Optimizing Quantum Circuits, Fast and Slow

Amanda Xu
University of Wisconsin-Madison
Madison, WI, USA
axu44@wisc.edu

Abtin Molavi
University of Wisconsin-Madison
Madison, WI, USA
amolavi@wisc.edu

Swamit Tannu
University of Wisconsin-Madison
Madison, WI, USA
swamit@cs.wisc.edu

Aws Albarghouthi
University of Wisconsin-Madison
Madison, WI, USA
aws@cs.wisc.edu

Abstract

Optimizing quantum circuits is critical: the number of quantum operations needs to be minimized for a successful evaluation of a circuit on a quantum processor. In this paper we unify two disparate ideas for optimizing quantum circuits, *rewrite rules*, which are fast standard optimizer passes, and *unitary synthesis*, which is slow, requiring a search through the space of circuits. We present a clean, unifying framework for thinking of rewriting and resynthesis as abstract circuit transformations. We then present a radically simple algorithm, GUOQ, for optimizing quantum circuits that exploits the synergies of rewriting and resynthesis. Our extensive evaluation demonstrates the ability of GUOQ to strongly outperform existing optimizers on a wide range of benchmarks.

CCS Concepts: **• Software and its engineering → Compilers**; **• Hardware → Quantum computation**.

Keywords: quantum-circuit optimization; unitary synthesis; superoptimization; quantum computing

ACM Reference Format:
Amanda Xu, Abtin Molavi, Swamit Tannu, and Aws Albarghouthi. 2025. Optimizing Quantum Circuits, Fast and Slow. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '25), March 30–April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3669940.3707240

1 Introduction

Quantum computing enables efficient simulation of quantum mechanical phenomena, promising to catalyze advances in quantum physics, chemistry, materials science, and beyond. Near-term quantum computers with more than a thousand qubits operating in a noisy environment without error

ASPLOS '25, March 30–April 3, 2025, Rotterdam, Netherlands
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0698-1/25/03
https://doi.org/10.1145/3669940.3707240

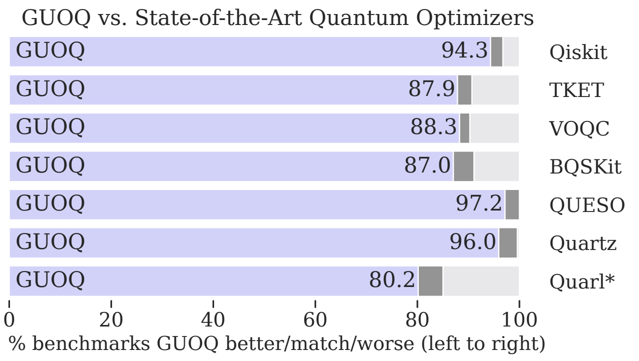


Figure 1. Summary of GUOQ compared to state-of-the-art on 2-qubit-gate reduction for the IBMQ20 gate set. GUOQ and BQSKit are allowed to approximate the circuit up to $\epsilon = 10^{-8}$. *Quarl requires an NVIDIA A100 (40GB) GPU to run.

correction have already been deployed, marking the current era of *Noisy Intermediate Scale Quantum* (NISQ) computing [49]. Recent groundbreaking experiments have implemented error-corrected *logical* qubits and demonstrated potential for reducing *logical error* [7, 43]. Although many challenges remain, *fault-tolerant quantum computing* (FTQC) is on the horizon.

In both NISQ and FTQC, reducing errors is a critical obstacle to overcome. Every quantum operation has a probability of failure causing a quantum execution to quickly devolve into random noise. The NISQ paradigm aims to mitigate these errors in the absence of error correction primarily by reducing the number of operations. However, error correction in FTQC is not a panacea and introduces its own unique bottlenecks [9, 59], which can render the error correction scheme useless if left untamed. Especially in the near term, FTQC architectures may face challenges in handling large circuit depths due to physical imperfections such as two-level system (TLS) drift, qubit leakage, high-energy particle strikes, and crosstalk [1, 7, 38]. Therefore, it is of utmost importance to reduce the number of operations for FTQC as well.

Current approaches tackling quantum circuit optimization primarily apply *peephole optimization* using a fixed

	Rewrite rules	Resynthesis
Fast	✓	✗
Limited by # gates	✓	✗
Limited by # qubits	✗	✓
Approximate	✗	✓

Table 1. Characteristics of rewrite rules and resynthesis.

set of *rewrite rules*. Some tools use a small set of hand-crafted rules [20, 29, 40], while others automatically synthesize rules [67, 68]. The idea is to apply rewrite rules in a schedule, transforming subcircuits to semantically equivalent ones with fewer operations. *Rewrite rules are fast* to apply—match a pattern and rewrite it—but inherently only perform local optimizations.

An orthogonal line of work has been studying the problem of *unitary synthesis*. A unitary matrix precisely represents the semantics of a quantum program. Some quantum algorithms are simple to state in the form of a unitary but nontrivial to decompose into elementary operations that can be executed on hardware [15, 18]. Thus, a large body of work has focused on synthesizing quantum circuits that implement a given unitary matrix [4, 13, 26, 44, 51, 52, 60, 63, 70]. Recent works [45, 66] have applied these algorithms to optimize quantum circuits by partitioning large circuits into manageably-sized *subcircuits* consisting of a few qubits at most and then *resynthesizing* each subcircuit to produce a new subcircuit whose unitary is equivalent, or close enough, to the original subcircuit's unitary. *Unitary synthesis is slow*: usually a combinatorial search problem through the space of circuits, but can apply to deep subcircuits.

Rewrite rules and resynthesis have their own strengths and weaknesses—see Table 1. Individual rewrite rules are fast to apply but are limited to small patterns with few gates. On the other hand, resynthesis is slow but can support circuits with an arbitrary number of gates because the primary limiting factor is the number of qubits. Critically, resynthesis has the power to optimize deep subcircuits and it can *approximate* the solution to some degree. Rewrite rules are too small and rigid to find meaningful approximations. Approximations can unlock new circuit optimizations but must be applied carefully to avoid introducing too much error.

Inspired by how humans combine fast and slow modes of thinking [25], *Systems 1 and 2*, we ask the following question:

Can we design an optimization approach that can synergistically combine the powers of optimizing quantum circuits fast, via rewrite rules, and slow, via resynthesis?

We answer this question affirmatively: we demonstrate that we can unify rewriting and resynthesis and propose a simple optimization algorithm that significantly outperforms existing approaches in the literature.

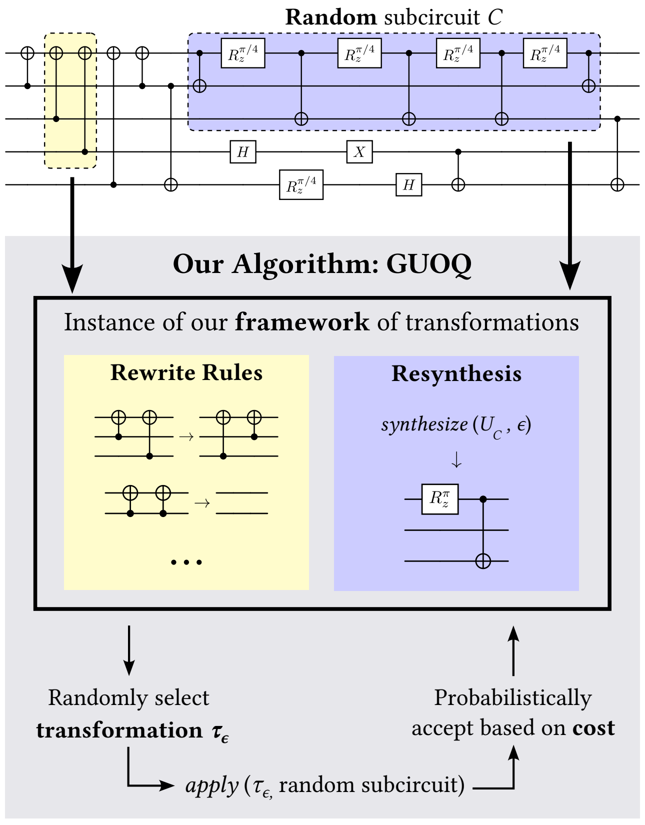
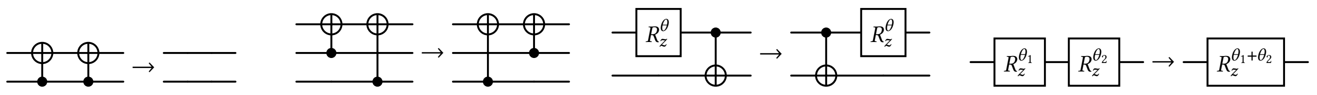

Random subcircuit C
$R_z^{\pi/4}$
H
X
R_z^{π}
Our Algorithm: GUOQ
Instance of our **framework** of transformations
Rewrite Rules
Resynthesis
synthesize (U_C, ϵ)
Randomly select **transformation** τ_ϵ
Probabilistically accept based on **cost**
apply (τ_ϵ, random subcircuit)

Figure 2. Overview of our approach

Our approach. We propose a framework to unify rewrite rules and resynthesis for optimizing quantum circuits (see Fig. 2). The key insight is that we can abstract both optimizations into a closed-box circuit *transformation* with a degree of approximation ϵ. Our flexible and generic framework allows arbitrary transformations, which can be applied freely. We prove a simple additive upper bound on the final approximation after applying a sequence of transformations.

We can instantiate our framework using a set of circuit transformations. The key challenge is deciding in what order to apply these transformations—this is the *phase ordering problem* that has plagued optimizing compilers for decades! We discover that, perhaps surprisingly, a simple and lightweight *simulated annealing*-like algorithm is the most effective solution, outperforming more clever heuristics or algorithms. The lack of structure in our problem lends itself to an approach that randomly and quickly alternates between fast and slow optimization, as opposed to sophisticated approaches guided by hand-crafted heuristics or reinforcement learning. Let this serve as another bitter lesson [62] that simple methods often prevail.

We implement our algorithm, GUOQ (Good Unified Optimizations for Quantum), and provide an extensive evaluation against state-of-the-art optimizers and *superoptimizers* using



(a) CX cancel **(b)** CX commute **(c)** R_z^θ commute CX control **(d)** Basic R_z^θ merge

Figure 3. Examples of rewrite rules. Observe how the rules with R_z^θ use *symbolic* θ angles.

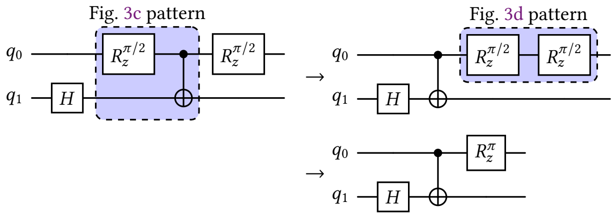


Figure 4. Example of applying the rule from Fig. 3c followed by the rule from Fig. 3d.

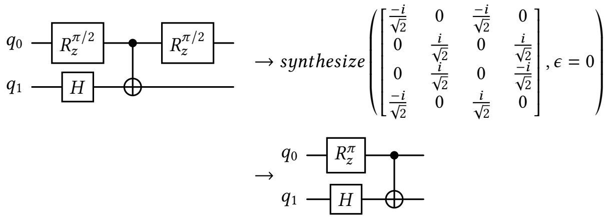

q_0
q_1
H
$R_z^{\pi/2}$
R_z^{π}
$\rightarrow$ *synthesize* $\left(\begin{bmatrix} \frac{-i}{\sqrt{2}} & 0 & \frac{-i}{\sqrt{2}} & 0 \\ 0 & \frac{i}{\sqrt{2}} & 0 & \frac{i}{\sqrt{2}} \\ 0 & \frac{i}{\sqrt{2}} & 0 & \frac{-i}{\sqrt{2}} \\ \frac{-i}{\sqrt{2}} & 0 & \frac{i}{\sqrt{2}} & 0 \end{bmatrix}, \epsilon = 0\right)$

Figure 5. Example of resynthesizing the initial Fig. 4 circuit.

a benchmark suite with 247 diverse quantum circuits implementing near- and long-term algorithms. Our evaluation demonstrates the following: (1) GUOQ significantly outperforms state-of-the-art tools (see Fig. 1 for a summary), (2) GUOQ's randomized search approach is critical for efficiently combining rewriting and resynthesis, and (3) GUOQ can flexibly perform well in the NISQ and FTQC regimes. For instance, GUOQ outperforms the recent superoptimizer, Quarl [32], in terms of 2-qubit-gate reduction on 80% of the benchmarks. This is despite the fact that Quarl uses a specialized deep reinforcement learning technique for quantum-circuit optimization, requires more computational resources (GPUs), and has been trained on portions of the benchmark suite.

Contributions. We make the following contributions:

- A framework that abstracts the inner workings of rewriting and resynthesis into closed-box circuit *transformations* with *approximate* semantic guarantees.
- A lightweight algorithm, GUOQ, inspired by simulated annealing, that searches the space of transformations.
- An implementation of GUOQ and thorough evaluation considering both NISQ and FTQC that demonstrates its effectiveness compared to state-of-the-art optimizers.

2 Background and Overview

In this section, we provide the high-level background required for understanding our approach and use examples to highlight the differences between rewrite rules and resynthesis. We also describe an overview of our approach along with some concrete examples showing how rewrite rules and resynthesis can work together.

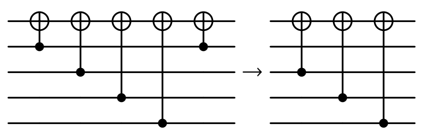
2.1 Quantum Circuits Background

Quantum circuits. Quantum circuits are composed of combinations of quantum operations (or *gates*) applied to *qubits*. Some operations, like the *Hadamard* gate (H), are only applied to a single qubit, whereas operations like the *controlled-NOT* gate (CX) apply to an ordered pair of qubits. Another common class of quantum gates includes rotational gates parameterized on input angles, such as the R_z^θ gate.

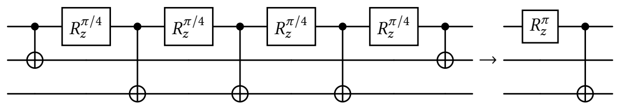
Consider the example circuit on the left sides of Figs. 4 and 5, where each horizontal wire corresponds to a qubit—e.g., the H gate is the first gate applied to qubit q_1.

Rewrite rules. A rewrite rule is a pair of semantically equivalent circuits. Although rewrite rules are in principle bidirectional, we will refer to the left-hand side as the *pattern* and the right-hand side as the *replacement* for simplicity. Fig. 3 shows examples of some commonly used rewrite rules. Applying rewrite rules to a circuit is simple. Begin by searching for a *match* for the pattern and if one exists, substitute it with the replacement. For example, in Fig. 4, there is a match for the pattern of the rule in Fig. 3c, shown by the highlighted subcircuit. Rewriting the match to the replacement allows the rule in Fig. 3d to apply and reduces the gate count by one. This general idea of composing rewrite rules is the heart of how we optimize quantum circuits using rewrite rules.

Resynthesis. Circuit *resynthesis* takes advantage of the vast line of work done in *unitary synthesis* to resynthesize circuits according to some optimization objective. A circuit can be represented as a unitary matrix. Given a circuit's unitary matrix, unitary synthesis constructs a new circuit, with fewer gates, whose unitary is within ϵ of the original unitary according to some distance metric. Note that the original circuit's structure is lost by converting it into a unitary and the



(a) Rewrite rules better



(b) Resynthesis better

Figure 6. Comparing rewrite rules and resynthesis.

synthesis algorithm needs to search for a new circuit structure from scratch. This is inherently a slow search through the space of circuits, e.g., the BQSKit compiler [69] performs a bottom-up search using two-qubit subcircuits.

Fig. 5 illustrates circuit resynthesis using the same initial circuit as Fig. 4 and no approximation by setting $\epsilon = 0$. Observe how the final circuit is equivalent to the result from applying rewrite rules because we can apply the rule in Fig. 3c to push the R_z^{π} gate across the control of the CX.

2.2 Comparing Rewriting and Resynthesis

Recall that resynthesis is limited by the number of qubits in the circuit, because the size of the unitary is exponential in the number of qubits. Fig. 6a is an example of a circuit where it is better to apply rewrite rules. The structure closely resembles the circuit for the *quantum Fourier transform* (QFT) [12], a critical subroutine in many quantum algorithms. This circuit involves too many qubits for unitary synthesis to succeed in a reasonable amount of time. However, it only takes a few applications of two rewrite rules (Figs. 3a and 3b) to quickly get to the right-hand side. Even if we partition this circuit into more tractable 3-qubit subcircuits to resynthesize, we would not be guaranteed to reach the right-hand side. In fact, it would require a series of lucky coincidences over multiple rounds of partitioning the circuit.

Resynthesis involving fewer qubits though can compensate for the limited-sized patterns in rewrite rules. Fig. 6b is an example of a deep circuit where resynthesis can accelerate the search. Although we can achieve the optimized circuit using rewrite rules, it requires a long sequence of several of the rules from Fig. 3 in a very specific order. Resynthesis can discover the circuit on the right-hand side all at once because the circuit only involves 3 qubits.

As we saw in the above examples, there are complementary qualities between rewrite rules and resynthesis.

2.3 Our Approach

Unifying rewrite rules and resynthesis. Our framework to unify rewrite rules and resynthesis introduces an abstraction for *transforming* circuits. We specify a function called a circuit *transformation*, denoted τ_ϵ, that returns a circuit that is semantically equivalent to the original up to the approximation ϵ. Beyond this guarantee, transformations are closed-box. Given a set of rewrite rules and resynthesis algorithms, we can instantiate a set of transformations. Crucially, our framework allows us to apply circuit transformations in any order, and we prove an upper bound on the final approximation degree when applying an arbitrary sequence of transformations.

Optimization objectives. Optimizing quantum circuits requires diverse optimization objectives depending on the application. In NISQ, two-qubit gates are the dominant source of noise whereas in FTQC, T gates are the most costly to perform in an error-corrected fashion, followed by two-qubit gates. We view these gate-minimizing optimization objectives as *soft* constraints in our search. Our *hard* constraint when resynthesizing subcircuits is staying within the specified global error threshold ϵ_f. Allowing more error can allow the synthesis algorithm to find a solution with fewer gates, so it is critical to find a balance between these two competing optimization objectives. For example, an objective for NISQ might be the following: $\text{argmin}_{C'}\ \text{2Q-COUNT}(C')$ s.t. $\epsilon_{C'} \leq \epsilon_f$.

The GUOQ algorithm. The vast and discrete landscape for optimizing quantum circuits provides sparse navigation for traversing it. GUOQ is our simple algorithm inspired by simulated annealing [28] that rapidly and randomly searches the space of transformations. We find that an approach like GUOQ is well-suited for solving our problem because it has minimal memory requirements, is easy to implement, and explores the solution space much faster than other approaches. At a high level, the algorithm maintains a single candidate and applies randomly chosen transformations to randomly chosen subcircuits. Transformations with $\epsilon = 0$ can be applied an unlimited number of times while transformations with $\epsilon > 0$ are limited based on the target error threshold. If a transformation preserves or reduces gates related to the optimization objective, it is always accepted. Otherwise, it is only accepted with a small probability.

A concrete example. As a primer, we provide an example demonstrating the benefits of combining rewrite rules and resynthesis on the `barenco_tof_10` benchmark, which is an implementation of a multi-control Toffoli gate [5], and the `qft_20` benchmark, which implements the *quantum Fourier transform* (QFT) [12]. These benchmarks are critical building blocks for the famous Shor's algorithm [57]. Fig. 7 shows the two-qubit gate count of the best solution over an hour of search for GUOQ using 1) rewrite rules only, 2) resynthesis only, and 3) rewrite rules and resynthesis combined.

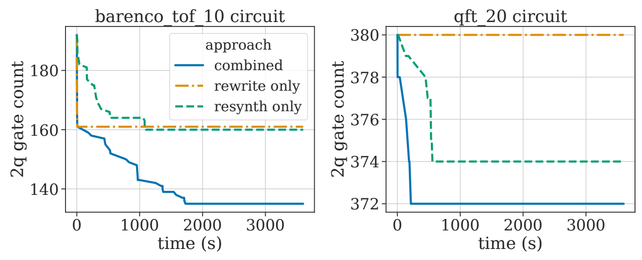


Figure 7. An example showing the two-qubit gate count of the current best solution for `barenco_tof_10` and `qft_20` over an hour of search using 1) only rewrite rules, 2) only resynthesis, and 3) rewrite rules and resynthesis combined.

As we can see, rewrite rules on their own quickly get stuck in a local minimum for almost the entire search, whereas resynthesis on its own is able to gradually make progress but is too slow. Combining rewrite rules and resynthesis allows resynthesis to progress the search when rewrite rules get stuck by mutating the circuit and "teleporting" the candidate solution to a different area of the optimization landscape. Working in tandem, rewrite rules and resynthesis can go beyond the capabilities of either alone.

3 Quantum Circuits and Approximations

We now formalize the necessary background for understanding quantum circuits and their semantics. The notion of approximate semantics and subcircuits will be critical for our framework.

Semantics. A quantum gate is a linear transformation of the state vector. A gate g acting on m qubits can be represented by a $2^m \times 2^m$ *unitary* matrix U_g. Composing the gates in a circuit with n qubits using matrix multiplication results in a $2^n \times 2^n$ unitary matrix exactly representing the semantics of the circuit.

Example 3.1. The semantics of the T and CX gates are the following unitary matrices:

$$U_T := \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix},\ U_{CX} := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Consider the circuit $C := T\ q_1;\ CX\ q_0\ q_1;$. The unitary U_C representing the semantics of C is precisely $U_{CX}(I \otimes U_T)$, where the tensor product $(I \otimes U_T)$ with the identity matrix denotes that the T gate is applied to the second qubit.

Circuit equivalence. Circuits are semantically equivalent if their unitaries are exactly equal. Additionally, some circuits may be equivalent up to *global phase* because two quantum states $|\psi_1\rangle$ and $|\psi_2\rangle = e^{i\phi}\,|\psi_1\rangle$ are observationally indistinguishable [42] for any angle $\phi \in \mathbb{R}$. Two circuits C and C' are semantically equivalent modulo global phase, denoted $C \equiv C'$, if and only if $U_C = e^{i\phi} U_{C'}$ for some angle ϕ.

Approximating circuits. We can approximate the semantics of a circuit C arbitrarily by defining a notion of *distance*, which is a function of the original and approximated circuits' unitaries. The *Hilbert–Schmidt distance* (Δ) is a convenient distance function used in prior work [44, 45] due to its ability to handle equivalence modulo global phase and ease of computation. The Hilbert–Schmidt distance between two unitary matrices is formally defined in Def. 3.2. Using this definition, we can define approximate circuit equivalence as shown in Def. 3.3.

Definition 3.2 (Hilbert–Schmidt distance)**.** Let U and U' be two $N \times N$ unitary matrices. $\Delta(U, U') := \sqrt{1 - \frac{\|Tr(U^\dagger U')\|^2}{N^2}}$.

Definition 3.3 (Approximate circuit equivalence)**.** Two circuits C and C' are ϵ-equivalent, denoted $C \equiv_\epsilon C'$, if and only if $\Delta(U_C, U_{C'}) \le \epsilon$.

Subcircuits. To define a subcircuit, it is best to consider a DAG representation of a circuit, where the nodes are gates and the wires between gates are directed from left to right. Then a subcircuit is precisely a *convex* subgraph of this DAG. As defined in prior work [68], a *convex* subgraph is a subgraph that contains every path that exists between its nodes in the original graph. Intuitively, this requirement enforces continuous qubit wires in the subcircuit. For example, Fig. 2 (top) shows two highlighted subcircuits.

4 A Unified Optimization Framework

We are ready to formally describe our framework unifying rewrite rules and resynthesis. Our framework introduces abstract *transformations* with an associated error tolerance ϵ. We show how to represent rewrite rules and resynthesis in this unifying framework and prove an upper bound on the error when composing these transformations in arbitrary sequences. This underlies the design of GUOQ, which applies transformations in arbitrary orders.

4.1 Circuit Transformations

The following definition presents an abstraction of a circuit transformation as a function that takes a circuit C and produces an ϵ-equivalent C'.

Definition 4.1 (Transformation)**.** Let C represent a circuit type. A transformation $\tau_\epsilon : C \to C$ accepts a circuit C and returns a circuit C' such that $C \equiv_\epsilon C'$.

Rewrite rules and resynthesis can be represented in this framework as transformations over subcircuits. Consider a rewrite rule $C_1 \longrightarrow C_2$. The transformation

$$\tau_0(C) := \begin{cases} C_2 & \text{if } C = C_1 \\ C & \text{otherwise} \end{cases}$$

captures the rewrite rule by transforming circuits that are syntactically equivalent to C_1 (up to qubit renaming) and acting as the identity function otherwise. Observe how this transformation carries an $\epsilon = 0$ because rewrite rules preserve exact semantic equivalence.

Similarly, we can define a transformation representing resynthesis. Assume there exists a circuit resynthesis function $\textsc{resynth} : (\mathcal{C} \times \mathbb{R}) \rightarrow \mathcal{C}$ that given a circuit C and error tolerance ϵ, returns a circuit C' such that $C \equiv_\epsilon C'$. The transformation is simply $\tau_\epsilon(C) \coloneqq \textsc{resynth}(C, \epsilon)$. An example of such a circuit resynthesis function is a thin wrapper around a unitary synthesis function that computes the input circuit's unitary before invoking unitary synthesis.

4.2 Composing Transformations

Composing transformations is not as straightforward as composing rewrite rules because transformations are allowed to approximate the circuit. Prior work [45] shows how to upper bound the error when approximating *disjoint* partitions of a circuit. We present a flexible and generic framework that allows us to apply transformations in an arbitrary fashion. In particular, we can apply a transformation to a subcircuit that only *partially* contains a previously transformed subcircuit.

Formalized in Thm. 4.2, we prove that the upper bound on the error when composing a finite sequence of transformations is the sum of all the errors from each transformation. Without loss of generality, we can assume all transformations have the same error.

Theorem 4.2. *Suppose we are given a set of transformations $\tau_\epsilon^1, \ldots, \tau_\epsilon^n$. Let $C_0, \ldots, C_n$ be a sequence of circuits such that C_i is the result of applying transformation τ_ϵ^i to a subcircuit of C_{i-1} for all $1 \leq i \leq n$. Then, $C_0 \equiv_{n\epsilon} C_n$.*

5 GUOQ: A Stochastic Algorithm

In this section, we begin by formally stating the quantum-circuit optimization problem. Optimizing quantum circuits is hard because of the large search space and the difficulty of simulating quantum circuits. Next we describe our algorithm GUOQ for solving this problem given an instantiation of our framework. GUOQ is fast, flexible, and easy to implement. It applies a given set of transformations in a randomized fashion inspired by *simulated annealing*, a classic algorithm for solving discrete optimization problems. Finally, we discuss implementation details for optimizing our algorithm.

5.1 Optimization objectives for quantum circuits

Different quantum computing hardwares and paradigms will require unique optimization objectives. For example, on NISQ hardware it is critical to reduce two-qubit gate count. Other optimization objectives include T count and circuit *depth*, rather than gate count. Our approach is flexible and we can define any cost function, $\textsc{cost} : \mathcal{C} \rightarrow \mathbb{R}$, to minimize, where $\mathcal{C}$ is the set of all circuits.

Example 5.1 (Multiple optimization objectives)**.** Consider the FTQC setting where we want to reduce primarily T gates, followed by CX gates. We can model this optimization objective by defining COST as $2 \cdot \#_T(C) + \#_{CX}(C)$, where $\#_T(C)$ and $\#_{CX}(C)$ are the T and CX gate counts, respectively.

Transformations in our framework can be approximate, so it is natural to accept as input an error tolerance that the result should not exceed. Using this error tolerance as a hard constraint and COST as a soft constraint we can formulate the problem as a succinct constrained optimization problem.

Definition 5.2 (Quantum-Circuit Optimization Problem)**.** Given a circuit C and an *error tolerance* $\epsilon_f \geq 0$, the quantum-circuit optimization problem is the following constrained optimization problem:

$$\underset{C'}{\operatorname{argmin}}\ \textsc{cost}(C') \quad \text{s.t. } \Delta(C, C') \leq \epsilon_f$$

5.2 The GUOQ Algorithm

We propose an algorithm inspired by simulated annealing. Simulated annealing is a general algorithm for solving optimization problems with large search spaces. It has many nice properties such as being fast, memory-efficient, easy to implement, and interruptible at any time to obtain a partial solution. These properties, inherited in our algorithm, unlock an effective approach for solving a problem that is incredibly difficult to craft or learn predictive heuristics for.

Alg. 1 shows the pseudocode for our algorithm. The inputs to the algorithm are the inputs to the quantum-circuit optimization problem and a set of transformations $\mathcal{T}$. Given a set of rewrite rules and resynthesis methods, we can instantiate $\mathcal{T}$ as discussed in § 4. The "moves" we are allowed to make to modify the current solution are precisely the transformations in $\mathcal{T}$. The heart of the algorithm simply randomly samples a transformation and randomly samples a subcircuit of the current solution circuit to apply the transformation to. This move is always accepted if it improves or preserves the quality of the solution with respect to COST. Otherwise, it is accepted with some small probability that can be tuned using the temperature hyperparameter t. We adapt the standard acceptance probability for simulated annealing [28], which approaches 0 as the candidate solution cost increases.

The remainder of the algorithm ensures the upper bound on the error in the final solution does not exceed the specified tolerance ϵ_f. Using Thm. 4.2, we can simply keep track of the sum of all the errors across all transformations applied. If applying a transformation would exceed the error bound, then we abstain and skip to the next iteration where we have the opportunity to sample a rewrite rule transformation with $\epsilon = 0$. Thm. 5.3 states the correctness of GUOQ.

Theorem 5.3 (Correctness of GUOQ)**.** *Let C' be the result of $\textsc{guoq}(C, \epsilon_f, \mathcal{T})$ for any circuit C, error tolerance $\epsilon_f \geq 0$, and set of transformations $\mathcal{T}$. Then, $C \equiv_{\epsilon_f} C'$.*

Algorithm 1 The GUOQ Algorithm

```
1:  procedure GUOQ(circuit $C$, error $\epsilon_f$, transformations $\mathcal{T}$)
2:      initialize $C_{best}$ and $C_{curr}$ to $C$
3:      initialize error$_{best}$ and error$_{curr}$ to 0
4:      while within time limit do
5:          Randomly select transformation $\tau_\epsilon$ in $\mathcal{T}$
6:          if error$_{curr} + \epsilon > \epsilon_f$ then
7:              continue
8:          Randomly select subcircuit $C_s$ in $C_{curr}$
9:          $C^\tau_{curr} \leftarrow$ result of replacing $C_s$ with $\tau_\epsilon(C_s)$ in $C_{curr}$
10:         if COST$(C^\tau_{curr}) \leq$ COST$(C_{curr})$ then
11:             $C_{curr} \leftarrow C^\tau_{curr}$
12:             error$_{curr} \leftarrow$ error$_{curr} + \epsilon$
13:         else with probability $\exp\left(-t\frac{\text{COST}(C^\tau_{curr})}{\text{COST}(C_{curr})}\right)$
14:             $C_{curr} \leftarrow C^\tau_{curr}$
15:             error$_{curr} \leftarrow$ error$_{curr} + \epsilon$
16:         if COST$(C_{curr}) <$ COST$(C_{best})$ then
17:             $C_{best} \leftarrow C_{curr}$
18:             error$_{best} \leftarrow$ error$_{curr}$
19:     return $C_{best}$
```

5.3 How to implement GUOQ efficiently

We now discuss key implementation ideas that improve the performance of GUOQ in practice.

Weighing fast & slow. Applying a transformation to a circuit is decomposed into selecting a transformation to apply and a location in the circuit to apply it to. In practice, we limit the probability of choosing resynthesis to 1.5% of the time, since it is expensive. In the remaining 98.5% of the time, we uniformly sample one of the rewrite rules.

Randomly selecting subcircuits. We choose a random subcircuit to apply the transformation to by picking a node uniformly at random in the circuit DAG to begin constructing a subcircuit from. For rewrite rules transformations, completely random subcircuits will likely not be transformed nontrivially. We optimize this step by starting at a random node and performing a full pass through the circuit, replacing every disjoint match of the left-hand side with the right-hand side of the rule. For resynthesis transformations, we start at a random node and grow a subcircuit greedily until we cannot add more nodes without exceeding the qubit limit. We only apply resynthesis to a single subcircuit per iteration.

Applying resynthesis asynchronously. Invoking a unitary synthesis subroutine, even for a circuit with only 3 qubits, is slow and takes on the order of seconds or minutes to return a solution. To make better use of our time, we choose to make these calls asynchronously so we can apply rewrite rules concurrently. If the result of resynthesis is accepted, we effectively discard all modifications from rewrite rules made in the interim.

Gate set	Gates	Architecture
IBMQ20 [2]	$U1^\theta, U2^{\theta_1,\theta_2}, U3^{\theta_1,\theta_2,\theta_3}, CX$	Supercond.
IBM-EAGLE [2]	R_z^θ, SX, X, CX	Supercond.
IONQ [61]	$R_x^\theta, R_y^\theta, R_z^\theta, R_{xx}^\theta$	Ion Trap
Nam [40]	R_z^θ, H, X, CX	None
Clifford + T [17]	$T, T^\dagger, S, S^\dagger, H, X, CX$	Fault Tolerant

Table 2. Summary of gate sets.

6 Implementation and Evaluation

We implemented GUOQ in Java. GUOQ interfaces with existing resynthesis tools [44, 69] and can be instantiated with arbitrary rewrite rules and gate sets. We designed our evaluation of GUOQ to answer the following research questions:

Q1 How does GUOQ compare to state-of-the-art tools?
Q2 What's the effect of unifying rewriting & resynthesis?
Q3 What's the best way to apply rewriting & resynthesis?
Q4 Does GUOQ extend to fault-tolerant computing (FTQC)?

We focus on NISQ in the first three questions using the diverse gate sets and tools available and then explore FTQC in Q4.

Gate sets. Our approach is flexible and can handle arbitrary gate sets. We evaluate on a variety of gate sets for promising quantum architectures, summarized in Table 2. The Nam gate set is not realized directly on hardware but is studied extensively in prior work due to its resemblance to the Clifford + T gate set.

Benchmarks. We consider all the benchmarks used in prior optimization work [32, 67] as well as benchmarks used in approximate optimization [45]. Prior optimization work primarily focuses on circuits with fewer than 2,000 gates. We expand the suite to larger application circuits considered in prior mapping-and-routing work [73], and circuits implementing standard algorithms. Experimenting with larger circuits is key because total gate count is the primary metric that affects the scalability of optimizers for circuits with more than a few qubits. Our benchmark suite of 247 circuits includes important quantum algorithms in the near and long term such as QAOA [14], VQE [47], QPE [30], QFT [12], Grover's [18], and Shor's [57]. To ensure a fair comparison between each tool's optimization phase, the input circuit throughout this evaluation is always already decomposed into the target gate set. Fig. 15 in Appendix B summarizes the total gate counts of all the input circuits. The benchmark circuits act on 4 to 36 qubits.

Metrics. For NISQ, we focus on two-qubit gate reduction because two-qubit gates have orders of magnitude higher error rates compared to single-qubit gates. Gate reduction is computed as $1 - \frac{\text{optimized count}}{\text{original count}}$. We also compute the circuit *fidelity*, or success probability, to emphasize that two-qubit gates are the dominant source of error. The fidelity of a gate

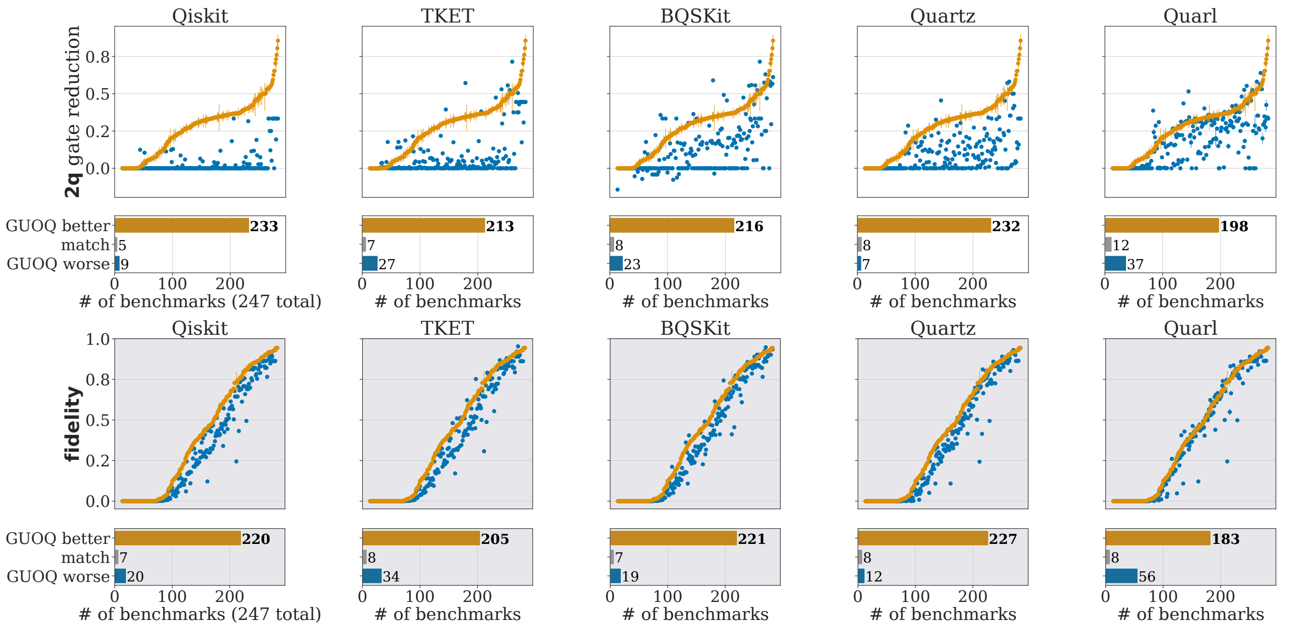


Figure 8. Comparison against state-of-the-art optimizers on the IBM-EAGLE gate set. Each graph has a bar chart summarizing the number of benchmarks GUOQ outperforms, matches, or underperforms the tool in the title. In each graph, orange is GUOQ and blue is the tool in the title. Each point is the mean metric and a 95% confidence interval for a number of runs on a single benchmark. The points are sorted based on GUOQ and a point where the orange lies above the blue indicates GUOQ is better.

is 1− its error rate and the fidelity of a circuit is the product of its gate fidelities. For IBMQ20 and IBM-EAGLE, we use the calibration data for the IBM Washington device available in Qiskit; for the Ion gate set, we use data for the IonQ Forte device [22]. In Q4, we consider different metrics for FTQC.

In our plots, each point corresponds to a benchmark. For each benchmark circuit and tool, we compute the mean metric for a number of runs of the tool (10 trials for GUOQ) and a 95% confidence interval. For readability, we present the benchmarks in increasing order sorted based on GUOQ. A point where GUOQ lies above the respective point for the other tool implies that GUOQ outperforms the other tool. The bar plot below each scatter plot summarizes the number of benchmarks GUOQ on average outperforms, matches, and underperforms, respectively, the tool in the title.

Q1: Comparison with state-of-the-art optimizers

State of the art. We compared GUOQ to 7 state-of-the-art optimizers listed in Table 3. Our goal is to compare the optimization phase of various tools so we do not invoke mapping and routing and tools are allowed to change the input circuit's connectivity if they support this feature. We exclude PyZX [29] from this research question because its primary optimization objective is to reduce T gate count and often either increases or preserves the two-qubit gate count. We compare against PyZX in Q4 where we explore T reduction.

Tool	Superoptimize	Approach
Qiskit [2]	✗	fixed sequence of passes
TKET [58]	✗	fixed sequence of passes
VOQC [20]	✗	fixed sequence of passes
BQSKit [69]	✓	partition + resynthesize
QUESO [67]	✓	beam search + rewrite rules
Quartz [68]	✓	beam search + rewrite rules
Quarl [32]	✓	reinf. learning + rewrite rules

Table 3. State-of-the-art optimizers.

Instantiation of GUOQ. GUOQ uses rewrite rules generated by QUESO [67] and does not consider any size-increasing rewrite rules. To reduce two-qubit gate count, GUOQ uses BQSKit [69] for resynthesis and the optimization objective is to maximize fidelity. We limit random subcircuits to have at most 3 qubits. The temperature hyperparameter t is set to 10, corresponding to a very small probability of accepting a worse solution. We chose this value empirically by performing a sweep of values from 0 (always accept) to 10.

Experimental setup. Unless otherwise indicated, we allocated each tool 1 hour, 32GB of RAM, and 1 CPU core on an AMD EPYC 7763 64-Core Processor. Quarl was run on a cluster of machines and was allocated 64GB of RAM, 1 NVIDIA A100-SXM4-40GB or 80GB GPU, and 1 CPU core. We only evaluate a tool on its supported gate sets. For approximate tools, we enforce an error upper bound of 10^{-8}, which is (1)

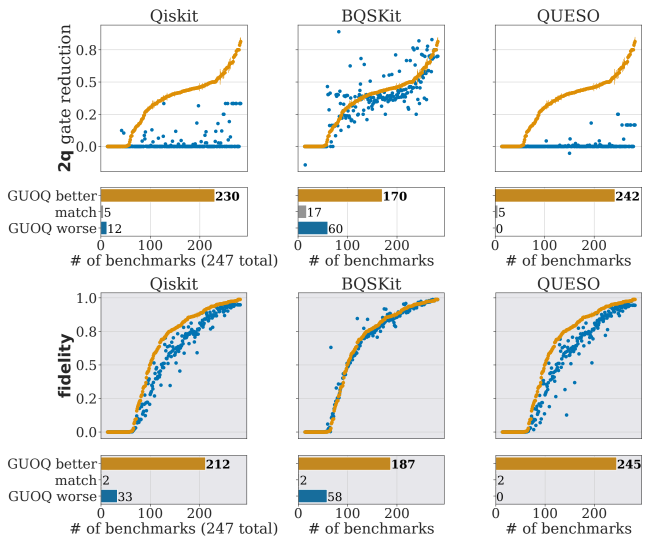


Figure 9. GUOQ vs state-of-the-art on the IONQ gate set.

many orders of magnitude smaller than the error rate of a single two-qubit gate in NISQ (10^{-3} [27]) and (2) on-par with the logical error rate of a single error correction cycle for FTQC (10^{-6} to 10^{-9} [41]) or an arbitrary-angle approximation.

We run Quarl for 3 trials with rotation merging, following their paper's experimental setup. For Qiskit and BQSKit, we use the most powerful optimization levels, which are 3 and 4, respectively. We report the best solution found within the time and memory limits for all tools. That is, we use the partial solutions that GUOQ, QUESO, Quartz, and Quarl provide and use the original circuit for the other tools.

Results. The results for the IBMQ20, IBM-EAGLE, and Nam gate sets are all similar, so we only show the plots for the IBM-EAGLE gate set. Fig. 8 shows the comparison between GUOQ and state-of-the-art with respect to two-qubit gate reduction and fidelity on the IBM-EAGLE gate set. Recall that benchmarks where the orange point lies above the blue point are ones where GUOQ outperforms the tool in the title. For example, consider the left-most column where GUOQ outperforms Qiskit with respect to two-qubit gate reduction on 233/247 of the benchmarks, matches on 5, and underperforms on 9. We observe a very similar story in the fidelity graph on the bottom. This result holds in general against all other tools and GUOQ outperforms state-of-the-art on *at least* 80% and 74% of the benchmarks with respect to two-qubit gate reduction and fidelity, respectively. Recall that Quarl requires a GPU to run its specialized reinforcement learning and trains on a subset of the benchmark suite. GUOQ reduces two-qubit gate count by 28% on average while the next best tool, Quarl, has an average reduction of 18% and the best industrial toolkit, TKET, achieves 7% average reduction.

The results in Fig. 9 on the IONQ gate set depict a similar story. As we can see in the comparison against QUESO, a tool

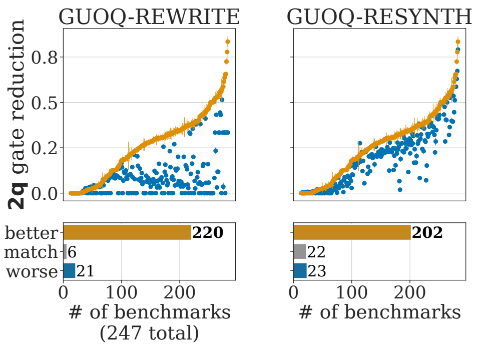


Figure 10. GUOQ vs using only rewrite rules or resynthesis.

which synthesizes rewrite rules, the IONQ gate set is challenging for QUESO because rewrite rules are limited to patterns with a maximum of 3 gates to limit the combinatorial explosion of rules. GUOQ performs well because resynthesis can compensate for the limited rewrite rules. We will see in Q4 an example of the opposite effect. This demonstrates how the effectiveness of rewrite rules and resynthesis varies across different gate sets. Thus, unifying them provides a generic approach for optimizing diverse circuits.

Q1 summary. GUOQ significantly outperforms state-of-the-art across all gate sets. In the *worst* case, GUOQ outperforms other tools on 69% and 74% of the benchmarks with respect to two-qubit gate reduction and fidelity, respectively. In particular, GUOQ achieves an average of 28% two-qubit gate reduction on the IBM-EAGLE gate set while the state-of-the-art superoptimizer (requires GPU) and industrial toolkit achieve average reductions of 18% and 7%, respectively.

Q2: Effect of combining rewriting and resynthesis

We explored this question by fixing the IBMQ20 gate set and running GUOQ with three different sets of transformations. GUOQ-REWRITE only uses rewrite rules synthesized by QUESO, GUOQ-RESYNTH only uses resynthesis, and GUOQ uses both.

Results. Fig. 10 shows that removing either rewrite rules or resynthesis is overall detrimental to the performance of GUOQ. Similar to Fig. 8, the title of each plot is the baseline and points below the curve are benchmarks where GUOQ outperforms the baseline. Observe how most of reduction comes from using resynthesis because it is a powerful optimization on its own. Interleaving with rewrite rules, which can take care of simple optimizations quickly, pushes the reduction even further.

Q2 summary. We can exploit the synergy between rewrite rules and resynthesis to achieve well beyond the capabilities of either alone.

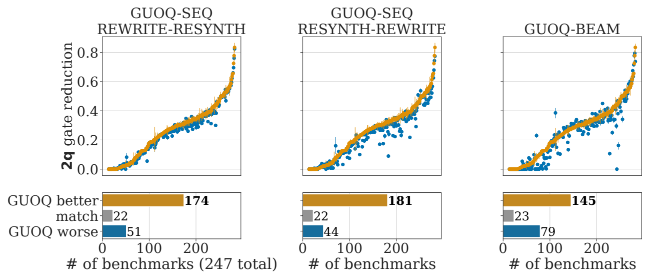


Figure 11. Comparing GUOQ against other search algorithms

Q3: How to combine rewriting and resynthesis?

To answer this question, we fixed the IBMQ20 gate set and set of rewrite rule and resynthesis transformations while varying the search algorithm. We compare GUOQ against three alternate search algorithms for combining rewrite rules and resynthesis: (1) GUOQ-SEQ-REWRITE-RESYNTH, which spends the first half of the allotted time running GUOQ with rewrite rules only, then switches to running with resynthesis only, (2) GUOQ-SEQ-RESYNTH-REWRITE does the opposite, and (3) GUOQ-BEAM uses the MaxBeam algorithm of QUESO [67] to instantiate our framework.

Results. Fig. 11 shows the results. GUOQ outperforms the coarse interleaving GUOQ-SEQ-RESYNTH-REWRITE and GUOQ-SEQ-REWRITE-RESYNTH use on a majority of the benchmarks. This implies that tightly interleaving these different transformations is preferable to choosing a fixed ordering. We also see that relative to each other: GUOQ-SEQ-RESYNTH-REWRITE and GUOQ-SEQ-REWRITE-RESYNTH result in different solutions, which is further evidence that the ordering matters.

We now turn our attention to GUOQ-BEAM, which allows arbitrary orderings of rewrite rules and resynthesis but does not randomly sample transformations like GUOQ. Instead, GUOQ-BEAM maintains a large bounded priority queue of candidates and attempts to apply every transformation in each iteration. Doing so generates many candidates in each iteration (one for each transformation successfully applied), which saturates the queue quickly with solutions of the same cost. In fact, the solutions are generally a few local transformations away from one another so the search makes much slower progress compared to GUOQ. The influx of candidates to the bounded queue also causes many solutions to be pruned, effectively wasting the time spent generating those candidates. Especially for larger circuits, the sizable queue is memory intensive, causing further slowdowns. In summary, the benefit of beam search considering many candidates to avoid local minima is lost in this problem setting.

Q3 summary. GUOQ achieves the best results by tightly interleaving rewrite rules and resynthesis using a simple, lightweight randomized algorithm.

Q4: Does GUOQ extend to FTQC?

In this research question, we shift our focus to the fault-tolerant Clifford + T gate set, where the desired optimization objective is an amalgamation of two NP-hard optimization problems [65]. We want to primarily reduce T gates [9] and reducing two-qubit gates is secondary, but still critical. Error correction is not perfect and the longer a quantum computation runs, the higher the risk of accruing uncorrectable logical error. Two-qubit gates, specifically CX in the Clifford + T gate set, increase the circuit runtime disproportionately because they inherently require more time compared to single-qubit Clifford gates [34] and architectural constraints can limit parallel execution [6, 21]. Furthermore, we anticipate that the problem of CX congestion will be exacerbated in compact FTQC architectures [33] with less routing space. Lastly, recent work [16] has significantly reduced the space-time cost of preparing a T magic state, indicating that the focus on solely reducing T gates may soon need to be recalibrated.

We instantiate GUOQ with Synthetiq [44], a state-of-the-art unitary synthesis algorithm for finite gate sets, and rewrite rules generated by QUESO [67]. We consider two additional tools in this comparison: (1) PyZX [29], a state-of-the-art optimizer for reducing T count using the rewrite-rule-based ZX-calculus and (2) our implementation of a BQSKit-style partitioning optimizer [45] that uses Synthetiq.

Results. The top and bottom rows of Fig. 12 show the comparison against other tools with respect to T and CX gate reduction respectively. GUOQ outperforms all tools except PyZX with respect to T reduction and outperforms all tools with respect to CX gate reduction. Observe how reducing T gates is hard and a general-purpose tool, like Qiskit, only reduces the number of T gates on 5% of the benchmarks. PyZX is a domain-specific optimizer that uses a powerful graph-based theory, the ZX-calculus, which is specialized for reducing phase gates, but does not reduce CX gates at all.

Digging deeper, we find that GUOQ is unable to surpass PyZX in T gate reduction because unitary synthesis for finite gate sets is much harder than for continuous gate sets. Fig. 13 shows the results from the same ablation as in Q2 and we find that rewrite rules contribute more than resynthesis.

To get a sense for how "good" PyZX's solution is, we ran GUOQ on the output of PyZX for the 243 benchmarks it provided a solution for within the time and memory limits. We discovered that GUOQ can drastically reduce the CX gate count without increasing T gate count! Fig. 14 shows this result, which is exciting because PyZX on its own does not reduce CX gate count. Extending PyZX with GUOQ pushes the boundaries for the multifaceted FTQC optimization objective.

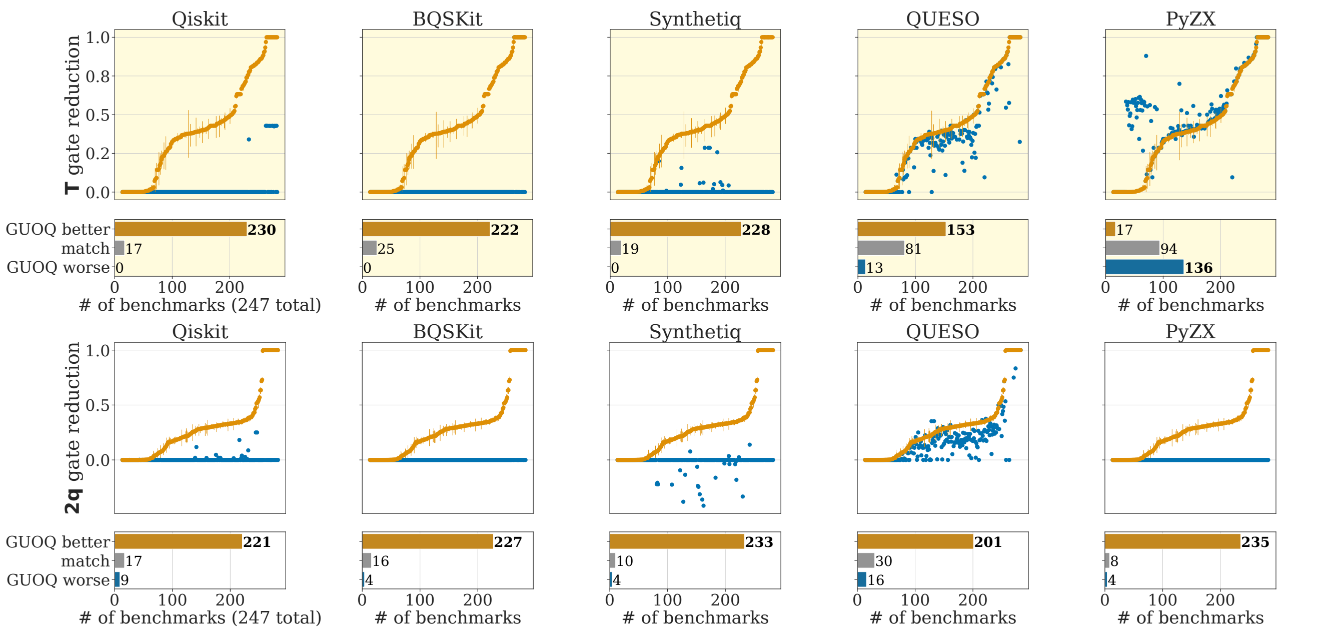


Figure 12. Comparison against state-of-the-art optimizers on the Clifford + T gate set.

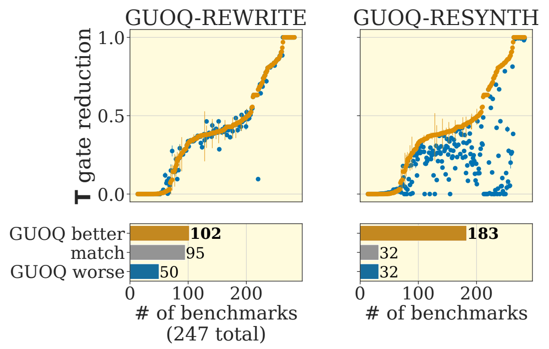


Figure 13. Revisiting Q2 for the Clifford + T gate set.

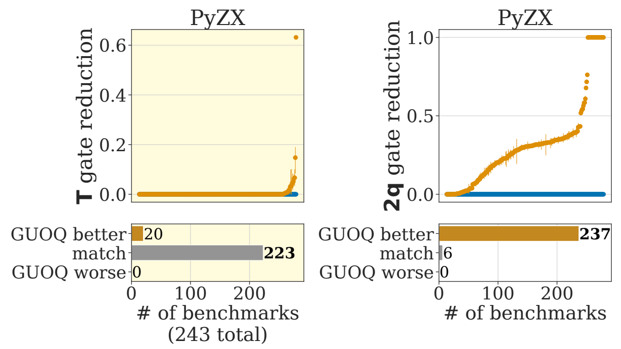


Figure 14. Running GUOQ on PyZX output.

Q4 summary. GUOQ outperforms all tools with respect to T gate reduction except PyZX, which it only outperforms or matches on 45% of the benchmarks. However, GUOQ vastly outperforms PyZX (and other tools) with respect to CX gate reduction on *at least* 81% of the benchmarks, which is also critical in FTQC. Additionally, when run on the output of PyZX, GUOQ can reduce CX gate count on average by 32% without increasing T gate count.

7 Related Work

Quantum optimizers. Traditional quantum-circuit optimizers primarily use a fixed set of hand-crafted optimizations [2, 3, 8, 20, 40, 46, 58] applied in a fixed sequence. PyZX [29] takes a variant of this approach: representing a circuit as a ZX-diagram and applying the graphical rewrite rules of the ZX-calculus. GUOQ instead performs a fine-grained search over arbitrary optimizations.

(Quantum) superoptimizers. The idea of *superoptimizing* classical programs has been around for decades [37, 64]. This vast line of work [11, 23, 36, 39, 48, 54, 55] stems from the idea of finding the *optimal* solution for small programs that can later be applied in peephole optimizers. Approaches like STOKE [55] use MCMC [19] to superoptimize x86 assembly by randomly mutating the program. The ergodic theory behind MCMC lends itself to applications where testing correctness is easy. However, quantum circuits cannot be efficiently simulated on classical hardware [42].

Rewrite rules and unitary synthesis have both been used as the basis of quantum-circuit *superoptimizers*. Quartz [68] and QUESO [67] synthesize rewrite rules for a given gate set and use beam search to explore the space of rule application schedules. Quarl [32] applies reinforcement learning to schedule the application of rules generated by Quartz. None of these apply approximate circuit transformations. On the other hand, resynthesis-based superoptimizers [45, 69] optimize circuits by performing a single pass of partitioning into subcircuits, followed by applying unitary synthesis to each subcircuit. This approach circumvents the qubit count limitations of unitary synthesis, but is rigid and misses potential optimization opportunities that straddle the boundary between two adjacent partitions. In contrast, GUOQ is not limited to resynthesizing disjoint subcircuits of the original circuit. GUOQ can freely choose subcircuits to resynthesize by using Thm. 4.2 to bound the error when composing applications of resynthesis.

In a similar spirit to Quarl, other recent approaches have applied reinforcement learning to superoptimize quantum circuits. MQTPredictor [50] predicts the optimal passes and device with respect to an optimization objective but currently only considers a subset of Qiskit and TKET passes. AlphaTensor-Quantum [53] is a closed-source approach that uses reinforcement learning for tensor decomposition to optimize T count.

Domain-specific optimizers. Other work targets specific applications like Hamiltonian simulation [31, 35] or variational algorithms [24]. Some tools operate at a lower level of abstraction than we do by considering gate *pulses* [56] or a higher level by starting from a program written in a high-level quantum programming language [71, 72]. In contrast, GUOQ is designed to be flexible for diverse quantum assembly circuits and architectures.

Unitary synthesis. Extensive prior work has considered the unitary synthesis problem for both finite and parameterized gate sets. For finite gate sets, some approaches [4, 63] provide theoretical guarantees of optimality in terms of circuit size, whereas others [26, 44] sacrifice optimality for improved runtime. For parameterized gate sets, several techniques [13, 51, 52, 60, 70] use numerical optimization to instantiate template circuits. However, all of these algorithms can only be applied to circuits with a handful of qubits.

8 Conclusions

We have described a generic and flexible framework for unifying rewriting and resynthesis for quantum-circuit optimization along with a simple and effective algorithm parameterized on an instantiation of this framework. Our approach, GUOQ, outperforms state-of-the-art optimizers in both near (NISQ) and long (FTQC) term quantum computing paradigms. For future work, we are interested in developing *symbolic* unitary synthesis so we can learn general transformations on the fly—as opposed to ones with highly specific angles—that will be more likely to apply later in the search.

Acknowledgments

We thank the anonymous reviewers and our shepherd, Jens Palsberg, for their thoughtful feedback. We are grateful to the CHTC [10] team for their assistance in our experimentation and the BQSKit team for their guidance on using BQSKit. This work is supported by NSF grants #1652140 and #2212232 and awards from Meta and Amazon. This research is also partially supported by the OVCRGE at the University of Wisconsin-Madison with funding from the Wisconsin Alumni Research Foundation.

A Proofs

Thm. 4.2. By induction on n. *Base case:* Trivially, $C_0 \equiv_0 C_0$. *Induction case:* Assume $C_0 \equiv_{k\epsilon} C_k$ for $k \geq 0$ as our inductive hypothesis. We will show $C_0 \equiv_{(k+1)\epsilon} C_{k+1}$. We have $C_k \equiv_\epsilon C_{k+1}$ by the proof of [45, §3.8] for disjoint partitions. Let $U := U_{C_0}$, $U' := U_{C_k}$, $U'' := U_{C_{k+1}}$, $\epsilon_1 := k\epsilon$, and $\epsilon_2 := \epsilon$. Now it suffices to show $\Delta(U, U'') \leq \epsilon_1 + \epsilon_2 = (k+1)\epsilon$.

$$\begin{aligned}
&\Delta(U, U'') \\
&= \sqrt{1 - \frac{\|Tr(U^\dagger U'')\|^2}{N^2}} && \text{Def. 3.2} \\
&= \sqrt{1 - \frac{\|Tr(U^\dagger U' U'^\dagger U'')\|^2}{N^2}} && U'U'^\dagger = I \\
&= \sqrt{1 - \frac{\|Tr[(U^\dagger U')(U'^\dagger U'')]\|^2}{N^2}} && \text{Grouping terms} \\
&\leq \sqrt{1 - \frac{\|Tr(U^\dagger U')\|^2}{N^2}} + \sqrt{1 - \frac{\|Tr(U'^\dagger U'')\|^2}{N^2}} && \text{[45, §3.8]} \\
&\leq \epsilon_1 + \epsilon_2 && \text{Defs. 3.2 and 3.3}
\end{aligned}$$

By definition of $\equiv_\epsilon$, $C_0 \equiv_{(k+1)\epsilon} C_{k+1}$, as desired. □

Thm. 5.3. Follows directly from Thm. 4.2 and Alg. 1, line 6.

B Benchmark Data

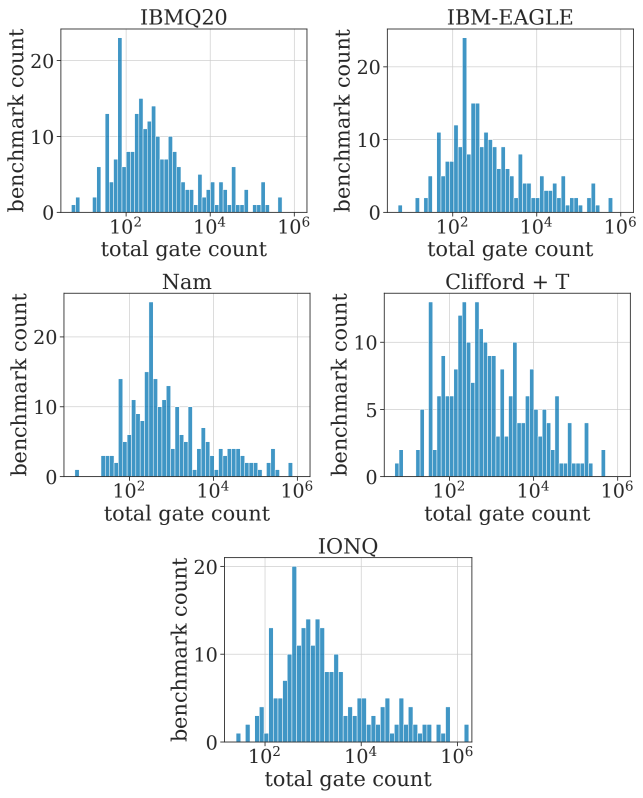


Figure 15. Summary of the benchmarks' original total gate counts across all gate sets (log-scale x-axis).

C Artifact Appendix

C.1 Abstract

This artifact is for GUOQ, a quantum-circuit optimizer built on top of QUESO [67]. It unifies rewrite rules and unitary synthesis, and uses a radically fast and simple random algorithm to leverage the complementary characteristics of both optimizations. This artifact contains the GUOQ source code, benchmark suite, scripts to run the experiments and plot the results, Docker images with everything pre-installed, results from the paper's evaluation, as well as a README with instructions. There are two Docker images: one contains the Quarl [32] baseline and the other contains everything else.

C.2 Artifact check-list (meta-information)

- **Algorithm:** Simulated annealing, rewrite rules, unitary synthesis.
- **Run-time environment:** Linux, amd64.
- **Hardware:** The full experimental evaluation requires an NVIDIA A100 (40GB) GPU for one of the baselines (Quarl [32]).
- **Output:** Figures 1, 7, and 8-15 from the paper, average reduction reported in RQ1 and RQ4 summaries, and number of qubits in the benchmarks.
- **Experiments:** Scripts, Docker images, and detailed README with instructions on reproducing the experiments.
- **How much disk space required (approximately)?:** 50GB.
- **How much time is needed to prepare workflow (approximately)?:** 30 minutes.
- **How much time is needed to complete experiments (approximately)?:** 26 hours for limited set. 4 years for full set.
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:** Apache License 2.0.
- **Archived (provide DOI)?:** Yes: https://doi.org/10.5281/zenodo.14055562.

C.3 Description

C.3.1 How to access. The instructions in the Zenodo README discuss how to access the Docker images (recommended) or install from source.

C.3.2 Hardware dependencies. We recommend running this artifact on a Linux machine with at least 32GB of RAM. The Docker images are built for the `linux/amd64` platform. One of the baseline tools requires an NVIDIA A100 (40GB) GPU and at least 64GB of RAM. If you have access to these resources, then you may run those experiments. If not, our data for those is included in the artifact.

C.3.3 Software dependencies. Using the provided Docker images requires no additional software dependencies. The README discusses which software dependencies are required when installing from source.

C.4 Installation

The only dependency that needs to be installed is Docker. The Docker images have everything else needed installed. The README also included direcitons on installing GUOQ from source and references to the baseline tools' instructions.

C.5 Basic Test

Each Docker image contains a kick-the-tires script to run a small set of experiments and plot the results. See the "Kick the Tires" instructions in the README.

C.6 Experimental Workflow

The README explains the directory structure of the main Docker image in detail. We provide scripts to run each tool and gather the results as well as scripts that instrument entire sets of experiments and plot the results. Most of the scripts used to run tools are written in Python and the experiment scripts are in Bash. The plotting script is in Python. All the new results can be found in `/home/fresh_results` and the generated graphs are saved to `/home/artifact/graphs`. Logs for GUOQ will be included in the results directory along with JSON summaries of the results.

C.7 Evaluation and expected results

The provided scripts execute the entire evaluation workflow. They generate the figures from the paper and print out in-text claims. These outputs allow users to reproduce all claims from the paper.

GUOQ is a random algorithm, so the results may have some variability. In our evaluation, we ran it for 10 trials and reported the mean metric with a 95% confidence interval. The limited script only runs GUOQ for 1 trial and does not use the full hour timeout from the paper. Additionally, the results may vary if run on a different machine than used in our evaluation. All of these factors may contribute to some minor deviations in the results from the paper.

C.8 Experiment customization

It is simple to invoke GUOQ with new benchmarks and we provide the necessary commands in the final section of the README. GUOQ can also be used with new gate sets. QUESO can be extended with new gate sets to generate rewrite rules for GUOQ. If necessary, adding a new unitary synthesis algorithm should be a straightforward extension of the resynthesis Python server `resynth.py` and the Java code that sends requests to this server (e.g., `Bqskit.java`).

C.9 Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-badging
- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html

References

[1] Rajeev Acharya, Igor Aleiner, Richard Allen, Trond I. Andersen, Markus Ansmann, Frank Arute, Kunal Arya, Abraham Asfaw, Juan Atalaya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Joao Basso, Andreas Bengtsson, Sergio Boixo, Gina Bortoli, Alexandre Bourassa, Jenna Bovaird, Leon Brill, Michael Broughton, Bob B. Buckley, David A. Buell, Tim Burger, Brian Burkett, Nicholas Bushnell, Yu Chen, Zijun Chen, Ben Chiaro, Josh Cogan, Roberto Collins, Paul Conner, William Courtney, Alexander L. Crook, Ben Curtin, Dripto M. Debroy, Alexander Del Toro Barba, Sean Demura, Andrew Dunsworth, Daniel Eppens, Catherine Erickson, Lara Faoro, Edward Farhi, Reza Fatemi, Leslie Flores Burgos, Ebrahim Forati, Austin G. Fowler, Brooks Foxen, William Giang, Craig Gidney, Dar Gilboa, Marissa Giustina, Alejandro Grajales Dau, Jonathan A. Gross, Steve Habegger, Michael C. Hamilton, Matthew P. Harrigan, Sean D. Harrington, Oscar Higgott, Jeremy Hilton, Markus Hoffmann, Sabrina Hong, Trent Huang, Ashley Huff, William J. Huggins, Lev B. Ioffe, Sergei V. Isakov, Justin Iveland, Evan Jeffrey, Zhang Jiang, Cody Jones, Pavol Juhas, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Tanuj Khattar, Mostafa Khezri, Mária Kieferová, Seon Kim, Alexei Kitaev, Paul V. Klimov, Andrey R. Klots, Alexander N. Korotkov, Fedor Kostritsa, John Mark Kreikebaum, David Landhuis, Pavel Laptev, Kim-Ming Lau, Lily Laws, Joonho Lee, Kenny Lee, Brian J. Lester, Alexander Lill, Wayne Liu, Aditya Locharla, Erik Lucero, Fionn D. Malone, Jeffrey Marshall, Orion Martin, Jarrod R. McClean, Trevor McCourt, Matt McEwen, Anthony Megrant, Bernardo Meurer Costa, Xiao Mi, Kevin C. Miao, Masoud Mohseni, Shirin Montazeri, Alexis Morvan, Emily Mount, Wojciech Mruczkiewicz, Ofer Naaman, Matthew Neeley, Charles Neill, Ani Nersisyan, Hartmut Neven, Michael Newman, Jiun How Ng, Anthony Nguyen, Murray Nguyen, Murphy Yuezhen Niu, Thomas E. O'Brien, Alex Opremcak, John Platt, Andre Petukhov, Rebecca Potter, Leonid P. Pryadko, Chris Quintana, Pedram Roushan, Nicholas C. Rubin, Negar Saei, Daniel Sank, Kannan Sankaragomathi, Kevin J. Satzinger, Henry F. Schurkus, Christopher Schuster, Michael J. Shearn, Aaron Shorter, Vladimir Shvarts, Jindra Skruzny, Vadim Smelyanskiy, W. Clarke Smith, George Sterling, Doug Strain, Marco Szalay, Alfredo Torres, Guifre Vidal, Benjamin Villalonga, Catherine Vollgraff Heidweiller, Theodore White, Cheng Xing, Z. Jamie Yao, Ping Yeh, Juhwan Yoo, Grayson Young, Adam Zalcman, Yaxing Zhang, Ningfeng Zhu, and Google Quantum AI. 2023. Suppressing quantum errors by scaling a surface code logical qubit. *Nature* 614, 7949 (2023), 676–681. https://doi.org/10.1038/s41586-022-05434-1

[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] Matthew Amy and Vlad Gheorghiu. 2020. staq—A full-stack quantum processing toolkit. *Quantum Science and Technology* 5, 3 (jun 2020), 034016. https://doi.org/10.1088/2058-9565/ab9359

[4] Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. 2013. A Meet-in-the-Middle Algorithm for Fast Synthesis of Depth-Optimal Quantum Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 6 (2013), 818–830. https://doi.org/10.1109/TCAD.2013.2244643

[5] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. 1995. Elementary gates for quantum computation. *Phys. Rev. A* 52 (Nov 1995), 3457–3467. Issue 5. https://doi.org/10.1103/PhysRevA.52.3457

[6] Michael Beverland, Vadym Kliuchnikov, and Eddie Schoute. 2022. Surface Code Compilation via Edge-Disjoint Paths. *PRX Quantum* 3 (May 2022), 020342. Issue 2. https://doi.org/10.1103/PRXQuantum.3.020342

[7] Dolev Bluvstein, Simon J. Evered, Alexandra A. Geim, Sophie H. Li, Hengyun Zhou, Tom Manovitz, Sepehr Ebadi, Madelyn Cain, Marcin Kalinowski, Dominik Hangleiter, J. Pablo Bonilla Ataides, Nishad Maskara, Iris Cong, Xun Gao, Pedro Sales Rodriguez, Thomas Karolyshyn, Giulia Semeghini, Michael J. Gullans, Markus Greiner, Vladan Vuletić, and Mikhail D. Lukin. 2024. Logical quantum processor based on reconfigurable atom arrays. *Nature* 626, 7997 (2024), 58–65. https://doi.org/10.1038/s41586-023-06927-3

[8] Colin Campbell, Frederic T. Chong, Denny Dahl, Paige Frederick, Palash Goiporia, Pranav Gokhale, Benjamin Hall, Salahedeen Issa, Eric Jones, Stephanie Lee, Andrew Litteken, Victory Omole, David Owusu-Antwi, Michael A. Perlin, Rich Rines, Kaitlin N. Smith, Noah Goss, Akel Hashim, Ravi Naik, Ed Younis, Daniel Lobser, Christopher G. Yale, Benchen Huang, and Ji Liu. 2023. Superstaq: Deep Optimization of Quantum Programs . In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE Computer Society, Los Alamitos, CA, USA, 1020–1032. https://doi.org/10.1109/QCE57702.2023.00116

[9] Earl T. Campbell, Barbara M. Terhal, and Christophe Vuillot. 2017. Roads towards fault-tolerant universal quantum computation. *Nature* 549, 7671 (2017), 172–179. https://doi.org/10.1038/nature23460

[10] Center for High Throughput Computing. 2006. Center for High Throughput Computing. https://doi.org/10.21231/GNT1-HW21

[11] Berkeley Churchill, Rahul Sharma, JF Bastien, and Alex Aiken. 2017. Sound Loop Superoptimization for Google Native Client. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. Association for Computing Machinery, New York, NY, USA, 313–326. https://doi.org/10.1145/3037697.3037754

[12] Don Coppersmith. 2002. An approximate Fourier transform useful in quantum factoring. *arXiv preprint quant-ph/0201067* (2002).

[13] Marc G. Davis, Ethan Smith, Ana Tudor, Koushik Sen, Irfan Siddiqi, and Costin Iancu. 2020. Towards Optimal Topology Aware Quantum Circuit Synthesis. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*. 223–234. https://doi.org/10.1109/QCE49297.2020.00036

[14] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A Quantum Approximate Optimization Algorithm. arXiv:1411.4028 [quant-ph]

[15] Richard P. Feynman. 1982. Simulating physics with computers. *International Journal of Theoretical Physics* 21, 6 (1982), 467–488. https://doi.org/10.1007/BF02650179

[16] Craig Gidney, Noah Shutty, and Cody Jones. 2024. Magic state cultivation: growing T states as cheap as CNOT gates. arXiv:2409.17595 [quant-ph] https://arxiv.org/abs/2409.17595

[17] Daniel Gottesman. 1998. Theory of fault-tolerant quantum computation. *Phys. Rev. A* 57 (Jan 1998), 127–137. Issue 1. https://doi.org/10.1103/PhysRevA.57.127

[18] Lov K. Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* (Philadelphia, Pennsylvania, USA) *(STOC '96)*. Association for Computing Machinery, New York, NY, USA, 212–219. https://doi.org/10.1145/237814.237866

[19] W. K. Hastings. 1970. Monte Carlo Sampling Methods Using Markov Chains and Their Applications. *Biometrika* 57, 1 (1970), 97–109. http://www.jstor.org/stable/2334940

[20] Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. 2021. A verified optimizer for Quantum circuits. *Proc. ACM Program. Lang.* 5, POPL, Article 37 (jan 2021), 29 pages. https://doi.org/10.1145/3434318

[21] Fei Hua, Yanhao Chen, Yuwei Jin, Chi Zhang, Ari Hayes, Youtao Zhang, and Eddy Z. Zhang. 2021. AutoBraid: A Framework for Enabling Efficient Surface Code Communication in Quantum Computing. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO '21)*. Association for Computing Machinery, New York, NY, USA, 925–936. https://doi.org/10.1145/3466752.3480072

[22] IonQ. 2024. IonQ Forte. https://ionq.com/quantum-systems/forte.

[23] Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. 2019. TASO: optimizing deep learning computation with automatic generation of graph substitutions. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) *(SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 47–62. https://doi.org/10.1145/3341301.3359630

[24] Yuwei Jin, Zirui Li, Fei Hua, Tianyi Hao, Huiyang Zhou, Yipeng Huang, and Eddy Z. Zhang. 2023. Tetris: A Compilation Framework for VQA Applications in Quantum Computing. (9 2023). arXiv:2309.01905 [quant-ph]

[25] Daniel Kahneman. 2011. *Thinking, fast and slow.* macmillan.

[26] Chan Gu Kang and Hakjoo Oh. 2023. Modular Component-Based Quantum Circuit Synthesis. *Proc. ACM Program. Lang.* 7, OOPSLA1, Article 87 (apr 2023), 28 pages. https://doi.org/10.1145/3586039

[27] Ilyas Khan and Jenni Strabley. 2024. Quantinuum extends its significant lead in quantum computing, achieving historic milestones for hardware fidelity and Quantum Volume. https://www.quantinuum.com/blog/quantinuum-extends-its-significant-lead-in-quantum-computing-achieving-historic-milestones-for-hardware-fidelity-and-quantum-volume

[28] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by Simulated Annealing. *Science* 220, 4598 (1983), 671–680. https://doi.org/10.1126/science.220.4598.671 arXiv:https://www.science.org/doi/pdf/10.1126/science.220.4598.671

[29] Aleks Kissinger and John van de Wetering. 2020. PyZX: Large Scale Automated Diagrammatic Reasoning. In Proceedings 16th International Conference on *Quantum Physics and Logic,* Chapman University, Orange, CA, USA., 10-14 June 2019 *(Electronic Proceedings in Theoretical Computer Science, Vol. 318)*, Bob Coecke and Matthew Leifer (Eds.). Open Publishing Association, 229–241. https://doi.org/10.4204/EPTCS.318.14

[30] A Yu Kitaev. 1995. Quantum measurements and the Abelian stabilizer problem. *arXiv preprint quant-ph/9511026* (1995).

[31] 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* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 554–569. https://doi.org/10.1145/3503222.3507715

[32] Zikun Li, Jinjun Peng, Yixuan Mei, Sina Lin, Yi Wu, Oded Padon, and Zhihao Jia. 2024. Quarl: A Learning-Based Quantum Circuit Optimizer.

Proc. ACM Program. Lang. 8, OOPSLA1, Article 114 (apr 2024), 28 pages. https://doi.org/10.1145/3649831

[33] Daniel Litinski. 2018. A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery. *Quantum* (2018). https://doi.org/10.22331/q-2019-03-05-128

[34] Daniel Litinski and Felix von Oppen. 2018. Lattice Surgery with a Twist: Simplifying Clifford Gates of Surface Codes. *Quantum* 2 (May 2018), 62. https://doi.org/10.22331/q-2018-05-04-62

[35] Yuhao Liu, Shize Che, Junyu Zhou, Yunong Shi, and Gushu Li. 2024. Fermihedral: On the Optimal Compilation for Fermion-to-Qubit Encoding. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (<conf-loc>, <city>La Jolla</city>, <state>CA</state>, <country>USA</country>, </conf-loc>) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 382–397. https://doi.org/10.1145/3620666.3651371

[36] Zhengyang Liu, Stefan Mada, and John Regehr. 2024. Minotaur: A SIMD-Oriented Synthesizing Superoptimizer. *Proc. ACM Program. Lang.* 8, OOPSLA2, Article 326 (Oct. 2024), 25 pages. https://doi.org/10.1145/3689766

[37] Henry Massalin. 1987. Superoptimizer: a look at the smallest program. *SIGARCH Comput. Archit. News* 15, 5 (oct 1987), 122–126. https://doi.org/10.1145/36177.36194

[38] Matt McEwen, Lara Faoro, Kunal Arya, Andrew Dunsworth, Trent Huang, Seon Kim, Brian Burkett, Austin Fowler, Frank Arute, Joseph C. Bardin, Andreas Bengtsson, Alexander Bilmes, Bob B. Buckley, Nicholas Bushnell, Zijun Chen, Roberto Collins, Sean Demura, Alan R. Derk, Catherine Erickson, Marissa Giustina, Sean D. Harrington, Sabrina Hong, Evan Jeffrey, Julian Kelly, Paul V. Klimov, Fedor Kostritsa, Pavel Laptev, Aditya Locharla, Xiao Mi, Kevin C. Miao, Shirin Montazeri, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Alex Opremcak, Chris Quintana, Nicholas Redd, Pedram Roushan, Daniel Sank, Kevin J. Satzinger, Vladimir Shvarts, Theodore White, Z. Jamie Yao, Ping Yeh, Juhwan Yoo, Yu Chen, Vadim Smelyanskiy, John M. Martinis, Hartmut Neven, Anthony Megrant, Lev Ioffe, and Rami Barends. 2022. Resolving catastrophic error bursts from cosmic rays in large arrays of superconducting qubits. *Nature Physics* 18, 1 (2022), 107–111. https://doi.org/10.1038/s41567-021-01432-8

[39] Manasij Mukherjee, Pranav Kant, Zhengyang Liu, and John Regehr. 2020. Dataflow-based pruning for speeding up superoptimization. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 177 (nov 2020), 24 pages. https://doi.org/10.1145/3428245

[40] Yunseong Nam, Neil J Ross, Yuan Su, Andrew M Childs, and Dmitri Maslov. 2018. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Information* 4, 1 (2018), 1–12. https://doi.org/10.1038/s41534-018-0072-4

[41] Hartmut Neven and Julian Kelly. 2023. Suppressing quantum errors by scaling a surface code logical qubit. https://research.google/blog/suppressing-quantum-errors-by-scaling-a-surface-code-logical-qubit/

[42] Michael A Nielsen and Isaac Chuang. 2002. Quantum computation and quantum information.

[43] A. Paetznick, M. P. da Silva, C. Ryan-Anderson, J. M. Bello-Rivas, J. P. Campora III, A. Chernoguzov, J. M. Dreiling, C. Foltz, F. Frachon, J. P. Gaebler, T. M. Gatterman, L. Grans-Samuelsson, D. Gresh, D. Hayes, N. Hewitt, C. Holliman, C. V. Horst, J. Johansen, D. Lucchetti, Y. Matsuoka, M. Mills, S. A. Moses, B. Neyenhuis, A. Paz, J. Pino, P. Siegfried, A. Sundaram, D. Tom, S. J. Wernli, M. Zanner, R. P. Stutz, and K. M. Svore. 2024. Demonstration of logical qubits and repeated error correction with better-than-physical error rates. arXiv:2404.02280 [quant-ph] https://arxiv.org/abs/2404.02280

[44] Anouk Paradis, Jasper Dekoninck, Benjamin Bichsel, and Martin Vechev. 2024. Synthetiq: Fast and Versatile Quantum Circuit Synthesis. *Proc. ACM Program. Lang.* 8, OOPSLA1, Article 96 (apr 2024), 28 pages. https://doi.org/10.1145/3649813

[45] 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* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 514–528. https://doi.org/10.1145/3503222.3507739

[46] Jennifer Paykin, Albert T. Schmitz, Mohannad Ibrahim, Xin-Chuan Wu, and A. Y. Matsuura. 2023. PCOAST: A Pauli-Based Quantum Circuit Optimization Framework . In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE Computer Society, Los Alamitos, CA, USA, 715–726. https://doi.org/10.1109/QCE57702.2023.00087

[47] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. 2014. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications* 5, 1 (2014), 4213. https://doi.org/10.1038/ncomms5213

[48] Phitchaya Mangpo Phothilimthana, Aditya Thakur, Rastislav Bodik, and Dinakar Dhurjati. 2016. Scaling up Superoptimization. *SIGARCH Comput. Archit. News* 44, 2 (mar 2016), 297–310. https://doi.org/10.1145/2980024.2872387

[49] John Preskill. 2018. Quantum Computing in the NISQ era and beyond. *Quantum* 2 (Aug. 2018), 79. https://doi.org/10.22331/q-2018-08-06-79

[50] Nils Quetschlich, Lukas Burgholzer, and Robert Wille. 2024. MQT Predictor: Automatic Device Selection with Device-Specific Circuit Compilation for Quantum Computing. *ACM Transactions on Quantum Computing* (jun 2024). https://doi.org/10.1145/3673241 Just Accepted.

[51] Péter Rakyta and Zoltán Zimborás. 2022. Approaching the theoretical limit in quantum gate decomposition. *Quantum* 6 (May 2022), 710. https://doi.org/10.22331/q-2022-05-11-710

[52] Péter Rakyta and Zoltán Zimborás. 2022. Efficient quantum gate decomposition via adaptive circuit compression. (3 2022). arXiv:2203.04426 [quant-ph]

[53] Francisco J. R. Ruiz, Tuomas Laakkonen, Johannes Bausch, Matej Balog, Mohammadamin Barekatain, Francisco J. H. Heras, Alexander Novikov, Nathan Fitzpatrick, Bernardino Romera-Paredes, John van de Wetering, Alhussein Fawzi, Konstantinos Meichanetzidis, and Pushmeet Kohli. 2024. Quantum Circuit Optimization with AlphaTensor. arXiv:2402.14396 [quant-ph]

[54] Raimondas Sasnauskas, Yang Chen, Peter Collingbourne, Jeroen Ketema, Jubi Taneja, and John Regehr. 2017. Souper: A Synthesizing Superoptimizer. https://arxiv.org/abs/1711.04422

[55] Eric Schkufza, Rahul Sharma, and Alex Aiken. 2013. Stochastic superoptimization. *SIGPLAN Not.* 48, 4 (mar 2013), 305–316. https://doi.org/10.1145/2499368.2451150

[56] Yunong Shi, Nelson Leung, Pranav Gokhale, Zane Rossi, David I. Schuster, Henry Hoffmann, and Frederic T. Chong. 2019. Optimized Compilation of Aggregated Instructions for Realistic Quantum Computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 1031–1044. https://doi.org/10.1145/3297858.3304018

[57] P.W. Shor. 1994. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 124–134. https://doi.org/10.1109/SFCS.1994.365700

[58] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. 2020. t|ket⟩: a retargetable compiler for NISQ devices. *Quantum Science and Technology* 6, 1 (2020), 014003. https://doi.org/10.1088/2058-9565/ab8e92

[59] Luka Skoric, Dan E. Browne, Kenton M. Barnes, Neil I. Gillespie, and Earl T. Campbell. 2023. Parallel window decoding enables scalable

fault tolerant quantum computation. *Nature Communications* 14, 1 (2023), 7040. https://doi.org/10.1038/s41467-023-42482-1

[60] Ethan Smith, Marc Grau Davis, Jeffrey Larson, Ed Younis, Lindsay Bassman Oftelie, Wim Lavrijsen, and Costin Iancu. 2023. LEAP: Scaling Numerical Optimization Based Synthesis Using an Incremental Approach. *ACM Transactions on Quantum Computing* 4, 1, Article 5 (feb 2023), 23 pages. https://doi.org/10.1145/3548693

[61] IonQ staff. 2024. Getting started with Native Gates. https://ionq.com/docs/getting-started-with-native-gates.

[62] Rich Sutton. 2019. The Bitter Lesson. Available at http://www.incompleteideas.net/IncIdeas/BitterLesson.html.

[63] Robert R. Tucci. 2005. An Introduction to Cartan's KAK Decomposition for QC Programmers. arXiv:quant-ph/0507171 [quant-ph]

[64] Valentin F. Turchin. 1986. The concept of a supercompiler. *ACM Trans. Program. Lang. Syst.* 8, 3 (June 1986), 292–325. https://doi.org/10.1145/5956.5957

[65] John van de Wetering and Matt Amy. 2024. Optimising quantum circuits is generally hard. arXiv:2310.05958 [quant-ph] https://arxiv.org/abs/2310.05958

[66] Xin-Chuan Wu, Marc Grau Davis, Frederic T. Chong, and Costin Iancu. 2022. QGo: Scalable Quantum Circuit Optimization Using Automated Synthesis. arXiv:2012.09835 [quant-ph]

[67] Amanda Xu, Abtin Molavi, Lauren Pick, Swamit Tannu, and Aws Albarghouthi. 2023. Synthesizing Quantum-Circuit Optimizers. *Proc. ACM Program. Lang.* 7, PLDI, Article 140 (jun 2023), 25 pages. https://doi.org/10.1145/3591254

[68] Mingkuan Xu, Zikun Li, Oded Padon, Sina Lin, Jessica Pointing, Auguste Hirth, Henry Ma, Jens Palsberg, Alex Aiken, Umut A. Acar, and Zhihao Jia. 2022. Quartz: superoptimization of Quantum circuits. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (San Diego, CA, USA) *(PLDI 2022)*. Association for Computing Machinery, New York, NY, USA, 625–640. https://doi.org/10.1145/3519939.3523433

[69] Ed Younis, Costin C Iancu, Wim Lavrijsen, Marc Davis, Ethan Smith, and USDOE. 2021. Berkeley Quantum Synthesis Toolkit (BQSKit) v1. https://doi.org/10.11578/dc.20210603.2

[70] Ed Younis, Koushik Sen, Katherine Yelick, and Costin Iancu. 2021. QFAST: Conflating Search and Numerical Optimization for Scalable Quantum Circuit Synthesis . In *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE Computer Society, Los Alamitos, CA, USA, 232–243. https://doi.org/10.1109/QCE52317.2021.00041

[71] Charles Yuan and Michael Carbin. 2022. Tower: data structures in Quantum superposition. *Proc. ACM Program. Lang.* 6, OOPSLA2, Article 134 (oct 2022), 30 pages. https://doi.org/10.1145/3563297

[72] Charles Yuan and Michael Carbin. 2024. The T-Complexity Costs of Error Correction for Control Flow in Quantum Computation. *Proc. ACM Program. Lang.* 8, PLDI, Article 167 (jun 2024), 26 pages. https://doi.org/10.1145/3656397

[73] Alwin Zulehner, Alexandru Paler, and Robert Wille. 2019. An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 7 (2019), 1226–1236. https://doi.org/10.1109/TCAD.2018.2846658