OpaQue: Program Output Obfuscation for Quantum Software Circuits in Quantum Clouds

Tirthak Patel
Rice University
Houston, USA
tirthak.patel@rice.edu

Aditya Ranjan
Northeastern University
Boston, USA
ranjan.ad@northeastern.edu

Daniel Silver
Northeastern University
Boston, USA
silver.da@northeastern.edu

Harshitta Gandhi*
QBit Solutions Research
Miami, USA
gandhi.ha@northeastern.edu

William Cutler*
Oxford University
Oxford, United Kingdom
cutler.wi@northeastern.edu

Devesh Tiwari
Northeastern University
Boston, USA
d.tiwari@northeastern.edu

Abstract

Recent quantum software engineering efforts have made significant progress in testing and debugging quantum algorithms – however, providing confidentiality and privacy to quantum software in the cloud remains an unexplored critical area. OpaQue is the first solution to obfuscate quantum software and output to prevent the leaking of confidential information over the cloud. OpaQue implements a lightweight, scalable, and effective solution based on the unique principles of quantum computing to achieve this task.

CCS Concepts

• **Computer systems organization → Quantum computing**.

Keywords

Quantum Computing, Quantum Software Privacy, Quantum Cloud

ACM Reference Format:
Tirthak Patel, Aditya Ranjan, Daniel Silver, Harshitta Gandhi, William Cutler, and Devesh Tiwari. 2025. OpaQue: Program Output Obfuscation for Quantum Software Circuits in Quantum Clouds. In *2025 International Conference on Supercomputing (ICS '25), June 08–11, 2025, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3721145.3725771

1 Introduction

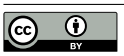
Quantum computing is an emerging technology that has the potential to accelerate and make possible the execution of many large-scale scientific, optimization, and machine-learning tasks [7, 25]. As quantum computing technology advances, multiple cloud-based quantum computing platforms are leveraged to develop and execute classically infeasible mission-critical tasks by government agencies and industry partners [13, 14, 27]. The solutions to these tasks are often business-sensitive and should be protected (e.g., the solution to a classically infeasible problem relevant to a defense program). Due to the nascent stage of quantum cloud computing, the cloud computing providers have full access to the end users' mission-sensitive programs and the output of such programs [24, 28]. Some prior art has recognized the importance of security and privacy for quantum software and focused on this challenge, although not solving the same problem as this work (protecting and providing confidentiality to the output of quantum software programs).

In particular, prior software engineering relevant efforts have primarily focused on testing and debugging quantum circuits [22, 34, 36] instead of providing confidentiality and privacy to quantum software. Prior works related to encrypting quantum information over networks [4, 35, 37] and securing third-party quantum compilers [29, 32] assume that the cloud hardware provider is an uncompromised entity that does not have intentional or unintentional snoopers on the quantum cloud platform that can analyze the output of quantum software. Even if the cloud provider is uncompromised, organizations may not want to disclose their tasks, proprietary quantum software code, and program solutions to the cloud provider. Protecting this information from the cloud provider is a non-trivial challenge as the user is essentially asking the hardware provider to run the "wrong" code and observe the "wrong" output but be able to recover the "correct" quantum output from the "wrong" output on the user's end. To achieve this, we propose OpaQue.

In the near term, it is anticipated that only a few entities in the world may have access to powerful quantum computers, and these quantum computers will be used to solve previously unsolved large-scale optimization problems, possibly without an explicit trust model between the cloud service provider and the user. Therefore, the solutions to such large-scale optimization problems will be considered sensitive and must be protected. OpaQue takes the first few steps toward preparing us for that future – by developing a novel software engineering solution for providing confidentiality and protection to quantum software programs (open-sourced) – further advancing the recent progress of quantum software engineering [22, 34, 36] that the community can build upon. We first provide a primer on relevant quantum computing concepts to help readers better contextualize the contributions of this work.

Quantum Computing: Brief Background. Qubits, Quantum Gates, and Circuits. The fundamental unit of quantum computing is the qubit, which is capable of representing a superposition (linear

*The authors contributed to this work while they were students at Northeastern University.



ICS '25, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-1537-2/25/06
https://doi.org/10.1145/3721145.3725771

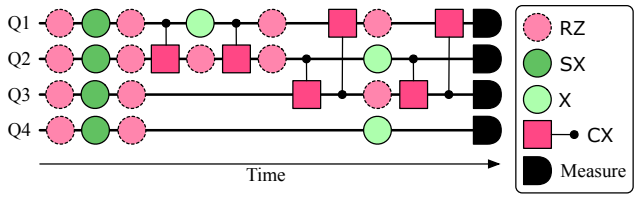


Figure 1: An example circuit representation of a quantum algorithm. The horizontal lines represent qubits with gates being applied to them in order from left to right.

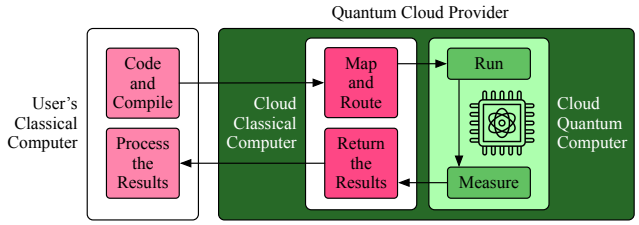


Figure 2: A quantum algorithm's execution consists of several steps on the user's and cloud's sides.

combination) of two orthogonal basis states. This is represented as $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are the complex amplitudes of the constituent basis states. Upon measurement, this superposition collapses such that the probability of measuring the state $|0\rangle$ is $\|\alpha\|^2$ and $\|\beta\|^2$ for measuring the $|1\rangle$ state. A general system of n entangled qubits is represented as a superposition of 2^n basis states: $|\psi\rangle = \sum_{k=0}^{k=2^n-1} \alpha_k|k\rangle$. As with one qubit, when the multi-qubit state is measured, it manifests as a projection to one of the basis states (where now the basis state is a state of n qubits). The probability of measuring state $|k\rangle$ is $\|\alpha_k\|^2$.

Quantum gates are applied to manipulate the qubit state. For example, the gates on IBM quantum computers consist of the CX, SX, X, and RZ gates [1]. SX, X, and RZ gates are one-qubit gates. In the Bloch-sphere representation, where a qubit's superposition state is represented on the surface of a sphere, one-qubit gates are categorized by the axis around which the rotation takes place and the rotation angle. The generalized rotation gate $\mathtt{RZ}(\theta)$ rotates the qubit state about the z-axis by angle θ. Similarly, the generalized $\mathtt{RX}(\beta)$ gate performs a rotation about the x-axis by angle β. The generalized RX gate is decomposed into the basis gates before execution on the IBM platform. Note that $\mathtt{X} = i\mathtt{RX}(\pi)$ and $\mathtt{SX} = e^{i\frac{\pi}{4}}\mathtt{RX}(\frac{\pi}{2})$. The CX (controlled-X) gate is a two-qubit entanglement gate and applies the X gate to the "target" qubit only when the "control" qubit is $|1\rangle$. All the gates can be represented using unitary matrices. A unitary matrix is a matrix U such that $U^\dagger U = I$, where $U^\dagger$ is the complex conjugate transpose of U and I is the identity matrix.

A quantum software program comprises a sequence of one- and multi-qubit gates, as shown in Fig. 1. The sequence is referred to as a quantum circuit. A quantum circuit also has a unitary matrix representation, U. At the end of the sequence, the qubits are measured to get the output (solution to the problem). This circuit must be prepared and measured multiple times to get a probability distribution over its basis states. *This probability distribution is the output of the quantum program, referred to as program output (solution).*

Quantum Computing Hardware and Noise. While OpaQue is generally applicable to any technology, in this paper, we evaluate it on IBM's superconducting-qubit computers as they have the advantage of being relatively easy to fabricate and operate, making them a popular choice [4, 18, 23].

A challenge in executing programs on real quantum computers is hardware noise effects. These noise effects include the state preparation and measurement (SPAM) errors, which cause the qubit to be incorrectly initialized and measured, the gate errors, which refer to the qubit state being incorrectly modified, and the decoherence errors, which refer to the loss of the coherence of a qubit's state. These errors are an order of magnitude higher for the CX gates than the X and SX gates; RZ gates do not have any error as they are implemented virtually using change of reference. The impact of hardware noise on program output adds challenge for OpaQue as it cannot apply obfuscation techniques that would increase the effects of hardware noise; the quality of the recovered "correct" program output should not be worse than what the user would have observed if the output was not obfuscated. *A useful quantum computing software engineering solution must demonstrate its effectiveness on current noise-prone quantum computers – this is why OpaQue is designed for and evaluated on real quantum hardware available on current quantum clouds.*

Quantum Execution Workflow. Fig. 2 shows the execution workflow of a quantum computing workload. The user writes the code and transpiles the circuit on their end. Then, the code is sent to the cloud, where the mapping and routing operations are performed to convert the circuit to a format that can be executed on the qubit connectivity of the selected quantum computer. Note that these steps can also be performed on the user's end, in which case the hardware provider simply runs the circuit. *Where exactly these steps are performed is not of consequence to OpaQue, as OpaQue is applied before these steps are run.* Then, the compiled circuit is run on the hardware, and the output is measured and returned to the user, who can then analyze the results.

Limitations of Existing Work. Previous works, as described below, are useful, but these approaches do not address the problem solved by OpaQue and cannot be modified to achieve OpaQue's goals. Their threat model is unrealistic, and the domain is limited to networks or compilers only – unlike OpaQue that targets the quantum software program itself (the first quantum software engineering toolkit of its kind).

Quantum Encryption for Communication Networks. Some previous works have focused on private quantum network protocols [4, 35, 37]. As an instance, Balaji et al. [4] implement a lattice cryptographic technique for post-quantum encryption with robustness against the Man-In-The-Middle and Sybil attacks. By the same token, Xu et al. [35] propose a system to solve the issue of hiding preambles via random repetition and nested hash coding to encrypt network data in a post-quantum manner. *These works require that the cloud provider is trusted (unlike OpaQue) as the code/data transferred over the network needs to be decrypted before the code is executed on the quantum computer.*

Protection from Third-Party Software. Some previous works have also focused on protecting quantum code from third-party compilers [24, 28, 29, 32]. For example, Suresh et al. [32] attempt to confuse the compiler by adding CX gates, which are removed after compilation and before hardware execution to get the correct output. Consistent with the above paper, Saki et al. [29] propose a split compilation approach where different code sections are sent to different compilers so that no one compiler has access to the whole code. The code is stitched back together before hardware execution. On the other hand, Phalak et al. [24] propose verification techniques to validate that third-party vendors are indeed dedicating the promised resources to the quantum code. *These works also do not obfuscate the output from the hardware as the circuits are decoded pre-execution and, hence, do not provide any privacy guarantee or confidentiality from the cloud hardware providers.*

OpaQue: Approach and Contributions. OpaQue is a simple yet effective obfuscation technique to obfuscate the quantum program output (and its circuit structure) of quantum software. OpaQue's simplicity lies in its approach to randomly inject X gates (similar to classical NOT gates) at the end of the quantum circuit. The location of X gate injection becomes the key – there are 2^n different permutations of injection sites, where n is the number of qubits in the quantum program. This approach allows the program output to become scrambled so it can only be decoded by the user (who holds the decoding key) but not the cloud provider.

While promising, as our evaluation demonstrates (Sec. 4), only injecting X gates does not provide significant structural divergence despite leveraging quantum circuit synthesis to "pull" back the appended X gates and merge them in the inner layers of the circuit. To mitigate this challenge, OpaQue demonstrates how to intelligently inject RX pairs and design a novel quantum circuit synthesis pass that strategically decomposes the new "X gates and RX pairs added" circuit into multiple blocks, transforms them in semantically equivalent but structurally different blocks, and then, combines them together – the resultant software circuit, by design, successfully obfuscates the location of added X gate and RX gate.

OpaQue's design demonstrates (a) how to exploit the reversibility property of quantum gates to obfuscate the output yet ensure that the injected gates do not alter the original program logic, (b) how to design its gate injection and circuit synthesis procedure such that the resulting obfuscated circuit does not become more sensitive to quantum hardware errors due to addition of new gates (Sec 2). Obfuscating circuit output and structure is naturally likely to increase gate count and depth (as a side effect) – but OpaQue demonstrates how its impact can be mitigated on real quantum hardware. *OpaQue's data and software are available at: https://zenodo.org/doi/10.5281/zenodo.10896069.*

OpaQue's Contributions:

- OpaQue presents the first approach to obfuscate quantum software circuits and outputs in quantum cloud environments – to ensure that quantum cloud providers cannot infer the solutions to previously unsolved, classically infeasible, large-scale optimization problems that can be business-sensitive and need to be protected in the client-side quantum software.
- OpaQue presents a design and implementation of a simple yet effective software framework for achieving its goal by (a) an intelligent combination of X gate and RX gate injections and (b) a novel quantum circuit synthesis procedure. OpaQue obfuscates the circuit output successfully. Despite the injection of additional quantum gates, OpaQue maintains the same solution quality as the original un-obfuscated quantum circuit – even in the presence of hardware errors.
- Our real-hardware and simulation evaluation of a diverse set of algorithms (up to 128 qubits) demonstrates that OpaQue *successfully hides the program output, solution states, and circuit structures*, using probability distribution distance and circuit structure distance metrics, while maintaining low compilation times and program output error.

Scope, Threat Model, and Threats to Validity. All prior works assume that quantum cloud hardware providers are trusted and are not snooping on the solutions to the quantum programs they execute on their hardware. In contrast, OpaQue assumes that only the user's local system is trusted. All other components involved in executing quantum code, including third-party compilers, networks, software stack on the quantum cloud, and hardware, are assumed to be open to snooping – hence, this threat model is more realistic and wider than prior works. We note that OpaQue is applicable to all types of quantum applications and hardware technologies, as it works at the level of the logical quantum circuit.

We recognize that all scientific works have limitations or threats to validity – OpaQue is not an exception. OpaQue's novel obfuscation and circuit synthesis components increase the confidentiality on the client side but may make the software maintenance, DevOps, and program comprehension more challenging if the original circuit is not shared. We are hopeful that OpaQue's novel obfuscation and circuit synthesis contributions may open future avenues for programming, testing, debugging, and repairing in quantum software engineering.

We note that although OpaQue obfuscates a quantum program's circuit structure as a side-benefit to make the obfuscation of the circuit output more effective, OpaQue does not make any theoretical claims about privatizing or hiding parts of the original quantum algorithm. Empirically, we demonstrate that OpaQue significantly obfuscates a quantum program's circuit structure using graph-based distances (Sec. 4), although this is not OpaQue's primary goal.

We note that while it is tempting, classical code obfuscation techniques are unsuitable to be applied in the quantum computing domain because of differences in computing models and observability. Quantum computing fundamentally relies on entanglement and superposition principles to perform computation; therefore, any classical method to inject randomness or obfuscation cannot be "disentangled" without destroying the quantum state and collapsing the computation.

2 OpaQue: Design and Implementation

We begin this section by providing an overview of OpaQue's design and then delve into the design details of each step implemented by OpaQue.

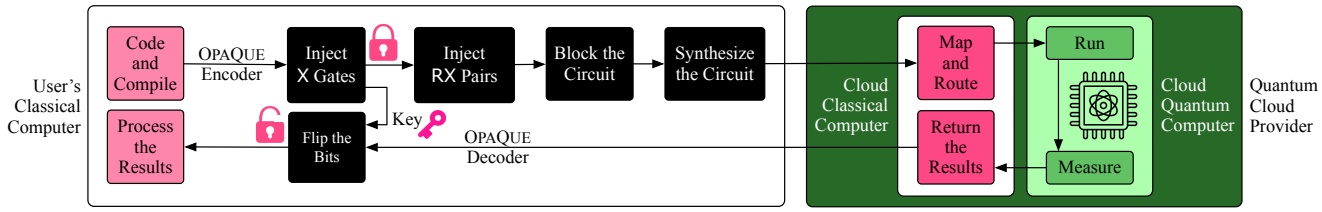


Figure 3: OpaQue's encoding and decoding procedures. All of OpaQue's steps are run entirely on the user side.

2.1 Overview of OpaQue's Design

Fig. 3 shows the execution workflow of a quantum program with OpaQue's steps included. OpaQue's encoding process consists of four steps. First, to hide the output, OpaQue randomly selects qubits and injects X gates at the end of the quantum circuits. This gives 2^n different permutations of X-gate injections, where n is the number of qubits in the quantum program. The chosen permutation becomes the decoding key when the output is returned after circuit execution. Second, to obfuscate the structure of the circuit, OpaQue injects RX gates with random rotations throughout the circuit.

As the third and fourth steps, to obfuscate the location of where the RX and X gates are inserted, OpaQue divides the circuit into two-qubit blocks and synthesizes them into a different gate structure that is mathematically equivalent to the original logic. At the end of these four steps, OpaQue-generated code has a significantly different circuit structure and program output distribution compared to the original code – this OpaQue-generated code is sent to be executed on the quantum cloud. OpaQue's decoder is a simple, one-step procedure. Post execution, when the program output is returned to the user, the bits in the output state need to be flipped in accordance with the key generated during the encoding process. Once this is done, the correct output is obtained, which only the user can access. Next, we describe the X-gate injection step.

2.2 Output Obfuscation using X-Gates

Injecting X gates at the end of the circuit (after all other gates in the original circuit logic have been executed) performs the function of flipping the output state bits. For example, consider a two-qubit program output with the following output probability distribution: $p(|00\rangle) = 0.6$, $p(|01\rangle) = 0.1$, $p(|10\rangle) = 0.1$, and $p(|11\rangle) = 0.2$. Let us set our key to be 01, i.e., we insert an X gate on and flip the first qubit (the least significant digit), and we do not insert an X gate on the second qubit (the most significant digit). This key will perform the following transformation: $|00\rangle \rightarrow |01\rangle$, $|01\rangle \rightarrow |00\rangle$, $|10\rangle \rightarrow |11\rangle$, and $|11\rangle \rightarrow |10\rangle$. Post the X-gate injection, the output probability distribution becomes: $p(|00\rangle) = 0.1$, $p(|01\rangle) = 0.6$, $p(|10\rangle) = 0.2$, and $p(|11\rangle) = 0.1$. This output probability distribution is different from the output distribution of the original distribution. In fact, the highest probability state, which is usually the most important state for most quantum algorithms, shifts from $|00\rangle$ to $|01\rangle$. Thus, it cannot be identified correctly post-obfuscation. *OpaQue specifically chooses X gates for injection because X gates serve as pure inverters, which makes it possible to decode the output with the key. Injecting any other quantum gate with arbitrary angles would require high-overhead and impractical tomography procedures to decode the original program output.*

The X-gate injection is shown visually in Fig. 4 (Step ❶). While the injected X gates are operationally the same as the other X gates in the circuit, we label them with a "X" for visualization purposes. In the above two-qubit example, the key is a randomly selected permutation from a uniform distribution of four possible permutations. Thus, it may appear that it can be guessed. However, in general, for an n-qubit output, the key is selected from 2^n possible permutations. Thus, it quickly becomes untenable to guess the key for realistic quantum algorithms. For example, a 32-qubit algorithm has over one billion different permutations, and a 128-qubit algorithm has over one hundred trillion trillion trillion different permutations. We study algorithms of both of these sizes in the evaluation section (Sec. 4). However, while it is not possible to surmise this key for realistic medium-to-large quantum algorithms in a brute-force manner, an adversary may still be able to inspect the circuit structure and identify where the X gates are injected as they have access to the circuit. We will see later in this section how OpaQue hides the injected X gates. Before that, we go over how OpaQue obfuscates the entire circuit structure using RX-gate injections.

2.3 Structure Obfuscation using RX-Gates

While the X gates obfuscate the output, the rest of the circuit structure can still be inspected by the adversary to extract the locations of X gate injection. Using circuit synthesis, one can "pull" X gates injected at the end of the circuits in the inner layers of the circuits to achieve partial structural obfuscation. While useful, our evaluation (Sec. 4) shows that such an approach is not as effective as desired – that is, it does not yield sufficient structural obfuscation.

Additionally, structural obfuscation for quantum circuits is useful for other purposes, too – for example, it prevents the adversary from learning about the functionality of the circuit. It is also useful to obfuscate sub-regions within a quantum circuit as those sub-regions may consist of full quantum algorithms by themselves. As an instance, the Quantum Fourier Transform (QFT) [21] logic frequently shows up in other quantum algorithm circuits. Thus, an adversary might be able to identify the region of the quantum circuit that forms the QFT logic if that region is not obfuscated. As another example, *the inputs to quantum algorithms are always supplied as circuit gates (e.g., parameterized rotation angles).* Therefore, it is essential to obfuscate the circuit structure to obfuscate any inputs as well. To facilitate obfuscation throughout the circuit, OpaQue injects RX-gate pairs with random angles throughout the circuit, as shown in Fig. 4 (Step ❷).

A pair consists of an RX gate with a randomly chosen angle and its inverse, an RX gate with the negative of the first angle, injected on the same qubit in direct sequence. We form pairs to ensure that whatever computational logic is inserted is immediately

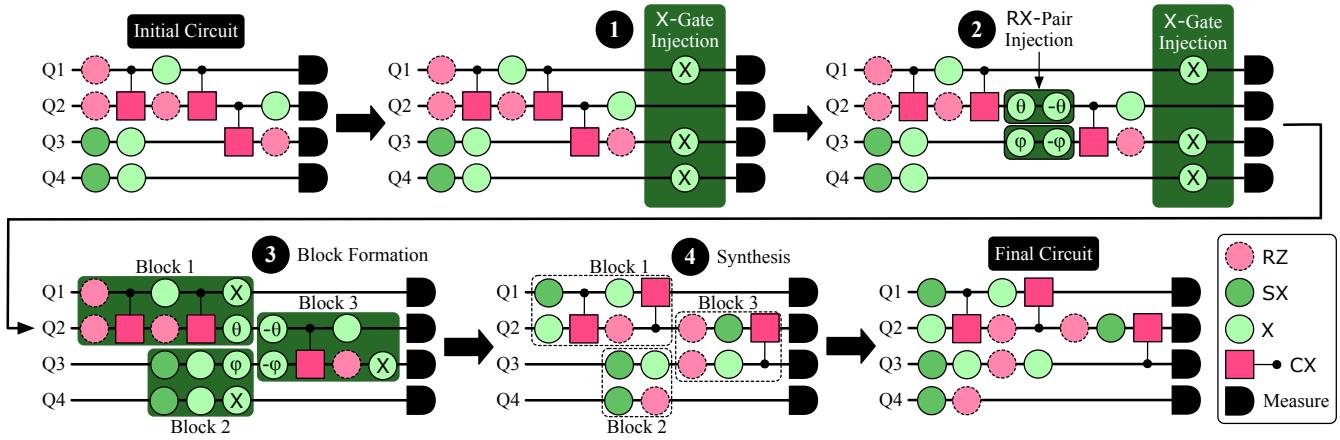


Figure 4: OpaQue's four-step encoding process. (1) Inject X gates on arbitrary qubits to obfuscate the output. (2) Inject RX pairs with random rotation angles throughout the circuit to obstruct the circuit structure. (3) Divide the circuit into several two-qubit blocks. (4) Synthesize the blocks into new gate logic, generating a circuit with obfuscated structure and output.

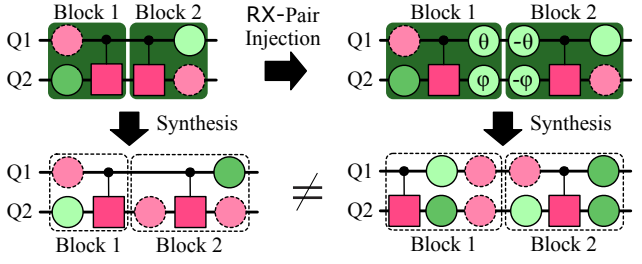


Figure 5: RX-pair injection obfuscates the circuit by generating synthesized circuits with different structures.

reverted and has no impact on the original program semantics – we only want the X-gate injections at the circuit end to affect the output as we can control its effect by decoding using the key. We choose the RX gate instead of other rotation gates as it provides sufficient diversity in terms of obfuscation because it decomposes into multiple hardware basis gates (RZ, SX, and X), allowing for diverse obfuscation patterns. One may ask how inserting pairs of RX gates obfuscates the structure since it does not affect the computation at all. We explain this next, along with how we choose the circuit locations to inject the RX gates.

2.4 Circuit Block Formation for Synthesis

We want to be able to hide the injected RX gates by generating new circuit logic. The way to achieve this is by using the process of synthesis. Synthesis takes a unitary matrix U corresponding to a circuit C and generates a new circuit $\hat{C}$ such that $\hat{C}$'s unitary representation is also U. However, this procedure scales exponentially in the number of qubits as the unitary dimensions are $2^n \times 2^n$ for an n-qubit circuit. Therefore, to perform this operation in a scalable manner, we must divide the circuit into manageable blocks such that each block can be separately synthesized, and the synthesized blocks can then be put back together to form the full quantum circuit. OpaQue restricts the size of the blocks to two qubits as the small size allows for scalable and exact synthesis [17] — the synthesis of bigger sized blocks may be accelerated by generating approximate solutions; however, we want to restrict our synthesis to exact solutions so as not to add unnecessary error to the original program output.

Fig. 4 (Step 3) provides a visual example of how these blocks are formed. In the example, we form three two-qubit blocks. Note that the injected X gates at the end can be absorbed into the last-most block for their respective qubits. The example also shows how *the RX gates are injected at the edges of the blocks. This allows for one gate from each pair to get absorbed into two separate blocks.* This means that each block now represents a different quantum logic than it would in the original circuit without the RX-gate injection. For example, the unitary representing block 3 would be different if it did not have the $\mathrm{RX}(-\theta)$ gate on qubit 2 and $\mathrm{RX}(-\phi)$ gate on qubit 3. When these blocks are now synthesized, they will generate very different logic compared to if they did not have the RX gates (depicted in Fig. 5). Thus, if one were to remove any given block or a region consisting of multiple blocks from the circuit, it obfuscates the logic. In fact, even if they executed that region, the circuit would generate meaningless output as the logic of the sub-regions is altered by RX gate injections. As a note, there is no benefit to having full pairs within a block as the computation cancels itself and will have no impact on the synthesis of that block. Note also that these RX-gate injections have no impact on the overall circuit output, even though they scramble the structure and block/sub-region outputs.

Algorithm 1 shows the pseudo-code for how these blocks are formed for a quantum circuit. The circuit gates are processed one by one in topological order – which is the order of the gates from the left to the right of the circuit – and assigned to different blocks. A block continues to be formed until a two-qubit gate is encountered that has one qubit in one block and the second qubit in another block or the circuit end is reached. The complexity of this algorithm is $O(g)$, where g is the number of gates in the circuit as it has to go through each gate only once. While there are other methods of forming these blocks (e.g., greedy heuristics to form deeper blocks), we chose this method due to its speed and result quality (Sec. 4). Once the blocks are formed, the RX gates are injected at the edges. An adversary cannot identify their edges and injection locations as

Algorithm 1 Algorithm for dividing the circuit into blocks.

```
G ← All of the circuit gates in topological order
B ← {∅}                          ▷ Set of all blocks, each block is a set of gates
for g in G do
    if g is a one-qubit gate then
        q ← The qubit that g runs on
        if b includes q for any b ∈ B then add g to b
        else create a new k ∈ B and add g to k
    else if g is a two-qubit gate then
        q1, q2 ← The two qubits that g runs on
        if b includes q1, q2 for any b ∈ B then add g to b
        else if b includes one of q1, q2 for any b ∈ B then
            Complete block b, create a k ∈ B, and add g to k
        else create a new k ∈ B and add g to k
return B
```

the logic gets altered by synthesis, and the block boundaries do not persist in the final full circuit.

2.5 Synthesis to Generate the Final Circuit

As mentioned earlier, there are a large number of different ways of realizing the same quantum logic. That means that a given unitary matrix, U, can be realized using many different circuit structures. We use synthesis to take a unitary matrix U corresponding to a block circuit C and generate a new circuit $\hat{C}$ such that $\hat{C}$'s unitary representation is also U. The U matrix for a block is simply calculated by multiplying the matrices (Kronecker product) of all the gates within the block. While there are many different ways of synthesizing a U, we use KAK decomposition [17] to construct the new circuit gate by gate, as it is an efficient method to synthesize two-qubit unitaries exactly. *We generate k circuits for the same block such that the generated circuits have the same number of two-qubit gates as the original block circuit because two-qubit gates tend to be highly noisy and can degrade the output quality.* Note that just because the synthesized circuit has the same number of two-qubit gates as the original circuit does not imply that those gates will be in the same position and orientation, which adds to the obfuscation.

On the other hand, we allow leeway in terms of the number of one-qubit physical (SX+X) and virtual (RZ) gates as these gates have little-to-none error effects and, therefore, can be of help for obfuscation. A naïve approach would call for selecting the circuit with the least number of SX+X gates out of the k generated circuits as that would have the least noise effects. However, OpaQue also considers the difference in the structure of the generated block and the original block for the purpose of obfuscation. To compare the structures of two circuits, it converts them into Directed Acyclic Graphs (DAGs) and uses the NetLSD divergence [5, 33] metric to assess their degree of similarity. NetLSD is calculated by computing the spectra of the normalized Laplacian matrices corresponding to the two DAGs and comparing their spectral node signature distributions. Thus, out of the generated k circuits, OpaQue takes the top circuits with the fewest SX+X gates, and out of these circuits, it selects the one with the highest NetLSD divergence to the original block circuit (Fig. 6). This ensures that OpaQue achieves a balance between gate count minimization and structural difference maximization.

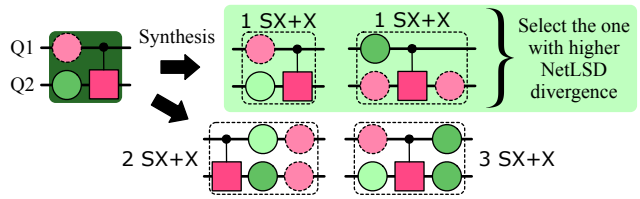


Figure 6: OpaQue selects the circuit with the lowest number of SX+X gates and the highest structural difference to the original circuit.

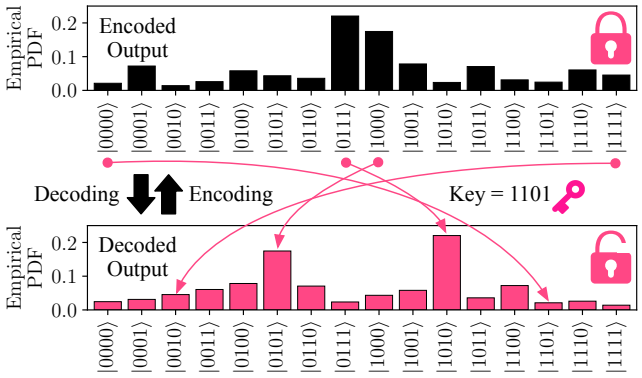


Figure 7: An illustration of how the probability distribution gets shifted during the encoding procedure and can be revered back using the key using the decoding procedure.

Fig. 4 (Step ❹) shows three aspects of note. First, the synthesized blocks have the same number of CX gates as the original blocks, but the gates are at different positions and have different orientations. Second, the synthesized blocks can have more or less one-qubit gates than the original circuit, and these gates can be in completely different positions structurally. Generally, the synthesized block has more one-qubit gates than the original block for stronger obfuscation, as we demonstrate in the evaluation (Sec. 4). Last, the figure shows the final fully encoded circuit is structurally completely different than the initial (original) circuit, with all traces of X-gate and RX-gate injections and block boundaries erased (as we evaluate in Sec. 4 using a structural distance metric). If an adversary were to separate any region of the circuit and inspect it, they would observe a completely different structure and output than they would have otherwise. In fact, it is not possible to perform one-to-one correspondence of any region in the original and encoded circuits due to gate injections and synthesis. Next, we discuss how the user can decode the output.

2.6 Output Recovery Using OpaQue

As Fig. 7 shows, OpaQue's decoding is a quick step that is performed on the user side using classical resources. As the user has access to the encoding key, they can simply flip the bits in the output states in accordance with the key. As the figure shows, the user receives the encoded output in which probabilities of all the states are scrambled. If we consider the two states with the highest probabilities, $|0111\rangle$ and $|1000\rangle$, and apply the key 1101, we get $|0111\rangle \rightarrow |1010\rangle$ and $|1000\rangle \rightarrow |0101\rangle$. Similarly, all the states can be decoded to get the complete output distribution.

Table 1: Quantum software benchmarks.

Algorithm	Description
ADD	Quantum Adder Circuit [10]
ADV	Google's Quantum Advantage Algorithm [3]
DNN	Deep Quantum Neutral Network [30]
HLF	Hidden Linear Function Circuit [8]
MULT	Quantum Multiplier Circuit [16]
QAOA	Quantum Alternating Operator Ansatz [11]
QFT	Quantum Fourier Transform [21]
SAT	Quantum Boolean Satisfiability Algorithm [31]
TFIM	Transverse Field Ising Model [6]
VQE	Variational Quantum Eigensolver [20]
WSTATE	W-State Preparation/Assessment Circuit [12]
XY	XY Quantum Heisenberg Model Circuit [6]

We note that OpaQue ensures that the obfuscated circuit maintains correctness. The synthesis process maintains functional equivalence, and the injected X gates only modify the final measurement outcomes in a controlled manner. As a result, the original quantum computation remains intact, and only the client can decode the correct output. On the other hand, an adversary aware of OpaQue's approach might attempt to isolate RX gates, but OpaQue mitigates this risk through (1) synthesis, where RX gates are absorbed into circuit logic, preventing straightforward reversal. And, (2) block-wise transformations, where two-qubit block synthesis generates alternative circuit representations, making recovery exponentially difficult. OpaQue's synthesis procedure also ensures that the injected OpaQue gates are merged within the circuit structure, making pattern-based attacks challenging. Further, because the hardware provider does not recieve the original circuit when OpaQue is employed, hardware-based side channel attacks for fingerprinting are also difficult.

That concludes the discussion of OpaQue's design. Next, we present its evaluation methodology.

3 OpaQue's Evaluation Methodology

Experimental Setup. We evaluate OpaQue using IBM's quantum computing cloud platform [9]. This platform provides support for ideal quantum simulation, noisy quantum simulation, and real hardware execution. To implement and evaluate OpaQue on the client side, we used Qiskit [2] (version 0.36.0), which is a Python-based (version 3.9.7) programming framework that implements a wide range of quantum computing features. We used Qiskit's Aer library (version 0.10.4) to convert blocks into unitary matrices for synthesis. The Qiskit transpiler was used to synthesize blocks into gate logic using KAK decomposition [17], convert circuits into IBM-compatible basis gates, and run optimization passes during compilation. We set the number of circuits generated for each block to select from to be 3 ($k = 3$) as we observe diminishing returns beyond that point. Qiskit converters were used to convert circuits into NetworkX [15] (version 2.6.3) graph representations, and the NetRD [19] library (version 0.3.0) was used to calculate distances between two graphs to assess the similarity of two circuits. All of the above steps were run entirely on the client side. To run quantum simulations and real-hardware executions on the IBM cloud, we used Qiskit's IBMQ Provider library (version 0.19.0).

Quantum Software Circuits and Benchmarks. Table 1 lists the quantum software benchmarks used to evaluate OpaQue, representing a wide variety of quantum algorithms in terms of algorithmic domain and circuit properties.

TFIM and XY are Hamiltonian evolution algorithms for material simulations. ADD and MULT carry out quantum arithmetic operations. HLF is a search algorithm to find likely solution states. QFT is a quantum benchmark with an equal output distribution for all states. ADV is the algorithm used to establish a quantum advantage over classical computing. WSTATE describes the entanglement property of an n-qubit system. DNN is a quantum neural network design. QAOA is a general variational optimization algorithm. SAT uses Grover's search technique to solve the boolean satisfiability problem. Lastly, VQE is the algorithm used to optimize for the ground state energy of a molecule group.

Real-System Quantum Cloud Platform. We used the IBM quantum cloud as the quantum hardware provider to perform our experimental campaign. **We used three IBM quantum computers for real-hardware algorithm executions chosen based on algorithm sizes.** We used the 127-qubit Washington computer (the largest available) to run circuits larger than 27 qubits. It was not possible to run the 128-qubit circuit as computers of that size are not yet available and larger circuits sizes would face a high degree of noise, making output uninterpretable. We used IBM's 27-qubit Toronto computer to run circuits of size 8-27 qubits. We used IBM's seven-qubit Lagos computer to perform real-hardware executions for circuits of size seven qubits or less. We ran 32,000 shots per circuit, as it is the maximum allowed. The four hardware-compatible basis gates on the IBM computers are CX, SX, X, and RZ. Therefore, all of our analysis is with respect to these gates. But, OpaQue is compatible with any other basis gate set.

We also used IBM's validated QASM simulator to perform ideal simulations (i.e., no noise) to produce the ideal circuit outputs – needed for comparison with the ground truth to demonstrate OpaQue's effectiveness. We ran 100,000 shots per circuit. This is the number of times a circuit is prepared, run, and measured to generate the output distribution. A circuit has to be run multiple times because each run only produces one output state (bit string). We simulate circuits up to 32 qubits in size, as it is not possible to simulate circuits beyond that size due to the exponential scaling requirements of simulating quantum algorithms. For larger algorithms, we show the circuit characteristics as they can be compiled and synthesized but not run or simulated.

Evaluation Metrics. To assess the quality of the output, we use two metrics: (a) **Total Variation Distance (TVD)**, and (b) **Dominant State Percentile**.

The **Total Variation Distance (TVD)** is used to quantify the difference between two output probability distributions. The TVD of two outputs is calculated as $\frac{1}{2}\sum_{k=1}^{2^n} |p_1(k) - p_2(k)|$, where $p_1(k)$ is the probability of state k in the first output and $p_2(k)$ is the probability of state k in the second output. The closer the TVD is to 1, the bigger the difference between the two output distributions; the closer it is to 0, the smaller the difference. The TVD is used to compare OpaQue's output to the baseline circuit's output and the ideal output.

The **Dominant State Percentile** is used to quantify how close a technique is to identifying the true dominant or winner solution state, i.e., the state with the highest probability. Many quantum algorithms have one solution that needs to be identified. If that state is the one with the highest probability, then it is in the 100^{th} percentile. The lower the percentile, the more difficult it is to identify the dominant state. However, a value of less than 100^{th} percentile, no matter how close it is to the 100^{th} percentile, indicates that the technique has failed to identify the dominant state.

To assess the structural similarities of any two circuits, we convert them into Directed Acyclic Graphs (DAGs) and use a widely-used graph distance metric, NetLSD, to examine their differences. The **NetLSD divergence** [5, 33] is calculated by computing the spectra of the normalized Laplacian matrices corresponding to the two DAGs and comparing their spectral node signature distributions, i.e., the heat signatures. A value greater than 10^2 indicates reasonably high dissimilarity [33].

We also consider and quantify circuit properties such as the **number of `CX` gates** (two-qubit gates), **the number of `SX+X` gates** (one-qubit physical gates), and **the number of `RZ` gates** (one-qubit virtual gates). These metrics help analyze how much the circuit size increases due to OpaQue's obfuscation process. We also look at the **circuit depth**, which is the number of `CX` gates in the circuit's critical path.

Comparisons. We compare OpaQue to the **baseline technique**, which performs *no output or structural obfuscation* but benefits from all the optimizations supported by the Qiskit compiler, including synthesis for gate count reduction. This allows us to compare OpaQue w.r.t. circuit characteristics (e.g., increase in the number of quantum gates and structure of the circuit) and solution quality (e.g., identification of dominant solution state, program output error in terms of TVD). We note that currently, there are no other solutions that provide obfuscation of quantum circuit outputs. We also compare OpaQue to the **uncorrected** output, i.e., when OpaQue's decoder is not deployed, to demonstrate the strength of OpaQue's obfuscation. We also investigate the strength of the two obfuscation techniques employed by OpaQue on their own by comparing to versions of OpaQue that only implement **circuit-end `X`-gate injections** or **throughout-circuit `RX`-gate injections**.

4 OpaQue's Evaluation and Analysis

> **RQ1.** *How well does OpaQue obfuscate the program output of the quantum circuits?*

We first analyze OpaQue's effectiveness to obfuscate the program output of the quantum circuits. Fig. 8 shows the total variation distance (TVD) between the output produced by the baseline circuit and the output produced by OpaQue's obfuscated circuit, but not corrected by the OpaQue decoder. The bars representing this TVD are labeled as "Uncorrected". Recall that the baseline circuit does not employ any obfuscation. If OpaQue is entirely ineffective, then the OpaQue without the decoder ("Uncorrected") bars would result in almost zero TVD – that is, despite the obfuscation, one does not need to decode the output as the uncorrected output is already the same as the baseline circuit without obfuscation. However, we

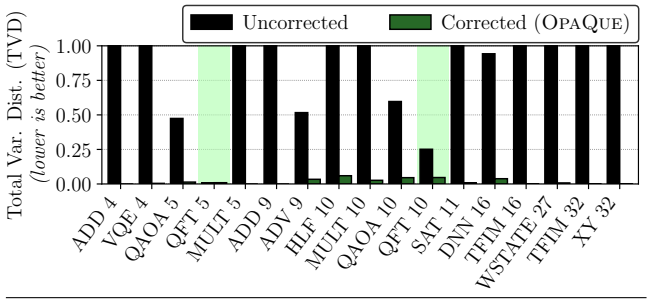


Figure 8: When OpaQue's output is not corrected, it has a high TVD relative to the baseline. When it is corrected using OpaQue's decoder, the TVD becomes negligible. QFT is the only exception due to its uniform output distribution. The numbers next to the algorithm names indicate the qubit count of the respective algorithm.

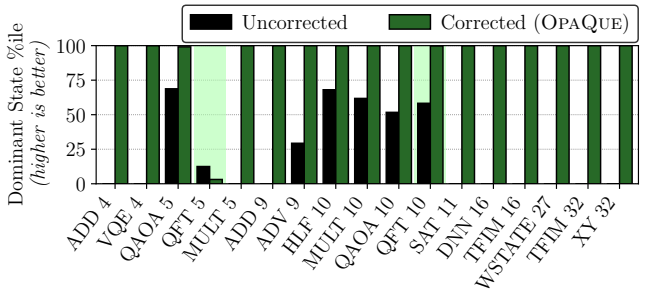


Figure 9: For algorithms with a dominant output state (all but QFT), the uncorrected output cannot identify it for any of the algorithms, while the output corrected with the OpaQue decoder can identify the dominant state for all algorithms.

observed this is indeed not the case. In most cases (Fig. 8), the TVD bars for the "Uncorrected" case are almost close to one, as desired.

Next, we observe that Fig. 8 shows another set of bars after the OpaQue decoder is applied, and the output is corrected on the client side(labeled as "Corrected"). Once the OpaQue decoder is applied, the TVD between OpaQue's output and the baseline output is reduced to a negligible amount (<0.05 across all algorithms). This indicates that the corrected OpaQue output is very similar to the original output, demonstrating the effectiveness of the OpaQue. As a note, two QFT circuits of different sizes show low TVD for the "Uncorrected" case because the QFT program scrambles the input and generates an equal probability for all states. In this case, even if OpaQue scrambles and shifts the probabilities of the output states, all states still have similar probabilities.

While the TVD results are promising, another important metric is the ability to identify the dominant state in the output of the quantum algorithm. Fig. 9 shows the percentile at which the actual dominant state (i.e., the dominant state in the output of the baseline circuit) falls in the uncorrected output as well as the corrected output. Ideally, this should be 100%. The uncorrected output is not able to identify the dominant state for any of the algorithms. In fact, across all algorithms, the dominant state falls in the 75^{th} percentile or less. For most algorithms, it is in the 0^{th} percentile. This means that the dominant state is not even close to being the highest probability state in the uncorrected output for any of the

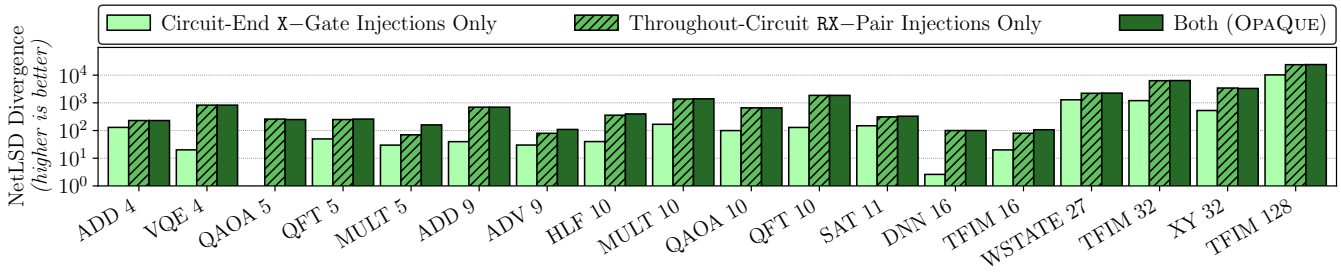


Figure 10: The comparison of the NetLSD divergence of the original circuit with the corresponding OpaQue-generated circuits shows that RX-pair injections help generate circuits that are structurally very different than the original circuit. On the other hand, the circuit-end X-gate injections help obfuscate the output but not the structure. Note the log scale.

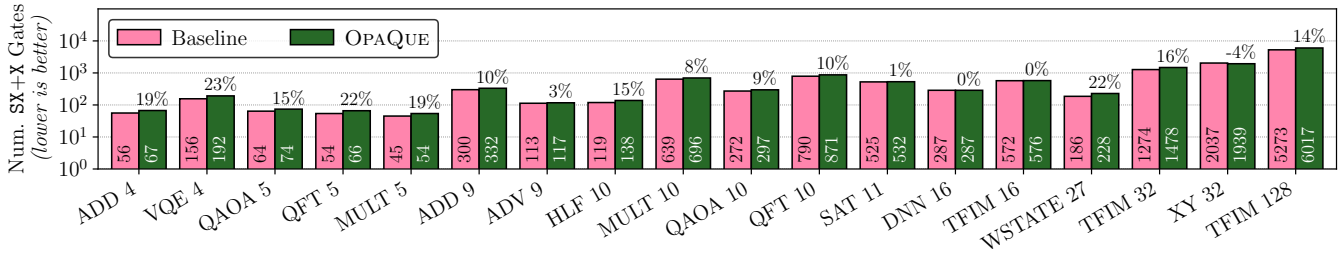


Figure 11: OpaQue increases the number of one-qubit physical gates (SX and X) by 13.6% on average for the purpose of structural and output obfuscation. It increases these gates as they have negligible noise effects as compared to CX gates. The numbers inside the bars indicate the raw gate counts for the respective techniques, and the numbers atop OpaQue's bars indicate the percent increase in OpaQue's counts relative to the baseline. Note the log scale.

algorithms. Thus, an adversary with access to the obfuscated output cannot identify the dominant state of the algorithm without the key to decode the output. On the other hand, when the output is corrected by the OpaQue decoder, the dominant state is identified by OpaQue across all algorithms with a dominant state (which are all algorithms except for QFT). Recall that in the case of QFT, while there is no dominant state in the ideal scenario because the simulation has statistical variability and generates one state with a slightly higher probability than others in the baseline circuit output, the results simply indicate the ability of OpaQue to match that state.

Answer to RQ1. *OpaQue successfully obfuscates the output probability distribution and the dominant state from an adversary. Moreover, it can also recover the output locally on the user's end when it is returned by the hardware provider and decoded using the key.*

RQ2. *Does OpaQue produce a significantly different quantum circuit structure than the original quantum circuit provided by the user?*

The purpose of this evaluation is to show that OpaQue produces a significantly different circuit structure than the original circuit with the help of RZ-gate pair injection.

Fig. 10 shows the NetLSD divergence metric to compare the structures of the OpaQue-generated circuits to the baseline circuits. The figure also shows the distance of the baseline circuit to the circuit with only synthesized X-gate injections at the end, the circuit with only synthesized RZ-gate injections throughout the circuit, and the combined effect (OpaQue).

The circuit-end X-gate injections obfuscate the output effectively, but we observe that the circuit-end X-gate injections have a limited impact on obfuscating the circuit structure (Fig. 10). This is because synthesizing the X gates into the circuit only modifies the circuit structure at the end of the circuit, but the rest of the circuit structure would remain the same as the baseline (original) circuit. Thus, while X gate injection is an effective technique to obfuscate the output, it is not sufficient to obfuscate the circuit structure – motivating our design element to inject RX gates throughout the circuit. Indeed, injecting the RX gates throughout the circuit, dividing them into disjoint blocks, and synthesizing the blocks show promising results. Recall: injecting only the RX gates does obfuscate the output; RX gates are injected in pairs to ensure that the program's intent remains intact.

Fig. 10 shows that for all of the algorithms, the structural obfuscation that injecting the RX gates provides is orders of magnitudes greater than structural obfuscation facilitated by the circuit-end X-gate injections. As an instance, for the MULT 10 algorithm, the NetLSD divergence with only circuit-end X-gate injections is 1.7×10^2, with only throughout-circuit RX-gate injections is 1.4×10^3, and with both (OpaQue) is also 1.4×10^3. The trend is similar for other algorithms; they all have a divergence of $> 10^2$ with OpaQue.

Answer to RQ2. *Our experimental results confirm that OpaQue's combination of X-gate and RX-pair injection is effective to achieve both desired outcomes: output obfuscation and structural obfuscation (producing the NetLSD divergence of $> 10^2$).*

RQ3. *Does OpaQue increase the count of quantum gates and impact the program output?*

Table 2: Compilation times of the algorithms in seconds.

Algorithm	Baseline	OpaQue	Algorithm	Baseline	OpaQue
ADD 4	2.4	4.1	QAOA 10	6.3	12.0
VQE 4	4.0	8.0	QFT 10	15.0	32.8
QAOA 5	2.4	4.0	SAT 11	8.6	22.9
QFT 5	2.4	3.9	DNN 16	6.1	11.4
MULT 5	2.2	3.2	TFIM 16	10.0	20.6
ADD 9	6.9	13.5	WSTATE 27	5.3	10.3
ADV 9	3.3	8.0	TFIM 32	27.6	54.9
HLF 10	3.7	7.0	XY 32	33.3	93.2
MULT 10	11.7	24.1	TFIM 128	80.0	230.4

OpaQue injects different types of gates to achieve program output obfuscation. Unfortunately, such injections can increase the gate count and circuit depth, which can have side effects on the solution quality produced by OpaQue on real noisy quantum hardware (i.e., OpaQue's circuit producing higher TVD than the original unobfuscated circuit).

First, we compare the number of CX gates in the OpaQue-generated circuits and the corresponding baseline circuits. We do not plot this result as *across all algorithms, the number of* CX *gates remains the exact same as the baseline circuit.* This is by design, as OpaQue ensures during the block synthesis process that the number of CX gates are not increased to avoid the impact of their high error rates on the program output on noisy quantum computers. This enables OpaQue not to increase the output error, as we see in the next subsection. Note also that because of the CX count remaining the same and the synthesis procedure only being applied to two-qubit blocks, OpaQue also does not incur any increase in circuit depth (number of CX gates on the critical path). We do not show this result to avoid repetition, as all algorithms incur a 0% increase in circuit depth.

Second, we compare the number of SX+X gates in the OpaQue-generated circuits to the number of SX+X gates in the baseline circuits in Fig. 11. Even though these two one-qubit gates are different in terms of their impact on the quantum state, we group the two together because they have the same error rates as far as the hardware noise effects are concerned. On average, the number of SX+X gates is increased by 13.6%, and the number of RZ gates is increased by 11.6% (not shown for brevity; RZ gates are virtual gates and have no contribution to the error effects). While the increases are non-negligible, they do not have much impact on the output noise – as we will observe in the next part of the evaluation where we demonstrate real-hardware results – due to the little-to-none error impact of the one-qubit gates.

Answer to RQ3. *OpaQue does not increase the two-qubit CX gate count (by design) – CX gates have the highest error rate on real machines and, hence, impact the program output on real machines. On average, the number of single-qubit* SX+X *gates is increased by 13.6%, but as shown next, this magnitude of increase has negligible impact on the program output.*

RQ4. *What is the compilation overhead of OpaQue?*

Analytically examining the compilation overhead and complexity of OpaQue is challenging, as the overhead of synthesis can vary from algorithm to algorithm based on the intricacies of the circuit structure. Nonetheless, we empirically analyze the compilation overhead of OpaQue. Table 2 shows the compilation times of all the algorithms for the baseline circuit and OpaQue-generated circuit. The table shows that the compilation times with OpaQue increase by 2 × on average over the baseline compilation times. These increases are negligible compared to the queue wait times on quantum cloud, which tend to be in the order of hours [26]. We also note that decoding in OpaQue is a simple classical bit-flipping operation that does not introduce meaningful computational overhead. In the worst case, this would involve flipping the qubits of each shot if each shot produces a different state. Thus, it scales linearly with the number of shots.

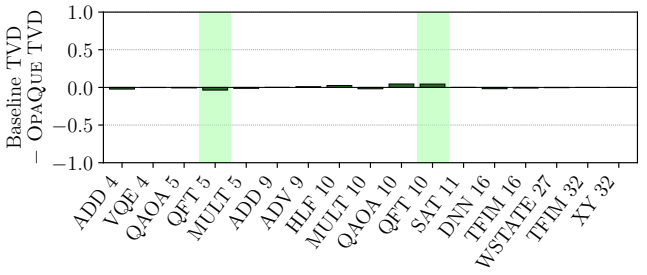


Figure 12: Execution on real quantum hardware shows that OpaQue's corrected output is able to achieve a similar TVD to the ideal output as the baseline output.

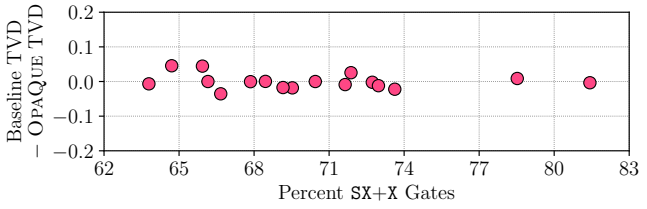


Figure 13: The TVD difference between the baseline and OpaQue outputs does not increase with increase in the percent of SX+X gates in the circuit (relative to SX+X+CX).

Answer to RQ4. *OpaQue's compilation overhead is low and practical – often orders of magnitude lower than the queue wait times on quantum cloud platforms.*

RQ5. *Is OpaQue effective on real quantum machines?*

We now examine OpaQue's performance on real quantum computers for algorithms up to 32 qubits. The noise levels only impact the output and not the circuit structure; therefore, we only analyze the output metrics.

Fig. 12 shows the difference between the TVD of the baseline output on noisy IBM computers (relative to the ideal simulation output) and the TVD of OpaQue's output on noisy IBM computers (also relative to the ideal simulation output). When this metric is positive, it indicates that the baseline has a higher error, and when it is negative, it indicates that OpaQue has a lower error. The figure shows that both TVDs are largely similar (maximum difference of 0.05 or 5%). That is, the hardware noise similarly impacts both techniques – this is primarily due to OpaQue's design element that enforces the CX gate count in the obfuscated circuit to be the same as the baseline circuit. In fact, as Fig. 13 shows, the difference between the baseline's TVD and OpaQue's TVD remains low even if SX+X

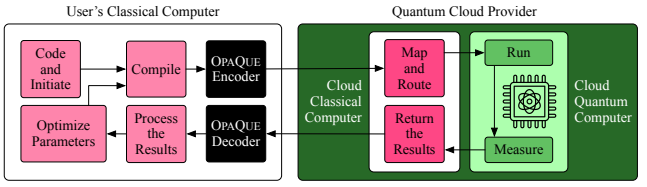


Figure 14: Variational quantum algorithms like QAOA follow a hybrid quantum-classical approach to iteratively optimize the parameters of a circuit to achieve a specific objective. OpaQue works to obfuscate each iteration.

gates form a large percent (> 75%) of all the physical gates in the circuit (each point in the figure corresponds to one real-hardware-evaluated algorithm). Thus, the additional one-qubit physical gates added by OpaQue do not have a major impact on the output error in noisy environments, as CX gates largely dominate the errors.

We also note that due to the high noise level, the baseline output is also not able to identify the dominant state correctly in many cases. Nonetheless, OpaQue is able to identify the dominant state whenever the baseline does (not shown in a figure). These results demonstrate how OpaQue successfully scrambles the output and decodes it in a high-noise setting.

Answer to RQ5. *OpaQue's remains effective on real quantum hardware and does not increase the output error – OpaQue's corrected output is able to achieve a similar TVD to the ideal output as the baseline output.*

4.1 Case Study: Iterative Optimization of a Variational Algorithm using OpaQue

We now perform a case study of a variational algorithm using the 4-qubit QAOA MaxCut problem. In a variational algorithm, the circuit gate angles are treated as parameters, and these parameters are tuned to optimize for a specific objective. Thus, many machine learning or optimization problems fall under the domain of variational algorithms [20]. We perform this case study for a variational algorithm because it involves iteratively running the same parameterized circuit with different parameters. Thus, it can be argued that the optimizer can converge to a sub-optimal solution or take longer to converge if OpaQue obfuscates the circuit for each iteration. Our experimental results demonstrate that OpaQue continues to be effective for iterative variational algorithms – the optimization quality and time to convergence are not impacted by OpaQue's obfuscation.

Fig. 14 shows the iterative structure that a variational algorithm follows with OpaQue. Before shipping out the quantum circuit to the cloud for execution at each iteration, OpaQue decodes the resulting output from the previous iteration before the output is fed to the optimizer, and then, OpaQue obfuscates the circuit in preparation for the next round of optimization. We focus on the 4-qubit QAOA MaxCut problem, as QAOA provides a specific circuit template suitable for many variational problems. The MaxCut problem involves dividing the nodes of a graph into two sets such that the number of edges between the two sets is maximized.

In our case, the graph has four nodes connected to form a rectangle. Therefore, the problem has two solutions, "1010" and "0101",

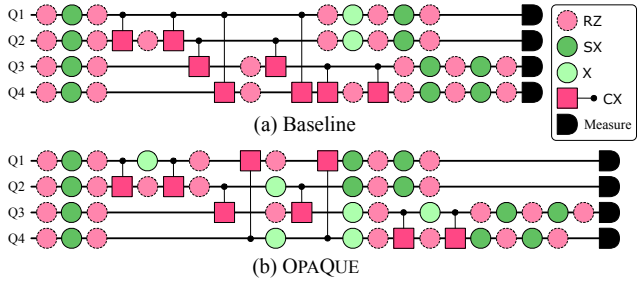


Figure 15: Visual circuit structure comparison of the baseline circuit and the OpaQue-generated circuit for the last iteration shows how different the two circuits are with all traces of X- and RX-gate injections being erased.

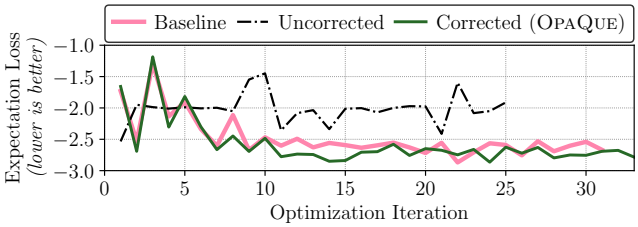


Figure 16: The loss curve with OpaQue's corrected output traces the baseline loss curve. However, when the output is not corrected, the variational algorithm cannot be optimized due to the scrambled results.

where 0 and 1 are labels for the two sets, and the bitstring indicates the set that the nodes belong to in order from 1 to 4. The goal of the optimizer is to adjust the parameters such that these two solutions have the highest probabilities and can be successfully identified. We chose this small problem so that we can investigate it visually.

We first visually examine how the OpaQue generated circuit differs from the baseline circuit in this case. Fig. 15 shows the two circuits during the last optimization iteration. Note also that the circuit structure for each iteration varies because the synthesis step generates different circuit structures (the parameterized angles vary from one iteration to another). Thus, there is no concern of an adversary being able to infer information about the algorithm – the concern would be present if the circuit structure remains the same throughout iterations. In the figure, we only show the last-iteration circuits for brevity. The two circuits look substantially different (NetLSD is 2.3×10^2) in terms of the number of gates and the gate placement within the circuits. In addition, the RZ gates in both circuits have very different angles (not shown here for clarity). Thus, *the structure is successfully obfuscated.*

Next, we assess the success of OpaQue's output obfuscation when run on a real quantum computer (IBM Lagos). Fig. 16 shows the loss curves over the optimization iterations for three cases: baseline (no obfuscation), uncorrected (OpaQue encoder is applied, but the decoder is not, i.e., the perspective of the adversary), and corrected (both OpaQue encoder and decoder are applied). We make several observations. First, we observe that the loss curve of OpaQue closely follows the loss curve of the baseline circuit. This means that while OpaQue successfully obfuscates every iteration, it is still able to optimize variational algorithms just as efficiently and

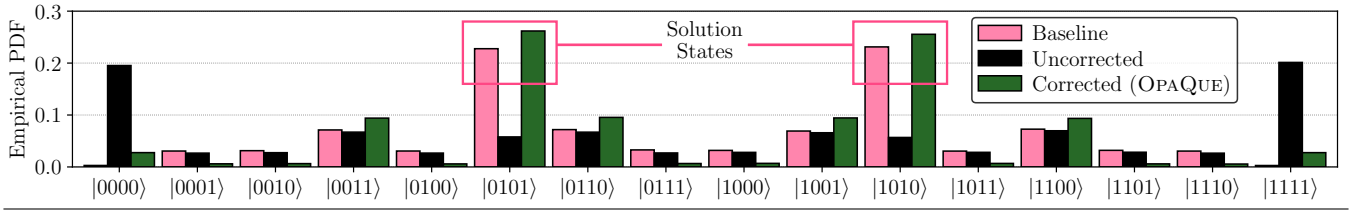


Figure 17: The output corrected with OpaQue's decoder can achieve an output distribution similar to the original circuit's baseline output. It is thus able to identify the two solution states for the QAOA MaxCut problem with the dominant probabilities. The obfuscated uncorrected output is not able to identify these states.

effectively as the baseline technique – in other words, the solution quality and time to convergence are not affected.

Second, when the output is not corrected with the OpaQue decoder, but the circuit remains obfuscated, the optimizer is not able to make any progress toward minimizing the loss as each iteration generates scrambled outputs of different types, which cannot be used toward the optimization of a specific objective. This indicates that the cloud provider, which has access to only uncorrected output, cannot make forward progress in an iterative optimization problem. Third, Fig. 17 shows that OpaQue maintains the solution quality. The corrected OpaQue output distribution over all the states closely resembles the baseline output distribution. Both techniques are able to identify the two solution states as they have the dominant probabilities. On the other hand, the circuit optimized using the uncorrected output has a completely different distribution, where the two states with the highest probabilities are non-solution states. Thus, *an adversary that tries to optimize this problem using the uncorrected output will neither be able to identify the original circuit structure nor the correct solution. This demonstrates the applicability of OpaQue to variational algorithms.*

5 Conclusion

This work introduces OpaQue to protect quantum code and output from adversarial snooping in the quantum cloud. OpaQue leverages circuit-end X-gate injections for output obfuscation and throughout-circuit RX-gate injections for structural obfuscation. Through extensive simulations and real-hardware evaluations, we demonstrate the effectiveness of OpaQue in obfuscating circuit structure and output while maintaining the output fidelity of the algorithm.

Data Availability: https://zenodo.org/doi/10.5281/zenodo.10896069.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable and insightful feedback. This work was supported by Northeastern University and NSF Awards 1910601, 2124897. This work was also supported by Rice University, the Rice University George R. Brown School of Engineering and Computing, and the Rice University Department of Computer Science. This work was supported by the DOE Quantum Testbed Finder Award DE-SC0024301. This work was also supported by the Ken Kennedy Institute and Rice Quantum Initiative, which is part of the Smalley-Curl Institute.

References

[1] 2021. IBM Quantum. https://quantum-computing.ibm.com/

[2] Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, David Bucher, Francisco Jose Cabrera-Hernández, Jorge Carballo-Franquis, Adrian Chen, Chun-Fu Chen, Jerry M. Chow, Antonio D. Córcoles-Gonzales, Abigail J. Cross, Andrew Cross, Juan Cruz-Benito, Chris Culver, Salvador De La Puente González, Enrique De La Torre, Delton Ding, Eugene Dumitrescu, Ivan Duran, Pieter Eendebak, Mark Everitt, Ismael Faro Sertage, Albert Frisch, Andreas Fuhrer, Jay Gambetta, Borja Godoy Gago, Juan Gomez-Mosquera, Donny Greenberg, Ikko Hamamura, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Hiroshi Horii, Shaohan Hu, Takashi Imamichi, Toshinari Itoko, Ali Javadi-Abhari, Naoki Kanazawa, Anton Karazeev, Kevin Krsulich, Peng Liu, Yang Luh, Yunho Maeng, Manoel Marques, Francisco Jose Martín-Fernández, Douglas T. McClure, David McKay, Srujan Meesala, Antonio Mezzacapo, Nikolaj Moll, Diego Moreda Rodríguez, Giacomo Nannicini, Paul Nation, Pauline Ollitrault, Lee James O'Riordan, Hanhee Paik, Jesús Pérez, Anna Phan, Marco Pistoia, Viktor Prutyanov, Max Reuter, Julia Rice, Abdón Rodríguez Davila, Raymond Harry Putra Rudy, Mingi Ryu, Ninad Sathaye, Chris Schnabel, Eddie Schoute, Kanav Setia, Yunong Shi, Adenilton Silva, Yukio Siraichi, Seyon Sivarajah, John A. Smolin, Mathias Soeken, Hitomi Takahashi, Ivano Tavernelli, Charles Taylor, Pete Taylour, Kenso Trabing, Matthew Treinish, Wes Turner, Desiree Vogt-Lee, Christophe Vuillot, Jonathan A. Wildstrom, Jessica Wilson, Erick Winston, Christopher Wood, Stephen Wood, Stefan Wörner, Ismail Yunus Akhalwaya, and Christa Zoufal. 2019. *Qiskit: An Open-source Framework for Quantum Computing*. https://doi.org/10.5281/zenodo.2562111

[3] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. 2019. Quantum Supremacy Using a Programmable Superconducting Processor. *Nature* 574, 7779 (2019), 505–510. https://doi.org/10.1038/s41586-019-1666-5

[4] Ashwin Balaji and Sanjay Kumar Dhurandher. 2022. Reliable Data Communication using Post-Quantum Encryption in Internet of Everything. *International Journal of Communication Systems* 35, 13 (2022), e5246.

[5] Anirban Banerjee. 2012. Structural Distance and Evolutionary Relationship of Networks. *Biosystems* 107, 3 (2012), 186–196.

[6] Lindsay Bassman, Connor Powers, and Wibe A de Jong. 2021. ArQTiC: A Full-Stack Software Package for Simulating Materials on Quantum Computers. *arXiv preprint arXiv:2106.04749* (2021).

[7] Francesco Bova, Avi Goldfarb, and Roger G Melko. 2021. Commercial Applications of Quantum Computing. *EPJ quantum technology* 8, 1 (2021), 2.

[8] Sergey Bravyi, David Gosset, and Robert König. 2018. Quantum Advantage with Shallow Circuits. *Science* 362, 6412 (2018), 308–311.

[9] Davide Castelvecchi. 2017. IBM's Quantum Cloud Computer Goes Commercial. *Nature News* 543, 7644 (2017), 159.

[10] Steven A Cuccaro, Thomas G Draper, Samuel A Kutin, and David Petrie Moulton. 2004. A New Quantum Ripple-Carry Addition Circuit. *arXiv preprint quant-ph/0410184* (2004).

[11] Edward Farhi and Aram W Harrow. 2016. Quantum Supremacy through the Quantum Approximate Optimization Algorithm. *arXiv preprint arXiv:1602.07674* (2016).

[12] Michael Fleischhauer and Mikhail D Lukin. 2002. Quantum Memory for Photons: Dark-State Polaritons. *Physical Review A* 65, 2 (2002), 022314.

[13] Elizabeth Gibney. 2017. Billion-Euro Quantum Project Takes Shape. *Nature* 545, 7652 (2017), 16.

[14] Elizabeth Gibney. 2019. The Quantum Gold Rush. *Nature* 574, 7776 (2019), 22–24.

[15] Aric Hagberg and Drew Conway. 2020. NetworkX: Network Analysis with Python. *URL: https://networkx.github.io* (2020).

[16] Andrew Hancock, Austin Garcia, Jacob Shedenhelm, Jordan Cowen, and Calista Carey. 2019. Cirq: A Python Framework for Creating, Editing, and Invoking Quantum Circuits. *URL https://github.com/quantumlib/Cirq* (2019).

[17] Navin Khaneja and Steffen J Glaser. 2001. Cartan Decomposition of SU (2n) and Control of Spin systems. *Chemical Physics* 267, 1-3 (2001), 11–23.

[18] Gushu Li, Anbang Wu, Yunong Shi, Ali Javadi-Abhari, Yufei Ding, and Yuan Xie. 2022. Paulihedral: A Generalized Block-wise Compiler Optimization Framework for Quantum Simulation Kernels. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 554–569.

[19] Stefan McCabe, Leo Torres, Timothy LaRock, Syed Arefinul Haque, Chia-Hung Yang, Harrison Hartle, and Brennan Klein. 2020. NetRD: A Library for Network Reconstruction and Graph Distances. *arXiv preprint arXiv:2010.16019* (2020).

[20] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. 2016. The Theory of Variational Hybrid Quantum-Classical Algorithms. *New Journal of Physics* 18, 2 (2016), 023023.

[21] Victor Namias. 1980. The Fractional order Fourier Transform and its Application to Quantum Mechanics. *IMA Journal of Applied Mathematics* 25, 3 (1980), 241–265.

[22] Matteo Paltenghi and Michael Pradel. 2023. MorphQ: Metamorphic testing of the Qiskit quantum computing platform. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2413–2424.

[23] Tirthak Patel, Ed Younis, Costin Iancu, Wibe de Jong, and Devesh Tiwari. 2022. QUEST: Systematically Approximating Quantum Circuits for Higher Output Fidelity. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 514–528.

[24] Koustubh Phalak, Abdullah Ash-Saki, Mahabubul Alam, Rasit Onur Topaloglu, and Swaroop Ghosh. 2021. Quantum PUF for Security and Trust in Quantum Computing. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 11, 2 (2021), 333–342.

[25] John Preskill. 2021. Quantum Computing 40 Years Later. *arXiv preprint arXiv:2106.10522* (2021).

[26] Gokul Subramanian Ravi, Kaitlin N Smith, Pranav Gokhale, and Frederic T Chong. 2021. Quantum Computing in the Cloud: Analyzing Job and Machine Characteristics. In *2021 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 39–50.

[27] Michael G Raymer and Christopher Monroe. 2019. The US National Quantum Initiative. *Quantum Science and Technology* 4, 2 (2019), 020504.

[28] Abdullah Ash Saki, Mahabubul Alam, Koustubh Phalak, Aakarshitha Suresh, Rasit Onur Topaloglu, and Swaroop Ghosh. 2021. A Survey and Tutorial on Security and Resilience of Quantum Computing. In *2021 IEEE European Test Symposium (ETS)*. IEEE, 1–10.

[29] Abdullah Ash Saki, Aakarshitha Suresh, Rasit Onur Topaloglu, and Swaroop Ghosh. 2021. Split Compilation for Security of Quantum Circuits. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–7.

[30] Samuel A Stein, Betis Baheri, Daniel Chen, Ying Mao, Qiang Guan, Ang Li, Shuai Xu, and Caiwen Ding. 2022. QuClassi: A Hybrid Deep Neural Network Architecture based on Quantum State Fidelity. *Proceedings of Machine Learning and Systems* 4 (2022).

[31] Juexiao Su, Tianheng Tu, and Lei He. 2016. A Quantum Annealing Approach for Boolean Satisfiability Problem. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[32] Aakarshitha Suresh, Abdullah Ash Saki, Mahababul Alam, Rasit Onur Topaloglu, and Swaroop Ghosh. 2021. A Quantum Circuit Obfuscation Methodology for Security and Privacy. In *Workshop on Hardware and Architectural Support for Security and Privacy*. 1–5.

[33] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander Bronstein, and Emmanuel Müller. 2018. NetLSD: Hearing the Shape of a Graph. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2347–2356.

[34] Jiyuan Wang, Qian Zhang, Guoqing Harry Xu, and Miryung Kim. 2021. Qdiff: Differential testing of quantum software stacks. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 692–704.

[35] Dongyang Xu, Lei Liu, Ning Zhang, Mianxiong Dong, Victor CM Leung, and James A Ritcey. 2023. Nested Hash Access with Post Quantum Encryption for Mission-Critical IoT Communications. *IEEE Internet of Things Journal* (2023).

[36] Jiaming Ye, Shangzhou Xia, Fuyuan Zhang, Paolo Arcaini, Lei Ma, Jianjun Zhao, and Fuyuki Ishikawa. 2023. QuraTest: Integrating quantum specific features in quantum program testing. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1149–1161.

[37] Yuanjing Zhang, Tao Shang, Ranyiliu Chen, and Jianwei Liu. 2022. Instantiation of Quantum Point Obfuscation. *Quantum Information Processing* 21 (2022), 1–16.