Appendix 1: AI Assistance and Prompt Log

This appendix transparently records how AI assistance (GitHub Copilot Chat / large language models) was used during development of the Assignment 2 deliverables. It includes: (i) the substantive prompts provided by the author, (ii) the major code / documentation templates generated or refactored in response, and (iii) brief commentary on how each AI-generated artifact was validated or modified before inclusion.

Scope of AI Use

AI assistance (limited strictly to Jupyter notebook development, not manuscript polishing) was used for:

- Drafting structured helper functions (Hadamard test wrapper, state preparation utilities, QFT comparison plotting cell).
- Converting ad hoc exploratory code into reproducible, parameterized notebook cells.
- Refactoring variable naming for consistency (lowercase forms: `num_qubits`, `state_vec`, `qft_err`).
- Creating safety/error handling patterns (normalization checks, phase alignment helper, error propagation logic for Hadamard test results).
- Generating comparison/visualization scaffolds (magnitude/phase, noise sweep GIF, baseline reconstruction cell) to ensure cells run independently.

No AI system executed experiments autonomously: every circuit build, simulation run, and numerical comparison was re-run locally and outputs visually inspected. Statistical outputs (standard errors, entropy measures) were recomputed from raw counts.

Key Author Prompts (Verbatim or Lightly Normalized)

Below are the principal prompts extracted from the working notes ("AI prompts.txt") and the refinement chat. (Punctuation lightly normalized; code semantics unchanged.)

1. **Circuit Construction Inquiry:** "Is there any advantage of using controlled phase over unitary gate as the mpl is not able to show box of gate on the line for target qubit?"
2. **Task Completion Check (Q1.1):** "Is the question 1.1 complete as given below?" (followed by the custom `Rotation_gate` implementation and three-qubit erroneous QFT circuit code).
3. **Phase-Gate Variant Request:** "Give phase version of my unitary gate code."
4. **Transformation / Refactor Prompt:** "Used copilot to align comments and replace qc with c in the phase version of my unitary code." (Implicit request for stylistic consistency.)
5. **End-to-End Hadamard Test Workflow Specification:** Numbered list (1–10) specifying: build and normalize padded vector; prepare state; combine with erroneous QFT; use Aer + Hadamard test to extract complex amplitudes; compute matrix product baseline; implement controlled U' construction; add sampling error bars; plot real/imag error vs. j; produce baseline comparisons (erroneous vs. exact vs. classical vs. `QFTGate`); plot noisy vs. classical QFT magnitude/phase.

6. **Iteration Guidance:** Prompts requesting refinement of LaTeX tables, figure captions, and QPE summary text (e.g., correcting entropy phrasing, removing tab-prefixed control sequences that broke compilation).

7. **Noise Model Clarification:** Implicit prompts in code asking for a unitarity-preserving parameterization of U_{noise} with $\delta = i\sqrt{\varepsilon(2-\varepsilon)}$ for $\varepsilon \in [0, 0.1]$.

8. **Global Phase Alignment Assistance:** Prompt context around creating an `align_global_phase` utility to facilitate visual comparison of amplitude phases.

9. **Statistical Error Propagation:** Requests to derive magnitude / phase error bars from separate real/imag Hadamard test binomial uncertainties.

10. **Template Consolidation:** Prompts leading to generation of a self-contained Q1.3 comparison cell that reconstructs any missing prerequisites so execution order does not matter.

Representative AI-Influenced Code Templates

The following higher-level structural templates originated from AI suggestions, then adapted:

State Preparation Utility (simplified extract).

```
prep_circuit, state_vec, vec_padded = build_state_preparation(vec_raw, dim)
# Ensures zero-padding, normalization, and labeled amplitude encoding gate.
```

Hadamard Test Wrapper (conceptual form).

```
results = hadamard_test_all_j(shots, qft_circuit=qft_err,
                              prep_circuit=prep_circuit, state_vec=state_vec)
# Returns per-j estimates of Re/Im with binomial standard errors.
```

Global Phase Alignment Helper.

```
def align_global_phase(a, ref):
    for r in ref:
        if abs(r) > 1e-12:
            return a * np.conj(r)/abs(r), ref * np.conj(r)/abs(r)
    return a, ref
```

Self-Contained Baseline Comparison Cell. Rebuilds missing objects (erroneous QFT, correct QFT, classical DFT, library `QFTGate`) and produces magnitude / phase plots with propagated standard errors for Hadamard-test estimates.

Noisy R_3 Injection Pattern.

```
Unoise = [[1-eps, 1j*sqrt(eps*(2-eps))], [1j*sqrt(eps*(2-eps)), 1-eps]]
# Inserted before (or after, variant) each controlled R3 location; validated unitarity.
```

Validation and Human Oversight

All AI-suggested code was:

- Executed locally (Qiskit Aer) to confirm numerical outputs and absence of runtime warnings.
- Compared against exact matrix multiplications (`Operator(circuit).data @ state_vec`).
- Inspected for phase conventions (ensuring consistent $e^{+2\pi ijk/N}$ sign with classical DFT implementation).
- Adjusted for naming consistency and removal of deprecated helper aliases.
- Re-run after integrating entropy computations and zero-probability filtering (QPE section).

Non-AI Contributions

The author manually:

- Selected the input vector, padding rule, and normalization.
- Implemented and verified the erroneous QFT gate substitutions (all would-be R_3 replaced by R_2).
- Chose shot counts (10k vs. 1M) to illustrate variance reduction.
- Interpreted convergence trends (QPE entropy and absolute phase error decay) and wrote final discussion.
- Repaired LaTeX formatting issues (e.g., removal of stray control characters) after AI-produced drafts.

Limitations of AI Outputs

Notebook-focused AI suggestions occasionally:

- Proposed very large shot counts (increasing runtime) without adaptive stopping criteria.
- Omitted explicit seeding for stochastic elements (noise draws), requiring manual insertion for reproducibility.
- Simplified error propagation (initially ignoring covariance between real and imaginary estimates) which was later refined.
- Generated redundant reconstruction code blocks; these were consolidated to a single self-contained comparison cell.

All such issues were manually reviewed and corrected in the final notebooks.

Summary

The AI assistance accelerated boilerplate generation and refactoring but did not replace original reasoning about quantum circuit correctness, statistical interpretation, or experimental design. All reported numerical results were regenerated deterministically from the final notebooks.

End of Appendix 1.