Noname manuscript No.
(will be inserted by the editor)

An Empirical Study Into the Effects of Transpilation on Quantum Circuit Smells

Manuel De Stefano · Dario Di Nucci · Fabio Palomba · Andrea De Lucia

Received: date / Accepted: date

Abstract Quantum computing is a promising field that can solve complex problems beyond traditional computers' capabilities. Developing high-quality quantum software applications, called quantum software engineering, has recently gained attention. However, quantum software development faces challenges related to code quality. A recent study found that many open-source quantum programs are affected by quantum-specific code smells, with *long circuit* being the most common. While the study provided relevant insights into the prevalence of code smells in quantum circuits, it did not explore the potential effect of transpilation, a necessary step for executing quantum computer programs, on the emergence of code smells. Indeed, transpilation might alter those characteristics employed to detect the presence of a smell on a circuit. To address this limitation, we present a new study investigating the impact of transpilation on quantum-specific code smells and how different target gate sets affect the results. We conducted experiments on 17 open-source quantum programs alongside a set of 100 synthetic circuits. We found that transpilation can significantly alter the metrics that are used to detect code smells, even into previously smell-free circuits, with the *long circuit* smell being the most susceptible to transpilation. Furthermore, the choice of the gate set significantly influences the presence and severity of code smells in transpiled circuits, highlighting the need for careful gate set selection to mitigate their impact. These findings have implications for circuit optimization and high-quality quantum software development. Further research is needed to understand the consequences of code smells and their potential impact on quantum computations, considering the characteristics and constraints of different gate sets and hardware platforms.

Keywords Empirical Software Engineering, Quantum Computing, Quantum Software Engineering, Quantum Software Quality, Code Smells

Manuel De Stefano, Dario Di Nucci, Fabio Palomba, Andrea De Lucia
Software Engineering (SeSa) Lab - University of Salerno (Italy)
E-mail: madestefano@unisa.it, ddinucci@unisa.it, fpalomba@unisa.it, adelucia@unisa.it

1 Introduction

Quantum computing has recently gained much attention due to its ability to solve problems that traditional computers cannot [23, 24]. To make this potential available to all, the field of quantum software engineering has emerged [27, 38, 39, 40]. Researchers have been designing and implementing high-quality quantum software applications to exploit the quantum computer's computational speed since the Talavera Manifesto was published [4, 9, 10, 41, 47]. However, the field is still in its early stages, and more research is needed to address the maintenance, evolution, and overall quality aspects of quantum software development [16, 17, 19].

A recent study revealed that quantum software development faces challenges similar to traditional software development, such as code smells affecting the program's quality [14]. This study investigated the prevalence of these code smells in 15 open-source quantum programs using QSmell, which detected eight quantum-specific smells. The results showed that 73.33% of the programs contained at least one smell, the most common being the Long Circuit smell.

However, the study did not investigate the effects of transpilation on the presence of these code smells, which might be a limitation. Indeed, transpiling a quantum circuit involves converting a high-level quantum program into a form suitable for specific quantum hardware. It optimizes the circuit by decomposing complex gates, rearranging qubits to match hardware connectivity, minimizing gate count, and managing available resources. The process ensures the circuit retains functionality while maximizing performance and compatibility with the target hardware platform. Hence, it is a necessary step to execute the program on a quantum computer. However, it can alter the circuit's characteristics, such as the width, the depth, and the gates. Characteristics that QSmell uses to detect quantum code smells.

To address this limitation, we present a new study investigating whether the quantum-specific code smells detected in the original circuit persisted after transpilation. We also investigated whether different target gate sets affect the results. The experiment was conducted on the same set of circuits employed by the original study [14].

Our research suggests that transpilation can significantly alter the metrics indicating the presence of smells even in circuits that previously showed non-troublesome metrics values. The *long circuit* and *initialization of quibits* smell are particularly vulnerable to transpilation. The gate set plays a significant role in detecting code smells in transpiled circuits. Therefore, it is important to envision and develop detection techniques considering the target gate set on which the circuit will be transpiled. Further research is necessary to fully understand the consequences of code smells and their potential impact on quantum computations, considering the specific characteristics and limitations of different gate sets and hardware platforms.

The remainder of the paper is organized as follows. Section 2 presents all the necessary background information and the relevant related work. Section 3 describes the experimental procedure in all stages, and its limitations. Section 4 describes the achieved results, while section 5 discusses them. Section 6 wraps up and proposes future research directions.

2 Background and Related Work

2.1 Quantum Computing and Transpilation

Quantum computing utilizes principles from quantum mechanics to perform computations [23, 24]. Unlike classical computing, which uses bits with values of zero or one, quantum computing uses qubits that can exist in superpositions of the zero and one states. Quantum information is manipulated using quantum gates, which are unitary transformations. Quantum programming languages store classical and quantum information in registers and express quantum programs as quantum circuits, where gates are applied in a specific order. Quantum circuits can be executed on real quantum devices or simulators, and the output is measured and stored in classical registers. A fault occurs if the observed probability distribution from running the circuit multiple times does not match the expected distribution.

Transpiling a quantum circuit refers to transforming a quantum circuit written in one quantum programming language or representation into another, often to optimize the circuit for a specific quantum hardware platform. Quantum circuits are typically expressed using a high-level quantum programming language, such as Qiskit, Cirq, or Quil. These languages provide a convenient way for researchers and developers to describe quantum algorithms and operations. However, quantum hardware platforms often have specific requirements and constraints, such as limitations on gate connectivity, gate set availability, and gate execution times. Transpiling bridges the gap between the high-level quantum programming language and the specific hardware platform. It involves converting the circuit into a form compatible with the target hardware while preserving the functionality of the original circuit. The transpiler performs a series of optimizations and transformations to achieve this goal [3]:

Gate Synthesis. Converting gates or gate sequences into an equivalent set of gates supported by the target hardware. This operation may involve decomposing complex gates into a combination of simpler gates from the hardware's gate set.

Gate Mapping. Rearranging the qubits in the circuit to match the connectivity constraints of the target hardware. Different hardware platforms have different qubit connectivity layouts, and the transpiler aims to find an optimal mapping that minimizes the number of required additional gates (e.g., SWAP gates) for connecting qubits that are not directly connected.

Gate Optimization. Applying techniques to minimize the overall number of gates in the circuit, reduce gate depth, or optimize other metrics. This optimization can improve the circuit's performance by reducing the potential for errors and minimizing the time required for execution.

Resource Allocation. Ensuring that the circuit does not exceed the resources available on the target hardware, e.g., the number of qubits.

By transpiling a quantum circuit, developers can make their quantum algorithms compatible with specific quantum hardware, improve the circuit's performance, and take advantage of different platforms' unique features and constraints.

In quantum computing, a basis gate set is a specific set of quantum gates that are the building blocks for creating and describing quantum circuits [46]. These gates are carefully chosen for basic quantum operations such as rotation and phase shift [46]. A set of universal quantum gates refers to any set of gates that can be used to build any quantum operation possible on a quantum computer. Any other unitary operation can be expressed as a finite sequence of gates from the set [46]. However, since the number of possible quantum gates is uncountable, whereas the number of finite sequences from a finite set is countable, it is technically impossible to have anything less than an uncountable set of gates [46]. To solve this problem, it is only required that any quantum operation can be approximated by a sequence of gates from this finite set [46]. One universal gate set commonly used in quantum computing includes the rotation operators $R_x(\theta)$, $R_y(\theta)$, $R_z(\theta)$, the phase shift gate $P(\phi)$, and $CNOT$. When a circuit is transpiled, it is transpiled in the basis gate set that is supported by the target machine [2].

```python
from qiskit import QuantumCircuit

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

Listing 1 Python example

Listing 1 represents a simple quantum circuit designed to create the renowned Bell state [29]. Visualizing this circuit without any transpilation, it appears as illustrated in Figure 1. This circuit is relatively straightforward, employing only Hadamard and C-NOT gates. However, when we transpile this uncomplicated circuit into the basis gate set supported by the IBM quantum machine 'ibm_perth,' the circuit undergoes significant transformations, as depicted in Figure 2. Notably, two RZ gates and an SX gate (represented as $\sqrt{X}$ in the figure) are required to achieve the same quantum effect as a Hadamard gate.

2.2 Quantum Software Engineering

Research in quantum software engineering (QSE) is still in its early stages. The "Talavera Manifesto" was proposed during the first International Workshop on Quantum Software Engineering, outlining the fundamental principles of QSE [39]. Subsequent studies have discussed various challenges and directions

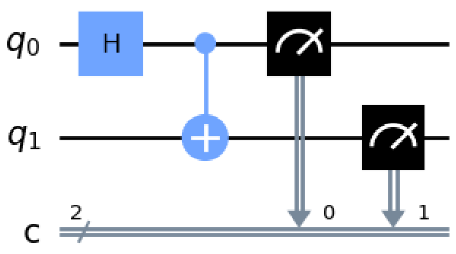

Fig. 1 Quantum circuit representing the Bell state before transpilation.

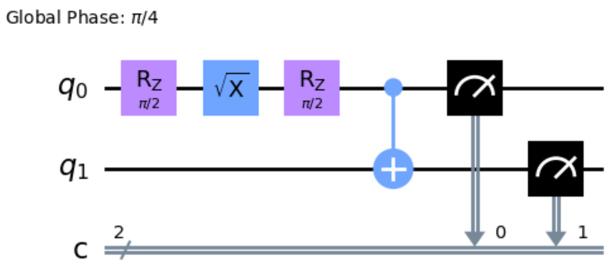

Fig. 2 Quantum circuit representing the Bell state transpiled for the 'ibm_perth' gate set.

in QSE research. Zhao *et al.* [48] provided a comprehensive overview of the quantum software life cycle, including requirements analysis, design, implementation, and testing. They highlighted the need for a comprehensive software engineering discipline for quantum software development.

Piattini *et al.* [40] emphasized four priority areas: software design of quantum hybrid systems, testing techniques for quantum programming, quantum program quality, and re-engineering and modernization of classical-quantum information systems. They suggested that researchers should consider the lack of software engineering knowledge among quantum computer scientists and should not wait for stable quantum programming languages to develop software engineering techniques.

Developing quantum programs presents various challenges that must be overcome for effective design and development. El Aoun *et al.* [19] conducted an empirical investigation to understand these challenges from a developer's perspective. They analyzed popular forums and platforms like Stack Exchange and GitHub to identify frequently asked questions and concerns about quantum software engineering. Their findings revealed that developers often inquire about the theory behind quantum programming, specific data structures and algorithms, the implementation of quantum-related tasks, and the lack of learning resources. These challenges highlight the need for a better understanding of quantum theory and the development of appropriate techniques and tools for quantum programming.

De Stefano *et al.* [16] emphasized the importance of systematic investigations into the state of quantum programming. They conducted a mining analysis of popular quantum programming frameworks on GitHub and surveyed the contributors of these repositories. The study revealed that the adoption of quantum programming still needs to be improved, and various challenges must be addressed. These challenges extend beyond technical concerns and encompass socio-technical matters as well. The research community must pay attention to these challenges and work towards advancing the field of quantum software engineering. Researchers and practitioners can facilitate knowledge transfer and contribute to the growth and development of quantum programming and its associated disciplines by conducting systematic investigations and addressing the identified challenges.

Afterward, they conducted a systematic mapping study [17] of QSE research to identify the most investigated topics and the types of studies conducted. They found that most research has primarily focused on software testing. In contrast, other areas, such as software engineering management or quantum software maintenance and quality, have received less attention. This lack of research in quantum software maintenance and quality indicates a gap in understanding and addressing the challenges associated with these aspects. It emphasizes the need for further investigations to develop effective methods and techniques for maintaining and ensuring the quality of quantum software.

2.3 Code Smells

Developing and maintaining software code is time-consuming and puts developers under constant pressure. This pressure makes developers prioritize tasks to release software quickly, sometimes sacrificing quality, introducing technical debt [15]. One common form of technical debt is code smells, i.e., developers' poor design or implementation choices during software development and maintenance [20].

In recent years, researchers have extensively studied code smells, examining their causes, evolution, and impact on software [7, 13, 30, 43, 44]. They have also investigated methods to detect these design issues automatically. Many of these techniques rely on heuristic approaches based on structural code metrics, such as size and complexity, while others use textual content or version history [28, 31, 32, 42]. However, these heuristic techniques have limitations, leading to suboptimal results.

Therefore, researchers have explored machine-learning-based approaches for code smell detection to address the limitations of heuristic techniques. While initial results appeared promising [6], these techniques also have practical limitations [18, 34, 35, 36, 37]. One major challenge across both heuristic and machine-learning-based methods is the choice of metrics, as existing metrics

Table 1 Original catalog of smells provided by Chen *et al.* [14] alongside the violated best practice, acronym, and description. CG, LPQ, and IM were never found to occur.

Best Practice	Smell Name	Acronym	Description
Getting a circuit to run on hardware	Use of Customized Gates	CG	Any customized gate is decomposable into built-in framework operators. This decomposition requires a substantially higher number of operators when compared to the equivalent solution made exclusively of built-in operators.
Using Circuit-Operation to reduce circuit size	Repeated set of Operations on Circuit	ROC	Due to technological and physical limitations, the number of operations one can pass to a quantum computer is limited; therefore, the circuit implementing the whole algorithm should be prepared in such a way that the number of a sequential repeated set of operations to be performed is the least possible.
Use sweeps when possoble	Non-parametrized Circuit	NC	Real devices work in a shared policy. The circuit should be designed parametrically to provide the different initial values simultaneously, avoid queuing different ones, and reduce communication payloads.
Short Gate Depth	Long Circuit	LC	Unitary gates and measurements are prone to errors (especially due to quantum noise). The higher the depth of the circuit and/or wider the circuit, the higher the probability of affecting a quantum circuit's intended behavior.
Terminal Measurements	Intermediate Measurement	IM	Measurements affect the state of the entire system, making it prone to more errors. They should be postponed to the last operation on the circuit to avoid error propagation.
Keep Qubits Busy	Idle Qubits	IdQ	With current technology, it is possible to ensure the correctness of a state only for very short periods. Idle qubits for too long increase the loss of quantum information that may jeopardize the results of a quantum circuit.
Delay Initialization of Qubits	Initialization of Qubits	IQ	Keeping the coherence of a quantum excited state is technologically difficult. Hence, initially, one should keep it in its ground state (i.e., in state $\|0\rangle$) as long as possible.
Qubit Picking	No-alignment between the Logical and Physical Qubits	LPQ	The topology of real qubits impacts the circuit behavior, i.e., the results obtained from the circuit can change according to the physical qubits configuration. Not aligning the logical qubits to the proper physical qubits may lead to less accurate results.

often have limited explanatory power in distinguishing between smelly and non-smelly code instances.

As a result, code smell detection remains an open challenge, and researchers are beginning to explore alternative approaches, including deep learning [25, 26], to find more effective solutions to this problem.

2.4 Quantum-specific Code Smells

Chen *et al.* [14] recently focused on the quality aspects of quantum programs. They conducted an empirical study to answer two research questions:

RQ1 How do practitioners perceive quantum-specific code smells?
RQ2 What is the prevalence of quantum-specific code smells?

To answer RQ1, eight quantum-specific smells were identified based on best practices in QC, and developers who have contributed to quantum open-source projects were surveyed to assess their opinions on these smells. The results achieved by answering this question were collected in a catalog of quantum-specific code smells, which is summarized in Table 1.

To answer RQ2, the authors developed a tool called QSmell, which automatically computes metrics indicating the presence of quantum-specific smells (depicted in Table 1) in quantum programs (QPs) through dynamic and static analysis. They defined specific thresholds that can be applied to these metrics to conduct this task. When conducting a dynamic analysis of a QP, QSmell requires an execution matrix as input. Each row of this matrix corresponds to either a quantum or classical bit, while each column represents a timestamp in the circuit. Each cell of the matrix indicates a quantum operation that took place in the circuit. To begin, the module collects the set of qubits from the qc object's data and then proceeds to iterate through all the operations performed on each qubit. In summary, this process involves analyzing the quantum operations that occur in a circuit by examining the execution matrix for the QP. This dynamic analysis procedure computes all smells except NC and LPQ. NC and LPQ smells are computed by static analysis. It takes a source code Python files and analyzes it using Python AST to detect the smells that are concerned with information about the execution backend.

The tool was then used to evaluate 15 QPs (circuits) empirically. They showed that 11 QPs (73.33%) contain at least one smell and, on average, a program has three smells. Furthermore, the long circuit (LC) is the most prevalent smell present in 53.33% of the subjects. Interestingly, CG, LPQ, and IM were never found to occur. The study's main limitation was that it did not consider the transpilation of quantum programs that might alter the properties of the circuit used by QSmell to detect the quantum-specific smells.

Although the study provided valuable insights, it also had limitations, including the transpilation process. The study excluded smells that required direct source code evaluation. All source code was transpiled into a target gate set before being analyzed. This choice was applied to all smells except for CG. As a result, the identification of smells was conducted solely on the transpiled code, not the original code. The code analyzed for the CG smell was not transpiled because this step would have removed custom gates. These limitations make it difficult to determine to what extent the transpilation process impacted the persistence of the smells and whether the circuits were considered smelly solely because they were evaluated with a particular gate set. Our study aims to overcome this limitation by investigating the effect of transpilation on the presence of code smells from two points of view. On the one hand, we evaluate the smelliness pre and post-transpilation. On the other hand, we evaluate the smelliness among multiple versions of the same circuit transpiled to different gate sets.

3 Experimental Design

Following the Goal-Question-Metric (GQM) approach delineated by Basili *et al.* [11], we define the objective of our study as:

Characterize the influence of transpilation on quantum-specific code smells **for the purpose of** assessing **with respect to** its potential in mitigating or exacerbating the presence of these smells **from the point of view of** both researchers and practitioners **in the context of** quantum software development process.

Given our goal, we put forth the following **Research Questions** (RQs):

RQ$_1$. How does transpilation affect the presence of quantum-specific code smells?
RQ$_2$. How does transpiling to different gate sets affect the presence of quantum-specific code smells?

Figure 3 and Figure 4 overview the research method applied to address these questions. As further elaborated in the remainder of the section, we applied a quasi-experimental procedure on 17 quantum circuits, taken from literature [14], affected by quantum-specific smells. We detected the smells before and after transpilation. We also compared the smells of circuits transpiled to different target gate sets.

3.1 Context of The Study

Independent Variable. The independent variable to answer both research questions was the target gate set used for transpiling quantum circuits. The target gate set represents the set of gates the circuit is transpiled to, affecting the resulting properties and behavior of the circuit. Specifically, the study considered five target gate sets, whose details are reported in Table 2, that differ in the types of gates they contain. The transpilation process is consistent across all the gate sets, but changing the target gate set also changes the final circuit; therefore, the same circuit transpiled to different gate sets may produce

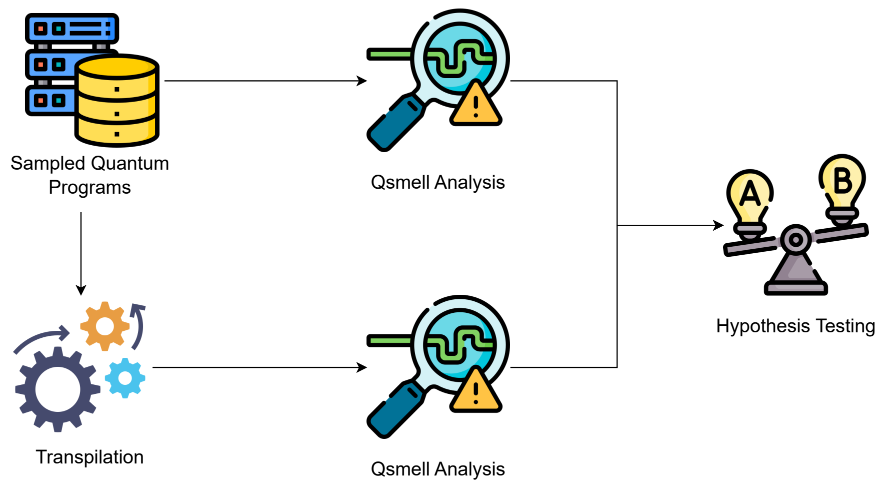

Fig. 3 Graphical representation of the experimental process for the RQ1. To answer the first research question, we ran QSmell on the transpiled and non-transpiled versions for each sample circuit to gather metrics and smells; then, we tested for the difference.

different results. Besides the `original` and `none` transpilations (representing the target gate set from the original publication [14] and the untranspiled circuit, respectively), we also considered four distinct gate sets: `ibm_perth`, `ibm_sherbrokee`, `rpcx`, and `simple`. The rationale behind this selection stems from both practicality and the need to capture diverse gate set representations. The gate sets `ibm_perth` and `ibm_sherbrokee` were chosen due to their real-world significance. The IBM quantum provider, the original developer and distributor of Qiskit [2], supports both. Given our focus on Qiskit code execution, these gate sets were the most relevant. However, different quantum machines can share the same basis gate set. Specifically, the `ibm_perth` gate set, characterized by the basis gates CX, ID, RZ, SX, and X, is not supported only by the IBM machine 'ibm_perth', but also by other, such as 'ibm_nairobi'. The same applies to the `ibm_sherbrokee` gate set, which is not supported only by the 'ibm_sherbrokee' machine but also by other machines. This observation leads us to conclude that IBM machines predominantly support this configuration, making our study directly applicable to real-world scenarios. The `rpcx` and `simple` gate sets were chosen to capture the essence of universal gate sets [46] but with a difference in granularity. The `rpcx` gate set is comprehensive, comprising the CX, RX, RY, RZ, and P gates. This ensemble, including the C-Not gate, phase gate, and rotation gates across all axes, encapsulates a universal set and is well-acknowledged in quantum literature [46]. Contrarily, the `simple` gate set is a more streamlined representation. It mimics the `rpcx` set but utilizes only the CX and U3 gates. The U3 gate, specifically offered by IBM, is a versatile, parametrized gate capable of rotations across all three axes and phase alterations based on given parameters. It can emulate the functionalities of RX, RY, RZ, and P gates by fixing two parameters. However,

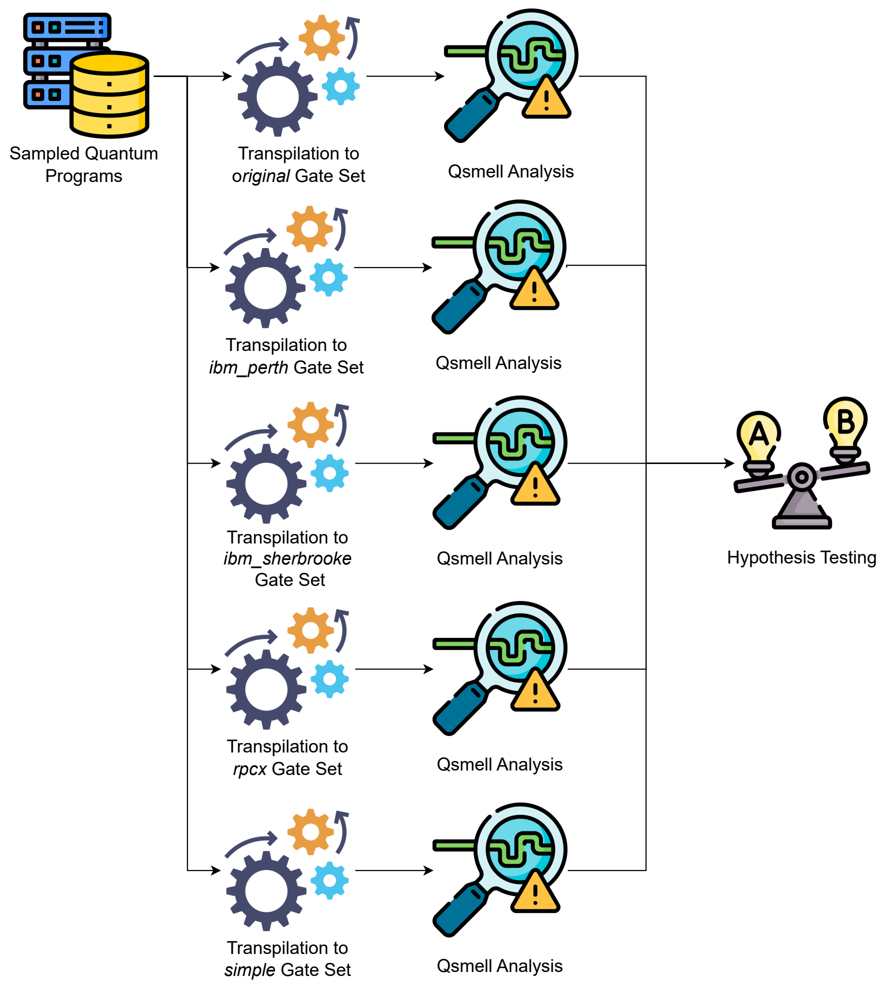
Sampled Quantum Programs
Transpilation to *original* Gate Set
Transpilation to *ibm_perth* Gate Set
Transpilation to *ibm_sherbrooke* Gate Set
Transpilation to *rpcx* Gate Set
Transpilation to *simple* Gate Set
Qsmell Analysis
Qsmell Analysis
Qsmell Analysis
Qsmell Analysis
Qsmell Analysis
Hypothesis Testing

Fig. 4 Graphical representation of the experimental process for the RQ2. To answer the second research question, we transpiled each sample circuit to the chosen gate set; then, we ran QSmell on each version to gather metrics and smells and tested for differences. If a significant difference was observed, we conducted a posthoc analysis.

this simplicity means the transpiler has a different set of gates to work with during the transpilation phase, offering alternate (but not necessarily superior or inferior) transpilation strategies. In essence, these two gate sets, while both being universal, present varied gate counts, further enriching our study. To answer our first research question, we considered only `None` and `original` transpilation to measure the before/after transpilation effect, while `original`, `ibm_perth`, `ibm_sherbrokee`, `rpcx`, and `simple` gates sets were employed to answer our second research question.

Table 2 The different gate sets and the raw one used in the study and their corresponding descriptions. The gate sets include the original gate set used in the literature on quantum-specific smells, two gate sets employed by IBM quantum machines (ibm_perth and ibm_sherbrooke), and two known universal gate sets (rpcx and simple).

Name	Gates	Description
`original`	U1, U2, U3, RZ, SX, X, CX, ID	Gate set employed in the original literature on quantum smells [14].
`ibm_perth`	CX, ID, RZ, SX, X	Gate set employed by the IBM quantum machine `ibm_perth` [2].
`ibm_sherbrokee`	ECR, ID, RZ, SX, X	Gate set employed by the IBM quantum machine `ibm_sherbrooke` [2].
`rpcx`	CX, RX, RY, RZ, P	Known universal gate set composed of the controlled-not gate, all axes rotation gates, and phase gate [46].
`simple`	CX, U3	Similar to `rpcx` but relying on the U3 parametrized rotation gate, which can perform rotation on all the axes and in the phase [2, 46].
`none`	-	The raw circuit defined in the source code, without transpilation [14].

Dependent Variables. The dependent variables considered are the metrics used to detect quantum-specific smells, as described in literature [14]. The details of these metrics are depicted in Table 3. We selected only these from the original set of smells since NC and LPQ were computed with static analysis on the original code and not on the execution matrix generated by QSmell (see Section 2 for further details about these analyses) [14], so they were not affected by transpilation. CG and IM were discarded from the study since they were never detected, both in the original and our study. Hence, we conducted a dedicated discussion in Section 5.

Hypotheses. After defining the independent and dependent variables for both research questions, we formulated the following sets of null (H_0) and alternative (H_1) hypotheses to be tested. Given

$$M = \{LC, IQ, IdQ, ROC\}$$

the sets of considered metrics (M) (as depicted in Table 3), we formulated the following set of hypotheses to answer RQ1:

$H_0^{(1)}(m)$: *Transpilation has no impact on the metric* $m \in M$
$H_1^{(1)}(m)$: *Transpilation impacts on the metric* $m \in M$

Similarly, we formulated the following hypotheses to answer RQ2:

$H_0^{(2)}(m)$: *Transpiling to different gates set has no impact on the metric* $m \in M$
$H_1^{(2)}(m)$: *Transpiling to different gates set impacts on the metric* $m \in M$

Table 3 Dependent variables, metric descriptions, and smelliness thresholds, as depicted by Chen *et al.* [14]. It must be noted that the smell and the metric share the same name as reported in the original publication [14].

Acronym	Name	Metric Description	Smelliness Threshold
LC	Long Circuit	Likelihood of a circuit not having any error as $(1-e)^{l \cdot c}$, whereas e is the maximum error of any active gate of the real device that is used to run the circuit, and l is the maximum number of operations in any qubit and c is the maximum number of parallel operations in the circuit.	Lower than 0.5
IQ	Initialization of Qubits	Maximum number of operations performed in the circuit between the initialization of any qubit (usually the first operation applied to the qubit) and the second operation applied to the same qubit.	Median value on original distribution: Higher than 0
IdQ	Idle Qubits	Maximum number of circuit operations between one operation using a qubit and the subsequent operation where that qubit is used again.	Median value on original distribution: Higher than 0
ROC	Repeated Set of Operations on Circuits	Number of sequentially repeated sets of operations.	Higher than 1

Population and Sample. The interested population of this study comprises all the possible existing quantum circuits. However, for the sake of fair comparison in this study, we choose a sample of 17 quantum circuits, written in QISKIT, that were already objects of investigation in a previous study on quantum-specific smells [14]. In particular, the selection process involved several steps, as described in the original publication [14] in which three umbrella projects were selected (i.e., qiskit-machinelearning, qiskit-terra, and qiskit-nature) containing multiple Quantum Programs (QPs). From these umbrella projects, 17 programs representing the sample were chosen. Two of these circuits were employed by the original publication to validate the tool and, hence, were excluded from the empirical study whose results are reported in Table 5 of the original paper [14]. For the sake of the sample's significance, we chose to employ all subjects available, hence conducting our experiment on all 17 samples. However, we were aware of the limitation of the sample size in our study. Nevertheless, we faced the same issue as the original publication from which we took the samples for comparison. Chen *et al.* [14] had already conducted extensive research to identify possible subjects from open-source projects that were part of real-world projects, not toy or learning ones. Despite this, we tried to find other possible samples to increase the generalizability of our results. To begin with, we looked for samples from a replication package of our previous work [16]. This work included a collection of quantum-related repositories classified as *Libraries/Frameworks*, which could be considered nontrivial and contain possible samples. However, despite working with quantum libraries like Qiskit, we could not find an explicit use of quantum circuits that could be integrated into our sample. We also referred to other published work to find a dataset of quantum circuits for analysis. Specifically, we examined two studies: the one conducted by Paltenghi *et al.* [33] and the one conducted by Campos *et al.* [12]. While the former is a collection of bugs minimized on change records, the latter proposes creating a dataset of reproducible quantum bugs unavailable at the time of writing. As such, neither study helped provide us with a dataset to use. Therefore, we used the Qiskit functionality to generate synthetic circuits (i.e., random circuits) [1] to conduct the same analysis on a broader set of circuits. Using this functionality, we created a new sample of 100 circuits with a width and depth ranging from the minimum to the maximum width and depth of the original samples. We applied the same analysis process that we employed for the original circuits. However, we must emphasize that these synthetic circuits are made by randomly putting gates one on top of another without additional criteria, i.e., such circuits could lack semantics to solve real problems.

3.2 Data Collection and Analysis

We used the original code for the sample circuits from Cheng *et al.*. [14]: the selection of these circuits was based on our willingness to replicate the original study, which led us to rely on their same dataset [14]. We transpiled each circuit using Qiskit's transpiler with the selected target gate set. Then, we measured the smells' metrics using the QSmell tool [14]. To evaluate our pre- and post-transpilation results and address RQ$_1$, we ran QSmell on the raw circuit (the `none` gate set in Table 2) and on the circuits transpiled to the `original` gate set (i.e., in the form that evaluated by Chen *et al.*).

We then used the Wilcoxon Signed Rank Test [45] to test our first set of hypotheses. We needed a paired statistical test since we evaluated a pre/post-treatment scenario where the transpilation corresponds to the treatment that our subjects (the circuits) underwent. Furthermore, we needed a non-parametric statistical test because our data did not follow a normal distribution, confirmed by the Shapiro-Wilk test. Hence, our choice fell on the Wilcoxon Signed Rank Test, a paired non-parametric statistical test [45]. In particular, we applied this test to the metrics distributions of the `none` and `original` treatments. To determine the difference between the two groups, we used Hedge's g, a method of measuring effect size [22]. This approach is efficient when working with small sample sizes of less than 20. A g value of one indicates a difference of one standard deviation between the groups, a g value of two corresponds to a two standard deviation difference, and so on. We adhere to the following guidelines to interpret Hedges' g values: 0.2 for a small effect, 0.5 for a medium effect, and 0.8 for a large effect [22].

Being in a similar situation as the previous set of hypotheses, i.e., with paired non-parametric data, but with more than one distribution to consider, we choose the Friedman Test [21] alongside posthoc analysis to evaluate our second set of hypotheses, i.e., to evaluate the metrics among the different transpilation targets (Table 2. Since the Friedman test, in the case of a significant result, is able only to tell whether there is *any* difference between *all* the tested distributions [21], we needed a posthoc analysis to understand which were the distribution *actually* differing. We used the pairwise Wilcoxon signed-rank test to perform the posthoc analysis, which implies making multiple comparisons among all the considered distributions. When multiple comparisons are made, the Bonferroni correction is applied by adjusting the significance threshold. This method is essential to maintain the overall reliability of the conclusions by controlling the cumulative probability of Type I errors [8].

In all cases, we set the significance level at $\alpha = 0.05$. To reject the null hypothesis in favor of the alternative, the *p-value* obtained from the tests had to be less than α. This approach was applied to both the original samples dataset and the synthetic samples dataset separately.

3.3 Threats to Validity

In the following, we discuss the threats to the validity of our study.

Threats to Construct Validity. It is essential to ensure that the methods used reflect the intended subject to ensure accuracy in conducting a study, also known as construct validity. For our study, we utilized QSMELL to measure our circuits' smelliness, which could affect our findings' validity. However, it is worth noting that QSMELL has been previously validated, which may alleviate this concern.

Threats to Conclusion Validity. Threats to conclusion validity refer to factors or conditions affecting the researcher's ability to draw accurate conclusions from data analysis. In our case, this is mainly related to non-parametric statistical tests, which have a lower statistical power, i.e., less ability to detect significant effects or differences between groups if they exist. Their usage was the only option since the conditions for applying parametric tests were unmet.

Threats to Internal Validity. Internal validity is a critical factor in research, as it determines the degree to which a study can establish a definitive causal relationship between independent and dependent variables. When working with human subjects, researchers often use a within-subjects experimental design, administering multiple treatments in different orders to mitigate the impact of *learning* or *maturation*. However, our study's subjects are quantum programs, and the treatment is a deterministic algorithm: learning or maturation effects cannot occur, as the output is consistently the same whenever a circuit and specific target gate set are utilized. Additionally, each treatment is applied to the original circuit code each time without impacting the subsequent transpilations.

Threats to External Validity. One possible external validity threat is the limited sample size of only 17 circuits, which were all created using Qiskit, the same quantum technology. However, this decision was crucial to ensure a fair comparison with the previous study [14], which exclusively employed real circuits rather than synthetic or toy circuits. It must also be noted that the original publication [14] already conducted large research of real-world (and not trivial) quantum circuits in open source systems, only finding the reported 17 samples. We conducted another large search of real-world quantum circuits in a dataset of a previous publication [16] (i.e., the repositories identified as libraries/frameworks). Still, we found no additional items to be put in the experimental sample. We tried to overcome such limitation by employing a set of synthetic circuits generated by the Qiskit `random_circuit` utility, which, however, generates circuits by randomly putting gates in sequences, hence creating circuits that are not necessarily similar to the ones created by developers. Nevertheless, it could be possible to replicate our proposed experiments with more real-world circuits when a greater dataset is available. Although we specifically selected circuits previously identified to have quantum-specific issues, our findings might not universally apply to all quantum circuits. Finally, our research only focuses on quantum-specific code smells. Transpiling quantum circuits could impact other quality issues, and in particular, the presence of traditional code smells.

4 Analysis of the Results

In this section, we delve into the findings of our study, structured around the research objectives and hypotheses articulated earlier. As we analyze the collected data, one particular aspect merits immediate clarification. Identifying the ROC smell requires identifying repeating patterns within quantum circuits, which means that its practicality diminishes with increasing circuit depth since the number of patterns to examine increases exponentially. This limitation became evident when considering the `shor` circuit. The circuit possesses significant depth in its `rpcx` and `simple` transpiled versions, rendering the ROC smell method impractical for discerning patterns. Consequently, we excluded the `shor` circuit from our ROC analysis for our second research question. With this context in place, let us proceed with a detailed presentation of our results.

4.1 **RQ**$_1$: Effects of Transpilation on the Smell Presence

Table 4 compares the smelliness metrics before and after transpilation. Most data points have a zero value when analyzing the IQ metric before transpilation, with the first, second, and third quartiles all at this value. There is only one outlier, with a value of one. After transpilation, there is a significant shift in the distribution of the IQ metric. The median value remains unchanged, indicating no central tendency or spread change. However, the maximum value

Table 4 Results of QSmells run on the circuits before and after transpilation

	None				Original			
Subject	IQ	IdQ	LC	ROC	IQ	IdQ	LC	ROC
adapt_vqe	0.00	0.00	0.87	0.00	5.00	19.00	0.00	7.00
ae	0.00	0.00	0.65	0.00	0.00	821.00	0.00	49.00
fae	0.00	0.00	0.61	4.00	0.00	0.00	0.10	4.00
grover	0.00	0.00	0.81	1.00	1.00	2.00	0.22	1.00
hhl	0.00	0.00	0.93	0.00	1.00	1.00	0.75	0.00
iae	0.00	0.00	0.78	4.00	0.00	0.00	0.22	4.00
ipe	0.00	0.00	0.96	0.00	0.00	0.00	0.96	0.00
mlae	0.00	0.00	0.53	15.00	0.00	0.00	0.01	15.00
phase_estimation	0.00	0.00	0.96	0.00	0.00	0.00	0.96	0.00
qaoa	1.00	1.00	0.81	0.00	0.00	0.00	0.81	0.00
qeom	0.00	0.00	0.87	0.00	0.00	19.00	0.01	0.00
qgan	0.00	0.00	0.87	0.00	0.00	0.00	0.75	0.00
qsvc	0.00	0.00	0.93	0.00	0.00	1.00	0.49	1.00
shor	0.00	0.00	0.08	N/A	15,650.00	15,650.00	0.00	N/A
vqc	0.00	0.00	0.87	0.00	0.00	1.00	0.39	2.00
vqd	0.00	0.00	0.93	0.00	0.00	0.00	0.70	0.00
vqe	0.00	0.00	0.93	0.00	0.00	0.00	0.70	0.00
Mean	0.06	0.06	0.79	1.50	921.00	971.41	0.41	5.18
Std	0.24	0.24	0.22	3.84	3,795.50	3,787.77	0.36	12.33
Min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	0.00	0.00	0.78	0.00	0.00	0.00	0.26	0.00
50%	0.00	0.00	0.87	0.00	0.00	0.00	0.39	0.50
75%	0.00	0.00	0.93	0.25	0.00	2.00	0.75	4.00
Max	1.00	1.00	0.96	49.00	15,650.00	15,650.00	0.96	15.00

has significantly increased to 15,650, representing a substantial change in the upper range of IQ values after transpilation. The number of outliers has also increased to three, and the proportion of data falling above the smelliness threshold has changed.

For the IdQ metric before transpilation, the distribution is very similar to the IQ metric, with a median and an IQR of zero. There is only one outlier, which is also above the smelliness threshold. After transpilation, the boxplot displays a median value of zero, indicating no change in central tendency. The IQR remains at 2.0, indicating a narrow spread. However, the maximum value has also increased to 15,650, indicating a significant change in the upper range of IdQ values compared to the previous distribution. The number of outliers and the data falling above the smelliness threshold have also increased.

The table reveals a median value of 0.87 when examining the LC metric before transpilation. Approximately 25% of the data falls below 0.78, and 75% falls below 0.93. The data has a moderate spread, indicated by the interquartile range (IQR) of 0.15. There are only two outliers in the data. After transpilation, the LC metric shows a median value of 0.39, which is slightly lower than the

Table 5 Results of the Wilcoxon signed-rank test for the comparison of the pre- and post-transpilation metrics.

Metric	*p-value*	Hedge's g	Interpretation
LC	**<0.01**	1.20	Large
IQ	0.17	-0.34	Small
IdQ	**0.02**	-0.35	Small
ROC	0.10	-0.39	Small

previous distribution. The interquartile range (IQR) remains similar at 0.74, indicating a consistent spread. Notably, the maximum value remains unchanged at 0.96, indicating that the upper bound of LC values has not changed after transpilation.

Lastly, the ROC metric shows a median zero value, with 25% of the data below 0.25 and 75% below 15.0. The data has a substantial spread, with an IQR of 14.75. There are two outliers with values of 4 and 15 above the smelliness threshold. After transpilation, the ROC metric shows a median value of 0.5, which is a decrease compared to the previous distribution. The IQR shows a value of 4.0, indicating a consistent spread. However, the maximum value has increased to 49, reflecting a notable change in the upper range of ROC values after transpilation.

Table 5 presents the results of the Wilcoxon signed-rank tests used to compare the pre- and post-treatment measures for specific metrics. We conducted the tests under a two-sided alternative hypothesis and reported the *p-values* and effect size for each metric. Significant *p-values* are reported in bold. Based on the findings, the LC metric revealed a considerable gap between the measurements taken before and after the treatment ($p - value = 0.001$), with a remarkably positive effect size ($g = 1.196$), which suggests that the LC values for the first group (pre-treatment) were distinctly higher than those for the second group (post-treatment). The IdQ metric also showed a significant difference ($p-value = 0.020$) but with a small negative difference ($g = -0.354$), indicating that the IdQ values before treatment are lower than the ones after treatment. However, the IQ metric did not show any significant difference ($p-value = 0.170$) despite the small negative effect size ($g = -0.335$). Likewise, the ROC metric did not reveal any significant difference ($p - value = 0.100$), with a slightly lower effect size ($g = 0.393$).

Having discussed the above data, we can reject $H_0^{(1)}(LC)$ and $H_0^{(1)}(IdQ)$ in favor of their respective alternative ones. However, we failed to reject $H_0^{(1)}(IQ)$ and $H_0^{(1)}(ROC)$.

Concerning the synthetic sample, Figure 5 shows the distribution of the smelliness metrics computed on the synthetic samples. What is immediately noticeable is that the values of the two distributions are almost the same, except for some variations in LC and IdQ metrics. Indeed, this is also confirmed by the descriptive statistics.

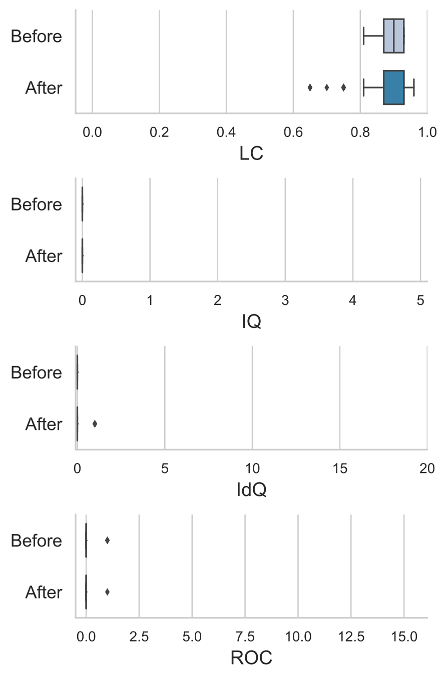

Fig. 5 Boxplot displaying the distribution of smelliness metrics before and after transpilation for the synthetic samples.

Before transpilation, the LC metric mean was 0.8793, with a modest spread around the mean. The majority of circuits had a consistent LC metric value. After transpilation, the mean increased slightly to 0.8812 but with greater variability. Some circuits had a significant decrease in the LC metric, with a wider range of LC values post-transpilation. This scenario differs from the original sample, where the variability of the LC metric was much more marked.

For the IQ metric, all the samples in both treatments are flattened on a value of zero.

The value of IdQ before transpilation is consistently zero, indicating no variation. In contrast, the after-transportation group has some variability, with a low mean of 0.05 and a standard deviation of 0.22. Although most values are zeros, the maximum value reaches 1.00, indicating a significant outlier. The before group aligns closely with the after group, except for the outlier.

The mean ROC metric before transpilation was 0.03, indicating a low average ROC value with some variability. However, most circuits had ROC metrics close to zero. After transpilation, the mean ROC metric was slightly lower at 0.01, and most circuits still had ROC metrics close to zero. This scenario resembles what was observed with the original sample.

Focusing on Table 6 it is possible to observe that, differently from the original sample, only the IdQ metric showed a statistically significant difference, with a *p-value* of 0.035 and a slightly small negative effect size ($g = -0.322$).

Table 6 Results of the Wilcoxon signed-rank test for comparing the pre- and post-transpilation metrics of the synthetic samples.

Metric	*p-value*	Hedge's g	Interpretation
LC	0.12	-0.03	Negligible
IQ	N/A	N/A	N/A
IdQ	**0.04**	-0.32	Small
ROC	0.35	0.14	Small

This was expected since descriptive statistics showed very similar values for all the metrics. Hence, this data allows us to reject only $H_0^{(1)}(IdQ)$

Main findings for RQ$_1$

In the original sample, transpilation significantly impacts identifying quantum-specific code smells, particularly in the Long Circuit (LC) and Idle Qubits (IdQ) metrics. The difference in smelliness values before and after transpilation is noteworthy. The LC metric decreases after transpilation, while the IdQ metric increases. However, the IQ and ROC metrics show no significant changes. In the synthetic and much bigger sample, transpilation significantly affects only the IdQ metric.

4.2 **RQ$_2$**: Impact of Different Gate Sets on the Smell Presence

Table 7, Table 8, Table 9, Table 10 show the values of the IQ, IdQ, LC, and ROC metrics for each gate set, respectively, alongside descriptive statistics, which provide valuable insights into the patterns observed for each metric.

The IQ metric distributions are mostly minimal for all treatments, concentrated around zero, suggesting that most data falls below the smelliness threshold, except for outliers representing rare cases of IQ smell. The highest IQ values are observed in the `ibm_perth` treatment (593.0), `ibm_sherbrokee` (24,764.0), `original` (15,650.0), `rpcx` (583.0), and `simple` (12,008.0).

The IdQ metric distributions vary across treatments. Outliers in all treatments indicate significant deviations in the IdQ values. The `ibm_perth` treatment shows a consistent distribution of IdQ, while `ibm_sherbrokee` exhibits greater variability. `original` primarily concentrates its IdQ values around zero, while both `rpcx` and `simple` have stable distributions.

The LC metric highlights distinct differences among the treatments. The median LC values for `original`, `ibm_perth`, and `ibm_sherbrokee` fall below the smelliness threshold, with `original` at 0.39 and both `ibm_perth` and `ibm_sherbrokee` at 0.46.

The ROC metric provides insights into the presence of repeated sets of operations. The `ibm_perth` treatment has an average ROC value of approximately 1.7, indicating a moderate count of repeated sets of operations. In contrast,

Table 7 IQ metric across different treatments with summary statistics

Subject	`original`	`ibm_perth`	`ibm_sherbrokee`	`rpcx`	`simple`
adapt_vqe	5.00	5.00	0.00	5.00	5.00
ae	0.00	0.00	2.00	0.00	8.00
fae	0.00	0.00	0.00	0.00	0.00
grover	1.00	1.00	1.00	0.00	0.00
hhl	1.00	0.00	0.00	1.00	0.00
iae	0.00	0.00	0.00	0.00	0.00
ipe	0.00	0.00	0.00	0.00	0.00
mlae	0.00	0.00	0.00	0.00	0.00
phase_estimation	0.00	0.00	0.00	0.00	0.00
qaoa	0.00	0.00	0.00	0.00	0.00
qeom	0.00	0.00	0.00	17.00	14.00
qgan	0.00	0.00	0.00	0.00	0.00
qsvc	0.00	0.00	0.00	0.00	0.00
shor	15,650.00	593.00	24,764.00	583.00	12,008.00
vqc	0.00	0.00	0.00	0.00	0.00
vqd	0.00	0.00	4.00	0.00	0.00
vqe	0.00	0.00	4.00	0.00	0.00
Mean	921.00	35.23	1,457.35	35.64	707.94
Std	3,795.50	143.73	6,005.98	141.11	2,911.96
Min	0.00	0.00	0.00	0.00	0.00
25%	0.00	0.00	0.00	0.00	0.00
50%	0.00	0.00	0.00	0.00	0.00
75%	0.00	0.00	1.00	0.00	0.00
Max	15,650.00	593.00	24,764.00	583.00	12,008.00

`ibm_sherbrokee` has a lower average ROC value of around 1.1, suggesting fewer repeated sets.

Table 11 showcases the results of the Friedman test, which aimed to compare different target gate sets based on the four metrics.

Among the metrics analyzed, the LC, IdQ, and ROC metrics exhibit significant differences among the target gate sets. The LC metric shows a highly significant difference (*p-value* = 0.000), suggesting notable variations in the performance of long circuits across different gate sets. Similarly, the IdQ and ROC metrics display significant differences (textitp-value = 0.001 and 0.002, respectively), indicating variations in the behavior of idle qubits and repeated sets of operations on the circuit based on the gate sets used. In contrast, the IQ metric does not reveal a significant difference (textitp-value = 0.690), implying that the choice of gate set does not strongly influence this metric.

Table 8 IdQ metric across different treatments with summary statistics

Subject	`original`	`ibm_perth`	`ibm_sherbrokee`	`rpcx`	`simple`
adapt_vqe	19.00	15.00	23.00	10.00	22.00
ae	821.00	8.00	18.00	11.00	10.00
fae	0.00	0.00	0.00	0.00	0.00
grover	2.00	2.00	1.00	1.00	0.00
hhl	1.00	0.00	1.00	1.00	0.00
iae	0.00	0.00	0.00	0.00	0.00
ipe	0.00	0.00	0.00	0.00	0.00
mlae	0.00	0.00	0.00	0.00	0.00
phase_estimation	0.00	0.00	0.00	0.00	0.00
qaoa	0.00	0.00	0.00	0.00	0.00
qeom	19.00	25.00	35.00	17.00	14.00
qgan	0.00	0.00	2.00	0.00	0.00
qsvc	1.00	1.00	1.00	1.00	1.00
shor	15,650.00	11,664.00	24,764.00	11,503.00	12,008.00
vqc	1.00	1.00	2.00	1.00	1.00
vqd	0.00	0.00	4.00	0.00	0.00
vqe	0.00	0.00	4.00	0.00	0.00
Mean	971.41	689.17	1,462.05	679.11	709.17
Std	3,787.70	2,828.15	6,004.78	2,789.25	2,911.64
Min	0.00	0.00	0.00	0.00	0.00
25%	0.00	0.00	0.00	0.00	0.00
50%	0.00	0.00	1.00	0.00	0.00
75%	2.00	2.00	4.00	1.00	1.00
Max	15,650.00	11,664.00	24,764.00	11,503.00	12,008.00

The following presents the post-hoc analysis results for LC, IdQ, and ROC. Since the Friedman test gave no statistically significant result for IQ, we did not conduct a post-hoc analysis, and we failed to reject $H_0^{(2)}(IQ)$.

Table 12 shows the pairwise Wilcoxon signed-rank test results for comparing different target gate sets regarding the LC metric. Among the comparisons made for the LC metric, the pairs `ibm_perth` and `rpcx` and `ibm_perth` and `simple` exhibited significant *p-values* of 0.038 and 0.021, respectively. These significant textitp-values indicate notable differences in the LC metric between these pairs of gate sets. When analyzing the effect sizes measured by Hedge's g, we observed values of -0.307 for the `ibm_perth` vs. `rpcx` comparison and -0.379 for the `ibm_perth` vs. `simple` comparison, both indicating a tendency of `ibm_perth` to have lower values (more smelly) than the other two. These effect sizes suggest moderate to large differences in the LC metric between the compared gate sets. On the other hand, the remaining comparisons did not yield significant *p-values*, implying that we did not observe significant differences in the LC metric between those specific pairs of gate sets. Nonetheless, it is important to note that although the *p-values* were not significant, Hedge's *g*

Table 9 LC metric across different treatments with summary statistics

Subject	`original`	`ibm_perth`	`ibm_sherbrokee`	`rpcx`	`simple`
adapt_vqe	0.00	0.00	0.00	0.00	0.00
ae	0.00	0.03	0.00	0.03	0.02
fae	0.10	0.70	0.70	0.87	0.87
grover	0.22	0.53	0.49	0.53	0.70
hhl	0.75	0.53	0.49	0.75	0.81
iae	0.22	0.87	0.87	0.93	0.93
ipe	0.96	0.96	0.96	0.96	0.96
mlae	0.01	0.87	0.87	0.93	0.93
phase_estimation	0.96	0.96	0.96	0.96	0.96
qaoa	0.81	0.81	0.81	0.87	0.93
qeom	0.01	0.00	0.00	0.01	0.02
qgan	0.75	0.46	0.42	0.70	0.75
qsvc	0.49	0.37	0.24	0.42	0.49
shor	0.00	0.00	0.00	0.00	0.00
vqc	0.39	0.19	0.12	0.34	0.39
vqd	0.70	0.46	0.46	0.93	0.93
vqe	0.70	0.46	0.46	0.93	0.93
Mean	0.41	0.48	0.46	0.59	0.62
Std	0.36	0.34	0.35	0.38	0.38
Min	0.00	0.00	0.00	0.00	0.00
25%	0.01	0.19	0.12	0.34	0.39
50%	0.39	0.46	0.46	0.75	0.81
75%	0.75	0.81	0.81	0.93	0.93
Max	0.96	0.96	0.96	0.96	0.96

can still provide valuable information about the effect sizes. In this context, effect sizes ranged from -0.471 to 0.182, indicating moderate differences in the LC metric for those non-significant comparisons. In this scenario, we can confidently reject $H_0^{(2)}(LC)$ in favor of the alternative hypothesis.

Table 13 presents the results of the pairwise Wilcoxon signed-rank test, which serves as a post-hoc analysis for the IdQ (Idle Qubits) metric, comparing different target gate sets. The *p-values* obtained after applying the Bonferroni correction are examined to determine significant differences between the pairs, and the effect size, measured by Hedge's g, provides insights into the magnitude of these differences. None of the pairwise comparisons for the IdQ metric reached statistical significance, as all *p-values* are above the threshold of 0.05. The Bonferroni correction, which adjusts for multiple comparisons, likely contributed to this outcome. The relatively small differences between the gate sets may not have surpassed the stringent significance threshold. Furthermore, when

Table 10 ROC metric across different treatments with summary statistics

Subject	`original`	`ibm_perth`	`ibm_sherbrokee`	`rpcx`	`simple`
adapt_vqe	7.00	3.00	1.00	3.00	0.00
ae	49.00	0.00	1.00	0.00	0.00
fae	4.00	1.00	1.00	0.00	0.00
grover	1.00	0.00	0.00	0.00	1.00
hhl	0.00	2.00	1.00	0.00	0.00
iae	4.00	0.00	0.00	0.00	0.00
ipe	0.00	0.00	0.00	0.00	0.00
mlae	15.00	0.00	0.00	0.00	0.00
phase_estimation	0.00	0.00	0.00	0.00	0.00
qaoa	0.00	0.00	0.00	0.00	0.00
qeom	0.00	10.00	3.00	0.00	0.00
qgan	0.00	1.00	1.00	0.00	0.00
qsvc	1.00	2.00	2.00	1.00	1.00
shor	N/A	N/A	N/A	N/A	N/A
vqc	2.00	4.00	3.00	2.00	2.00
vqd	0.00	2.00	2.00	0.00	0.00
vqe	0.00	2.00	2.00	0.00	0.00
Mean	5.18	1.68	1.06	0.37	0.25
Std	12.33	2.54	1.06	0.88	0.57
Min	0.00	0.00	0.00	0.00	0.00
25%	0.00	0.00	0.00	0.00	0.00
50%	0.50	1.00	1.00	0.00	0.00
75%	4.00	2.00	2.00	0.00	0.00
Max	49.00	10.00	3.00	3.00	2.00

Table 11 Results of the Friedman test for the comparison among the different target gate sets. Significant p-values are highlighted in bold.

Metric	Q	*p-value*
LC	31.88	**<0.01**
IQ	2.25	0.69
IdQ	19.66	**<0.01**
ROC	17.08	**<0.01**

considering the effect sizes measured by Hedge's g, the values range from -0.161 to 0.163, indicating small effect sizes. These values suggest that the observed differences in the IdQ metric between the compared gate sets are modest. Despite this, since the Friedman test gave a significant result, we can

Table 12 Results of the pairwise Wilcoxon signed-rank test for comparing the different target gate sets for the LC metric. Significant p-values are highlighted in bold.

Gate Set A	Gate Set B	*p-value*	Hedge's g	Interpretation
`ibm_perth`	`ibm_sherbrokee`	0.34	0.06	Small
`ibm_perth`	`original`	1.00	0.18	Small
`ibm_perth`	`rpcx`	**0.04**	-0.31	Small
`ibm_perth`	`simple`	**0.02**	-0.38	Small
`ibm_sherbrokee`	`original`	1.00	0.12	Small
`ibm_sherbrokee`	`rpcx`	**0.02**	-0.36	Small
`ibm_sherbrokee`	`simple`	**0.02**	-0.43	Small
`original`	`rpcx`	0.45	-0.47	Medium
`original`	`simple`	0.06	-0.54	Medium
`rpcx`	`simple`	0.21	-0.07	Small

Table 13 Results of the pairwise Wilcoxon signed-rank test for the comparison among the different target gate sets for the IdQ metric. Significant p-values are highlighted in bold.

Gate Set A	Gate Set B	*p-value*	Hedge's g	Interpretation
`ibm_perth`	`ibm_sherbrokee`	0.11	-0.16	Small
`ibm_perth`	`original`	1.00	-0.08	Negligible
`ibm_perth`	`rpcx`	1.00	0.00	Negligible
`ibm_perth`	`simple`	1.00	-0.01	Negligible
`ibm_sherbrokee`	`original`	1.00	0.10	Negligible
`ibm_sherbrokee`	`rpcx`	0.14	0.16	Small
`ibm_sherbrokee`	`simple`	0.06	0.16	Small
`original`	`rpcx`	0.59	0.09	Negligible
`original`	`simple`	1.00	0.08	Negligible
`rpcx`	`simple`	1.00	-0.01	Negligible

reject $H_0^{(2)}(IdQ)$ in favor of the alternative hypothesis, although some deeper analyses and considerations should be carried out.

The pairwise Wilcoxon signed-rank test results comparing different target gate sets for the ROC metric are shown in Table 14. None of the comparisons resulted in *p-values* below the significance threshold of 0.05, indicating no statistically significant differences between the gate sets. It is important to note that the Bonferroni correction was applied, which adjusts the significance threshold to account for multiple comparisons. The differences between the distributions may also have contributed to the lack of statistical significance. Despite the absence of significant findings, the effect sizes measured by Hedge's g for the non-significant comparisons are worth noting. These effect sizes ranged from -0.459 to 0.926, suggesting small to moderate differences in the ROC metric between the gate sets. Although these differences were not statistically significant, they may still be practically relevant. Nevertheless, the Friedman test produced a significant result, allowing us to reject $H_0^{(2)}(ROC)$ in favor

Table 14 Results of the pairwise Wilcoxon signed-rank test for the comparison among the different target gate sets for the ROC metric. Significant p-values are highlighted in bold.

Gate Set A	Gate Set B	*p-value*	Hedge's g	Interpretation
`ibm_perth`	`ibm_sherbrokee`	1.00	0.31	Small
`ibm_perth`	`original`	1.00	-0.38	Small
`ibm_perth`	`rpcx`	0.13	0.67	Medium
`ibm_perth`	`simple`	0.11	0.76	Medium
`ibm_sherbrokee`	`original`	1.00	-0.46	Medium
`ibm_sherbrokee`	`rpcx`	0.47	0.69	Medium
`ibm_sherbrokee`	`simple`	0.09	0.93	Large
`original`	`rpcx`	0.34	0.54	Medium
`original`	`simple`	0.58	0.55	Medium
`rpcx`	`simple`	1.00	0.16	Small

of the alternative hypothesis. However, further analysis and consideration are necessary.

Concerning the synthetic data, Figure 6 depicts the distribution of the metrics among the different target gate sets. The IQ metric varies across different gate sets. The `ibm_perth` and `ibm_sherbrokee` sets have moderate variability, with most circuits having low IQ values. The `original` and `simple` sets have no variability, with all circuits having minimal or no IQ values. The `rpcx` set has lower variability than `ibm_perth` and `ibm_sherbrokee`, with most circuits having low IQ values. Comparing these results with the original sample, it is possible to note that the distributions in the synthetic data are generally concentrated around lower values, with most circuits across all treatments having minimal IQ values. This is particularly evident in the `original` and `simple` sets, which show no variability. In contrast, the `ibm_perth`, `ibm_sherbrokee`, and `rpcx` sets display some degree of variability, although most of their circuits still have low IQ values. The presence of maximum values at 4.0 and 2.0 in the `ibm_perth` and `rpcx` sets, respectively, suggests the existence of a few outliers with slightly higher IQ metrics.

IdQ metric analysis shows variability across different sets. `ibm_perth` has a mean IdQ of 0.35, `ibm_sherbrokee` has a slightly higher average IdQ of 0.47, `original` has a lower average IdQ of 0.05, and `rpcx` has a mean IdQ of 0.2. Both `original` and `simple` have an average IdQ of 0.05, indicating minimal variability. Comparing these results with the original sample, the IdQ metric distributions show variability across treatments. While outliers are present in all treatments, indicating significant deviations, the `ibm_perth` treatment demonstrates a relatively consistent distribution of IdQ. In contrast, `ibm_sherbrokee` exhibits more considerable variability. `original` primarily concentrates its IdQ values around zero, similar to `simple`. Both `rpcx` and `simple` maintain stable distributions with minimal extremes.

Figure 6 shows that the mean and median values of LC were higher in all treatments as compared to the original dataset. The `original` treatment had

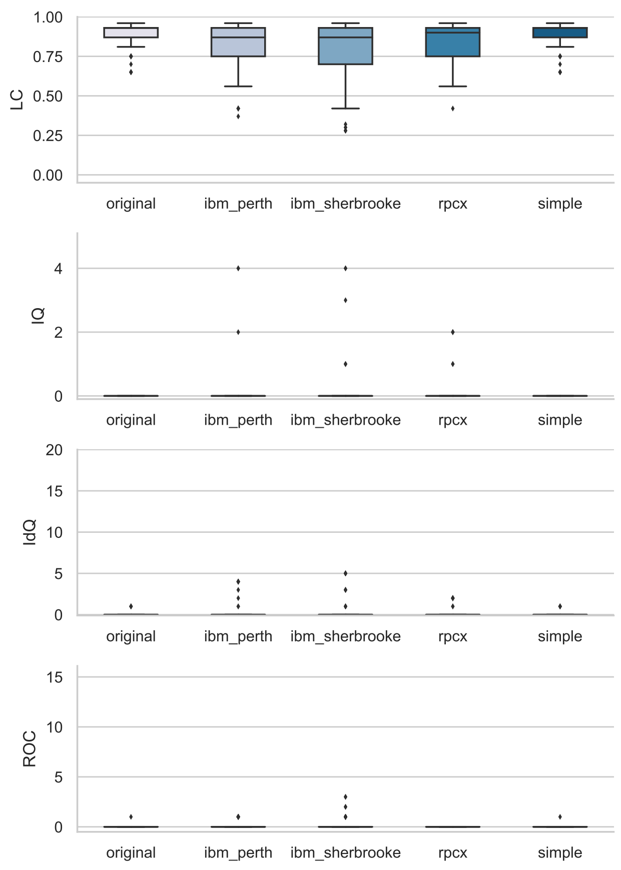

Fig. 6 Boxplot displaying the distribution of smelliness metrics among the different gate sets for the synthetic samples.

the highest mean LC value and the smallest standard deviation. The `ibm_perth` and `rpcx` treatments had a moderate variability, while `ibm_sherbrokee` is the most variable.

ROC over the various gate sets reveals differences in the presence of repeated sets of operations across groups. The `ibm_perth` has a mean ROC of 0.19 with a standard deviation of 0.39, while `ibm_sherbrokee` has a higher average ROC of 0.26. In contrast, `original` has a much lower average ROC of 0.01, while the `rpcx` treatment shows a uniform ROC distribution with an average and standard deviation of zero. `simple` treatment mirrors the pattern of the `original` group. Comparing this result with the original sample, it is possible to note some contrasts in the ROC metric. The `ibm_perth` treatment, with an average ROC value of approximately 1.70 in the original data, indicates a moderate count of repeated sets of operations. In contrast, `ibm_sherbrokee`, with a lower average ROC value of around 1.10, suggests fewer repeated sets. This

Table 15 Results of the Friedman test for comparing the different target gate sets of the synthetic samples. Significant p-values are highlighted in bold.

Metric	Q	*p-value*
LC	207.27	**<0.01**
IQ	12.03	**0.02**
IdQ	45.86	**<0.01**
ROC	70.35	**<0.01**

Table 16 Results of the pairwise Wilcoxon signed-rank test for comparing the different target gate sets for the IQ metric on the synthetic data. Significant adjusted p-values are highlighted in bold.

Gate Set A	Gate Set B	*p-adjust*	Hedge's *g*	Interpretation
`ibm_perth`	`ibm_sherbrokee`	6.75	-0.06	Negligible
`ibm_perth`	`original`	1.02	0.24	Small
`ibm_perth`	`rpcx`	5.81	0.04	Negligible
`ibm_perth`	`simple`	1.02	0.24	Small
`ibm_sherbrokee`	`original`	**<0.01**	0.30	Small
`ibm_sherbrokee`	`rpcx`	5.66	0.11	Small
`ibm_sherbrokee`	`simple`	**<0.01**	0.30	Small
`original`	`rpcx`	**<0.01**	-0.31	Small
`original`	`simple`	N/A	N/A	N/A
`rpcx`	`simple`	**<0.01**	0.31	Small

comparison highlights differences in the complexity and structure of quantum circuits across different treatments, with some treatments like `ibm_perth` and `ibm_sherbrokee` showing more complexity in terms of repeated operations than others like `original` and `simple`.

Table 15 depicts the results of the Friedman test for all the metrics computed on the synthetic dataset, among all the gate sets. It is immediately possible to note that differently from the original sample, for all the metrics a statistically significant result was achieved, with very low *p-values*, while with the original data, the IQ metric was not significant.

Table 16 shows the post-hoc analysis results for the IQ metric. The metric was not subject to post-hoc analysis in the original data because the Friedman test did not produce a statistically significant result. The table shows that only four comparisons produced a statistically significant result: `ibm_sherbrokee` vs. `original`, `ibm_sherbrokee` vs. `simple`, `original` vs. `rpcx`, and `rpcx` vs. `simple`. In all cases, the effect size was between 0.20 and 0.40, indicating a moderate positive difference. Hence, this data allows us to reject $H_0^{(2)}(IQ)$

Table 17 depicts the post-hoc analysis results for the IdQ metrics on the synthetic dataset. These results differed from the original data presented in Table 13. In this case, all treatments except `original` and `simple` had an identical distribution, which resulted in no significant findings. However, all

Table 17 Results of the pairwise Wilcoxon signed-rank test for comparing the different target gate sets for the IdQ metric on the synthetic data. Significant adjusted p-values are highlighted in bold.

Gate Set A	Gate Set B	*p-adjust*	Hedge's g	Interpretation
`ibm_perth`	`ibm_sherbrokee`	0.28	-0.09988	Negligible
`ibm_perth`	`original`	**<0.01**	0.38470	Small
`ibm_perth`	`rpcx`	**<0.01**	0.17240	Small
`ibm_perth`	`simple`	**<0.01**	0.38470	Small
`ibm_sherbrokee`	`original`	**<0.01**	0.44677	Medium
`ibm_sherbrokee`	`rpcx`	**<0.01**	0.26570	Small
`ibm_sherbrokee`	`simple`	**<0.01**	0.44677	Medium
`original`	`rpcx`	**<0.01**	-0.33778	Small
`original`	`simple`	N/A	N/A	N/A
`rpcx`	`simple`	**<0.01**	0.33778	Small

Table 18 Results of the pairwise Wilcoxon signed-rank test for comparing the different target gate sets for the LC metric on the synthetic data. Significant adjusted p-values are highlighted in bold.

Gate Set A	Gate Set B	*p-adjust*	Hedge's g	Interpretation
`ibm_perth`	`ibm_sherbrokee`	**<0.01**	0.134524	Small
`ibm_perth`	`original`	**<0.01**	-0.558788	Medium
`ibm_perth`	`rpcx`	**<0.01**	-0.278662	Small
`ibm_perth`	`simple`	**<0.01**	-0.558788	Medium
`ibm_sherbrokee`	`original`	**<0.01**	-0.624340	Medium
`ibm_sherbrokee`	`rpcx`	**<0.01**	-0.385460	Small
`ibm_sherbrokee`	`simple`	**<0.01**	-0.624340	Medium
`original`	`rpcx`	**<0.01**	0.296363	Small
`original`	`simple`	N/A	N/A	N/A
`rpcx`	`simple`	**<0.01**	-0.296363	Small

treatments except `ibm_perth` vs. `ibm_sherbrokee` and `original` vs. `rpcx` showed a moderate positive effect size, with Hedges' g values ranging between 0.20 and 0.50 (in absolute value). This indicates that there was a noticeable difference between the treatments. Hence, this data allows us to reject $H_0^{(2)}(Idq)$

Table 18 presents the outcomes of the post-hoc analysis for the LC metric. It is noticeable that in comparison to the results reported in Table 12, where only `ibm_perth` vs. `rpcx`, `ibm_perth` vs. `simple`, `ibm_sherbrokee` vs. `rpcx` and `ibm_sherbrokee` vs. `simple` showed a statistically significant result with moderate negative effect size, in this scenario, `ibm_perth` vs. `original`, `ibm_perth` vs. `ibm_sherbrokee`, and `ibm_sherbrokee` vs. `original` were also statistically significant. Not only were the *p-values* remarkably low, but also the effect sizes were moderate to high, ranging between 0.30 and 0.60 (in absolute values). All effect sizes were negative except for `ibm_perth` vs. `ibm_sherbrokee`. Hence, this data allows us to reject $H_0^{(2)}(LC)$

Table 19 Results of the pairwise Wilcoxon signed-rank test for comparing the different target gate sets for the ROC metric on the synthetic data. Significant adjusted p-values are highlighted in bold.

Gate Set A	Gate Set B	*p-adjust*	Hedge's *g*	Interpretation
`ibm_perth`	`ibm_sherbrokee`	0.83	-0.14	Small
`ibm_perth`	`original`	**<0.01**	0.62	Medium
`ibm_perth`	`rpcx`	**<0.01**	0.68	Medium
`ibm_perth`	`simple`	**<0.01**	0.62	Medium
`ibm_sherbrokee`	`original`	**<0.01**	0.58	Medium
`ibm_sherbrokee`	`rpcx`	**<0.01**	0.61	Medium
`ibm_sherbrokee`	`simple`	**<0.01**	0.58	Medium
`original`	`rpcx`	3.17	0.14	Small
`original`	`simple`	N/A	N/A	N/A
`rpcx`	`simple`	3.17	-0.14	Small

Table 19 presents the outcomes of the post-hoc analysis for the ROC metric. It is noticeable that in comparison to the results reported in Table 14, where no statistically significant result was found, here we have all but `ibm_perth` vs. `ibm_sherbrokee`, `original` vs. `rpcx`, `original` vs. `simple`, and `rpcx` vs. `simple` showing a statistically significant result. Like in the case previously discussed, all the *p-values* are remarkably low, and effect size values are fairly large, with values ranging between 0.50 and 0.60 (in absolute values) and always positive. Hence, this data allows us to reject $H_0^{(2)}(ROC)$

Main findings for RQ$_2$

When converting quantum circuits to different gate sets, there are significant variations in quantum-specific code issues. LC, IdQ, and ROC metrics demonstrate notable differences among gate sets, indicating that *smelliness* can relate to the executing environment. These differences are even more significant in the synthetic sample, where LC, IdQ, ROC, and IQ metrics have notable differences when converting to different gate sets.

5 Discussion and Lessons Learnt

5.1 On the Effects of Transpilation on the Smell Presence

The study unveils significant findings regarding the impact of transpilation on circuit smells, shedding light on the intricate relationship between the two. The results demonstrate a notable distinction in the presence of smells before and after transpilation, at least for some specific smells. Remarkably, the study reveals that circuits initially unaffected by any smells can acquire them post-transpilation, with the effect particularly pronounced for the LC smell.

Table 20 Results of the comparison of the number of pre- and post-transpilation smells by applying the thresholds [14].

Smell	Before	After
LC	1	10
IQ	1	4
IdQ	1	8
ROC	3	6

There seems to be a noticeable rise in smells following transpilation, which can be attributed to the circuit's changes during the process. In particular, the LC smell appears very sensitive to transpilation because it considers aspects that undergo significant alterations during the process. These changes can potentially upset the circuit's delicate equilibrium, resulting in new smells not present beforehand.

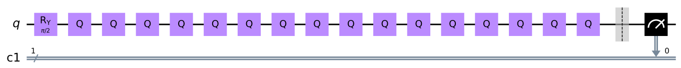

Fig. 7 MLAE circuit representation before transpilation

To explain this phenomenon, we consider a circuit named "mlae". The circuit initially had specific metrics (i.e., Width 2, Depth 18, 1 Qubit, 1 Classical bit, and 18 Gates), which can be easily inferred by Figure 7. However, after transpilation, significant metric changes resulted in a Width of 2, a Depth of 130, 1 Qubit, 1 Classical bit, and 130 Gates, which can be observed in Figure 8. The non-transpiled version had an LC value of 0.53; hence, the LC smell did not affect it. In contrast, the transpiled version had an LC value of 0.01, indicating a substantial increase in the intensity of the LC smell.

Furthermore, while other smells experience changes after transpilation, they are comparatively less affected. Intriguingly, the IQ smell, which pertains to the initialization of the circuit, remains unaffected by transpilation since the initialization is preserved throughout the process. This observation suggests that certain smells are more resilient to the transformations induced by transpilation, allowing them to persist or remain absent even after the circuit undergoes significant modifications.

As Table 20 depicts, by applying on the metrics the thresholds that allow the detection of the smells [14], it is possible to note how the number of smelly circuits changes before and after transpilation. Despite the variation can be statistically significant or not, it is possible to appreciate such change. The case of LC smell is the most evident, with a variation of nine more smelly circuits after the transpilation.

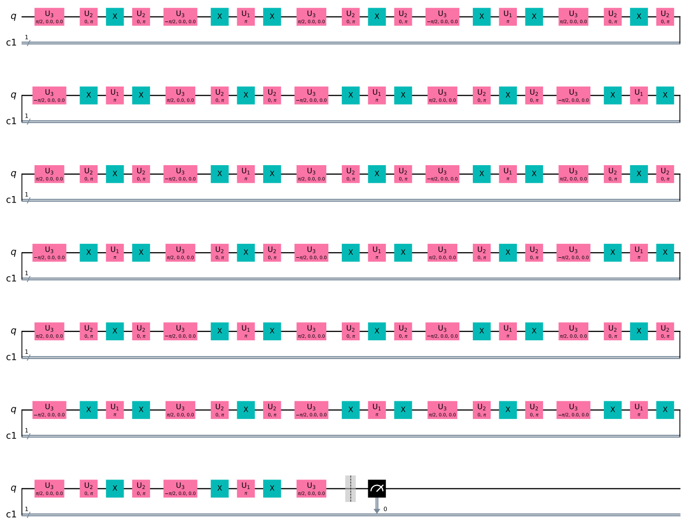
Fig. 8 MLAE circuit representation after transpilation to `original` gate set.

The implications of these findings are significant, as they suggest that a circuit that initially exhibits favorable characteristics and is deemed *good* by various metrics can unexpectedly acquire smells following transpilation. This predicament places developers in a powerless position, unable to control or prevent the emergence of these smells despite their best efforts in circuit design and optimization. This result acquires even more importance if we reflect on the possible impacts of such smells. For example, the LC and ROC smells, which imply a deeper circuit, can seriously impact the correctness of the execution of the circuit. Deeper circuits are more prone to errors during their execution in a real quantum machine [5, 14, 23]. Therefore, further investigation is imperative to comprehend the impact and potential harm these smells can inflict.

Take Away Message. Transpilation can cause circuit smells, especially LC smell. Some previously smell-free circuits may develop smells after transpilation. IQ smell is largely unaffected. Further exploration is needed to understand the impact and consequences of these smells. Developers may have limited ability to prevent their appearance.

Table 21 Results of the comparison among the different target gate sets for the different metrics.

smell	`original`	`ibm_perth`	`ibm_sherbrokee`	`rpcx`	`simple`
LC	10	9	11	6	6
IQ	4	3	5	4	4
IdQ	8	7	11	8	6
ROC	6	7	5	2	1

5.2 On the Impact of Different Gate Sets on the Smell Presence

The investigation demonstrates that selecting gate sets when transpiling quantum circuits can significantly impact the level of *smelliness* and the number and type of smells in the circuits. The presence and severity of smells vary depending on the type of gate set used. Developers must pay particular attention to LC smells as they are susceptible to gate set selection. This result is evident if we consider Table 21, where similarly to Table 20, we applied the smells thresholds [14] on each metric to detect the smelly circuits. Indeed, LC is the smell showing more variation in affected circuits given the target gate set. The situation for IdQ and ROC is similar. Different gate sets use different fundamental gates, which can significantly affect the structure of the transpiled circuits and disrupt the interactions between qubits and gates. Conversely, a gate set choice does not significantly affect IQ smells, demonstrating that the transpiler's optimization strategies can handle idle qubits regardless of the selected gate set. However, developers have limited control over gate set selection, usually determined by the underlying hardware and software stack. Therefore, it is crucial to understand the impact of the gate set choice on circuit smelliness to anticipate potential issues and make informed decisions during development.

Future studies should comprehensively evaluate the impact of the gate set selection on circuit smelliness, including a broader range of gate sets, additional metrics, and a more extensive collection of quantum circuits. Researchers should also consider hardware characteristics, such as machine architecture, noise levels, and error rates, to fully understand the interplay between gate sets, hardware characteristics, and circuit smells. Researchers can develop practical tools and techniques for optimizing quantum circuits by understanding these factors more deeply. It is crucial to exercise caution when evaluating the concept of quantum-specific smells due to significant variations in smelliness metrics and the number of smells observed across different gate sets. Although identified smells offer valuable insights, the variability introduced by different gate sets suggests that the concept of smells may have little universal applicability or standardization. Therefore, future research should refine and contextualize the concept of quantum-specific smells, considering the specific characteristics and constraints associated with different gate sets and hardware platforms.

🔍 **Take Away Message.** The choice of gate set can affect the level and types of smells in quantum circuits, particularly LC smells. Developers should consider this impact, and further research is needed to refine the concept of quantum-specific smells for different gate sets and hardware.

6 Conclusion and Future Work

This study aimed to investigate the effect of transpilation on quantum-specific smells and how different gate sets impact them. We aimed to answer two research questions employing the original dataset of Chen *et al.* [14] comprising 17 circuits and one created by generating 100 synthetic circuits.

On the one hand, we examined the effect of transpilation on the presence of quantum-specific code smells. In particular, we investigated how transpiling can impact the metrics used by QSmell to detect the smells. The results showed these metrics could be impacted, particularly LC and IdQ. However, IdQ showed this significance over the two analyzed data samples, while LC was only on the original one.

On the other hand, we investigated the effect of transpiling to different gate sets on the presence of quantum-specific code smells. The results showed that the choice of gate set significantly impacts the presence and severity of smells in transpiled circuits. Different gate sets introduce distinct fundamental gates and optimization strategies, leading to circuit structural changes. The results obtained on the synthetic data highlighted this phenomenon even more.

These findings have significant implications for developers, as they highlight the limited control over the emergence of smells in transpiled circuits. Developers must be aware of the potential for circuit smells to arise after transpilation and carefully consider the choice of the gate set to mitigate their impact whenever possible. Indeed, further research is needed to assess the presence of the smell by creating a more solid and validated set of rules and thresholds, which sparks from the initial definition provided by Chen *et al.* [14]. Further research is needed to fully understand the consequences of these smells and their potential harm to quantum computations. Furthermore, the variability observed across different gate sets shows the need for careful evaluation and contextualization when applying the concept of quantum-specific smells, considering the characteristics and constraints associated with each gate set and hardware platform.

Data Availability

The datasets generated during and/or analysed during the current study are available in the figshare repository:
`https://doi.org/10.6084/m9.figshare.23799210`

Acknowledgements

This work has been partially supported by project "QUASAR: QUAntum software engineering for Secure, Affordable, and Reliable systems", grant 2022T2E39C, under the PRIN 2022 MUR program funded by by the EU - NGEU.

References

1. Api reference for qiskit.circuit.random.random_circuit. `https://docs.quantum.ibm.com/api/qiskit/0.19/qiskit.circuit.random.random_circuit`
2. Qiskit 0.43.0 documentation. `https://qiskit.org/documentation/`
3. Transpiler (qiskit.transpiler) — Qiskit 0.43.1 documentation. `https://qiskit.org/documentation/apidoc/transpiler.html`
4. Ahmad, A., Khan, A.A., Waseem, M., Fahmideh, M., Mikkonen, T.: Towards process centered architecting for quantum software systems. In: 2022 IEEE International Conference on Quantum Software (QSW), pp. 26–31. IEEE (2022)
5. Aleksandrowicz, G., Alexander, T., Barkoutsos, P., Bello, L., Ben-Haim, Y., Bucher, D., Cabrera-Hernández, F.J., Carballo-Franquis, J., Chen, A., Chen, C.F., et al.: Qiskit: An open-source framework for quantum computing. Accessed on: Mar **16** (2019)
6. Arcelli Fontana, F., Mäntylä, M.V., Zanoni, M., Marino, A.: Comparing and experimenting machine learning techniques for code smell detection. Empirical Softw. Engg. **21**(3), 1143–1191 (2016)
7. Arcoverde, R., Garcia, A., Figueiredo, E.: Understanding the longevity of code smells: preliminary results of an explanatory survey. In: Workshop on Refactoring Tools, pp. 33–36 (2011)
8. Armstrong, R.A.: When to use the Bonferroni correction. Ophthalmic and Physiological Optics **34**(5), 502–508 (2014). DOI 10.1111/opo.12131
9. Awan, U., Hannola, L., Tandon, A., Goyal, R.K., Dhir, A.: Quantum computing challenges in the software industry. a fuzzy ahp-based approach. Information and Software Technology **147**, 106896 (2022)
10. García de la Barrera, A., García-Rodríguez de Guzmán, I., Polo, M., Piattini, M.: Quantum software testing: State of the art. Journal of Software: Evolution and Process p. e2419 (2021)
11. Basili, V.R., Caldiera, G., Rombach, H.D.: The goal question metric approach. Encyclopedia of Software Engineering (1994)
12. Campos, J., Souto, A.: Qbugs: A collection of reproducible bugs in quantum algorithms and a supporting infrastructure to enable controlled quantum software testing and debugging experiments. arXiv preprint arXiv:2103.16968 (2021)

13. Chatzigeorgiou, A., Manakos, A.: Investigating the evolution of bad smells in object-oriented code. In: International Conference on the Quality of Information and Communications Technology, pp. 106–115. IEEE (2010)
14. Chen, Q., Câmara, R., Campos, J., Souto, A., Ahmed, I.: The smelly eight: An empirical study on the prevalence of code smells in quantum computing. In: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE) (2023)
15. Cunningham, W.: The wycash portfolio management system. OOPSLA-92 (1992)
16. De Stefano, M., Pecorelli, F., Di Nucci, D., Palomba, F., De Lucia, A.: Software engineering for quantum programming: How far are we? Journal of Systems and Software **190**, 111326 (2022). DOI 10.1016/j.jss.2022.111326
17. De Stefano, M., Pecorelli, F., Di Nucci, D., Palomba, F., De Lucia, A.: The Quantum Frontier of Software Engineering: A Systematic Mapping Study (2023). DOI 10.48550/arXiv.2305.19683
18. Di Nucci, D., Palomba, F., Tamburri, D., Serebrenik, A., De Lucia, A.: Detecting code smells using machine learning techniques: Are we there yet? In: Int. Conf. on Software Analysis, Evolution, and Reengineering (2018)
19. El aoun, M.R., Li, H., Khomh, F., Openja, M.: Understanding quantum software engineering challenges: An empirical study on stack exchange forums and github issues. In: 37th International Conference on Software Maintenance and Evolution (ICSME) (2021)
20. Fowler, M., Beck, K.: Refactoring: Improving the design of existing code. Addison-Wesley Longman Publishing Co., Inc. (1999)
21. Friedman, M.: The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. Journal of the American Statistical Association **32**(200), 675–701 (1937). DOI 10.1080/01621459.1937.10503522
22. Hedges, L.V.: Distribution Theory for Glass's Estimator of Effect size and Related Estimators. Journal of Educational Statistics **6**(2), 107–128 (1981). DOI 10.3102/10769986006002107
23. Hoare, T., Milner, R.: Grand challenges for computing research. The Computer Journal **48**(1), 49–52 (2005)
24. Knight, W.: Serious quantum computers are finally here. what are we going to do with them. MIT Technology Review. Retrieved on October **30**, 2018 (2018)
25. Lin, T., Fu, X., Chen, F., Li, L.: A novel approach for code smells detection based on deep leaning. In: International Conference on Applied Cryptography in Computer and Communications, pp. 171–174. Springer (2021)
26. Liu, H., Jin, J., Xu, Z., Bu, Y., Zou, Y., Zhang, L.: Deep learning based code smell detection. Transactions on Software Engineering (2019)
27. Moguel, E., Berrocal, J., García-Alonso, J., Murillo, J.M.: A roadmap for quantum software engineering: Applying the lessons learned from the classics. In: Q-SET@ QCE, pp. 5–13 (2020)
28. Moha, N., Gueheneuc, Y.G., Duchien, L., Le Meur, A.F.: Decor: A method for the specification and detection of code and design smells. Transactions

on Software Engineering **36**(1), 20–36 (2009)
29. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information, 10th anniversary ed edn. Cambridge University Press, Cambridge ; New York (2010)
30. Palomba, F., Bavota, G., Di Penta, M., Fasano, F., Oliveto, R., De Lucia, A.: A large-scale empirical study on the lifecycle of code smell co-occurrences. Information and Software Technology **99** (2018)
31. Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., Poshyvanyk, D., De Lucia, A.: Mining version histories for detecting code smells. Transactions on Software Engineering **41**(5), 462–489 (2014)
32. Palomba, F., Panichella, A., De Lucia, A., Oliveto, R., Zaidman, A.: A textual-based technique for smell detection. In: International conference on program comprehension (ICPC), pp. 1–10. IEEE (2016)
33. Paltenghi, M., Pradel, M.: Bugs in quantum computing platforms: an empirical study. Proceedings of the ACM on Programming Languages **6**(OOPSLA1), 1–27 (2022)
34. Pecorelli, F., Di Nucci, D., De Roover, C., De Lucia, A.: A large empirical assessment of the role of data balancing in machine-learning-based code smell detection. Journal of Systems and Software p. 110693 (2020)
35. Pecorelli, F., Lujan, S., Lenarduzzi, V., Palomba, F., De Lucia, A.: On the adequacy of static analysis warnings with respect to code smell prediction. Empirical Software Engineering **27**(3), 1–44 (2022)
36. Pecorelli, F., Palomba, F., Di Nucci, D., De Lucia, A.: Comparing heuristic and machine learning approaches for metric-based code smell detection. In: International Conference on Program Comprehension (ICPC), pp. 93–104. IEEE (2019)
37. Pecorelli, F., Palomba, F., Khomh, F., De Lucia, A.: Developer-driven code smell prioritization. In: International Conference on Mining Software Repositories (2020)
38. Piattini, M., Peterssen, G., Pérez-Castillo, R.: Quantum computing: A new software engineering golden age. ACM SIGSOFT Software Engineering Notes **45**(3), 12–14 (2020)
39. Piattini, M., Peterssen, G., Pérez-Castillo, R., Hevia, J.L., Serrano, M.A., Hernández, G., de Guzmán, I.G.R., Paradela, C.A., Polo, M., Murina, E., et al.: The talavera manifesto for quantum software engineering and programming. In: QANSWER, pp. 1–5 (2020)
40. Piattini, M., Serrano, M., Perez-Castillo, R., Petersen, G., Hevia, J.L.: Toward a quantum software engineering. IT Professional **23**(1), 62–66 (2021)
41. Shi, Y., Gokhale, P., Murali, P., Baker, J.M., Duckering, C., Ding, Y., Brown, N.C., Chamberland, C., Javadi-Abhari, A., Cross, A.W., et al.: Resource-efficient quantum computing by breaking abstractions. Proceedings of the IEEE **108**(8), 1353–1370 (2020)
42. Tsantalis, N., Chatzigeorgiou, A.: Identification of move method refactoring opportunities. Transactions on Software Engineering **35**(3), 347–367 (2009)

43. Tufano, M., Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., De Lucia, A., Poshyvanyk, D.: An empirical investigation into the nature of test smells. In: International conference on automated software engineering, pp. 4–15 (2016)
44. Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Di Penta, M., De Lucia, A., Poshyvanyk, D.: When and why your code starts to smell bad. In: International Conference on Software Engineering, vol. 1, pp. 403–414. IEEE (2015)
45. Wilcoxon, F.: Individual Comparisons by Ranking Methods. In: S. Kotz, N.L. Johnson (eds.) Breakthroughs in Statistics: Methodology and Distribution, Springer Series in Statistics, pp. 196–202. Springer, New York, NY (1992). DOI 10.1007/978-1-4612-4380-9_16
46. Williams, C.P.: Quantum Gates. In: C.P. Williams (ed.) Explorations in Quantum Computing, Texts in Computer Science, pp. 51–122. Springer, London (2011). DOI 10.1007/978-1-84628-887-6_2
47. Yarkoni, S., Raponi, E., Bäck, T., Schmitt, S.: Quantum annealing for industry applications: Introduction and review. Reports on Progress in Physics (2022)
48. Zhao, J.: Quantum software engineering: Landscapes and horizons. arXiv preprint arXiv:2007.07047 (2020)