the T1 and T2 times. The attribute changes over time and may differ notably between two calibrations of the quantum computer [10]. Hence, it must be determined periodically by a provenance system. In a quantum computer, the qubits are interconnected in a so-called topology, and gates operating on two qubits can only be executed on directly connected qubits [4, 26]. Thus, if a gate is to be performed on not directly connected qubits, additional SWAP gates must be inserted, increasing the error probability [18]. Therefore, the qubit connectivity (QC3) is an important provenance attribute. Furthermore, quantum computers only implement a limited gate set (QC4) physically, and other gates have to be mapped to a subroutine by the quantum compiler [1, 18]. The gate fidelities (QC5) and gate times (QC6) for the complete gate set of the quantum computer influence the execution time and error probability of quantum circuits and are crucial provenance attributes [8]. Thereby, the fidelity of one-qubit gates on all qubits and two-qubit gates on connected qubits should be recorded, as they may differ significantly [4, 10]. Lastly, the accuracy of measurements on all qubits is gathered as readout fidelities (QC7) or readout errors [8, 12]. This data is the basis for mitigating their influence on the final result distribution of a quantum circuit execution (see Section 3.4) [25].

3.3 | Compilation category

The quantum compiler is in charge of mapping the abstract quantum circuit to the machine instructions for the execution on a concrete quantum computer [1, 18]. For this, it assigns the qubits assumed by the quantum circuit to the real qubits offered by the quantum computer. Due to the different decoherence times and connectivities of the qubits, different assignments lead to varying execution times and error probabilities [4, 26]. Therefore, it is important to collect the qubit assignments (C1) performed by the compiler as provenance data. In the same way, gates defined in the quantum circuit have to be mapped to gates provided by the quantum computer. Thereby, gates that are not physically implemented have to be mapped to a subroutine of provided gates that realises the required gate [18]. There are many possible mappings for a gate, and each of these mappings influences the execution time and error probability. Hence, the performed gate mappings (C2) have to be gathered. Additionally, quantum compilers can often be configured to optimise the mappings regarding a certain optimisation goal (C3), such as the circuit size or resulting accuracy [1, 8, 30]. To ensure reproducibility and compare the results for various optimisation goals, this information can be important provenance data. As the mapping of the qubits and gates is an NP-hard problem, often randomised compilers are used [11]. Thereby, the random seed (C4) should be collected, as the resulting mappings are otherwise not reproducible [26]. The last attribute is the compilation time (C5), which can differ significantly for various quantum compilers or optimisation goals [18, 26].

3.4 | Execution category

In the last category, provenance data about the execution of a quantum circuit is considered. Thereby, the input data (E1) for the execution and the produced output data (E2) must be gathered [28]. This allows comparing the results of executions on different quantum computers with diverse hardware characteristics [8]. The output data is usually a probability distribution of results, which occur when executing the circuit multiple times [4, 36]. The number of executions is referred to as the number of shots (E3) and is collected as provenance data [1]. An insufficient number of shots increases the influence of statistical errors, and thus, is an important attribute to analyse unexpected errors. Furthermore, intermediate results (E4) can help to increase the understandability of quantum computations [8]. However, as measurements destroy the superposition of a qubit, the collection of such data is usually not possible [2]. Exceptions are so-called variational algorithms, such as VQE [37] or QAOA [38], for which multiple iterations of quantum and classical processing occur [39]. Therefore, the intermediate results from each iteration should be gathered for variational algorithms. Also, the number of iterations (E5) that are required can vary depending on the input data and used quantum computer [39]. Another important provenance attribute is the execution time (E6) of the quantum circuit and the whole hybrid application, which may comprise the execution of classical software artefacts, for example, for Simon’s [40] or Shor’s algorithm [31], and multiple iterations for variational algorithms. Finally, the influence of readout errors can be reduced using readout-error mitigation techniques (E7), and the applied technique should be recorded to enable a comparison of different techniques [12, 25].

4 | QPROV: A PROVENANCE SYSTEM FOR QUANTUM COMPUTING

In this section, we present QProv, a provenance system for quantum computing, which enables to collect the provenance attributes described in the previous section (RQ2). Furthermore, it provides the functionality to query, visualise, and analyse the gathered data.

4.1 | Architecture of the provenance system

In the following, we introduce the system architecture of QProv and the related components, as shown in Figure 3. The QProv UI consists of two components: (i) the Visualiser, which enables to graphically display collected provenance data, for example, the temporal evolution of qubit decoherence times or gate errors, and (ii) the Querying Tool, which allows retrieving specific provenance data for a certain use case. The QProv Backend provides an HTTP REST API to enable the communication with the UI components, as well as external components. Other components in the backend are the Provenance Import/Export, which allows to import and
export provenance data, and the **Provenance Query Processor** to handle the queries that are created through the Querying Tool and to return the requested data. Additionally, the **Provenance Aggregator** provides the functionality to aggregate the collected low-level provenance data to gain additional value, e.g., by calculating the calibration matrix from readout fidelities as presented in Section 5.1. The **Provenance Analysser** provides statistical techniques, as well as machine learning approaches, to analyse the provenance data to retrieve valuable insights. Finally, all collected provenance data is stored in a database, which is managed by the **QProv Repository** component of the backend. Furthermore, there are some external components, which collect provenance attributes from different categories. The **Provenance Collector** periodically accesses the APIs from various quantum cloud offerings, such as IBMQ or Rigetti Forest, to retrieve the current hardware characteristics of their quantum computers. It is plugin based and can be extended for other quantum computers. The **NISQ Analysser** [11] selects a suitable quantum computer for the execution of a quantum algorithm on the given input data. Thereby, the quantum circuit is analysed, and compilers are used to determine the hardware-dependent width and depth. Hence, provenance data from the quantum circuit and compilation category is collected by the NISQ Analysser. Finally, **OpenTOSCA** is a deployment system that can be used to deploy and execute hybrid applications with all required quantum and classical software artefacts, and therefore, to gather provenance attributes from the execution category [41].

### 4.2 Mapping to PROV

To achieve portability of the collected provenance data and interoperability between other provenance systems and analysis services, we utilise the PROV standard [19] introduced in Section 2.1 to represent and store the quantum provenance data. Thereby, it also allows benefiting from different libraries implementing the PROV standard and providing functionalities, such as the import and export of provenance data or the query support [21]. The extensibility mechanism of PROV enables the definition of new entities, activities, and agents for a target domain. Hence, their important attributes can be clearly defined to ease the creation and analysis of provenance graphs. Thus, we extended the PROV meta-model to collect the required provenance data in the quantum computing domain. An excerpt from the extension of the PROV meta-model for quantum computing covering the provenance attributes from the quantum computer category (see Section 3.2) is shown in Figure 4. For example, **Quantum Computers** extend PROV agents, as they perform an activity when executing a quantum circuit. They define the provided **qubits** (QC1 in Figure 2) and the supported **gateSet** (QC4) as attributes. The current characteristics of a quantum computer are described by **Qubit** and **Gate** entities. Thereby, the decoherence times (QC2) are captured by the **t1Time** and **t2Time** attributes and the readout fidelities (QC7) by the **readoutFidelity** attribute. Furthermore, the qubit connectivity (QC3) is represented by the **connectedQubits** attribute. Additionally, current data about gates is stored by the **gateFidelity** (QC5) and **gateTime** (QC6) attributes. The entire extension can be found in the Github repository1 of our prototype. Moreover, an example provenance graph for a quantum computation using our presented meta-model extension is discussed in Section 5.3.

### 4.3 Prototypical implementation

To prove the practical feasibility of our approach, we prototypically implemented the QProv system. The prototype is implemented in Java and is publicly available as an open-source project on Github1. Our provenance meta-model, as well as the import and export of provenance data, rely on the PROV standard. Therefore, we integrated the **ProvToolbox2**, a Java library implementing PROV and related functionalities, into our prototype. We also realised the provenance collector and added a plugin to access the IBMQ API to retrieve

---

1. https://github.com/UST-QuAntl/qprov
2. https://github.com/lucmoreau/ProvToolbox
required provenance data. Furthermore, the collector can also execute calibration circuits to generate data that is not available over the API, for example, to calculate the calibration matrix for a quantum computer (see Section 5.1). The QProv UI is implemented in TypeScript and integrated into the QCAtlas, a platform for sharing quantum software [1, 42]. It enables visualising the current characteristics of quantum computers, their temporal evolution, or provenance graphs of executed quantum circuits (see Section 5). The collected provenance data is stored in a PostgreSQL database, for which weekly backups are performed to guarantee their long-term storage. Thereby, the relational database was selected, as the ProvToolbox already provides corresponding utility functions. However, QProv can also be easily extended to use a NoSQL database to benefit from their scalability and high-availability [43].

5 | CASE STUDY

In this section, we present a case study showing how QProv can be used to reduce the overhead when mitigating readout errors, to visualise the temporal evolution of quantum computer characteristics, and to collect provenance graphs for quantum computations.

5.1 | Error mitigation using the calibration matrix

As already mentioned in Section 2.2, the impact of readout errors on the results of quantum circuit executions can be reduced by using readout-error mitigation or unfolding techniques [8, 44]. Many of these unfolding techniques rely on the so-called calibration or response matrix, which can be determined for quantum computers as depicted in Figure 5 [12]. Thereby, calibration circuits are generated and executed, preparing each possible state in the register of the quantum computer and performing a subsequent measurement [4]. Based on the execution results of the calibration circuits, the calibration matrix is calculated. For this, each result of a calibration circuit execution is normalised and then used as a column of the calibration matrix. Afterwards, the matrix is inverted and can be applied to the results of the quantum circuit executions to get the mitigated results [8, 12]. Therefore, the calibration matrix must be regular, but otherwise, there are different unfolding techniques available, such as the iterative Bayesian unfolding or the iterative dynamically stabilised unfolding [13, 44]. However, for a quantum computer with \( n \) qubits, the determination of the calibration matrix requires the execution of \( 2^n \) calibration circuits, as each of the \( 2^n \) possible states has to be prepared and measured. In addition, a high number of shots is needed to reduce the influence of statistical errors [4, 12]. Hence, the matrix calculation for a single execution of a quantum circuit is inefficient. To increase the efficiency and enable the reuse of the calibration matrices, QProv periodically calculates them for various quantum computers and provides them in a uniform manner through the HTTP REST API. Thereby, the periodic execution of the calibration circuits and the re-calculation of the calibration matrix is required, as the readout errors on the various qubits change over time, and especially between different calibrations [10]. In future work, we plan to integrate more unfolding techniques into QProv to evaluate their characteristics and help the user in selecting a suitable unfolding technique to mitigate errors in his quantum computation.

5.2 | Decoherence times, readout errors, and gate errors

In the following section, we present a subset of the collected provenance data for the ibmq_valencia quantum computer and discuss implications for the development and execution of quantum circuits. The topology of the ibmq_valencia is depicted in Figure 6a, containing five qubits. Thereby, the two qubits \( q_0 \) and \( q_2 \) are symmetrical in the topology, that is, both are only connected to qubit \( q_1 \). Hence, when allocating qubits of a quantum circuit on them, the quantum compiler should consider further hardware characteristics, such as decoherence times, readout errors, and gate errors [10, 18]. Figure 6b displays the temporal evolution of the T2 times over 30 days for the two qubits. In this period, the T2 times differ between 22 and 85 \( \mu s \). The differences in the decoherence times depend on manufacturing and experimental parameters, such as the current temperature, which can change over time and especially between calibrations [10]. Furthermore, sometimes \( q_0 \) provides better decoherence times and on other days \( q_2 \). Thus, current provenance data about the decoherence times of the various
**Figure 5** Periodic determination of the calibration matrix by executing calibration circuits.

**Figure 6** Evaluation of the collected provenance data about ibmq_valencia. (a) Topology of the ibmq_valencia, (b) Decoherence times, (c) Readout-errors, (d) CX gate errors.
qubits is crucial for quantum compilers. The same also applies to the readout errors of the qubits, for which the temporal evolution is shown in Figure 6c. Thereby, the readout errors differ significantly between 3% and 20% during our analysis period. Finally, Figure 6d presents a time series of the error rates of CX gates executed on two different qubit connections. These error rates also change over time and have values between 0.8% and 5.4% in our analysed time frame. Similar to the qubit characteristics, the error rates depend on the current experimental conditions and the quality of the last calibration, which cannot be performed perfectly [10]. Hence, current data about all these provenance attributes are important to achieve good compilation results [18, 26] or support other use cases, such as the selection of a suitable quantum computer for the execution of a certain quantum circuit [8, 11]. However, the possible relationships between the values of the different attributes are an open question and can be further analysed by collecting provenance data over a longer period. The source data used to create the figures, as well as further collected sample provenance data, is available on Github.5

5.3 | Provenance graph for a quantum circuit execution

To demonstrate the usage of QProv, an example provenance graph for the execution of a quantum circuit is depicted in Figure 7. Thereby, the graphical notation of the PROV standard [19], as well as our extension presented in Section 4.2 are used. Some of the attributes, entities, activities, and agents in the provenance graph are omitted for space reasons. However, the complete graph is publicly available on Github using the XML serialization of the PROV standard, as well as the graphical representation exported from QProv.

On the left, the input data for the quantum computation is depicted, that is, the base circuit and the classical input data. The base circuit is initialised with the input data using the basis encoding [32]. This changes the circuit attributes, for example, the depth increases because the initialisation circuit is added to the beginning of the base circuit. Then, the circuit is compiled using the Qiskit transpiler, changing its attributes again. Finally, the circuit is executed on the ibmq_valencia quantum computer. For this, the current characteristics of the quantum computer when executing the circuit are collected in the graph by corresponding qubit and gate entities, as exemplarily shown for qubit q0 with its T1 and T2 times. The last entity in the graph contains the resulting output data of the quantum computation.

By collecting provenance graphs for the execution of quantum circuits, the information and data that influence the results can be visualised and stored for a later analysis. Therefore, the provenance graphs can be used to compare different encoding schemes for the input data, optimisation goals in the quantum compiler, or the usage of various quantum computers for the execution.

6 | DISCUSSION

In this section, we discuss potential application areas for quantum provenance and the QProv system, limitations of our approach, and possible extensions regarding other quantum computing models.

Quantum provenance can be used to support the use cases discussed in Section 2.2 and Section 5, such as the compilation of quantum circuits, the selection of suitable hardware for their execution, or the mitigation of occurring errors. However, there are also additional application areas for quantum provenance, especially in the quantum software engineering domain [45]. For example, a provenance system could be used to document the decisions and actions during the development lifecycle of quantum applications to enable a later analysis and improvement of this process [8, 46]. This might require the collection of additional provenance categories and attributes. However, QProv provides an extensible data model and enables the integration of different components collecting the required data (see Section 4). Other possible use cases in this development lifecycle are decision support systems to select a suitable encoding for the developed quantum circuit [33].

Our quantum provenance approach relies on the availability of various required provenance attributes over the APIs of the quantum hardware providers, which are then periodically retrieved (see Section 4.1). If the required data is not available over the API, it can also be determined by QProv using corresponding calibration circuits as exemplary discussed for the periodic calculation of the calibration matrix (see Section 5.1). However, this experimental determination of provenance data may incur additional monetary costs, limiting the applicability of our approach.

As already discussed, QProv can also be extended to collect provenance attributes targeting other quantum computing models, such as the adiabatic model [47, 48]. For example, when solving a *quadratic unconstrained binary optimisation (QUBO)* problem, a corresponding provenance category could be introduced to gather the relevant provenance attributes about the QUBO, similar to the quantum circuit category for the gate-based quantum-computing model. In the same way, the quantum computer category can be extended to collect the characteristics of adiabatic quantum computers. However, the detailed analysis of the required provenance categories and attributes is out of the scope of this work.

7 | CONCLUSION AND FUTURE WORK

The restricted capabilities of today’s quantum computers lead to difficult challenges when developing and executing quantum circuits, for example, mitigating occurring errors, optimising the quantum circuit during compilation, or selecting a suitable quantum computer. To tackle these challenges, provenance data...
about quantum computers, quantum circuits, and their execution are needed. In this paper, we identified relevant provenance attributes in the area of quantum computing. Furthermore, we presented QProv, a provenance system to automatically collect and store the required provenance attributes.

In future work, we plan to incorporate our quantum provenance system into existing workflow provenance approaches. Quantum computations can be part of a larger workflow [6], for example, including classical pre- and post-processing tasks or tasks that use the results of the quantum computation. Therefore, the collection and analysis of provenance data about the classical, as well as the quantum parts should be integrated. Additionally, we want to collect and evaluate the described provenance data over a longer period of time to obtain valuable insights, such as how often the calibration matrix should be updated to get a good trade-off between the costs of executing the calibration circuits and the quality of the achieved mitigation. Finally, we plan to extend our provenance approach regarding other quantum computing models, for example, the adiabatic model.

ACKNOWLEDGEMENTS
The authors would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in Simulation Technology (EXC 2075 – 390740016) at the University of Stuttgart. This work was partially funded by the BMWi project PlanQK (01MK20005N). Also, we are grateful to IBM for providing open access to its quantum computers.

ORCID
Benjamin Weder 🔗 https://orcid.org/0000-0002-6761-6243
Johanna Barzen 🔗 https://orcid.org/0000-0001-8397-7973
Frank Leymann 🔗 https://orcid.org/0000-0002-9123-259X
Marie Salm 🔗 https://orcid.org/0000-0002-2180-250X
Karoline Wild 🔗 https://orcid.org/0000-0001-7803-6386

REFERENCES
1. Leymann, F., et al.: Quantum in the cloud: application potentials and research opportunities. In: Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER), pp. 9–24. ScitePress Setubal (2020)
2. Nielsen, M.A., Chuang, I.: Quantum Computation and Quantum Information. American Association of Physics Teachers Maryland, US (2010)
3. Rieffel, E.G., Polak, W.H.: Quantum Computing: A Gentle Introduction. MIT Press Cambridge (2011)
4. Leymann, F., Barzen, J.: The bitter truth about gate-based quantum algorithms in the NISQ era. Quan. Sci. Techn. 5(4), 044007 (2020)
5. Preskill, J.: Quantum computing in the NISQ era and beyond. Quantum. 2, 79 (2018)
6. Weder, B., et al.: Integrating Quantum Computing into Workflow Modeling and Execution. In: Proceedings of the 13th IEEE/ACM International Conference on Utility and Cloud Computing (UCC), pp. 279–291. IEEE Manhattan (2020). https://doi.org/10.1109/UCC48980.2020.00046
7. LaRose, R.: Overview and comparison of gate level quantum software platforms. Quantum. 3, 130 (2019)
8. Weder, B., et al.: The quantum software lifecycle. In: Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software (APEQS), pp. 2–9. ACM New York (2020). https://doi.org/10.1145/3412451.3428497
9. National Academies of Sciences: Engineering, and Medicine: ‘Quantum Computing: Progress and Prospects’. National Academies Press Washington (2019)
10. Tannu, S.S., Qureshi, M.K.: Not all qubits are created equal: a case for Variability-Aware Policies for NISQ-era quantum computers. In: Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 987–999. ACM New York (2019). https://doi.org/10.1145/3297858.3304007
11. Salm, M., et al.: The NISQ analyzer: Automating the selection of quantum computers for quantum algorithms. In: Proceedings of the 14th Symposium and Summer School on Service-Oriented Computing (SocSoc), pp. 66–85. Springer Berlin (2020)
12. Maciejewski, F.B., et al.: Mitigation of readout noise in near-term quantum devices by classical post-processing based on detector tomography. Quantum. 4, 257 (2020). http://doi.org/10.22331/q-2020-04-24-257
13. Brenner, L., et al.: Comparison of unfolding methods using RooFit/unfold. Int. J. Mod. Phys. A 35 (24), 2050145 (2020). https://doi.org/10.1142/S0217751X20501456
14. Freire, J., et al.: Provenance for computational tasks: a survey. Comput. Sci. Eng. 10(3), 11–21 (2008)
15. Herschel, M., Diestelkämper, R., Ben Laltmar, H.: A survey on provenance: What for? What form? What from?. VLDB J. 26(6), 881–906 (2017). https://doi.org/10.1007/s00778-017-0486-1
16. Anand, M.K., Bowers, S., Ludäscher, B.: Techniques for efficiently querying scientific workflow provenance graphs. EDBT. 10, 287–298 (2010). https://doi.org/10.1145/1730041.1739078
17. Pérez, B., et al.: A systematic review of provenance systems. Knowl. Inf. Syst. 57(3), 495–543 (2018)
18. Sivarajah, S., et al.: t|ket$: a retargetable compiler for NISQ devices. Quantum Sci. Techn. 6(1), 014003 (2020). https://doi.org/10.1088/2058-9565/ab8e92
19. World Wide Web Consortium: PROV Model Primer. W3C Cambridge (2013)
20. Simmhan, Y.L., Plale, B., Gannon, D.: A survey of data provenance in e-science. ACM SIGMOD Rec. 34(3), 31–36 (2005). https://doi.org/10.1145/1084805.1084812
21. Missier, P., et al.: Provenance graph abstraction by node grouping. School of Computing Science Technical Report Series. (2013)
22. Deelman, E., et al.: Pegasus, a workflow management system for science automation. Future Generat. Comput. Syst. 46, 17–35 (2015). http://doi.org/10.1016/j.future.2014.10.008
23. Ludäscher B., et al.: Scientific workflow management and the Kepler toolbox. In: IEEE 31st International Conference on Computer Design (ICCD), pp. 419–426. IEEE Manhattan (2013). http://doi.org/10.1109/ICCID.2013.6657074
24. Li, X., et al.: Inferring user actions from provenance logs. In: IEEE Trustcom/BigDataSE/ISPA, vol. 1, pp. 742–749. IEEE Manhattan (2015). http://doi.org/10.1109/Trustcom.2015.442
25. Endo, S., Benjamin, S.C., Li, Y.: Practical Quantum Error Mitigation for small molecules and quantum magnets. Nature. 549(7671), 242–246 (2017)
26. Häner, T., et al.: A software methodology for compiling quantum programs. Quantum Sci. Techn. 3(2), 020501 (2018). https://doi.org/10.1088/2058-9565/aa93ec
27. Suchara, M., et al.: QuRE: The Quantum Resource Estimator toolbox. In: IEEE 31st International Conference on Computer Design (ICCD), pp. 419–426. IEEE Manhattan (2013). http://doi.org/10.1109/ICCID.2013.6657074
28. Martinez, D., et al.: An analysis of ontological entities to represent knowledge on quantum computing algorithms and implementations. In: Proceedings of the Conference on Digital Curation Technologies (Quarator). Vol. 2836 of CEUR Workshop Proceedings, pp. 1–9. CEUR-WS.org (2021)
29. Weder, B., et al.: Automated quantum hardware selection for quantum workflows. Electronics. 10(8) (2021)
30. Svore, K.M., et al.: A layered software architecture for quantum computing design tools. Computer. 39(1), 74–83 (2006)
31. Shor, PW.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM Rev. 41(2), 303–332 (1999). https://doi.org/10.1137/s0036144598347011
32. LaRose, R., Coyle, B.: Robust data encodings for quantum classifiers. Phys. Rev. 102(3), 032420 (2020)
33. Weigold, M., et al.: Data encoding patterns for quantum computing. In: Proceedings of the 27th Conference on Pattern Languages of Programs. The Hillside Group (2021)
34. Knill, E., Laflamme, R.: Theory of quantum error-correcting codes. Physical Rev. 53(2), 900–911 (1997). https://doi.org/10.1103/physreva.55.590
35. Reed, M.D., et al.: Realization of three-qubit quantum error correction with superconducting circuits. Nature. 482(7385), 382–385 (2012). https://doi.org/10.1038/nature10786
36. Barzen, J., et al.: Relevance of near-term quantum computing in the cloud: A humanities perspective. In: Cloud Computing and Services Science, vol. 1399, pp. 25–58. Springer, Berlin (2021). https://doi.org/10.1007/978-3-030-72369-9_2
37. Kandala, A., et al.: Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. Nature. 549(7671), 242–246 (2017)
38. Farhi, E., Goldstone, J., Gutmann, S.: A Quantum Approximate Optimisation Algorithm, arXiv 14114028 (2014). https://arxiv.org/abs/1411.4028
39. McClean, JR., et al.: The theory of variational hybrid quantum-classical algorithms. New J. Phys. 18(2), 023023 (2016). https://doi.org/10.1088/1367-2630/18/2/023023
40. Simon, D.R.: On the Power of Quantum Computation. SIAM J. Comput. 26(5), 1474–1483 (1997). https://doi.org/10.1137/s0097539796298637
41. Wild, K., et al.: TOSCA4QC: two modeling styles for TOSCA to automate the deployment and orchestration of quantum applications. In: Proceedings of the 24th International Enterprise Distributed Object Computing Conference (EDOC), pp. 125–134. IEEE Manhattan (2020). https://doi.org/10.1109/EDOC49727.2020.00024
42. Leymann, F., Barzen, J., Falkenthal, M.: Towards a platform for sharing quantum software. In: Proceedings of the 13th Advanced Summer School on Service Oriented Computing. IBM Technical Report, pp. 70–74. Endicott (2019)
43. Gessert, F., et al.: Nosql database systems: a survey and decision guidance. Comput. Sci. Res. Dev. 32(3), 353–365 (2017)
44. Nachman, B., et al.: Unfolding quantum computer readout noise. npj Quant. Inf. 6(1), (2020). http://doi.org/10.1038/s41534-020-00309-7
45. Piattini, M., et al.: Toward a quantum software engineering. IT Professional 23(1), 62–66 (2021)
46. Agrawal, R., Gunopoulos, D., Leymann, F.: Mining process models from workflow logs. In: International Conference on Extending Database Technology, pp. 467–483. Springer, Berlin (1998)
47. Aharonov, D., et al.: Adiabatic Quantum Computation Is Equivalent to Standard Quantum Computation. SIAM Rev. 50(4), 755–787 (2008). http://doi.org/10.1137/071004970
48. Date, P., et al.: Efficiently embedding QUBO problems on adiabatic quantum computers. Quant. Inf. Process. 18(4), 1–31 (2019)

How to cite this article: Weder, B., et al.: QProv: A provenance system for quantum computing. IET Quant. Comm. 1–11 (2021). https://doi.org/10.1049/qtc2.12012