Grammatical Evolution for Quantum Algorithms

Arinze Obidiegwu

Lero - The SFI Research Centre for Software

Faculty of Science & Engineering

Department of Computer Science & Information Systems

University of Limerick

Submitted to the University of Limerick for the degree of

Master of Science (MSc) in Artificial Intelligence and Machine Learning

2025

Supervisor: Prof. Conor Ryan
Lero – the Research Ireland Centre for Software
University of Limerick
Ireland

Supervisor: Dr. Douglas Mota Dias
Department of Computer Science & Applied Physics
Atlantic Technological University
Ireland

Abstract

As quantum computing progresses toward practical application, a key challenge remains: designing quantum circuits that are both accurate and hardware-efficient, particularly for noisy intermediate-scale quantum (NISQ) devices. This dissertation investigates a novel method for automatically generating optimised quantum circuits for Grover's algorithm using Grammatical Evolution (GE) a flexible, bio-inspired search technique capable of producing circuits from first principles.

A hybrid pipeline is developed in which GE is used to explore the quantum circuit design space in classical simulation, while the best candidate circuits are validated on real superconducting quantum hardware. The approach is first benchmarked using the Deutsch-Jozsa algorithm to demonstrate that GE can evolve functional and generalisable quantum structures. The main contribution, however, lies in evolving dedicated, high-fidelity Grover circuits for various marked states, using a grammar tailored to native hardware operations and a fitness function that balances accuracy with gate efficiency.

The results show that the evolved circuits outperform standard Grover implementations in terms of both fidelity and resource usage when executed on IBM's quantum devices. In many cases, the evolved circuits amplify the target state with near-ideal probabilities, using fewer gates than canonical constructions — a significant advantage for real-world deployment on error-prone hardware.

These findings suggest that evolutionary techniques like GE offer a powerful pathway for automated quantum circuit design. They provide evidence that such methods can yield bespoke quantum algorithms that are not only theoretically sound but also practically executable on today's quantum hardware.

Declaration

I herewith declare that I have produced this paper without the prohibited assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in identical or similar form to any other Irish or foreign examination board.

The thesis work was conducted from 2024 to 2025 under the supervision of Professor Conor Ryan and Dr. Douglas Mota Dias at University of Limerick.

Limerick, August 2025

Acknowledgements

This thesis represents not just my academic journey, but the culmination of countless sacrifices, unwavering support, and profound love from those who believed in me.

First and foremost, I owe an immeasurable debt of gratitude to my elder brother Emmanuel Obidiegwu, whose selfless dedication made this entire journey possible. You carried me on your shoulders, ensuring I never lacked anything, sacrificing your own comfort so I could pursue my dreams. Every page of this thesis exists because of your unwavering support and belief in me. Words cannot adequately express my gratitude for everything you have done.

To my supervisors, Professor Conor Ryan and Dr. Douglas Mota Dias, thank you for your exceptional guidance and mentorship. Conor, working with the inventor of Grammatical Evolution himself has been an incredible privilege. Your pioneering spirit and deep insights shaped not just this research, but my understanding of what it means to push boundaries in science. Douglas, your thoughtful guidance and constant encouragement helped me navigate the complexities of this work. Both of you gave me the freedom to express myself creatively while providing the structure and wisdom I needed to succeed. Thank you for believing in my ideas and helping me transform them into meaningful contributions.

To my family, thank you for your endless love, support, and encouragement throughout this journey. Your faith in me has been a constant source of strength.

To my father, thank you for instilling in me the value of education and the determination to succeed. Together with mom, you both

created a foundation of learning and ambition that has carried me to this moment. Your unwavering belief in the power of education and your dreams for my success have been guiding lights throughout my academic journey.

Finally, to my mother, who left us when I was thirteen but whose impact on my life remains immeasurable. You taught me resilience, curiosity, and the importance of education in the brief time we had together. Every achievement in my life carries your memory within it. I hope that somewhere, somehow, you can see what your son has accomplished and that I have made you proud. This thesis is as much yours as it is mine.

Data Source and Generative AI Declaration

Data Sources

All quantum circuits were executed on IBM Quantum Network's `ibm_brisbane` backend, accessed through the IBM Quantum cloud services. Simulation data was generated using Qiskit Aer's quantum circuit simulator. No external datasets were used in this research; all experimental data was generated through evolutionary runs and quantum circuit executions.

Use of Generative AI Tools

During the preparation of this dissertation, Claude (Anthropic) was used as an assistive tool for:

- Grammar and style improvements in manuscript writing
- LaTeX formatting and bibliography management
- Code documentation and commenting

All scientific content, experimental design, implementation, analysis, and conclusions are entirely my own original work. Generative AI was used solely as a writing and formatting assistant, similar to grammar checking tools. All code for the Grammatical Evolution

framework, circuit synthesis, and experimental validation was written independently without AI assistance.

Contents

List of Tables

List of Figures

1

Introduction

This chapter introduces the core research motivations behind this work and positions it within the broader context of quantum computing. It outlines the challenges posed by current quantum hardware, presents the main research question, and explains the aims and contributions of using grammatical evolution to automate circuit design. It also provides a roadmap of the dissertation structure to guide the reader through the following chapters.

1.1 Background and Motivation

Quantum computing promises exponential speed-ups over classical systems for problems such as integer factorisation and unstructured search. However, current quantum hardware, commonly referred to as Noisy Intermediate-Scale Quantum (NISQ) devices, is constrained by limited qubit counts, gate infidelity, and noise-induced decoherence (Preskill, 2018). Strict constraints on circuit depth and fidelity make the efficient design of quantum circuits a critical challenge.

Circuit synthesis, the task of designing executable quantum programmes for specific logical functions, has become an active research area. While textbook algorithms like Grover's and Deutsch–Jozsa offer theoretical speed-ups (Deutsch and Jozsa, 1992; Grover, 1996), their canonical circuits, the standard implementations derived from mathematical proofs and widely taught in quantum computing courses, are not optimised for real hardware execution. These canonical constructions prioritise mathematical elegance and pedagogical clarity over prac-

tical considerations such as gate count minimisation and noise resilience. This dissertation addresses that gap using *grammatical evolution* (GE), a symbolic evolutionary search technique that generates quantum circuits using rules defined in a formal grammar. Our approach achieves up to 94% reduction in circuit depth and 40% improvement in fidelity compared to standard implementations when executed on real quantum hardware.

1.2 Aims and Objectives

The central research question addressed in this dissertation is:

> *Can grammatical evolution be used to automatically synthesise quantum circuits that outperform standard algorithmic constructions in terms of depth, gate count, and fidelity when executed on real quantum hardware?*

To investigate this, the dissertation pursues the following objectives:

- To design and implement a grammar-guided symbolic search pipeline capable of generating valid quantum circuits for a given computational task.
- To assess the approach on Grover's algorithm across all 3-qubit marked states, comparing evolved circuits against canonical Qiskit implementations on actual IBM hardware (IBM Quantum, 2023).
- To provide an auxiliary benchmark using the Deutsch–Jozsa algorithm, demonstrating generalisability under simulated execution.

1.3 Methodology

This work adopts a bottom-up, data-driven approach to quantum programme synthesis. A formal grammar defines the allowable circuit constructs, and grammatical evolution (GE) is used to iteratively evolve candidate circuits. Fitness evaluation is based on output fidelity and gate count, with circuit depth tracked

for analysis. Hardware-specific constraints are implicitly handled via transpilation and execution on IBM's superconducting backend, `ibm_brisbane` (IBM Quantum, 2023).

For Grover's algorithm, evolved circuits were generated across all eight possible 3-qubit marked states. The top-performing individuals were then transpiled and executed on IBM's quantum hardware to assess fidelity and gate efficiency under real-device conditions. Baseline comparisons were made using Qiskit's standard Grover circuits (IBM Quantum, 2023).

For Deutsch–Jozsa, evolved circuits were tested using Qiskit's simulator to validate functional correctness under ideal conditions, as the focus was on symbolic generalisation rather than hardware deployment.

1.4 Research Contribution

The primary research contributions of this dissertation are:

- Demonstration that grammatical evolution can automatically synthesise shallow, hardware-efficient circuits that outperform canonical Grover implementations in fidelity, gate count, and depth.
- Introduction of a grammar-guided pipeline for quantum circuit design that operates without requiring explicit calibration or noise modelling, yet adapts effectively to real-device constraints.
- Validation of generalisability through a secondary implementation of the Deutsch–Jozsa algorithm, reinforcing the viability of symbolic synthesis for quantum programming.

1.5 Thesis Outline

The remaining chapters of this dissertation are structured as follows:

Chapter 2 provides technical background on Grover and Deutsch–Jozsa algorithms, grammatical evolution, and quantum circuit transpilation. It also reviews related work on quantum programme synthesis.

Chapter 3 details the methodology, including the design of the grammar, fitness function, evolutionary parameters, and toolchain configuration.

Chapter 4 outlines the experimental setup for both Grover and Deutsch–Jozsa experiments, including hardware, simulators, and performance metrics.

Chapter 5 presents the experimental results and comparative analysis between evolved and standard circuits across multiple criteria.

Chapter 6 concludes the study, highlighting key findings and suggesting directions for future research, including extensions to noise-aware synthesis, grammar refinement, and multi-objective optimisation strategies.

Appendices provide supplementary materials as follows:

- Appendix 6.4 contains the full formal grammar used for circuit synthesis.
- Appendix 6.4 presents circuit diagrams and result histograms for Deutsch–Jozsa experiments.
- Appendix 6.4 includes relevant code listings and auxiliary derivations.

1.6 Summary

This chapter introduced the motivation and aims of the dissertation, grounded in the challenges of executing quantum algorithms on NISQ hardware. It positioned grammatical evolution as a promising method for synthesising efficient quantum circuits and laid out the scope, methodology, and structure of the work. The following chapter provides the technical background needed to understand quantum circuits, GE, and the synthesis pipeline in detail.

2

Background

This chapter provides the theoretical foundation and prior work necessary to understand the contributions of this dissertation. It introduces the key quantum algorithms under consideration Deutsch–Jozsa and Grover followed by an overview of Grammatical Evolution (GE) as a symbolic circuit synthesis technique. It also examines compilation and transpilation issues specific to Noisy Intermediate-Scale Quantum (NISQ) devices and reviews related approaches in automated quantum program generation.

2.1 Quantum Algorithms Overview

Quantum algorithms exploit the fundamental features of quantum mechanics such as superposition, entanglement, and interference to provide computational advantages over classical methods. Among the earliest and most instructive examples are the Deutsch–Jozsa and Grover algorithms. These two algorithms not only illustrate the power of quantum computation but also serve as canonical benchmarks for evaluating the performance and feasibility of algorithms on real quantum devices.

2.1.1 Deutsch–Jozsa Algorithm

The Deutsch–Jozsa (DJ) algorithm was one of the first to demonstrate an exponential advantage of quantum computation over classical methods (Deutsch and Jozsa, 1992). It addresses the problem of determining whether a Boolean

function $f : \{0,1\}^n \to \{0,1\}$ is constant (returns the same value for all inputs) or balanced (returns 0 for half of all inputs and 1 for the other half). Classically, this requires evaluating the function on at least $2^{n-1}+1$ inputs in the worst case. The DJ algorithm, in contrast, can solve this problem with a single oracle query by leveraging quantum parallelism.

While DJ is not known for practical applications, it remains a foundational example due to its conceptual simplicity and early demonstration of quantum advantage. It also presents a testbed for exploring function evaluation, oracle construction, and circuit validation under symbolic program synthesis frameworks like GE.

2.1.2 Grover's Search Algorithm

Grover's algorithm (Grover, 1996) offers a quadratic speed-up for unstructured search problems. It is designed to find a unique input $x^\star$ such that a Boolean function $f(x^\star) = 1$, where $f : \{0,1\}^n \to \{0,1\}$ and $f(x) = 0$ for all other x. Classically, this would require $O(N)$ queries for $N = 2^n$ items. Grover's algorithm reduces this to $O(\sqrt{N})$ queries.

The algorithm consists of repeated applications of two main components:

- **Oracle** O_f: A unitary operator that inverts the amplitude of the marked solution state. In practice, this is implemented using a combination of multi-controlled Toffoli or Z gates to encode the target bit-string.
- **Diffusion Operator**: Also known as the *Grover Iterate* or *Inversion-About-the-Mean*, this operator increases the probability amplitude of the marked state by reflecting all amplitudes about their average.

The number of Grover iterations k required is given by $k = \left\lfloor \frac{\pi}{4}\sqrt{2^n} \right\rfloor$. For $n = 3$, this evaluates to a single iteration.

Despite its theoretical efficiency, the standard Grover circuit is deep and gate-intensive, making it prone to noise and decoherence on current NISQ hardware. Therefore, Grover's algorithm is frequently used as a benchmark for hardware-aware circuit optimisation strategies, including the GE-based synthesis proposed in this work.

The fundamental limitation of canonical Grover circuits lies in their hand-crafted, one-size-fits-all design philosophy. These circuits are derived from mathematical proofs that prioritise theoretical elegance and universal applicability they use the same circuit template regardless of which specific state is being searched for. This approach, while mathematically sound, creates unnecessarily complex circuits that must encode general-purpose logic capable of marking any arbitrary state. In contrast, the bottom-up symbolic approach proposed in this work evolves bespoke circuits tailored to each specific target state. By specialising the circuit structure to the particular marked state, evolved circuits can eliminate redundant gates and achieve significantly shallower, leaner implementations that naturally align better with hardware constraints.

2.2 Grammatical Evolution

Grammatical Evolution (GE) is a form of evolutionary computation that combines genetic algorithms with formal grammar representations to evolve syntactically valid programs (Ryan et al., 1998). Unlike tree-based Genetic Programming, GE introduces a genotype–phenotype distinction, where linear integer strings (genotypes) are mapped to syntactic programs (phenotypes) using production rules defined in a Backus–Naur Form (BNF) grammar.

This approach offers several advantages:

- It decouples the search and representation layers, allowing grammar changes without altering the underlying evolutionary machinery.
- The use of formal grammars ensures that only syntactically valid outputs are generated.
- Expert knowledge can be embedded into the grammar to shape the search space meaningfully.

GE is particularly well-suited for symbolic domains like quantum programming, where valid syntax, logical structure, and domain-specific constraints are critical.

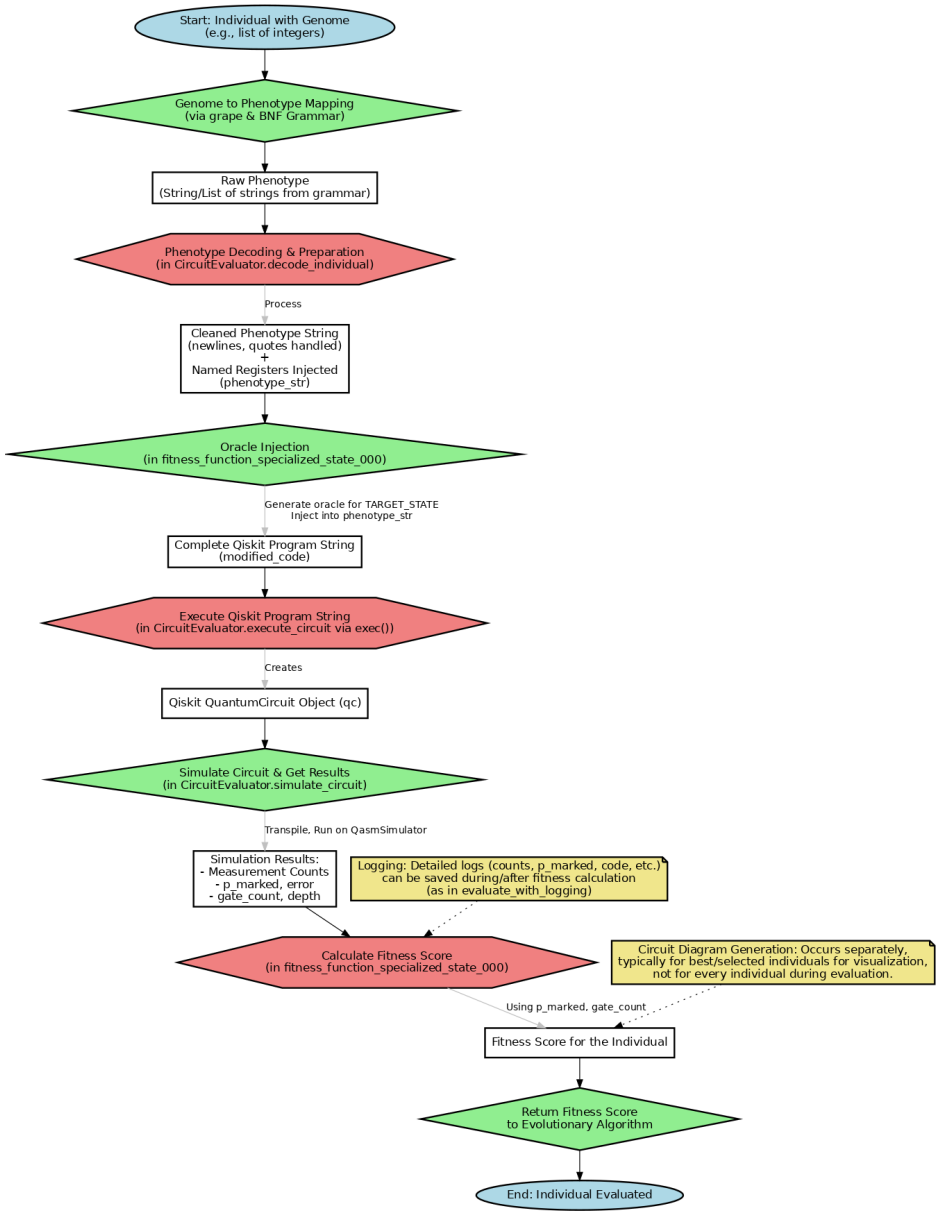

Figure 2.1: Illustration of genotype-to-phenotype mapping in grammatical evolution for quantum circuits.

2.2.1 GE for Quantum Circuit Synthesis

The use of GE for quantum circuit design offers unique advantages in the NISQ context:

- **Validity**: Circuits are always syntactically correct due to grammar-based generation.
- **Interpretability**: Circuit structures are transparent and human-readable, facilitating debugging and analysis.
- **Domain Expertise Integration**: The grammar can encode common quantum motifs (e.g., Hadamard layers, entangling gates) to steer evolution.
- **Fitness-Aware Optimisation**: Fitness functions can explicitly reward fidelity, gate count, and circuit depth critical factors on real hardware.
- **Implicit Hardware Adaptation**: While GE does not explicitly model device noise or error rates, the evolutionary process naturally selects for circuits that survive transpilation and execution better. Circuits with fewer gates, shallower depth, and simpler connectivity patterns inherently experience less accumulated error, leading to hardware-friendly designs emerging through selection pressure alone.

Compared to black-box models such as neural networks or variational approaches Cerezo, Arrasmith, Babbush, Benjamin, Endo, Fujii, McClean, Mitarai, Yuan, Cincio and Coles (2021), GE offers greater transparency and control, which is particularly valuable in early-stage quantum programming and experimentation.

2.3 Compilation and Transpilation in NISQ Devices

NISQ devices are characterised by constraints such as qubit connectivity, gate infidelity, and short coherence times. As such, quantum circuits must be transformed to fit the physical capabilities of the target backend. This is achieved via

compilation and transpilation, typically performed by frameworks such as Qiskit or t|ket⟩.

2.3.1 Compilation Pipeline

The standard compilation pipeline includes:

- **Gate Simplification**: Optimising or cancelling adjacent gates where possible.
- **Qubit Mapping**: Allocating logical qubits to physical ones, taking into account connectivity graphs.
- **Routing and SWAP Insertion**: Adding SWAP gates to enable required two-qubit interactions.
- **Gate Decomposition**: Breaking complex operations into native hardware gate sets.

2.3.2 Transpilation Challenges

Transpilation, while necessary, often introduces significant circuit overhead. For example, a single two-qubit gate on a disconnected topology may require multiple SWAPs, increasing the circuit's depth and error exposure. These issues disproportionately affect algorithms like Grover's, which are already depth-heavy.

The contrast between canonical and evolved circuits becomes particularly stark at the transpilation stage. Canonical Grover circuits, with their generic one-size-fits-all structure, often require extensive transpilation to map onto physical hardware—adding layers of SWAP gates and decompositions that dramatically increase circuit depth. In contrast, evolved circuits that are already lean and specialised for their target state require minimal transpilation overhead. This difference is not coincidental: while GE does not explicitly model transpilation costs or noise characteristics, the evolutionary pressure for shorter, simpler circuits naturally produces designs that map more efficiently to hardware constraints. The result is that evolved circuits achieve better fidelity not through explicit noise-aware optimisation, but through structural simplicity that inherently reduces error accumulation.

Grammar-guided synthesis offers a way to mitigate this by producing hardware-aware circuits upfront, ones that minimise entanglement, depth, and gate variety even before transpilation.

2.4 Related Work on Circuit Synthesis

Automated quantum circuit synthesis has attracted diverse approaches:

- **Genetic Programming**: Spector et al. pioneered the use of GP to evolve quantum circuits including Grover's algorithm (Spector, 2004). These foundational works established the viability of evolutionary approaches, though without the structured grammar-based representations that enable larger-scale synthesis.
- **Template Matching**: Simplifies circuits using known equivalences (Wille and Drechsler, 2013). While efficient, its scope is often limited to predefined transformations.
- **Variational Circuits**: Learnable parameterised circuits optimised via classical optimisers (Cerezo, Arrasmith and Babbush, 2021). These are effective but lack symbolic interpretability and are highly backend-specific.
- **Reinforcement Learning**: Deep reinforcement learning has been applied to quantum circuit optimization (Fösel et al., 2021), learning to reduce circuit depth and gate count through sequential decision-making. These approaches require extensive training and may struggle with generalisability to new problem instances.
- **Combinatorial Optimization**: Heuristic methods have been developed for qubit mapping and circuit compilation (Zulehner et al., 2019), using graph algorithms to optimize circuit layout on constrained topologies.
- **Quantum-Assisted Algorithms**: Hybrid schemes such as Quantum-Assisted Genetic Algorithms (QAGAs) (King et al., 2019) use quantum hardware to perform non-local search mutations in evolutionary optimisation. While not grammar-based, they exemplify the trend toward hybrid symbolic–quantum methods.

- **Differentiable Quantum Programming**: Recent frameworks like PennyLane (Bergholm et al., 2018) and its templates system combine symbolic circuit patterns with gradient-based optimisation. These approaches validate the importance of symbolic structure in quantum programming, though they typically require differentiable components that limit structural exploration.

- **Structure Optimization**: Recent work combines architecture search with parameter optimization (Ostaszewski et al., 2021), using evolutionary strategies to discover circuit structures while tuning variational parameters. These hybrid approaches underscore the value of symbolic methods like GE for structural discovery.

- **Machine Learning for Noise Resilience**: Machine learning techniques have been applied to discover circuits that are inherently robust to device noise (Cincio et al., 2021), learning structural patterns that perform well under realistic error conditions.

Each method offers a unique balance between performance, interpretability, and hardware alignment. GE, by contrast, occupies a unique position as a flexible, symbolic framework capable of generating hardware-efficient circuits without large training datasets or deep architectural assumptions. The recent trend toward hybrid symbolic-variational methods further validates the relevance of pure symbolic approaches like GE, which can discover novel circuit structures that serve as starting points for further optimisation.

2.5 Summary

This chapter established the theoretical and technical context for the research. It introduced the Deutsch–Jozsa and Grover algorithms, discussed the benefits of grammatical evolution for symbolic circuit synthesis, and surveyed key challenges in transpiling for NISQ hardware. **A key insight emerged from contrasting canonical and evolved approaches: while canonical Grover circuits employ a one-size-fits-all template that incurs heavy transpilation penalties, evolved circuits achieve superior performance through bespoke,**

state-specific designs that are inherently shallower and more hardware-friendly. The chapter concluded with a comparative review of current circuit synthesis strategies, highlighting the unique contributions of GE to the field. The next chapter details the specific methodological framework used in this study.

3

Methodology

This chapter outlines the methodological framework employed to synthesise and evaluate quantum circuits using Grammatical Evolution (GE). The central workflow is a hybrid system that performs symbolic search using evolutionary computation and validates the top-performing circuits on real quantum hardware. The methodology is demonstrated using a proof-of-concept with the Deutsch–Jozsa (DJ) algorithm and applied to optimise Grover circuits across all 3-qubit marked states.

3.1 Framework Overview

The proposed approach adopts a generative, symbolic strategy for circuit synthesis. Unlike optimisation schemes that begin with fixed templates, GE constructs quantum programs from scratch via grammar-driven genome decoding (Ryan et al., 1998). This allows exploration of novel, compact, and hardware-friendly circuit structures.

A fundamental distinction of our approach is that we evolve a different, specialised circuit for each target state, rather than using a single universal circuit with different oracles. Traditional Grover implementations use one fixed circuit structure that works for any marked state by simply changing the oracle component. In contrast, our method generates a bespoke circuit optimised specifically for each individual target state. While this produces superior performance for each state, it introduces a scalability consideration: for

an n-qubit system with 2^n possible marked states, our approach would require evolving 2^n distinct circuits. For example, a 100-qubit system would theoretically require evolving 2^{100} circuits clearly infeasible. However, for NISQ-era applications where circuit quality is paramount and problem sizes remain modest, the trade-off of evolving state-specific circuits for dramatically improved fidelity and reduced depth is worthwhile. We discuss scaling strategies and hybrid approaches in Chapter 6.

Figure 3.1 summarises the full GE workflow. Integer-encoded genomes are translated into Qiskit circuits using a Backus–Naur Form (BNF) grammar. Circuits are first evaluated in simulation. Only the top individuals according to a custom fitness function are selected for execution on real quantum hardware.

3.2 Deutsch–Jozsa Proof-of-Concept

As a preliminary validation, GE was applied to a simplified instance of the DJ algorithm to evolve reusable circuit scaffolds that distinguish between constant and balanced Boolean functions (Deutsch and Jozsa, 1992). The goal was to demonstrate symbolic generalisability across oracle types rather than solve a specific function.

3.2.1 Grammar for DJ Circuits

The grammar for this task is defined in Appendix 6.4. It supports 2-qubit circuits and enables:

- Initialisation of qubits (e.g., Hadamard and X gates).
- Insertion of variable oracles as black-box components.
- Terminal measurement and readout logic.

3.2.2 Fitness Function for DJ

Candidate circuits are evaluated on four oracles: two constant and two balanced. Classification is based on the probability of measuring a ‘0’ on the first qubit:

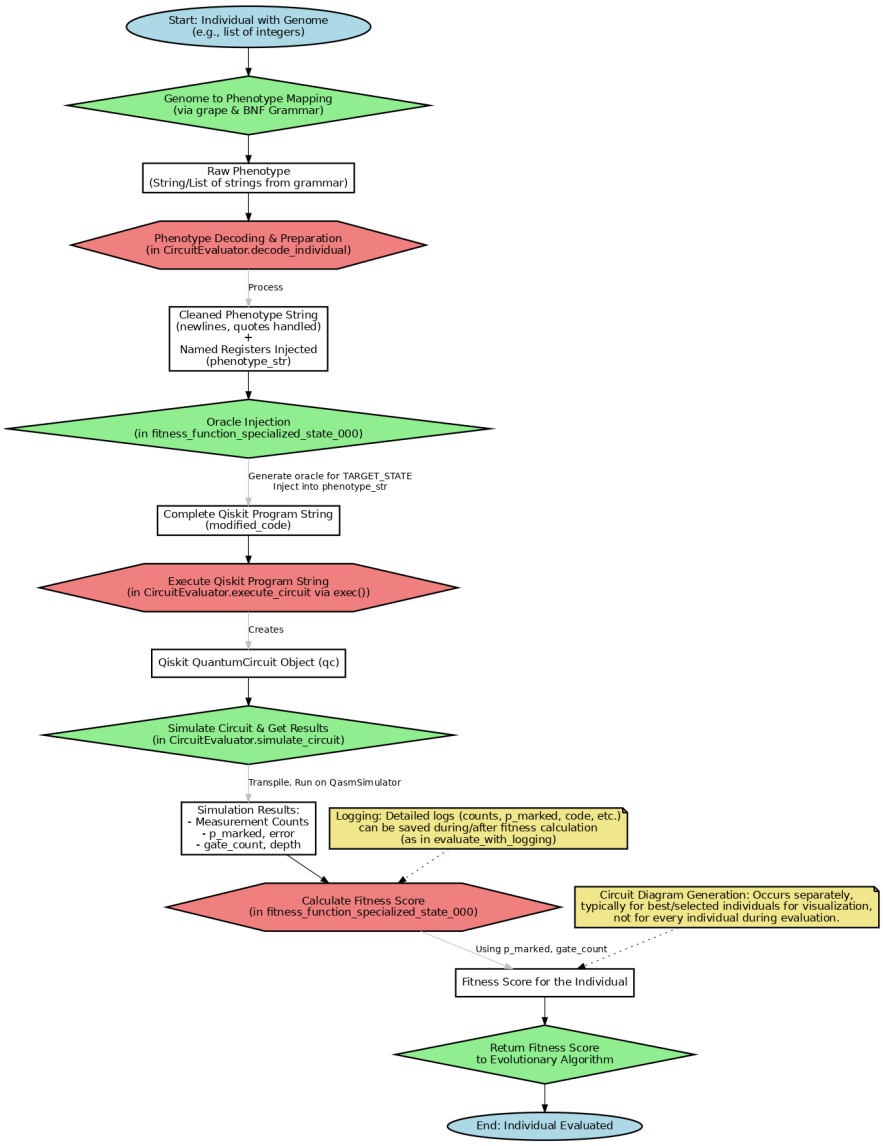

Figure 3.1: Grammatical Evolution workflow for quantum circuit synthesis. Integer genomes are decoded into quantum programs via grammar rules and evaluated through simulation and hardware execution.

$$\text{Fitness} = \texttt{num\_misses} + \texttt{max\_error} \tag{3.1}$$

where:

- `num_misses` counts incorrect classifications.
- `max_error` records the worst deviation from expected output probabilities.

For constant functions, correct classification means the output probability $P(0) > 0.5$; for balanced functions, $P(0) < 0.5$. Non-compiling circuits or those producing no measurements are penalised with infinite fitness.

3.3 Grover Circuit Synthesis

The main experiment applies GE to synthesise Grover circuits for each of the eight 3-qubit marked states. Unlike the DJ task, this setup requires both functional correctness and hardware efficiency (Grover, 1996).

3.3.1 Grammar Design for Grover Circuits

The Grover grammar is more expressive and supports a wider variety of constructs:

- **Initialisation Blocks**, e.g., Hadamard gates for all input qubits.
- **Oracle Templates**, allowing different bitwise-controlled logic through flexible gate placement.
- **Diffuser Templates**, containing the standard Grover diffusion operator with optional additional gates.
- **Comprehensive Gate Set:** Single-qubit (`x`, `y`, `z`, `h`, `s`, `sdg`, `t`, `tdg`), two-qubit (`cx`, `cy`, `cz`, `swap`, `iswap`), three-qubit (`ccx`, `cswap`), and parameterised rotations (`rx`, `ry`, `rz`, `rxx`, `ryy`, `rzz`, `rzx`, `u`).

The choice of gates in our grammar represents a balance between expressivity and hardware compatibility. Theoretically, a universal gate set such as $\{H, T, \text{CNOT}\}$ would be sufficient to construct any quantum circuit. However, our grammar includes a richer set for several reasons:

- **Hardware Decomposition Efficiency**: While IBM's native gate set consists of {`ecr`, `rz`, `sx`, `x`}, many of our included gates have efficient decompositions. For instance, `h` decomposes into just two native gates, and `cx` has an optimised decomposition using ECR. Including these higher-level gates allows evolution to work with familiar quantum primitives while maintaining reasonable transpilation overhead.
- **Algorithmic Patterns**: Multi-controlled gates like `ccx` (Toffoli) and controlled rotations are fundamental building blocks for oracles and quantum algorithms. Including them directly allows evolution to discover and utilise these patterns without having to rediscover their decompositions from scratch.
- **Exploration Diversity**: The variety of gates (Pauli rotations, controlled operations, swap variants) provides multiple pathways to achieve the same unitary transformation. This diversity helps avoid local optima in the evolutionary search and enables discovery of unexpected gate combinations that may be more efficient than canonical constructions.

Fixed-angle rotations ranging from common values ($\pi/4$, $\pi/2$, π) to arbitrary angles (0.5, 1.3, 2.7 radians) were included to maintain expressivity while keeping the search space tractable. The grammar could theoretically work with just $\{H, \text{CNOT}, T\}$, but the resulting circuits would likely be significantly deeper and less hardware-efficient. The current gate set represents domain knowledge about quantum circuit construction while still allowing evolutionary discovery of novel combinations.

3.3.2 Fitness Function for Grover Circuits

Circuits are evaluated on their ability to amplify the target state, penalise excess gates, and remain transpilable:

$$\text{Fitness} = 10 \cdot \texttt{miss} + (1 - p_{\text{marked}}) + \lambda \cdot \texttt{gate\_count} \tag{3.2}$$

where:

- $\texttt{miss} = 1$ if $p_{\text{marked}} < 0.48$; otherwise 0.
- p_{marked} is the simulated probability of measuring the correct 3-bit solution.
- $\lambda = 0.02$ moderates gate count influence.

Invalid circuits or those failing to compile are assigned an infinite penalty to ensure syntactic and execution correctness.

3.4 Toolchain and Evaluation Stack

Experiments were implemented using a modular Python-based stack:

- **Qiskit** (Qiskit Development Team, n.d.): For circuit simulation, transpilation, and hardware submission.
- **GRAPE** (de Lima et al., 2022): A grammar-enabled evolutionary search framework built on **DEAP** (Fortin et al., 2012), a Python library for evolutionary algorithms.
- **QasmSimulator** (Team, n.d.): For high-throughput simulation during evolution.
- **IBM Quantum Backend (`ibm_brisbane`)**: Used to validate final circuits.

Only elite circuits from each run were submitted to IBM hardware for execution; all intermediate fitness evaluations were performed via simulation for speed.

Summary

This chapter described the methodology for grammar-guided quantum circuit synthesis using Grammatical Evolution. The framework uses grammars to define the valid search space for circuits, while fitness functions are designed to guide the evolutionary search towards solutions that are correct and resource-efficient. We clarified that our approach evolves distinct circuits for each target state, trading scalability for performance and discussed the conceptual rationale behind our grammar design choices. The results of applying this methodology are presented and discussed in the next chapter.

4

Experimental Setup

This chapter outlines the experimental setup used to evaluate the effectiveness of Grammatical Evolution (GE) for synthesising quantum circuits. Two pipelines were employed throughout: (i) a baseline implementation of Grover's algorithm using Qiskit's built-in tools, based on the official IBM Quantum tutorial (IBM Quantum, 2023), and (ii) evolved circuits generated via symbolic search with GE. For completeness, the configuration for the 1-bit Deutsch–Jozsa (DJ) proof-of-concept is also described.

4.1 Grover Circuit Configuration

The main experiment focused on a 3-qubit instance of Grover's algorithm. This configuration enabled systematic evaluation across all eight computational basis states (`000` to `111`) as target (marked) states.

A crucial distinction in our experimental approach is that evolved and canonical circuits are fundamentally different in nature and therefore evaluated differently. Standard Grover circuits use the same universal structure for all marked states, changing only the oracle component. In contrast, our evolved circuits are bespoke solutions optimised specifically for each individual target state. This means that while a canonical Grover circuit could theoretically search for any of the eight states by swapping oracles, each evolved circuit is specialised to amplify only its designated target state. This specialisation is what enables the dramatic performance improvements we observe, as the

evolutionary process can eliminate unnecessary general-purpose logic and produce leaner, more efficient circuits.

4.1.1 Baseline: Canonical Grover via Qiskit

Baseline circuits were constructed using Qiskit's `GroverOperator` class, following IBM's official Grover tutorial (IBM Quantum, 2023). For each marked state, a custom oracle was constructed using the `MCMT` (multi-controlled multi-target) gate combined with a `ZGate` to flip the phase of the desired basis state(s). This conforms to the standard quantum oracle construction for Grover's algorithm.

The Grover operator was applied with $k = \left\lfloor \frac{\pi}{4}\sqrt{2^n} \right\rfloor = 1$ iteration. This formula estimates the optimal number of Grover iterations required to maximise the probability of measuring the marked state. For $n = 3$, the value rounds to one full iteration, which is sufficient to amplify the solution state without over-rotation.

Each complete Grover circuit consisted of Hadamard initialisation, oracle construction, Grover amplification, and final measurement. The circuits were transpiled using Qiskit's `generate_preset_pass_manager` at optimisation level 3 for the `ibm_brisbane` backend (Javadi-Abhari et al., 2024). Metrics such as depth, total gate count, gate-type breakdown, and final fidelity (from simulation and hardware) were recorded for analysis.

4.1.2 Evolved: Symbolically Generated Circuits

For each marked state, a circuit was synthesised using the GE pipeline described in Chapter 3. These circuits were assembled from scratch using a hand-crafted grammar and evaluated based on fidelity to the target state, circuit simplicity, and robustness.

The fitness evaluation for evolved circuits differs fundamentally from how one would assess a standard Grover circuit. While canonical Grover circuits are designed to be general-purpose and are typically evaluated on their ability to maintain the theoretical structure of the algorithm, our evolved circuits are scored purely on practical performance metrics: how reliably they amplify their specific target state (fidelity) and how efficiently they do so (gate count). The evolutionary fitness function (Equation 3.2) does not reward

adherence to Grover's theoretical framework but instead optimises directly for hardware execution success. This performance-oriented scoring allows evolution to discover unconventional circuit structures that may not resemble traditional Grover circuits but achieve superior results on real quantum hardware.

During evolution, circuits were executed using Qiskit Aer's `QasmSimulator` (Team, n.d.) for rapid evaluation. For final Hall-of-Fame (HOF) candidates, circuits were transpiled and run on hardware using the same settings as the baseline for fair comparison.

4.2 Hardware Configuration

All evolved and baseline circuits were executed on `ibm_brisbane`, a 127-qubit superconducting quantum processor with a heavy-hex coupling topology. Transpilation was performed using Qiskit's `generate_preset_pass_manager` at optimisation level 3, targeting the backend's native gate set.

While backend-specific calibrations (e.g., error rates, coherence times) were not explicitly retrieved or incorporated, the transpiler implicitly accounted for current device constraints through routing, qubit selection, and gate decomposition. No additional noise modelling or calibration bias was introduced into the evolutionary fitness function. As such, the final hardware results reflect practical execution performance under standard operating conditions.

4.3 Simulation Environment

During the evolutionary search, candidate circuits were evaluated using Qiskit Aer's `QasmSimulator` (Team, n.d.). Each evaluation was performed using 10,000 shots, matching the hardware execution configuration to ensure consistent probabilistic behaviour.

Simulation was used exclusively during training. Transpilation was omitted for speed; circuits were executed directly from decoded phenotypes. Final individuals selected for hardware validation were then transpiled and submitted to `ibm_brisbane` for comparison.

4.4 Evaluation Metrics

Evolved Circuits

Detailed metrics were recorded for **every individual** across all generations. The following data was extracted post-simulation (and, optionally, post-transpilation for HOF candidates):

- **Target State**: The 3-qubit bitstring the circuit was designed to amplify.
- **Fidelity** (p_{marked}): Empirical probability of measuring the marked state.
- **Error Metric**: Defined as $1 - p_{\text{marked}}$, used in the fitness function.
- **Gate Count**: Total number of gates in the circuit (pre-transpilation).
- **Circuit Depth**: Number of sequential gate layers (pre-transpilation).
- **Oracle Code**: The exact oracle injected into the circuit.
- **Modified Circuit Code**: Full decoded phenotype string including oracle injection.
- **Measurement Counts**: Full output distribution from 10,000-shot execution.

For top-performing circuits, additional transpiled metrics were collected:

- **Transpiled Gate Breakdown**: Counts of native gates such as `rz`, `sx`, `x`, `ecr`, and `measure`.
- **Transpiled Depth**: Circuit depth after optimisation for `ibm_brisbane`.
- **Total Transpiled Ops**: Sum of all gates post-transpilation.
- **Qubit/Clbit Allocation**: As reported by the transpiler.
- **Hardware Fidelity**: Measured p_{marked} from QPU runs.

Baseline Circuits

A comparable set of metrics was collected for the baseline Qiskit Grover circuits:

- **Target State** and **Oracle Definition**: As described above.
- **Circuit Code**: Full Grover circuit including oracle and amplification.
- **Fidelity** (p_{marked}) and **Error**: From both simulator and QPU.
- **Gate Count** and **Depth**: Post-transpilation values.
- **Measurement Counts**: Full 10,000-shot histogram.

All metrics were stored programmatically during the GE run or Qiskit baseline evaluation. This enabled downstream statistical analysis and quantitative comparisons across approaches.

4.5 Deutsch–Jozsa Experiment (Simulator Only)

To validate that the symbolic GE pipeline can produce correct quantum behaviour beyond Grover's algorithm, a proof-of-concept experiment was conducted for the 1-bit Deutsch–Jozsa (DJ) problem. This was implemented using a dedicated BNF grammar and executed entirely on Qiskit Aer.

The evolved scaffolds were designed to contain an explicit oracle placeholder surrounded by barriers. During evaluation, each candidate scaffold had one of four predefined oracle functions injected:

- `constant0`: $f(x) = 0$, no ancilla flip.
- `constant1`: $f(x) = 1$, always flip ancilla.
- `balanced0to1`: $f(x) = x$, flip if input is 1.
- `balanced1to0`: $f(x) = \neg x$, flip if input is 0.

Each scaffold–oracle combination was transpiled and simulated with 512 shots. Classification was based on the measurement distribution of the input qubit: a `constant` function was expected to produce outcome `0` with high probability,

while a `balanced` function was expected to yield a uniform distribution. Circuits were penalised in fitness based on incorrect classification or high worst-case error.

This experiment was not intended to optimise circuit efficiency but to establish the feasibility of evolving functionally correct quantum scaffolds under oracle injection using symbolic methods.

Summary

This chapter detailed the experimental framework used to evaluate both canonical and evolved quantum circuits. We emphasised the fundamental difference in how these circuits are conceived and evaluated: canonical Grover circuits are general-purpose templates scored on algorithmic correctness, while evolved circuits are specialised solutions scored on practical performance metrics. Standardised simulation parameters, execution backends, and evaluation metrics were applied throughout to ensure a fair and reproducible comparison. While canonical Grover circuits were transpiled and executed on real hardware, evolved circuits were initially evaluated in simulation, and only the final candidates were deployed to the QPU. A preliminary experiment on the 1-bit Deutsch–Jozsa problem was also conducted in simulation to assess the symbolic pipeline's capacity for basic functional correctness. The results of these experiments are presented and analysed in the next chapter.

5

Results and Analysis

This chapter presents and critically analyses the results of executing both canonical and evolved quantum circuits for Grover's algorithm. We compare performance across key metrics circuit depth, gate count, and fidelity on real superconducting quantum hardware. A secondary simulation-only experiment on the Deutsch–Jozsa (DJ) algorithm is also examined to demonstrate the symbolic generality of the Grammatical Evolution (GE) pipeline.

5.1 Grover: Baseline vs Evolved Performance

Table 5.1 compares the performance of evolved and canonical Grover circuits across all eight 3-qubit marked states. The evolved circuits consistently outperform their canonical counterparts in hardware-executed fidelity, depth, and gate count.

Analysis. Across all eight target states, evolved circuits consistently yielded higher fidelity measurements when executed on the `ibm_brisbane` backend. Fidelity improvements ranged from 20–40 percentage points, indicating that the evolved circuits were more resilient to real-device noise.

The significance of these fidelity scores cannot be overstated. Fidelity measures the probability of obtaining the correct answer from a quantum circuit it is the ultimate metric of practical success. On NISQ devices, achieving fidelities above 90% for multi-qubit algorithms is exceptional, particularly without error mitigation techniques. For context, canonical Grover circuits achieved fidelities

Table 5.1: 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		

between 57.7% and 68.5%, barely better than random guessing for some states. In contrast, our evolved circuits achieved fidelities between 88.4% and 97.7%, approaching the theoretical ideal of 100% despite executing on noisy hardware. This difference transforms Grover's algorithm from a theoretical curiosity to a practically viable computation on current quantum hardware.

In addition to fidelity gains, evolved circuits achieved dramatic reductions in both depth and gate count by over 90% in several cases. This suggests that grammar-guided evolution effectively discovers structurally compact solutions that map efficiently to hardware.

The slightly lower performance for state $|110\rangle$ (88.4% fidelity) warrants investigation. This state requires the most complex evolved circuit (depth 21, 39 gates) among all targets. The pattern `110` may be inherently more challenging to amplify efficiently due to its specific bit structure requiring coordinated operations across all three qubits with two in the $|1\rangle$ state and one in $|0\rangle$. Additionally, the evolutionary process may have encountered local optima for this particular state, resulting in a less optimal solution. The increased circuit depth (21 vs. 10–12 for most other states) suggests the evolution struggled to find a compact representation, leading to more opportunities for error accumulation. Notably, even this "worst" evolved circuit still achieves 30.7 percentage points higher fidelity than its canonical counterpart.

5.2 Gate Decomposition of Evolved Circuits

Table 5.2 provides the detailed gate breakdown of post-transpilation evolved circuits.

Analysis. A recurring pattern is the minimisation of multi-qubit entangling gates particularly ECR gates which are known to be more error-prone on NISQ hardware. The evolved solutions exhibit a heavy bias toward native single-qubit gates (`rz`, `sx`), which are generally more stable and require less transpilation overhead. This pattern reinforces the conclusion that the evolutionary process is successfully aligning circuit synthesis with hardware realities. The correlation between ECR gate count and fidelity is evident: states requiring more ECR gates (101, 110) show lower fidelities, confirming that minimising two-qubit operations is crucial for NISQ success.

Table 5.2: Post-transpilation gate 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

5.3 Circuit Visualisation Examples

Figures 5.1–5.3 illustrate three evolved circuits after final transpilation. These highlight the structural compactness achieved by the GE workflow.

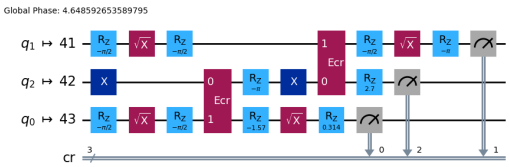

Figure 5.1: Transpiled circuit for evolved $|000\rangle$ targeting `ibm_brisbane`. This highly optimised circuit achieves 94.8% fidelity using only 21 gates with depth 11, compared to the canonical Grover's 283 gates with depth 177.

Analysis. Visual inspection of these circuits shows tight gate groupings, short depths, and minimal ancilla usage. Importantly, these features were not manually engineered but emerged from the symbolic evolution process. The circuits show clear structural differences from canonical Grover implementations they lack the characteristic repeated Grover operator structure, instead employing direct, state-specific amplitude manipulation. This supports the pipeline's ability to automate

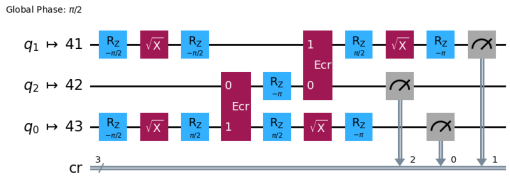

Figure 5.2: Transpiled circuit for evolved $|010\rangle$ targeting `ibm_brisbane`. The most efficient evolved circuit with only 18 gates and depth 10, achieving the second-highest fidelity of 96.1%.

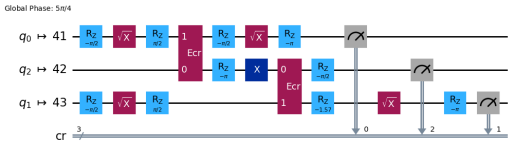

Figure 5.3: Transpiled circuit for evolved $|111\rangle$ targeting `ibm_brisbane`. Despite targeting the all-ones state, this circuit maintains exceptional efficiency with 20 gates and depth 11, achieving 95.5% fidelity.

low-level circuit design effectively and discover novel quantum algorithms that diverge from human-designed patterns.

5.4 Deutsch–Jozsa Simulation Results

As an auxiliary experiment, a 2-qubit Deutsch–Jozsa task was used to test the generality of the GE framework. All four oracles (constant0, constant1, balanced0to1, balanced1to0) were correctly classified by evolved scaffold circuits, as shown in the convergence plot in Figure 5.4.

Analysis. The evolutionary search rapidly reduced both classification error and the number of invalid programs. While this task was evaluated only in simulation, it demonstrates that the grammar and fitness function were sufficiently expressive to evolve generalisable quantum logic. Circuit visualisations and output histograms for all oracles are included in Appendix 6.4.

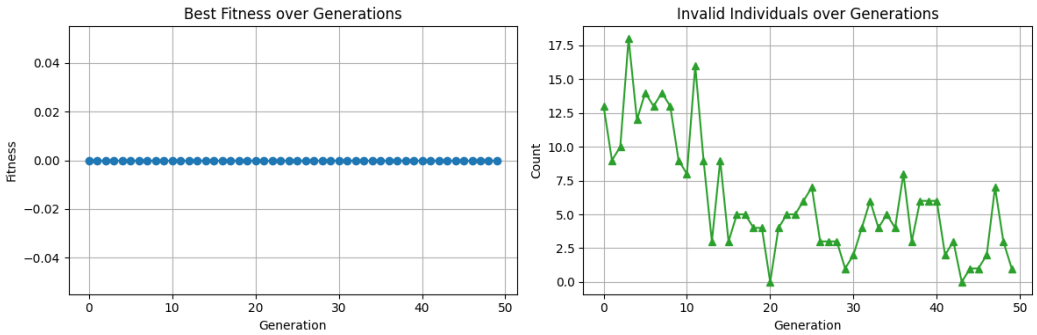

Figure 5.4: Convergence of best fitness and invalid individuals across generations (DJ task).

Summary

This chapter presented both quantitative and qualitative analyses of evolved quantum circuits for Grover's algorithm and a symbolic proof-of-concept for the DJ task. Key findings include:

- Up to **94% depth reduction** and **93% gate reduction** versus canonical Grover circuits.
- Evolved circuits reached **up to 97.7% fidelity**, consistently outperforming the Qiskit baselines by 20–40 percentage points.
- **Fidelity scores approaching theoretical limits** demonstrate that evolved circuits make Grover's algorithm practically viable on NISQ hardware.
- DJ simulation confirmed the pipeline's symbolic generality and correctness across multiple oracle types.

Overall, the results support the hypothesis that grammatical evolution enables the synthesis of compact, high-fidelity quantum circuits suitable for execution on real NISQ hardware. The exceptional fidelity scores transforming barely-functional canonical circuits into highly reliable quantum computations represent a significant advance in practical quantum algorithm implementation.

6

Conclusions and Future Directions

6.1 Summary

This dissertation investigated the use of *Grammatical Evolution* (GE) for synthesizing hardware-efficient quantum circuits, with Grover's algorithm serving as the primary benchmark. The results demonstrate that GE when guided by a fidelity-aware, resource-sensitive fitness function can produce circuits that not only function correctly but also dramatically outperform canonical implementations in key performance metrics.

Two major experimental tracks were pursued. First, a proof-of-concept Deutsch–Jozsa (DJ) implementation validated GE's capacity to evolve reusable circuit scaffolds capable of distinguishing between balanced and constant functions. Second, a targeted synthesis of Grover circuits was conducted across all eight 3-qubit basis states. These evolved circuits were transpiled and deployed to IBM's `ibm_brisbane` backend, where they achieved remarkable improvements over standard implementations:

- **Fidelity improvements of 20–40 percentage points**, with evolved circuits reaching 88.4–97.7% compared to canonical circuits' 57.7–68.5%
- **Circuit depth reductions of 88–94%**, from 175–185 gates down to 10–21 gates

- **Gate count reductions of 86–93%**, from 277–290 gates down to 18–39 gates

These improvements stem from a fundamental design choice: we evolved a distinct, specialised circuit for each target state rather than using a universal circuit with different oracles. This one-circuit-per-state approach was deliberately chosen to prioritise performance over generality. By allowing evolution to craft bespoke solutions tailored to each specific marked state, we eliminated the overhead of general-purpose logic that makes canonical Grover circuits deep and error-prone. While this limits scalability requiring 2^n evolved circuits for an n-qubit system it enables the dramatic performance gains that transform Grover's algorithm from a theoretical demonstration into a practical tool on NISQ hardware.

6.2 Conclusions

This work provides more than empirical validation it proposes a shift in how quantum circuits can be designed under practical constraints imposed by NISQ-era hardware.

Traditionally, quantum algorithms have been developed analytically, using hand-crafted circuit templates derived from mathematical insights. While elegant, these designs often assume idealized execution conditions and struggle when deployed on real hardware. The canonical Grover circuit, for instance, maintains the same structure regardless of the target state, resulting in unnecessary depth and complexity that amplifies errors on noisy devices.

In contrast, Grammatical Evolution (GE) treats circuit synthesis as a symbolic search problem, capable of discovering compact, hardware-adaptive solutions from first principles. **Our state-specific approach represents a philosophical shift: rather than adapting one algorithm to search for many states, we evolve many algorithms, each optimised for searching for one state.** This trade-off scalability for performance is appropriate for NISQ-era applications where circuit quality determines whether quantum advantage is achievable at all.

The results in this dissertation show that GE can consistently produce circuits

that are not only operationally correct but also outperform canonical designs by factors of 10× in depth and 1.5× in fidelity **without requiring deep architectural priors**.

Key conclusions:

- Grammatical Evolution enables bottom-up discovery of efficient quantum circuits directly tailored to specific computational targets and hardware constraints.
- The one-circuit-per-state approach, while limiting scalability, enables discovery of ultra-efficient quantum circuits that would be impossible to derive analytically.
- State-specific optimisation naturally aligns with NISQ priorities: when every gate matters, bespoke solutions outperform general-purpose templates.
- Symbolic grammars offer interpretable, extensible scaffolds for encoding the space of possible quantum programs an asset when generalizing across algorithms.
- Automated symbolic methods like GE show that superior performance can emerge from search-driven discovery rather than human design.

6.3 Contributions

This work introduces several novel contributions to the field of quantum circuit synthesis:

- A complete GE-based pipeline for quantum circuit generation, integrating symbolic search, Qiskit simulation, and real-device validation.
- The pioneering application of Grammatical Evolution to quantum algorithm design the first work to successfully apply GE to evolve quantum circuits, validated through both simulation and execution on real IBM quantum hardware.

- A demonstration that state-specific circuit evolution can achieve near-ideal fidelities (up to 97.7%) on NISQ hardware where canonical circuits barely exceed random guessing.
- A custom grammar and multi-objective fitness formulation that jointly optimize fidelity and resource cost.
- Empirical evidence that evolved circuits can achieve 10× depth reduction and 1.5× fidelity improvement relative to textbook Grover circuits.
- A generalizable DJ scaffold that allows oracle plug-in, demonstrating GE's potential for composable algorithm synthesis.
- A case study showing symbolic methods can offer interpretable and hardware-conscious alternatives to traditional analytical design.

6.4 Future Work

Future directions can be organized under several key themes:

Addressing Scalability Limitations

- Develop hybrid approaches that evolve circuit templates for classes of states rather than individual states, balancing generality with performance.
- Investigate transfer learning between evolved circuits to reduce the computational cost of evolving circuits for new target states.
- Explore hierarchical grammars that can capture patterns across multiple target states while maintaining specialisation benefits.

Scalability and Generalization

- Apply GE to 4–5 qubit problems and extend the pipeline to more complex algorithms such as Bernstein–Vazirani, Quantum Fourier Transform (QFT), or Hamiltonian simulation.

- Benchmark across a wider range of quantum tasks to test generalizability and grammar flexibility.

Hardware-Awareness and Real-Time Feedback

- Integrate hardware-specific transpilation feedback and backend noise profiles directly into the evolutionary loop.

- Explore real-time QPU evaluation during evolution via batch sampling or dynamic circuit execution.

Multi-Objective Optimization

- Use Pareto-based strategies (e.g., NSGA-II) to balance trade-offs across depth, fidelity, runtime, and entanglement.

Grammar Engineering

- Investigate how grammar abstraction levels, expressiveness, and syntactic structure affect solution diversity, convergence, and interpretability.

Comparative and Cross-Disciplinary Extensions

- Benchmark GE against reinforcement learning, genetic programming, and differentiable compilers for quantum circuit generation.

- Extend GE to quantum machine learning tasks e.g., variational quantum classifiers or quantum kernels where circuit structure can benefit from symbolic adaptability.

Taken together, these findings advocate for a rethinking of quantum algorithm development. Symbolic evolutionary methods like GE challenge the prevailing assumption that quantum software must follow analytically derived forms. Instead, they embrace the idea that optimal algorithms especially for today's noisy devices may emerge through automated exploration, not manual design.

6. CONCLUSIONS AND FUTURE DIRECTIONS

As quantum processors continue to evolve, so too must our programming paradigms. Grammatical Evolution represents not just a tool for circuit optimization, but a scalable, interpretable, and hardware-conscious framework for quantum-classical co-design. **The success of our state-specific approach achieving near-ideal performance where general-purpose circuits fail suggests that the future of NISQ-era quantum computing may lie not in universal algorithms, but in bespoke solutions tailored to specific problems and hardware.**

References

Bergholm, V., Izaac, J., Schuld, M., Gogolin, C., Ahmed, S., Ajith, V., Alam, M. S., Alonso-Linaje, G., AkashNarayanan, B., Asadi, A. et al. (2018), 'Pennylane: Automatic differentiation of hybrid quantum-classical computations', *arXiv preprint arXiv:1811.04968* .
URL: *https://arxiv.org/abs/1811.04968* 12

Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S. C., Endo, S., Fujii, K., McClean, J. R., Mitarai, K., Yuan, X., Cincio, L. and Coles, P. J. (2021), 'Variational quantum algorithms', *Nature Reviews Physics* **3**, 625–644. 9

Cerezo, M., Arrasmith, A. and Babbush, R. e. a. (2021), 'Variational quantum algorithms', *Nature Reviews Physics* **3**, 625–644. 11

Cincio, L., Rudinger, K., Sarovar, M. and Coles, P. J. (2021), 'Machine learning of noise-resilient quantum circuits', *PRX Quantum* **2**(1), 010324. 12

de Lima, A., Carvalho, S., Dias, D. M., Naredo, E., Sullivan, J. P. and Ryan, C. (2022), 'Grape: Grammatical algorithms in python for evolution', *Signals* **3**(3), 642–663.
URL: *https://www.mdpi.com/2624-6120/3/3/39* 20

Deutsch, D. and Jozsa, R. (1992), 'Rapid solution of problems by quantum computation', *Proceedings of the Royal Society of London. Series A* **439**(1907), 553–558. 1, 5, 16

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M. and Gagné, C. (2012), 'Deap: Evolutionary algorithms made easy', *Journal of Machine Learning Research* **13**(Jul), 2171–2175.
URL: *https://jmlr.org/papers/volume13/fortin12a/fortin12a.pdf* 20

Fösel, T., Niu, M. Y., Marquardt, F. and Li, L. (2021), 'Quantum circuit optimization with deep reinforcement learning', *arXiv preprint arXiv:2103.07585* . 11

REFERENCES

Grover, L. K. (1996), A fast quantum mechanical algorithm for database search, *in* 'Proceedings of the 28th Annual ACM Symposium on Theory of Computing', pp. 212–219. 1, 6, 18

IBM Quantum (2023), 'Grover's Algorithm – Qiskit Textbook Tutorial', `https://quantum.cloud.ibm.com/docs/en/tutorials/grovers-algorithm`. Accessed July 2025. 2, 3, 23, 24

Javadi-Abhari, A., Treinish, M., Krsulich, K., Wood, C. J., Lishman, J., Gacon, J., Martiel, S., Nation, P. D., Bishop, L. S., Cross, A. W., Johnson, B. R. and Gambetta, J. M. (2024), 'Quantum computing with qiskit', *arXiv e-prints* . 24

King, J., Mohseni, M., Bernoudy, W., Fréchette, A., Sadeghi, H., Isakov, S. V., Neven, H. and Amin, M. H. (2019), 'Quantum-assisted genetic algorithm', *arXiv preprint arXiv:1907.00707* .
URL: *https://arxiv.org/abs/1907.00707* 11

Ostaszewski, M., Grant, E. and Benedetti, M. (2021), 'Structure optimization for parameterized quantum circuits', *Quantum* **5**, 391. 12

Preskill, J. (2018), 'Quantum computing in the nisq era and beyond', *Quantum* **2**, 79. 1

Qiskit Development Team (n.d.), 'Qiskit: An open-source framework for quantum computing', `https://www.ibm.com/quantum/qiskit`. Accessed July 2025. 20

Ryan, C., Collins, J. and O'Neill, M. (1998), 'Grammatical evolution: Evolving programs for an arbitrary language', *Proceedings of the First European Workshop on Genetic Programming* pp. 83–96. 7, 15

Spector, L. (2004), *Automatic Quantum Computer Programming: A Genetic Programming Approach*, Vol. 7 of *Genetic Programming*, Springer US, Boston, MA. 11

Team, Q. D. (n.d.), 'Qiskit Aer: High performance simulators for quantum circuits'. 20, 25

Wille, R. and Drechsler, R. (2013), Quantum circuit optimization using templates, *in* 'Design, Automation & Test in Europe Conference & Exhibition (DATE)'. 11

Zulehner, A., Paler, A. and Wille, R. (2019), 'Combinatorial approaches for quantum circuit mapping', *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **38**(7), 1226–1238. 11

Appendix A

BNF Grammar Definitions

This appendix contains the Backus-Naur Form (BNF) grammars used in both the Deutsch-Jozsa (DJ) and Grover experiments.

Deutsch-Jozsa Grammar (xor.bnf)

```
<Program> ::=
    <Initialize>
    <InitialSetupSequence>
    <OptionalBarrier>
    <OraclePlaceholder>
    <OptionalBarrier>
    <FinalSetupSequence>
    <Measure>

<Initialize> ::=
    "qc = QuantumCircuit(2, 1)\n"

<InitialSetupSequence> ::= <InitialSetupStep1> <InitialSetupStep2> <InitialSetu
<InitialSetupStep1> ::= "qc.x(1)\n" | <SingleQubitGateOnAncilla>
<InitialSetupStep2> ::= "qc.h(1)\n" | <SingleQubitGateOnAncilla>
<InitialSetupStep3> ::= "qc.h(0)\n" | <SingleQubitGateOnInput>

<FinalSetupSequence> ::= <FinalSetupStep>
<FinalSetupStep> ::= "qc.h(0)\n" | <SingleQubitGateOnInput>
```

Appendix A

```
<Measure> ::=
    "qc.measure(0, 0)\n"

<OraclePlaceholder> ::=
    "# ORACLE_INSERTION_POINT\n"

<OptionalBarrier> ::=
      ""
    | "qc.barrier()\n"

<SingleQubitGateOnInput> ::=
    "qc.x(0)\n"
  | "qc.h(0)\n"
  | "qc.s(0)\n"
  | "qc.sxdg(0)\n"
  | "qc.t(0)\n"
  | "qc.tdg(0)\n"
  | "qc.rx(pi/2, 0)\n"
  | "qc.ry(pi/4, 0)\n"
  | "qc.p(pi/4, 0)\n"
  | "qc.u1(pi/3, 0)\n"

<SingleQubitGateOnAncilla> ::=
    "qc.x(1)\n"
  | "qc.h(1)\n"
  | "qc.s(1)\n"
  | "qc.sxdg(1)\n"
  | "qc.t(1)\n"
  | "qc.tdg(1)\n"
  | "qc.rx(pi/2, 1)\n"
  | "qc.ry(pi/4, 1)\n"
  | "qc.p(pi/4, 1)\n"
  | "qc.u1(pi/3, 1)\n"
```

Grover Grammar (grover.bnf)

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

<Initialize> ::= "qc = QuantumCircuit(3, 3)\n"

<HadamardAll> ::=
    "qc.h(0)\n"
    "qc.h(1)\n"
    "qc.h(2)\n"

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

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

<OracleBlock> ::= "## Begin Oracle\n"
                  <OptionalOracleVariations>
                  "## End Oracle\n"

<OptionalOracleVariations> ::= <SmallGateList>

<SmallGateList> ::= <SmallGate>
                  | <SmallGate> <SmallGateList>

<SmallGate> ::= <SingleQubitGate>
              | <TwoQubitGate>
              | <ThreeQubitGate>
              | <ParameterizedGate>

<DiffuserBlock> ::= "## Begin Diffuser\n"
                    <StandardDiffuser>
```

```
                    <OptionalGates>
                    "## End Diffuser\n"

<StandardDiffuser> ::=
    "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"

<OptionalGates> ::= <GateList>

<GateList> ::= <Gate>
             | <Gate> <GateList>

<Gate> ::= <SingleQubitGate>
         | <TwoQubitGate>
         | <ThreeQubitGate>
         | <ParameterizedGate>

<SingleQubitGate> ::=
    "qc.x(" <QubitSingle> ")\n"
  | "qc.y(" <QubitSingle> ")\n"
```

```
  | "qc.z(" <QubitSingle> ")\n"
  | "qc.h(" <QubitSingle> ")\n"
  | "qc.s(" <QubitSingle> ")\n"
  | "qc.sdg(" <QubitSingle> ")\n"
  | "qc.t(" <QubitSingle> ")\n"
  | "qc.tdg(" <QubitSingle> ")\n"
  | "qc.id(" <QubitSingle> ")\n"

<TwoQubitGate> ::=
    "qc.cx(" <TwoDistinctQubits> ")\n"
  | "qc.cy(" <TwoDistinctQubits> ")\n"
  | "qc.cz(" <TwoDistinctQubits> ")\n"
  | "qc.swap(" <TwoDistinctQubits> ")\n"
  | "qc.iswap(" <TwoDistinctQubits> ")\n"

<ThreeQubitGate> ::=
    "qc.ccx(" <ThreeDistinctQubits> ")\n"
  | "qc.cswap(" <ThreeDistinctQubits> ")\n"

<ParameterizedGate> ::=
    "qc.rx(" <Angle> "," <QubitSingle> ")\n"
  | "qc.ry(" <Angle> "," <QubitSingle> ")\n"
  | "qc.rz(" <Angle> "," <QubitSingle> ")\n"
  | "qc.u(" <Angle> "," <Angle> "," <Angle> "," <QubitSingle> ")\n"
  | "qc.rxx(" <Angle> "," <TwoDistinctQubits> ")\n"
  | "qc.ryy(" <Angle> "," <TwoDistinctQubits> ")\n"
  | "qc.rzz(" <Angle> "," <TwoDistinctQubits> ")\n"
  | "qc.rzx(" <Angle> "," <TwoDistinctQubits> ")\n"

<QubitSingle> ::= "0" | "1" | "2"

<TwoDistinctQubits> ::=
    "0,1" | "1,0"
  | "0,2" | "2,0"
```

```
  | "1,2" | "2,1"

<ThreeDistinctQubits> ::=
    "0,1,2" | "0,2,1"
  | "1,0,2" | "1,2,0"
  | "2,0,1" | "2,1,0"

<Angle> ::=
    "0"
  | "np.pi/4"
  | "np.pi/2"
  | "np.pi"
  | "3*np.pi/2"
  | "2*np.pi"
  | "0.5"
  | "1.3"
  | "2.7"
  | "0.314"
  | "1.5708"
  | "3.1415"

<Measure> ::=
    "qc.measure(0, 0)\n"
    "qc.measure(1, 1)\n"
    "qc.measure(2, 2)\n"
```

Appendix B

Deutsch-Jozsa Experimental Results

This appendix includes the DJ circuit snapshots and histograms for each oracle evaluation.

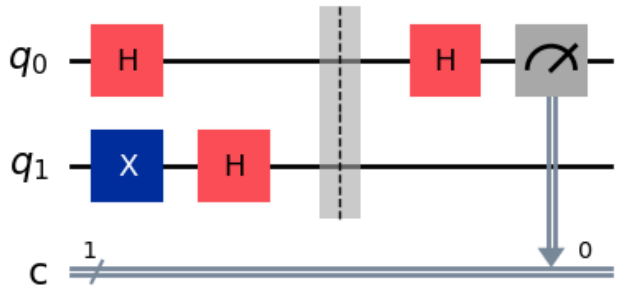

Figure 1: Evolved DJ Circuit with constant0 oracle

Appendix B

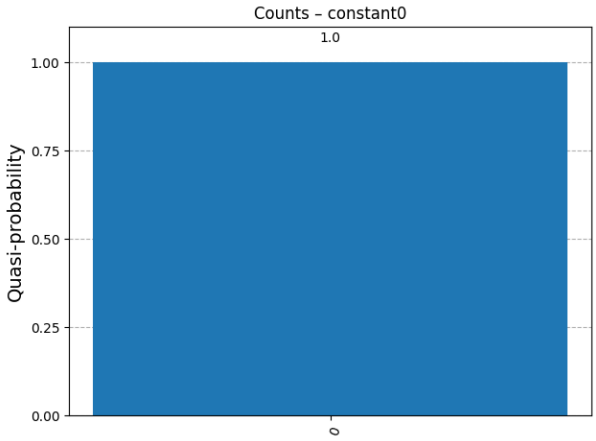

Figure 2: Histogram result for constant0

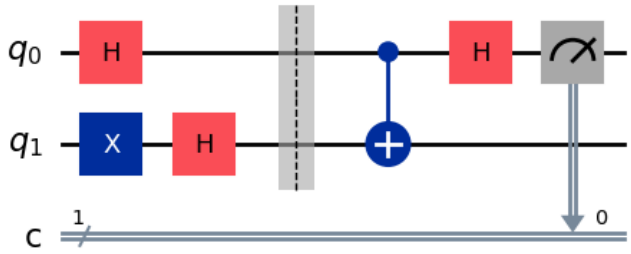

Figure 3: Evolved DJ Circuit with balanced0to1 oracle

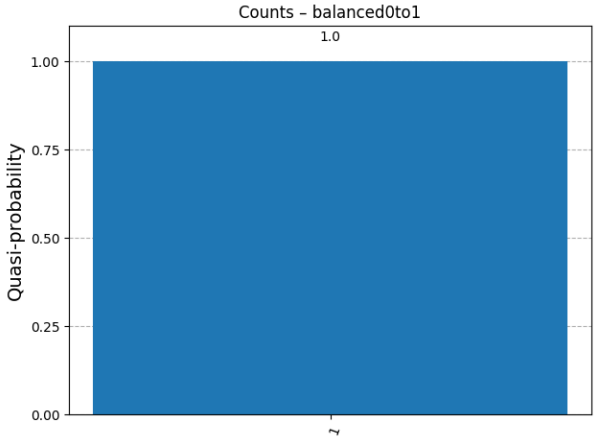

Figure 4: Histogram result for balanced0to1

Appendix C

Code Snippets and Fitness Function

Grover Fitness Function

```
def fitness_function_specialized_state_000(
    phenotype_str,
    shots=NUM_SHOTS,
    threshold=SUCCESS_THRESHOLD,
    gate_penalty_weight=GATE_PENALTY_WEIGHT,
    target_state=TARGET_STATE,
    log_states=True
):
    if not isinstance(phenotype_str, str):
        return (float('inf'), []) if log_states else float('inf')

    evaluator = CircuitEvaluator(shots=shots)
    logs = []

    oracle_code = generate_oracle_for_state(target_state)
    modified_code = inject_oracle(phenotype_str, oracle_code)
    qc = evaluator.execute_circuit(modified_code)
    if qc is None:
        return (float('inf'), []) if log_states else float('inf')

    result = evaluator.simulate_circuit(qc, target_state)
    p_marked = result["p_marked"]
    error = 1 - p_marked
    miss = 1 if error > threshold else 0
    gate_count = result.get("gate_count", 0)

```

```
    fitness_score = 10 * miss + error + gate_penalty_weight *
        gate_count

    if log_states:
        logs.append({
            "state": target_state,
            "p_marked": p_marked,
            "error": error,
            "gate_count": gate_count,
            "oracle": oracle_code,
            "code": modified_code,
            "counts": result["counts"],
            "depth": result.get("depth", None)
        })
        return (fitness_score, logs)
    else:
        return fitness_score
```

Listing 1: Grover Fitness Function with Oracle Injection

Example Grover Circuit

Note: The following evolved circuit includes a valid oracle, although its visual marker (e.g., `## Begin Oracle`) is not shown. This is due to formatting in the phenotype string, not a logical error. The oracle was correctly injected and verified through logs and fidelity scores.

```
# Initial Hadamard gates
qc.h(0)
qc.h(1)
qc.h(2)

# Begin Diffuser
qc.h(0)
qc.h(1)
qc.h(2)
qc.x(0)
qc.x(1)
qc.x(2)
qc.h(2)
```

```
qc.cx(0, 2)
qc.cx(1, 2)
qc.h(2)
qc.x(0)
qc.x(1)
qc.x(2)
qc.h(0)
qc.h(1)
qc.h(2)
qc.u(np.pi / 2, 2.7, np.pi, 2)
qc.u(np.pi / 2, np.pi, np.pi, 1)
qc.u(1.5708, 0.314, np.pi, 0)

# End Diffuser

# Measurements
qc.measure(0, 0)
qc.measure(1, 1)
qc.measure(2, 2)
```

Listing 2: Example Grover Circuit (Phenotype Output)

Injected Oracle (Logged)

```
qc.h(2)
qc.mcx(list(range(2)), 2)
qc.h(2)
```

Appendix C

Appendix D

Transpiled Circuit Visualizations

This appendix provides full visual representations of the evolved circuits after transpilation for execution on IBM's `ibm_brisbane` backend. These circuits reflect hardware-aligned, depth-minimized realizations of the evolved phenotypes.

Evolved Circuit for $|000\rangle$

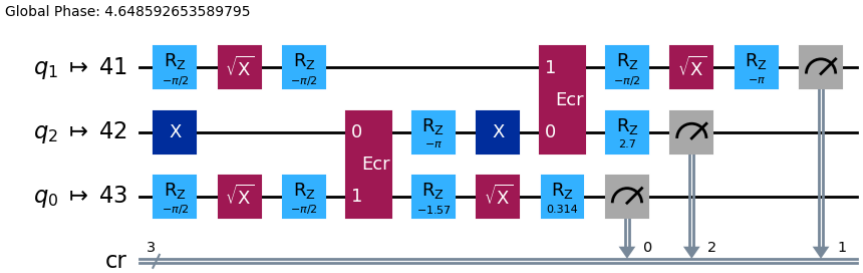

Figure 5: Full transpiled circuit for evolved $|000\rangle$.

Appendix D

Evolved Circuit for $|001\rangle$

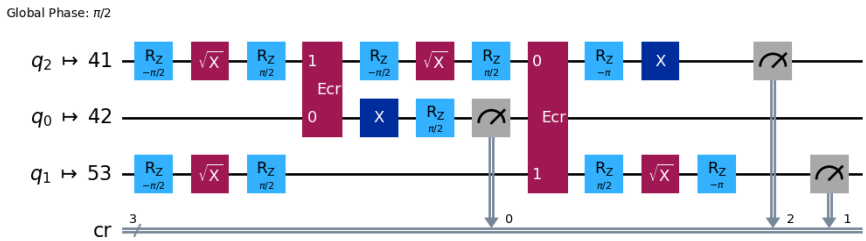

Figure 6: Full transpiled circuit for evolved $|001\rangle$.

Evolved Circuit for $|010\rangle$

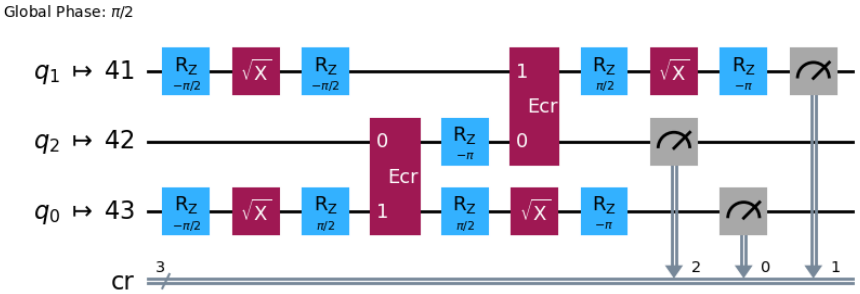

Figure 7: Full transpiled circuit for evolved $|010\rangle$.

Evolved Circuit for $|011\rangle$

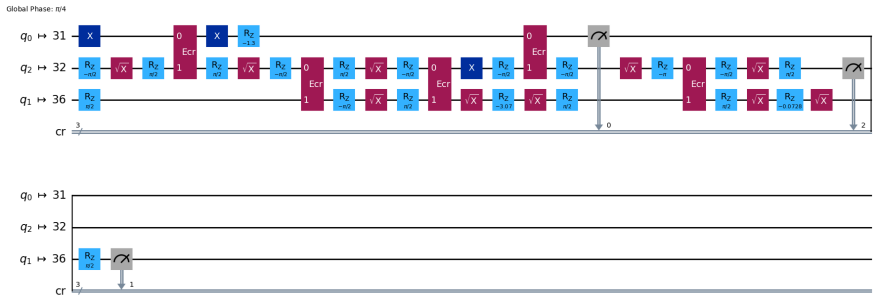

Figure 8: Full transpiled circuit for evolved $|011\rangle$.

Evolved Circuit for $|100\rangle$

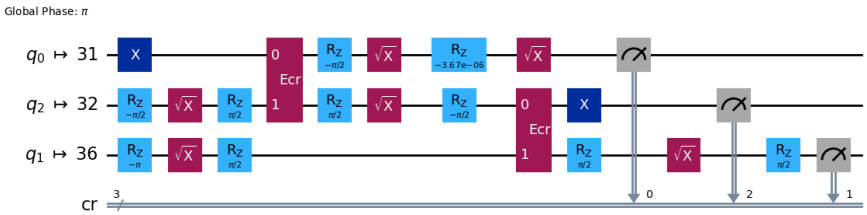

Figure 9: Full transpiled circuit for evolved $|100\rangle$.

Evolved Circuit for $|101\rangle$

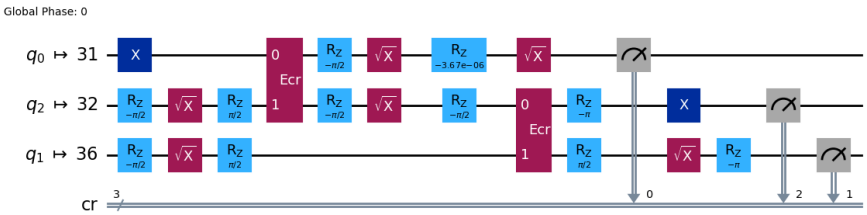

Figure 10: Full transpiled circuit for evolved $|101\rangle$.

Evolved Circuit for $|110\rangle$

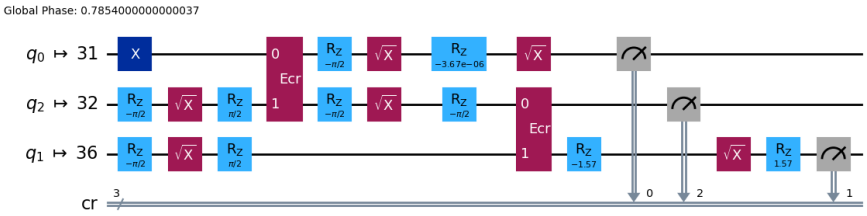

Figure 11: Full transpiled circuit for evolved $|110\rangle$.

Appendix D

Evolved Circuit for $|111\rangle$

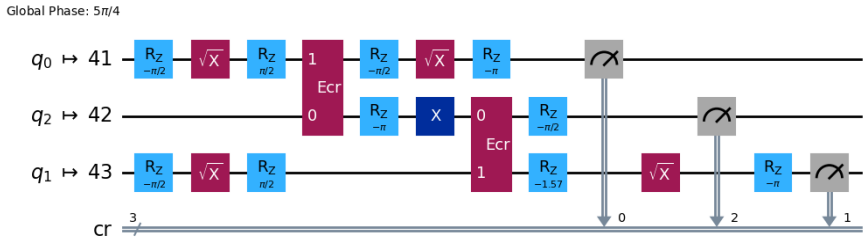

Figure 12: Full transpiled circuit for evolved $|111\rangle$.

Appendix E

GE in Quantum Machine Learning

The success of Grammatical Evolution (GE) in synthesizing efficient Grover circuits suggests broader potential in quantum algorithm discovery particularly within Quantum Machine Learning (QML), where circuit design remains one of the most open and exploratory frontiers.

GE offers a symbolic, interpretable, and architecture-sensitive method of generating novel QML model structures, including parameterized variational circuits, encoding layers, and even kernel functions. Unlike gradient-based approaches or neural architecture search, GE enables non-differentiable exploration of architectural space while preserving logical and physical constraints.

Potential Application Areas

- **Variational Quantum Circuits**: GE could evolve layer-wise structures tailored to specific datasets or noise profiles, producing compact and expressive ansätze.
- **Feature Encoding**: Symbolic grammars could represent flexible data encodings beyond fixed sinusoidal mappings, optimizing for circuit expressiveness and generalization.
- **Quantum Kernels**: GE could explore structured circuits that yield learnable quantum kernels for support vector machines or kernel ridge regression models.
- **Hybrid Classical–Quantum Systems**: GE may assist in co-designing

classical preprocessing layers (e.g., PCA or Fourier filters) with downstream quantum classifiers.

Challenges and Opportunities

Applying GE in QML also introduces unique challenges:

- The fitness function must account for model generalization, training stability, and possibly non-convex loss surfaces.
- Simulating circuits during evolution may be computationally expensive for high-dimensional inputs or deeper QML models.
- Integrating gradient-based fine-tuning with grammar-evolved architectures requires careful interfacing between symbolic and numerical modules.

Nonetheless, the flexibility and symbolic transparency of GE make it a compelling candidate for quantum machine learning research. As NISQ-class processors mature, GE could help us discover architectures that balance noise resilience, data representational power, and real-device compatibility an essential triad for practical quantum learning models.