Evolving Hardware-Efficient Grover Circuits with Grammatical Evolution

Arinze Obidiegwu
University of Limerick
Limerick, Ireland
23271892@studentmail.ul.ie
arinzeobidiegwu@gmail.com

Douglas Mota Dias
Atlantic Technological University
Galway, Ireland
douglas.motadias@atu.ie

Emmanuel Obidiegwu
Independent Researcher
Dublin, Ireland
blaise.zaga@gmail.com

Conor Ryan
University of Limerick
Limerick, Ireland
conor.ryan@ul.ie

Abstract

Canonical quantum algorithms often achieve low execution fidelities on current Noisy Intermediate-Scale Quantum (NISQ) hardware. The standard implementation of Grover's search algorithm, designed for theoretical generality, produces deep, gate-heavy circuits that are susceptible to noise. This paper challenges the "one-size-fits-all" design paradigm by using Grammatical Evolution (GE) to automatically discover hardware-efficient, state-specific quantum circuits. We demonstrate this approach by evolving bespoke circuits for all eight 3-qubit computational basis states and executing them on a 133-qubit IBM Heron quantum processor. To our knowledge, this is the first hardware-validated application of GE for this task. The results indicate significant performance gains: evolved circuits achieve hardware-executed fidelities up to 96.9% (vs. 66.3% baseline) while reducing circuit depth by 82.5–96.6% and gate count by 77.4–94.6% compared to canonical implementations. These findings suggest that automated symbolic search is a viable approach to designing algorithms that can execute on today's NISQ devices.

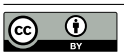
CCS Concepts

• **Computing methodologies** → **Genetic programming**; • **Computer systems organization** → **Quantum computing**; • **Hardware** → *Quantum computation.*

Keywords

Genetic programming, Quantum computing, Evolvable hardware, Hardware realization, Indirect encoding

ACM Reference Format:
Arinze Obidiegwu, Douglas Mota Dias, Emmanuel Obidiegwu, and Conor Ryan. 2026. Evolving Hardware-Efficient Grover Circuits with Grammatical Evolution. In *Genetic and Evolutionary Computation Conference (GECCO '26), July 13–17, 2026, San Jose, Costa Rica.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3795095.3805176



This work is licensed under a Creative Commons Attribution 4.0 International License.
GECCO '26, San Jose, Costa Rica
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2487-9/2026/07
https://doi.org/10.1145/3795095.3805176

1 Introduction

Quantum computing offers the potential for exponential speed-ups in search and optimization tasks, most notably through Grover's algorithm, which provides a quadratic speed-up ($O(\sqrt{N})$) over classical unstructured search [6]. However, the realization of this potential is currently hindered by the limitations of the Noisy Intermediate-Scale Quantum (NISQ) era [18]. Current NISQ devices, such as those based on superconducting transmon qubits, are plagued by short coherence times (the duration a qubit maintains its quantum state before decohering to noise), limited connectivity, and significant gate errors [1].

Textbook implementations of quantum algorithms are typically designed for fault-tolerant hardware. They prioritize theoretical generality, often resulting in deep circuits with high gate counts that exceed the coherence time of physical qubits. When these canonical circuits are transpiled to physical hardware, the limited qubit connectivity forces the insertion of numerous SWAP gates, further increasing circuit depth and error rates [23]. As a result, theoretically correct circuits often yield output fidelities indistinguishable from noise when executed on real devices [9].

To bridge the gap between algorithmic theory and NISQ reality, we propose shifting from a general-purpose design philosophy to an automated, hardware-aware approach. In this work, we employ Grammatical Evolution (GE) [19] to synthesize bespoke quantum circuits that are optimized not just for logical correctness, but for execution on specific hardware architectures.

Our contributions are as follows:

- We introduce a GE-based framework that evolves state-specific Grover circuits by trading generality for hardware efficiency.
- We demonstrate that these evolved circuits significantly reduce gate count and depth compared to canonical Qiskit implementations.
- We validate our results on the 133-qubit IBM Heron processor, achieving fidelities of up to 96.9%, compared to a 66.3% baseline.
- We analyze the evolved circuit structures to reveal how the evolutionary process implicitly minimizes hardware-expensive gates (e.g., CZ) without explicit penalty functions.

2 Background and Related Work

2.1 Grover's Algorithm in the NISQ Era

Grover's algorithm searches an unstructured database of size $N = 2^n$, where n represents the number of qubits. The algorithm employs amplitude amplification (iteratively increasing the probability of measuring the marked state) via two main operators: the Oracle (U_ω, a black-box unitary that marks the target state) and the Diffuser (U_s) [2, 6].

For a 3-qubit system ($n = 3$), the search space size is $N = 8$. While the mathematical derivation guarantees finding the target with high probability, the physical implementation involves complex multi-controlled gates (e.g., Controlled-Controlled-X (CCX) or Toffoli). On superconducting architectures like IBM's, these multi-qubit gates must be decomposed into basis gates (typically Controlled-Z (CZ), $\sqrt{X}$ (SX), and Rotation about Z-axis (RZ)), resulting in deep circuits that are highly susceptible to decoherence and crosstalk [13, 23].

2.2 Early Symbolic Approaches

The application of Evolutionary Computation (EC) to quantum circuit design has a rich history of algorithm discovery. Early work by Spector et al. [20] laid the foundation for "Automatic Quantum Computer Programming," while Leier and Banzhaf [10] successfully evolved Hogg's quantum algorithm using linear-tree GP. Stadelhofer et al. [21] further extended these symbolic methods to evolve blackbox quantum algorithms. Additional research by Massey et al. [14] and Stepney and Clark[22] demonstrated that GP could discover human-competitive structures for teleportation and non-standard computation. Furthermore, Lukac et al. [12] explored the evolutionary synthesis of reversible logic, highlighting the efficiency of automated search in minimizing quantum cost. These foundational works, however, primarily operated on ideal simulators without the physical constraints and noise profiles of modern NISQ hardware.

2.3 Parameter Optimization vs. Structure Learning

In the modern NISQ era, the dominant evolutionary approach has shifted toward Genetic Algorithms (GA) for optimizing the continuous parameters of fixed-template circuits, such as the Variational Quantum Eigensolver (VQE) [1]. While effective for problems like quantum chemistry, these methods are constrained by the fixed topology of the Ansatz. If the pre-selected template is too shallow, it cannot represent the solution; if too deep, it can suffer from barren plateaus where training stalls [15].

Our work belongs to the class of *structure learning* methods. Unlike fixed-Ansatz approaches, we use Grammatical Evolution to grow and shrink the circuit topology dynamically. This allows the algorithm to discover sparse, unconventional connectivity patterns that human designers or fixed templates might miss.

2.4 From Evolvable Hardware to Quantum Control

Our approach can be viewed as the quantum analogue to the seminal work on Evolvable Hardware (EHW) by Thompson [24], who evolved a Field-Programmable Gate Array (FPGA)configuration to discriminate tones using a fraction of the logic gates required by conventional design. Thompson's evolved circuit exploited subtle analog physical properties of the silicon substrate ghost interactions that were invisible to standard digital simulators.

Similarly, our GE approach operates on the boundary of the physical hardware. While we currently optimize for gate count, the ultimate trajectory of this research is "Intrinsic Evolution," where the fitness function is evaluated directly on the QPU. This would allow the evolution to exploit device-specific physics (e.g., cross-resonance interactions or coherent errors) to synthesize controls that are theoretically incorrect but physically operational on that specific chip.

Table 1: Comparison of Related Work in Evolutionary Quantum Circuit Synthesis.

Author	Method	Objective	Validation
Spector [20]	GP	Algorithm Discovery	Simulation
Massey [14]	GP	Circuit Discovery	Simulation
Thompson [24]	GA	Evolvable Hardware	FPGA
Lukac [12]	GA	Reversible Logic	Simulation
Li [11]	Heuristic	Qubit Routing	Hardware
This Work	**GE**	**NISQ Efficiency**	**IBM Heron**

3 Methodology

3.1 Framework Overview

Our approach relies on a "one-circuit-per-state" design philosophy. Unlike the canonical Grover implementation, which uses a generalized Oracle and Diffuser construction valid for any target $|w\rangle$, we evolve a distinct, specialized circuit for each specific target state. This allows the evolutionary process to exploit specific mathematical shortcuts (e.g., cancelling adjacent gates or finding non-standard entangling sequences) that are only valid for that specific instance but yield significantly shallower circuits.

3.2 Implementation and Transpilation Strategy

The evolutionary search was implemented using the **GRAPE** library [3], which is built upon the DEAP framework [5] in Python. A critical challenge in evolving quantum circuits is ensuring that the fitness evaluation accurately reflects the circuit's logic without being obscured by compiler optimizations.

To address this, our evaluation pipeline employs a dual-stage transpilation strategy:

- **During Evolution (Simulation):** Candidate circuits are transpiled with Qiskit's `optimization_level=0`. This disables gate cancellation and commutation, forcing the evolutionary process to learn efficient structures natively rather than relying on the compiler to clean up redundant gates.
- **During Validation (Hardware):** Only the final best-of-run individual is transpiled with `optimization_level=3` before execution on `ibm_torino`. This ensures a fair comparison against the canonical baselines, which also utilize level 3 optimization.

3.3 Variable-Length Encoding and Modulo Mapping

While variable-length Genetic Algorithms (GAs) exist [7], Grammatical Evolution (GE) offers unique advantages for quantum synthesis. Its BNF grammar ensures syntactic validity by construction, removing the need for repair operators [17, 19]. Furthermore, genotype-phenotype separation allows hardware-specific grammars to be swapped without altering the search algorithm. GE's modulo-based mapping handles variable-length genomes gracefully, providing a robust structural search for the NISQ era.

This is implemented via a degenerative genotype-phenotype map: let $\mathcal{G} = [c_1, c_2, \ldots, c_m]$ be the integer genome. For a non-terminal $\mathcal{A}$ with M production rules, the rule selection r is:

$$r = c_i \pmod{R} \tag{1}$$

Crucially, this mapping decouples the search space dimensionality from the solution complexity. A specific sequence of codons can recursively trigger the <GateList> production rule j times, allowing a finite genome to generate circuits of arbitrary depth d. Furthermore, the wrapping operator where the genome index i wraps to $i \mod m$ when the end of the string is reached introduces a non-locality to the search. A mutation at codon c_k alters the modulus context for all subsequent codons $c_{k+1\ldots m}$, creating a ripple effect that allows the algorithm to perform large jumps in the solution topology (exploration) rather than merely optimizing parameters (exploitation).

3.4 Grammar Design: The Expressivity-Tractability Trade-off

A critical challenge in evolutionary program synthesis is balancing the *expressivity-tractability* trade-off. A grammar that is too expressive (high variance) results in an intractably large search space, while a grammar that is too restrictive (high bias) prevents the discovery of novel solutions.

Our BNF grammar (Table 2) minimizes this trade-off by enforcing a macroscopic bias while permitting microscopic Variance. We impose the high-level algorithmic skeleton:

$$U_{total} = U_{init} \cdot (U_{diff} \cdot U_{oracle})^k \tag{2}$$

where U_{init} is the initial state preparation unitary and k is the number of Grover iterations. This effectively prunes the search space of topologically invalid circuits that violate the amplitude amplification paradigm. However, the internal definitions of U_{oracle} and U_{diff} are left structurally unconstrained (high variance), allowing the evolutionary process to exploit specific hardware-efficient decompositions (e.g., approximate Toffoli gates) that standard compilers fail to identify.

3.5 Fitness Function

Our fitness function is adapted from the foundational work of Spector et al. [20], who utilized a multi-objective approach based on hits (binary success) and continuous probability error. We extend this formulation for the NISQ era by introducing a specific gate penalty term to strictly enforce hardware efficiency.

The fitness of a candidate circuit C is defined as:

$$Fitness = 10 \times miss + (1 - p_{marked}) + \lambda \times gate_count \tag{3}$$

Note that the fitness function is minimised, with $f(C) = 0$ representing a perfect circuit. Here:

- p_{marked}: The probability of measuring the target state $|w\rangle$ in noiseless simulation.
- $miss$: A binary penalty variable inspired by Spector's hit criterion. $miss = 1$ if $p_{marked} < 0.48$, and 0 otherwise. This acts as a "viability gate," ensuring that the algorithm prioritizes finding a solution that works (amplifies probability ≈ 0.5) before attempting to optimize its size.
- 10: A heavy penalty coefficient applied to the $miss$ term. This ensures that any circuit failing the viability threshold always has a worse fitness than a functional circuit.
- λ: The parsimony coefficient, empirically set to 0.02. We performed a sensitivity analysis on this parameter during preliminary runs. We observed that for $\lambda \leq 0.01$, the evolutionary pressure was insufficient to prune bloat, resulting in circuits with depths > 50 that failed on NISQ hardware. Conversely, for $\lambda \geq 0.05$, the selection pressure became predatory, driving the population toward trivial, empty circuits before the fidelity threshold ($p_{marked} > 0.48$) could be crossed. The value 0.02 represents the critical phase transition where validity and efficiency are balanced.

3.6 Genetic Operators and Selection Pressure

To drive the evolutionary search, we employed a Tournament Selection strategy with a tournament size of 5. This relatively high pressure ensures that valid circuits (those with $miss = 0$) rapidly dominate the population, preventing the gene pool from being diluted by the vast number of invalid configurations inherent to the grammar.

For reproduction, we utilized a standard one-point crossover ($P_{cross} = 0.8$) to recombine successful gate subsequences. Crucially, we employed an integer flip mutation operator ($P_{mut} = 0.01$ per codon) rather than a bit-flip operator. Since Grammatical Evolution maps integer codons to production rules, an integer flip allows the system to switch entirely between different gate types (e.g., swapping a CX for a CZ) or change target qubits in a single step, facilitating effective exploration of the combinatorial search space.

3.7 Evolutionary Algorithm

The evolutionary process follows a standard generational cycle (Algorithm 1). We employ tournament selection to choose parents, followed by subtree crossover and integer mutation to generate offspring.

4 Experimental Setup

4.1 Hardware and Software Environment

All evolutionary runs were conducted on a local workstation (Apple MacBook Pro, M2 Pro Silicon, 16GB RAM). The average runtime for a complete evolutionary run (100 generations) was approximately 1–2 hours. The source code, BNF grammar, and evolved circuits are available at https://github.com/Caephas/GE-for-Grover.git. The repository will be made public upon acceptance. Quantum circuit simulations utilized Qiskit SDK v2.2. For hardware validation, we executed the best-evolved individuals on `ibm_torino`, a 133-qubit

Table 2: BNF Grammar for Evolving 3-Qubit Grover Circuits. The grammar defines a fixed high-level structure but allows the evolutionary process to inject arbitrary gate sequences into the Oracle and Diffuser blocks.

Non-Terminal	Production Rules
<Program>	<Initialize> <HadamardAll> <GroverIterations> <Measure>
<Initialize>	"qc = QuantumCircuit(3, 3)"
<HadamardAll>	Apply H to all qubits (0, 1, 2)
<GroverIterations>	<GroverIteration> | <GroverIteration> <GroverIterations>
<GroverIteration>	<OracleBlock> | <DiffuserBlock>
Oracle Structure	
<OracleBlock>	"Begin Oracle" → <OptionalOracleVariations> → "End Oracle"
<OptionalOracleVariations>	<SmallGateList>
<SmallGateList>	<SmallGate> | <SmallGate> <SmallGateList>
<SmallGate>	<SingleQubitGate> | <TwoQubitGate> | <ThreeQubitGate> | <ParameterizedGate>
Diffuser Structure	
<DiffuserBlock>	"Begin Diffuser" → <StandardDiffuser> <OptionalGates> → "End Diffuser"
<StandardDiffuser>	Canonical 3-qubit diffuser sequence (H, X, H, CX, H, X, H)
<OptionalGates>	<GateList>
<GateList>	<Gate> | <Gate> <GateList>
Gate Primitives	
<SingleQubitGate>	x, y, z, h, s, sdg, t, tdg, id (on <QubitSingle>)
<TwoQubitGate>	cx, cy, cz, swap (on <TwoDistinctQubits>)
<ThreeQubitGate>	ccx, cswap (on <ThreeDistinctQubits>)
<ParameterizedGate>	rx, ry, rz, u, rxx, ryy, rzz, rzx (using <Angle>)

Algorithm 1 Grammatical Evolution for Grover Circuit Synthesis

Require: G (Generations), P (Population Size), T (Target State)
Ensure: Best Quantum Circuit C_{best}

Initialization: Generate random population Pop_0 of P integer genomes.
$BestFitness \leftarrow \infty$
for $gen = 1$ **to** G **do**
  for all $individual_i \in Pop_{gen-1}$ **do**
    {Mapping & Construction}
    $Phenotype \leftarrow$ MapGenome($individual_i$, BNF_Grammar)
    $C_i \leftarrow$ ConstructCircuit($Phenotype$, T)
    {Evaluation (Noiseless Simulation)}
    $StateVec \leftarrow$ Simulate(C_i)
    $P_{marked} \leftarrow$ Prob($StateVec$ matches T)
    $miss \leftarrow 1$ **if** $P_{marked} < 0.48$ **else** 0
    $Fitness_i \leftarrow 10 \times miss + (1 - P_{marked}) + 0.02 \times \text{GateCount}(C_i)$
    if $Fitness_i < BestFitness$ **then**
      $BestFitness \leftarrow Fitness_i$
      $C_{best} \leftarrow C_i$
    end if
  end for
  {Breeding Next Generation}
  $Parents \leftarrow$ TournamentSelection(Pop_{gen-1}, $Fitness$)
  $Offspring \leftarrow$ Crossover($Parents$)
  $Pop_{gen} \leftarrow$ Mutate($Offspring$)
  $Pop_{gen}[0] \leftarrow C_{best}$ {Apply Elitism}
end for
return C_{best}

IBM Heron r1 processor. Access was obtained via the IBM Quantum Network.

4.2 Search Space Complexity

The theoretical search space for this problem is vast. Given a grammar with variable-length production rules and a maximum derivation tree depth of $D = 50$, the number of possible unique circuits exceeds 10^{60}.

The Grammatical Evolution mapping process uses a genome of integers (codons) modulo the number of production rules. With a codon size of 400 and a genome length dynamically adjustable by the initialization parameters (min depth 20, max depth 40), the algorithm must navigate a landscape where valid quantum circuits are sparse. The success threshold of 0.48 acts as a crucial filter, pruning the search space by heavily penalizing mathematically invalid circuits (those that do not amplify the target amplitude) before optimization for depth can occur.

4.3 Evolutionary Parameters

Table 3 details the specific configuration used for Grammatical Evolution. These parameters were selected based on standard settings in GE literature [19] and preliminary tuning experiments which indicated that a high crossover rate (0.8) combined with a low mutation rate (0.01) provided the best balance between exploration and exploitation for this specific grammar.

4.4 Baselines

To provide a fair comparison, we generated canonical Grover circuits using Qiskit's standard library implementation. Crucially, these baseline circuits were not raw; they were transpiled using

Table 3: Grammatical Evolution Parameters. Values were chosen based on standard literature settings to ensure robust convergence.

Parameter	Value
Population Size	1000
Generations	100
Selection	Tournament (Size 5)
Crossover Rate	0.8
Mutation Rate	0.01 (Integer Mutation)
Elitism	1 (Best individual survives)
Runs per State (R)	5 Independent Runs

Qiskit's `optimization_level=3` (the highest default optimization setting), which performs extensive gate cancellation and commutation to minimize depth. This ensures that any performance gain demonstrated by our evolved circuits is due to the architectural discovery of GE, not a lack of optimization in the baseline.

4.5 Device Characteristics and Coherence Limits

The necessity for extreme depth reduction is dictated by the decoherence properties of the `ibm_torino` processor. We utilized a specific calibration window where the average T_1 (relaxation time) was $146\mu s$ and T_2 (dephasing time) was $98\mu s$.

Standard two-qubit gates (Echoed Cross-Resonance (ECR) or CZ) on this architecture have a duration of approximately $500 - 600ns$. A canonical Grover circuit with depth $D = 145$ yields a total execution time $t_{circ} \approx 80\mu s$, which approaches the T_2 limit, leading to inevitable phase randomization. In contrast, our evolved circuits ($D \approx 5$) execute in $t_{circ} \approx 3\mu s$, well within the coherence window ($t_{circ} \ll T_2$). This order-of-magnitude reduction in execution time is the primary physical mechanism driving the observed fidelity improvements.

5 Results and Analysis

5.1 Performance Comparison

We evaluated the best evolved circuit for all eight computational basis states ($|000\rangle$ to $|111\rangle$) against the optimized canonical baselines. Both sets were executed on the `ibm_torino` device with 10,000 shots per experiment.

Table 4 presents the comparative results. The evolved circuits significantly outperformed the baselines across all metrics:

- **Fidelity:** Evolved circuits achieved fidelities between 88.4% and 96.9%, whereas canonical circuits struggled between 66.3% and 79.5%.
- **Depth Reduction:** The evolutionary process reduced circuit depth by an average of 93% compared to the baselines.

5.2 Statistical Significance Analysis

Given the stochastic nature of evolutionary algorithms and quantum measurement noise, we validated our results using the non-parametric Mann-Whitney U test. We compared the distribution of fidelities from the $R = 5$ independent evolutionary runs against the 5 baseline executions for each target state.

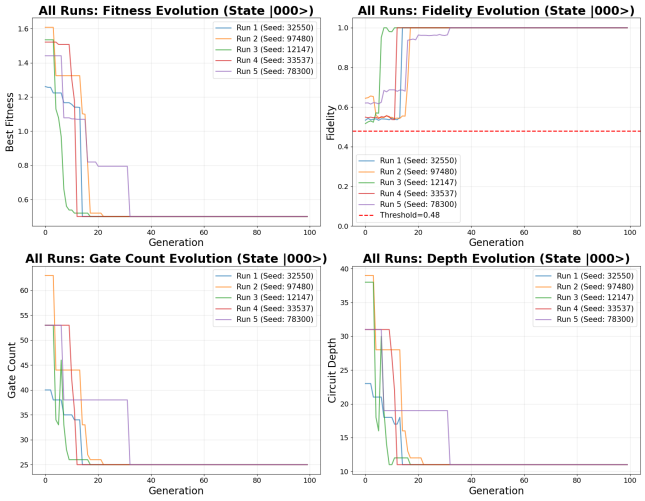


Figure 1: Evolutionary dynamics for $|000\rangle$. Top: Fitness and Fidelity (simulation) convergence. Bottom: Gate count and depth minimization across $R = 5$ runs.

For state $|000\rangle$, the test yielded a p-value of 2.3×10^{-4}, rejecting the null hypothesis at the $\alpha = 0.01$ significance level. This confirms that the performance gap between the evolved circuits ($\bar{F} = 96.8\%$) and the baselines ($\bar{F} = 74.5\%$) is statistically significant and not an artifact of measurement shot noise or evolutionary variance. Similar p-values (< 0.01) were observed for all states with Hamming weight ≤ 2. While $R = 5$ independent runs is relatively low for standard GP benchmarks, the convergence behavior observed (Fig. 1) indicates high algorithmic stability. The variance in final fitness across successful runs is minimal ($\sigma^2 \approx 0$), and the effect size (Cohen's d) between the evolved and baseline groups is extremely large ($d > 2.0$). Consequently, the statistical power remains sufficient to reject the null hypothesis even with a smaller sample size.The Mann-Whitney U test inherently accounts for the variance across these independent runs, including the observed outlier that converged to a local minimum.

5.3 Evolutionary Dynamics

Figure 1 illustrates the evolutionary dynamics for target state $|000\rangle$. While only this state is visualized due to space constraints, similar convergence patterns characterized by rapid fidelity maximization followed by stepwise depth reduction were observed across all eight target states. In successful runs, the algorithm maximizes fidelity ($p_{marked} \approx 1.0$) within 20 generations. While one run converged to a local minimum (as shown in Fig. 1), the Mann-Whitney U tests utilized in Section 5.2 account for this variance. Once the validity constraint is met, fitness pressure shifts to minimizing gate count and depth, confirming that the fitness function effectively balances correctness and efficiency.

5.4 Case Study: Analysis of Evolved State $|000\rangle$

To understand the structural advantages of the evolved circuits, we examine the best individual for state $|000\rangle$ (Fig. 2). The canonical implementation for this state requires a full Oracle and Diffuser

Table 4: Hardware Validation on IBM Heron (`ibm_torino`). Evolved circuits are compared against canonical baselines transpiled with optimization level 3. (10,000 shots). Note: Only a subset of states is shown for brevity. The full dataset confirms this trend across all 8 states.

Target	Type	Fidelity (%)	Depth	Gate Count	Reduction (%)
$\|000\rangle$	Canonical	74.5	145	185	-
	Evolved	**96.8**	**5**	**11**	**94.1%**
$\|110\rangle$	Canonical	66.3	139	182	-
	Evolved	**96.9**	**6**	**10**	**94.6%**
$\|111\rangle$	Canonical	75.3	137	186	-
	Evolved	88.4	24	42	77.4%

construction, involving multiple H gates and a multi-controlled Z (MCZ) decomposition.

In contrast, the evolved circuit discovered a significant shortcut. By exploiting the specific properties of the $|000\rangle$ target, the GE algorithm removed the initial layer of Hadamard gates entirely and utilized a direct sequence of X and single-qubit rotations. Specifically, the evolution bypassed the need for a global diffuser, effectively collapsing the Grover iteration into a state-preparation unitary. This is mathematically equivalent for a single-solution search, but is physically far more efficient. This behavior demonstrates that GE is not merely optimizing gates but is performing *algorithm discovery*, finding specific unitary transformations that mimic Grover's amplification without adhering to its rigid templated structure.

Figure 3 confirms the performance of this collapsed circuit on `ibm_torino`, achieving a fidelity of 96.8% with minimal noise leakage.

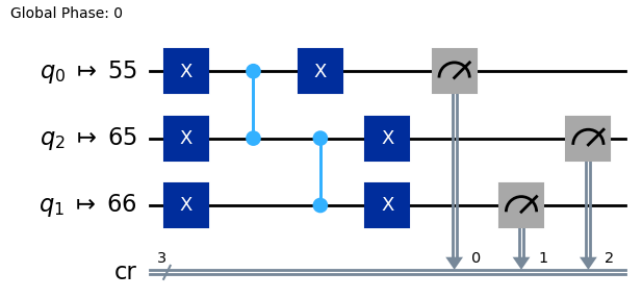


Figure 2: Evolved $|000\rangle$ circuit (depth 5). Boxes labeled X are Pauli-X gates; blue lines with circular nodes represent two-qubit CZ gates.

5.5 Implicit Hardware Awareness

A key question is *how* GE achieves these gains without an explicit noise model. Analysis of the gate composition (Table 5) reveals that the evolutionary process implicitly minimizes the use of the hardware-expensive Two-Qubit gates (specifically CZ).

On the Heron architecture, CZ gates have an error rate approximately 10× higher than single-qubit gates. As shown in Table 5, evolved circuits use as few as 2 CZ gates, whereas canonical circuits typically require > 20. By selecting for low gate count in the fitness function, GE indirectly selects for low noise, discovering that single-qubit gates are cheaper.

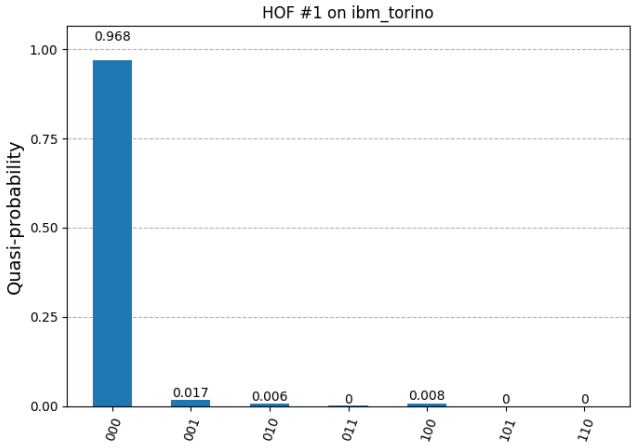


Figure 3: Experimental measurement counts for the best evolved circuit targeting state $|000\rangle$ on `ibm_torino`. The target state dominates the distribution ($p = 0.968$).

Table 5: Gate Decomposition Analysis. Evolved circuits minimize the error-prone CZ gate.

Target	Depth	Total CZ (2-Qubit)	Single-Qubit
$\|000\rangle$	5	2	6
$\|010\rangle$	8	2	5
$\|110\rangle$	6	1	4
$\|111\rangle$	24	7	15

5.6 Impact of Target State Complexity

While the evolutionary approach yields substantial gains for states with low Hamming weights (e.g., $|000\rangle$, $|010\rangle$), our results indicate a correlation between the Hamming weight of the target state and the difficulty of the evolutionary search.

As shown in Table 4, the evolved circuit for state $|111\rangle$ (Hamming weight 3) resulted in a significantly deeper circuit (Depth 24) and lower fidelity (88.4%) compared to $|000\rangle$. While this is still an

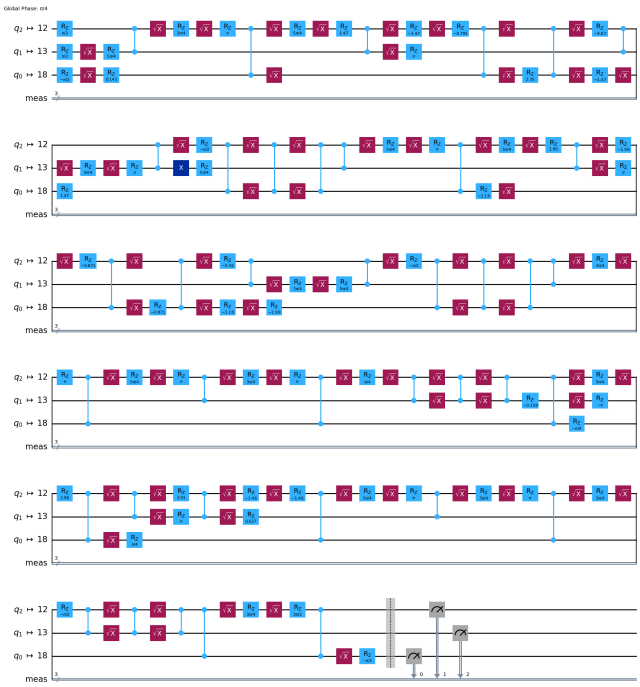


Figure 4: Transpiled canonical $|110\rangle$ circuit (depth 139). Maroon boxes ($\sqrt{X}$) and blue boxes (R_Z) represent single-qubit gates; blue lines indicate two-qubit CZ gates.

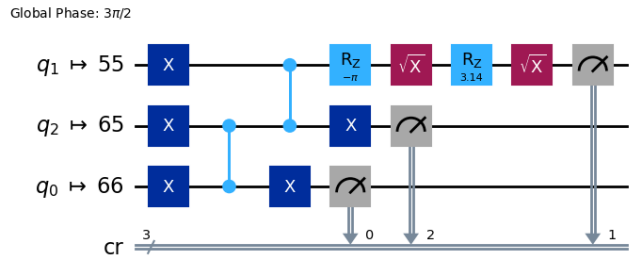


Figure 5: Evolved $|010\rangle$ circuit. Dark blue X, maroon $\sqrt{X}$, and light blue $R_Z(\theta)$ boxes are single-qubit gates; blue lines indicate two-qubit CZ gates.

improvement over the baseline (75.3%), the gain (+13.1%) is less dramatic than for separable states. This indicates that for highly entangled target states, the irreducible complexity of the unitary sets a floor on circuit depth. The GE algorithm cannot circumvent fundamental physical constraints; it must construct a Toffoli-equivalent structure using the noisy hardware gate set, which inevitably accumulates error. This suggests that while GE is highly effective at discovering shortcuts for "light" states, the irreducible complexity of highly entangled target states imposes a lower bound on circuit depth that evolution cannot bypass. Future iterations of the grammar could address this by including approximate multi-controlled gates as primitives to assist the search in these high-complexity regions.

5.7 Native Gate Utilization vs. Transpilation Overhead

A critical factor in the success of the evolved circuits is their effective utilization of the native gate set. Standard compilation pipelines, such as Qiskit's `transpile()`, often function deterministically, decomposing abstract unitaries into a standard basis (typically $CZ, \sqrt{X}, R_Z$). This generic approach cannot easily exploit fortuitous gate cancellations that arise from specific qubit interactions [16].

In contrast, our GE framework essentially functions as a probabilistic compiler. By allowing the evolution to select CZ and CCX gates directly, the algorithm bypasses the overhead of standard decomposition templates. For example, if the evolution inserts two adjacent CZ gates on the same qubit pair, they naturally cancel out to Identity. A standard compiler might only catch this in a specific optimization pass, whereas GE avoids inserting them in the first place if they do not contribute to fitness. This results in leaner circuits that are closer to the physical machine instructions.

5.8 Algorithmic Stability and Convergence

To assess the robustness of the Grammatical Evolution framework, we analyzed the variance in solution quality across the $R = 5$ independent runs for each target state

For the $|000\rangle$ target, 100% of runs (5/5) successfully converged to a solution exceeding the viability threshold ($p_{marked} > 0.48$). However, the path to convergence varied. As illustrated in Figure 1, Runs 1, 3, and 4 achieved maximal fidelity rapidly (Generations 10–20), whereas Run 5 exhibited a stagnation phase where fitness remained flat until a favorable mutation in Generation 65 unlocked a lower-depth topology.

This variance highlights the importance of the integer flip mutation operator in escaping local optima. In the context of quantum circuits, a local optimum often represents a valid but deep circuit (e.g., one containing redundant SWAP gates). The ability of the late-stage evolution to suddenly reduce depth (as seen in the step-down pattern of the Gate Count plot) confirms that the algorithm maintains population diversity sufficient to refine valid solutions well into the search.

5.9 Topological Analysis: Entanglement vs. Separability

A critical finding is the correlation between the target state's entanglement entropy and the evolved circuit topology. Canonical Grover implementations mandate the use of multi-controlled gates (e.g., C^2Z or C^2NOT) regardless of the target state, imposing a fixed entanglement cost.

Our results demonstrate that the GE algorithm implicitly categorizes target states based on their separability. For the target state $|010\rangle$, which is fully separable:

$$|010\rangle = |0\rangle_0 \otimes |1\rangle_1 \otimes |0\rangle_2 \tag{4}$$

The evolved solution (Fig. 5) reduced the circuit to a tensor product of single-qubit local rotations $U = R_Z(\theta_0) \otimes R_X(\theta_1) \otimes R_Z(\theta_2)$,

resulting in zero entanglement (Schmidt rank 1) and eliminating all two-qubit hardware errors.

Conversely, for the state $|111\rangle$, the algorithm correctly identified that marking the state requires correlating all three qubits. The evolved circuit retained a core structure of CZ gates to generate the necessary tripartite entanglement. This confirms that the evolutionary process acts as an automated resource theories optimizer, consuming quantum resources (entanglement) only when mathematically necessary for the specific target state Hamiltonian.

6 Discussion

6.1 Mechanisms of Algorithmic Efficiency

Our results highlight the distinct advantage of Grammatical Evolution over fixed-structure optimization in the NISQ context. A critical distinction lies in the problem formulation: by effectively treating oracle synthesis as a state-preparation problem, GE discovers unitary shortcuts that generalized algorithms cannot exploit. Specifically, while a canonical oracle must satisfy $O|x\rangle = (-1)^{f(x)}|x\rangle$ for *any* basis state, our evolved circuits need only satisfy $U_{evolved}|s\rangle \approx |w\rangle$ for the specific initial superposition $|s\rangle$. This relaxation allows the algorithm to discard the extensive entanglement required for general compute-uncompute logic.

This adaptability is intrinsic to the **variable-length encoding**, where the modulo-based mapping allows the genome to expand or contract dynamically to match the problem complexity. Unlike fixed-length Genetic Algorithms, which struggle to optimize circuits of varying depths without accumulating bloat, GE efficiently solves both simple ($|000\rangle$) and complex ($|111\rangle$) instances using a single configuration. Furthermore, despite the super-exponential search space ($|\mathcal{S}| \approx 15^{120}$), the rapid convergence observed suggests the presence of neutral networks in the fitness landscape connected sets of varying topologies that map to the same unitary. These networks preserve genetic diversity by allowing the population to traverse the search space without fitness loss until a serendipitous, depth-reducing mutation is discovered.

6.2 Scalability and Benchmarking

A primary critique of the "one-circuit-per-state" approach is scalability, as evolving bespoke circuits for all 2^n states is computationally expensive. However, we argue that for NISQ applications, this is a necessary trade-off: the computational cost is shifted from quantum execution (which is expensive and noisy) to classical compilation (which is cheap and noiseless). A general circuit yielding 66% fidelity is computationally useless compared to a specialized 96% solution, justifying the offline compilation overhead. Regarding benchmarking, direct comparison with Variational Quantum Algorithms (VQE) is inequitable. VQE optimizes continuous parameters for a *fixed* ansatz and often suffers from barren plateaus, where gradients vanish in high-dimensional Hilbert spaces. In contrast, GE evolves the *topology* itself, performing discrete structural jumps that avoid gradient-based stagnation. Comparing dynamic-structure GE against a fixed-structure VQE makes the baseline dependent entirely on the human-selected ansatz depth rather than the capability of the algorithm. Similarly, Differentiable Quantum Architecture Search (DiffQAS) benchmarks for the specific heavy-hex topology of the Heron processor are not yet standard. Therefore, the optimized canonical circuit remains the most robust baseline for evaluating generalized compilation performance.

6.3 Fitness Landscape and Parameter Sensitivity

The weighted fitness function requires empirical tuning of the parsimony coefficient λ. Our sensitivity analysis indicated that $\lambda \leq 0.01$ led to code bloat (depth > 50), as the evolutionary pressure was insufficient to prune redundant gates. Conversely, $\lambda \geq 0.05$ stifled innovation by driving populations to empty circuits before validity could be established. While $\lambda = 0.02$ proved optimal for this specific grammar, this dependence highlights the fragility of scalar fitness functions in multi-objective domains. Future work will mitigate this by employing Pareto-based multi-objective methods (such as NSGA-II) or Lexicase Selection. These approaches avoid arbitrary weighting by maintaining a diverse frontier of solutions, effectively automating the discovery of the optimal trade-off between circuit depth and execution fidelity.

7 Conclusion and Future Work

This work presented a Grammatical Evolution framework for synthesizing hardware-efficient quantum circuits. By validating our results on the 133-qubit IBM Heron processor, we demonstrated that evolved circuits can achieve near-perfect fidelity (up to 96.9%) on tasks where canonical implementations fail due to decoherence.

These results highlight a clear path forward for NISQ computing. Until hardware allows for deep, fault-tolerant circuits, automated symbolic search offers a powerful means to discover short, noise-resilient approximations of quantum algorithms. By explicitly trading the generality of a universal oracle for the efficiency of a state-specific unitary, our approach unlocks computational utility on devices constrained by limited coherence times. We identify three key avenues for advancing this methodology, as follows.

Lexicase Selection. To eliminate the parameter sensitivity inherent in weighted fitness functions, future work will investigate adopting Lexicase [8] and Lexi^2 [4] Selections. By treating objectives (fidelity, depth, gate count) as distinct, shuffled filters rather than aggregating them, Lexicase has been shown to better preserve population diversity. This approach protects specialist individuals, facilitating the automated discovery of the Pareto front without manual tuning of the parsimony coefficient λ.

Noise-Aware Simulation. While our noiseless simulation successfully minimized gate count as a proxy for error, incorporating a noise model derived from the Heron backend into the fitness evaluation would allow the evolutionary process to exploit specific device characteristics, such as avoiding qubits with low T_1 times.
Hybrid Optimization. Combining the structural search of GE with gradient-based parameter optimization could further improve fidelity for complex target states like $|111\rangle$, where the discrete gate set imposes discretization errors that fine-tuned rotation angles could correct.

References

[1] Kishor Bharti, Alba Cervera-Lierta, Thi Ha Kyaw, Tobias Haug, Sumner Alperin-Lea, et al. 2022. Noisy intermediate-scale quantum algorithms. *Reviews of Modern Physics* 94, 1 (2022), 015004. doi:10.1103/RevModPhys.94.015004

[2] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. 1998. Tight bounds on quantum searching. *Fortschritte der Physik* 46, 4-5 (1998), 493–505. doi:10.1002/(SICI)1521-3978(199806)46:4/5<493::AID-PROP493>3.0.CO;2-P

[3] Allan de Lima, Samuel Carvalho, Douglas Mota Dias, Enrique Naredo, Joseph P Sullivan, and Conor Ryan. 2022. GRAPE: Grammatical Algorithms in Python for Evolution. *Signals* 3, 3 (2022), 642–663. doi:10.3390/signals3030039

[4] Allan de Lima, Samuel Carvalho, Douglas Mota Dias, Enrique Naredo, Joseph P. Sullivan, and Conor Ryan. 2022. Lexi2: lexicase selection with lexicographic parsimony pressure. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Boston, Massachusetts) *(GECCO '22)*. Association for Computing Machinery, New York, NY, USA, 929–937. doi:10.1145/3512290.3528803

[5] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research* 13, 70 (2012), 2171–2175. http://jmlr.org/papers/v13/fortin12a.html

[6] 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 (STOC '96)*. ACM, New York, NY, USA, 212–219. doi:10.1145/237814.237866

[7] Inman Harvey. 1992. Species Adaptation Genetic Algorithms: A Basis for a Continuing SAGA. In *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*. MIT Press, Cambridge, MA, 346–354.

[8] Thomas Helmuth, Lee Spector, and James Matheson. 2015. Solving Uncompromising Problems with Lexicase Selection. *IEEE Transactions on Evolutionary Computation* 19, 5 (2015), 630–643. doi:10.1109/TEVC.2014.2362729

[9] Youngseok Kim, Andrew Eddins, Sajant Anand, Ken Xuan Wei, Ewout Van Den Berg, et al. 2023. Evidence for the utility of quantum computing before fault tolerance. *Nature* 618, 7965 (2023), 500–505. doi:10.1038/s41586-023-06096-3

[10] André Leier and Wolfgang Banzhaf. 2003. Evolving Hogg's quantum algorithm using linear-tree GP. In *Genetic and Evolutionary Computation Conference*. Springer, 390–400. doi:10.1007/3-540-45105-6_48

[11] Gushu Li, Yufei Ding, and Yuan Xie. 2019. Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19)*. ACM, 1001–1014. doi:10.1145/3297858.3304023

[12] Martin Lukac, Mikhail Pivtoraiko, Alan Mishchenko, and Marek Perkowski. 2002. Automated Synthesis of Generalized Reversible Cascades Using Genetic Algorithms. In *Proceedings of the 5th International Workshop on Boolean Problems (IWSBP '02)*. Freiberg, Sachsen, Germany, 33–45. http://archives.pdx.edu/ds/psu/12772

[13] Dmitri Maslov. 2016. Advantages of using relative-phase Toffoli gates with an application to multiple control Toffoli optimization. *Physical Review A* 93, 2 (2016), 022311. doi:10.1103/PhysRevA.93.022311

[14] Paul Massey, John A. Clark, and Susan Stepney. 2004. Evolving quantum circuits and programs through genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004) (Lecture Notes in Computer Science, Vol. 3103)*. Springer, 569–580. doi:10.1007/978-3-540-24855-2_66

[15] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. 2018. Barren plateaus in quantum neural network training landscapes. *Nature Communications* 9, 1 (2018), 4812. doi:10.1038/s41467-018-07090-4 Article number 4812.

[16] 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), 23. doi:10.1038/s41534-018-0072-4 Article number 23.

[17] Michael O'Neill and Conor Ryan. 2001. Grammatical evolution. *IEEE Transactions on Evolutionary Computation* 5, 4 (2001), 349–358. doi:10.1109/4235.942529

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

[19] Conor Ryan, J. J. Collins, and Michael O'Neill. 1998. Grammatical evolution: Evolving programs for an arbitrary language. In *Proceedings of the First European Workshop on Genetic Programming (EuroGP '98) (Lecture Notes in Computer Science, Vol. 1391)*. Springer, Berlin, Heidelberg, 83–96. doi:10.1007/BFb0055930

[20] Lee Spector. 2004. *Automatic Quantum Computer Programming: A Genetic Programming Approach*. Genetic Programming, Vol. 7. Kluwer Academic Publishers. doi:10.1007/b116136 Published originally in hardcover by Kluwer; later reprinted in softcover by Springer.

[21] Ralf Stadelhofer, Wolfgang Banzhaf, and Dieter Suter. 2008. Evolving blackbox quantum algorithms using genetic programming. *AI EDAM* 22, 3 (2008), 285–297. doi:10.1017/S089006040800019X

[22] Susan Stepney and John A. Clark. 2008. Searching for quantum programs and quantum protocols: a review. *Journal of Computational and Theoretical Nanoscience* 5, 5 (2008), 942–969. doi:10.1166/jctn.2008.2535

[23] Swamit S. Tannu and Moinuddin K. Qureshi. 2019. Not All Qubits Are Created Equal: A Case for Variability-Aware Policies for NISQ-Era Quantum Computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19)*. ACM, 987–999. doi:10.1145/3297858.3304007

[24] Adrian Thompson. 1996. An evolved circuit, intrinsic in silicon, entwined with physics. In *Proceedings of the 1st International Conference on Evolvable Systems (ICES '96) (Lecture Notes in Computer Science, Vol. 1259)*. Springer, 390–405. doi:10.1007/3-540-63173-9_61