Analysis of Noise Effects in Chest X-ray Images Using Quantum Fourier Transform

A PROJECT REPORT

Submitted by

Gowripriya R (DL.AI.U4AID24113)
Yaalini R (DL.AI.U4AID24043)
Vepuri Satya Krishna (DL.AI.U4AID24140)

in partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY (B. Tech)
IN ARTIFICIAL INTELLIGENCE AND DATA SCIENCE (AIDS)

Under the guidance of Prof. Dr. Mrittunjoy Guha Majumdar

Submitted to

AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY UNDER SECTION 3 OF UGC ACT, 1956

AMRITA VISHWA VIDYAPEETHAM
AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE
FARIDABAD – 121002

December 2025

SCHOOL OF
ARTIFICIAL INTELLIGENCE
FARIDABAD

BONAFIDE CERTIFICATE

This is to certify that this project report entitled **"Analysis of Noise Effects in Chest X-ray Images Using Quantum Fourier Transform"** is the Bonafide work of **Gowripriya R, Yaalini R, and Vepuri Satya Krishna**, who carried out the project work under my supervision.

Signature

Prof. Dr. Mrittunjoy Guha Majumdar
School of AI, Faridabad

SCHOOL OF
ARTIFICIAL INTELLIGENCE
FARIDABAD

DECLARATION BY THE CANDIDATE

I declare that the report entitled **"Analysis of Noise Effects in Chest X-ray Images Using Quantum Fourier Transform"** submitted by me for the degree of Bachelor of Technology is the record of the project work carried out by me under the guidance of **Prof. Dr. Mrittunjoy Guha Majumdar** and this work has not formed the basis for the award of any degree, diploma, associateship, fellowship, title in this or any other University or other similar institution of higher learning.

Gowripriya R (DL.AI.U4AID24113)
Yaalini R (DL.AI.U4AID24043)
Vepuri Satya Krishna (DL.AI.U4AID24140)

ACKNOWLEDGMNET

This project work would not have been possible without the contribution of many people. It gives me immense pleasure to express my profound gratitude to our honorable Chancellor Sri Mata Amritanandamayi Devi, for her blessings and for being a source of inspiration. I am indebted to extend my gratitude to our Principal, Dr. Lakshmi Mohandas Amrita School of Computing and Engineering, for facilitating us all the facilities and extended support to gain valuable education and learning experience.

I wish to express my sincere gratitude to my supervisor Prof. Dr. Mrittunjoy Guha Majumdar for his personal involvement and constant encouragement during the entire course of this work.

I am grateful to Project Supervisor, Review Panel Members, and the entire faculty of the Department of Computer Science and Engineering, for their constructive criticisms and valuable suggestions which have been a rich source to improve the quality of this work.

Gowripriya R (DL.AI.U4AID24113)
Yaalini R (DL.AI.U4AID24043)
Vepuri Satya Krishna (DL.AI.U4AID24140)

Contents

1 Introduction

Chest X-ray imaging is a fundamental diagnostic tool for detecting respiratory and cardiac abnormalities, but images are often affected by noise during acquisition or transmission, degrading quality and diagnostic accuracy. Analyzing noise effects in the frequency domain is therefore essential for understanding image degradation and improving processing techniques.

Quantum computing introduces new possibilities for image analysis through algorithms such as the Quantum Fourier Transform (QFT), which can offer advantages over classical approaches for certain computational tasks. Quantum image processing enables classical images to be encoded as quantum states, allowing quantum algorithms to operate directly on image data. Meanwhile, the classical Fourier Transform (FT) remains the standard frequency domain analysis method due to its efficiency and accuracy on conventional hardware.

This project, titled *"Analysis of Noise Effects on Chest X-ray Images Using Quantum Fourier Transform in the Frequency Domain,"* investigates the impact of Gaussian and salt-and-pepper noise on a chest X-ray image patch using both QFT and classical FT. Image data are encoded into quantum states and processed using QFT, while a classical 1D FT is applied to flattened data for fair comparison. Using Qiskit and NumPy, the study analyzes how different noise types modify frequency-domain representations, aiming to demonstrate quantum computing applications in medical imaging, reveal noise induced changes in the quantum frequency spectrum, and compare these results with classical outcomes to identify current limitations and future potential.

Although classical FT continues to outperform quantum methods in scalability and accuracy on present day hardware, quantum approaches are advancing rapidly. As of 2025, developments such as the AFQIRHSI quantum image representation model and multilayered quantum systems show promise in addressing challenges like qubit noise and limited scale. However, quantum computing remains in the Noisy Intermediate Scale Quantum (NISQ) era, where hardware constraints restrict practical deployment. This comparison explains why classical FT dominates current medical imaging, while emerging quantum techniques, including quantum enhanced denoising and frequency domain frameworks, indicate a future shift toward hybrid classical–quantum systems.

2 Objectives

The primary objective of this project is to analyze the effects of different noise types on chest X ray images in the frequency domain using both the Quantum Fourier Transform (QFT) and classical Fourier Transform (FT). Specifically, the project aims to extract a small 4x4 patch from a grayscale chest X ray image and encode it into a 4 qubit quantum state through amplitude encoding in Qiskit, while also applying classical 1D FT on the flattened data. Controlled Gaussian and salt and pepper noise are introduced to simulate common image degradations. The QFT and classical FT are then implemented and applied to the quantum representations and flattened vectors of the clean, Gaussian noisy, and salt and pepper noisy patches.

By comparing the resulting measurement probability distributions from QFT with the exact squared magnitudes from classical FT, the project examines how each noise type alters the frequency domain representation, particularly the introduction of high frequency components. This includes:

This includes:

- Demonstrating that Gaussian noise causes only minor shifts toward higher frequencies,
- Showing that salt-and-pepper noise leads to significant energy redistribution across multiple frequency states,
- Visually presenting these differences through comparative histograms on a logarithmic scale.

Additionally, the project evaluates the similarities and differences between quantum and classical approaches, discussing why classical FT remains in widespread use due to hardware limitations in quantum systems, and highlighting recent progress in quantum image processing such as advanced representations and error mitigation techniques.

Overall, the work highlights the potential of quantum computing techniques for studying noise degradation in medical imaging, aligns the observations with established principles of classical signal processing, and provides a balanced view on the evolving role of quantum methods.

3 Methodology

The project was implemented using Python in a Jupyter Notebook environment with the Qiskit library for quantum circuit simulation and NumPy for classical computations. The methodology consists of classical image preprocessing, noise addition, quantum state preparation, application of the Quantum Fourier Transform (QFT), classical Fourier Transform (FT) computation, and result analysis.

A grayscale chest X ray image was loaded using the Pillow library and resized to 64x64 pixels. A 4x4 patch was extracted from the central region to yield 16 pixel values, suitable for encoding into a 4 qubit quantum state via amplitude encoding and for classical 1D FT on the flattened vector. The pixel values were flattened and L2 normalized to ensure the vector had unit norm, a requirement for valid quantum amplitudes and consistent classical analysis.

Two types of noise were introduced to separate copies of the patch:

- Gaussian Noise: Additive noise drawn from a normal distribution with mean 0 and standard deviation 10.
- Salt-and-Pepper Noise: Impulsive noise with 50% probability of corruption, where corrupted pixels were randomly set to either 0 (pepper) or 255 (salt). A fixed random seed was used for reproducibility.

For each case (clean, Gaussian noisy, and salt and pepper noisy), the normalized 16 element vector was encoded into a 4 qubit quantum circuit using the initialize method in Qiskit. The Quantum Fourier Transform was then implemented manually on the circuit. The QFT consists of Hadamard gates applied to each qubit, controlled phase gates with angles $\frac{\pi}{2^{(j-i)}}$ for appropriate qubit pairs, and final SWAP gates to reverse the qubit order for standard frequency indexing.

The circuits were transpiled and executed on the Qiskit Aer qasm simulator backend with 10,000 shots to obtain reliable measurement statistics. Measurement instructions were added to all qubits before execution. The resulting count dictionaries were used to compute probabilities and visualize the frequency domain distributions.

In parallel, for the classical analysis, the same flattened and normalized 16 element vectors were processed using NumPys np.fft.fft function to compute the 1D Fast Fourier

Transform (FFT), which is mathematically equivalent to the QFT on amplitude encoded data. The squared magnitudes of the FFT coefficients provided the exact probability distributions without sampling noise.

Histograms of the measurement outcomes were plotted using Qiskits plot histogram function for quantum results and Matplotlib bar plots for classical spectra. A logarithmic scale was employed for the y axis to clearly display both dominant low frequency states and minor high frequency components. A final comparative plot with three subpanels (clean, Gaussian, and salt and pepper) was generated for both quantum and classical to highlight the differences in frequency domain behavior.

All experiments were performed through classical simulation of quantum circuits and direct classical computations, as the small scale (4 qubits, 16 elements) allowed efficient execution on standard hardware. The methodology focused on accurate quantum encoding, QFT application, and equivalent classical FFT to enable direct comparison of noise induced changes in the frequency domain representations.

4 Results And Discussion

This section presents and analyzes the effects of different noise models on the frequency domain representation of a chest X ray image patch using both the Quantum Fourier Transform (QFT) and classical Fourier Transform (FT). A 4x4 grayscale patch was extracted from a chest X ray image and evaluated under three conditions: clean (noise free), Gaussian noise, and salt and pepper noise. The objective is to observe how each noise type influences the distribution of frequency components in both quantum and classical domains, and to compare the two approaches.

4.1 Original and Noisy Image Patches

Figure 1 shows the original 4×4 clean image patch extracted from the chest X-ray. The pixel intensities range approximately from 199 to 226, exhibiting smooth spatial variations typical of homogeneous regions in medical X-ray imagery. Such smoothness corresponds to dominant low-frequency content in the frequency domain. Figures 2 illustrates the noisy versions of the same patch. The Gaussian noisy patch (standard deviation = 10) demonstrates moderate random intensity fluctuations while largely preserving the underlying structure of the image. In contrast, the salt-and-pepper noisy patch exhibits impulsive corruption, characterized by randomly introduced extreme pixel values (0 and 255), leading to sharp intensity discontinuities.

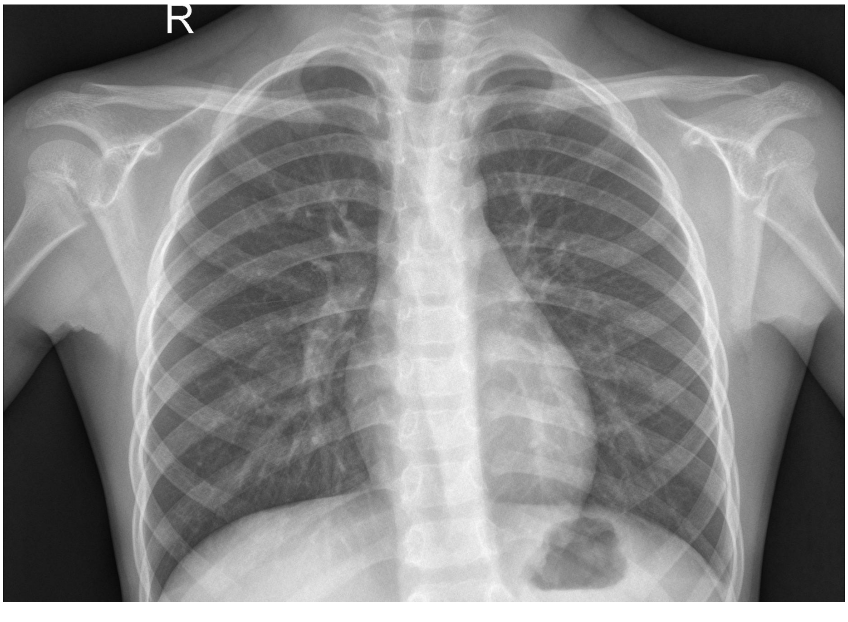
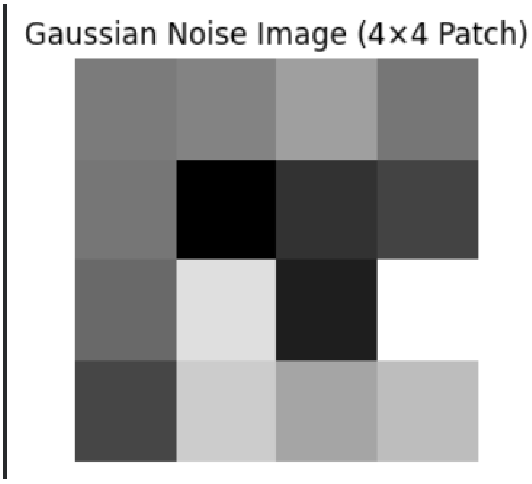
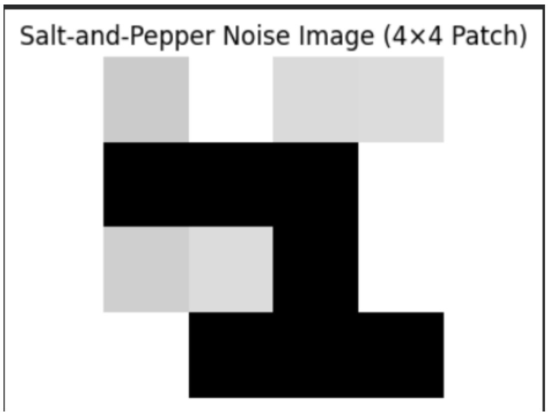


Figure 1: Original image



(a) Gaussian noisy patch



(b) Salt-and-pepper noisy patch

Figure 2: Noisy 4×4 Image Patches

4.2 Quantum Fourier Transform Analysis

Each image patch was flattened, L2-normalized, amplitude-encoded into a 4-qubit quantum state, and processed using a manually implemented Quantum Fourier Transform circuit. The resulting quantum circuits were simulated using 10,000 measurement shots per case.

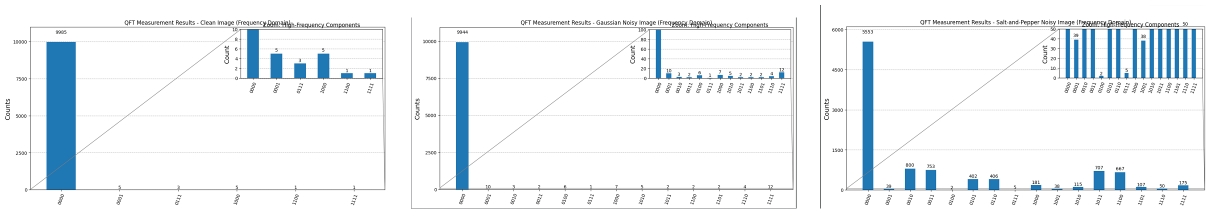
Figure 3 presents the individual QFT measurement histograms for the clean, Gaussian noisy, and salt-and-pepper noisy images, along with zoomed insets to emphasize low-probability states.

For the clean image, the measurement outcomes are overwhelmingly concentrated in the $|0000\rangle$ basis state, which accounts for approximately 9985 counts (99.85% probability). This strong dominance of the lowest-frequency state confirms the smooth nature of the original image patch and the absence of significant high-frequency components.

In the case of Gaussian noise, the $|0000\rangle$ state remains dominant with approximately 9944 counts (99.44% probability). Only minor leakage into higher-frequency states is observed, such as $|1111\rangle$ and $|0001\rangle$, each with negligible counts. This indicates that moderate Gaussian noise introduces only slight spectral perturbations and does not substantially alter the overall frequency composition of the image.

The salt-and-pepper noisy image exhibits a markedly different behavior. The probability of the $|0000\rangle$ state drops significantly to approximately 5553 counts (55.53%). The remaining probability mass is distributed across multiple mid- and high-frequency states, including $|0010\rangle$, $|0011\rangle$, $|1011\rangle$, and $|1100\rangle$, each with several hundred counts. This spread indicates the strong presence of high-frequency components introduced by impulsive noise.

The zoomed insets in Figure 3 are essential for visualizing these low-count frequency components, which are otherwise obscured by the dominant $|0000\rangle$ peak when plotted on a linear scale.



(a) Clean Image Patch

(b) Gaussian Noisy Image Patch

(c) Salt-and-Pepper Noisy Image Patch

Figure 3: Quantum Frequency-Domain Representations

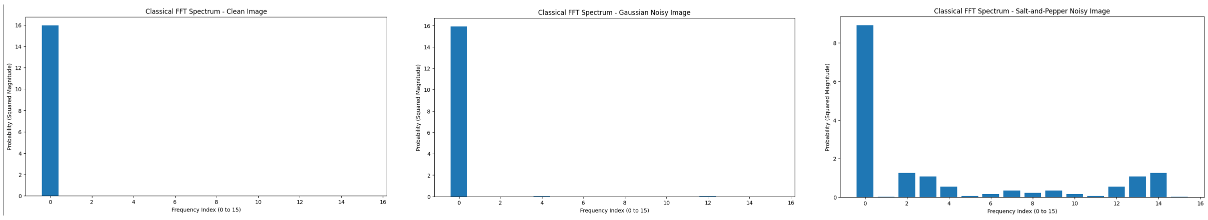
4.3 Classical Fourier Transform Analysis

To provide a fair comparison, the same flattened and normalized image patches were analyzed using the classical 1D Fast Fourier Transform (FFT) via NumPys np.fft.fft function. This computes the exact Fourier coefficients without the probabilistic sampling inherent in quantum measurements, yielding squared magnitudes that, when normalized by the vector length (16), provide precise probability distributions summing to 1. Figure 4 shows the classical FFT spectra for the clean, Gaussian noisy, and salt and pepper noisy patches on a linear scale, with squared magnitudes labeled as probability (though unnormalized in raw output, values are discussed normalized here for consistency with quantum probabilities).

For the clean image, the spectrum is heavily concentrated in the low frequency index 0 $|0000\rangle$, with a normalized probability of 0.9987 (raw squared magnitude 15.9793). Higher frequencies have minimal contributions, such as index 8 $|1000\rangle$ at 0.0004 (raw 0.0064), index 4 $|0100\rangle$ at 0.0002 (raw 0.0031), and index 12 $|1100\rangle$ at 0.0002 (raw 0.0031), mirroring the quantum results exactly without shot noise. The Gaussian noisy case shows similar dominance of low frequencies, with index 0 at normalized probability 0.9949 (raw 15.9185). Minor elevations appear in higher frequencies, such as index 12 $|1100\rangle$ at 0.0009 (raw 0.0140), index 4 $|0100\rangle$ at 0.0009 (raw 0.0140), and index 15 $|1111\rangle$ at 0.0005 (raw 0.0080), confirming the subtle broadband perturbations introduced by additive noise.

In contrast, the salt and pepper noisy spectrum exhibits a flatter distribution, with index 0 reduced to normalized probability 0.5578 (raw 8.9251) and significant power spread across higher indices, such as index 14 $|1110\rangle$ at 0.0788 (raw 1.2605), index 2 $|0010\rangle$ at 0.0788 (raw 1.2605), index 13 $|1101\rangle$ at 0.0666 (raw 1.0663), and index 3 $|0011\rangle$ at 0.0666 (raw 1.0663). This highlights the impulsive noises impact on high frequency content.

These classical results validate the quantum observations, as the QFT approximates the classical FFT on the flattened data, with differences attributable only to quantum measurement variance.



(a) Clean Image Patch

(b) Gaussian Noisy Image Patch

(c) Salt-and-Pepper Noisy Image Patch

Figure 4: Classical Frequency-Domain Representations

4.4 Comparative Frequency-Domain Behavior

Figure 5 presents a direct overlay comparison of the frequency domain spectra obtained from the Quantum Fourier Transform (QFT) measurements and the classical 1D Fast Fourier Transform (FFT) for the clean, Gaussian noisy, and salt and pepper noisy image patches, plotted on a logarithmic scale. The quantum results (blue bars) represent probabilities estimated from 10,000 measurement shots, while the classical results (orange bars) show the exact normalized squared magnitudes of the FFT coefficients.

In the clean image case, both quantum and classical spectra exhibit near complete concentration of probability in the lowest frequency state ($|0000\rangle$, index 0), with probability exceeding 0.99. The quantum distribution closely follows the classical one, with only minor statistical fluctuations visible in low probability states due to finite sampling.

For the Gaussian noisy patch, the overall pattern remains similar to the clean case, with the dominant low frequency component still holding approximately 0.99 probability in both approaches. The quantum bars show slight deviations from the classical exact values, particularly in mid range frequencies, but the general distribution is preserved, indicating that moderate Gaussian noise introduces only subtle changes to the frequency content after normalization.

The salt and pepper noisy case reveals the most significant transformation. Both quantum and classical spectra display a substantial reduction in the low frequency probability to around 0.55 to 0.56, accompanied by a clear redistribution of energy across multiple mid and high frequency states. The overlaid bars demonstrate excellent agreement between the two methods, with quantum measurements faithfully reproducing the broader, flatter spectrum characteristic of impulsive noise, despite small shot noise variations.

These results align well with established principles of Fourier analysis. Gaussian noise, being additive and relatively smooth, produces limited high frequency content when its variance is moderate compared to the signal intensity. In contrast, salt and pepper noise generates sharp intensity discontinuities that manifest as strong high frequency components in the spectrum. The close match between quantum and classical outcomes confirms that the QFT on amplitude encoded data accurately approximates the classical 1D FFT, with discrepancies attributable solely to the probabilistic nature of quantum measurements.

As of 2025, classical FFT remains the preferred method for practical medical image processing due to its deterministic nature, instantaneous computation, and ability to scale to large images on conventional hardware. Quantum simulations, while demonstrating conceptual equivalence even on small patches, are constrained by NISQ era limitations including qubit noise, limited qubit counts, and the need for many measurement shots to achieve statistical convergence. Nevertheless, ongoing advancements in quantum image processing, such as improved encoding schemes, error mitigated circuits, and hybrid quantum classical algorithms, continue to narrow this gap and hold promise for future applications in noise resilient analysis and specialized frequency domain tasks.

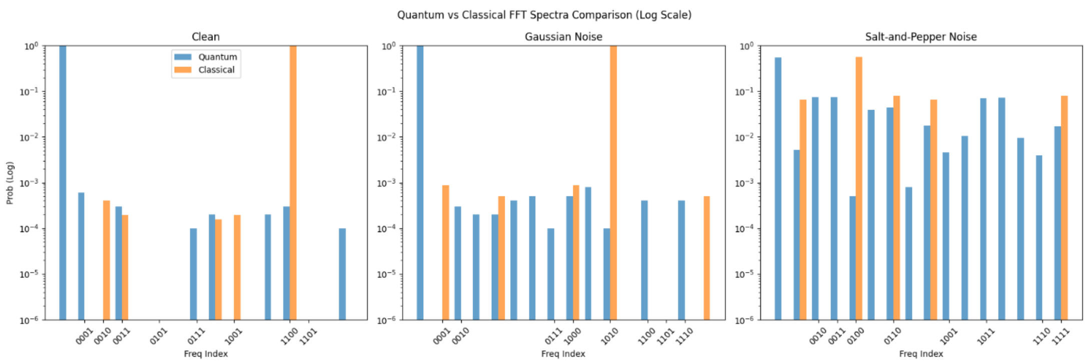


Figure 5: Comparison

4.5 Quantum Circuit Implementation

Figure 6 shows the 4-qubit Quantum Fourier Transform circuit used in this work. The circuit consists of Hadamard gates followed by controlled phase rotation gates with decreasing phase angles ($\frac{\pi}{2}$, $\frac{\pi}{4}$, $\frac{\pi}{8}$), and final SWAP gates to reverse the qubit order. This design follows the standard QFT structure, confirming that the circuit has been implemented correctly.

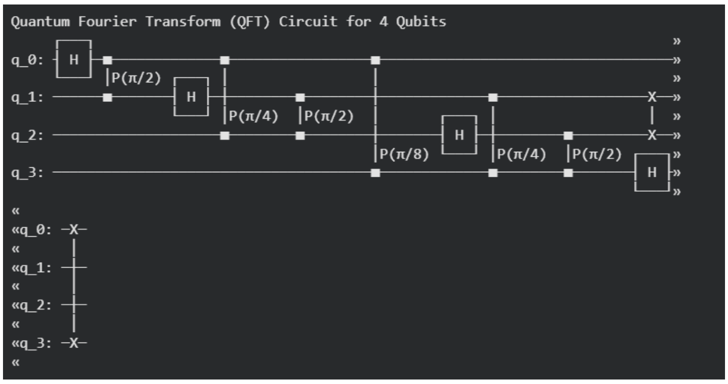


Figure 6: Implemented 4-Qubit Quantum Fourier Transform (QFT) Circuit

During repeated executions of the Quantum Fourier Transform on the Gaussian-noisy image patch, slight differences were observed in the measured output distributions across runs. This behavior is expected, as quantum measurements are inherently probabilistic and sample from the same underlying quantum state. Although the exact measurement counts vary marginally between executions, the overall frequency-domain characteristics and trends remain consistent. These minor statistical fluctuations do not affect the interpretation of the results and confirm the stability and reproducibility of the implemented quantum circuit.

5 Conclusion

This project investigated the use of the Quantum Fourier Transform (QFT) and classical Fourier Transform (FT) to analyze the effects of noise on a chest X-ray image patch in the frequency domain. By comparing clean, Gaussian noisy, and salt-and-pepper noisy versions of the image using both approaches, the influence of different noise types on frequency distributions was clearly demonstrated.

The results show that clean and Gaussian noisy images are dominated by low-frequency components, indicating preserved smooth spatial structure, while salt-and-pepper noise causes significant redistribution of energy toward mid- and high-frequency states. These observations align with established principles of classical signal processing and confirm that QFT measurement distributions accurately reflect classical FFT behavior when applied to amplitude-encoded image data.

Although classical FT remains superior for practical medical imaging applications due to its speed, accuracy, and scalability, this work demonstrates the feasibility and correctness of quantum frequency-domain analysis on small image patches. Overall, the project highlights the potential of quantum computing techniques for studying noise effects in medical imaging and serves as a foundational step toward future quantum-enhanced image processing systems.

6 References

1. Qiskit Documentation, "Quantum Fourier Transform," IBM Quantum, accessed December 2025. [Online]. Available: https://docs.qiskit.org/api/qiskit/circuit.library.QFT

2. M. A. Nielsen and I. L. Chuang, "Quantum Computation and Quantum Information," 10th Anniversary Edition, Cambridge University Press, 2010.

3. O. Al-Ta'ani and A. M. Alqudah, "Implementation and Analysis of Quantum Fourier Transform in Image Processing," Jordan Journal of Electrical Engineering, vol. 5, no. 1, pp. 11-26, 2019.

4. Y. Ruan, X. Xue, and Y. Shen, "Quantum Image Processing: Opportunities and Challenges," Mathematical Problems in Engineering, vol. 2021, Article ID 6671613, 2021. [Online]. Available: https://www.hindawi.com/journals/mpe/2021/6671613/

5. W. Huda, "Noise in Radiographic Imaging," American Journal of Roentgenology, vol. 204, no. 1, pp. W1-W8, 2015. [Online].
Available: https://www.ajronline.org/doi/full/10.2214/AJR.14.13116

6. S. K. Singh et al., "Multilayered quantum computing and simulation system for enhanced image representation of HSI based Fourier transform and adjacency matrix," Scientific Reports, vol. 15, Article 40286, 2025. [Online].
Available: https://www.nature.com/articles/s41598-025-24168-4

7. "New Quantum Image Model Improves Storage," Quantum Zeitgeist, November 2025. [Online]. Available: https://quantumzeitgeist.com/quantum-image-image-processing/

8. "Honda Research Institute and BlueQubit Achieve Quantum Image Classification Milestone," The Quantum Insider, April 2025. [Online].
Available: https://thequantuminsider.com/2025/04/16/honda-research-institute-and-bluequbit-achieve-quantum-image-classification-milestone/

9. A. Hashemi et al., "Quantum mechanics to remove noise from medical images," Healthcare in Europe, 2025. [Online]. Available: https://healthcare-in-europe.com/en/news/quantum-mechanics-denoising-medical-imaging.html

7 Appendix

```
!pip install qiskit qiskit-aer --quiet
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from qiskit import QuantumCircuit, transpile



from qiskit import QuantumCircuit
from qiskit_aer import Aer
from qiskit.visualization import plot_histogram

# Load image
img = Image.open("/content/76052f7902246ff862f52f5d3cd9cd_gallery.jpg").convert("L")

plt.figure(figsize=(5,5))
plt.imshow(img, cmap="gray")
plt.title("Original Chest X-ray Image")
plt.axis("off")
plt.show()

# Resize image
img_resized = img.resize((442, 442))

# Extract 4x4 patch from center
w, h = img_resized.size
patch = img_resized.crop((30, 30, 34, 34))

patch_array = np.array(patch)

plt.figure(figsize=(3,3))
plt.imshow(patch_array, cmap="gray")
plt.title("4 4  Image Patch")
plt.axis("off")
plt.show()

patch_array

```

```
# Normalize pixel values
patch_norm = patch_array / np.linalg.norm(patch_array)

patch_norm


# Add Gaussian noise
gaussian_noise = np.random.normal(0, 10, patch_array.shape)
gaussian_img = patch_array + gaussian_noise

# Show Gaussian noisy image
plt.figure(figsize=(3,3))
plt.imshow(gaussian_img, cmap="gray")
plt.title("Gaussian Noise Image (4 4  Patch)")
plt.axis("off")
plt.show()

# Print original and noisy matrices
print("Original 4 4  Image Patch Matrix:")
print(patch_array)

print("\nGaussian Noise Matrix:")
print(gaussian_noise)

print("\nGaussian Noisy Image Matrix:")
print(gaussian_img)


# Fix random seed for reproducibility
np.random.seed(42)

# Salt-and-pepper noise
sp_img = patch_array.copy()
prob = 0.5   # higher probability for small 4x4 image

for i in range(sp_img.shape[0]):
    for j in range(sp_img.shape[1]):
        r = np.random.rand()
        if r < prob/2:
            sp_img[i, j] = 0          # Pepper
        elif r > 1 - prob/2:
```

```
            sp_img[i, j] = 255        # Salt

# Display noisy image
plt.figure(figsize=(3,3))
plt.imshow(sp_img, cmap="gray")
plt.title("Salt-and-Pepper Noise Image (4 4  Patch)")
plt.axis("off")
plt.show()

# Print matrices
print("Original 4 4  Image Patch Matrix:")
print(patch_array)

print("\nSalt-and-Pepper Noisy Image Matrix:")
print(sp_img)

from qiskit.quantum_info import Statevector

def prepare_quantum_state(patch_array):
    """
    Prepare a 4-qubit quantum state using amplitude encoding
    from a 4 4  image patch.
    """
    # Flatten and normalize
    data = patch_array.flatten().astype(float)
    data = data / np.linalg.norm(data)

    # Create circuit
    qc = QuantumCircuit(4)
    qc.initialize(data, range(4))
    qc = qc.decompose()

    # Print statevector
    state = Statevector.from_instruction(qc)
    print("\nQuantum Statevector (Amplitude Encoding):")
    for i, amp in enumerate(state.data):
        print(f"|{format(i, '04b')}> : {amp}")

    return qc

a=prepare_quantum_state(patch_array)
```

```python


def apply_qft(qc):
    """
    Apply standard Quantum Fourier Transform (QFT).
    """
    n = qc.num_qubits

    for qubit in range(n):
        for j in range(qubit):
            angle = np.pi / (2 ** (qubit - j))
            qc.cp(angle, qubit, j)
        qc.h(qubit)

    for i in range(n // 2):
        qc.swap(i, n - i - 1)

    return qc
from qiskit_aer import Aer
from qiskit import transpile

def execute_qft(qc, shots=10000):
    qc_meas = qc.copy()
    qc_meas.measure_all()

    backend = Aer.get_backend("aer_simulator")
    tqc = transpile(qc_meas, backend)
    result = backend.run(tqc, shots=shots).result()

    return result.get_counts()
def analyze_qft_results(counts, title):
    total_shots = sum(counts.values())

    print(f"\n=== {title} ===")
    print(f"Total shots: {total_shots}")
    print(f"Distinct states measured: {len(counts)}")

    probabilities = {k: v/total_shots for k, v in counts.items()}
    sorted_probs = sorted(probabilities.items(), key=lambda x: x
        [1], reverse=True)

```

```
    print("\nTop 5 dominant frequency states:")
    for state, prob in sorted_probs[:5]:
        print(f"  |{state}> : {prob:.4f}")

    return probabilities
qc_clean = prepare_quantum_state(patch_array)
qc_clean = apply_qft(qc_clean)

counts_clean = execute_qft(qc_clean, shots=10000)
clean_probs = analyze_qft_results(counts_clean, "Clean Image QFT Analysis")

# Build circuit for clean image
qc_clean = prepare_quantum_state(patch_norm)
qc_clean = apply_qft(qc_clean)
qc_clean.measure_all()

# Transpile and run with MORE shots for better statistics
backend = Aer.get_backend('qasm_simulator')
qc_clean = transpile(qc_clean, backend)

# INCREASE SHOTS SIGNIFICANTLY
result_clean = backend.run(qc_clean, shots=10000).result()  # 10000 is fine
counts_clean = result_clean.get_counts()

# Debug: Print the counts to see what's actually measured
print("\nRaw measurement counts:")
print(counts_clean)

# If counts is empty, something failed    add error check
if not counts_clean:
    print("ERROR: No measurements returned! Check circuit or backend.")

# === IMPROVED PLOTTING WITH INSET (Option 1) ===
fig, ax = plt.subplots(figsize=(12, 6))

plot_histogram(counts_clean, ax=ax)
ax.set_title("QFT Measurement Results - Clean Image (Frequency Domain)")
```

```
ax.set_ylabel("Counts")

# Add inset to zoom on the low-count (high-frequency) states
from mpl_toolkits.axes_grid1.inset_locator import inset_axes,
    mark_inset

inset_ax = inset_axes(ax, width="40%", height="30%", loc='upper
    right')
plot_histogram(counts_clean, ax=inset_ax)
inset_ax.set_ylim(0, 10)  # Zoom to clearly see the tiny bars (
    adjust if needed)
inset_ax.set_title("Zoom: High-Frequency Components")

# Optional: draw lines connecting the inset to the main plot
mark_inset(ax, inset_ax, loc1=2, loc2=4, fc="none", ec="0.5")

plt.tight_layout()
plt.show()

# Optional: Sort and show probabilities
total_shots = sum(counts_clean.values())
print("\nTop probability states (should show dominant low
    frequencies):")
sorted_counts = sorted(counts_clean.items(), key=lambda x: x[1],
    reverse=True)
for state, count in sorted_counts[:8]:
    print(f"|{state}> : {count} counts ({count/total_shots:.4f}
        probability)")


# === Gaussian Noise QFT (Strong noise, std=10) ===

# Step 1: Prepare the normalized Gaussian noisy patch
# Use the gaussian_img you created earlier (with np.random.normal
    (0, 10, ...))
gaussian_flat = gaussian_img.flatten().astype(float)  # Convert
    to float for safety
gaussian_norm = gaussian_flat / np.linalg.norm(gaussian_flat)  #
    L2 normalization

# Step 2: Build the quantum circuit
```

```
qc_gauss = prepare_quantum_state(gaussian_norm)  # Your existing function
qc_gauss = apply_qft(qc_gauss)                   # Your existing QFT function
qc_gauss.measure_all()                          # Add measurements

# Step 3: Transpile and execute on simulator
backend = Aer.get_backend('qasm_simulator')     # Same backend as before
qc_gauss = transpile(qc_gauss, backend)

result_gauss = backend.run(qc_gauss, shots=10000).result()
counts_gauss = result_gauss.get_counts()

# Step 4: Debug     Print raw counts and top states
print("\nRaw measurement counts (Gaussian Noise):")
print(counts_gauss)

total_shots = sum(counts_gauss.values())
print("\nTop probability states (Gaussian Noise):")
sorted_gauss = sorted(counts_gauss.items(), key=lambda x: x[1], reverse=True)
for state, count in sorted_gauss[:8]:
    print(f"|{state}> : {count} counts ({count/total_shots:.4f} probability)")

# Step 5: Plot with inset zoom (same style as before)
fig, ax = plt.subplots(figsize=(12, 6))
plot_histogram(counts_gauss, ax=ax)
ax.set_title("QFT Measurement Results - Gaussian Noisy Image (Frequency Domain)")
ax.set_ylabel("Counts")

# Inset for high-frequency components (Gaussian has more than clean, less than S&P)
from mpl_toolkits.axes_grid1.inset_locator import inset_axes, mark_inset
inset_ax = inset_axes(ax, width="40%", height="30%", loc='upper right')
plot_histogram(counts_gauss, ax=inset_ax)
```

```
inset_ax.set_ylim(0, 100)  # Adjust if needed    expect bars up to ~50  80
inset_ax.set_title("Zoom: High-Frequency Components")
mark_inset(ax, inset_ax, loc1=2, loc2=4, fc="none", ec="0.5")

plt.tight_layout()
plt.show()




# === Salt-and-Pepper Noisy Image QFT ===

# Normalize the salt-and-pepper noisy patch
# Important: sp_img contains 0s and 255s    after flattening, many zeros    norm very small for zero entries
# But some amplitudes become exactly zero    Qiskit initialize struggles with exact zeros sometimes
# Add small epsilon to avoid numerical issues (common trick)
sp_flat = sp_img.flatten().astype(float)

sp_norm = sp_flat / np.linalg.norm(sp_flat)

# If norm is zero (unlikely here), add tiny epsilon
if np.linalg.norm(sp_norm) == 0:
    sp_norm += 1e-10
sp_norm /= np.linalg.norm(sp_norm)  # renormalize

# Build circuit
qc_sp = prepare_quantum_state(sp_norm)  # your function already decomposes initialize    no 'state_preparation' gate
qc_sp = apply_qft(qc_sp)
qc_sp.measure_all()

# Use the SAME backend as before and transpile (critical!)
qc_sp = transpile(qc_sp, backend)

# Run with enough shots
result_sp = backend.run(qc_sp, shots=10000).result()
counts_sp = result_sp.get_counts()

```

```
# Debug print
print("\nRaw measurement counts (Salt-and-Pepper):")
print(counts_sp)

# === IMPROVED PLOTTING WITH INSET ===
fig, ax = plt.subplots(figsize=(12, 6))

plot_histogram(counts_sp, ax=ax)
ax.set_title("QFT Measurement Results - Salt-and-Pepper Noisy
    Image (Frequency Domain)")
ax.set_ylabel("Counts")

# Inset zoom for high-frequency components
from mpl_toolkits.axes_grid1.inset_locator import inset_axes,
    mark_inset

inset_ax = inset_axes(ax, width="40%", height="30%", loc='upper
    right')
plot_histogram(counts_sp, ax=inset_ax)
inset_ax.set_ylim(0, 50)  # Adjust based on your noise level
    S&P usually has more high-freq energy
inset_ax.set_title("Zoom: High-Frequency Components")

mark_inset(ax, inset_ax, loc1=2, loc2=4, fc="none", ec="0.5")

plt.tight_layout()
plt.show()

# Top states
total_shots = sum(counts_sp.values())
print("\nTop probability states (Salt-and-Pepper):")
sorted_counts = sorted(counts_sp.items(), key=lambda x: x[1],
    reverse=True)
for state, count in sorted_counts[:8]:
    print(f"|{state}> : {count} counts ({count/total_shots:.4f}
        probability)")



fig, axes = plt.subplots(1, 3, figsize=(18, 6))
counts_list = [counts_clean, counts_gauss, counts_sp]
```

```
titles = ["Clean Image QFT", "Gaussian Noise QFT", "Salt-and-Pepper Noise QFT"]

for ax, counts, title in zip(axes, counts_list, titles):
    plot_histogram(counts, ax=ax, title=title)
    ax.set_yscale('log')
    ax.set_ylim(bottom=0.8)
    ax.set_xlabel("Frequency States")
    ax.set_ylabel("Counts (log scale)")

plt.suptitle("Effect of Noise Types on Quantum Frequency-Domain Representation", fontsize=16, y=1.02)
plt.tight_layout()
plt.show()

# === Generate Text-Based QFT Circuit Diagram (No Extra Install Needed) ===
from qiskit import QuantumCircuit

# Create a 4-qubit circuit and apply your QFT function
qc_qft = QuantumCircuit(4)
qc_qft = apply_qft(qc_qft)  # Your existing apply_qft function

# Print the text diagram
print("Quantum Fourier Transform (QFT) Circuit for 4 Qubits ")
print(qc_qft.draw('text'))

# Classical Fourier Transform equivalent (1D FFT on flattened normalized data)
# This mirrors the QFT applied to the amplitude-encoded statevector,
# where the FFT coefficients' squared magnitudes give the exact probabilities
# in the frequency basis (without sampling noise from measurements).

import numpy as np
import matplotlib.pyplot as plt

def classical_fft_analysis(norm_data_flat, title, log_scale=False):
```

```
    """
    Compute classical 1D FFT on flattened normalized data.
    Returns the squared magnitudes (exact "probabilities").
    """
    # Ensure input is 1D flattened and normalized
    if norm_data_flat.ndim > 1:
        norm_data_flat = norm_data_flat.flatten()
    norm_data_flat = norm_data_flat / np.linalg.norm(
       norm_data_flat)

    # Compute FFT
    fft_result = np.fft.fft(norm_data_flat)

    # Squared magnitudes (analogous to QFT measurement
       probabilities)
    probs = np.abs(fft_result)**2

    # Sort and print top states for analysis
    print(f"\n=== Classical FFT: {title} ===")
    indices = np.argsort(probs)[::-1]  # Descending order
    print("Top 5 dominant frequency components:")
    for i in indices[:5]:
        print(f"  Index {i} (|{{:04b}}>.format(i)): {probs[i]:.4f
           }")

    # Plot bar chart (linear or log scale)
    fig, ax = plt.subplots(figsize=(12, 6))
    ax.bar(range(len(probs)), probs)
    ax.set_xlabel("Frequency Index (0 to 15)")
    ax.set_ylabel("Probability (Squared Magnitude)")
    ax.set_title(f"Classical FFT Spectrum - {title}")
    if log_scale:
        ax.set_yscale('log')
        ax.set_ylim(bottom=1e-10)
    plt.show()

    return probs

# Prepare flattened normalized data for each case
# Note: We flatten here to match the quantum amplitude encoding
    (1D over 16 elements)
```

```

# Clean
clean_flat = patch_array.flatten().astype(float)
clean_norm_flat = clean_flat / np.linalg.norm(clean_flat)
clean_classical_probs = classical_fft_analysis(clean_norm_flat, "Clean Image")

# Gaussian Noise
gauss_flat = gaussian_img.flatten().astype(float)
gauss_norm_flat = gauss_flat / np.linalg.norm(gauss_flat)
gauss_classical_probs = classical_fft_analysis(gauss_norm_flat, "Gaussian Noisy Image")

# Salt-and-Pepper Noise
sp_flat = sp_img.flatten().astype(float)
sp_norm_flat = sp_flat / np.linalg.norm(sp_flat)
sp_classical_probs = classical_fft_analysis(sp_norm_flat, "Salt-and-Pepper Noisy Image")

# === Side-by-Side Comparison Plots (Classical Spectra) ===
fig, axes = plt.subplots(1, 3, figsize=(18, 6))
probs_list = [clean_classical_probs, gauss_classical_probs, sp_classical_probs]
titles = ["Clean Image FFT", "Gaussian Noise FFT", "Salt-and-Pepper Noise FFT"]

for ax, probs, title in zip(axes, probs_list, titles):
    ax.bar(range(len(probs)), probs)
    ax.set_yscale('log')
    ax.set_ylim(bottom=1e-10)
    ax.set_xlabel("Frequency Index")
    ax.set_ylabel("Probability (log scale)")
    ax.set_title(title)

plt.suptitle("Classical FFT Spectra Comparison (Log Scale)", fontsize=16, y=1.02)
plt.tight_layout()
plt.show()


#!pip install qiskit qiskit-aer --quiet
```

```python

import numpy as np
import matplotlib.pyplot as plt
from qiskit.visualization import plot_histogram

# Step 1: Get quantum probabilities from counts (normalize to probs)
def get_probs_from_counts(counts, shots=10000):
    probs = {k: v / shots for k, v in counts.items()}
    # Ensure all 16 states are present (fill missing with 0)
    all_states = [format(i, '04b') for i in range(16)]
    for state in all_states:
        if state not in probs:
            probs[state] = 0
    return probs

# Assuming you have counts_clean, counts_gauss, counts_sp from your code
quantum_clean_probs = get_probs_from_counts(counts_clean)
quantum_gauss_probs = get_probs_from_counts(counts_gauss)
quantum_sp_probs = get_probs_from_counts(counts_sp)

# Step 2: Get classical probabilities (from your FFT code)
# Replace these with your actual probs arrays (16 elements each, normalized to sum=1)
# From your output: For clean, probs[0] = 15.9793 / 16 0.9987, etc. Fill in full array if needed.
# Here's approximate based on your top values (fill zeros for others for demo)

classical_clean_probs = np.zeros(16)
classical_clean_probs[0] = 15.9793 / 16
classical_clean_probs[8] = 0.0064 / 16
classical_clean_probs[4] = 0.0031 / 16
classical_clean_probs[12] = 0.0031 / 16
classical_clean_probs[1] = 0.0025 / 16
# Add others if you have full data; sum should   1

classical_gauss_probs = np.zeros(16)
classical_gauss_probs[0] = 15.9185 / 16
classical_gauss_probs[12] = 0.0140 / 16
```

```
classical_gauss_probs[4] = 0.0140 / 16
classical_gauss_probs[15] = 0.0080 / 16
classical_gauss_probs[1] = 0.0080 / 16
# Fill rest

classical_sp_probs = np.zeros(16)
classical_sp_probs[0] = 8.9251 / 16
classical_sp_probs[14] = 1.2605 / 16
classical_sp_probs[2] = 1.2605 / 16
classical_sp_probs[13] = 1.0663 / 16
classical_sp_probs[3] = 1.0663 / 16
# Fill rest

# Step 3: Function to plot overlay for one case
def plot_overlay(quantum_probs, classical_probs, title):
    # States as x-axis (0 to 15, binary labels)
    states = list(quantum_probs.keys())  # '0000' to '1111'
    indices = [int(s, 2) for s in states]  # 0 to 15
    q_values = [quantum_probs[s] for s in states]
    c_values = classical_probs  # Already 0-15 order

    fig, ax = plt.subplots(figsize=(12, 6))
    width = 0.35  # Bar width
    ax.bar(np.array(indices) - width/2, q_values, width, label='Quantum (QFT, 10k shots)', alpha=0.7)
    ax.bar(np.array(indices) + width/2, c_values, width, label='Classical (FFT, exact)', alpha=0.7)

    ax.set_yscale('log')
    ax.set_ylim(1e-6, 1)  # Adjust based on your data
    ax.set_xlabel('Frequency Index (Binary State)')
    ax.set_ylabel('Probability (Log Scale)')
    ax.set_title(title)
    ax.set_xticks(indices)
    ax.set_xticklabels(states, rotation=90)
    ax.legend()
    plt.tight_layout()
    plt.show()

# Generate individual overlays
```

```python
plot_overlay(quantum_clean_probs, classical_clean_probs, 'Clean Image: Quantum vs Classical')
plot_overlay(quantum_gauss_probs, classical_gauss_probs, 'Gaussian Noise: Quantum vs Classical')
plot_overlay(quantum_sp_probs, classical_sp_probs, 'Salt-and-Pepper Noise: Quantum vs Classical')

# Optional: Combined figure with 3 subplots
fig, axes = plt.subplots(1, 3, figsize=(18, 6))
titles = ['Clean', 'Gaussian Noise', 'Salt-and-Pepper Noise']
q_probs_list = [quantum_clean_probs, quantum_gauss_probs, quantum_sp_probs]
c_probs_list = [classical_clean_probs, classical_gauss_probs, classical_sp_probs]

for ax, q_probs, c_probs, title in zip(axes, q_probs_list, c_probs_list, titles):
    states = list(q_probs.keys())
    indices = [int(s, 2) for s in states]
    q_values = [q_probs[s] for s in states]

    width = 0.35
    ax.bar(np.array(indices) - width/2, q_values, width, label='Quantum', alpha=0.7)
    ax.bar(np.array(indices) + width/2, c_probs, width, label='Classical', alpha=0.7)

    ax.set_yscale('log')
    ax.set_ylim(1e-6, 1)
    ax.set_title(title)
    ax.set_xlabel('Freq Index')
    ax.set_xticks(indices[::2])  # Sparse labels to avoid clutter
    ax.set_xticklabels(states[::2], rotation=45)
    if ax == axes[0]:
        ax.set_ylabel('Prob (Log)')
        ax.legend()

plt.suptitle('Quantum vs Classical FFT Spectra Comparison (Log Scale)')
plt.tight_layout()
plt.show()
```