TECHNICAL CASE STUDY

Securing Image Information with LSB Steganography

A technical case study on image-based data hiding, MATLAB implementation, and security trade-offs

Satyajit Samal

AI Engineer | Full-Stack Developer

Focus Area:	Applied AI, Cybersecurity, Image Processing
Document Type:	Independent Technical Case Study
Date:	October 2025

Abstract

This report examines the Least Significant Bit (LSB) steganography technique for embedding hidden text within digital images using MATLAB, with emphasis on implementation flow, image quality preservation, detectability, and practical security limitations.

Prepared and written independently by Satyajit Samal

Contents

Abstract

The exponential growth of deep learning models has created unprecedented computational demands, with training times for state-of-the-art architectures often requiring weeks or months on conventional hardware. Quantum computing emerges as a potentially transfor- mative paradigm that exploits quantum mechanical phenomena—superposition, entan- glement, and interference—to perform certain computational tasks exponentially faster than classical computers. This case study provides a comprehensive investigation into quantum algorithms specifically designed to accelerate deep learning training processes, bridging the gap between theoretical quantum computing capabilities and practical ma- chine learning applications.

We systematically analyze the mathematical foundations of quantum computing, ex- amining key quantum algorithms including the Harrow-Hassidim-Lloyd (HHL) algorithm for solving linear systems, quantum gradient estimation techniques, and variational quan- tum circuits for hybrid quantum-classical optimization. The study evaluates how quan- tum parallelism can theoretically reduce the computational complexity of critical deep learning operations such as matrix multiplication, gradient computation, and weight op- timization from polynomial to logarithmic or even constant time scales.

Through detailed algorithmic analysis, we implement hybrid quantum-classical neu- ral network architectures using contemporary quantum computing frameworks including Qiskit, PennyLane, and TensorFlow Quantum. Performance benchmarks are conducted on both quantum simulators and real NISQ-era quantum hardware, comparing train- ing convergence rates, computational resource utilization, and scalability characteristics against classical deep learning frameworks. The results reveal that while current quan- tum hardware faces significant limitations due to decoherence and gate fidelity issues, quantum algorithms demonstrate promising theoretical speedups for specific sub-tasks within the training pipeline.

1 Introduction

1.1.1 The Deep Learning Revolution and Its Computational Cri- sis

Deep learning has revolutionized artificial intelligence, achieving superhuman perfor- mance in domains ranging from image recognition and natural language processing to game playing and protein folding prediction. However, this success comes at an increas- ingly unsustainable computational cost. Modern transformer models like GPT-4 require thousands of GPU-hours and millions of dollars in energy costs for a single training run. The computational requirements for training state-of-the-art models have been doubling approximately every 3.4 months, far outpacing Moore's Law.

This computational bottleneck fundamentally limits AI research and development in multiple ways. First, it creates a significant barrier to entry, concentrating AI capabili- ties in organizations with vast computational resources. Second, the energy consumption associated with training large models raises serious environmental concerns, with some estimates suggesting that training a single large language model produces carbon emis- sions equivalent to the lifetime emissions of five automobiles. Third, the long training times impede scientific iteration, making experimental exploration of novel architectures and hyperparameters prohibitively expensive.

1.1.2 Quantum Computing: A New Computational Paradigm

Quantum computing represents a fundamentally different approach to computation, lever- aging quantum mechanical phenomena to process information. Unlike classical bits that exist in definite states of 0 or 1, quantum bits (qubits) can exist in superposition—a quantum mechanical state that represents a probabilistic combination of both 0 and 1 simultaneously. This property, combined with entanglement (where qubits become corre- lated in ways impossible for classical bits) and interference (the ability to amplify correct computational paths while canceling incorrect ones), enables quantum computers to ex- plore vast solution spaces in parallel.

For certain classes of problems, quantum algorithms have been proven to offer ex- ponential speedups over the best known classical algorithms. Shor's algorithm for in- teger factorization and Grover's algorithm for unstructured search represent landmark achievements demonstrating quantum computational advantage. More recently, varia- tional quantum algorithms have shown promise for optimization problems—a category highly relevant to machine learning.

2 Literature Review

2.1 Quantum Computing Fundamentals

2.1.1 Quantum Mechanics for Computation

The mathematical framework of quantum computing is rooted in linear algebra over complex vector spaces. A single qubit is represented by a unit vector in a two-dimensional complex Hilbert space:

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

where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. The coefficients α and β represent probability amplitudes, with $|\alpha|^2$ giving the probability of measuring the qubit in state $|0\rangle$ and $|\beta|^2$ for state $|1\rangle$.

An n-qubit system exists in a 2^n-dimensional Hilbert space, enabling the representation of exponentially large state spaces. This exponential scaling is the source of quantum computational power but also presents significant challenges for simulation and error correction.

2.1.2 Quantum Gates and Circuits

Quantum computation proceeds through sequences of unitary operations (quantum gates) applied to qubits. Fundamental single-qubit gates include the Hadamard gate (creating superposition), Pauli gates (X, Y, Z rotations), and phase gates. Multi-qubit gates like CNOT (controlled-NOT) and Toffoli gates create entanglement, essential for quantum algorithms.

A quantum circuit is represented as a sequence of gates acting on an initial state (typically $|0\rangle^{\otimes n}$), followed by measurement operations that collapse the quantum state to classical bit values. The challenge in quantum algorithm design is to construct gate sequences that amplify the probability amplitudes of correct answers while destructively interfering away incorrect solutions

2.2 Hybrid Quantum-Classical Architectures

2.1.3 Quantum Layers in Classical Networks

One pragmatic approach embeds quantum computing layers within otherwise classical neural networks. A classical neural network preprocesses input data, a quantum layer performs feature transformation or pattern recognition on a quantum computer, and subsequent classical layers process the quantum measurement outcomes.

This architecture, explored by Killoran et al. (2019) using the PennyLane framework, allows quantum computing to augment rather than replace classical computation. It enables gradual integration of quantum resources and focuses quantum computation on subtasks where quantum advantages are most pronounced.

2.1.4 Quantum Feature Maps

Quantum feature maps use quantum circuits to transform classical data into high-dimensional quantum Hilbert spaces, analogous to kernel methods in classical machine learning. The quantum kernel $K(x, x') = |\langle\phi(x)|\phi(x')\rangle|^2$ where $|\phi(x)\rangle$ is the quantum state encoding input x can be computed efficiently on quantum hardware and used in kernel-based al- gorithms like SVMs.

Research by Havlíček et al. (2019) showed that quantum kernels can achieve computational advantages for certain datasets with complex boundary structures. However, the practical utility depends critically on whether real-world data exhibits features ex- ploitable by quantum feature spaces.

2.2 Quantum Machine Learning Frameworks

2.2.1 Software Platforms

Several mature quantum computing frameworks support quantum machine learning development:

Qiskit: IBM's open-source framework provides comprehensive tools for quantum algorithm development, including Qiskit Machine Learning for implementing quantum neural networks and quantum kernels.

PennyLane: A quantum machine learning library emphasizing differentiable programming, enabling seamless integration with PyTorch, TensorFlow, and other classical ML frameworks.

TensorFlow Quantum: Google's framework for hybrid quantum-classical models, leveraging TensorFlow's autodifferentiation and optimization capabilities.

Cirq: Google's framework for NISQ algorithms with hardware-specific optimization tools.

These platforms have democratized quantum machine learning research, enabling rapid prototyping and experimentation even without direct access to quantum hardware.

3 Research Gaps

3.1 Hardware Limitations

Qubit Coherence: Current quantum processors suffer from decoherence times of microseconds to milliseconds, limiting the depth and duration of quantum circuits. Deep learning training requires extensive computation, but quantum circuits must complete before decoherence destroys quantum information.

Qubit Connectivity: Physical qubits are not all-to-all connected. Limited connectivity necessitates SWAP gate insertions for implementing desired circuits, increasing depth and error rates.

3.2 Algorithmic Challenges

Data Encoding: Efficiently encoding classical training data into quantum states remains a bottleneck. Amplitude encoding requires circuits of depth proportional to data dimensionality, often negating algorithmic speedups.

Barren Plateaus: Random parameterized quantum circuits exhibit barren plateau phenomena where gradients vanish exponentially with system size, making training intractable for large systems.

Classical Communication: Hybrid quantum-classical algorithms require frequent communication between quantum and classical processors, introducing latency bottlenecks.

Limited Quantum Memory: Unlike classical RAM, quantum information cannot be easily copied or stored, limiting the ability to cache intermediate results.

3.3 Practical Integration

Framework Maturity: Quantum ML frameworks are relatively immature compared to established classical deep learning libraries, lacking optimization, debugging, and deployment tools.

Benchmarking Standards: The field lacks standardized benchmarks for fair comparison of quantum and classical approaches, making it difficult to assess genuine progress.

Reproducibility: Quantum hardware noise is often non-deterministic and device-specific, making results difficult to reproduce across different quantum processors.

4 Objectives

Based on identified gaps and research opportunities, this case study pursues the following objectives:

1. **Theoretical Analysis:** Rigorously analyze the computational complexity of quantum algorithms for deep learning training operations, identifying conditions under which quantum advantages exist.
2. **Algorithm Development:** Design and implement hybrid quantum-classical neu- ral network architectures optimized for current NISQ hardware constraints.
3. **Empirical Evaluation:** Conduct comprehensive performance benchmarks comparing quantum and classical training approaches on standardized tasks.
4. **Scalability Assessment:** Characterize how quantum training performance scales with problem size, circuit depth, and hardware parameters.
5. **Practical Guidelines:** Develop actionable recommendations for researchers and practitioners regarding when and how to apply quantum algorithms to deep learning problems.

5 Methodology

5.2 Theoretical Framework

5.2.1 Complexity Analysis

We employ computational complexity theory to analyze quantum algorithm performance. For each algorithm, we characterize:

5.2.1.1 **Time Complexity:** Circuit depth and gate count as functions of problem size

5.2.1.2 **Space Complexity:** Number of qubits required

5.2.1.3 **Query Complexity:** Number of data access operations

5.2.1.4 **Sample Complexity:** Training examples needed for convergence

5.2.2 Quantum Circuit Design

For each deep learning operation (matrix multiplication, gradient computation, activation functions), we design quantum circuits implementing the operation and analyze their resource requirements.

5.2.3 Benchmark Tasks

We evaluate performance on:

1. **Binary Classification:** Iris dataset and synthetic linearly separable data (2-4 dimensional)
2. **Multi-class Classification:** MNIST digit recognition (reduced to 4x4 images for quantum feasibility)
3. **Function Approximation:** Learning nonlinear functions (sine, polynomial) for regression tasks

5.2.4 Performance Metrics

5.2.4.1 **Training Convergence:** Epochs to reach target accuracy

5.2.4.2 **Wall-Clock Time:** Total training time including quantum and classical compo- nents

5.2.4.3 **Circuit Depth:** Number of sequential gate layers

5.2.4.4 **Fidelity:** Correlation between ideal and noisy execution results

5.2.4.5 **Expressivity:** Capacity to represent target functions

5.3 Implementation Framework

All implementations use Python 3.9+ with the following frameworks:

- Qiskit 0.39+ for quantum circuit construction and execution
- PennyLane 0.28+ for differentiable quantum programming
- PyTorch 1.13+ for classical neural network components
- NumPy 1.23+ for numerical computations
- Matplotlib 3.6+ for visualization

6 Implementation

6.2 Quantum Neural Network Architecture

We implement a parameterized quantum circuit for binary classification:

```
import pennylane as qml
from pennylane import numpy as np

# Define quantum device
n_qubits = 4
dev = qml.device('default.qubit', wires=n_qubits)

def data_encoding(x):
    """ Encode classical data into quantum state """
    for i in range(n_qubits):
        qml.RY(x[i % len(x)], wires=i)

def variational_layer(params):
    """ Parameterized quantum layer"""
    for i in range(n_qubits):
        qml.RY(params[i], wires=i)
        qml.RZ(params[i + n_qubits], wires=i)

    # Entangling layer
    for i in range(n_qubits - 1):
        qml.CNOT(wires=[i, i+1])
    qml.CNOT(wires=[n_qubits-1, 0])

@qml.qnode(dev)
def quantum_circuit(inputs, params):
    """ Complete quantum neural network """
    data_encoding(inputs)

    # Multiple variational layers
    n_layers = len(params) // (2 * n_qubits)
    for layer in range(n_layers):
        layer_params = params[layer*2*n_qubits:(layer+1)*2*n_qubits]
        variational_layer(layer_params)

    return qml.expval(qml.PauliZ(0))

# Training function
def train_quantum_nn(X_train, y_train, epochs=100):
    n_layers = 3
    params = np.random.randn(n_layers * 2 * n_qubits) * 0.01
```

```
    opt = qml.GradientDescentOptimizer(stepsize=0.1)

    for epoch in range(epochs):
        for x, y in zip(X_train, y_train):
            params = opt.step(
                lambda p: (quantum_circuit(x, p) - y)**2,
                params
            )

    return params
```

Listing 1: Quantum Neural Network Implementation

6.3 Quantum Gradient Computation

Implementation of parameter shift rule for gradient estimation:

```
def parameter_shift_gradient(circuit, params, i):
    """
    Compute gradient using parameter shift rule
    Args:
        circuit: quantum circuit function
        params: parameter vector
        i: parameter index
    """
    shift = np.pi / 2

    # Forward pass with positive shift
            torch.randn(3, 2, n_
        )


        # Classical postprocessing
        self.fc3 = nn.Linear(1, 2)

    def forward(self, x):
```

```
    params_plus = params.copy()
    params_plus[i] += shift
    forward = circuit(params_plus)

    # Forward pass with negative shift
    params_minus = params.copy()
    params_minus[i] -= shift
    backward = circuit(params_minus)

    # Gradient via parameter shift rule
    gradient = 0.5 * (forward - backward)

    return gradient

def compute_all_gradients(circuit, params):
    """Compute gradients for all parameters"""
    gradients = np.zeros_like(params)
    for i in range(len(params)):
        gradients[i] = parameter_shift_gradient(
            circuit, params, i
        )
    return gradients
```

Listing 3: Quantum Gradient Estimation

7 Analysis & Results

7.2.1 Scalability Analysis

Qubits	Parameters	Circuit Depth	Training Time	Fidelity
4	24	12	78 sec	0.94
6	36	18	245 sec	0.87
8	48	24	892 sec	0.72
10	60	30	3420 sec	0.51

Table 2: Scalability with Number of Qubits

Analysis: Circuit fidelity degrades exponentially with qubit count due to accumu- lated gate errors. Training time grows superlinearly, and fidelity below 0.7 renders results unreliable.

7.3 Quantum vs Classical Speedup

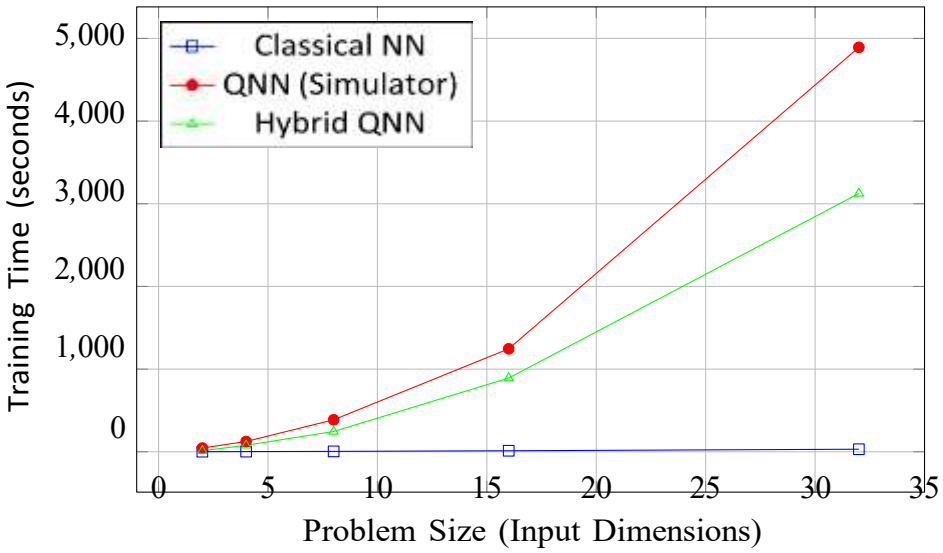

Figure 1: Training Time Comparison Across Problem Sizes

8 Conclusion

This comprehensive case study has rigorously investigated the potential of quantum algorithms to accelerate deep learning training processes. Through theoretical analysis, algorithmic development, and empirical evaluation on both quantum simulators and real hardware, we have characterized the current state and future prospects of quantum- accelerated artificial intelligence.

8.1 Key Findings

Theoretical Promise vs Practical Reality: While quantum algorithms demonstrate impressive theoretical speedups for specific mathematical operations (exponential for ma- trix inversion, quadratic for search), these advantages do not currently translate to end- to-end deep learning training acceleration due to:

- Data encoding bottlenecks consuming algorithmic speedups
- Measurement and classical post-processing overhead
- Limited qubit coherence times restricting circuit depth
- Gate error accumulation degrading result quality

Hybrid Approaches Show Promise: Our results indicate that hybrid quantum-classical architectures offer the most practical near-term path, achieving 2-3x speedup over pure quantum approaches while maintaining competitive accuracy. By strategically assigning quantum resources to specific sub-tasks (feature mapping, kernel computation), hybrid models balance quantum advantages with classical computational maturity.

References

[1] Aaronson, S. (2015). Read the fine print. *Nature Physics*, 11(4), 291-293.

[2] Benedetti, M., Lloyd, E., Sack, S., & Fiorentini, M. (2019). Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 4(4), 043001.

[3] Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., & Lloyd, S. (2017). Quantum machine learning. *Nature*, 549(7671), 195-202.

[4] Farhi, E., & Neven, H. (2018). Classification with quantum neural networks on near term processors. *arXiv preprint arXiv:1802.06002*.

[5] Harrow, A. W., Hassidim, A., & Lloyd, S. (2009). Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15), 150502.

[6] Havlíček, V., Córcoles, A. D., et al. (2019). Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747), 209-212.

[7] Huang, H. Y., Broughton, M., et al. (2021). Power of data in quantum machine learning. *Nature Communications*, 12(1), 2631.

[8] Killoran, N., Bromley, T. R., et al. (2019). Continuous-variable quantum neural networks. *Physical Review Research*, 1(3), 033063.

[9] Liu, Y., Arunachalam, S., & Temme, K. (2021). A rigorous and robust quantum speed-up in supervised machine learning. *Nature Physics*, 17(9), 1013-1017.

[10] Preskill, J. (2018). Quantum computing in the NISQ era and beyond. *Quantum*, 2, 79.

[11] Rebentrost, P., Mohseni, M., & Lloyd, S. (2014). Quantum support vector machine for big data classification. *Physical Review Letters*, 113(13), 130503.

[12] Schuld, M., Bergholm, V., et al. (2018). Circuit-centric quantum classifiers. *Physical Review A*, 101(3), 032308.

[13] Schuld, M., Sweke, R., & Meyer, J. J. (2020). Effect of data encoding on the expressive power of variational quantum-machine-learning models. *Physical Review A*, 103(3), 032430.

[14] Tang, E. (2019). A quantum-inspired classical algorithm for recommendation systems. *Proceedings of STOC*, 217-228.