QuantME4VQA: Modeling and Executing Variational Quantum Algorithms Using Workflows

Martin Beisel, Johanna Barzen, Marvin Bechtold, Frank Leymann,
Felix Truger, Benjamin Weder

University of Stuttgart, Institute of Architecture of Application Systems, Germany,
{lastname}@iaas.uni-stuttgart.de

QuantME4VQA: Modeling and Executing Variational Quantum Algorithms using Workflows

Martin Beisel[a], Johanna Barzen[b], Marvin Bechtold[c],
Frank Leymann[d], Felix Truger[e], Benjamin Weder[f]
Institute of Architecture of Application Systems, University of Stuttgart, Germany
{lastname@iaas.uni-stuttgart.de}

Keywords: Quantum Workflows, Quantum Computing, Variational Quantum Algorithms, Modeling Extension

Abstract: The execution of quantum algorithms typically requires classical pre- and post-processing, making quantum applications inherently hybrid. Classical resources are of particular importance for so-called variational quantum algorithms, as they leverage classical computational power to overcome the limitations imposed by today's noisy quantum devices. However, the additional complex tasks required by these algorithms, complicate the challenge of integrating quantum circuit executions with classical programs. To overcome this challenge, we leverage the advantages and versatility of workflows, and introduce a workflow modeling extension for orchestrating variational quantum algorithms. The modeling extension comprises new task types, data objects, and a comprehensible graphical notation. Furthermore, we ensure interoperability and portability by providing a method for transforming all new modeling constructs to native modeling constructs, and showcase the practical feasibility of our modeling extension by presenting a system architecture, prototype, and case study.

1 INTRODUCTION

Quantum computing enables to solve various complex problems, e.g., in machine learning (DeBenedictis, 2018) or chemistry (Cao et al., 2018), in a faster, more precise, or energy-efficient manner. The computational advantage of quantum algorithms over their classical counterparts is achieved by leveraging quantum mechanical phenomena, such as superposition and entanglement (Preskill, 2018). Today's quantum devices are still error-prone and limited in their number of qubits (Leymann and Barzen, 2020). However, so-called *Variational Quantum Algorithms (VQAs)* (Cerezo et al., 2021a) have emerged, enabling meaningful computations on these devices by combining classical and quantum resources.

In general, the execution of quantum algorithms requires additional pre- and post-processing steps, such as encoding classical data for quantum devices or evaluating quantum execution results (Leymann and Barzen, 2020). Hence, quantum applications are inherently hybrid, comprising various classical and quantum programs. Analogous to classical software engineering, it is good practice to modularize the functionalities of quantum programs (Beisel et al., 2023). Therefore, developers can reuse existing, well-tested components as building blocks for quantum applications, making their development and operation less time-consuming, error-prone, and expensive.

As a result, the execution of a quantum application requires the orchestration of various quantum and classical programs, which often use different data formats, interfaces, and programming languages (Leymann and Barzen, 2021). A well-established technology for orchestrating heterogeneous processes are workflows (Leymann and Roller, 1999). Workflows provide many advantages, such as robustness, scalability, and transactional processing. Hence, by integrating quantum programs using workflows, they can inherently benefit from these advantages.

Although workflows enable the integration of arbitrary tasks, many quantum-related tasks have specific characteristics and requirements, complicating their modeling and configuration. Therefore, defining quantum workflows requires immense expertise in quantum computing and workflow technologies, as well as deep mathematical knowledge.

[a] https://orcid.org/0000-0003-2617-751X
[b] https://orcid.org/0000-0001-8397-7973
[c] https://orcid.org/0000-0002-7770-7296
[d] https://orcid.org/0000-0002-9123-259X
[e] https://orcid.org/0000-0001-6587-6431
[f] https://orcid.org/0000-0002-6761-6243

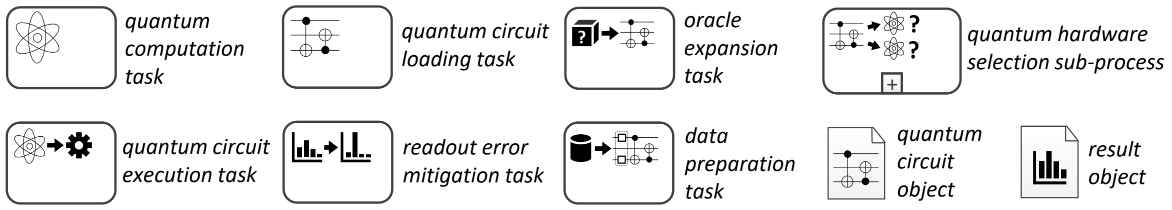
quantum computation task
quantum circuit loading task
oracle expansion task
quantum hardware selection sub-process
quantum circuit execution task
readout error mitigation task
data preparation task
quantum circuit object
result object

Figure 1: Existing QuantME modeling constructs and their graphical notation (based on (Weder et al., 2020b, 2021))

To facilitate the modeling of VQAs using workflows, we (i) extend the *Quantum Modeling Extension* (QUANTME) (Weder et al., 2020b) and introduce explicit modeling constructs for reoccurring tasks. Our extension is called QUANTME4VQA and comprises five new task types and three new data objects. To improve the interoperability of QUANTME4VQA, we (ii) present an approach for the transformation of these modeling constructs into native workflow fragments. Furthermore, we (iii) prove the practical feasibility of our approach by providing a system architecture and a prototypical implementation. Finally, we (iv) demonstrate its suitability in a case study.

The remainder of this paper is structured as follows: In Section 2, fundamentals are described, and the problems tackled in this work are stated. Section 3 introduces QUANTME4VQA and describes all new modeling constructs. Section 4 shows their usage in a case study. Section 5 explains how QUANTME4VQA modeling constructs are transformed into native workflow fragments. Section 6 presents a system architecture for QUANTME4VQA and showcases its functionality by a prototypical implementation. Finally, Section 7 discusses related work and Section 8 summarizes the paper and discusses future work.

2 FUNDAMENTALS & PROBLEM STATEMENT

In this section, we establish fundamentals about the structure and functionality of VQAs, quantum workflows, and QUANTME. Afterward, we discuss existing problems and present our research questions.

2.1 VQAs

VQAs have emerged as a new class of quantum algorithms to overcome the limitations imposed by today's quantum devices (Cerezo et al., 2021a). They follow a learning-based approach, parameterizing gates in quantum circuits and optimizing parameter values in order to find the optimal solution. To determine good circuit parameters, VQAs alternate between executing a parameterized quantum circuit on a quantum device, and classically optimizing its parameters based on the previous execution results. Hence, they work well with shallow quantum circuits, i.e., circuits whose depth is independent of the input size (Bravyi et al., 2020), and can take advantage of the plethora of classical optimization methods. Moreover, promising techniques to further improve the performance of VQAs are being developed, e.g., classical circuit parameter warm-starting (Galda et al., 2021), initial state warm-starting (Egger et al., 2021), or custom optimization techniques (Rad et al., 2022).

2.2 Quantum Workflows & QuantME

Quantum applications comprise a multitude of programs implementing different tasks whose control and data flow must be ensured (Leymann and Barzen, 2021). Workflows are used to orchestrate heterogeneous tasks in various domains, such as business process management or e-Science (Ellis, 1999; Görlach et al., 2011). Thereby, workflow technologies enable the modeling of a set of activities, their partial order, and the data flow using workflow languages, such as *BPMN* (OMG, 2011) or *BPEL* (OASIS, 2007). By modeling and executing quantum applications using workflows, they can benefit from their advantages, e.g., robustness, scalability, and monitoring. The complete workflow model, containing all information required to orchestrate the quantum application, can then automatically be executed using a *workflow engine* (Leymann and Roller, 1999).

However, modeling quantum applications is a complex task that requires knowledge about quantum computing and workflow technologies (Vietz et al., 2021; Weder et al., 2022). To facilitate the modeling process of quantum applications, Weder et al. (2020b) have introduced the workflow modeling extension QUANTME, which is compatible with various imperative workflow languages, such as BPMN or BPEL. It comprises modeling constructs for commonly occurring tasks when orchestrating quantum applications (Weder et al., 2020a, 2021). An overview of all QUANTME modeling constructs is shown in Figure 1. These modeling constructs are tailored for the requirements of quantum-specific processing steps and ease the understanding and configuration for workflow modelers. For example, the quantum circuit execution task has properties defining which quantum provider and device shall be used. Furthermore, QuantME improves the understandability of the workflow by clearly visualizing task types and endorses good software design by advocating well-established concepts such as modularization.

Since existing workflow engines do not natively support QuantME modeling constructs, a transformation step is necessary that converts all QuantME modeling constructs into native modeling constructs. This transformation is done using reusable workflow fragments. A description of the method can be found in Section 5. Thus, no workflow engine extensions are needed, and QuantME can be used to integrate quantum tasks in existing workflows.

2.3 Problem Statement

Despite the availability of a custom-tailored quantum modeling extension, the orchestration of VQAs remains difficult, as the original set of QuantME modeling constructs does not focus on VQAs and hence, does not support crucial tasks of a VQA, e.g., parameter optimization. Typically, these tasks are complex, requiring knowledge about quantum computing, as well as a deep understanding of the used algorithms and implementations. Hence, non-experts are unable to exchange single constituents, e.g., an optimization algorithm, to compare different implementations. Thus, additional modeling constructs are required to support workflow modelers in defining and configuring all VQA steps in an easy-to-understand manner. This leads us to our research question:

"What workflow modeling constructs are required to facilitate the modeling of VQAs and what configuration options do they need to provide?"

3 MODELING VARIATIONAL QUANTUM ALGORITHMS

In the following, we present QuantME4VQA, an extension to QuantME facilitating the modeling of VQAs in workflows. Since QuantME4VQA is meant to be used in combination with QuantME, and QuantME modeling constructs are essential building blocks of VQAs, we refer to all defined modeling constructs using QuantME4VQA. For visualization, we employ the widely used graphical notation of the BPMN standard (OMG, 2011).

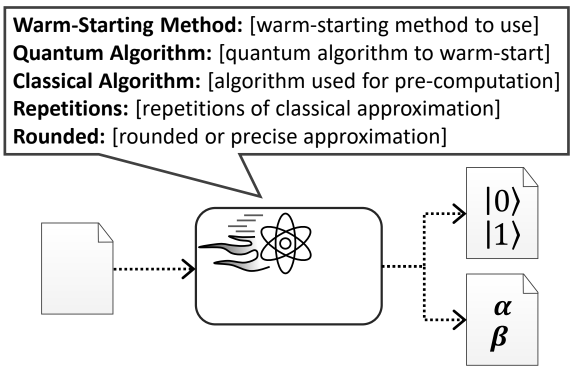
Warm-Starting Method: [warm-starting method to use]
Quantum Algorithm: [quantum algorithm to warm-start]
Classical Algorithm: [algorithm used for pre-computation]
Repetitions: [repetitions of classical approximation]
Rounded: [rounded or precise approximation]
$|0\rangle$
$|1\rangle$
α
β

Figure 2: Overview of the warm-starting task's properties, required input, and produced output

3.1 Warm-Starting VQAs

To improve the performance of existing quantum algorithms, so-called warm-starting methods were developed, which can be divided into two categories: They either pre-compute (i) improved initial values for the VQA's parameters (Galda et al., 2021) or (ii) an approximate solution of the targeted problem instance that can be used to prepare a biased initial state for the quantum circuit to start closer to the optimal solution (Egger et al., 2021).

To facilitate the integration of these methods, the *warm-starting task* is introduced as a new task type. Its semantics is the preparation of initial parameters or approximate solutions to improve the performance of a quantum algorithm. Figure 2 gives an overview of the warm-starting task's properties, required input, and the produced output. It can be configured using the following five properties: (i) the *Warm-Starting Method* identifies the technique that is applied to warm-start the quantum algorithm, (ii) the *Quantum Algorithm* identifies which type of quantum algorithm is warm-started, e.g., the Quantum Approximate Optimization Algorithm (Farhi et al., 2014), (iii) an optional *Classical Algorithm* used for the approximation, (iv) an optional *Repetitions* property stating the number of times the approximation algorithm shall be executed, and (v) an optional *Rounded* property defining whether the approximation returns a rounded or precise result. The warm-starting task's input data object contains, e.g., the problem instance that shall be solved by the algorithm, such as a specific graph. Depending on the type of warm-starting method, the warm-starting task produces either an *initial state object* containing information about the initial state that shall be used by the quantum circuit, or a *parameterization object* that contains the pre-computed parameters to be used as starting points for the circuit parameter optimization process, as its output.

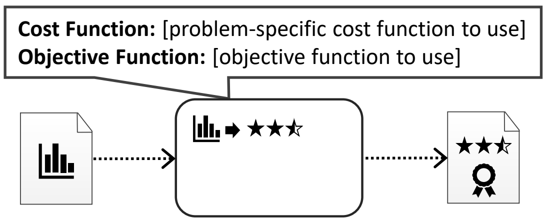
Cost Function: [problem-specific cost function to use]
Objective Function: [objective function to use]

Figure 3: Overview of the result evaluation task's properties, required input, and produced output

3.2 Evaluating Results

A quantum algorithm's result is determined by measuring the quantum device's state once the execution of the algorithm is complete (Kaye et al., 2006). When measuring a quantum state, it collapses into a classical state, represented by a bit-string. However, due to the probabilistic nature of quantum mechanics, the measured bit-string does not necessarily match the expected result (National Academies of Sciences, Engineering, and Medicine, 2019). Hence, quantum circuits are executed and measured multiple times, such that accurate estimations of the probability distributions are retrieved. To interpret these probability distributions, they need to be evaluated classically. For VQAs, an algorithm- and problem-specific cost function understanding the bit-string's semantics is used to evaluate each measured bit-string (Cerezo et al., 2021b). A bit-string representing a good solution is given a low cost value, whereas bad solutions are rated with a high value. To compute a single value describing the overall quality of the execution results based on the measured bit-strings' frequencies and costs, an objective function, e.g., expectation value, is used. Subsequently, this so-called objective value can be used to optimize the circuit parameters of a VQA.

The quality of the quantum execution results is analyzed by the *result evaluation task*. Figure 3 showcases the result evaluation task's properties, as well as the required input and produced output objects. It has two properties: (i) the *Cost Function* identifies the problem-specific cost function for determining a bit-string's quality and (ii) the *Objective Function* identifies the function for evaluating the overall quality of the execution result, based on the bit-strings' costs and frequencies. Moreover, the result evaluation task offers different objective function-specific properties, for configuring the hyperparameters of promising objective functions, e.g., for CVaR (Barkoutsos et al., 2020) or Gibbs (Li et al., 2020). The result evaluation task's required input is a *result object* containing the quantum circuit execution results. As an output, it produces an *evaluation result object*, which comprises information about the overall quality retrieved through the evaluation of the input.

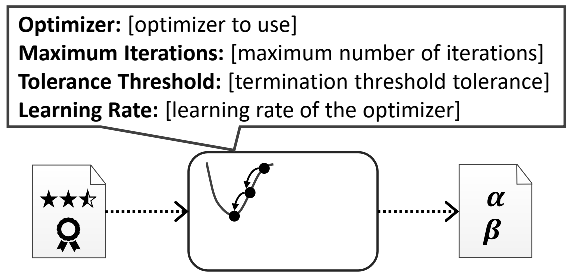
Optimizer: [optimizer to use]
Maximum Iterations: [maximum number of iterations]
Tolerance Threshold: [termination threshold tolerance]
Learning Rate: [learning rate of the optimizer]
α
β

Figure 4: Overview of the parameter optimization task's properties, required input, and produced output

3.3 Optimizing Parameters

The result quality achieved by a VQA heavily depends on the chosen circuit parameters (Cerezo et al., 2021a). Pre-computing suitable initial parameters can lead to good solutions, however, finding optimal parameters is expected to be an NP-hard problem (Bittel and Kliesch, 2021). Hence, VQAs use optimization algorithms, commonly called *optimizers*, to iteratively find better parameters such that a locally optimal solution is found once the optimization process converges. However, the choice of the optimizer and its configuration can affect the convergence time and the final result quality (Pellow-Jarman et al., 2021).

To facilitate a flexible integration and configuration of different optimizers, the *parameter optimization task* is introduced. An overview of its properties, as well as the required input and produced output, are shown in Figure 4. The semantics of the parameter optimization task is the search for improved circuit parameters based on the results achieved with previous parameter values. It has four properties: (i) the *Optimizer* identifying the optimization algorithm that shall be used, (ii) an optional *Maximum Number of Iterations* limiting the duration of the optimization process, (iii) an optional *Tolerance Threshold* used by the optimization algorithm to determine convergence, and (iv) an optional *Learning Rate* property, which can be used to fine-tune the learning rate of the optimization algorithm. The parameter optimization task's required input is a *result evaluation object* describing the quality achieved with the previous iteration's parameters. As an output, it produces a *parameterization object*, which includes the parameters that shall be used in the VQA's next iteration.

3.4 Cutting Quantum Circuits

Quantum circuit cutting describes a set of techniques used to break down a quantum circuit into smaller sub-circuits requiring fewer qubits (Peng et al., 2019; Mitarai and Fujii, 2021). Classical post-processing

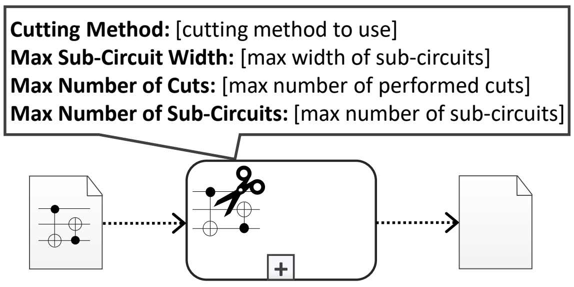
Cutting Method: [cutting method to use]
Max Sub-Circuit Width: [max width of sub-circuits]
Max Number of Cuts: [max number of performed cuts]
Max Number of Sub-Circuits: [max number of sub-circuits]

Figure 5: Overview of the circuit cutting sub-process's properties, required input, and produced output

is used to construct the outcome of the original circuit by using the execution results of the generated sub-circuits. Thus, circuit cutting enables the execution of circuits whose width exceeds a quantum device's number of physically available qubits. Moreover, even if a circuit fits a quantum device, applying circuit cutting can reduce the effect of noise on the result (Tang et al., 2021). To cut a circuit, there are different cutting techniques (Lowe et al., 2022): (i) A *gate cut* replaces a two-qubit gate with a set of single-qubit gates, and (ii) *wire cuts* insert a set of measurement and state-preparation operations. However, the number of sub-circuits grows exponentially with the number of cuts needed to separate the original circuit. Therefore, it is important to separate a quantum circuit with as few cuts as possible.

To facilitate the usage and configuration of different circuit cutting techniques in workflows, we introduce the *circuit cutting sub-process*. Its semantics is the cutting of a quantum circuit into sub-circuits and the combination of their execution result. This means that all quantum circuit execution tasks defined within the sub-processes automatically operate on the cut circuits. Furthermore, all tasks using the execution results, use the probability distribution constructed from the sub-circuit results. In Figure 5 an overview of the circuit cutting sub-process's properties, as well as its required input and produced output, is shown. The circuit cutting sub-process has four properties: (i) the *Cutting Method* identifies which technique should be used to cut the original quantum circuit into smaller sub-circuits, (ii) an optional *Max Sub-Circuit Width* to define the maximum number of qubits used by the resulting sub-circuits, (iii) an optional *Max Number of Cuts* to limit the number of performed cuts, and (iv) an optional *Max Number of Sub-Circuits* limiting the number of sub-circuits the original quantum circuit is cut into. The circuit cutting sub-process requires a *quantum circuit object* containing the quantum circuit that shall be cut and executed as its input, and produces a generic data object containing the problem-specific result of the sub-process as output.

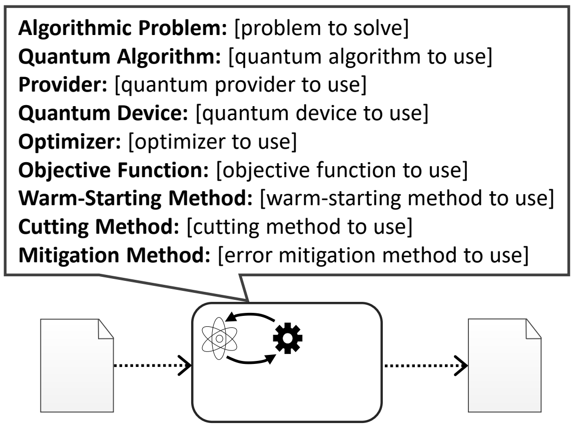
Algorithmic Problem: [problem to solve]
Quantum Algorithm: [quantum algorithm to use]
Provider: [quantum provider to use]
Quantum Device: [quantum device to use]
Optimizer: [optimizer to use]
Objective Function: [objective function to use]
Warm-Starting Method: [warm-starting method to use]
Cutting Method: [cutting method to use]
Mitigation Method: [error mitigation method to use]

Figure 6: Overview of the variational quantum algorithm task's properties, required input, and produced output

3.5 Executing Pre-configured VQAs

Despite the previously introduced task types, orchestrating a VQA is a challenging task for modelers that lack experience with VQAs. To facilitate the integration of pre-configured VQAs, we provide the *variational quantum algorithm task* as a high-level modeling construct. It combines the previously introduced QUANTME4VQA tasks in a single modeling construct, using only their most crucial properties. Figure 6 gives an overview of the task's properties, as well as its input and output. As many properties of the variational quantum algorithm task are part of the previously presented tasks, we only explain the remaining properties here. The *Algorithmic Problem* identifies the problem that shall be solved by the VQA, e.g., the travelling salesman problem. It is used to load a suitable quantum circuit and to select a matching cost function. To execute the quantum circuits, a *Quantum Provider* and a *Quantum Device* must be selected. This can either be done manually or by using a quantum hardware selection sub-process (Weder et al., 2021). The input and output of the task are generic data objects, as they are problem-specific.

4 CASE STUDY

To demonstrate the suitability of the previously introduced modeling constructs, we use them to specify a typical VQA as a workflow. Exemplary, we solve the Maximum Cut (MaxCut) problem, a graph partitioning problem with many application areas (Barrett et al., 2020; Poland and Zeugmann, 2006), using the Quantum Approximate Optimization Algorithm (QAOA) (Farhi et al., 2014). Figure 7 (top) shows a workflow model implementing QAOA for MaxCut using QUANTME4VQA modeling constructs.

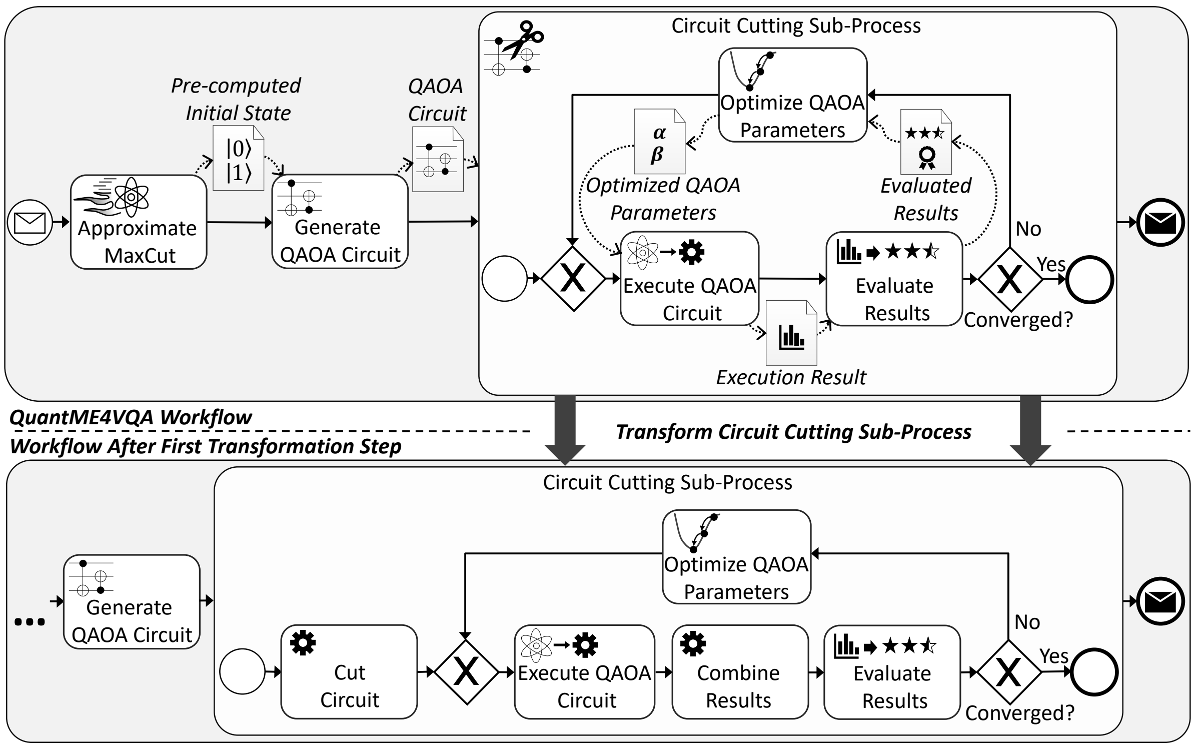

Figure 7: QUANTME4VQA workflow solving MaxCut (top) and transformation of its circuit cutting sub-process (bottom)

The workflow is instantiated when a request message containing a problem instance, i.e., the graph that shall be solved, is received. First, a warm-starting task is utilized to classically approximate a solution by using the Goemans-Williamson algorithm (Goemans and Williamson, 1995). Subsequently, the approximation is used as an initial state when generating the QAOA circuit via a quantum circuit loading task, facilitating the search for the optimal solution (Egger et al., 2021). To reduce the size of the quantum circuit, a circuit cutting sub-process, containing all tasks of the optimization loop, is used. Within the optimization loop, the quantum circuits are executed by a quantum circuit execution task. Subsequently, the quality of the execution results is assessed by a result evaluation task. Next, the evaluated results are used to optimize the QAOA circuit parameters by utilizing a parameter optimization task, which initiates the next iteration of the loop with a new set of QAOA parameters. The optimization loop is stopped once no parameters that further improve the quality of the execution results can be found. Last, the final result is sent to the process initiator via a message end event.

The discussed use-case and a step-by-step tutorial showcasing its setup and execution using the prototype presented in Section 6.2 can be found on GitHub (University of Stuttgart, 2023).

5 TRANSFORMATION METHOD

To improve the portability of QUANTME4VQA modeling constructs, we transform them into native workflow fragments that can be executed on any workflow engine supporting the used workflow language. To ensure compatibility between the existing QUANTME modeling constructs and the ones introduced in this work, we follow the same semi-automatic three-step transformation method (Weder et al., 2020b).

First, the modeler specifies a workflow using native and QUANTME modeling constructs. Then, all QUANTME modeling constructs contained in the workflow are iteratively replaced using so-called *QuantME Replacement Models (QRM)*, which consist of (i) a *detector* and (ii) a *replacement fragment*. The detector determines, whether a modeled QUANTME task can be substituted by the replacement fragment of the QRM or not. In the last step of the transformation, modelers can manually refine the transformed workflow before deploying and executing it.

For the warm-starting, result evaluation, parameter optimization, and variational quantum algorithm tasks, the previously discussed method is applied. However, the transformation of the circuit cutting sub-process requires an extension of the method. Circuit cutting is a two-step process, consisting of a circuit cutting and a result combination step, which are

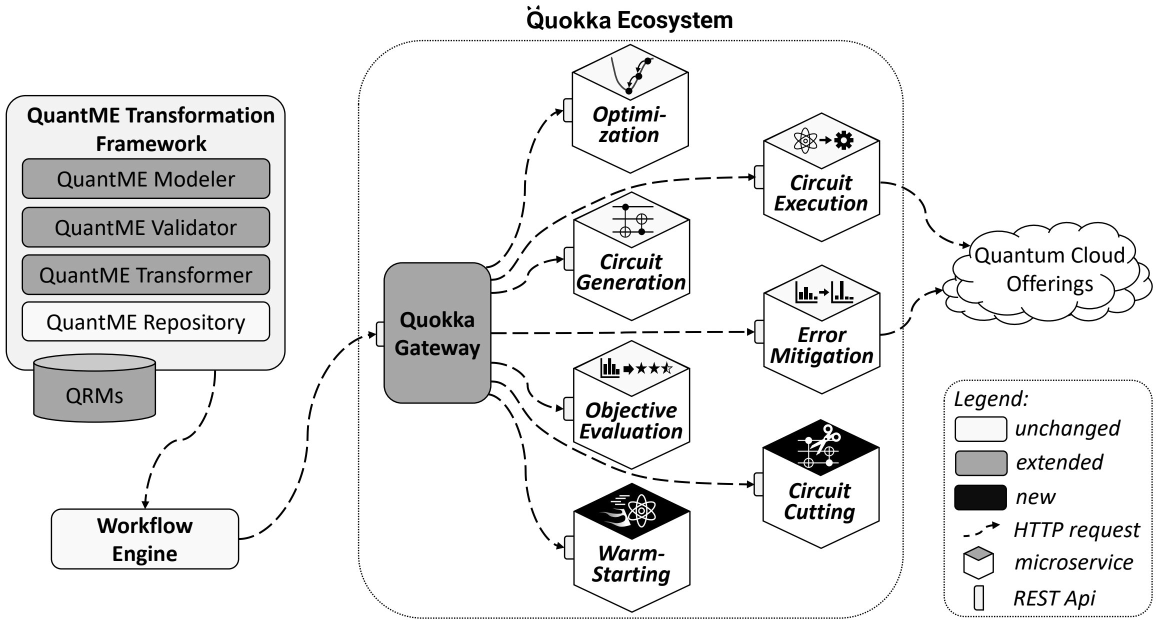

Figure 8: System architecture for modeling and executing QUANTME4VQA workflows

not performed in immediate succession. In an intermediate step, the circuits need to be executed, and error mitigation can be used to improve result quality.

Figure 7 (bottom) shows the positioning of circuit cutting and result combination when transforming the previously discussed workflow. As cutting the circuit in each iteration of the loop leads to significant overhead, it is performed before the loop starts. Thus, the transformation routine must determine the correct position of the circuit cutting and result combination modeling constructs, depending on the tasks contained in the given circuit cutting sub-process.

To fulfill the requirements identified above, we define the following transformation rules: (i) Replacement fragments for the circuit cutting sub-process must consist of two modeling constructs that are either tasks or sub-processes, where the first implements circuit cutting and the second implements result combination. (ii) The modeling construct implementing circuit cutting is inserted directly after the start event of the circuit cutting sub-process. (iii) The modeling construct implementing result combination is placed after each quantum circuit execution task within the circuit cutting sub-process. If a quantum circuit execution task is directly followed by an error mitigation task, it is instead inserted after this task. (iv) All quantum circuit execution tasks within the circuit cutting sub-process receive the sub-circuits as their input. (v) Circuit cutting sub-processes must be transformed before the other QUANTME modeling constructs contained in them. Subsequently, these remaining modeling constructs must be transformed.

6 PROTOTYPICAL EVALUATION

In this section, we show the practical feasibility of our approach by presenting a suitable system architecture and providing a prototypical implementation for it.

6.1 System Architecture

To realize our approach, we propose a system architecture comprising a workflow framework and a quantum service ecosystem introduced in previous works (Weder et al., 2020b; Beisel et al., 2023). Figure 8 gives an overview of all components.

The *QuantME Transformation Framework* supports the modeling and transformation of quantum workflows. The *QuantME Modeler*, is a graphical modeler, which was extended to support the modeling of workflows using the new QUANTME4VQA task types and their graphical notation. To ensure the validity of these modeling constructs, the *QuantME Validator* checks whether they satisfy all imposed constraints, e.g., required properties and matching data types. Hence, the QuantME Validator had to be extended for the new modeling constructs. The *QuantME Transformer* handles the transformation of all QUANTME4VQA modeling constructs to native BPMN. In this work, it has been extended to support the transformation process of the circuit cutting sub-process as described in Section 5. The *QuantME Repository* manages QUANTME-related data such as QRMs. To support the new modeling constructs, we extended the data storage by several QRMs. To enable an automatic execution of the modeled quantum workflows, they are uploaded to a *Workflow Engine*.

The *Quokka Ecosystem* (Beisel et al., 2023) is an extensible open-source microservice ecosystem, providing a variety of functionalities for executing quantum algorithms. To facilitate the request routing for clients, the *Quokka Gateway* unites the endpoints of all services in a single interface. The *Optimization Service*, handles the parameter optimization for VQAs. The *Circuit Generation Service* provides an interface for generating quantum circuit implementations based on different algorithm-specific parameters. These can be executed using the *Circuit Execution Service*, which accesses different *Quantum Cloud Offerings* to run the circuits. To improve the quality of erroneous circuit execution results, the *Error Mitigation Service* can be used. The quality of the execution results is assessed by the *Objective Evaluation Service*. The new *Circuit Cutting Service* enables cutting a quantum circuit into a set of smaller sub-circuits, whose execution results can then be combined to construct the result of the original circuit. The *Warm-Starting Service* provides different methods for pre-computing circuit parameters and approximate solutions to warm-start quantum algorithms.

6.2 Prototype

In this section, we describe the implementation details of our prototype, which implements the previously shown system architecture. The QuantME Transformation Framework is built on the JavaScript-based standalone version of the *Camunda BPMN Modeler* (Camunda, 2023a) that can be extended via a plugin system. In this work, we extended the plugins enabling the modeling and transformation of quantum workflows, to additionally support all new modeling constructs and transformation rules. To execute the modeled workflows, we utilize the *Camunda BPMN Workflow Engine* (Camunda, 2023b).

The Quokka Gateway is implemented as a *Spring Cloud Gateway* providing a single REST API, which forwards incoming requests to the respective services. To facilitate the integration of code written with mostly Python-based quantum SDKs, all Quokka microservices are implemented in Python. The Circuit Cutting Service provides cutting functionalities, such as the circuit cutting method by Tang et al. (2021), whose implementation is publicly available as part of the Circuit Knitting Toolbox (IBM, 2023). The Warm-Starting Service enables the pre-computation of circuit parameters and initial state approximations, based on a given problem instance. For example, it implements the fixed angle conjecture for QAOA on regular MaxCut graphs (Wurtz and Lykov, 2021) and MaxCut approximation using the Goemans-Williamson algorithm (Goemans and Williamson, 1995) for warm-started QAOA (Egger et al., 2021).

7 RELATED WORK

Agnostiq provides *Covalent* (Agnostiq, 2023), a Pythonic open-source tool that specializes in orchestrating heterogeneous computing tasks, such as quantum or HPC tasks. To generate a workflow, developers annotate functions in their code with Covalent-specific decorators. Workflows can be executed locally or deployed in the cloud and subsequently be monitored via a user interface. Zapata Computing's workflow tool *Orquestra* (Zapata Computing, 2023) focuses on orchestrating quantum applications. Similar to Covalent, developers define workflows by using code decorators. Furthermore, Orquestra provides functions that support users in generating and executing quantum circuits. In contrast to our approach, these two frameworks do not provide graphical modeling capabilities or quantum task types to facilitate the configuration and integration of common quantum tasks based on existing reusable implementations. Furthermore, our approach prevents a vendor lock-in by supporting different workflow languages, that enable features such as transactional processing.

Different works employ reusable workflow fragments to refine workflows. Sethi et al. (2012) demonstrate the usability of reusable workflow fragments in different domains and analyze how they affect development time. Atkinson et al. (2017) highlight the benefits of reusable workflow fragments and predict their growing importance. Belhajjame and Grigori (2021) analyze the reusability of service-based workflow fragments and discuss how it can be improved. Eberle et al. (2009) introduce a workflow fragment repository for composing new workflows by reusing existing workflow fragments. Képes et al. (2016) introduce a method that enables a situation-aware adaptation of workflows using workflow fragments.

Several works showcase similar workflow modeling extensions for different domains. Falazi et al. (2019) introduce BLOCKME, a modeling extension for integrating blockchain operations in workflows, and present an approach for transforming it into standard-compliant BPMN. Graja et al. (2016) propose a BPMN modeling extension for the integration of cyber-physical systems. Breitenbücher et al. (2015) introduce SITME, a modeling extension for situational dependencies that introduces a new event type and the concept of situational scopes.

8 CONCLUSION & OUTLOOK

VQAs enable meaningful computations on today's quantum devices. However, their high complexity makes their orchestration complicated and error-prone. To facilitate the orchestration of VQAs, we introduced the language-independent workflow modeling extension QuantME4VQA, which provides custom-tailored modeling constructs for different tasks of a VQA. It comprises new task types and data objects, as well as a graphical notation to ease workflow modeling and understanding. To ensure the interoperability and portability of workflow models using our modeling extension, we presented an approach to transform them into native workflow models. Finally, we validated the practical feasibility of QuantME4VQA by presenting a case study as well as a system architecture and prototype supporting it.

In future work, we plan to evaluate our approach for additional use cases and analyze the achieved degree of simplification in a user study. To improve the performance of circuit cutting in workflows, we plan to evaluate what conditions need to be fulfilled to make a parallelized quantum circuit execution using different quantum devices more efficient than the execution on a single quantum device. Based on these results, we want to analyze how parallelizing the execution of sub-circuits on different quantum devices affects the performance of VQAs and how it can be efficiently integrated in the circuit cutting sub-process.

ACKNOWLEDGEMENTS

This work was funded by the BMWK projects *EniQmA* (01MQ22007B), *PlanQK* (01MK20005N), and *SeQuenC* (01MQ22009B).

REFERENCES

Agnostiq (2023). Covalent Platform. https://www.covalent.xyz/.

Atkinson, M., Gesing, S., Montagnat, J., and Taylor, I. (2017). Scientific workflows: Past, present and future. *Future Generation Computer Systems*, 75:216–227.

Barkoutsos, P. K., Nannicini, G., Robert, A., Tavernelli, I., and Woerner, S. (2020). Improving Variational Quantum Optimization using CVaR. *Quantum*, 4:256.

Barrett, T., Clements, W., Foerster, J., and Lvovsky, A. (2020). Exploratory Combinatorial Optimization with Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3243–3250.

Beisel, M., Barzen, J., Garhofer, S., Leymann, F., Truger, F., Weder, B., and Yussupov, V. (2023). Quokka: A Service Ecosystem for Workflow-Based Execution of Variational Quantum Algorithms. In *Service-Oriented Computing – ICSOC 2022 Workshops*. Springer.

Belhajjame, K. and Grigori, D. (2021). *On Reuse in Service-Based Workflows*, pages 77–87. Springer.

Bittel, L. and Kliesch, M. (2021). Training Variational Quantum Algorithms Is NP-Hard. *Physical Review Letters*, 127(12):120502.

Bravyi, S., Gosset, D., Koenig, R., and Tomamichel, M. (2020). Quantum advantage with noisy shallow circuits. *Nature Physics*, 16(10):1040–1045.

Breitenbücher, U., Hirmer, P., Képes, K., Kopp, O., Leymann, F., and Wieland, M. (2015). A Situation-Aware Workflow Modelling Extension. In *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services (iiWAS 2015)*, pages 478–484. ACM.

Camunda (2023a). Camunda BPMN Modeler. https://camunda.com/products/camunda-bpm/modeler.

Camunda (2023b). Camunda BPMN Workflow Engine. https://camunda.com/products/camunda-bpm/bpmn-engine.

Cao, Y., Romero, J., and Aspuru-Guzik, A. (2018). Potential of quantum computing for drug discovery. *IBM Journal of Research and Development*, 62(6):6:1–6:20.

Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S. C., Endo, S., Fujii, et al. (2021a). Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644.

Cerezo, M., Sone, A., Volkoff, T., Cincio, L., and Coles, P. J. (2021b). Cost function dependent barren plateaus in shallow parametrized quantum circuits. *Nature communications*, 12(1):1791.

DeBenedictis, E. P. (2018). A Future with Quantum Machine Learning. *Computer*, 51(2):68–71.

Eberle, H., Unger, T., and Leymann, F. (2009). Process Fragments. In *On the Move to Meaningful Internet Systems: OTM 2009*, pages 398–405. Springer.

Egger, D. J., Mareček, J., and Woerner, S. (2021). Warm-starting quantum optimization. *Quantum*, 5:479.

Ellis, C. A. (1999). Workflow technology. *Computer Supported Cooperative Work, Trends in Software Series*, 7:29–54.

Falazi, G., Hahn, M., Breitenbücher, U., and Leymann, F. (2019). Modeling and Execution of Blockchain-aware Business Processes. *SICS Software-Intensive Cyber-Physical Systems*, 34(2):105–116.

Farhi, E., Goldstone, J., and Gutmann, S. (2014). A Quantum Approximate Optimization Algorithm. *arXiv:1411.4028*.

Galda, A., Liu, X., Lykov, D., Alexeev, Y., and Safro, I. (2021). Transferability of optimal QAOA parameters between random graphs. In *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 171–180. IEEE.

Goemans, M. X. and Williamson, D. P. (1995). Improved Approximation Algorithms for Maximum Cut and

Satisfiability Problems using Semidefinite Programming. *Journal of the ACM (JACM)*, 42(6):1115–1145.

Görlach, K., Sonntag, M., Karastoyanova, D., Leymann, F., and Reiter, M. (2011). Conventional Workflow Technology for Scientific Simulation. In *Guide to e-Science*, pages 323–352. Springer.

Graja, I., Kallel, S., Guermouche, N., and Kacem, A. H. (2016). BPMN4CPS: A BPMN Extension for Modeling Cyber-Physical Systems. In *2016 IEEE 25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 152–157.

IBM (2023). Circuit knitting toolbox. https://github.com/Qiskit-Extensions/circuit-knitting-toolbox.

Kaye, P., Laflamme, R., and Mosca, M. (2006). *An Introduction to Quantum Computing*. OUP.

Képes, K., Breitenbücher, U., Sáez, S. G., Guth, J., Leymann, F., and Wieland, M. (2016). Situation-Aware Execution and Dynamic Adaptation of Traditional Workflow Models. In *Proceedings of the 5th European Conference on Service-Oriented and Cloud Computing (ESOCC)*, pages 69–83. Springer.

Leymann, F. and Barzen, J. (2020). The bitter truth about gate-based quantum algorithms in the NISQ era. *Quantum Science and Technology*, pages 1–28.

Leymann, F. and Barzen, J. (2021). Hybrid Quantum Applications Need Two Orchestrations in Superposition: A Software Architecture Perspective. *arXiv:2103.04320*.

Leymann, F. and Roller, D. (1999). *Production Workflow: Concepts and Techniques*. Prentice Hall PTR.

Li, L., Fan, M., Coram, M., Riley, P., and Leichenauer, S. (2020). Quantum optimization with a novel Gibbs objective function and ansatz architecture search. *Phys. Rev. Research*, 2:023074.

Lowe, A., Medvidović, M., Hayes, A., O'Riordan, L. J., Bromley, T. R., Arrazola, J. M., and Killoran, N. (2022). Fast quantum circuit cutting with randomized measurements. *arXiv:2207.14734*.

Mitarai, K. and Fujii, K. (2021). Constructing a virtual two-qubit gate by sampling single-qubit operations. *New Journal of Physics*, 23(2):023021.

National Academies of Sciences, Engineering, and Medicine (2019). *Quantum Computing: Progress and Prospects*. The National Academies Press.

OASIS (2007). *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*. Organization for the Advancement of Structured Information Standards (OASIS).

OMG (2011). *Business Process Model and Notation (BPMN) Version 2.0*. Object Management Group (OMG).

Pellow-Jarman, A., Sinayskiy, I., Pillay, A., and Petruccione, F. (2021). A Comparison of Various Classical Optimizers for a Variational Quantum Linear Solver. *Quantum Information Processing*, 20(6):1–14.

Peng, T., Harrow, A., Ozols, M., and Wu, X. (2019). Simulating Large Quantum Circuits on a Small Quantum Computer. *Physical Review Letters*, 125:150504.

Poland, J. and Zeugmann, T. (2006). Clustering Pairwise Distances with Missing Data: Maximum Cuts Versus Normalized Cuts. In *Discovery Science*, pages 197–208. Springer.

Preskill, J. (2018). Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79.

Rad, A., Seif, A., and Linke, N. M. (2022). Surviving The Barren Plateau in Variational Quantum Circuits with Bayesian Learning Initialization. *arXiv:2203.02464*.

Sethi, R. J., Jo, H., and Gil, Y. (2012). Re-Using Workflow Fragments Across Multiple Data Domains. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pages 90–99. IEEE.

Tang, W., Tomesh, T., Suchara, M., Larson, J., and Martonosi, M. (2021). CutQC: Using Small Quantum Computers for Large Quantum Circuit Evaluations. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, pages 473–486. ACM.

University of Stuttgart (2023). Quantum workflows, modulo, and quantme use cases. https://github.com/UST-QuAntiL/QuantME-UseCases.

Vietz, D., Barzen, J., Leymann, F., Weder, B., and Yussupov, V. (2021). An Exploratory Study on the Challenges of Engineering Quantum Applications in the Cloud. In *Proceedings of the 2nd Quantum Software Engineering and Technology Workshop (Q-SET21)*, pages 1–12. CEUR Workshop Proceedings.

Weder, B., Barzen, J., Leymann, F., and Salm, M. (2021). Automated Quantum Hardware Selection for Quantum Workflows. *Electronics*, 10(8).

Weder, B., Barzen, J., Leymann, F., Salm, M., and Vietz, D. (2020a). The Quantum Software Lifecycle. In *Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software (APEQS 2020)*, pages 2–9. ACM.

Weder, B., Barzen, J., Leymann, F., and Vietz, D. (2022). Quantum Software Development Lifecycle. In *Quantum Software Engineering*, pages 61–83. Springer.

Weder, B., Breitenbücher, U., Leymann, F., and Wild, K. (2020b). Integrating Quantum Computing into Workflow Modeling and Execution. In *Proceedings of the 13th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2020)*, pages 279–291. IEEE.

Wurtz, J. and Lykov, D. (2021). Fixed-angle conjectures for the quantum approximate optimization algorithm on regular MaxCut graphs. *Physical Review A*, 104(5):052419.

Zapata Computing (2023). Orquestra Platform. https://www.zapatacomputing.com/orquestra-platform/.

All links were last followed on March 3, 2023.