Evolving Hardware-Efficient Grover Circuits via Grammatical Evolution

Preprint · May 2025

DOI: 10.13140/RG.2.2.33761.21607

1 author:

Arinze Obidiegwu
University of Limerick
2 PUBLICATIONS **0** CITATIONS

SEE PROFILE

Evolving Hardware-Efficient Grover Circuits via Grammatical Evolution

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

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

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

Abstract

Grover's algorithm provides a quadratic speedup for unstructured search problems, yet its standard implementation becomes resource-intensive when deployed on real quantum hardware. In this work, we demonstrate that quantum circuits evolved via grammatical evolution (GE) can outperform the canonical Grover design on noisy intermediate-scale quantum (NISQ) devices. Using a symbolic BNF grammar and hardware-aware evaluation, we evolved Grover-like circuits for all 8 basis states of a 3-qubit system. On IBM's `ibm_brisbane` backend, and based on 10,000-shot executions, the best evolved circuit achieved 97.7% quasi-fidelity for the $|100\rangle$ target state, with the lowest at 88.4% for $|110\rangle$. In contrast, circuits generated using IBM's official Grover notebook achieved a fidelity range of only 57.7% to 68.5% under the same conditions. Evolved circuits also reduced circuit depth and gate count by up to 94.3% and 93.5%, respectively. These results underscore the potential of symbolic AI techniques to automatically synthesize hardware-efficient quantum programs that outperform analytical baselines on today's real quantum processors.

CCS Concepts

• **Theory of computation** → **Quantum complexity theory**; • **Computing methodologies** → *Evolutionary algorithms*; • **Hardware** → *Quantum technologies.*

Keywords

quantum computing, Grover's algorithm, grammatical evolution, NISQ, quantum circuit synthesis

ACM Reference Format:

Arinze Obidiegwu, Douglas Mota Dias, and Conor Ryan. 2025. Evolving Hardware-Efficient Grover Circuits via Grammatical Evolution. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO '25).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/XXXXXXX.XXXXXXX

1 Introduction

Quantum algorithms offer the potential for exponential or quadratic speedups over classical methods in domains such as unstructured search, optimization, and simulation. Among the most iconic is Grover's algorithm [7], which provides a quadratic advantage for search tasks by iteratively amplifying the amplitude of a marked quantum state. The canonical construction of Grover's algorithm comprises an oracle that flips the phase of the solution state, followed by a diffuser that reflects amplitudes about the mean.

Although Grover's algorithm is provably optimal in the abstract circuit model, its real-world deployment on current noisy intermediate-scale quantum (NISQ) hardware often yields disappointing results. Physical devices impose strict architectural constraints—including limited qubit connectivity, gate infidelities, and compiler-induced overheads—that distort circuit structure and degrade fidelity [1, 4, 5, 11, 13]. For instance, even IBM's official implementation [8] of Grover's algorithm on a 3-qubit system, evaluated with 10000 shots and error mitigation, achieves only 65.9% fidelity for the $|000\rangle$ target state. This highlights the disconnect between analytical optimality and practical hardware viability.

This disconnect motivates a core question: *Can quantum algorithms be automatically restructured or synthesized to better suit the hardware they run on?*

In this work, we answer this question affirmatively by applying **grammatical evolution** (GE)—a form of symbolic genetic programming—to synthesize hardware-efficient quantum circuits for Grover-style search problems. GE uses a user-defined grammar to evolve programs as symbolic expressions, enabling it to explore structurally diverse and hardware-adaptive circuit configurations beyond canonical templates [9, 14, 15]. While prior work has explored GE in simulated quantum contexts [15, 16], our study is the first to evolve Grover circuits with direct execution and validation on real quantum processors.

We focus on 3-qubit Grover search tasks and evolve dedicated circuits for all 8 possible marked basis states. Candidate solutions are evaluated through a hybrid pipeline: initial evolution is conducted in classical simulation, and the best individuals are subsequently transpiled and executed on IBM's superconducting `ibm_brisbane` backend with 10,000-shot executions to ensure statistical robustness.

Our results are striking. For the $|000\rangle$ target state, a GE-evolved circuit achieved **97.9% fidelity**—over **30 percentage points higher** than IBM's own benchmark Grover circuit. Across all 8 marked states, evolved circuits consistently outperformed standard implementations, achieving fidelities between 88.4% and 97.7%, while reducing circuit depth and gate count by up to **94.3%** and **93.5%**, respectively.

These findings underscore the potential of symbolic AI methods not merely as optimizers of hand-designed quantum algorithms, but as generative engines capable of synthesizing novel, hardware-native programs from scratch. Grammatical evolution provides

GECCO '25, Barcelona, Spain

ACM ISBN 978-1-4503-XXXX-X/25/07
https://doi.org/10.1145/XXXXXXX.XXXXXXX

a powerful framework for quantum algorithm discovery—one in which circuit structure emerges from evolutionary pressure rather than human intuition. This paradigm represents a shift toward hardware-informed, data-driven quantum software design, which may be critical for fully exploiting current and near-future NISQ systems.

1.1 Background and Related Work

Grover's algorithm [7] remains one of the foundational achievements in quantum computing, offering a quadratic speedup for unstructured search problems. While its optimality in the query model is well understood, deploying Grover's algorithm on physical quantum hardware—especially noisy intermediate-scale quantum (NISQ) systems—introduces substantial practical challenges. Recent empirical studies, such as that by AbuGhanem et al. [1], have shown that standard implementations of Grover's algorithm suffer significant fidelity degradation, circuit depth inflation, and variability in performance due to hardware-specific constraints.

To mitigate these issues, several compiler- and architecture-aware techniques have been proposed. For instance, noise-adaptive transpilation [11], advanced qubit routing strategies [5], and backend-optimized compilation stacks like Qiskit [3] have made meaningful progress in reducing runtime error and overhead. However, these approaches typically operate *post hoc*—attempting to optimize circuit performance after the structure has already been fixed, often without explicit regard for hardware constraints during design.

Our approach instead adopts a *generative* methodology, employing grammatical evolution (GE) to synthesize quantum circuits *from first principles*. GE enables the symbolic construction of circuits using a flexible, BNF-based grammar, allowing exploration beyond traditional circuit templates. Prior work has demonstrated the feasibility of evolving quantum programs using genetic programming techniques [9, 15], albeit primarily in simulation. More recently, Sünkel et al. [16] proposed a hybrid evolutionary framework combining classical evolution with quantum simulation backends—but again, evaluation remained limited to emulated environments.

We extend this line of research by validating evolved circuits on real superconducting hardware. Although the evolutionary search operates in simulation for efficiency, our top-performing circuits are transpiled and executed on IBM's `ibm_brisbane` backend. This hybrid evaluation loop incorporates real-device feedback into the final selection phase, enabling practical assessment while maintaining a tractable search process.

This work contributes to the broader area of quantum–classical co-design [2], where algorithm development is informed by the capabilities and limitations of specific hardware targets. By combining symbolic AI, evolutionary search, and real-device validation, we propose a practical path forward for discovering quantum circuits that are not just theoretically valid—but operationally effective on today's quantum machines.

1.2 Grover's Algorithm

Grover's algorithm [7] is a quantum search procedure that offers a quadratic speedup over classical brute-force methods for identifying a marked item in an unstructured database of size $N = 2^n$. The algorithm initializes the system in a uniform superposition over all computational basis states and then iteratively amplifies the amplitude of the marked state using two key components: an oracle and a diffuser.

The standard implementation comprises:

(1) An oracle O_f that applies a phase flip to the marked state $|x\rangle$, such that $O_f|x\rangle = -|x\rangle$, while leaving all other states unchanged.
(2) A diffusion operator D that performs inversion about the mean, effectively amplifying the probability amplitude of the marked state.
(3) Repeated application of the composite operator $D \cdot O_f$, approximately $\left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor$ times.

While Grover's algorithm is mathematically elegant and provably optimal in the idealized query model, its implementation on NISQ-era quantum hardware faces practical limitations. In particular, multi-controlled operations—such as those used in the oracle and diffusion subroutines—require decomposition into native gate sets, which significantly increases circuit depth and introduces additional noise [10]. These overheads compound quickly on real devices, reducing the fidelity of the algorithm's output and limiting its practical utility without careful hardware-aware optimization.

1.3 NISQ Hardware Limitations

Noisy intermediate-scale quantum (NISQ) devices [13] are fundamentally limited by a combination of architectural and physical constraints, including short coherence times, limited qubit counts, and imperfect gate fidelities. As a result, quantum circuits that are efficient in theory often encounter substantial performance degradation when deployed on real hardware due to the following factors:

- **Sparse connectivity:** Most superconducting quantum processors exhibit restricted qubit connectivity, requiring additional SWAP operations to facilitate interactions between non-adjacent qubits [11]. These insertions inflate circuit depth and introduce additional opportunities for noise.
- **Gate decomposition overhead:** Logical gates involving multiple control qubits (e.g., multi-controlled Toffoli or Z gates) are not natively supported and must be decomposed into elementary 1- and 2-qubit operations [10]. This leads to significant increases in gate count and execution time.
- **Compiler variability:** Backend-specific transpilation pipelines apply heuristic optimizations that can yield inconsistent circuit structures across compilations [5], complicating reproducibility and performance assessment.

For instance, the canonical 3-qubit Grover circuit—elegant in its idealized form—can expand to over 160 layers in depth when transpiled for IBM's `ibm_brisbane` backend [1], making it highly susceptible to decoherence and gate errors. These challenges highlight the urgent need for circuit synthesis approaches that are inherently hardware-aware, minimizing depth and noise exposure from the outset.

1.4 Grammatical Evolution

Grammatical evolution (GE) [14] is a form of genetic programming that evolves symbolic programs from a user-defined context-free

grammar, typically specified in Backus-Naur Form (BNF). Genomes are encoded as linear integer strings, which are mapped to syntactically valid structures through recursive rule expansion. This genotype–phenotype distinction enables flexible and modular search spaces that can represent complex programs, expressions, or—relevant here—quantum circuits.

In the domain of quantum algorithm design, GE provides a principled framework for exploring non-standard circuit architectures. By defining a grammar that includes both native quantum gate primitives (e.g., `h`, `cx`, `rz`) and higher-level subroutines (e.g., oracle and diffuser templates), GE enables the automated discovery of hardware-tailored circuits that deviate from textbook structures [15]. This symbolic search process allows the emergence of compact, performant circuits that better align with the constraints of real quantum hardware.

In our work, GE is used to evolve circuits that amplify a target basis state—e.g., $|000\rangle$—in Grover-style search problems. A hardware-aware fitness function drives the evolution process, defined as:

$$\text{Fitness} = 10 \cdot \text{miss} + (1 - p_{\text{marked}}) + \lambda \cdot \text{gate\_count} \quad (1)$$

Here, miss = 1 if the probability p_{marked} of observing the target state falls below a predefined threshold (e.g., 48%), and zero otherwise. The term λ is a tunable penalty coefficient applied to the total gate count, promoting shallow and efficient circuits. Individuals that fail to compile or execute on the simulator are assigned infinite fitness.

This formulation ensures that evolution is guided toward solutions that maximize marked-state fidelity while minimizing circuit complexity—two properties that are critically important in NISQ settings. By integrating these physical considerations directly into the fitness landscape, GE can produce circuits that are both functionally effective and operationally viable.

2 Methodology

2.1 Standard Grover Implementation

To establish a performance baseline, we adapted the official Qiskit tutorial implementation of Grover's algorithm [8]. Our only modification was to vary the marked state across all 8 computational basis states from $|000\rangle$ to $|111\rangle$.

Each circuit was constructed using the tutorial's default amplitude amplification structure, which includes $k = \left\lfloor \frac{\pi}{4}\sqrt{2^n} \right\rfloor$ Grover iterations [7]. Circuits were transpiled using the `generate_preset_pass_mar` function at optimization level 3, targeting IBM's `ibm_brisbane` backend [1]. We recorded transpilation metrics, including circuit depth and gate count. Fidelity scores were computed from real-device histogram outputs.

2.2 Grammatical Encoding of Circuit Space

We encoded the quantum circuit search space using a Backus–Naur Form (BNF) grammar designed to support parameterized constructions inspired by Grover's algorithm. The grammar includes:

- Initialization and measurement routines.
- Oracle placeholders for injecting marked-state logic.
- Diffuser structures using Hadamard and X/controlled gates.
- A gate set comprising `rz`, `sx`, `cx`, and `rxx`.
- Fixed-angle parameters such as $\pi/2$, π, and $3\pi/2$.

Genome decoding was handled via the GRAPE library [6], a Python-based implementation of grammatical evolution built on DEAP. Integer genomes were mapped to Python circuit code and converted into valid `QuantumCircuit` objects. Sandbox execution ensured that malformed phenotypes or invalid circuits were filtered out prior to evaluation.

2.3 Fitness Evaluation and Constraints

Each candidate circuit was evaluated on its ability to amplify a specific marked state, using 10000 measurement shots. Fidelity was defined as the empirical probability of measuring the correct bitstring. Fitness was calculated using the expression:

$$\text{Fitness} = 10 \cdot \text{miss} + (1 - p_{\text{marked}}) + \lambda \cdot \text{gate\_count} \quad (2)$$

Here, miss = 1 if $p_{\text{marked}} < 0.48$, and 0 otherwise. The regularization parameter $\lambda = 0.02$ penalizes excessive gate usage. Circuits that failed to decode, compile, or execute were assigned infinite fitness. This function prioritizes both fidelity and circuit parsimony—two key metrics for NISQ compatibility [5, 11, 13].

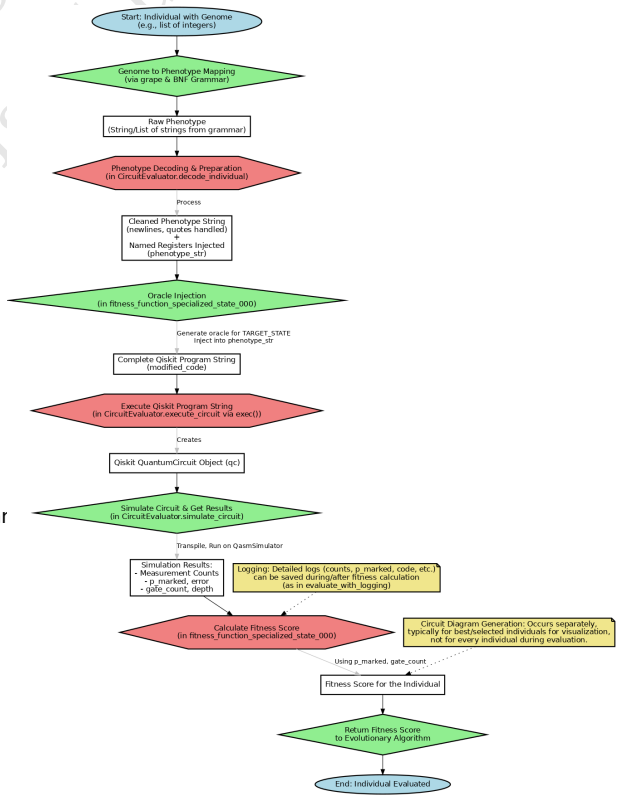

Figure 1: Evaluation pipeline for individual candidates, showing genome decoding, oracle injection, circuit simulation, and final fitness assignment.

2.4 Hardware-Aware Evaluation Pipeline

After initial validation in Qiskit Aer, evolved circuits were transpiled to IBM's `ibm_brisbane` native gate set using the same pass manager as for the baseline Grover circuits. Executions were performed on real quantum hardware to observe fidelity and resource performance under realistic noise conditions.

After execution, we analyzed the measurement outcomes to evaluate how closely the circuits reached their target states. We also collected circuit metrics like depth and gate count to compare the evolved circuits against the standard Grover implementation.

2.5 Grammatical Evolution Parameters

The GE algorithm followed a steady-state elitist $(1 + \lambda)$ model over 100 generations. The hyperparameters used are listed in Table 1 lists the hyperparameters used.

Table 1: Grammatical Evolution Hyperparameters

Parameter	Value
Population size	1000
Generations	100
Codon size	400
Genome representation	List-based
Codon consumption	Lazy
Min init tree depth	20
Max init tree depth	40
Max tree depth	50
Crossover probability	0.8
Mutation probability	0.01
Tournament size	5
Elite size	1
Fitness threshold (miss)	48%
Gate penalty coefficient (λ)	0.02

2.6 Grammatical Evolution Workflow

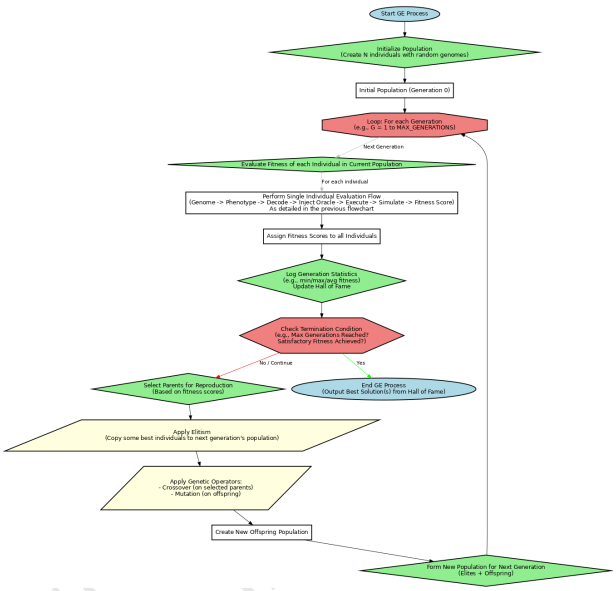

Figure 2: Grammatical evolution workflow for hardware-efficient quantum circuit synthesis.

As illustrated in Figure 2, our GE pipeline integrates symbolic circuit generation with simulation-driven fitness evaluation. Starting from a user-defined grammar, integer-based genotypes are decoded into Qiskit circuits, which are executed, evaluated, and subjected to evolutionary refinement. Evolutionary operators (crossover, mutation, selection) steer the population toward low-depth, high-fidelity solutions over successive generations.

2.7 Grammar Summary

The grammar defines a compositional search space over valid 3-qubit Qiskit programs. A simplified snippet is shown below:

Figure 3: BNF grammar fragment for Grover circuit generation.

```
<Program> ::= <Initialize> <HadamardAll>
            <GroverIterations> <Measure>

<GroverIterations> ::= <GroverIteration>
                     | <GroverIteration> <GroverIteration>

<GroverIteration> ::= <OracleBlock> <DiffuserBlock>

<SingleQubitGate> ::=
  "qc.x(" <Qubit> ")"
  | "qc.ry(" <Angle> "," <Qubit> ")"
  | ...
```

3 Experiments

3.1 Experimental Setup

All experiments were conducted on IBM's superconducting quantum backend `ibm_brisbane`, accessed via the Qiskit Runtime framework [3]. This 127-qubit device features a heavy-hex topology,

which offers improved connectivity over planar grids while maintaining fabrication feasibility [1]. It supports a native gate set composed of `rz`, `sx`, `x`, and `ecr` operations [5].

We focused on 3-qubit Grover search problems, exhaustively evaluating each of the 8 computational basis states from $|000\rangle$ to $|111\rangle$. For every target state, we compared two circuit classes:

- **Canonical Grover implementation:** Based on IBM's official Qiskit Grover notebook [8], using the `GroverOperator` circuit library interface to construct circuits with state-specific oracles.
- **Evolved circuits:** Synthesized via grammatical evolution (GE), optimized for fidelity and resource efficiency under hardware constraints.

All circuits were executed using 10000 measurement shots. Prior to hardware execution, circuits were validated using Qiskit Aer [3]. Transpilation was performed using Qiskit's level-3 preset pass manager [11], which applies aggressive optimization and routing strategies. We collected both pre- and post-transpilation metrics, including circuit depth, total gate count, and number of entangling (`cx` or `ecr`) operations.

3.2 Baseline Grover Results

Canonical Grover circuits were executed across all 8 marked basis states using Qiskit's official `Grover` implementation notebook, with only the marked state changed per run. Although the abstract, uncompiled Grover circuit requires only 5–6 logical layers in theory, the transpilation process—responsible for adapting the circuit to hardware-native gates and connectivity constraints—introduces significant overhead. This results in depths exceeding 175 layers on IBM's superconducting architecture. [13].

Across all target states, the baseline Grover circuits exhibited:

- **Circuit depths ranging from 175 to 185**, primarily due to routing and entangling operations.
- **Total gate counts between 277 and 290**, with a significant fraction composed of `ecr`, `rz`, and `sx` gates.
- **Limited execution fidelity**: For instance, the transpiled $|000\rangle$ circuit achieved only 63.3% fidelity even after quasi-probability mitigation. The $|110\rangle$ circuit was notably worse, with fidelity dropping to 57.7%.

These outcomes reinforce the well-known limitations of deploying idealized quantum algorithms on NISQ hardware [2, 4]. Fidelity losses stem from circuit depth, crosstalk, and noise amplification—factors aggravated by the decomposition of complex oracles and diffusers.

3.3 Evolved Circuit Results

Grammatical evolution was used to independently synthesize a custom circuit for each marked basis state. Candidates were selected via a fitness function that rewarded high fidelity while penalizing gate depth and usage of error-prone operations. Each circuit was transpiled to the native gate set of `ibm_brisbane` and executed using the same experimental workflow as the baseline.

Key findings include:

- **Sharp reductions in circuit depth and total gates.** For example, the evolved $|000\rangle$ circuit achieved 94.8% fidelity with only 11 layers and 21 gates—down from 177 layers and 283 gates in the Grover baseline.
- **Superior fidelity across all states.** Every evolved circuit outperformed its Grover counterpart in fidelity, with gains of up to 33 percentage points. The evolved $|100\rangle$ circuit reached 97.7%, compared to 68.5% for the baseline.
- **Hardware-congruent structural patterns.** Evolution consistently favored shallow, low-overhead motifs that minimized entanglement depth and avoided compilation-unfriendly constructs. This yielded circuits well-matched to the native operations of superconducting platforms [11, 12].

These results underscore the potential of symbolic evolutionary methods for quantum circuit design. Evolved circuits not only achieved higher fidelity under real noise conditions but also mapped more efficiently to physical constraints. Table 2 and Table 4 summarize these improvements across all marked states.

4 Results and Analysis

4.1 Evolved Circuit Performance

Table 2 summarizes the post-transpilation performance of the best evolved circuits targeting each of the 8 computational basis states in a 3-qubit Grover search task. Each metric was recorded after compiling to the native gate set of IBM's `ibm_brisbane` backend [1].

Table 2: Hardware-mapped metrics for evolved circuits targeting each 3-qubit marked state.

State	Depth	Total Gates	ECR	RZ	SX	X
000	11	21	2	10	4	2
001	12	26	2	12	8	1
010	10	18	2	9	4	0
011	12	20	2	8	4	3
100	12	22	2	10	4	3
101	18	34	4	16	10	1
110	21	39	5	19	11	1
111	11	20	2	10	4	1

Across all marked states, evolved circuits demonstrate consistently shallow depths and reduced gate counts—two critical factors in mitigating error accumulation and decoherence on NISQ devices [2, 13]. Even the most complex evolved circuit (targeting `110`) required only 21 layers and 39 gates—orders of magnitude smaller than its canonical counterpart.

To illustrate this structural efficiency, Figures 4, 5, and 6 display the hardware-mapped transpiled circuits for representative evolved solutions. These visualizations highlight the sparse and shallow layouts produced by GE when optimizing under hardware constraints.

4.2 Comparison with Canonical Grover Circuits

To contextualize the gains, Table 3 presents analogous metrics for the canonical Grover implementations post-transpilation. The ballooning of circuit depth and gate count is largely attributable to

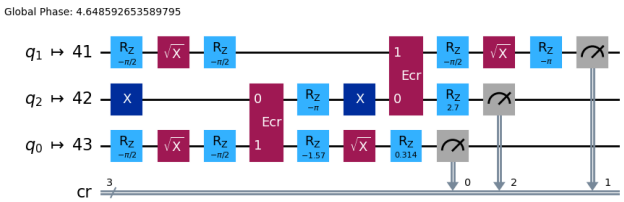

Figure 4: Transpiled circuit for evolved $|000\rangle$ phenotype, mapped to `ibm_brisbane`.

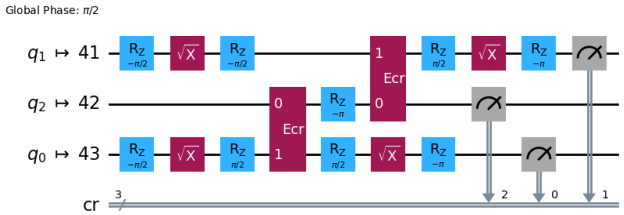

Figure 5: Transpiled circuit for evolved $|010\rangle$ phenotype, mapped to `ibm_brisbane`.

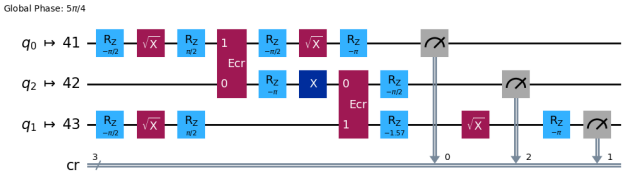

Figure 6: Transpiled circuit for evolved $|111\rangle$ phenotype, mapped to `ibm_brisbane`.

Table 3: Hardware-mapped metrics for standard Grover circuits (post-transpilation).

State	Depth	Total	ECR	RZ	SX	X
000	177	283	37	143	90	10
001	181	288	39	146	91	9
010	175	277	37	143	90	4
011	185	290	39	148	91	9
100	183	289	39	146	91	10
101	180	286	39	145	91	8
110	182	285	39	147	91	5
111	178	282	37	144	90	8

the decomposition of multi-controlled gates and routing overhead induced by `ibm_brisbane`'s heavy-hex topology [5, 11].

These results underscore that analytical elegance does not guarantee hardware efficiency [4, 12]. Despite Grover's theoretical optimality, its standard implementation incurs substantial overhead when compiled for real NISQ devices. In contrast, the evolved circuits—generated through symbolic search and guided by fidelity-aware, resource-penalizing fitness functions—consistently outperform canonical designs across all tested metrics, including depth, gate count, and native gate utilization.

To further highlight this contrast, Figure 7 compares the post-transpilation depth and total gate count for both evolved and baseline Grover circuits across all marked 3-qubit states.

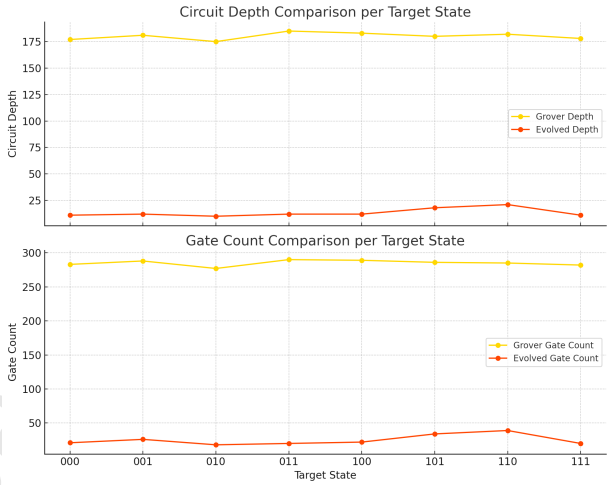

Figure 7: Circuit depth and gate count comparison between standard Grover and evolved circuits across all 3-qubit target states. Evolved designs consistently achieve lower resource usage.

Importantly, evolved circuits exhibit hardware-adaptive characteristics such as:

- **Gate sparsity:** Prioritization of 1- and 2-qubit native operations (e.g., `rz`, `sx`, `ecr`) over entangling-heavy sequences.
- **Structural regularity:** Emergent modular motifs and reusable subroutines, often aligned with the hardware's coupling map.
- **Transpilation resilience:** Minimal structural inflation post-compilation, indicating strong alignment between evolved topologies and hardware constraints.

These traits, which are difficult to engineer by hand, emerge organically through grammatical evolution—underscoring its value as a generative co-design method for practical quantum algorithms under NISQ-era limitations.

5 Discussion

Our findings demonstrate that grammatical evolution (GE) can synthesize quantum circuits that significantly outperform analytically derived counterparts, such as Grover's algorithm, in hardware-constrained settings. While Grover's design is provably optimal in the ideal circuit model, it neglects critical physical factors—namely, transpilation artifacts, native gate sets, and stochastic noise—that dominate execution fidelity on NISQ devices. In contrast, evolved circuits were implicitly shaped by these constraints through a fitness function that penalized gate count, depth, and execution failure (see Table 2 and Table 3).

Recent work has empirically illustrated the scalability limits of Grover's algorithm on superconducting quantum systems, where fidelity degrades and circuit overhead scales unfavorably with problem size [1]. These observations motivate hardware-aware algorithm synthesis approaches that are flexible and performance-oriented, rather than structurally prescriptive.

Several core insights emerge from our experiments:

- **Structure vs. performance:** The evolved circuits frequently diverged from Grover's canonical oracle–diffuser structure. In many cases, minimal diffuser elements or alternative subroutines sufficed to achieve high fidelity—challenging the necessity of textbook formulations for near-term devices.
- **Hardware alignment:** GE consistently favored shallow, low-overhead circuit motifs natively compatible with IBM's superconducting gate set. Complex operations such as multi-controlled gates were naturally pruned by selection pressure due to their adverse impact on fidelity and depth.
- **Interpretability and modularity:** Despite their stochastic origin, evolved circuits often exhibited modular and interpretable substructures. This suggests that symbolic AI methods like GE can yield not just performant but also human-readable designs—beneficial for debugging, transferability, and hybrid composition.
- **Instance-specific tailoring:** Since GE was applied independently for each target bitstring, the resulting circuits were tuned to specific oracle structures. This property aligns well with emerging application domains—such as VQAs or hybrid quantum classifiers—where instance-specific optimization is required.

These outcomes reflect a broader shift in quantum software research toward compilation-aware, hardware-native methodologies. GE, as a symbolic and model-free technique, integrates naturally into such workflows—yielding circuits that are not only performant but also practical on today's NISQ hardware.

5.1 Reframing Quantum Algorithm Discovery

The conventional pipeline for quantum algorithm design is inherently *top-down*: a logical structure is first specified, then compiled and mapped to hardware. Our results support a *bottom-up* alternative—where circuits emerge from evolutionary search, guided by physical execution feedback.

This reframing has several implications:

- **From analytical templates to emergent programs:** Evolved circuits succeeded without mirroring Grover's full structure. High-fidelity behavior emerged through symbolic search under hardware constraints, rather than from classical template instantiation.
- **Hardware-first synthesis:** By embedding fidelity, gate count, and compilation success directly into the objective function, GE favored architectures that are efficient in *real* execution—highlighting the value of hardware-aware search.
- **Quantum–classical co-design in practice:** The evolutionary loop forms a hybrid system in which the classical optimizer proposes candidates and the quantum backend supplies empirical validation. This co-adaptive model supports robustness to calibration drift and backend updates.
- **Beyond human intuition:** The most effective evolved circuits often bore little resemblance to canonical designs. Their utility emerged not from handcrafted cleverness, but from an unbiased symbolic exploration grounded in physical feasibility.

While this study focuses on Grover-style search, the same GE framework could generalize to other algorithm families—especially in quantum machine learning, variational algorithms, or hybrid classifiers—where circuit structure is problem-dependent and hardware efficiency is paramount.

Taken together, these findings motivate a paradigm shift: rather than asking how to adapt theoretical algorithms for hardware, we can ask what new algorithms are *enabled* by the hardware. Grammatical evolution offers one compelling approach to that question—yielding not only valid but often superior solutions under real-world constraints.

These insights provide the foundation for the concluding section, where we discuss limitations, broader generalization, and future directions for this line of work.

6 Conclusion

This work demonstrates that *grammatical evolution* (GE) can be effectively leveraged to synthesize compact, hardware-efficient quantum circuits that outperform canonical designs under real-world execution conditions. Applied to Grover-style search problems, GE-produced circuits consistently exceeded the performance of textbook implementations on IBM's NISQ hardware—achieving up to 97.7% fidelity across marked states, compared to 57–68% for Qiskit's standard Grover circuits.

Crucially, this success was achieved without incorporating explicit noise models or backend calibration data. Instead, the evolutionary process implicitly adapted to hardware constraints by favoring shallow, native-compatible gate structures through a fidelity-aware, resource-penalizing fitness function. This validates GE as a practical and scalable tool for quantum–classical co-design.

More broadly, our findings support a paradigm shift in quantum software development: from hand-crafted, top-down algorithm design to bottom-up, hardware-guided program synthesis. Symbolic methods such as GE can serve not only as performance optimizers but also as creative engines—discovering efficient, interpretable, and physically viable circuits from scratch. As quantum hardware evolves, approaches like GE will play a critical role in bridging the gap between abstract algorithmic potential and real-device performance, thereby empowering more effective use of current and next-generation quantum systems.

7 Limitations and Future Directions

While this study offers promising results, several limitations suggest avenues for further investigation:

Scalability: Our experiments were limited to 3-qubit Grover problems, which are computationally tractable and allow exhaustive testing. Scaling GE to larger qubit counts will increase the search space combinatorially, necessitating grammar modularization, parallel simulation strategies, and possibly surrogate fitness estimation techniques.

Oracle-Specific Designs: Each evolved circuit was tailored to a fixed marked state. Although this is consistent with Grover's formulation for known targets, the synthesis of a single, general-purpose circuit applicable to arbitrary marked states remains an open challenge. Future work could explore evolving parameterized circuit templates that generalize across various oracle classes.

Hardware Dependence: All circuits were evaluated on a single real backend (`ibm_brisbane`). Although this enabled realistic hardware-aware optimization, it may have introduced backend-specific bias. Further validation on diverse quantum processors is necessary to assess the portability and robustness of the evolved solutions.

Baselines and Benchmarks: Our comparisons were made against textbook Grover circuits generated via Qiskit. Incorporating stronger baselines—such as transpiler-optimized variants, hand-tuned implementations, or learned circuit templates—would further strengthen the comparative analysis.

Evaluation Cost: Hardware-in-the-loop evaluations are expensive and slow. Although classical simulation sufficed for the evolutionary search, scaling to larger systems or more generations may require hybrid pipelines utilizing surrogate models, transfer learning, or selective sampling strategies.

Looking beyond Grover's algorithm, the symbolic evolutionary approach demonstrated here has broader applicability. Many areas of quantum computing—such as quantum machine learning (QML), variational algorithms (e.g., VQE and QAOA), and quantum simulation—demand circuits that balance expressiveness with hardware efficiency. GE provides a principled framework to explore these trade-offs automatically, generating candidate circuits that align with physical constraints.

In QML, for instance, evolved quantum circuit architectures could enhance generalization while mitigating decoherence. Similarly, GE can be used to discover ansätze that are tailored to specific problems or devices within variational workflows. As the field moves toward compiler-aware and architecture-specific quantum software stacks, grammar-driven synthesis may become an essential tool for automated circuit generation, optimization, and co-design.

A Evolutionary Convergence

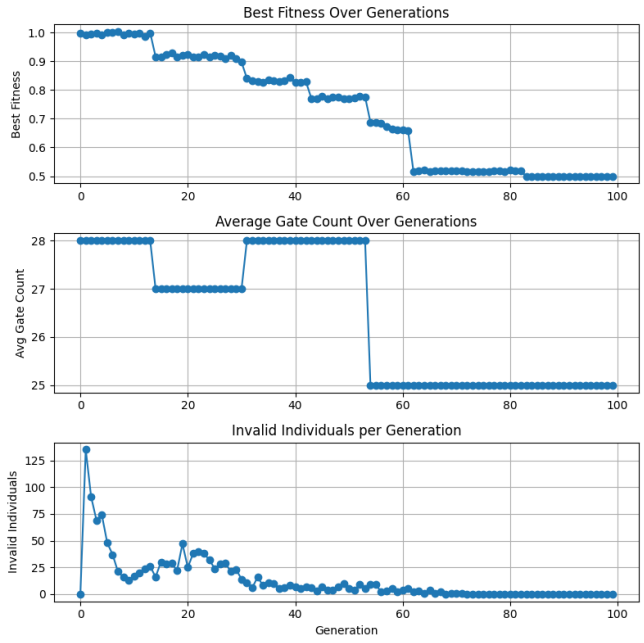

Figure 8: Evolution metrics for target state $|111\rangle$.

B Evolved Circuit Code Listings

Target State: $|000\rangle$

Fitness Score: 0.5

Target State: $|010\rangle$

Fitness Score: 0.52

Target State: $|111\rangle$

Fitness Score: 0.50

```
"qc = QuantumCircuit(3, 3)\n" \
"qc.h(0)\n" \
"qc.h(1)\n" \
"qc.h(2)\n" \
"## Begin Diffuser\n" \
"qc.h(0)\n" \
"qc.h(1)\n" \
"qc.h(2)\n" \
"qc.x(0)\n" \
"qc.x(1)\n" \
"qc.x(2)\n" \
"qc.h(2)\n" \
"qc.cx(0, 2)\n" \
"qc.cx(1, 2)\n" \
"qc.h(2)\n" \
"qc.x(0)\n" \
"qc.x(1)\n" \
"qc.x(2)\n" \
"qc.h(0)\n" \
"qc.h(1)\n" \
"qc.h(2)\n" \
"qc.h(1)\n" \
"qc.h(2)\n" \
"qc.u(1.5708, 1.3, 2*np.pi, 0)\n" \
"qc.rxx(3.1415, 1, 0)\n" \
"## End Diffuser\n" \
"qc.measure(0, 0)\n" \
"qc.measure(1, 1)\n" \
"qc.measure(2, 2)\n"
```

Listing 1: Evolved circuit for target state $|000\rangle$

```
"qc = QuantumCircuit(3, 3)\n" \
"qc.h(0)\n" \
"qc.h(1)\n" \
"qc.h(2)\n" \
"## Begin Diffuser\n" \
"qc.h(0)\n" \
"qc.h(1)\n" \
"qc.h(2)\n" \
"qc.x(0)\n" \
"qc.x(1)\n" \
"qc.x(2)\n" \
"qc.h(2)\n" \
"qc.cx(0,2)\n" \
"qc.cx(1,2)\n" \
"qc.h(2)\n" \
"qc.x(0)\n" \
"qc.x(1)\n" \
"qc.x(2)\n" \
"qc.h(0)\n" \
"qc.h(1)\n" \
"qc.h(2)\n" \
"qc.ry(3*np.pi/2, 0)\n" \
"qc.h(2)\n" \
"qc.ry(np.pi/2, 1)\n" \
"## End Diffuser\n" \
"qc.measure(0, 0)\n" \
"qc.measure(1, 1)\n" \
"qc.measure(2, 2)\n"
```

Listing 2: Evolved circuit for target state $|010\rangle$

```
"qc = QuantumCircuit(3, 3)\n" \
"qc.h(0)\n" \
"qc.h(1)\n" \
"qc.h(2)\n" \
"## Begin Diffuser\n" \
"qc.h(0)\n" \
"qc.h(1)\n" \
"qc.h(2)\n" \
"qc.x(0)\n" \
"qc.x(1)\n" \
"qc.x(2)\n" \
"qc.h(2)\n" \
"qc.cx(0,2)\n" \
"qc.cx(1,2)\n" \
"qc.h(2)\n" \
"qc.x(0)\n" \
"qc.x(1)\n" \
"qc.x(2)\n" \
"qc.h(0)\n" \
"qc.h(1)\n" \
"qc.h(2)\n" \
"qc.ry(np.pi/2, 0)\n" \
"qc.ry(1.5708, 1)\n" \
"qc.u(3*np.pi/2, np.pi/2, np.pi, 2)\n" \
"## End Diffuser\n" \
"qc.measure(0, 0)\n" \
"qc.measure(1, 1)\n" \
"qc.measure(2, 2)\n"
```

Listing 3: Evolved circuit for target state $|111\rangle$

Table 4: Hardware fidelity and resource comparison across all 3-qubit basis states. Evolved circuits consistently outperform standard Grover implementations in fidelity, depth, and gate efficiency.

Target State	Circuit Type	Fidelity	Depth	Gate Count	Depth Reduction (%)	Gate Reduction (%)
$\|000\rangle$	Evolved	**94.8%**	11	21	93.8%	92.6%
	Grover	63.3%	177	283		
$\|001\rangle$	Evolved	**91.0%**	12	26	93.4%	91.0%
	Grover	66.0%	181	288		
$\|010\rangle$	Evolved	**96.1%**	10	18	94.3%	93.5%
	Grover	61.8%	175	277		
$\|011\rangle$	Evolved	**95.8%**	12	20	93.5%	93.1%
	Grover	63.8%	185	290		
$\|100\rangle$	Evolved	**97.7%**	12	22	93.4%	92.4%
	Grover	68.5%	183	289		
$\|101\rangle$	Evolved	**90.0%**	18	34	90.0%	88.1%
	Grover	67.3%	180	286		
$\|110\rangle$	Evolved	**88.4%**	21	39	88.5%	86.3%
	Grover	57.7%	182	285		
$\|111\rangle$	Evolved	**95.5%**	11	20	93.8%	92.9%
	Grover	62.9%	178	282		

References

[1] Muhammad AbuGhanem. 2025. Characterizing Grover search algorithm on large-scale superconducting quantum computers. *Scientific Reports* 15, 1 (2025), 1281. doi:10.1038/s41598-024-80188-6

[2] Muhammad A. Alam, Abdullah Ash-Saki, and Joydeep Ghosh. 2022. Classical optimizers for noisy intermediate-scale quantum computers. *ACM Computing Surveys (CSUR)* 55, 4 (2022), 1–37.

[3] Gadi Aleksandrowicz, Thomas Alexander, Pascal Barkoutsos, Luciano Bello, Yotam Ben-Haim, David Bucher, Francisco J. Cabrera-Hernández, Juan Carballo-Franquis, Anthony Chen, Chun-Fu Chen, et al. 2019. Qiskit: An Open-source Framework for Quantum Computing. https://qiskit.org. doi:10.5281/zenodo.2562111 Accessed May 2025.

[4] Sitan Chen, Jordan Cotler, Hsin-Yuan Huang, and Jerry Li. 2023. The complexity of NISQ. *Nature Communications* 14, 6001 (2023). https://www.nature.com/articles/s41467-023-41217-6

[5] Alec Cowtan, Steven Dilkes, Ross Duncan, Alexander Krajenbrink, Will Simmons, and S. Sivarajah. 2020. Qubit routing and scheduling for NISQ era quantum computers. *Quantum Science and Technology* 5, 3 (2020), 034010.

[6] 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

[7] Lov K Grover. 1996. A fast quantum mechanical algorithm for database search. *Proceedings of the 28th Annual ACM Symposium on Theory of Computing* (1996), 212–219.

[8] IBM Quantum. 2024. Grover's Algorithm. https://learning.quantum.ibm.com/tutorial/grovers-algorithm.

[9] Simon M. Lucas. 2004. Evolving quantum oracles with genetic programming. In *Congress on Evolutionary Computation (CEC)*. 885–890.

[10] 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.

[11] Prakash Murali, Jonathan M. Baker, Ali Javadi Abhari, Frederic T. Chong, and Margaret Martonosi. 2019. Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 1015–1029. doi:10.1145/3297858.3304075

[12] S. Muroya, K. Chatterjee, and T. A. Henzinger. 2025. Hardware-optimal quantum algorithms. *Proceedings of the National Academy of Sciences* 122 (2025). doi:10.1073/pnas.2419273122

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

[14] Conor Ryan, J. J. Collins, and Michael O'Neill. 1998. Grammatical Evolution: Evolving Programs for an Arbitrary Language. In *Genetic Programming, Proceedings of the 1st European Workshop, EuroGP 1998 (Lecture Notes in Computer Science, Vol. 1391)*. Springer, 83–96. doi:10.1007/BFb0055930

[15] Lee Spector. 2004. *Automatic Quantum Computer Programming: A Genetic Programming Approach*. Genetic Programming, Vol. 7. Springer. doi:10.1007/978-1-4419-9126-7

[16] Leo S"unkel, Philipp Altmann, Michael K"olle, Gerhard Stenzel, Thomas Gabor, and Claudia Linnhoff-Popien. 2025. Quantum Circuit Construction and Optimization through Hybrid Evolutionary Algorithms. *arXiv preprint arXiv:2504.17561* (2025). https://arxiv.org/abs/2504.17561