qWard: A Unified Toolkit for Pre- and Post-Runtime Quantum Circuit Metrics

1st Cristian Marquez
Department of Systems and Computing Engineering
Universidad de los Andes
Bogotá, Colombia
c.marquezb@uniandes.edu.co

2nd Daniel Sierra
Electrical Engineering and Computer Science
The Catholic University of America
Washington DC, 20064
sierrasosa@cua.edu

3rd Kelly Garcés
Department of Systems and Computing Engineering
Universidad de los Andes
Bogotá, Colombia
kj.garces971@uniandes.edu.co

***Abstract*—As quantum computing (QC) matures towards practical applications, effective methods to evaluate quantum algorithms, circuits, and execution quality in quantum software are gaining more relevance. To apply such methods, a set of meaningful metrics must be defined, understood, calculated, and applied. In an effort to solve that problem, this paper categorizes quantum related metrics into pre-runtime (derived from the static analysis of circuits before execution) and post-runtime (derived from the analysis of execution results). It is worthwhile to mention that while popular quantum SDKs such as Qiskit, PennyLane, and Q# offer some circuit characteristics, there is often a gap between these native capabilities and the broader set of insightful pre- and post-runtime metrics in the literature. This paper introduces *qWard*, a Python library developed to bridge this gap by offering an initial implementation for the Qiskit SDK. *qWard* provides developers and researchers with a toolkit to calculate and visualize key pre-runtime and post-runtime metrics, facilitating the analysis of quantum circuits quality. The scope of this first qWard version is to provide support for the Qiskit SDK and the Qiskit AER simulator.**

***Index Terms*—Quantum computing, software quality, open-source library, Qiskit**

I. Introduction

QC is an emerging field based on quantum mechanics principles [1], enabling systems to leverage phenomena such as superposition and entanglement. This novel approach has the potential to address computational challenges currently impossible for classical hardware, promising advances in areas such as large-scale number factorization [2], cryptography [3], and complex model training [4], [5].

Despite its promise, QC is in its early stages and significant software engineering challenges must be overcome for its broad adoption. Such challenges include the distinction between logical and physical qubits, error correction, circuit optimization, long queue times, and the high price of quantum runtime services [6], [7].

The high cost associated with quantum runtime services represents a critical challenge [1], emphasizing the necessity for the development of robust methodologies and tools for quantum circuit analysis prior to execution.

[1]For example, by June of 2025, IBM offers services starting at \$1.6 USD per second for its pay-as-you-go access (see https://www.ibm.com/quantum/pricing), while Amazon Braket charges a per-task fee of \$0.30 plus a variable per-shot fee for on-demand QPU access (see https://aws.amazon.com/braket/pricing/).

To effectively perform such analyses and evaluate the overall quality of any given software asset, the use of metrics is essential. In the context of QC, the development process typically involves two global steps: first, developers prepare quantum states by applying gates and planning measurements, and second, execute the circuit to collect results. This workflow, which includes some intermediate steps such as transpilation, calibration, and optimization before actual execution, naturally lends itself to temporal metrics categorization. Consequently, this paper adopts the terms **pre-runtime metrics** for those measures computed after coding the algorithm but before its execution (e.g., number of gates, circuit depth and width) and **post-runtime metrics** for those derived from the execution (e.g., success rates, fidelity, execution time).

While classical software engineering benefits from established quality attributes and metrics, as surveyed in works like [8], their direct application to quantum systems is not always straightforward. Although some research suggests that traditional metrics can be applied using black-box methods [9], the unique characteristics of QC demand the development and adoption of quantum-specific metrics.

The literature identifies various categories of pre-runtime metrics. These metrics often focus on *maintenance aspects*, for instance, the complexity of gate operations, which cover metrics such as the number of qubits and the number of 2-gate operations [10]–[12]. Additionally, they address *circuit complexity aspects*, which includes structural attributes such as circuit width, depth, and gate complexity [12].

Complementing the pre-runtime perspective, post-runtime metrics commonly analyze *performance-related aspects* like execution time, precision, and accuracy [10], [13], *hardware quality aspects* such as qubit coherence times (T1, T2) and gate error rates [14], and *correctness and error aspects*, often evaluating circuit fidelity or success rates, for instance, through protocols like randomized benchmarking [15].

The classification of metrics into pre-runtime and post-runtime categories facilitates the systematic organization of the diverse array of metrics published in the literature and, at the same time, it helps to identify targets for understanding the quality of quantum circuits. Although there are known quantum resource estimates (QREs) such as Azure Quantum Resource Estimator from Microsoft [16], Qualtran from Google [17], BenchQ from Zapata AI [18], QuRE toolkit from University of California [19], and The Munich Quantum Toolkit supported by European Research Council [20], such tools are primarily focused on specific hardware or their usage is limited to estimating resources for fault-tolerant computing, and generally do not offer a broad toolkit for calculating and correlating a diverse range of metrics.

Consequently, the current study is motivated by the need for a tool to collect and analyze a wide spectrum of metrics. This motivation leads to the following research questions.

RQ1 : To what extent do existing QC SDKs facilitate the collection of pre- metrics and the calculation of post-runtime metrics?

RQ2 : Do QC SDKs effectively incorporate emerging metrics as reported in the literature?

RQ3 : How can a library be designed to address the need to collect and analyze a wide spectrum of quantum circuit metrics?

An outline of the paper follows. Section II provides background on quantum circuit metrics. Section III then details the metric strategies considered in our approach, while Section IV presents a comprehensive overview of the qWard library. Section V demonstrates the practical application of the library through code examples. Section VI elaborates on how our proposal deals with the posed research questions. Finally, Section VII concludes with a summary of the findings and a discussion of future directions.

II. Quantum Circuit Metrics Background

This section presents the foundational knowledge that precedes the development of the library. First, the QC workflow is explained, followed by clarification of the metric classification employed by qWard, and finally a review of state-of-the-art quantum metrics and existing analogous tools.

A. Quantum Computing Workflow and Measurement Challenges

The deterministic nature inherent in classical computing enables a development workflow in which developers remain unquestioning of their results. However, quantum phenomena are stochastic, which impedes developers from trusting their result after a single execution. As a consequence of this, the QC results are often presented as a histogram, and a new mindset of development appears which aims to design QC algorithms to maximize the chances of getting the right value.

To address this challenge, runtime quantum services introduce three distinct concepts to mitigate the inherent unpredictability of quantum phenomena. The initial concept, termed **Job**, refers to a singular invocation of the quantum service. Within the framework of a **Job**, the developer specifies the quantity of **Shots**, which denotes the number of iterations the algorithm executes within a single service request. Furthermore, if necessary, developers have the capability to execute **Jobs** in **Batches**, which serves to explore diverse configurations of the runtime service, including adjustments to circuit optimizations, variations of the initial position of the qubits, or changes in the route strategy of the qubits [2], with the objective of refining the circuit outcomes.

Beyond the stochastic nature of quantum computing, the cost of executing a quantum circuit presents a considerable barrier to the development of practical quantum algorithms, primarily due to the requirement of a large number of qubits. Current quantum cloud services charge considerable fees for runtime access, making the development and experimentation an expensive process. This economic reality demands the exploration of alternative approaches to pre-optimize and predict quantum circuit behavior utilizing classical resources prior to their execution on a QPU.

In light of this necessity, comprehensive metric analysis offers a promising solution to this challenge. The core concept involves extracting an extensive array of pre-runtime metrics that effectively encapsulate the key characteristics of a quantum circuit. The calculated array can serve as a *footprint*, effectively capturing the core attributes of the algorithm. Furthermore, employing a footprint mitigates the dimensional complexity of the problem, thereby facilitating the simulation of quantum systems. This transformation converts the challenge of manipulating large matrices into a vector analysis problem.

Therefore, the development of a comprehensive metric framework like qWard is not merely an academic exercise, but a practical necessity for making quantum computing development economically viable.

B. Metric Classification: Pre-runtime and Post-runtime

As introduced in Section I, quantum circuit metrics naturally divide into two temporal categories based on when they can be calculated relative to circuit execution.

Pre-runtime metrics are derived directly from the quantum circuit specifications through static analysis. These deterministic metrics, covering circuit depth, gate counts, and complexity indicators, can be extracted employing classical computational resources by modeling the circuit as a multi-connected node problem. In contrast, **post-runtime metrics** are evaluated through observation operations, requiring the utilization of quantum hardware or simulators, and exhibiting stochastic behavior. An example of post-runtime metrics includes success rates, fidelity, and error distributions.

This implementation of temporal classification fulfills the following fundamental objectives: 1) it enables circuit analysis by maximizing the insights gained from pre-runtime analysis before committing to quantum execution; and 2) it aligns with the hierarchical execution model (batches, jobs, shots) by recognizing that while pre-runtime metrics remain constant for a given circuit, post-runtime metrics exhibit statistical variation due to quantum uncertainty.

[2]Consult the official documentation of Qiskit's transpiler for more information regarding the adjustable parameters available within the IBM quantum service.

C. Literature about Quantum Metrics

Several initiatives and studies have been made that address quality of QC, and in that process researchers have introduced an extensive range of metrics and attributes relevant to our field of study. This quick review highlights the work called ”The Talavera manifesto” [21], which outlined the principles of QC design, establishing foundational quality concepts, although without specifying concrete metrics.

Building upon these foundational principles, it is pertinent to acknowledge the contribution of Cruz-Lemus et al. [22], who introduced a tool intended to address a similar issue to that of qWard, although from a different perspective. Cruz-Lemus introduced **QMetrics**, a web-based platform facilitating the computation and visualization of pre-runtime metrics. In contrast, our methodology diverges by offering a library suitable to be called from developers' applications. It additionally incorporates considerations of the capabilities provided by current quantum SDKs, it is not limited to pre-runtime metrics but opens to post-runtime, and it is intentionally designed as open-source.

To better contextualize the contributions of qWard, illustrative examples of quantum metrics proposed in recent literature are provided, categorized according to pre- and post-runtime.

1) Pre-runtime Metrics in Literature: For circuit structure and understandability, Cruz-Lemus et al. [23] and Zhao [24] introduced metrics to quantify the structural properties of quantum circuits. Similarly, Serrano et al. [25] discussed quality aspects such as compatibility and reliability, highlighting the need for standardized measurement approaches.

Complementing these structural approaches, more recently, complexity-focused metrics have emerged. For instance, Shami [26] proposes character complexity as a novel measure for quantum circuit analysis, while Bu et al. [27] explore the relationship between circuit sensitivity, magic, and coherence to runtime complexity. Furthermore, Cruz-Lemus et al. [22] describe additional pre-runtime metrics for circuit complexity and understandability.

2) Post-runtime Metrics in Literature: Post-runtime metric research covers verification methods, performance analysis, and benchmarkings. Beyond the comprehensive verification methods reviewed by Gheorghiu et al. [28], several studies focus on execution-based quality evaluation.

Within this execution-based approach, Lubinski et al. [29] present application-oriented performance benchmarks that evaluate quantum algorithms based on solution quality, result fidelity, and execution time metrics calculated from actual execution data. Cross et al. [30] introduce quantum volume as a holistic metric calculated from randomized circuit executions to characterize overall QPU performance.

From the error characterization perspective, Harper et al. [31] propose methods to generate models of quantum noise using execution results, while Bistron [32] discusses comprehensive benchmarking approaches that analyze post-execution data to assess QC capabilities across multiple dimensions, including gate fidelity and circuit success rates.

D. Gap identification

Despite the wealth of metrics proposed in the QC literature, current SDKs provide limited native support for comprehensive circuit analysis. This gap between theoretical research and practical tooling creates barriers for developers aiming to analyze their quantum algorithms.

More specifically, although Qiskit provides basic circuit properties, such as depth, width, and gate counts, it lacks implementation of advanced metrics widely discussed in the literature. For example, it did not include complexity metrics indicators such as T count, weighted complexity, and parallelism efficiency discussed in [26]; it neither include the characterization of entanglement metrics, such as the density and width of the entanglement gate presented in [27]. The absence of these metrics in the SDK quantum ecosystem forces researchers and developers to implement custom analysis code repeatedly, leading to inconsistent metric definitions and different calculation methodologies.

To bridge this gap, qWard addresses these challenges through three key design decisions. First, it implements the strategy pattern to facilitate the incorporation of new metrics and visualization while maintaining consistency through standardized interfaces. Second, it provides comprehensive schema validation using Pydantic, ensuring type safety and data integrity across all metric calculations, and at the same time provides unitary testing for validation. Third, it includes an integrated visualization system that supports the AER Qiskit Silumator Job and the IBM Quantum Runtime Job v2 to automatically generate relevant graphs of the registered metrics.

III. qWard Metrics

In this section we highlight the metrics considered in our approach which are inspired on the literature and integrated in the toolkit in an unified way.

One of the first pre-runtime metrics included are the *QiskitMetrics*. The reason for selecting Qiskit among the various quantum computing providers available in the market is because IBM offers a growing community of developers, an important number of open source projects using their technology stack [33], comprehensive documentation, and offers a complimentary quantum runtime service time for experimentation.

Beyond native Qiskit metrics, our approach extends its analytical capabilities by incorporating complexity metrics reported in the literature. Specifically, qWard includes an implementation of most of the complexity metrics presented in [27], which are organized under the *ComplexityMetrics* category. These complexity metrics, as defined by the authors, represent *”a measure of the minimum number of gates needed to implement a given unitary transformation”* and provide deeper insights into circuit structure and computational requirements.

Completing the comprehensive analysis toolkit, our approach incorporates post-execution evaluation referred to as

CircuitPerformance. This metrics reflect actual quantum circuit execution results, namely: success rate, fidelity, and statistical analysis including entropy, uniformity, and concentration. These post-runtime metrics enable researchers to assess the practical performance and reliability of their quantum circuits beyond theoretical predictions.

The following subsections include details of the specific metrics included in each category.

A. *QiskitMetrics, native circuit properties*

The **QiskitMetrics** category includes fundamental metrics directly from the **QuantumCircuit** (a class included in the Qiskit SDK). This category encompasses a set of basic circuit metrics, instruction-level metrics, and scheduling information when available.

Although IBM offers an API description in their documentation [34], the information is difficult to navigate because it focuses primarily on library usage, leaving the explanation of available metrics as a secondary topic. Consequently, we have investigated the QuantumCircuit instance to derive pertinent pre-runtime metrics.

The subsequent tables provide an overview of the available Qiskit metrics along with their descriptions.

TABLE I
QISKITMETRICS, BASIC CIRCUIT METRICS

Field Name	Description
depth	The number of steps required to execute a circuit
width	Total number of qubits and classical bits
size	Total number of operations
num_qubits	Number of quantum bits
num_clbits	Number of classical bits
num_ancillas	Number of ancilla qubits
num_parameters	Number of unbound parameters
has_calibrations	Boolean indicating if circuit has calibration data
has_layout	Boolean indicating if circuit has layout information
count_ops	Dictionary counting operations by type

B. *ComplexityMetrics: advanced circuit analysis*

The **ComplexityMetrics** integrates a collection of pre-runtime metrics based on the relationship between the characteristics of a quantum circuit as explained in [27]. The authors present a comprehensive collection of metrics designed for gate-based analysis, entanglement characterization, standardized comparisons, intricate parallelism evaluation, and complexity measure derivations.

The following tables outline the complexity metrics that have been implemented in qWard.

C. *CircuitPerformanceMetrics, execution analysis*

The **CircuitPerformanceMetrics** is designed to calculate post-runtime metrics based on the results of the quantum circuit execution. This category includes the computation of two primary metrics: success rates and fidelity. Additionally, it provides a collection of statistical metrics to support Batch analysis. In particular, this strategy not only facilitates the calculation of success rate and fidelity for individual job but also generates an aggregate analysis to summarize the overall batch execution. Furthermore, it is crucial to highlight that the current version of qWard supports both the AerJob of the AER simulator and the RuntimeJobV2 of Qiskit.

The following tables present the available metrics in detail.

TABLE II
QISKITMETRICS: INSTRUCTION & SCHEDULING METRICS

Field Name	Description
Instruction Metrics	
num_connected_components	Number of connected components
num_nonlocal_gates	Number of multi-qubit gates
num_tensor_factors	Number of tensor factors
num_unitary_factors	Number of unitary factors
instructions	Dictionary of circuit instructions grouped by operation type
Scheduling Metrics	
is_scheduled	Boolean indicating if circuit has been scheduled
layout	Hardware layout and routing information from transpiler
op_start_times	List of operation start times for scheduling
qubit_duration	Total execution time
qubit_start_time	Start timestamp of the circuit execution
qubit_stop_time	Stop timestamp of the circuit execution

TABLE III
COMPLEXITYMETRICS: GATE-BASED ANALYSIS

Field Name	Description
gate_count	Total number of gates
circuit_depth	The number of steps required to execute a circuit
t_count	Number of T gates (non-Clifford gates for fault tolerance)
cnot_count	Number of CNOT gates (two-qubit entangling gates)
two_qubit_count	Total count of two-qubit gates
multi_qubit_ratio	Ratio of multi-qubit gates to total gates

TABLE IV
COMPLEXITYMETRICS: ENTANGLEMENT & STANDARDIZED

Field Name	Description
Entanglement Metrics	
entangling_gate_density	Density of entangling gates relative to circuit size
entangling_width	Width of entanglement spread across qubits
Standardized Metrics	
circuit_volume	Product of depth and width
gate_density	Density of gates per time step
clifford_ratio	Ratio of Clifford gates to total gates
non_clifford_ratio	Ratio of non-Clifford gates (magic gates) to total

TABLE V
COMPLEXITYMETRICS: ADVANCED & DERIVED

Field Name	Description
Advanced Metrics	
parallelism_factor	Average number of gates executed per time step
parallelism_efficiency	Efficiency of parallel gate execution in circuit
circuit_efficiency	Overall resource utilization efficiency
quantum_resource_utilization	Utilization of quantum computational resources
Derived Metrics	
square_ratio	Ratio indicating balanced dimensions
weighted_complexity	Weighted complexity score based on gate types
normalized_weighted_complexity	Complexity score normalized per qubit

TABLE VI
CIRCUITPERFORMANCE: SUCCESS METRICS

Field Name	Description
Single Job Fields	
success_rate	Success rate from 0.0 to 1.0, indicating successful execution
error_rate	Error rate from 0.0 to 1.0 ($1 - success\_rate$)
total_shots	Total number of shots executed
successful_shots	Number of successful shots
Aggregate Analysis Fields	
mean_success_rate	Mean success rate across multiple jobs
std_success_rate	Standard deviation of success rates
min_success_rate	Minimum success rate observed
max_success_rate	Maximum success rate observed
total_trials	Total number of trials across all jobs

TABLE VII
CIRCUITPERFORMANCE: FIDELITY METRICS

Field Name	Description
Single Job Fields	
fidelity	Quantum fidelity from 0.0 to 1.0
has_expected_distribution	Boolean indicating if expected distribution was provided
method	Method used for fidelity calculation (theoretical_comparison or success_based)
confidence	Confidence level of fidelity calculation (high, medium, low)
Aggregate Analysis Fields	
mean_fidelity	Mean fidelity across multiple jobs
std_fidelity	Standard deviation of fidelities
min_fidelity	Minimum fidelity observed
max_fidelity	Maximum fidelity observed

TABLE VIII
CIRCUITPERFORMANCE: STATISTICAL METRICS

Field Name	Description
Single Job Fields	
entropy	Shannon entropy of the measurement distribution
uniformity	Uniformity measure from 0.0 to 1.0
concentration	Concentration measure from 0.0 to 1.0
dominant_outcome_probability	Probability of the most frequent outcome
num_unique_outcomes	Number of unique measurement outcomes
Aggregate Analysis Fields	
mean_entropy	Mean entropy across multiple jobs
mean_uniformity	Mean uniformity across multiple jobs
mean_concentration	Mean concentration across multiple jobs
mean_dominant_probability	Mean dominant probability across multiple jobs
std_entropy	Standard deviation of entropy values
std_uniformity	Standard deviation of uniformity values

IV. QWARD LIBRARY

In this section, we describe qWard, a Python library, designed to assist developers and researchers in analyzing quantum circuit quality based on the aforementioned metrics. It offers features to **calculate** and **visualize** metrics through graphs, thus facilitating informed decision-making during the development phase and supporting the comprehensive evaluation of quantum programs.

To achieve this goal, qWard enhances the quantum libraries and frameworks that allow interaction with a QPU. In particular, the initial version of qWard is focused on augmenting the Qiskit SDK's quality analysis capabilities. To illustrate its functionality, preliminary experiments were executed in both the Qiskit AER simulator and the IBM quantum runtime.

To ensure modularity and extensibility, qWard is designed with a modular approach, underscoring the separation of concerns through the strategy design pattern. According to [35], the strategy pattern is classified as a behavioral design pattern that defines a set of algorithms, encapsulates each, and allows for interchangeability at runtime. The pattern is divided into three main parts: a *Strategy* interface that defines a common contract for all algorithms, *ConcreteStrategy* classes that implement specific algorithms, and a *Context* object to delegate operations.

For in-depth reference and implementation details, detailed documentation of the API and more elaborate diagrams are available in the official documentation repository https://xthecapx.github.io/qiskit-qward/. Additionally, the alpha version can be accessed via the Python package manager PIP at the following URL https://pypi.org/project/qiskit-qward/.

To better understand the library architecture and its implementation, the following sections include a high-level design of the core features using a simplified version of a UML class diagram.

A. MetricCalculator strategy

Building on the foundation of the strategy pattern described above, the qWard **MetricCalculator** feature provides methods to calculate metrics derived from a quantum circuit and its execution results.

Figure 1 showcases the metric calculation strategy of qWard. Due to the limitation of space, the diagram omits details of the properties and methods implemented in some of the classes.

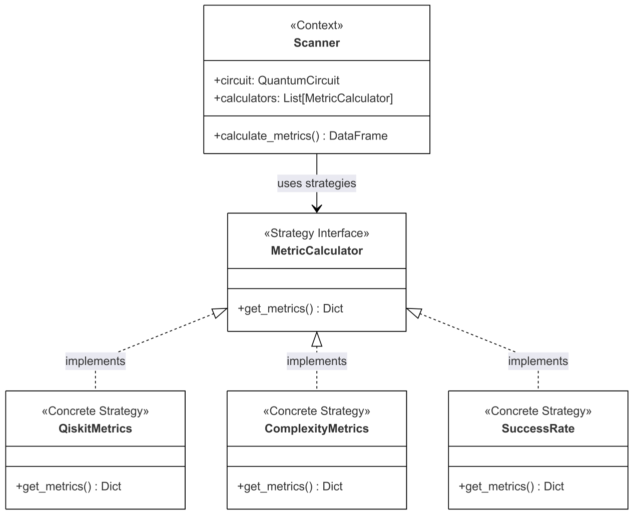

Fig. 1. High-level design of the calculator feature.

As illustrated in Figure 1, the **MetricCalculator** allows users to obtain *Metrics* by selecting different strategies. The initial version of qWard includes three strategies: The **Qiskit-Metrics** III-A, the **ComplexityMetrics** III-B, and the **Circuit-Performance** III-C.

Furthermore, the **Scanner** is responsible for executing the strategies chosen by the user, running the suitable algorithms, and organizing the output in a dictionary that maps the name of the chosen strategy with the resulting dataframe.

B. Visualization feature

Complementing the calculation capabilities, Figure 2 illustrates a high-level schematic of the qWard visualization feature. This design seeks to represent how the strategies controlled by the **Visualizer** match registered metrics within the aforementioned Scanner class.

As shown in Figure 2, it is important to consider the constructor of the **Visualizer** class, as it requires a reference to the **Scanner** upon its instantiation. The **Scanner** reference is needed to obtain a list of the calculation strategies selected by the user. In addition, the class incorporates a `ValueError` handling mechanisms that will be activated in scenarios where the user attempts to generate graphs of metrics which have not been previously computed.

Regarding the strategy implementation, within the **VisualizationStrategy**, it is important to notice the `PLOT_REGISTRY` property, which sets the available graphs for each of the registered metrics.

Concerning the output behavior, it is important to highlight the methods starting with the prefix `generate` that allow the **Visualizer** to return either a single Figure, or a Dictionary of Figures using keys that match the registered metrics.

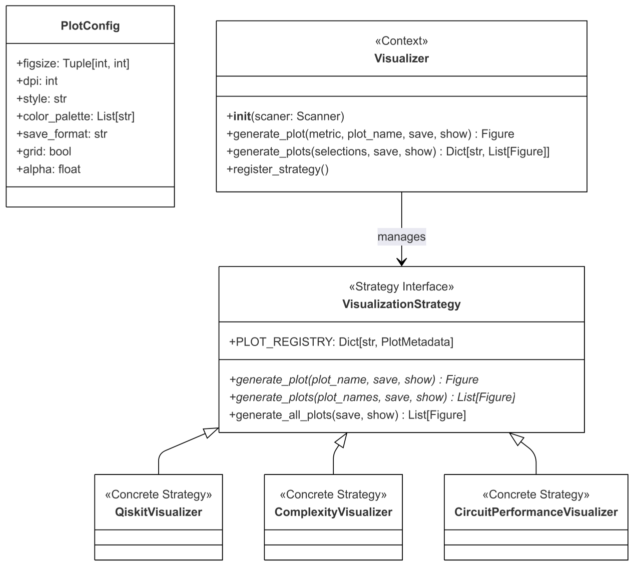
PlotConfig
+figsize: Tuple[int, int]
+dpi: int
+style: str
+color_palette: List[str]
+save_format: str
+grid: bool
+alpha: float
«Context»
Visualizer
+init(scaner: Scanner)
+generate_plot(metric, plot_name, save, show) : Figure
+generate_plots(selections, save, show) : Dict[str, List[Figure]]
+register_strategy()
manages
«Strategy Interface»
VisualizationStrategy
+PLOT_REGISTRY: Dict[str, PlotMetadata]
+generate_plot(plot_name, save, show) : Figure
+generate_plots(plot_names, save, show) : List[Figure]
+generate_all_plots(save, show) : List[Figure]
«Concrete Strategy»
QiskitVisualizer
«Concrete Strategy»
ComplexityVisualizer
«Concrete Strategy»
CircuitPerformanceVisualizer

Fig. 2. High-level design of the visualization feature.

V. QWARD IN ACTION

This section demonstrates how qWard helps quantum developers answer questions about their circuits. In particular, this example attempts to answer the following motivating questions: 1) *"How often did I get the results I wanted?" (Success rate)*; and 2) *"How close are my overall results to the ideal quantum behavior (Fidelity)?"*. The tutorial runs the analysis on a 2-qubit entangled state circuit using both the Qiskit AER simulator and the IBM quantum runtime service.

It is important to note that, given the space constraints, only visualizations pertinent to post-runtime metrics have been incorporated into the paper.To ensure reproducibility and further investigation, a Jupyter notebook addressing the example is available at [36].

A. Analyzing Circuit Performance

Imagine a quantum algorithm designed to entangle two qubits. Following the construction of the circuit, the developer seeks to analyze its performance characteristics. According to theoretical predictions, the expected output must either be $|00\rangle$ or $|11\rangle$, each occurring with equal probability. However, acknowledging that execution on a QPU is costly and faces noise and errors due to technological limitations, the developer chooses to utilize qWard to evaluate the algorithm's behavior before proceeding with execution.

Listing 1 provides the Qiskit code required to represent the Bell state of two qubits. Observe how line 4 denotes the measurement operation on both qubits, which according to the theoretical predictions should be either $|00\rangle$ or $|11\rangle$.

```
circuit = QuantumCircuit(2, 2)
circuit.h(0)
circuit.cx(0, 1)
circuit.measure([0,1], [0,1])
```

Listing 1. 2-qubit entagled qubits circuit

Following the preparation of the circuit, the next task involves configuring the backend environment to execute the algorithm. Subsection V-B shows the code for the simulator setup, while Subsection V-C focuses on the IBM runtime service.

B. *Simulator Analysis*

As mentioned above, the preparation of a backend simulator is straightforward, since it is as simple as instantiating the AerSimulator class provided by Qiskit AER as shown below.

It is imperative to mention that, the simulator was configured with three different backends to represent more realistic setups: 1) a noise-free backend, 2) a back-end incorporating a 5% depolarizing error applicable to all circuit gates along with a readout error of 10%, and 3) a backend introducing a Pauli error to all gates and a readout error of 5%.

```
simulator = AerSimulator()
sim_job = simulator.run(circuit, shots=1024)
```

Listing 2. Qiskit AER simulator noise-free

For illustration purposes, listing 3 shows the configurations for the second backend. Line 1 instantiates the Noise model class. Line 2 establishes a depolarizing error affecting single qubit gates. Line 3 incorporates the error into the NoiseModel and, finally, Line 4 constructs the backend simulator with the specified error.

```
1 noise_model = NoiseModel()
2 depol_error = depolarizing_error(0.05, 1)
3 noise_model.add_all_qubit_quantum_error(
      depol_error, ["u1", "u2", "u3"])
4 depolarizing_errors = AerSimulator(noise_model=
      noise_model)
```

Listing 3. Qiskit AER simulator with noise

C. *IBM Quantum runtime service Analysis*

Alternatively, for execution on actual quantum hardware, the procedure for setting up a backend for the IBM Quantum Runtime service is illustrated in the Listing 4. In line 1, a new instance of the service is created, whereas line 2 queries the least utilized QPU to carry out the circuit execution. It is important to note that it is feasible to configure simulator services or request a specific hardware setup; for further details, please consult the Qiskit Runtime documentation.

Analogous to the approach utilized with the backend simulator, the IBM runtime service encompasses the configuration of a Batch with three distinct Jobs: 1) a backend with no optimization, 2) a backend with an optimization level of 2, employing ”sabre” for both the layout and routing method, and 3) a backend with an optimization level of 3, using ”dense” as the layout method and ”lookahead” as the routing strategy. More details on the preparation of IBM runtime service jobs can be found in [36].

```
1 service = QiskitRuntimeService()
2 backend = service.least_busy(
3     simulator=False,
4     operational=True
5 )
```

Listing 4. IBM runtime service backend

Listing 5 illustrates the configuration of the two jobs that incorporate the optimization levels (lines 12-13).

```
1  pm = generate_preset_pass_manager(
2      optimization_level=1,
3      backend=backend
4  )
5  pm_2 = generate_preset_pass_manager(
6      optimization_level=2,
7      backend=backend
8  )
9
10 with Batch(backend=backend) as batch:
11     sampler = SamplerV2()
12     isa_circuit_1 = pm.run(circuit)
13     isa_circuit_2 = pm_2.run(circuit)
14     job_1 = sampler.run([isa_circuit])
15     job_2 = sampler.run([isa_circuit_2])
```

Listing 5. Running a batch on the IBM runtime service

With the completion of the circuit and backend, all necessary components are now in place to utilize qWard. The following section provides a summary of the process for integrating the circuit and the results within qWard.

D. *Integrating qWard*

Listing 6 outlines the procedure for creating the qWard **Scanner**. In this example, lines 2 pass down the reference of the circuit, lines 3 provide the reference of the job, and lines 4 specifie the requested strategies pertinent to the current analysis. It should be noted that, in line 7, the post-runtime *CircuitPerformanceMetrics* strategy requires the circuit and the jobs as context for comprehensive analysis. Furthermore, lines 10 and 11 define particular functions whose implementations are described in the listing 7. Finally, line 16 runs the calculate_metrics method which returns the requested metrics, thus launching the metric calculation process.

```
1  scanner = Scanner(
2      circuit=circuit,
3      job=sim_job,
4      strategies=[
5          QiskitMetrics,
6          ComplexityMetrics,
7          CircuitPerformanceMetrics(
8              circuit=circuit,
9              job=jobs,
10             success_criteria=bell_success_criteria
11             expected_distribution=
       expected_distribution
12         )
13     ]
14 )
15
16 metrics_dict = scanner.calculate_metrics()
```

Listing 6. Connecting jobs with qWard

Listing 7 allows us to clarify the custom functions referenced in the Scanner configuration. The success function, in lines 1-3, specifies that a measurement is successful if it yields either the result ”00” or ”11”. In the same way, the expected distribution function delineated in lines 5-6 represents the theoretical fidelity of a Bell state.

```
1 def bell_success_criteria(bitstring):
2     clean_result = bitstring.replace(" ", "")
3     return clean_result in ["00", "11"]
4
5 def expected_distribution():
6     return {"00": 0.5, "11": 0.5}
```

Listing 7. Success criteria and expected distribution functions

E. *Visualizing Performance Comparison*

Having calculated the metrics, the focus shifts to addressing the initial developer questions through graphs. In this context, the Visualizer class of qWard becomes relevant.

Listing 8 illustrates the necessary code for the initialization of the Visualizer. It is important to observe that, in line 2, a reference to the scanner is passed.

```
1 visualizer = Visualizer(
2     scanner=scanner,
3     output_dir="img/my_research"
4 )
```

Listing 8. Creating success rate comparison visualization

Once the visualizer is initialized, it is necessary to call the generate_plots method offered by the Visualizer class. This method accepts the `selections` parameter, which allows the user to configure the desired plots. Listing 9 demonstrates how to obtain the success error and fidelity plots, which facilitate the discussion of the questions formulated at the beginning of this section.

```
visualizer.generate_plots(
    selections={
        Metrics.CIRCUIT_PERFORMANCE: [
            Plots.CircuitPerformance.
    SUCCESS_ERROR_COMPARISON,
            Plots.CircuitPerformance.
    FIDELITY_COMPARISON
        ]
    }
)
```

Listing 9. CircuitPerformance graphs

Figures 3 and 4 show the resulting graphs. In each of the figures, the X-axis denotes the executed jobs, while the Y-axis represents the associated success rate. Note that Job 1 of figure 3 (the noiseless scenario run on the simulator) compared to the 3 jobs of figure 4 (run on physical hardware) show a performance divergence between the idealized projections as expected. Instead, Jobs 2 and 3, where the models accounting for noise become relevant, demonstrate outcomes comparable to those observed with quantum hardware.

Figure 5 presents the fidelity values associated with IBM execution. It is apparent from these data that the default configuration of the backend does not optimally suit the circuit. A layout method value of *dense* and a routing strategy of *lookahead* lead to enhanced fidelity (Job 3 in the chart). Moreover, the optimization level does not appear to enhance the outcome; actually, a fidelity reduction is observed with an optimization level of 2 compared to zero optimization.

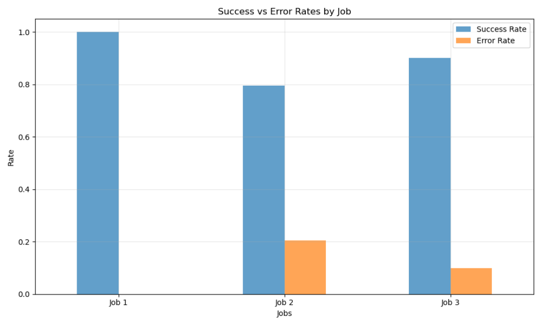

Fig. 3. Success vs. error rates for entangled qubits on AER simulator

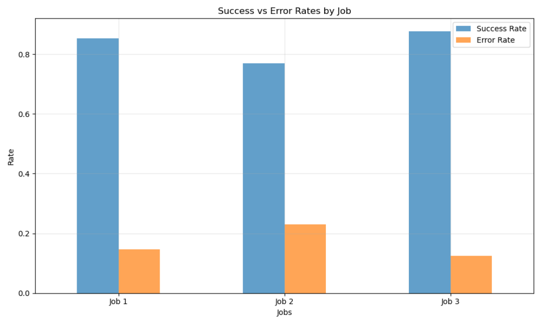

Fig. 4. Success vs. error rates for entangled qubits on IBM quantum hardware

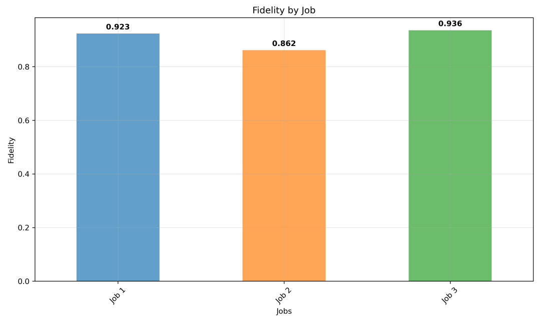

Fig. 5. Fidelity on IBM quantum hardware.

Based on the resulting graphs and calculated metrics, qWard provides concrete answers to the initial developer questions. Regarding *”How often did I get the results I wanted?”*, the hardware achieves a success rate of 0.833 ± 0.46 compared to 0.899 ± 0.84 in the simulator. For *”How close are my overall results to the ideal quantum behavior?”*, the average fidelity ranges from 0.862 to 0.936 on hardware (as shown in figure 5) versus near-perfect fidelity on the simulator, indicating a reasonably close approximation to ideal quantum behavior despite hardware noise.

In summary, this tutorial allows us to show that qWard works by converting raw quantum execution data into actionable insights, thereby aiding developers in comprehending and enhancing the real-world performance of their quantum algorithms.

VI. Discussion

In this section we recapitulate how our approach address the posed research questions. Related to RQ1, our research indicates that current QC SDKs offer insufficient support for detailed circuit analysis. In particular, after experimenting with Qiskit in the initial version of qWard, we notice that the library provides fundamental circuit characteristics such as depth, width, and gate counts, and it also provides basic tools for understanding the execution results like the histogram plot. However, it does not implement the advanced metrics that are extensively covered in the academic literature and lacks robust tools for execution analysis.

With reference to RQ2, it is evident that existing SDKs do not integrate emerging metrics from recent publications. The disparity between theoretical advances and practical implementation forces researchers to code the same analysis repeatedly, resulting in inconsistent definitions of metrics and varying calculation methodologies.

Finally, to answer RQ3, qWard shows that a metrics library can be effectively designed by implementing the strategy pattern for consistent metrics integration, using Pydantic for schema validation, and including visualization tools to contribute to the analysis. We also conclude that documentation is an essential part of these kinds of library, as it is crucial to inform the user about the related work supporting the metrics included on each strategy.

VII. Conclusions

As QC rapidly progresses while also facing significant challenges such as high error rates, substantial costs, and limited access, comprehensive metric analysis becomes increasingly vital. This paper began by conducting a quick literature review that identified key pre-runtime and post-runtime metrics for evaluating quantum circuits. The review made evident the gap between the properties available in current SDKs (e.g., Qiskit, PennyLane, or Q#) and the metrics reported in the literature. These findings led to the conclusion that a unified tool is needed to calculate a broad spectrum of such metrics.

To bridge this identified gap, the qWard library was introduced. qWard was designed to assist researchers and developers in the systematic collection and analysis of quantum circuit metrics. The purpose of the library is to empower users to make better informed decisions during the quantum software development process, facilitate benchmark preparations, and improve understanding of quantum programs.

Ultimately, qWard seeks to contribute to the maturation of the quantum software ecosystem by promoting a data-driven approach to circuit design and execution analysis. This involves supporting the understanding and visualization of resource utilization and, thereby, improving quantum computations.

A. Future Work

Looking ahead, the development of qWard is an ongoing effort of the TICSw research group of the Universidad de los Andes de Colombia, with several features envisioned to enhance its capabilities, including the following:

Expansion of Metrics Library and Community Contributions: As an open source project, qWard actively seeks community contributions. We envision qWard becoming a collaborative platform for the quantum computing community. Researchers developing novel pre-runtime or post-runtime metrics are invited to integrate their findings into the library's codebase.

Support for Additional Quantum SDKs: While currently focused on Qiskit, a long-term goal is to extend qWard's core functionalities to support other quantum programming SDKs, potentially transforming qWard into an extensible quantum metrics analysis platform.

Advanced Correlation Analysis: Once a comprehensive set of metrics is established, the next step involves analyzing the correlations between pre- and post-runtime metrics. Based on these results, we envision developing models to predict post-runtime metrics from a footprint vector of pre-runtime characteristics.

Visualization and Reporting Enhancements: As part of the beta version, we plan to increase qWard's built-in visualization capabilities and explore integrations with common data analysis and reporting tools.

B. AI Acknowledgments

The authors acknowledge the use of several AI-powered tools that assisted in preparing this manuscript and developing the Alpha version of qWard. Specifically, AI pair programming with Cursor was utilized for coding tasks. Writefull was used for grammar correction and text refinement. Google's Gemini 2.5 Pro and Claude 4 Sonnet provided assistance in connecting textual ideas and generating LaTeX code. Additionally, Elicit.com was used to execute queries and identify relevant academic articles. Although AI tools assisted, all content is owned by the authors, highlighting that all the generated content was directed and verified.

References

[1] D. Maslov, N. Yunseong, and J. Kim, "An outlook for quantum computing," 2019. [Online]. Available: http://www.ieee.org/publications_standards/publications/rights/index.html

[2] D. Willsch, M. Willsch, F. Jin, H. D. Raedt, and K. Michielsen, "Large-scale simulation of shor's quantum factoring algorithm," *Mathematics*, vol. 11, no. 19, 2023. [Online]. Available: https://www.mdpi.com/2227-7390/11/19/4222

[3] L. Upadhyay, "Quantum cryptography: A survey," *Advances in Intelligent Systems and Computing*, vol. 939, pp. 20–35, 2019. [Online]. Available: https://link-springer-com.ezproxy.uniandes.edu.co/chapter/10.1007/978-3-030-16681-6_3

[4] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature Publishing Group*, vol. 549, 2017.

[5] S. Garg and G. Ramakrishnan, "Advances in quantum deep learning: An overview," May 2020. [Online]. Available: https://arxiv.org/abs/2005.04316v1

[6] E. A. Mukesh Madanan, "Exploring quantum computing's potential breakthroughs and challenges," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 11, no. 11, p. 674–679, Dec. 2023.

[7] M. R. El Aoun, H. Li, F. Khomh, and M. Openja, "Understanding quantum software engineering challenges: An empirical study on stack exchange forums and github issues," *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2021. [Online]. Available: https://doi.org/10.1109/ICSME52107.2021.00037

[8] S. Silva, A. Tuyishime, T. Santilli, P. Pelliccione, and L. Iovino, "Quality metrics in software architecture," in *2023 IEEE 20th International Conference on Software Architecture (ICSA)*. Institute of Electrical and Electronics Engineers Inc., Mar. 2023, p. 58–69. [Online]. Available: https://ieeexplore.ieee.org/document/10092705

[9] S. S. Gill, A. Kumar, H. Singh, M. Singh, K. Kaur, M. Usman, and R. Buyya, "Quantum computing: A taxonomy, systematic review and future directions," *Software: Practice and Experience*, vol. 52, pp. 66–114, 2022. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3039

[10] E. Moguel, J. Rojo, D. Valencia, J. Berrocal, J. Garcia-Alonso, and J. M. Murillo, "Quantum service-oriented computing: current landscape and challenges," *Software Quality Journal*, vol. 30, pp. 983–1002, 12 2022. [Online]. Available: https://link.springer.com/article/10.1007/s11219-022-09589-y

[11] J. Alvarado-Valiente, J. Romero-Álvarez, A. Díaz, M. Rodríguez, I. García-Rodríguez, E. Moguel, J. Garcia-Alonso, and J. M. Murillo, "Quantum services generation and deployment process: A quality-oriented approach," *Communications in Computer and Information Science*, vol. 1871 CCIS, pp. 200–214, 2023. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-43703-8_15

[12] A. Díaz, J. Alvarado-Valiente, J. Romero-Álvarez, E. Moguel, J. Garcia-Alonso, M. Rodríguez, I. García-Rodríguez, and J. M. Murillo, "Service engineering for quantum computing: Ensuring high-quality quantum services," *Information and Software Technology*, vol. 179, p. 107643, Mar. 2024.

[13] J. Verduro, M. Rodríguez, and M. Piattini, "Software quality issues in quantum information systems," *Q-SET@QCE*, 2021.

[14] M. Alam, A. Ash-Saki, and S. Ghosh, "Addressing temporal variations in qubit quality metrics for parameterized quantum circuits," *Proceedings of the International Symposium on Low Power Electronics and Design*, vol. 2019-July, 7 2019.

[15] E. Knill, D. Leibfried, R. Reichle, J. Britton, R. B. Blakestad, J. D. Jost, C. Langer, R. Ozeri, S. Seidelin, and D. J. Wineland, "Randomized benchmarking of quantum gates," *Physical Review A*, vol. 77, no. 1, p. 012307, Jan. 2008, arXiv:0707.0963 [quant-ph].

[16] W. v. Dam, M. Mykhailova, and M. Soeken, "Using azure quantum resource estimator for assessing performance of fault tolerant quantum computation," May 2024, arXiv:2311.05801 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2311.05801

[17] M. P. Harrigan, T. Khattar, C. Yuan, A. Peduri, N. Yosri, F. D. Malone, R. Babbush, and N. C. Rubin, "Expressing and analyzing quantum algorithms with qualtran," Sep. 2024, arXiv:2409.04643 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2409.04643

[18] N. Pollard and K. Choudhary, "Benchqc: A benchmarking toolkit for quantum computation," Feb. 2025, arXiv:2502.09595 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2502.09595

[19] M. Suchara, J. Kubiatowicz, A. Faruque, F. T. Chong, C.-Y. Lai, and G. Paz, "Qure: The quantum resource estimator toolbox," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, Oct. 2013, p. 419–426. [Online]. Available: https://ieeexplore.ieee.org/document/6657074

[20] N. Quetschlich, L. Burgholzer, and R. Wille, "Mqt bench: Benchmarking software and design automation tools for quantum computing," *Quantum*, vol. 7, p. 1062, Jul. 2023, arXiv:2204.13719 [quant-ph].

[21] M. Piattini, G. Peterssen, R. Pérez-Castillo, J. L. Hevia, M. Serrano, G. Hernández, I. G. D. Guzmán, C. Paradela, M. Polo, E. Murina, L. Jiménez, J. C. Marqueño, R. Gallego, J. T. i Brugués, F. Phillipson, J. M. Murillo, A. Niño, and M. Rodríguez, "The talavera manifesto for quantum software engineering and programming," *QANSWER*, 2020.

[22] J. A. Cruz-Lemus, M. Rodríguez, R. Barba-Rojas, and M. Piattini, *Quantum Software Quality Metrics*. Cham: Springer Nature Switzerland, 2024, p. 125–142. [Online]. Available: https://doi.org/10.1007/978-3-031-64136-7_6

[23] J. A. Cruz-Lemus, L. A. Marcelo, and M. Piattini, "Towards a set of metrics for quantum circuits understandability," *Communications in Computer and Information Science*, vol. 1439 CCIS, pp. 239–249, 2021. [Online]. Available: https://link-springer-com.ezproxy.uniandes.edu.co/chapter/10.1007/978-3-030-85347-1_18

[24] J. Zhao, "Some size and structure metrics for quantum software," in *2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE)*, Jun. 2021, p. 22–27. [Online]. Available: https://ieeexplore.ieee.org/document/9474569

[25] M. A. Serrano, J. A. Cruz-Lemus, R. Perez-Castillo, and M. Piattini, "Quantum software components and platforms: Overview and quality assessment," *ACM Computing Surveys*, vol. 55, 12 2022.

[26] D. Shami, "Character complexity: A novel measure for quantum circuit analysis," Sep. 2024, arXiv:2408.09641 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2408.09641

[27] K. Bu, R. J. Garcia, A. Jaffe, D. E. Koh, and L. Li, "Complexity of quantum circuits via sensitivity, magic, and coherence," *Communications in Mathematical Physics*, vol. 405, no. 7, p. 161, Jul. 2024, arXiv:2204.12051 [quant-ph].

[28] A. Gheorghiu, T. Kapourniotis, and E. Kashefi, "Verification of quantum computation: An overview of existing approaches," *Theory of Computing Systems*, vol. 63, no. 4, p. 715–808, May 2019, arXiv:1709.06984 [quant-ph].

[29] T. Lubinski, S. Johri, P. Varosy, J. Coleman, L. Zhao, J. Necaise, C. H. Baldwin, K. Mayer, and T. Proctor, "Application-oriented performance benchmarks for quantum computing," *IEEE Transactions on Quantum Engineering*, vol. 4, p. 1–32, 2023, arXiv:2110.03137 [quant-ph].

[30] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, "Validating quantum computers using randomized model circuits," *Physical Review A*, vol. 100, no. 3, p. 032328, Sep. 2019, arXiv:1811.12926 [quant-ph].

[31] R. Harper, S. T. Flammia, and J. J. Wallman, "Efficient learning of quantum noise," *Nature Physics*, vol. 16, no. 12, p. 1184–1188, Dec. 2020, arXiv:1907.13022 [quant-ph].

[32] R. Bistroń, M. Rudziński, R. Kukulski, and K. Życzkowski, "Benchmarking quantum devices beyond classical capabilities," Feb. 2025, arXiv:2502.02575 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2502.02575

[33] IBM, "Qiskit ecosystem — ibm quantum computing." [Online]. Available: https://www.ibm.com/quantum/ecosystem

[34] ——, "Quantumcircuit (latest version)." [Online]. Available: https://docs.quantum.ibm.com/api/qiskit/docs.quantum.ibm.com/api/qiskit/qiskit.circuit.quantumcircuit

[35] R. Bala and K. K. Kaswan, "Strategy design pattern," *Global journal of computer science and technology*, vol. 14, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:51932930

[36] C. Marquez, "Jupyter notebook with qward in action implementation details." [Online]. Available: https://github.com/xthecapx/qiskit-qward/blob/main/qward/examples/qWard_paper.ipynb