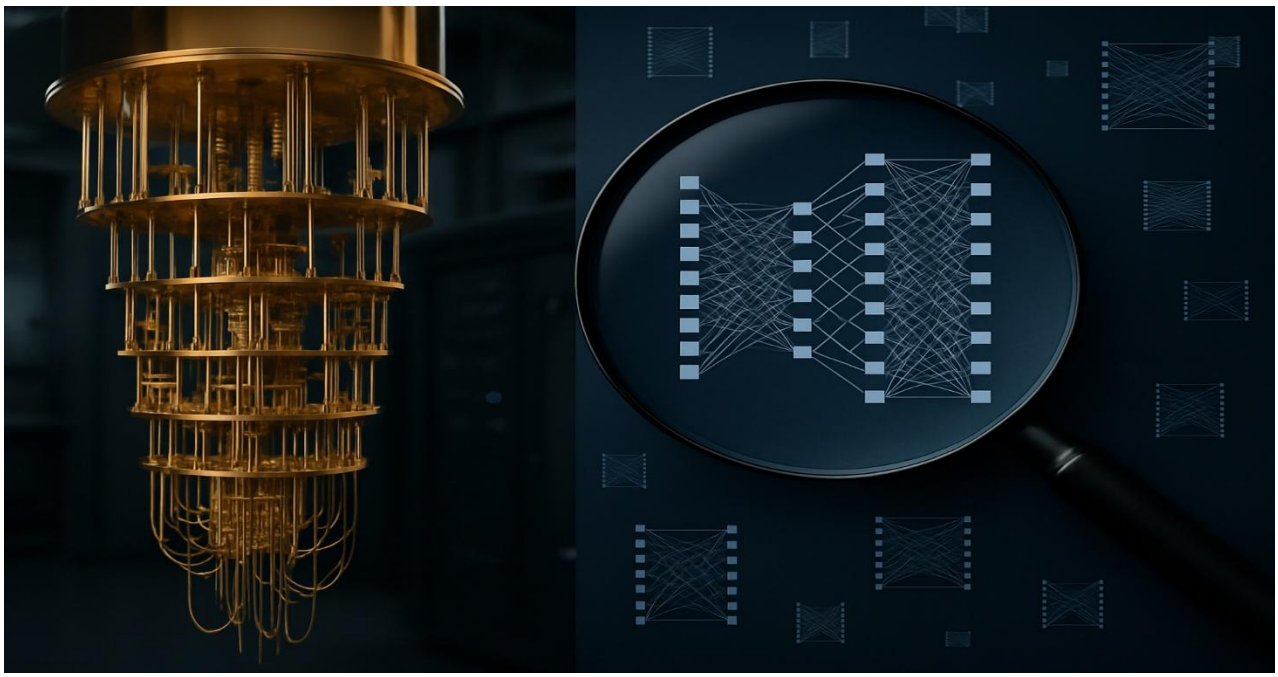
Implementation of Quantum-Inspired Evolutionary Algorithm for Convolutional Neural Architecture Search and a Comparative Study of Generated and Standard CNN Structures Using Various Datasets

Project Thesis in Electromobility ACES

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik
Prof. Dr.-Ing. Jörg Franke



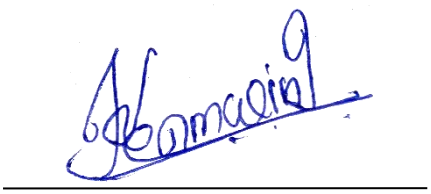
Author: Newin Jolly Kaniampuram, (B.Tech) 23146289

Supervisor: Prof. Dr.-Ing. Jörg Franke
Yufei Feng, M.Sc.

Submission date: 30.06.2025
Project duration: 5 months

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.



Erlangen/Nürnberg, the 30.06.2025

Newin Jolly Kaniampuram

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AutoML	Automated Machine Learning
CCNOT	Controlled-Controlled-NOT (Toffoli gate)
CNN	Convolutional Neural Network
CNOT	Controlled-NOT (quantum gate)
DARTS	Differentiable Architecture Search
EA	Evolutionary Algorithm
FLOPs	Floating Point Operations per Second
GA	Genetic Algorithm
GP	Genetic Programming
MNIST	Modified National Institute of Standards and Technology
MRI	Magnetic Resonance Imaging
NAS	Neural Architecture Search
PSO	Particle Swarm Optimization
QC	Quantum Computing
QEA	Quantum Evolutionary Algorithm
QIEA	Quantum-Inspired Evolutionary Algorithm
QIGA	Quantum-Inspired Genetic Algorithm
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network

1 Introduction

1.1 The Significance and Role of Artificial Intelligence and Deep Learning

Modern smartphones are capable of facial recognition, and virtual assistants such as Amazon's Alexa and Google's Gemini can process and respond to human speech demonstrating the remarkable progress achieved in the field of Artificial Intelligence (AI).. AI enables machines to perform tasks that typically require human intelligence, such as recognizing patterns, making decisions, and processing information [1]. Machine Learning (ML) constitutes a foundational component of Artificial Intelligence (AI), providing computational systems with the capability to learn from data and adapt their behavior based on experience, rather than relying on explicitly defined rules. By analysing vast amounts of information, machines can identify patterns and improve their accuracy over time. Deep Learning (DL) takes machine learning a step further by mimicking the way the human brain processes information. Using neural networks inspired by brain structures, deep learning excels at solving complex problems. Deep learning is transforming industries and shaping the future of technology from self-driving cars to detecting human diseases in medical images [2].

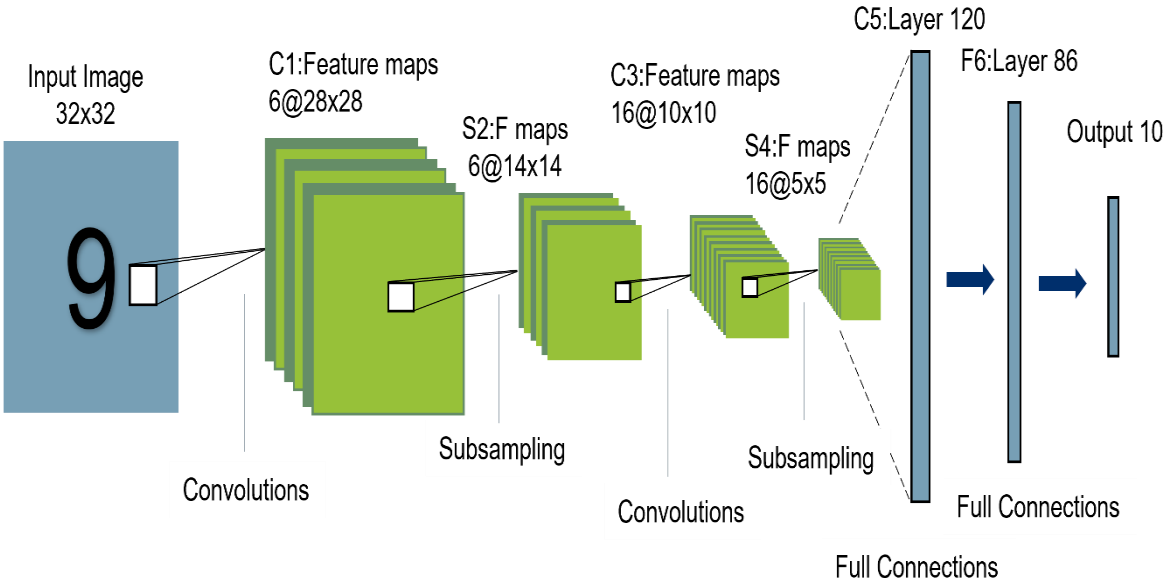


Figure 1:Architecture of LeNet-5 [3] .

Convolutional Neural Networks (CNNs) are a prominent deep learning model designed to handle classification tasks [3]. LeNet-5, introduced in 1998 by Yann LeCun, was the first practical and successful implementation of CNNs for real-world tasks, specifically for recognizing handwritten digits [4]. Later on, AlexNet was developed for the ImageNet challenge in 2012, demonstrated the power of deep learning for large-scale

image classification, significantly advancing applications in object detection and computer viosin tasks[5]. In 2014, VGGNet [6] was introduced, followed by GoogLeNet [7] and ResNet [8] in 2015, and later DenseNet. Each of these architectures brought significant advancements to the field, improving the performance and efficiency of convolutional neural networks. CNNs have shown exceptional performance in tasks ranging from recognizing handwritten digits to identifying objects in diverse and complex real-world environments.

The following are the layers of a Convolutional Neural Network (CNN), as illustrated in Figure 1:

1. Input Layer: Accepts the raw input image, typically in the form of pixel values.
2. Convolutional Layer: Extracts features from the input by applying filters (kernels) that detect patterns such as edges or textures.
3. Pooling Layer: Reduces the dimensions of the feature maps from the convolutional layer, retaining important information.
4. Fully Connected Layer: Combines the extracted features and maps them to the final output, such as class probabilities for classification tasks.
5. Output Layer: Produces the final predictions, such as the category of an image or other target outputs.

These layers together enable the CNN to effectively process and analyse image data.

Designing network architectures tailored for specific tasks remains challenging. This process often requires extensive expertise, and is time-consuming, and relies heavily on manual adjustments [9]. The choice of hyperparameters, such as kernel size, stride, padding, and pooling, plays a critical role in determining the overall performance of a CNN, as these parameters directly influence how the network extracts and processes features from the input data. Traditional neural architecture design procedures mostly rely on human expertise, which is prone to suboptimal outcomes due to the vast and complex search space of possible structures [10]. A search space refers to the complete set of possible solutions that an algorithm can explore during an optimization process. In the context of Convolutional Neural Networks (CNNs), the search space consists of all valid combinations of layer types, architectural structures, and hyperparameter settings that can be used to construct different network architectures.

This limitation highlights the necessity for automated design algorithms capable of generating task-specific network architectures.Neural Architecture Search (NAS) offers a systematic solution to this issue by automating the design of CNNs. NAS techniques require significantly less human intervention since they are data-driven rather than experience-driven [11]. NAS focuses on identifying appropriate CNN layers and their attributes, such as kernel size, the number of feature maps, and types of pooling operations. It formulates the architecture design as an optimization problem, aiming to discover configurations that deliver superior performance. The three operations listed below comprise the NAS workflow:

1. Selecting an architecture from a search space using specific search tactics.
2. Assessing the sampled architecture's performance.
3. Modifying the parameters in light of the performance [12].

Techniques such as reinforcement learning, evolutionary algorithms, random search, Bayesian optimization, and gradient-based methods have all been employed for Neural Architecture Search.

In recent years, BayesNAS has shown Improvements from thousands of GPU-days to 0.2 GPU-day [13]. Reinforcement learning-based NAS techniques frequently require a significant amount of processing power, such as several GPUs or distributed computer clusters [14]. Similarly, traditional evolutionary algorithms often struggle to avoid premature convergence and frequently lack the ability to maintain adequate diversity within their populations. For example, early NAS approaches required weeks of computation on high-performance clusters, rendering them impractical for broader adoption[15].

Despite its promise, Neural Architecture Search (NAS) faces significant computational challenges, as it is resource-intensive and requires considerable time to explore and evaluate potential network architectures. To address these limitations, this thesis explores the use of Quantum-Inspired Evolutionary Algorithms (QIEA) as a promising alternative for NAS, aiming to enhance search diversity and reduce resource consumption while maintaining or improving architectural performance.

1.2 Advancing Neural Architecture Search through Quantum-Inspired Evolutionary Algorithm.

Automated design algorithms have become a practical way to address these issues. Evolutionary algorithms represent one effective approach to designing efficient CNN architectures[16]. Evolutionary algorithms (EAs) are search and optimization methods inspired by natural selection. These algorithms search the solution space and progressively improve candidate solutions through guided evaluation and selection processes. While specific adjustments can enhance their performance, EAs are flexible, effective, and can be used for many problems without needing major changes [17].

An innovative optimization technique, Quantum-Inspired Evolutionary Algorithm (QIEA), integrates evolutionary algorithms with the concept of quantum computing. QIEA is a population-based algorithm that explores the problem space using the basic principles of quantum bits (qubits) and superposition of states [17] [18]. In quantum computing, a qubit serves as the basic unit of information. Unlike classical bits that exist in binary states (0 or 1), qubits can exist in a superposition of states, enabling parallel exploration of multiple configurations and promoting diversity in population initialization [19].

In this thesis, CNN architectures are represented using strings of qubits. Due to the principle of superposition, each qubit can represent multiple states simultaneously,

allowing the algorithm to explore a wide range of architectural configurations in parallel. During the evolutionary process, quantum rotation gates are applied to update the probability amplitudes of the qubits based on the fitness of the corresponding solutions. This mechanism enhances population diversity, mitigates the risk of premature convergence, and accelerates the convergence toward high-performing architectures. The QIEA framework thus provides an efficient means of navigating the complex architectural search space and offers a scalable alternative to the computationally intensive procedures associated with traditional Neural Architecture Search (NAS) methods.

This thesis aims to implement a Quantum-Inspired Evolutionary Algorithm (QIEA) for Neural Architecture Search (NAS) and to evaluate its effectiveness in optimizing Convolutional Neural Network (CNN) architectures for image classification tasks. To assess its performance, the architectures generated by QIEA are compared with standard CNN models having a similar number of layers, using multiple benchmark datasets.

2 Literature Review

The purpose of this review is to critically analyse the integration of Quantum-Inspired Evolutionary Algorithms into Neural Architecture Search, a field aimed at automating the design of high-performance neural networks. While traditional NAS methods such as reinforcement learning, evolutionary algorithms, have advanced automated architecture design, they face challenges like computational inefficiency, premature convergence, and limited search space exploration. QIEA, which leverages quantum computing principles to enhance evolutionary optimization, offers a promising solution. This review evaluates existing NAS frameworks, identifies gaps in scalability and theoretical foundations, and positions this thesis as a contribution to bridging these gaps through practical implementation and benchmarking.

2.1 Neural Architecture Search (NAS)

As noted by Elsken et al., Neural Architecture Search (NAS) is the process of automating the design of neural network architectures and is considered a subfield of AutoML [12]. NAS automatically discovers the best-performing CNN architectures by searching through a large-scale search space. NAS methods are data-driven rather than experience-driven, reducing the human intervention [10]. NAS has already outperformed manually designed architectures in tasks such as image classification, object detection, and semantic segmentation. As described by Elsken et al. [12], the NAS workflow is divided into three main steps:

1. Sampling architectures from the search space using specific search strategies.
2. Evaluating the performance of the sampled architectures for a given task.
3. Updating the hyperparameters based on the evaluation results to refine the search process.

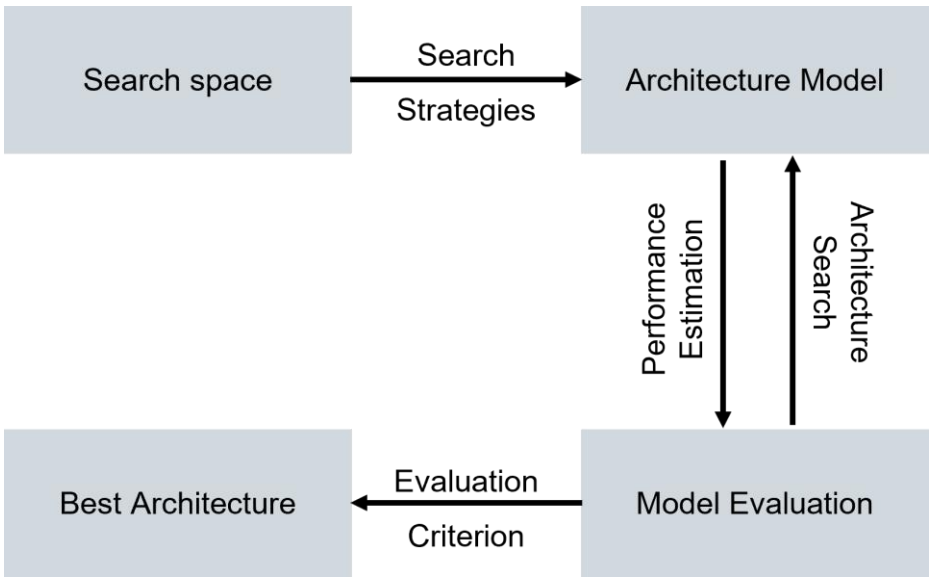


Figure 2:Principle of Neural Architecture Search [20].

The four key approaches to Neural Architecture Search namely, reinforcement learning–based methods, evolutionary algorithms, gradient-based optimization, and random search are summarized below.

2.1.1 Gradient-Based NAS

The differentiable architecture search method is a promising approach that uses gradient optimization to address the challenges of NAS [21]. Differentiable architecture search directly optimizes the architecture using a gradient-based approach. By treating architectural parameters as differentiable variables, this method allows for the use of gradient descent, enabling efficient exploration of neural network architectures. Additionally, this method significantly reduces computational costs compared to traditional approaches like reinforcement learning or evolutionary algorithms. For instance, Differentiable Architecture Search (DARTS) requires only about 4 GPU days, whereas traditional methods often demand thousands of GPU days [21]. The use of continuous relaxation further transforms discrete search spaces into continuous ones, making these methods scalable and efficient for large datasets and complex architectures [21].

2.1.2 Reinforcement Learning (RL) for NAS

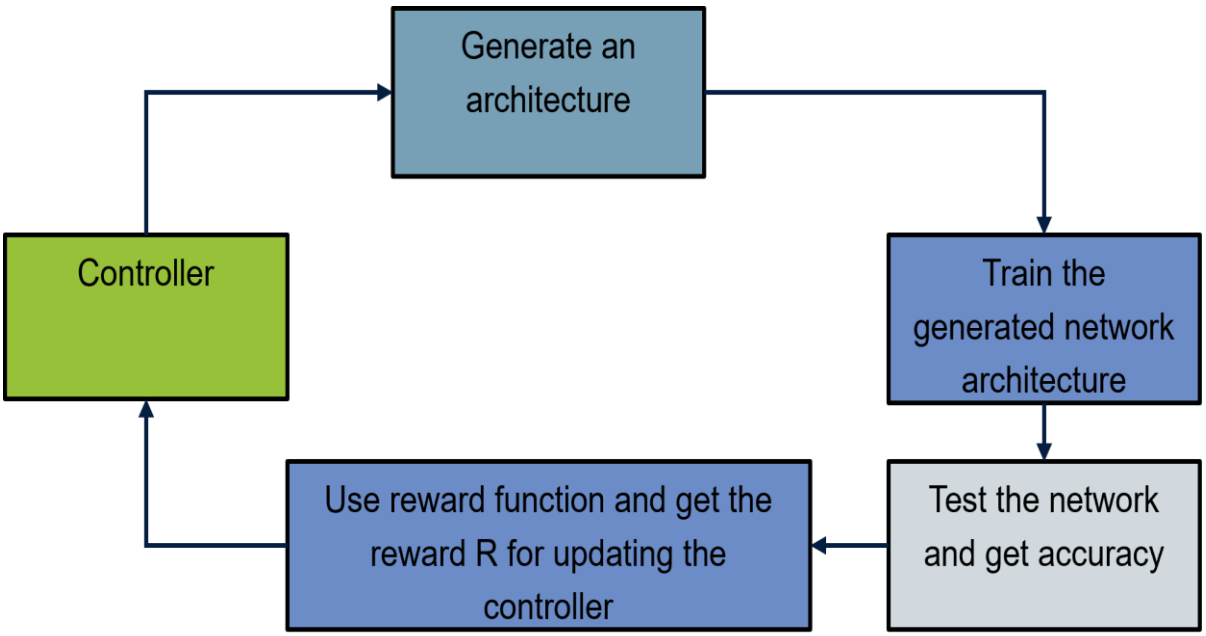


Figure 3:NAS implementation using Reinforcement Learning [22].

In RL-based NAS, a controller, typically a Recurrent Neural Network (RNN), generates candidate architectures by sampling from the search space. These architectures are trained, and their performance, such as validation accuracy, is used as a reward to update the controller through policy gradients. This iterative process helps the controller learn to propose better architectures over time [23]. While RL-based NAS is effective at exploring large and complex search spaces, it is computationally expensive, often requiring substantial GPU resources and time [22]. The effectiveness of RL also depends heavily on the design of the reward function, which must balance multiple

objectives, such as accuracy and efficiency, to guide the search effectively [22]. Despite its challenges, RL has been instrumental in advancing NAS, with methods like MetaQNN and NASNet demonstrating significant progress in the field.

2.1.3 Random Search for NAS

RandomNAS is a simple yet competitive method for Neural Architecture Search (NAS) that relies on randomly sampling architectures from a search space [24] [25]. The NAS process typically operates in two distinct phases:

1. In Training Phase, sampled architectures share weights within a supernet. These shared weights are updated iteratively during training to represent the performance of various candidate architectures efficiently.
2. In Search Phase, architectures are evaluated and ranked based on their performance within the weight-sharing framework. The top-performing architectures are then retrained independently to assess their actual performance without shared weights.

RandomNAS is more memory-efficient and can generate multiple architectures [25]. Enhancements like early stopping and weight sharing reduce computational costs and make RandomNAS surprisingly competitive with advanced NAS methods [24]. However, it faces challenges, such as difficulty accurately ranking top-performing architectures and a bias toward smaller networks that converge faster, which affects the ranking of larger, high-performing models [25].

While these methods have showed significant progress in Neural Architecture Search, they also come with limitations, such as high computational costs and limited scalability in complex search spaces. To address these challenges, Evolutionary Algorithm (EA) based NAS emerges as a promising approach. Inspired from the process of natural evolution, EAs offer a population-based optimization strategy that explores diverse architectural solutions, making them highly effective for NAS problems.

2.1.4 Evolutionary Algorithm for NAS

The application of Evolutionary Algorithms (EAs) in Neural Architecture Search (NAS) has been investigated as a population-based optimization approach inspired by the principles of natural evolution [26]. These population-based techniques, such as Genetic Algorithms (GAs) [27], Genetic Programming (GP) [28], and Particle Swarm Optimization (PSO) [29], solve complex optimization problems by simulating processes like mutation and crossover [30].

The process, as described by Liu et al. begins with initialization, where the initial population is created, with each individual represented as a "candidate solution" defined by parameters such as the number of layers or filter sizes [20]. Next is the fitness evaluation, where each candidate is assessed based on a fitness function, such as accuracy or efficiency, to determine its performance. The best-performing candidates, known as parents, are then selected based on their fitness scores. Through crossover,

parts of two parent candidates are combined to create "offspring," inheriting traits from both parents to generate new solutions. Mutation is applied to introduce random changes, such as altering parameters, ensuring diversity and avoiding premature convergence to suboptimal solutions. The offspring are added to the population, and this cycle of selection, crossover, and mutation repeats over multiple iterations. The process concludes when a stopping criterion is met, such as achieving satisfactory performance, completing a set number of iterations, or observing no further improvement.

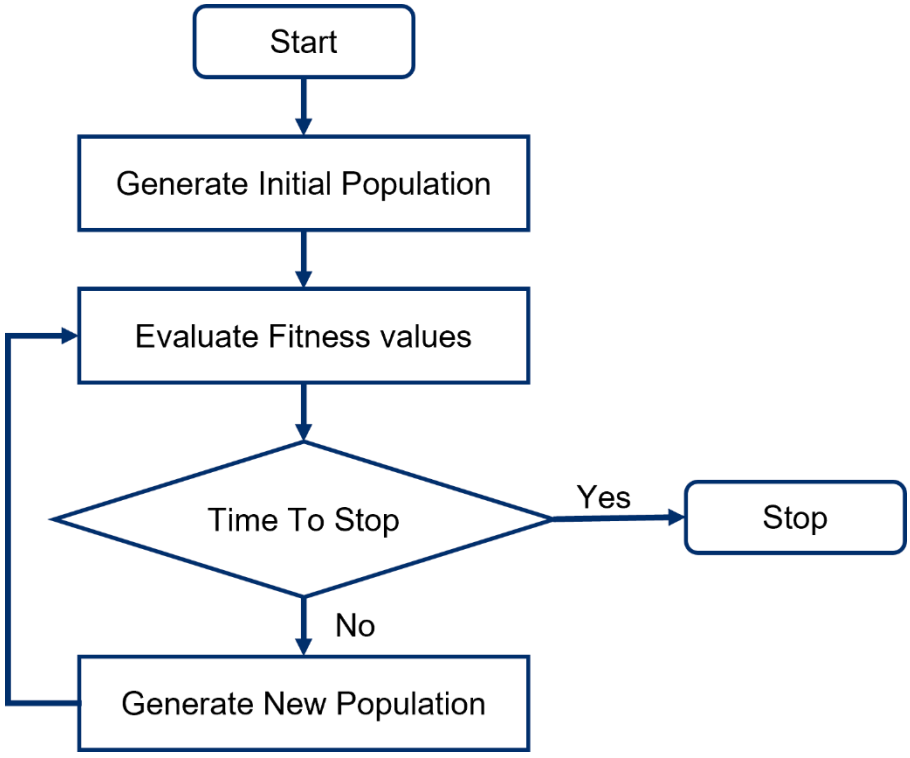


Figure 4:Flowchart of Evolutionary algorithm [31].

While EA-based NAS has been successfully applied in various domains, it also comes with significant computational challenges**.** Studies by Liu et al. have highlighted that EA-based NAS is computationally expensive, particularly for large search spaces or complex objectives [20]. Additionally, hyperparameter sensitivity is a critical factor, where improper tuning of parameters such as mutation probability or selection strategies can lead to suboptimal performance. Another limitation noted in recent research is limited exploration, where premature convergence may result in missing better-performing architectures. Furthermore, interpretability issues arise due to the complex nature of architectures generated through EA-based NAS, making it difficult to analyze and understand the evolution process [20].

To improve the efficiency of NAS, alternative methods such as Random Search have been explored. Research by Liu et al [32] analysed the disadvantages of previous NAS methods and found that in some cases, random search achieved comparable or even superior performance to EA-based NAS. However, random search still struggles with

large-scale search spaces, where the number of possible architectures grows exponentially. This has led to the exploration of Quantum Computing (QC) as a potential solution. QC leverages quantum parallelism and superposition, allowing the exploration of multiple architectures simultaneously. To address the computational limitations of classical NAS methods, two quantum-inspired paradigms have emerged: the Quantum Evolutionary Algorithm (QEA) and the Quantum-Inspired Evolutionary Algorithm (QIEA).

Quantum Evolutionary Algorithm (QEA). QEAs extend classical evolutionary search by adopting fundamental principles of quantum computing namely qubits, superposition, and entanglement as their core representation and variation mechanisms. Rather than employing binary or numerical encodings, each individual is modelled as a register of qubits, which probabilistically embodies a linear superposition of all possible binary solutions within the search space. Consequently, a single qubit-based individual can represent an exponentially large population of candidate solutions. Classical evolutionary components such as individual representation, fitness evaluation, and population dynamics are retained, but genetic operators are implemented via quantum gates that gradually collapse the superposed qubit register into a single high-fitness solution through successive measurement and rotation operations [33]. Despite its theoretical appeal, QEA's reliance on genuine quantum hardware poses significant practical challenges. Presently available quantum processors remain limited in qubit count, coherence time, and error rates, and the overhead of error correction further constrains large-scale implementation. As a result, fault-tolerant quantum execution of QEA is not yet feasible for Neural Architecture Search, motivating the development of quantum-inspired classical approximations such as QIEA that capture the representational advantages of qubits without requiring real quantum devices.

In contrast to QEA's dependence on physical quantum hardware, QIEA reproduces key quantum behaviors within a classical computing framework. Each candidate CNN architecture is encoded as a "quantum chromosome," with each qubit characterized by probability amplitudes α and β. Rather than executing real quantum gates, QIEA applies simulated quantum rotation gates to these amplitudes, guided by the relative fitness of previously observed solutions. During the observation phase, each qubit probabilistically collapses into a binary value, and the resulting bitstring decodes into a concrete CNN architecture for training and evaluation. This probabilistic representation maintains population diversity similarly to qubit superposition thereby mitigating premature convergence and enabling the algorithm to survey a vast architectural search space with far fewer explicit evaluations. By combining the exploratory power of quantum-inspired encodings with the reliability of classical execution, QIEA has been effectively applied to large-scale optimization problems like Neural Architecture Search, where it delivers high-performing architectures with substantially lower computational overhead than reinforcement-learning or traditional evolutionary methods.

2.2 Existing studies on QIEA for NAS.

The core idea behind QIEA is to utilize quantum bits (qubits) and quantum operators to guide the evolutionary search process. In contrast to classical evolutionary algorithms that operate on fixed binary representations, QIEA allows neural architectures to exist in a probabilistic superposition, enabling more efficient exploration of the search space.

Early contributions include the Quantum-Inspired Genetic Algorithm (QIGA) which encodes CNN architectures as sequences of qubits (quantum chromosomes) and employs quantum rotation gates within a genetic algorithm framework [34]. Empirical results demonstrate that QIGA can achieve comparable or better performance than traditional EA-based NAS methods while markedly reducing search time.

Building on this, quantum neural predictors have been proposed to further accelerate the evaluation phase of NAS[35] . These predictors such as quantum support vector machines, estimate the fitness of candidate architectures without full training, thereby decreasing computational overhead.

Comparative studies indicate that QIEAs can outperform conventional EA-based NAS in two key respects[36].

1. Convergence Speed: Quantum rotation updates guide the population toward high-fitness regions more rapidly than mutation and crossover alone.
2. Population Diversity: The inherent probabilistic encoding reduces premature convergence by preserving multiple candidate solutions in superposition.

Moreover, Unlike reinforcement learning-based NAS, which requires extensive training, QIEA reduces computational overhead while achieving competitive performance [22].

Despite its potential, QIEA for NAS still faces several open challenges:

1. Current QIEA implementations operate entirely within classical computing environments, relying on simulated quantum behaviour. This limits their scalability and prevents exploitation of genuine quantum advantages such as entanglement and true parallelism.
2. The performance of QIEA heavily depends on tuning parameters such as rotation angles, population size, and initialization ranges. Poor tuning can lead to suboptimal results or instability during training [34].
3. CNN architectures generated through QIEA can be difficult to interpret and analyse, posing challenges for understanding model behaviour and ensuring robustness, especially in safety-critical applications. [34].
4. Most QIEA-based NAS experiments have been conducted on small to medium-scale datasets such as CIFAR-10 or MNIST. Its effectiveness on large-scale datasets remains underexplored.

Table 1: Comparison of NAS approaches and their CIFAR-10 test accuracies [25].

Approach	Methods	Accuracy
RL-based NAS	ENAS, SETN	ENAS: 95.8
		SETN: 96.4
Random Search NAS	RSPS	96.8
EA-based NAS	EPS	97.9
Gradient-based NAS	DARTS-V1, DARTS-V2, GDAS	DARTS-V1: 97.2
		DARTS-V2: 97.5
		GDAS: 96.8

While existing studies demonstrate the promise of Quantum-Inspired Evolutionary Algorithms for improving search efficiency and reducing computational cost in NAS, there remains a lack of comprehensive evaluation against established methods across diverse benchmarks. In particular, systematic comparisons of QIEA-generated architectures with standard CNN models of equivalent complexity are scarce, and its scalability to larger datasets and complex tasks has not been fully explored. Therefore, a dedicated investigation into the performance of QIEA is essential to validate its practical advantages.

3 Fundamentals of QIEA

3.1 Overview of Convolutional Neural Networks (CNNs)

Artificial Intelligence (AI) is the umbrella discipline encompassing techniques that enable machines to perform tasks such as learning, reasoning, and decision-making. Machine Learning (ML) is a subset of AI that focuses on enabling systems to learn patterns and improve performance on tasks using data without being explicitly programmed. Deep Learning (DL), a subfield of ML, employs multilayer neural network architectures to automatically extract hierarchical features from complex data. DL has enabled breakthroughs in tasks like image recognition, natural language processing, and autonomous systems. [37]

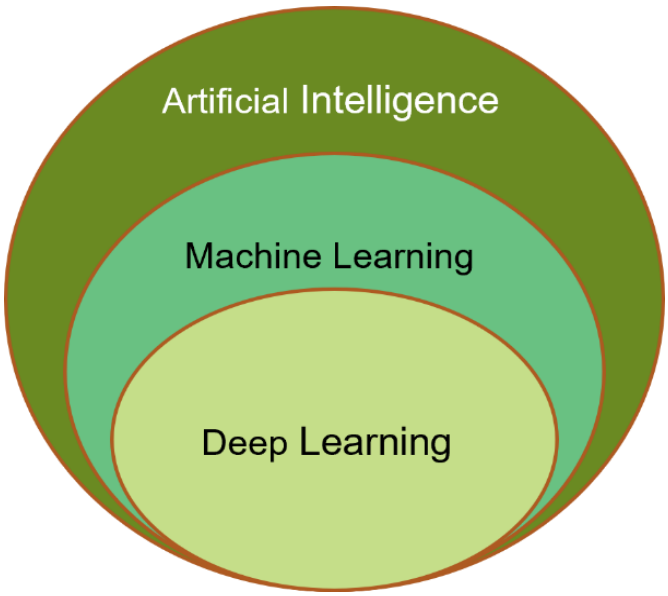


Figure 5: The taxonomy of AI, ML and DL [38].

Convolutional Neural Networks (CNNs) are a specialized type of supervised deep learning model designed to process data with a grid-like structure, such as images. They have become a fundamental component in the field of computer vision due to their ability to automatically and efficiently learn the features from input data. For instance, LeNet-5 and AlexNet are well known early CNNs. Figure 1 and Figure 5 shows the architectures of LeNet-5 and AlexNet respectively.

CNNs consist of following key components that enable their functionality.

1. Convolutional Layers: These layers apply filters (or kernels) to the input data, performing a mathematical operation called convolution. A filter is a small n × n matrix consisting of numbers that slides over the input image in the first layer, producing an output called a feature map [39]. During the convolution process, an element-wise multiplication occurs between the filter and the corresponding region of the input image, followed by summing the results to generate the output. After the convolution operation, an activation function is typically applied to

the feature map [40]. It introduces non-linearity to the model and enabling it to learn complex patterns. This combination of convolution and activation helps extract features like edges, textures, and patterns from the input image, with each filter detecting specific characteristics.

2. Pooling Layers: Pooling layers reduce the dimensions of the feature maps generated by the convolutional layers [41]. These layers use filters that slide over the feature map. Operations like max pooling select the maximum value in a region defined by the filter size, while average pooling calculates the average value in that region [39]. Pooling reduces spatial dimensionality and computational cost while retaining dominant activations that characterize the input [40].
3. Fully Connected Layers: The feature maps from the last convolutional layer are flattened into a series of numbers, which are then fed into the FC Layer as input. The FC layer is a neural network with multiple layers, where every neuron is connected to all the neurons in the next layer, forming a fully connected structure [40]. This layer combines the extracted features to make the final prediction. Each neuron in the FC layer has associated weights, which are adjusted during the training process using backpropagation. Backpropagation, a fundamental concept in neural networks, calculates the error between the predicted and actual outputs and propagates this error backward through the network to optimize the weights [42] [12]. After training, these weights are fine-tuned to enable accurate classification or prediction tasks. The final FC also known as output layer is commonly used for the final decision-making step in tasks like classification or regression and generally it uses softmax as activation function [41].
4. Activation Functions: Activation functions such as tanh, ReLU, sigmoid, and softmax at output are essential components in neural networks that introduce non-linearity, allowing the model to learn complex patterns and relationships in data [39]. They are applied after each neuron combines its inputs with weights to produce a result. They transform this result to decide if the neuron should activate and pass information to the next layer. Without activation functions, the network would behave like a simple linear model, regardless of the number of layers, and would not be able to handle complex tasks or approximate intricate functions. For classification tasks, the softmax function is commonly used in the final layer. It converts the outputs into probabilities, assigning a value between 0 and 1 to each class, with the total sum of probabilities equal to 1. This makes it essential for interpreting the output as class probabilities in multi-class classification tasks [41].

CNNs rely on several key hyperparameters that influence their performance:

5. Kernel Size: The kernel size defines the dimensions of the filter used in convolutional layers. It determines how much of the input image is analysed at a time. A larger kernel size can capture broader and more global features; however, it

requires more computations and may overlook fine details. On the other hand, a smaller kernel size captures finer and more localized details, but it may fail to capture larger patterns. The choice of kernel size depends on the specific task and the complexity of the input data [43].

6. Stride: Stride refers to the step size the filter moves across the input image during the convolution operation [43]. A smaller stride, such as 1, moves the filter one pixel or column at a time, creating a detailed and high-resolution output feature map, but at the cost of increased computational load. Larger stride values reduce computational cost but may omit fine-grained details from the feature maps [40]. Proper stride selection is crucial for balancing output detail and computational efficiency.
7. Padding: Padding adds extra pixels, usually zeros, around the edges of the input image before convolution [40]. This ensures that the filter can process edge regions and maintain the spatial dimensions of the output [44]. Without padding, the output size reduces with each convolution, which may cause the loss of important edge details.
8. Pooling Operations: Pooling reduces the dimensions of feature maps while retaining the most important information [41]. Max pooling preserves the strongest activations, thereby highlighting prominent patterns, while average pooling computes mean values to yield a smoother, more generalized feature representation. Both pooling methods help reduce computational complexity, minimize overfitting [40].

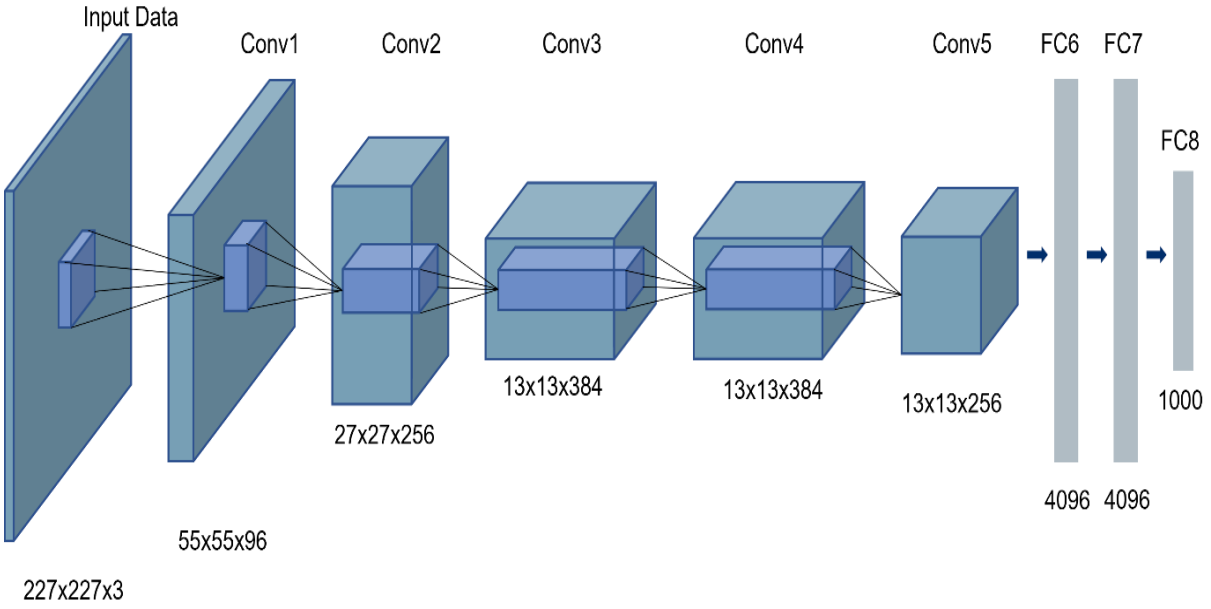


Figure 6: Architecture of AlexNet [39].

In CNNs, convolutional layers extract hierarchical features, pooling layers perform spatial dimensionality reduction, and fully connected layers generate the final

classification outputs. The selection of parameters, such as the number of layers, filter sizes, and activation functions, significantly impacts the performance of a CNN. Choosing the optimal configuration is a crucial and challenging process, as it requires extensive experimentation and expertise. This manual tuning procedure is not only time-consuming but also resource-intensive. Therefore, automating this process is essential for CNN design, improve efficiency, and reduce the dependency on expert knowledge.

To overcome this challenge, Neural Architecture Search (NAS) has been introduced as an automated approach to optimize CNN designs. NAS explores various architectural configurations to find the most efficient network structures for a given task. Traditional NAS methods, including Reinforcement Learning based NAS, Evolutionary Algorithm based NAS, and Gradient-based NAS, Random search based NAS, are discussed in detail in Section 2.1 of the Literature Review. While NAS significantly reduces manual effort, it still suffers from high computational costs, especially when applied to large search spaces.

Recent advancements in Quantum Computing (QC) and Quantum-Inspired Evolutionary Algorithms offer promising solutions to enhance NAS efficiency. Quantum principles, such as superposition and entanglement, enable faster exploration of the search space, improving both speed and optimization quality. In the next section, we introduce the fundamentals of Quantum Computing and explore how quantum principles can be leveraged to improve NAS methodologies.

3.2 Fundamentals of Quantum Computing

Quantum computing applies the principles of quantum mechanics to information processing, thereby enabling fundamentally different computational paradigms from those of classical machines. Unlike classical bits, which represent either a 0 or a 1, quantum bits, or also called as qubits, can exist in multiple states simultaneously and the phenomenon known as superposition [45][46][47]. This allows quantum computers to perform certain types of calculations more efficiently than classical computers. Another key principle in quantum computing is entanglement, where qubits become interconnected such that the state of one qubit directly influences the state of another, irrespective of their spatial separation [48]. This property is essential for certain quantum algorithms and for potential applications in secure communication. In practice, qubits may be realized using physical systems such as the spin of an electron in a semiconductor quantum dot, the energy levels of trapped ions, or the collective states of superconducting circuits, each of which must be carefully isolated and controlled to preserve quantum coherence [49].

Mathematically, a qubit's state can be expressed as:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad (1)$$

Here, α and β are complex probability amplitudes that determine the likelihood of the qubit being measured in state $|0\rangle$ or $|1\rangle$, respectively. The probabilities are given by $|\alpha|^2$ for state $|0\rangle$ and $|\beta|^2$ for state $|1\rangle$, with the constraint that

$$|\alpha|^2 + |\beta|^2 = 1 \quad (2)$$

This ensures that the total probability sums to one. The Bloch sphere is commonly employed to visualize a single qubit's state. In this model, a qubit's state is depicted as a point on the surface of a unit sphere. The north and south poles of the sphere correspond to the basis states $|0\rangle$ and $|1\rangle$, respectively. Any other point on the sphere represents a superposition of these basis states, characterized by two angles:

- The polar angle, θ - which defines the probability distribution between $|0\rangle$and $|1\rangle$.
- The azimuthal angle, φ - which determines the relative phase between the basis states.

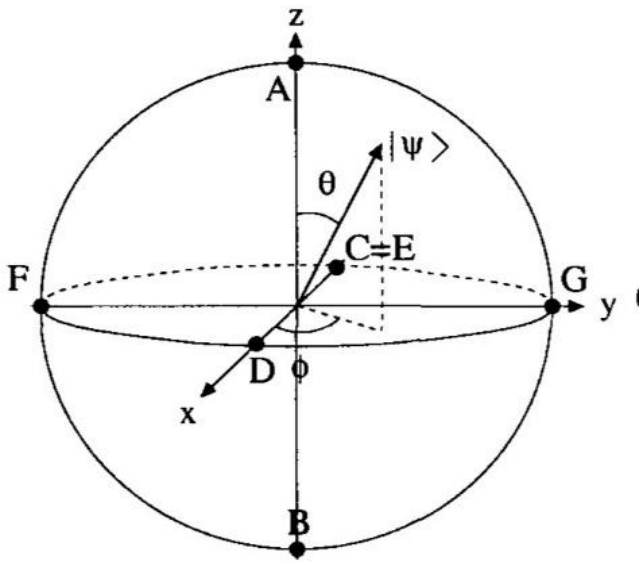


Figure 7: Bloch sphere [47].

This geometric representation provides an intuitive way to understand qubit states and their transformations.

Quantum gates serve as the fundamental operations for manipulating qubit states and constitute the building blocks of quantum circuits, analogous to the role of logic gates in classical circuits. Examples of quantum gates include the Hadamard gate (which generates superposition), the controlled-NOT (CNOT) gate (which produces entanglement), the Pauli-X/Y/Z gates, the phase (S and T) gates, the Toffoli (CCNOT) gate, and the SWAP gate. By arranging these gates in various configurations, complex quantum operations can be performed [46].

Practical realization of quantum computers remains challenging because qubits are exceptionally sensitive to environmental disturbances, necessitating precise control and isolation. Errors due to decoherence and other quantum noise must be corrected using quantum error correction techniques, which often involve encoding a single logical qubit into multiple physical qubits to protect information [50].

Despite these obstacles, continued research strives to overcome technical barriers and harness the full potential of quantum computing. As the technology matures, it holds the promise of revolutionizing fields such as cryptography, materials science, and complex system simulations.

The ability of a qubit to exist in superposition, simultaneously representing both $|0\rangle$ and $|1\rangle$ allows QIEA to encode many CNN configurations in a single quantum chromosome. When measured, these superposed states collapse into different binary strings, enabling the algorithm to explore a wide variety of architectures in parallel. This intrinsic multiplicity reduces the number of explicit evaluations needed and preserves structural diversity, thereby mitigating premature convergence and improving the likelihood of discovering high-performance networks.

3.3 Quantum Inspired Evolutionary Algorithms (QIEAs)

Two primary paradigms integrate quantum computing concepts with evolutionary algorithms: Quantum-Inspired Evolutionary Algorithms (QIEAs) and Quantum Evolutionary Algorithms (QEAs). QIEAs simulate quantum principles such as superposition and quantum gates on classical computing hardware. By contrast, QEAs are implemented directly on quantum hardware, leveraging genuine quantum phenomena [51]. QIEAs are optimization techniques that integrate principles from quantum computing into classical evolutionary algorithms. By leveraging concepts such as qubits, superposition, and quantum gates, QIEAs aim to enhance the exploration and exploitation capabilities of traditional evolutionary methods. This integration allows for a probabilistic representation of solutions, potentially leading to improved diversity and global search efficiency.

A notable example is the Quantum-Inspired Genetic Algorithm (QIGA), which utilizes quantum bits (qubits) to represent solutions [36]. This representation enables the algorithm to maintain a superposition of states, facilitating a more comprehensive search of the solution space. Operations analogous to quantum gates are applied to evolve the population, guiding the search towards optimal solutions. QIEAs have been applied to various complex optimization problems. For instance, the Quantum-Inspired Acromyrmex Evolutionary Algorithm (QIAEA) has been proposed as an efficient global optimization method for complex systems [52]. Additionally, QIEAs have been employed in feature subset selection tasks, demonstrating their versatility and effectiveness in handling high-dimensional data.

QIEAs, despite their quantum-inspired design, operate entirely on classical hardware by simulating quantum phenomena such as superposition and quantum gates.

Consequently, they do not require access to quantum processors yet still benefit from enhanced exploration and convergence characteristics. This capability permits researchers to investigate quantum-inspired optimization strategies without relying on actual quantum computing platforms.

In summary, Quantum-Inspired Evolutionary Algorithms represent a promising fusion of quantum computing concepts with classical evolutionary strategies, offering enhanced capabilities for solving complex optimization problems

4 Implementing Quantum-Inspired Evolutionary Algorithm for NAS

The Quantum-Inspired Evolutionary Algorithm for Neural Architecture Search leverages quantum principles to enhance the efficiency of traditional Evolutionary Algorithms in optimizing Convolutional Neural Networks. Unlike conventional NAS methods that rely on exhaustive searches or reinforcement learning, QIEA introduces a quantum-inspired representation of CNN architectures, allowing for a more efficient exploration of the search space.

The foundational methodology for this work builds upon the Quantum-Inspired Evolutionary Algorithm for CNN architecture search introduced by Ye et al. [34]. The research discussed the approach to encode CNN architectures as quantum chromosomes, where each quantum bit (qubit) represents a component of the CNN, such as the number of layers, filter sizes and pooling operations. The quantum state of each qubit allows for multiple architectural possibilities to coexist simultaneously, enabling a broader and more diverse search.

To optimize CNN architectures, QIEA follows an iterative evolutionary process, described in Algorithm 1 below:

Algorithm 1 Framework of QIEA

```
Initialize the population, Q with the proposed encoding strategy
q is individual chromosome in Q
while termination criterion is not satisfied do
    for q in Q do
            observe and evaluate q
        If q has best accuracy
            q_best = q
        update q with rotation U
    end for
end while
```

To optimize CNN architectures, QIEA follows an iterative evolutionary process:

1. Quantum Encoding of CNN Architectures: Each CNN candidate is represented using quantum probability amplitudes, allowing it to exist in multiple states before being observed.

2. Quantum Observation & Evaluation: The quantum states collapse into a specific CNN architecture, which is then evaluated based on its performance, for example, classification accuracy.
3. Evolutionary Update via Quantum Operators: Evolutionary operations, such as mutation, selection, and crossover, are applied using quantum rotation gates to update and refine the population of architectures.
4. Selection of Optimal Architecture: The highest-performing CNN architectures are retained, and the best one is selected and retrained, leading to an optimized neural network model.

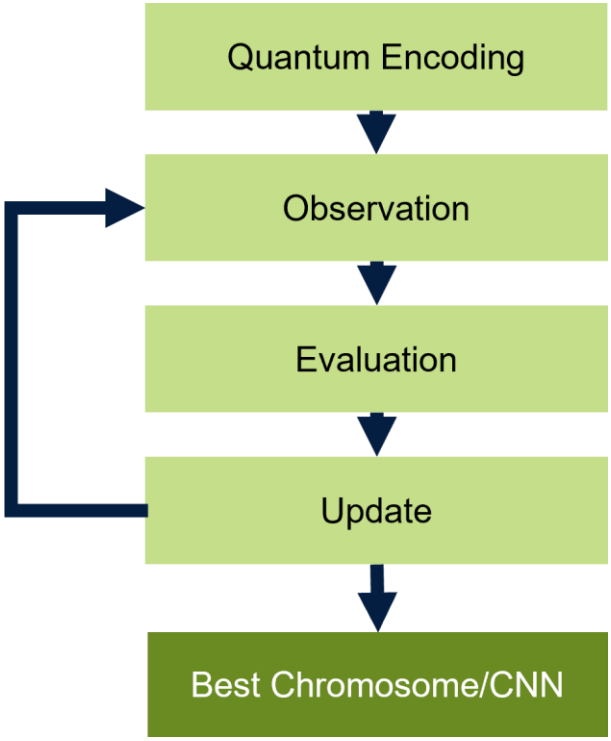


Figure 8: An iterative evolutionary process for single chromosome.

Figure 7 presents the high-level workflow of the Quantum-Inspired Evolutionary Algorithm (QIEA) for Neural Architecture Search. The process commences with Quantum Encoding, in which each candidate CNN architecture is represented as a quantum chromosome comprising qubits, each characterized by probability amplitudes α and β. During the Observation phase, these qubit states collapse into a definitive binary string according to the measurement rule, producing a concrete CNN configuration. For improved interpretability and layer-type identification, this binary string is then converted into a fixed-length dot–decimal notation. In the Evaluation step, the CNN decoded from the dot–decimal representation is briefly trained and its performance is recorded as the fitness score. The Update stage applies quantum rotation gates to modify the qubit amplitudes of each chromosome, with rotation angles

determined by the performance gap relative to the current best solution. This cycle of Observation, Evaluation, Update repeats until a predefined termination criterion is satisfied, at which point the algorithm returns the Best Chromosome, corresponding to the highest-fitness architecture.

By integrating quantum-inspired evolution, this approach reduces search complexity, enhances diversity in architectural exploration, and improves computational efficiency compared to conventional NAS methods. The next sections describe in detail the encoding strategies, evolutionary update mechanisms, and implementation details of QIEA for NAS.

4.1 Encoding Strategy for CNN Architecture

The encoding strategy in the Quantum-Inspired Evolutionary Algorithm (QIEA) plays a crucial role in representing Convolutional Neural Network (CNN) architectures as quantum chromosomes. Unlike traditional NAS methods that use fixed binary or numerical encoding, QIEA leverages qubits, allowing each CNN component to exist in superposition, enabling a broader and more diverse search space.

In traditional Neural Architecture Search (NAS), CNN architectures are typically encoded using binary strings, where each architecture is explicitly represented as a unique combination of bits. Each layer type, filter size, and other parameters are encoded as separate values. For example, 000 represent a layer with 3x3 filter, 32 feature maps and 001 represents another layer with 3x3 filter, 64 feature maps. Each binary string corresponds to one specific CNN architecture, meaning that if we want to evaluate multiple architectures, we have to individually test each one. This makes the search slow and computationally expensive because a large number of architectures must be explicitly represented and evaluated.

In contrast to classical binary encoding, which requires each CNN design to be represented and evaluated individually qubit-based encoding enables a single quantum chromosome to encapsulate a superposition of multiple architectural configurations. By parameterizing each qubit with probability amplitudes α and β, QIEA can sample diverse candidate architectures from a compact representation, thereby reducing the combinatorial explosion associated with exhaustive binary enumeration and accelerating the search process. The probabilistic nature of qubit encoding also helps maintain population diversity and mitigate premature convergence.

4.1.1 Quantum-Inspired Encoding in QIEA

However, in Quantum-Inspired Evolutionary Algorithm, it introduces a probabilistic quantum representation, where a single quantum chromosome can encode multiple CNN architectures at once.

This is done using qubits, which can exist in a superposition of states. Instead of just being 0 or 1, a qubit can be in both states simultaneously, meaning a single qubit can

hold more than one configuration at a time. Following is the representation of n number of Qubits.

$$\begin{pmatrix} \alpha 1 & \alpha 2 & \alpha 3 \ldots \alpha n \\ \beta 1 & \beta 2 & \beta 3 \ldots \beta n \end{pmatrix} \quad (3)$$

With, α is the probability of being 0

β is the probability of being 1

n is the number

For example, in a three-qubit system, the state of the quantum chromosome can be represented as:

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & 1 & \frac{1}{2} \\ \frac{1}{\sqrt{2}} & 0 & \frac{\sqrt{3}}{2} \end{pmatrix} \quad (4)$$

Here are the 4 states of the system represented as

$$\frac{1}{2\sqrt{2}}|000\rangle + \frac{\sqrt{3}}{2\sqrt{2}}|001\rangle + \frac{1}{2\sqrt{2}}|100\rangle + \frac{\sqrt{3}}{2\sqrt{2}}|101\rangle \quad (5)$$

In this case, the probabilities of measuring the system in the states $|000\rangle, |001\rangle, |100\rangle$ and $|101\rangle$ are $\frac{1}{8}$, $\frac{3}{8}$, $\frac{1}{8}$ and $\frac{3}{8}$ respectively. This means that the quantum system encodes information across four states at the same time, rather than requiring separate representations for each state. This is a key advantage of quantum encoding over classical binary encoding. In a classical representation, at least four separate binary strings (chromosomes) would be needed to represent these four states individually. However, a quantum chromosome with three qubits can represent all 2^3=8 states in superposition, each with an associated probability amplitude. This property of superposition enhances the diversity of the search space in QIEA, providing a more efficient representation compared to classical approaches. α and β are probability amplitudes.

QIEA maintains a population of quantum chromosomes, denoted as Q(t), at generation t. This population consists of n quantum chromosomes, represented as

$$Q(t) = \{q_t^1, q_t^2, \ldots, q_t^n\} \quad (6)$$

Each chromosome q_t^j is defined as a matrix that encodes quantum states

$$q_t^j = \begin{bmatrix} \alpha_{j1}^t & \alpha_{j2}^t \ldots . \alpha_{jm}^t \\ \beta_{j1}^t & \beta_{j1}^t \ldots . \beta_{jm}^t \end{bmatrix} \quad (7)$$

With, m is the string length of the chromosome.

The quantum chromosome is observed and collapses into a binary string, which represents the CNN architecture. The details of this binary representation and its role in defining the CNN structure are discussed in the next section.

4.1.2 Binary Representation of CNN Architectures

The CNN architecture consists of three primary types of layers:

- Convolutional Layers – responsible for feature extraction.
- Pooling Layers – used for dimensionality reduction.
- Fully Connected Layers – perform the final classification.

A Disabled Layer is included to facilitate variable-depth architectures by permitting the deactivation of specific layers. This ensures that architectures of varying depths can be efficiently represented.

In the configuration of each type of CNN layer, different attributes define their characteristics, with each attribute having a specific integer range. These values can be converted into binary strings for efficient representation. For the Convolutional Layer, the key parameters include filter size, number of feature maps, and stride size. Similarly, the Pooling Layer is defined by kernel size, stride size, and pooling type, such as max or average pooling. The Fully Connected Layer has a single key parameter, which is the number of neurons.

To represent an entire CNN architecture, the binary strings of all layers are combined into long binary sequence, called a chromosome. This encoding method allows the evolutionary algorithm to process and optimize different network architectures efficiently. Each type of layer has a different binary string length depending on the range of its attributes. For instance, in the Convolutional Layer, the total binary string length is 12 bits, with filter size ranging from 1 to 8, feature maps from 1 to 128, and stride size from 1 to 4. For example, a binary encoding of "000" for filter size corresponds to a value of 1, ensuring that the filter size is always valid and never set to zero.

From Table 1, it is observed that the Pooling Layer, Fully Connected Layer, and Disabled Layer each require 11 bits for encoding. Since the largest binary string required for a single layer is 12 bits, we can use two bytes (each containing 8 bits) to store this information. This encoding method enables the CNN architecture to be represented in a compact binary format, which can also be transformed into dot-decimal notation for easier processing and optimization within the QIEA.

The binary representations of each layer type are combined into a single binary string, known as a chromosome, which defines the entire CNN configuration. This encoding ensures that different architectures can be efficiently processed and evolved within the QIEA framework.

4.1.3 Dot-decimal Notation

To define specific layer types in the CNN architecture, we employ a fixed-length dot-decimal notation. This encoding was selected because its integer–fractional representation aligns intuitively with layer indices, providing greater human readability than alternative encodings such as hexadecimal. Each dot-decimal encodes a specific layer type along with its parameters, and a sequence of these notations collectively defines the entire CNN architecture.

Table 2: The parameters and their range of different types of CNN layers.

Layer Type	Parameters	Range	Number of Bits	Total number of bits	Example
Convolution	Filter size	[1,8]	3	12	2(001)
	Feature maps	[1,128]	7		8(000 0111)
	Stride size	[1,4]	2		4(11)
Pooling	Kernel size	[1,4]	2	11	2(01)
	Stride size	[1,4]	2		2(01)
	Type (1=max, 2=avg)	[1,2]	1		2(1)
	Place holder-un-used bits	[1,64]	6		16(00 1111)
Fully-Con-nected	Neurons	[1,2048]	11	11	256(00011 111111)
Disabled	Place holder	[1,2048]	11	11	256(00011 111111)

As shown in the Table 2, the layers have 12 and 11 bits assigned to each layer. To facilitate unified processing and simplify the encoding/decoding, each layer's binary representation is standardized to a fixed length of 16 bits. Although the raw parameter encodings require only 11 or 12 bits (depending on the layer type) padding is introduced by prepending 4 or 5 most-significant bits (MSBs) that uniquely identify the layer category. For example, convolutional layers use a 4-bit prefix (0000), while pooling, fully-connected, and disabled layers employ distinct 5-bit prefixes (00010, 00011, and 00100, respectively). These prefixes not only normalize the overall bit-string length but also embed type information directly into the binary code. Table 5 details the resulting dot-decimal ranges for each prefixed segment, illustrating how a 16-bit string maps to both a layer identifier and its parameter values.

Table 3 illustrates how individual CNN layers are encoded into binary segments and concatenated to form a chromosome. For convolutional layers, filter size, feature map count, and stride are each mapped to fixed-length binary substrings; for pooling layers, kernel size, stride, and pooling type are similarly encoded; and for fully connected layers, the neuron count is converted to a binary substring. These substrings are then concatenated to produce the full binary representation of each layer.

Table 3: Encoding of CNN architecture into a binary chromosome.

A convolutional layer			
Filter size = 3	Feature maps= 32		Stride = 2
010	0011111		01
010 0011111 01			
A pooling layer			
Kernel size = 2	Stride size =2	Max pooling	Place holder
01	01	1	000111
01 01 1 000010			
A fully connected layer			
256 neurons			
00011 111111			

The CNN structure in Binary chromosome is represented as:

010 0011111 01, 01 01 1 000010, 00011 111111

With reference from Table 4, for the Convolutional Layer, the dot-decimal notation starts at 0.0, where the most significant bit (MSB) represents an 8-bit binary string, and the least significant bit (LSB) represents another 8-bit binary string. The total length of the dot-decimal string is 16 bits, but only 12 bits are needed for encoding. To match the required length, 4 additional bits are added to the MSB position, bringing the sub-layer representation to 0.0/4, with a range from 0.0 to 15.255. Specifically, a 4-bit binary string ‘0000’ is added to the 12-bit MSB to ensure the dot-decimal notation maintains this range. For the Pooling Layer, which requires 11 bits for encoding, the corresponding sublayer representation starts at 16.0/5, covering values from 16.0 to 23.255 with an addition of ‘00010’ to the MSB. Similarly, the Fully Connected Layer is assigned the range 24.0/5, spanning from 24.0 to 31.255 by adding the string ‘00011’. This structured encoding approach ensures that different CNN layers are uniquely identified while allowing flexibility in representing variable-length architectures.

Table 4: The binary string range corresponding to each layer type.

Layer Type	Sublayer (Dot-decimal)	No of bits added to the binary string	Range (Dot-decimal)
Convolutional Layer	0.0	4	0.0 - 15.255
Pooling Layer	16.0	5	16.0 - 23.255
Fully-Connected Layer	24.0	5	24.0 - 31.255
Disabled Layer	32.0	5	32.0 - 39.255

Table 5: An Example for binary string and its Dot-decimal representation.

Layer Type	Binary Representation	Split of binary number	Dot-Decimal
Convolutional Layer	(0000)001 0001111 01	(0000)001 0 . 001111 01	2.61
Pooling Layer	(00010)01 01 0 001111	(00010)01 0 . 1 0 001111	18.143
Fully-Connected Layer	(00011)01111 111111	(00011)011 . 11 111111	27.255
Disabled Layer	(00100)01111 111111	(00100)011 . 11 111111	35.255

Table 6: An Example for Dot-decimal as a chromosome.

Convolutional (C)	Pooling (P)	Convolutional (C)	Disabled (D)	Fully-Connected (F)
2.61	18.143	2.61	35.255	27.255

Table 6 presents an example of a chromosome to demonstrate how a CNN architecture is encoded. With a layer length of 5, the quantum chromosome has a total length of 80. After initialization, this quantum chromosome collapses into a fixed binary string. In this example, the collapsed string is shown in Table 4, where each layer is represented by either a 12-bit or 11-bit binary string, depending on the layer type. To convert this into dot-decimal notation, 4 or 5 bits are added to the MSB of the binary string. Using dot-decimal values simplifies the identification of layer types and their parameters by enabling easy conversion between binary and dot-decimal representations. When decoding the binary string into a CNN, any Disabled Layer should remain inactive. This encoding strategy allows for flexible CNN architectures with a variable number of layers.

4.2 Population Initialization

The first step of the QIEA algorithm is to initialize a population of quantum chromosomes, each representing a candidate CNN architecture in qubit-string form. Based on predefined rules for CNN layer configuration, a fixed number of chromosomes is generated: each chromosome encodes layer types (convolutional, pooling, disabled, or fully connected) according to the maximum network depth L and number of fully connected layers F, and assigns qubits (α , β amplitudes) to each layer in compliance with quantum state normalization. This initialization ensures that the search begins with a diverse yet valid set of architectures, laying the foundation for subsequent evolutionary refinement.

During population initialization, several parameters need to be defined. Each individual in the quantum population is represented as a matrix, where each dimension corresponds to the α and β values of a qubit in the quantum chromosome.

To ensure that the generated CNN architectures are valid and well-structured, the initialization process follows specific rules:

- L represents the maximum number of CNN layers, and F represents the maximum number of Fully-Connected layers.
- The first layer is always a Convolutional Layer.
- From the second layer to (L–F) layers, the model can contain Convolutional, Pooling, or Disabled Layers.
- Every pooling layer must directly follow a convolutional layer.
- From (L-F) to (L-1) layers, any layer type can be selected until the first Fully-Connected Layer is added. After this, only Fully-Connected or Disabled Layers are allowed.
- The final layer is always a Fully-Connected Layer, with the number of neurons matching the number of output classes.

This approach allows the quantum chromosome to define flexible CNN architectures with a variable number of layers, ensuring both diversity and efficiency in the search process. After initializing the layers randomly, the qubits are also randomly assigned based on the layer type, as different layer types require different qubit configurations. In qubit initialization, the α(alpha) and β(beta) values are assigned such that they satisfy the normalization equation (2). This ensures that the qubit remains in a valid quantum state.

The CNN layers are initially assigned randomly based on predefined rules, and the quantum bits (qubits) are initialized according to the number of qubits required for each layer type. Once the layers are set for a particular chromosome, they remain unchanged during the training process. However, the parameters of the architecture, such as filter sizes, feature maps, and stride values, are continuously updated. This approach ensures that the algorithm optimizes within a specific architecture structure, leading to the discovery of a local minimum for that particular configuration. Multiple

such chromosomes are initialized and trained in parallel to identify the best-performing architecture. Keeping the layer types fixed throughout the entire evolutionary process ensures stability during optimization. If layer types were allowed to change such as converting a convolutional layer into a pooling layer during mutation or crossover, it could cause disruptive structural changes, making optimization unstable. Instead, by maintaining a consistent layer structure, the algorithm can fine-tune the parameters within that specific architecture, allowing it to effectively reach an optimal local minimum. By comparing multiple chromosomes based on their performance, we can identify the best-performing architecture for the given dataset, ensuring that the final selected model is optimally suited to the provided data.

4.3 Quantum observation

Algorithm 2 Quantum chromosome observation

Input:	QP (Quantum chromosome population)
	m (Length of quantum chromosome)
Output:	BCNN (Binary-encoded CNN architecture)
1	For each individual chromosome q in QP:
2	Set i = 1 (initialize index)
3	While I ≤ m:
4	If α > β
5	qi = 0
6	Else,
7	qi = 1
8	End if
9	End while loop
10	End for loop
11	Return BCNN

Prior to training, each quantum chromosome,composed of m qubits, must be converted into a binary string that defines a concrete CNN architecture. This conversion is performed by an observation rule, which compares the probability amplitudes α and β for each qubit: if$|\alpha|^2 > |\beta|^2$, the qubit collapses to 0; otherwise, it collapses to 1. The resulting binary sequence is then partitioned according to the bit-widths specified in Table 2, with each segment decoded into layer parameters (e.g., filter size, number of feature maps, stride) to construct the corresponding CNN layers. Because the initial α and β values are assigned randomly, reflecting the qubit

superposition, the effective search space of QIEA encompasses a vast variety of architectures.

Algorithm 3: QIEA-Based CNN training with Early stopping	
Step	Operation
Input:	Qin (Quantum chromosome population) epochsper_training (Max epochs per CNN training session) learning_rate (Learning rate for optimization) start_epoch_fraction (Fraction of epochs before checking early stopping) training_iterations (Max training iterations per chromosome) stop_threshold (No improvement iterations before stopping)
Output:	Optimized CNN architecture with best parameters Best chromosome configuration with trained CNN layers
1	For each chromosome q in Qin:
2	Initialize best_accuracy=0 and no-improvement counter = 0
3	For each training iteration itr in training_iterations:
4	Decode dot-decimal representation to construct CNN architecture
5	For each epoch e in epochs_per_training:
6	Train CNN using batch size s and learning rate lr
7	Compute current_accuracy
8	If current_accuracy > best_accuracy:
9	Update current_accuracy = best_accuracy
10	Store best CNN accuracy and structure
11	If epochs > (start_epoch_fraction × epochs_per_training)
12	Terminate training for this chromosome
13	Break loop and proceed to next training iteration
14	End of epochs training
15	Terminate iteration loop for this chromosome
16	Move to next chromosome in population
17	Apply Qubit Rotation and Update Chromosome
18	End of training iterations
19	Retrain the best CNN with additional epochs for final refinement
20	Return updated population Qout with trained CNN architectures

As described in Algorithm 2, the observation process is performed by comparing the probability amplitudes α and β for each qubit. If α greater than β, the observed state is set to 0; otherwise, it is set to 1. This process collapses the quantum chromosome into a fixed binary representation, which defines the structure of the CNN architecture before training begins.

4.4 Evaluation Strategy

The evaluation strategy ensures efficient training and selection of the best CNN architecture during the NAS process. Each quantum chromosome, once decoded into a CNN is trained on the training set, and its validation accuracy is recorded as the fitness score. After every training cycle, if the current validation accuracy exceeds the historical best for that chromosome, the model weights and architecture parameters are updated as the new “best” candidate. To optimize computational efficiency, an early-stopping criterion terminates training for a given chromosome when no improvement is observed over a specified number of consecutive epochs or iterations. This prevents overfitting and limits wasted computation on stagnant architectures. Once all chromosomes in the population have been evaluated in this manner, the highest-fitness architecture undergoes a final retraining phase often with extended epochs to fine-tune its parameters and maximize generalization performance. By combining rigorous fitness tracking, early termination of underperforming candidates, and a dedicated refinement stage for the top performer, this evaluation framework ensures both effective search convergence and judicious use of computational resources.

4.5 Quantum Gate-Based Update Mechanism for Qubits

After performing quantum observation and network evaluation, the next step in Quantum-Inspired Evolutionary Algorithm (QIEA) is updating the network using quantum gates. These gates help in improving the population by guiding chromosomes toward better solutions. The primary update mechanism involves applying a rotation gate U(θ) to the quantum chromosome:

$$U(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{8}$$

where θ represents the rotation angle, which determines how much the quantum state is adjusted. The direction of rotation is influenced by s (s depends on α_i, β_i) derived from Table 7. This ensures that the quantum chromosome converges toward fitter architectures over successive generations.

Unlike classical genetic algorithms that use mutation and crossover, QIEA only applies rotation gates. This is because quantum encoding already maintains probabilistic representation, making traditional mutation unnecessary. The rotation gate modifies the probability amplitudes, gradually increasing the likelihood of observing optimal CNN

architectures. Table 7 provides the rotation angles and update rules based on the current chromosome state, best chromosome, and fitness function.

The quantum rotation gate plays a crucial role in adjusting the probability of each bit (0 or 1) in the chromosome, enabling controlled evolution without the need for classical mutation. After evaluating multiple architectures, the best-performing one is identified, and its binary representation is extracted. This serves as a reference for refining other architectures in the population. For each layer in a given chromosome, the current binary representation is compared with the best-performing binary string. If an improvement is needed, quantum rotation gates are applied to gradually adjust the qubit states, guiding the search towards better architectures. These updates increase the probability of selecting higher-performing configurations while still allowing for diversity in exploration. After applying the quantum rotations, the updated qubits are converted back into binary format, effectively refining the chromosome representation. This iterative refinement process ensures that the CNN architectures continue evolving towards optimal configurations while maintaining a balance between exploration and exploitation in the search space.

Table 7: Rotation Angles for Quantum Gates in QIEA.

x_i	$best_i$	$f(x) \geq f(best)$	ϕ (Rotation Angle)	$s(\alpha_i\beta_i)$			
				$\alpha_i\beta_i > 0$	$\alpha_i\beta_i < 0$	$\alpha_i = 0$	$\beta_i = 0$
0	0	F	0	0	0	0	0
0	0	T	0	0	0	0	0
0	1	F	0	0	0	0	0
0	1	T	0.05π	-1	1	±1	0
1	0	F	0.01π	-1	1	±1	0
1	0	T	0.025π	1	-1	0	±1
1	1	F	0.005π	1	-1	0	±1
1	1	T	0.025π	1	-1	0	±1

Unlike traditional evolutionary algorithms, QIEA leverages a probabilistic representation where the probability amplitudes are gradually modified instead of applying direct mutations. The rotation angle ϕ determines the extent of change in each qubit's state during every iteration, guiding the search process toward optimal CNN architectures. Over time, this controlled adjustment ensures that higher-performing architectures become more likely to be selected, leading to an efficient and adaptive search process. This update mechanism allows QIEA to optimize CNN architectures effectively, enhancing performance while significantly reducing computational overhead.

In summary, QIEA for NAS encodes candidate CNNs as probabilistic quantum chromosomes and iteratively collapses them into binary architectures, evaluates their performance, and updates qubit amplitudes via simulated quantum rotations. This cycle of encoding, observation, evaluation, and update efficiently balances exploration and exploitation, accelerating convergence on high-performing network structures. Early stopping and a final retraining phase ensure computational efficiency and model robustness, demonstrating that quantum-inspired techniques can enhance architecture search using only classical hardware.

5 Experimental Results and Performance Evaluation.

The method described in Chapter 4 focuses on automating the design of Convolutional Neural Networks (CNNs) using a Quantum-Inspired Evolutionary Algorithm (QIEA) to improve search efficiency and performance. The process begins with quantum chromosome initialization, where a set of quantum chromosomes is generated, each representing a possible CNN architecture. These chromosomes are encoded using qubits that can exist in multiple states simultaneously, allowing diverse exploration of CNN configurations.

In the quantum observation step, the quantum chromosomes collapse into specific binary strings, converting the superposition of possible architectures into concrete binary-encoded CNNs ready for training. After each training session, the qubit rotation mechanism is applied to the chromosome, adjusting the probability amplitudes using quantum rotation gates. This guides the search toward better-performing architectures based on the results of previous training, helping the algorithm converge towards optimal solutions. The process repeats until a local minimum is found or a set number of iterations is reached.

During the CNN training phase, the observed architectures are trained on a dataset, with accuracy serving as the primary performance metric. Early stopping strategies are applied to prevent overfitting and save computational resources by halting training when no improvements are detected. In the evaluation step, trained CNNs are assessed, and the best-performing architectures are identified and saved. The iterative process of observation, training, rotation, and evaluation continues until the most efficient CNN architecture is discovered.

All experiments were implemented in Python and executed on a Linux based workstation equipped with NVIDIA GPUs to support both the quantum inspired operations and CNN training. At the outset of each run, key QIEA parameters, including population size, rotation angles, and maximum number of iterations, were initialized. A diverse suite of benchmark datasets, including MNIST, CIFAR 10, the Potato Leaf Disease dataset, and a Brain Tumor MRI dataset, was used to evaluate performance and assess the algorithm’s generalization across multiple image classification domains.

5.1 Experimental Setup

This section outlines the experimental environment, datasets used, and initial parameters configured to evaluate the performance of the Quantum-Inspired Evolutionary Algorithm (QIEA) for Neural Architecture Search (NAS).

Table 8 details the computational environment used for all experiments, specifying the GPU, CPU, memory, and operating system components selected to ensure efficient execution of both the quantum-inspired algorithm and CNN training workflows.

Table 8: Hardware Specifications

Component	Specification	Description
GPU	NVIDIA A6000	Accelerated deep learning computations
CPU	16 cores	General processing tasks
RAM	64 GB	Supports large datasets and parallel processing
Operating System	Linux	Stable and flexible for deep learning workloads

5.1.1 Dataset Details

To evaluate the generalization and robustness of the QIEA-based Neural Architecture Search, multiple datasets were selected from different domains. The chosen datasets vary in complexity, structure, and application areas, allowing a comprehensive assessment of the proposed method's effectiveness across diverse image classification tasks. The selection includes simple handwritten digits, complex natural images, medical imaging, and agricultural disease detection, ensuring that the model is tested on datasets with increasing levels of difficulty. This diverse selection helps analyse the adaptability of QIEA-optimized CNN architectures and their ability to generalize to different real-world applications.

The **MNIST** dataset, a classic benchmark for image classification tasks, was used as one of the initial datasets. It contains 60,000 grayscale handwritten digit images (28x28 pixels) for training and 10,000 images for testing. An additional 5,000 images from the training set were set aside for validation. The dataset covers 10 classes, representing digits from 0 to 9. Its simple structure makes it ideal for evaluating the baseline performance of the CNN architectures. MNIST's standardized format, moderate complexity, and extensive historical use as a benchmarking tool make it the de facto baseline for evaluating CNN architectures' capability to learn basic feature hierarchies and generalize to unseen data.

The **CIFAR-10** dataset was employed to test the models on more complex-coloured images. It consists of 60,000 32x32 RGB images spread across 10 classes, including Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, and Truck. The dataset is split into 50,000 training images, 10,000 testing images, and a validation set of 5,000 images extracted from the training data. CIFAR-10 poses a greater challenge compared to MNIST due to its colour channels and diverse image content. Sample images from different classes illustrate the dataset's variability.

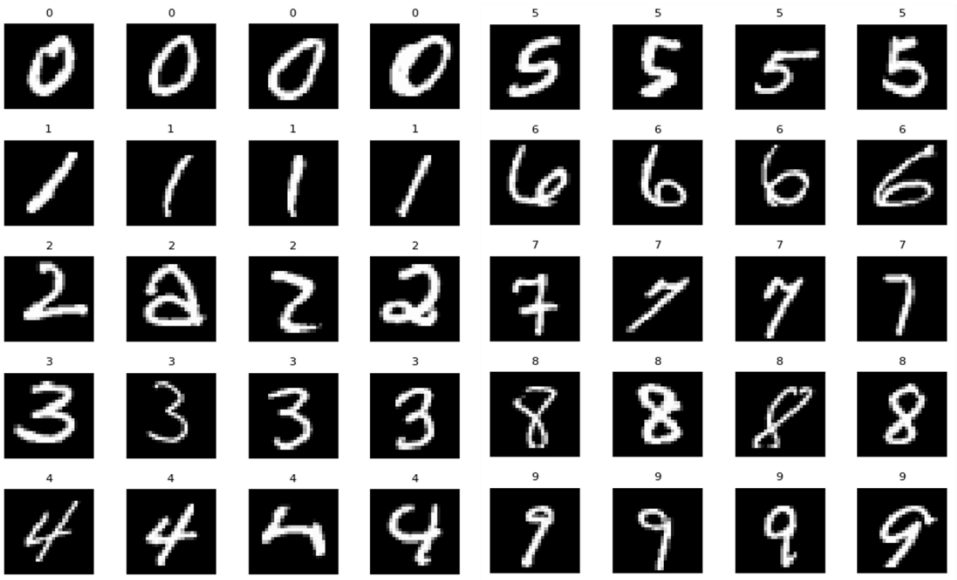


Figure 9: MNIST Dataset [51].

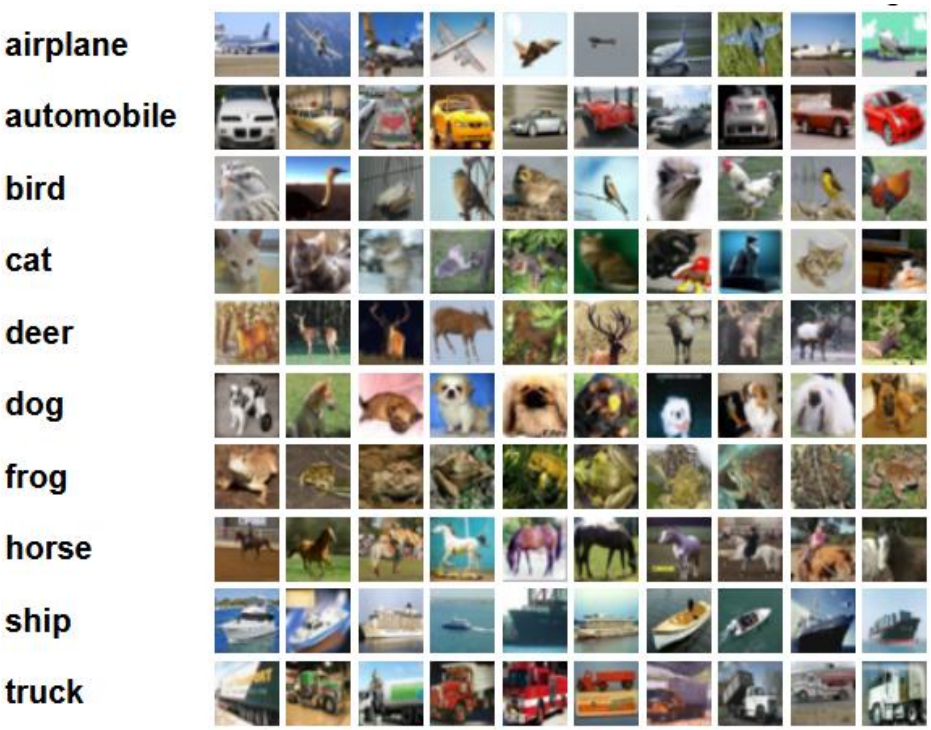


Figure 10: CIFAR10

The **Potato Leaf Disease Dataset** was incorporated to assess the model's performance in plant pathology applications. This dataset includes images of potato leaves categorized into three classes: Healthy, Early Blight, and Late Blight. The dataset comprises 100 images per class for training, 50 images per class for validation, and 80 images per class for testing. Images were resized to [specify input shape, e.g., 150x150x3] to ensure consistency across the experiments. Sample images from the

dataset highlight the visual differences between healthy and diseased leaves, emphasizing the complexity of the classification task.

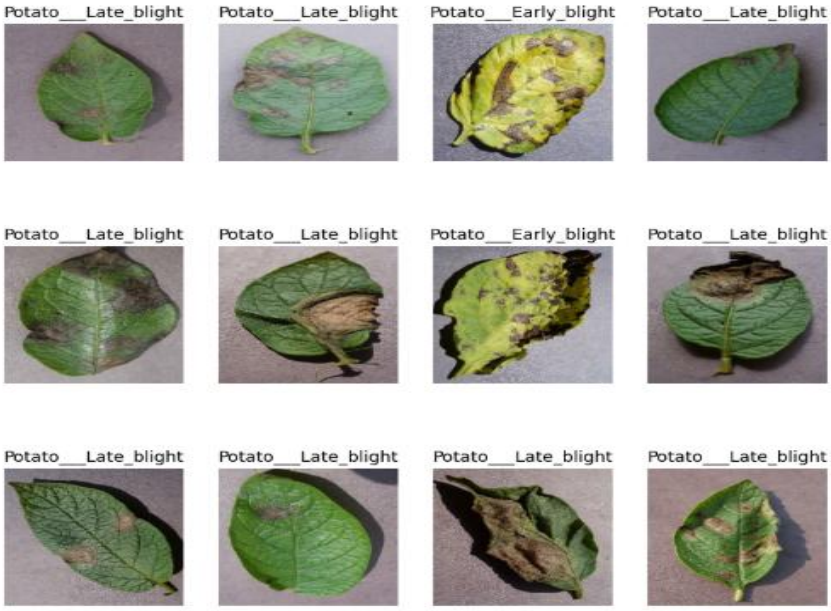


Figure 11:Potato Leaf disease Dataset sample images and class names

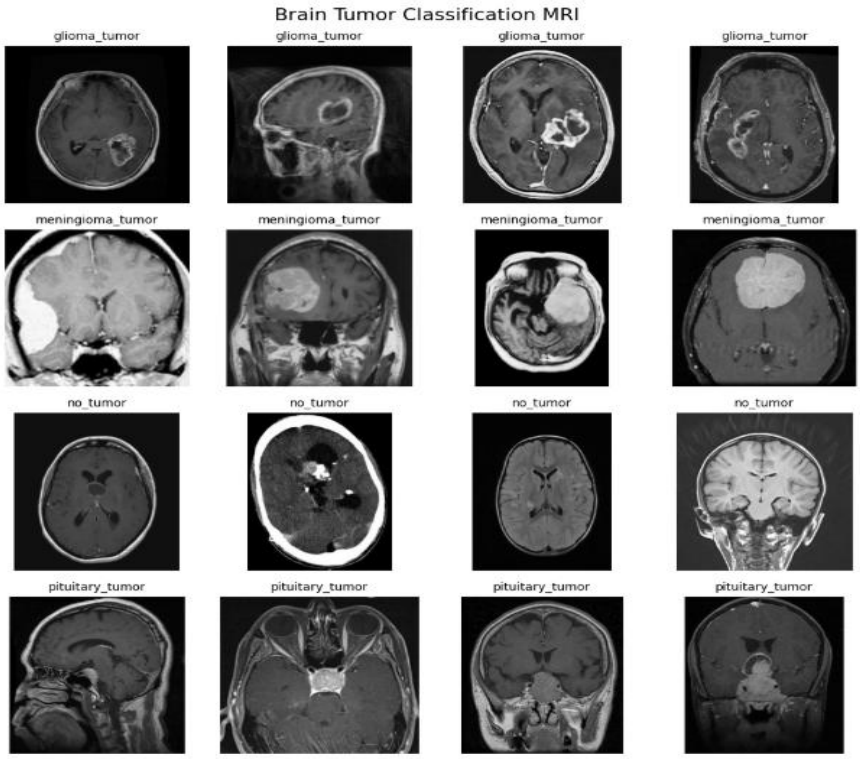


Figure 12:Brain Tumour Dataset images and class names.

Lastly, the **Brain Tumor Dataset** was used to evaluate the CNN architectures in the medical imaging domain. It contains MRI images labeled for tumor detection, categorized into four classes: Healthy, Glioma Tumor, Meningioma Tumor, and Pituitary Tumor. The dataset was split into training, validation, and testing sets, with [specify number] images allocated to each. Images were resized to [specify input shape] to match the model's input requirements. This dataset presents a significant challenge due to the subtle differences between healthy and tumor-affected tissues. Example images from each class provide insights into the dataset's complexity.

The Potato Leaf Disease and Brain Tumor datasets were preprocessed using a suite of image augmentation techniques such as random rotations, horizontal and vertical flips, zoom transformations, and pixel-value normalization, to enhance model generalization. All images were then resized to match the input dimensions required by the CNN architectures under evaluation. For comparative purposes, LeNet-5 was employed on the MNIST dataset, while AlexNet was used for CIFAR-10, the Potato Leaf Disease dataset, and the Brain Tumor dataset. This consistent selection of standard CNN backbones ensures a fair and meaningful assessment of the QIEA-optimized architectures against well-established baselines.

All the datasets used in this study were split based on an 80-20 ratio for training and validation. Specifically, 80% of the images from each dataset were allocated to the training set, while the remaining 20% were used for validation. This consistent split ensures a balanced approach to model training and hyperparameter tuning, allowing the models to generalize effectively while still being evaluated on unseen validation data. The testing sets for each dataset remain separate and untouched during training and validation, serving as an unbiased benchmark for final performance evaluation.

5.2 Performance Metrics

The performance evaluation of the Quantum-Inspired Evolutionary Algorithm based CNN architecture search was primarily focused on classification accuracy. Accuracy was used as the main metric to assess how well the optimized CNN architectures performed on the validation datasets. The final accuracy achieved by the best-performing CNN was recorded, providing a clear indication of the model's effectiveness. Although the primary focus was on accuracy, loss values were also monitored during training to ensure proper learning. Training and validation accuracy curves were plotted to visualize the model's progress.

In terms of computational efficiency, the study did not focus on detailed metrics such as GPU memory usage or floating-point operations per second (FLOPs). However, the total time taken to complete the entire program was recorded for every training process and for whole papulation. This included the time for initializing quantum chromosomes, training each CNN, applying quantum rotations, and evaluating architectures. Recording the total training time provided a general sense of the computational demands, even though specific efficiency metrics were not measured.

Table 9 lists the key hyperparameters and algorithmic variables that must be initialized prior to each run of the QIEA-based NAS program. These include population size, CNN layer configuration settings, training epochs, early stopping criteria, and optimization parameters, all of which can be adjusted to study their impact on search performance and model accuracy.

Table 9: Initial Parameters for QIEA and CNN Training.

Parameter	**Value**
Population Size	7-10 chromosomes
Training Epoch for Evaluation	25 iterations per chromosome
Rotation Angles	Dynamically adjusted based on fitness evaluations
Epochs per Training	10-15 epochs
Batch Size	16-128 images per batch
Learning Rate	0.01 - 0.001
Loss Function	Sparse Categorical Crossentropy
Performance Metric	Sparse Categorical Accuracy
Epoch-Based Early Stopping	Stops if validation accuracy doesn't improve after 80% of total epochs
Iteration-Based Early Stopping	Stops if accuracy stagnates for 80% of iterations
Training Epoch for best chromosome	30-50
Max CNN Layers (L)	10-15
Max Fully-Connected Layers (F)	2-4
Activation Function	ReLU
Optimizer	Adam, Adamax

Additionally, an epoch-based early stopping criterion was employed during CNN training: the process was terminated if validation accuracy failed to improve over 80 % of the total epochs, thereby reducing unnecessary computation. Likewise, at the chromosome iteration level, the algorithm compared the current best accuracy with that of previous iterations and halted further evolution when no improvement was observed over a predefined sequence of iterations.

Overall, the evaluation was centered around accuracy as the core performance metric, complemented by the measurement of total training time to give insights into the computational cost. This approach provided a straightforward yet effective means of assessing the performance of the QIEA-based CNN architecture search.

5.3 Visual Representation of Results and Analysis

The experimental results of the Quantum-Inspired Evolutionary Algorithm (QIEA) applied to optimize Convolutional Neural Network (CNN) architectures across different datasets. The visualizations include accuracy trends over training epochs. These graphs provide insight into the learning behavior and convergence characteristics of the QIEA-based NAS framework.

The Accuracy vs. Training Epochs graphs highlight how the best-performing CNN architectures, identified by QIEA, improve over successive epochs, demonstrating their capacity to learn and generalize effectively on the training and validation datasets. Meanwhile, the Accuracy vs. Iterations plots illustrate how each chromosome in the population evolves through quantum rotation operations. These visualizations clearly indicate that QIEA progressively converges toward optimal CNN structures with higher accuracy.

5.3.1 Results on MNIST Dataset

The Quantum-Inspired Evolutionary Algorithm (QIEA) was tested on the MNIST dataset to evaluate its effectiveness in optimizing Convolutional Neural Network (CNN) architectures. The experiment consisted of 10 independent test runs, with each test generating 10 different chromosomes representing candidate CNN architectures. From each test, the best-performing chromosome was selected and further trained for additional epochs to refine its accuracy. The goal was to assess whether QIEA successfully converges to an optimal architecture by iteratively refining the quantum chromosome representations.

The accuracy vs. epochs graph (Figure 12), provides validation of QIEA's performance. The graph illustrates a clear upward trend in accuracy as training epochs progress, demonstrating that the selected CNN architectures continue to improve with further training. Initially, accuracy starts at a lower value but gradually increases, indicating effective learning over successive epochs. The final training accuracy achieved is 99.30%, while the validation accuracy is 99.22%, and the test accuracy reaches 99.33%. This consistency between validation and test accuracy further confirms the generalization capability of the optimized CNN architecture. The steady increase in accuracy after each epoch highlights the effectiveness of QIEA in identifying well-structured CNNs capable of learning efficiently. The narrowing gap between training and validation accuracy suggests reduced overfitting, emphasizing the robustness of the discovered architecture. This result confirms that QIEA not only converges to a promising architecture but also enhances the training process, leading to a more optimized final model with high predictive accuracy.

Overall, the experimental results validate the effectiveness of QIEA for Neural Architecture Search (NAS). The observed convergence pattern in accuracy across iterations and epochs provides compelling evidence that QIEA successfully refines CNN architectures by leveraging quantum principles.

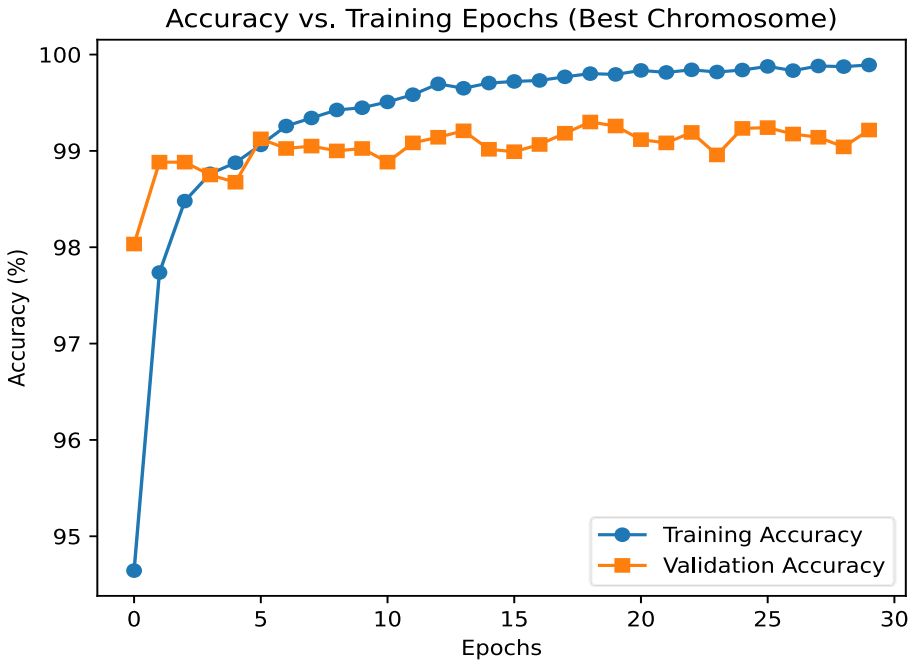


Figure 13:Accuracy over epochs for the best MNIST chromosome

To evaluate the performance of the Quantum-Inspired Evolutionary Algorithm (QIEA) for Neural Architecture Search (NAS), a comparative analysis was conducted between the well-established LeNet-5 architecture and the CNN model constructed using the QIEA-generated dot-decimal representation. The accuracy progression of both models over multiple training epochs is depicted in Figure 13. The results illustrate that while both architectures exhibit a steady increase in accuracy, the QIEA-designed CNN demonstrates a competitive performance, with its validation accuracy closely following that of LeNet-5. This suggests that the QIEA-based NAS effectively explores optimal configurations, generating architectures that can achieve comparable accuracy to manually designed networks. Moreover, the validation accuracy trends indicate the model's generalization capability, with both architectures stabilizing after a few epochs. The findings highlight the potential of QIEA-NAS in automating architecture search while maintaining high classification accuracy, reinforcing its applicability in deep learning model design

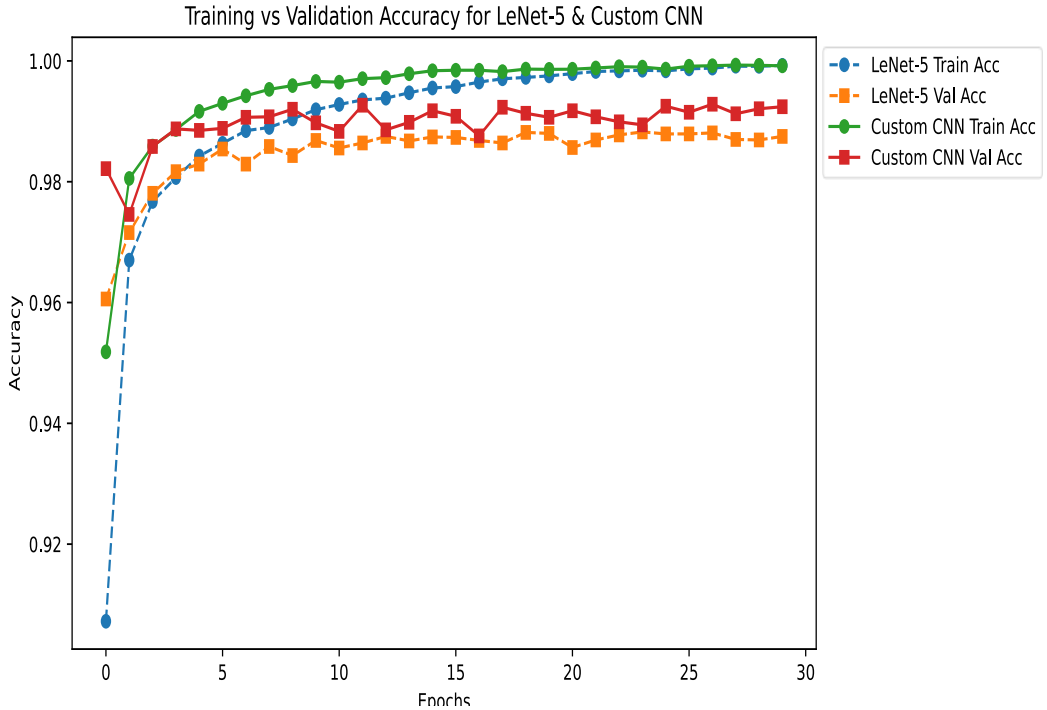


Figure 14: LeNet-5 vs. QIEA-CNN accuracy on MNIST.

5.3.2 Results on CIFAR-10 Dataset

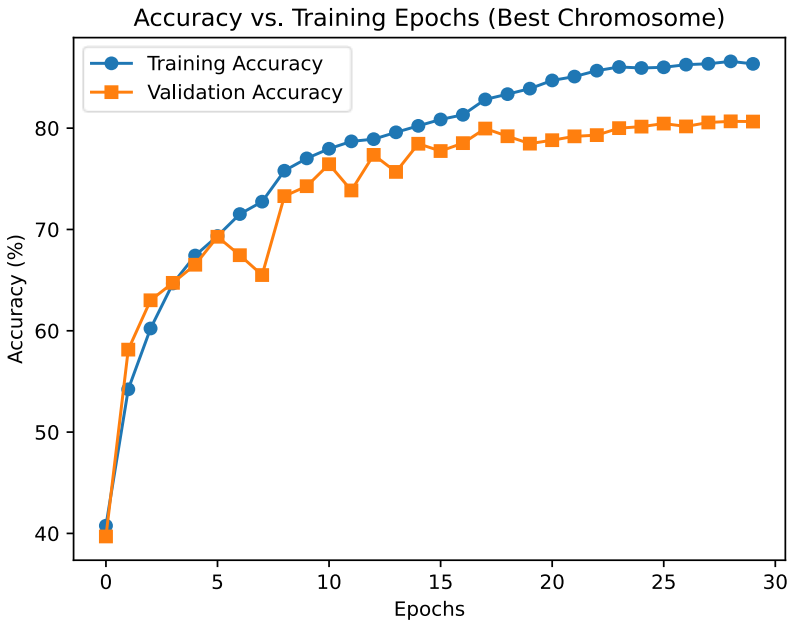


Figure 15: Training and validation accuracy over epochs on CIFAR-10.

The graph depicts the training and validation accuracy progression over 30 epochs for the best-performing chromosome generated during the NAS process on the CIFAR-10

dataset. CIFAR-10 is a widely adopted benchmark for image-classification research because its moderate complexity, three color channels, and well-established baselines facilitate direct comparisons across studies. In this experiment, the QIEA-optimized model demonstrates rapid improvement during the initial epochs—achieving approximately 87 % training accuracy and stabilizing around 81 % validation accuracy—with a small generalization gap indicative of effective feature learning and minimal overfitting.

To ensure a fair comparison with a standard CNN architecture, key hyperparameters—namely the number of layers, optimizer choice, and learning rate—were aligned with those used in AlexNet, while all other training settings (data augmentation, and early-stopping criteria) remained identical. Although we initially planned to retrain AlexNet on CIFAR-10, hardware constraints (insufficient GPU memory when upscaling 32×32 images to AlexNet's 224×224 input) rendered this infeasible. Consequently, CIFAR-10 serves as the primary benchmark for performance evaluation.

5.3.3 Results on Potato Leaf Disease Dataset

Potato leaf disease detection represents a real-world agricultural challenge, with significant implications for crop yield and food security. Unlike standardized benchmarks such as MNIST or CIFAR-10, this dataset comprises varied, high-resolution images of healthy and diseased leaves, making it a more realistic test of model robustness. Manually designing a custom CNN tailored to these images requires extensive trial and error in selecting layer configurations, filter sizes, and regularization strategies. By contrast, QIEA automates the architecture search process, efficiently discovering a specialized CNN that achieves high accuracy on this task. To benchmark against a well-known standard, AlexNet was trained, using identical hyperparameter settings to those used by QIEA enabling a direct comparison of performance gains attributable to the quantum-inspired search.

The figure 15 illustrates the performance of the best CNN architecture optimized using QIEA-based NAS. As the training progresses, the training accuracy steadily increases, reaching 94.5% after around 50 epochs. This upward trend indicates that the model effectively learns from the data and identifies key features necessary for the classification task. The validation accuracy stabilizes at 94.23%**,** showing that the model generalizes well to unseen data. The small gap between training and validation accuracy suggests minimal overfitting, indicating that the model is not memorizing the training data but instead generalizing the learned patterns effectively. These results highlight the effectiveness of QIEA in optimizing CNN architectures for the task of plant disease detection, achieving high performance in both training and validation phases.

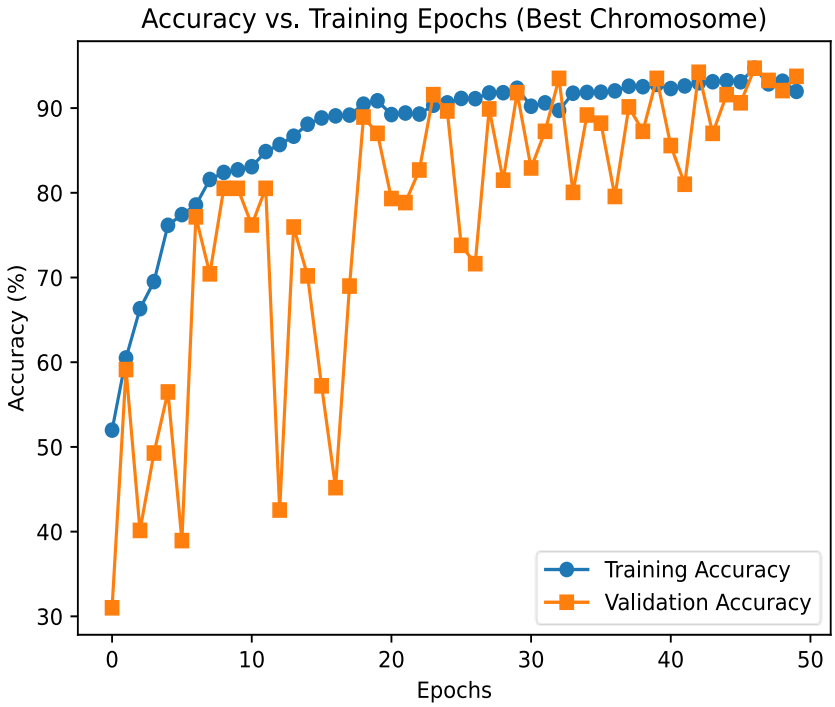


Figure 16:Accuracy progression over epochs Potato leaf disease Dataset.

The graph compares the performance of AlexNet and a Custom CNN, both of which have a similar number of layers. The AlexNet model shows relatively poor performance, with both training and validation accuracy stabilizing at approximately 40%. This suggests that AlexNet is not effectively learning or generalizing on the given task. In contrast, the Custom CNN demonstrates a much better performance. While its validation accuracy fluctuates throughout the training process, the overall trend shows noticeable improvements, reaching a peak of around 85% at the end of the training epochs. The training accuracy consistently increases and approaches 90%, indicating that the model is learning well from the data, although some fluctuations in validation accuracy suggest occasional challenges in generalizing.

This behavior in the Custom CNN suggests that, while the model is improving overall, its performance may not be as stable across all epochs, as it occasionally experiences minor setbacks. However, the Custom CNN still outperforms AlexNet, demonstrating better generalization and adaptability to the task. The disparity in performance between the two models highlights the importance of fine-tuning and customizing architectures to suit specific tasks, as well as the need for methods like early stopping or regularization to stabilize the learning process and enhance generalization. Despite the fluctuations, the Custom CNN's performance suggests that a more tailored approach to architecture design can be more effective than using established models like AlexNet in certain contexts.

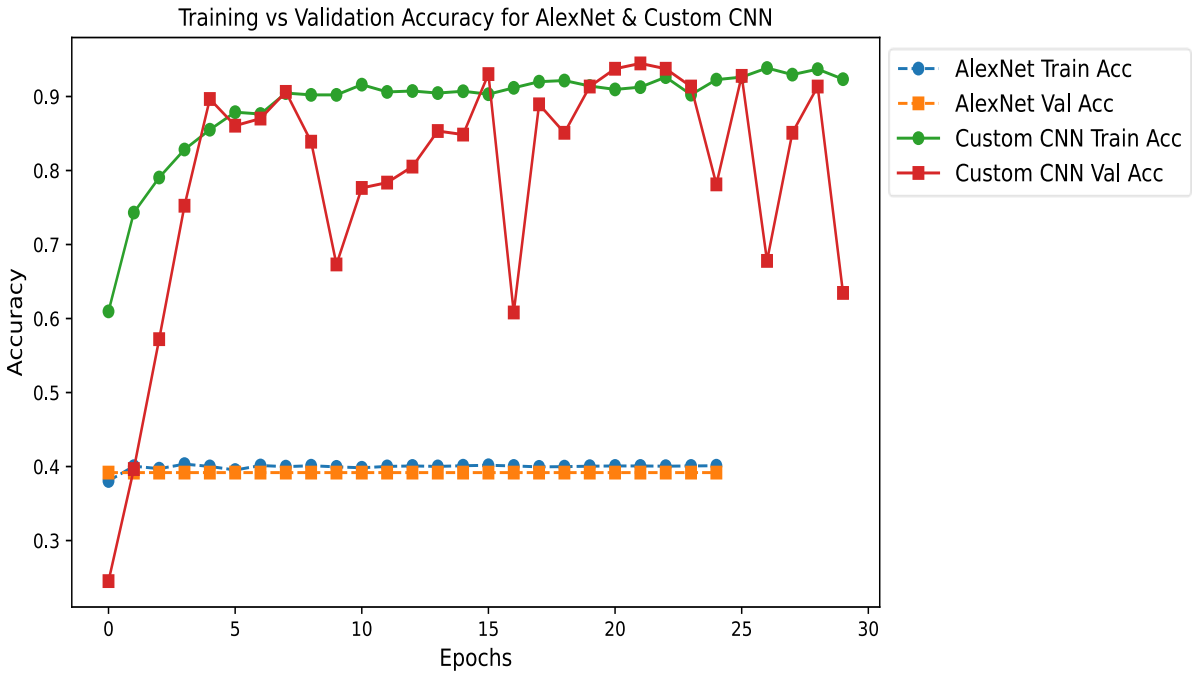


Figure 17: AlexNet vs. QIEA-CNN accuracy on Potato Leaf Disease.

5.3.4 Results on Brain Tumor Dataset

Brain tumor classification from MRI scans presents a critical healthcare challenge, as accurate early diagnosis can significantly influence patient outcomes. This dataset contains heterogeneous, high-resolution images, making it far more complex than controlled benchmarks like MNIST or CIFAR-10. Designing a bespoke CNN for this application demands extensive experimentation with network depth, kernel sizes, and regularization methods to capture subtle tissue contrasts. QIEA automates this design process by efficiently searching the architectural space for an optimal model tailored to tumor detection. For comparative evaluation, AlexNet was trained with identical hyperparameter settings to those used in the QIEA framework, thereby providing a direct baseline against which to measure the improvements afforded by the quantum-inspired search.

The Accuracy vs. Training Epochs graph (figure 17) for the Brain Tumor dataset shows the performance of the best chromosome during training over 50 epochs. The training accuracy starts at a lower value and steadily increases, eventually stabilizing around 90% by the 50th epoch. This consistent improvement in training accuracy indicates that the model is successfully learning from the training data, gradually enhancing its ability to classify brain tumor images.

On the other hand, the validation accuracy fluctuates throughout the training process. While it starts at approximately 30%, it gradually increases, but the progress is not as smooth as the training accuracy. By the end of the training, the validation accuracy stabilizes around 70%, indicating that the model is struggling to generalize well to the unseen validation data. These fluctuations suggest that the model may be overfitting

at certain epochs, where it performs well on the training data but not as effectively on the validation set.

Overall, the Custom CNN demonstrates strong performance on the training data, but the validation accuracy indicates that the model's generalization capabilities could be improved. The fluctuations in validation accuracy suggest the potential need for additional regularization techniques or other improvements to enhance the model's ability to generalize better to new, unseen data.

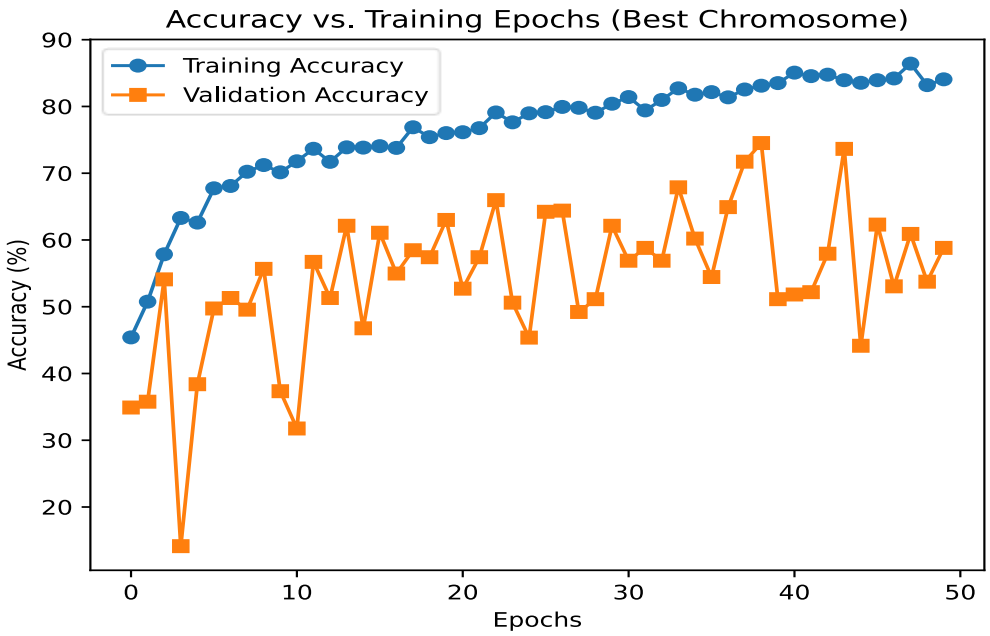


Figure 18:Accuracy graph for the best chromosome on the Brain tumor MRI.

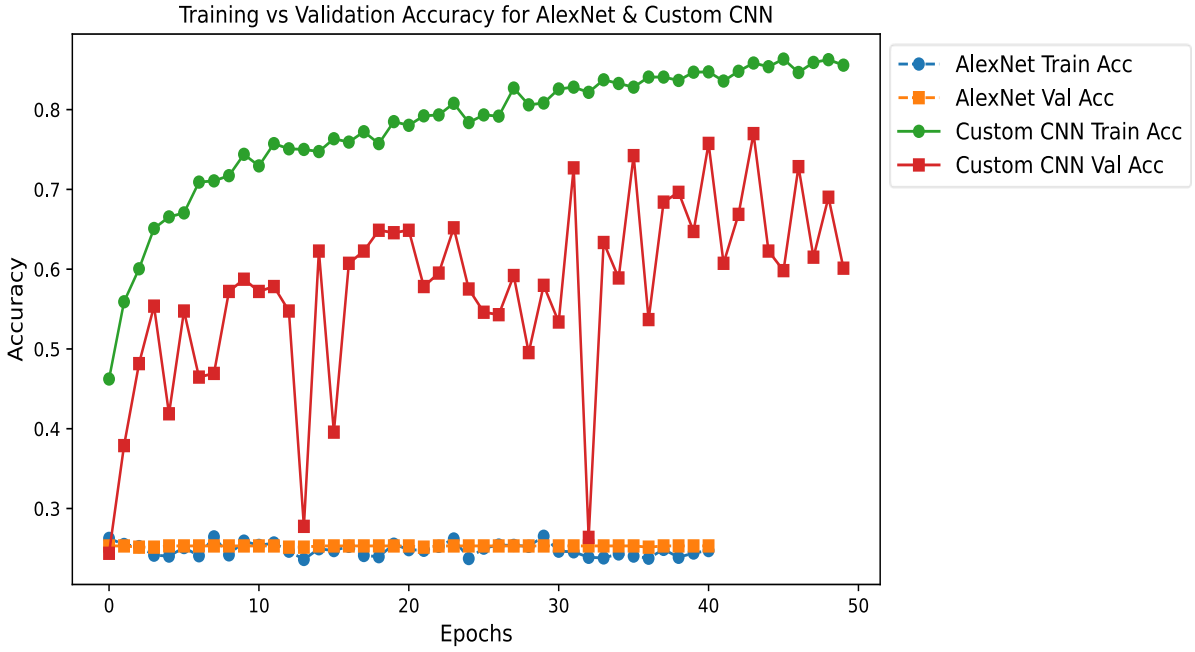


Figure 19:AlexNet vs. QIEA-CNN accuracy on Brain Tumor MRI

Figure 18 illustrates the training and validation accuracy trends over 50 epochs for two models: AlexNet and a Custom CNN. The Custom CNN demonstrates a steady improvement in both training and validation accuracy. In contrast, AlexNet maintains consistently low accuracy for both training and validation, suggesting underfitting or possible issues with model configuration or dataset suitability. The comparison highlights the superior performance of the Custom CNN for the given task.

6 Conclusion

The research presented in this thesis implemented a Quantum-Inspired Evolutionary Algorithm (QIEA) for the automated optimization of Convolutional Neural Network (CNN) architectures. The algorithm was evaluated across four diverse datasets: MNIST, CIFAR-10, Potato Leaf Disease, and Brain Tumor. The results demonstrate that QIEA is capable of discovering high-performing CNN architectures tailored to specific tasks. On the MNIST dataset, the model achieved a training accuracy of 99.3%, with both validation and test accuracies reaching 99.22%, indicating excellent generalization. For CIFAR-10, a more complex and colorful image dataset, the model obtained a training accuracy of 86.34%, validation accuracy of 80.65%, and test accuracy of 79.87%, which are respectable given the dataset's difficulty. In the Potato Leaf Disease dataset, the algorithm performed impressively, achieving 95.18% training accuracy, 94.47% validation accuracy, and 97.28% test accuracy, highlighting its suitability for agricultural image classification. However, on the Brain Tumor dataset, which presents high intra-class variability and limited data, the model faced challenges, attaining a training accuracy of 84.76%, but with a significant drop in validation (58.81%) and test accuracy (52.03%), suggesting overfitting and potential data imbalance. These trends confirm QIEA's adaptability and effectiveness across datasets, while also identifying areas for improvement in complex or data-limited domains.

While QIEA has demonstrated notable effectiveness, several areas offer opportunities for enhancement. One promising direction involves integrating prediction network performance as a criterion for early stopping, which could prevent overfitting and reduce unnecessary computational overhead. Additionally, refining the quantum rotation angle scheduling and improving the fitness evaluation criteria may lead to more stable convergence and efficient exploration of the architecture search space. As quantum computing technologies continue to advance, the partial integration of actual quantum processors into the framework could pave the way for hybrid quantum-classical neural architecture search (NAS) models, potentially unlocking further performance gains. Expanding the evaluation scope is also critical; while this study focused on image classification tasks, future research could extend QIEA to address more complex and diverse domains such as object detection, medical image segmentation, and multi-modal learning scenarios. Moreover, assessing the algorithm on larger-scale and imbalanced datasets would offer deeper insights into its robustness, generalization capabilities, and scalability across real-world applications.

A primary direction for subsequent research is the integration of actual quantum hardware into the QIEA framework. As quantum processors become increasingly available, deploying QIEA on hybrid quantum (classical platforms may exploit genuine quantum phenomena such as entanglement and superposition) to accelerate convergence and enlarge the effective search space [53].

In addition, extending QIEA to a multi-objective optimization paradigm will improve its practical utility. Rather than optimizing solely for classification accuracy, future implementations should concurrently consider model complexity, inference latency, and energy consumption. Such a comprehensive fitness evaluation will yield neural architectures better suited for deployment in resource-constrained or real-time environments.

Moreover, the applicability of QIEA can be broadened beyond image classification to encompass tasks such as object detection, semantic segmentation, and time-series forecasting. Adapting the quantum-inspired encoding and evaluation mechanisms to these domains will demonstrate the generality and robustness of the approach.

Finally, embedding hyperparameter optimization (automatically tuning learning rates, batch sizes, and regularization coefficients) within the QIEA loop promises to enhance overall training performance and reduce manual intervention, resulting in more efficient and reliable model development.

7 Bibliography

[1] M. Soori, B. Arezoo, and R. Dastres, "Artificial intelligence, machine learning and deep learning in advanced robotics, a review," *Cognitive Robotics*, vol. 3, 2023, doi: 10.1016/j.cogr.2023.04.001.

[2] A. Esteva *et al.,* "A guide to deep learning in healthcare," *Nature medicine*, early access. doi: 10.1038/s41591-018-0316-z.

[3] W. Zhou, H. Wang, and Z. Wan, "Ore Image Classification Based on Improved CNN," *Computers and Electrical Engineering*, vol. 99, p. 107819, 2022, doi: 10.1016/j.compeleceng.2022.107819.

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, 1998, doi: 10.1109/5.726791.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," 2012.

[6] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," Sep. 2014. [Online]. Available: http://arxiv.org/pdf/1409.1556v6

[7] C. Szegedy *et al.,* "Going Deeper with Convolutions," Sep. 2014. [Online]. Available: http://arxiv.org/pdf/1409.4842v1

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," Dec. 2015. [Online]. Available: http://arxiv.org/pdf/1512.03385v1

[9] Z. Shen, Y. Yang, Q. She, C. Wang, J. Ma, and Z. Lin, "Newton design: designing CNNs with the family of Newton's methods," *Sci. China Inf. Sci.*, vol. 66, no. 6, 2023, doi: 10.1007/s11432-021-3442-2.

[10] Ning Wang *et al.,* "NAS-FCOS: Fast Neural Architecture Search for Object Detection," vol. 4, 2020.

[11] Ning Wang, Yang Gao, Hao Chen, Peng Wang, Zhi Tian, Chunhua Shen, Yanning Zhang, "NAS-FCOS: Fast Neural Architecture Search for Object Detection," vol. 4, 2020.

[12] T. Elsken, J. H. Metzen, and F. Hutter, "Neural Architecture Search: A Survey," 2019. [Online]. Available: http://arxiv.org/pdf/1808.05377v3

[13] H. Zhou, M. Yang, J. Wang, and W. Pan, "BayesNAS: A Bayesian Approach for Neural Architecture Search," May. 2019. [Online]. Available: http://arxiv.org/pdf/1905.04919v2

[14] B. Zoph and Q. Le V, "Neural Architecture Search with Reinforcement Learning," Nov. 2016. [Online]. Available: http://arxiv.org/pdf/1611.01578v2

[15] Thomas Elsken, Jan Hendrik Metzen,Frank Hutter, "Neural Architecture Search: A Survey," 2019. [Online]. Available: http://arxiv.org/pdf/1808.05377v3

[16] J. Jin, Q. Zhang, J. He, and H. Yu, "Quantum Dynamic Optimization Algorithm for Neural Architecture Search on Image Classification," *Electronics*, vol. 11, no. 23, p. 3969, 2022, doi: 10.3390/electronics11233969.

[17] K.-H. Han and J.-H. Kim, "Quantum-inspired evolutionary algorithm for a class of combinatorial optimization," *IEEE Trans. Evol. Computat.*, vol. 6, no. 6, pp. 580–593, 2002, doi: 10.1109/TEVC.2002.804320.

[18] D. Szwarcman, D. Civitarese, and M. Vellasco, "Quantum-inspired evolutionary algorithm applied to neural architecture search," *Applied Soft Computing*, vol. 120, p. 108674, 2022, doi: 10.1016/j.asoc.2022.108674.

[19] M. D. Platel, S. Schliebs, and N. Kasabov, "Quantum-Inspired Evolutionary Algorithm: A Multimodel EDA," *IEEE Trans. Evol. Computat.*, vol. 13, no. 6, pp. 1218–1232, 2009, doi: 10.1109/TEVC.2008.2003010.

[20] X. Liu, J. Li, J. Zhao, B. Cao, R. Yan, and Z. Lyu, "Evolutionary Neural Architecture Search and Its Applications in Healthcare," *CMES*, vol. 139, no. 1, pp. 143–185, 2024, doi: 10.32604/cmes.2023.030391.

[21] S. Ali and M. A. Wani, "Gradient-Based Neural Architecture Search: A Comprehensive Evaluation," *MAKE*, vol. 5, no. 3, 2023, doi: 10.3390/make5030060.

[22] Anshumaan Chauhan, Siddhartha Bhattacharyya,S. Vadivel, "DQNAS: Neural Architecture Search using Reinforcement Learning," vol. 1, Jan. 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2301.06687

[23] B. Zoph and Q. Le V, "Neural Architecture Search with Reinforcement Learning," Nov. 2016. [Online]. Available: http://arxiv.org/pdf/1611.01578v2

[24] L. Li and A. Talwalkar, "Random Search and Reproducibility for Neural Architecture Search," *Conference on Uncertainty in Artificial Intelligence (UAI).* [Online]. Available: http://arxiv.org/pdf/1902.07638v3

[25] Yuhong Li, Cong Hao, Xiaofan Zhang, Jinjun Xiong, Wen-mei Hwu, Deming Chen, "Improving Random-Sampling Neural Archi-Tecture Search By Evolving The Proxy Search Space," 2021.

[26] L. Fogel, A. J. Owens, and M. J. Walsh, "Artificial Intelligence through Simulated Evolution," 1966. [Online]. Available: https://www.semanticscholar.org/paper/Artificial-Intelligence-through-Simulated-Evolution-Fogel-Owens/69022c885504a091680cf2dc9cfc84597332ac69

[27] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia tools and applications*, early access. doi: 10.1007/s11042-020-10139-6.

[28] M. J. Willis, H.G Hiden, P. Marenbach, B. McKay and G.A. Montague, "Genetic Programming: An Introduction And Survey Of Applications - Second International

Conference On Genetic Algorithms In Engineering Systems:Innovations And Appli," early access. doi: 10.1049/cp:19971199.

[29] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, Perth, WA, Australia, 1995, pp. 1942–1948, doi: 10.1109/ICNN.1995.488968.

[30] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A Survey on Evolutionary Neural Architecture Search," *IEEE transactions on neural networks and learning systems*, early access. doi: 10.1109/TNNLS.2021.3100554.

[31] V. Kachitvichyanukul, "Comparison of Three Evolutionary Algorithms: GA, PSO, and DE," *Industrial Engineering and Management Systems*, vol. 11, 2012, doi: 10.7232/iems.2012.11.3.215.

[32] Hanxiao Liu, Karen Simonyan,Yiming Yang, "DARTS: DIFFERENTIABLE ARCHITECTURE SEARCH," 2018. [Online]. Available: http://arxiv.org/pdf/1806.09055v2

[33] V. Reers and J. Lässig, "A new Pattern for Quantum Evolutionary Algorithms," 2022, doi: 10.18420/INF2022_98.

[34] Weiliang Ye, Ruijiao Liu,Yangyang Li,Licheng Jiao, *Quantum-Inspired Evolutionary Algorithm for Convolutional Neural Networks Architecture Search: 2020 conference proceedings*. Piscataway, NJ, USA: IEEE, 2020. [Online]. Available: https://ieeexplore.ieee.org/servlet/opac?punumber=9178820

[35] J. Jin, Q. Zhang, J. He, and H. Yu, "Quantum Dynamic Optimization Algorithm for Neural Architecture Search on Image Classification," *Electronics*, vol. 11, no. 23, p. 3969, 2022, doi: 10.3390/electronics11233969.

[36] A. Narayanan and M. Moore, "Quantum-inspired genetic algorithms," in *Proceedings of IEEE International Conference on Evolutionary Computation*, Nagoya, Japan, 1996, pp. 61–66, doi: 10.1109/ICEC.1996.542334.

[37] Pariwat Ongsulee, *Proceedings 2017 Fifteenth International Conference on ICT and Knowledge Engineering: November 22-24, 2017, Bangkok, Thailand*. Piscataway, NJ: IEEE, 2017. [Online]. Available: http://ieeexplore.ieee.org/servlet/opac?punumber=8250945

[38] Md Zahangir Alom1, Tarek M. Taha1, Chris Yakopcic1, Stefan Westberg1, Paheding Sidike2, Mst Shamima Nasrin1, Brian C Van Essen3, Abdul A S. Awwal3, and Vijayan K. Asari1, "The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches," vol. 2, 2018.

[39] P. Purwono, A. Ma'arif, W. Rahmaniar, H. I. K. Fathurrahman, A. Z. K. Frisky, and Q. M. u. Haq, "Understanding of Convolutional Neural Network (CNN): A Review," *IJRCS*, vol. 2, no. 4, 2022, doi: 10.31763/ijrcs.v2i4.888.

[40] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks," Nov. 2015. [Online]. Available: http://arxiv.org/pdf/1511.08458v2

[41] G. Wei, G. Li, J. Zhao, and A. He, "Development of a LeNet-5 Gas Identification CNN Structure for Electronic Noses," *Sensors (Basel, Switzerland)*, early access. doi: 10.3390/s19010217.

[42] Barry J. Wythoff, "Backpropagation neural networks A tutorial," 1992. [Online]. Available: https://doi.org/10.1016/0169-7439(93)80052-J

[43] S. Sood and H. Singh, "Effect of Kernel Size in Deep Learning-Based Convolutional Neural Networks for Image Classification," *ECS Trans.*, vol. 107, no. 1, pp. 8877–8884, 2022, doi: 10.1149/10701.8877ecst.

[44] Jianxin Wu, "Introduction to Convolutional Neural Networks," 2017.

[45] Prashant, "A Study on the basics of Quantum Computing," vol. 3, 2007.

[46] N. S. Yanofsky, "An Introduction to Quantum Computing," in *Proof, Computation and Agency*, J. van Benthem, A. Gupta, and R. Parikh, Eds., Dordrecht: Springer Netherlands, 2011.

[47] G. Benenti, G. Casati, and G. Strini, "Principles of Quantum Computation and Information," 2005.

[48] Y. Kanamori and S.-M. Yoo, "Quantum Computing: Principles and Applications," *Journal of International Technology and Information Management*, vol. 29, no. 2, pp. 43–71, 2020, doi: 10.58729/1941-6679.1410.

[49] M. T. Dejpasand and M. Sasani Ghamsari, "Research Trends in Quantum Computers by Focusing on Qubits as Their Building Blocks," *Quantum Reports*, vol. 5, no. 3, 2023, doi: 10.3390/quantum5030039.

[50] H. A. Bhat, F. A. Khanday, B. K. Kaushik, F. Bashir, and K. A. Shah, "Quantum Computing: Fundamentals, Implementations and Applications," *IEEE Open J. Nanotechnol.*, vol. 3, pp. 61–77, 2022, doi: 10.1109/OJNANO.2022.3178545.

[51] Yelleti Vivek1,2, Vadlamani Ravi1*, P. Radha Krishna2, "Quantum-Inspired Evolutionary Algorithms for Feature Subset Selection: A Comprehensive Survey," vol. 1, 2024. [Online]. Available: https://arxiv.org/abs/2407.17946

[52] O. Montiel, Y. Rubio, C. Olvera, and A. Rivera, "Quantum-Inspired Acromyrmex Evolutionary Algorithm," *Scientific reports*, early access. doi: 10.1038/s41598-019-48409-5.

[53] J. Liu, K. H. Lim, K. L. Wood, W. Huang, C. Guo, and H.-L. Huang, "Hybrid quantum-classical convolutional neural networks," *Sci. China Phys. Mech. Astron.*, vol. 64, no. 9, 2021, doi: 10.1007/s11433-021-1734-3.

8 Appendix A

Software Frameworks and Libraries

Software/Library	Version	Purpose/Description
Programming Language	Python 3.8	Programming environment
TensorFlow	2.8.0	Deep learning framework
CUDA	11.2	GPU acceleration library
cuDNN	8.1.2	Deep learning optimization library for GPUs
Keras	Integrated with TensorFlow	High-level API for building models
numpy	-	Numerical computations library
scikit-learn	-	Used for tasks like train-test splitting and preprocessing
matplotlib	-	Library for visualizing training progress and results
Optimizer- Adam, Adadelta	-	Optimization algorithm for training CNNs
Sparse Categorical Cross entropy	-	Loss function for classification tasks
Sparse Categorical Accuracy	-	Metric to evaluate classification accuracy
L2 Regularization	-	Regularization technique to prevent overfitting
HeNormal initializer	-	Weight initializer for ReLU-based models
ImageDataGenerator	-	For data augmentation
Batch Normalization and Dropout	-	Layers for normalization and preventing overfitting

Software and Library

1. Programming Language (Python 3.8)
   Python is the primary language used for developing the entire workflow due

to its flexibility, ease of use, and rich ecosystem of libraries. Python's simplicity and extensive support for machine learning frameworks like TensorFlow make it ideal for the QIEA-based NAS implementation.

2. TensorFlow (2.8.0)
   TensorFlow is the core deep learning framework used in this study. It provides an extensive platform for developing and deploying machine learning models, offering tools to define, train, and evaluate CNNs. TensorFlow 2.8.0 brings improved ease of use, enhanced performance, and GPU acceleration.
3. CUDA and cuDNN
   CUDA and cuDNN are libraries used for GPU acceleration. CUDA is a parallel computing platform and programming model developed by NVIDIA, allowing TensorFlow to leverage NVIDIA GPUs for faster computation. cuDNN is a GPU-accelerated library for deep neural networks that provides highly optimized implementations of standard deep learning operations, such as convolutions and matrix multiplications. Together, they allow TensorFlow to perform computations much faster than on a CPU alone, making the training of deep learning models more efficient.
4. Keras (integrated with TensorFlow)
   Keras is a high-level API integrated within TensorFlow, used to quickly build and experiment with CNN architectures. It abstracts the complexities of the underlying TensorFlow operations, providing an easy-to-use interface for defining and training deep learning models, which helps streamline the QIEA workflow.
5. NumPy
   NumPy is used for numerical computations in Python. It is essential for manipulating arrays and matrices, which are the foundational data structures in deep learning. NumPy is used throughout the process, from data pre-processing to the mathematical operations involved in CNN training and evaluation.
6. Optimizers
   In this study, optimization algorithms used for training CNNs Adam and Adadelta. The Adam optimizer is widely employed due to its ability to combine the benefits of both momentum and adaptive learning rates. It dynamically adjusts the learning rate for each parameter based on past gradients, allowing the model to converge faster and more efficiently, which is particularly advantageous for training deep neural networks. Adadelta is chosen for its ability to adapt learning rates based on the previous gradients, reducing the need for manual tuning. By leveraging these optimizers, the training process becomes more efficient, stable, and adaptable, helping achieve faster convergence and improved model performance.
7. Sparse Categorical Cross-Entropy (Loss Function)
   This loss function is used for multi-class classification tasks, where each target class is represented by an integer label. It compares the predicted class probabilities with the true labels and calculates the loss accordingly. This function is ideal for image classification problems like the ones considered in this study.

8. Sparse Categorical Accuracy (Metric)
   This metric is used to evaluate the classification accuracy of the CNN during training and testing. It compares the predicted labels with the actual labels and computes the percentage of correct predictions, providing insight into how well the model is performing.
9. ImageDataGenerator
   ImageDataGenerator is a utility used for real-time data augmentation, which is crucial for improving model generalization. By applying random transformations like rotations, flips, zooms, and shifts to the images during training, the model is exposed to a broader range of data, which helps prevent overfitting and enhances robustness.
10. Batch Normalization and Dropout
    Batch normalization stabilizes training by normalizing the outputs of each layer, reducing internal covariate shift, and accelerating convergence. Dropout is used to prevent overfitting by randomly "dropping" units during training, forcing the network to generalize better and reducing its dependency on any single neuron.
11. Regularization (L2 Regularization)
    L2 regularization, also known as weight decay, adds the sum of squared weights to the loss function. This discourages overly complex models, helping prevent overfitting by encouraging smaller weights and improving generalization, especially with high-dimensional or small datasets.
12. HeNormal Initializer
    The HeNormal initializer is used for layers with ReLU activation. This helps prevent the vanishing gradient problem, allowing deep networks to train more efficiently.