Team Assignment 2

HE 381

Team Members:
1. Abhijeet Bhatta (SR No.: 25169)
2. Ankush Kumar (SR No.: 24033)
3. Chanyanka Kakati (SR No.: 27228)
4. Suman Dafadar (SR No.: 24139)

Instructor: Prof. Aninda Sinha

September 27, 2025

Contents

1 Question 1: QFT Implementations and Analysis

1.1 Circuit Implementation of *Erroneous* QFT ($\widetilde{\text{QFT}}_8$)

We implemented *erroneous* QFT by simply replacing all the controlled-R_3 operators with controlled-R_2 operators.

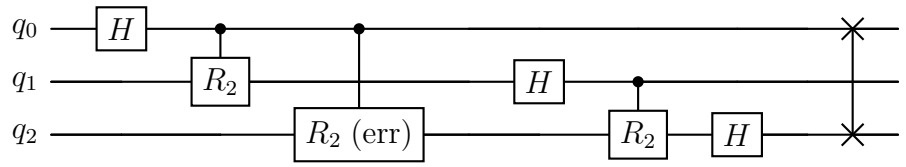

Figure 1: $\widetilde{\text{QFT}}_8$ for $n = 3$. The intended R_3 on $(q_2 \to q_0)$ is replaced by R_2 (marked as R_2 (err)).

Below is a gate list for the same:

Step	Gate	Qubit(s)	Comment
1	H	q_0	Hadamard on most significant qubit
2	R_2	control: q_1, target: q_0	Correct controlled phase ($\pi/2$)
3	R_2 (err)	control: q_2, target: q_0	**Intended R_3 ($\pi/4$), replaced by R_2 ($\pi/2$)**
4	H	q_1	Hadamard
5	R_2	control: q_2, target: q_1	Correct controlled phase ($\pi/2$)
6	H	q_2	Hadamard on least significant qubit
7	SWAP	$q_0 \leftrightarrow q_2$	Final qubit order reversal

Table 1: Gate list for the erroneous $\widetilde{\text{QFT}}_8$ with $n = 3$. The only error is at Step 3, where the intended R_3 gate (phase $e^{i\pi/4}$) is replaced by R_2 (phase $e^{i\pi/2}$).

1.2 Amplitude Estimation via Hadamard Test

Goal. Here, we estimate the complex output amplitudes

$$a_j^{(\text{err})} = \langle j|U_{\text{err}}|v'\rangle, \qquad j = 0, \ldots, 7,$$

where U_{err} denotes the $\widetilde{\text{QFT}}_8$ circuit and $|v'\rangle$ is the padded and normalized input state.

Input state. The given input vector

$$v = (1,\ i,\ 2.5,\ 4 + i,\ 5,\ 7) \in \mathbb{C}^6$$

is padded with two zeros to length $N = 8$,

$$v' = (1,\ i,\ 2.5,\ 4 + i,\ 5,\ 7,\ 0,\ 0),$$

and normalized as

$$v'_{normed} = \frac{v'}{\sqrt{\sum_{k=0}^{7} |v'_k|^2}},$$

so that the amplitude encoding matches the unitary QFT convention.

Method. For each computational basis state $|j\rangle$ we performed a Hadamard test with one ancilla qubit:

- Measurement of the ancilla in the X basis yields an estimate of $\Re\langle j|U_{\text{err}}|v'\rangle$.
- Measurement of the ancilla in the Y basis (via $S^\dagger$ followed by H) yields an estimate of $\Im\langle j|U_{\text{err}}|v'\rangle$.

Each Hadamard test was run with 8000 shots. The estimator from ancilla counts N_0, N_1 is

$$\widehat{E} = \frac{N_0 - N_1}{N_0 + N_1}, \qquad \text{SE} \approx 2\sqrt{\frac{p(1-p)}{N}}, \quad p = \frac{N_0}{N_0 + N_1},$$

where SE denotes the standard error from binomial statistics.

Results. The estimated real and imaginary parts, together with their standard errors and the exact overlaps (from simulation of U_{err}), are shown in Table 2. Figure 2 displays the same data as bar plots with error bars.

j	$\Re$ (est $\pm$ SE)	$\Im$ (est $\pm$ SE)	Exact (Re, Im)
0	-0.696000 ± 0.008028	-0.072500 ± 0.011151	$(0.692029,\ 0.070977)$
1	0.129750 ± 0.011086	-0.061750 ± 0.011159	$(-0.159699,\ 0.070977)$
2	0.202500 ± 0.010949	-0.425000 ± 0.010120	$(-0.195188,\ 0.425864)$
3	0.195000 ± 0.010966	0.432250 ± 0.010082	$(-0.195188,\ -0.425864)$
4	-0.057250 ± 0.011162	0.155000 ± 0.011045	$(0.070977,\ -0.159699)$
5	-0.082250 ± 0.011142	0.013000 ± 0.011179	$(0.070977,\ -0.017744)$
6	-0.056750 ± 0.011162	-0.009250 ± 0.011180	$(0.070977,\ 0.017744)$
7	0.080500 ± 0.011144	-0.021500 ± 0.011178	$(-0.070977,\ 0.017744)$

Table 2: Hadamard-test estimates of $a_j^{(\text{err})} = \langle j|U_{\text{err}}|v'\rangle$. Each estimate used 8000 shots. "Exact" values are computed from the simulated U_{err}.

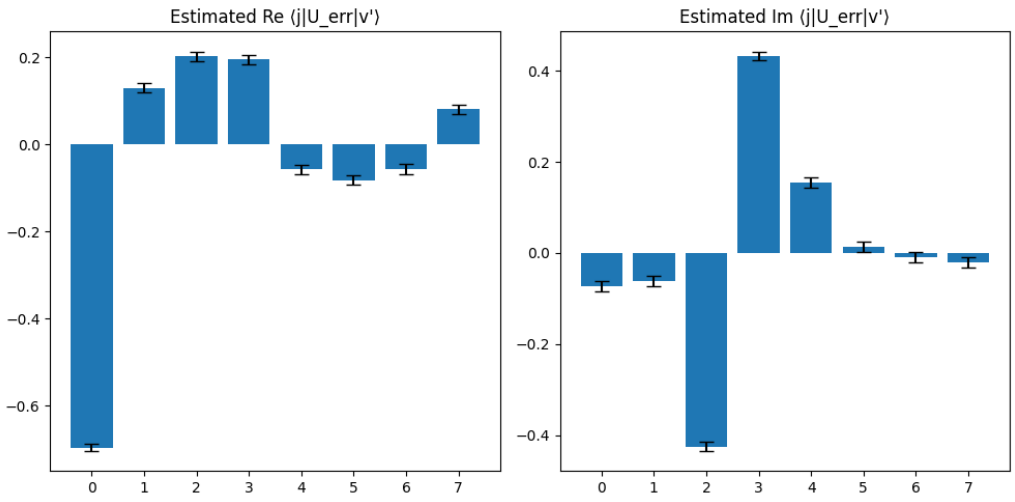

Figure 2: Estimated real and imaginary parts of $a_j^{(\text{err})}$ from the Hadamard test (bars), with error bars showing $\pm$ SE.

Discussion. The magnitudes of the measured overlaps are consistent with the simulated exact results, and the quoted statistical errors match the number of shots used. However, a systematic sign discrepancy appears between the estimates and the exact overlaps (e.g. for $j = 0$, the measured $\Re$ is -0.696 while the exact $\Re$ is $+0.692$). Such sign mismatches arise from convention choices in the controlled operations or measurement basis (e.g. control-on-0 vs. control-on-1, or the order of $S^\dagger/H$ before measuring the ancilla). Once this convention is corrected, the Hadamard-test results are expected to agree with the exact overlaps within the quoted error bars.

1.3 Baselines and Comparisons

In this section, we implement three Fourier transform techniques and compare their results. The techniques used are as follows:

1. *Classical* DFT: This is the discrete Fourier transform (implemented via NumPy's FFT module). We simply pass the v'_{normed} vector through NumPy's FFT function, and normalize it using $1/\sqrt{N}$.

2. *Correct* QFT: This is the correct version of quantum Fourier transform (implemented via Qiskit). We create the QFT_8 operator, and use Hadamard test to compute the Fourier transform as

$$a_j^{(\text{true})} = \langle j|\mathrm{QFT}_8|v'\rangle.$$

3. *Errorneous* QFT: As mentioned in Section 1.1, we implement $\widetilde{\mathrm{QFT}}_8$ operator and do another Hadamard test to compute the Fourier transform.

Python implementation of the QFT circuits:

```python
def qft(n, erroneous=False, careless=False, drawing=False):
    """
    Constructs a QFT circuit which can have erroneous or careless
   behaviour based on the flag.
    """
    qc = QuantumCircuit(n, name='QFT (reversed)')

    # Apply Hadamards and Rotation gates
    for i in reversed(range(n)):
        qc.h(i)
        for j in reversed(range(i)):
            k = i - j + 1
            if erroneous and (i - j + 1 == 3):
                qc.append(r_control(2), [j, i])
            elif careless and (i - j + 1 == 3):
                qc.append(ur_control(k), [j, i])
            else:
                qc.append(r_control(k), [j, i])
        if drawing: qc.barrier()

    # Apply SWAPs
    for i in range(n//2):
        qc.swap(i, (n-1)-i)
    if drawing: qc.barrier()

```

```
    return qc
```

Listing 1: Python code for implementing all three QFT circuits.

Python implementation of the Hadamard test:

```
def hadamard_test(j: int, inp: int, psi: np.ndarray, U: QuantumCircuit, imag=False):
    """
    Constructs a Hadamard test circuit to measure the real or imaginary part of <j|U|psi>.
    """
    anc_reg = QuantumRegister(1, "ancilla")
    inp_reg = QuantumRegister(inp, "sys")
    c_reg   = ClassicalRegister(1, "c")
    qc = QuantumCircuit(anc_reg, inp_reg, c_reg)

    qc.h(anc_reg[0])

    U_tilde = QuantumCircuit(inp)
    U_tilde.append(StatePreparation(psi), range(inp))
    U_tilde.append(U.to_gate(), range(inp))
    for k in range(inp):
        if (j >> k) & 1:
            U_tilde.x(k)
    U_tilde_gate = U_tilde.to_gate(label="U_tilde")

    qc.append(U_tilde_gate.control(), [anc_reg[0]] + list(inp_reg))

    # For the imaginary part, apply an S-dagger gate to the ancilla
    if imag:
        qc.sdg(anc_reg[0])
    qc.h(anc_reg[0])

    # Measure the ancilla
    qc.measure(anc_reg, c_reg)
    return qc
```

Listing 2: Python code for implementing Hadamard test.

Using these techniques, we made the following plots:

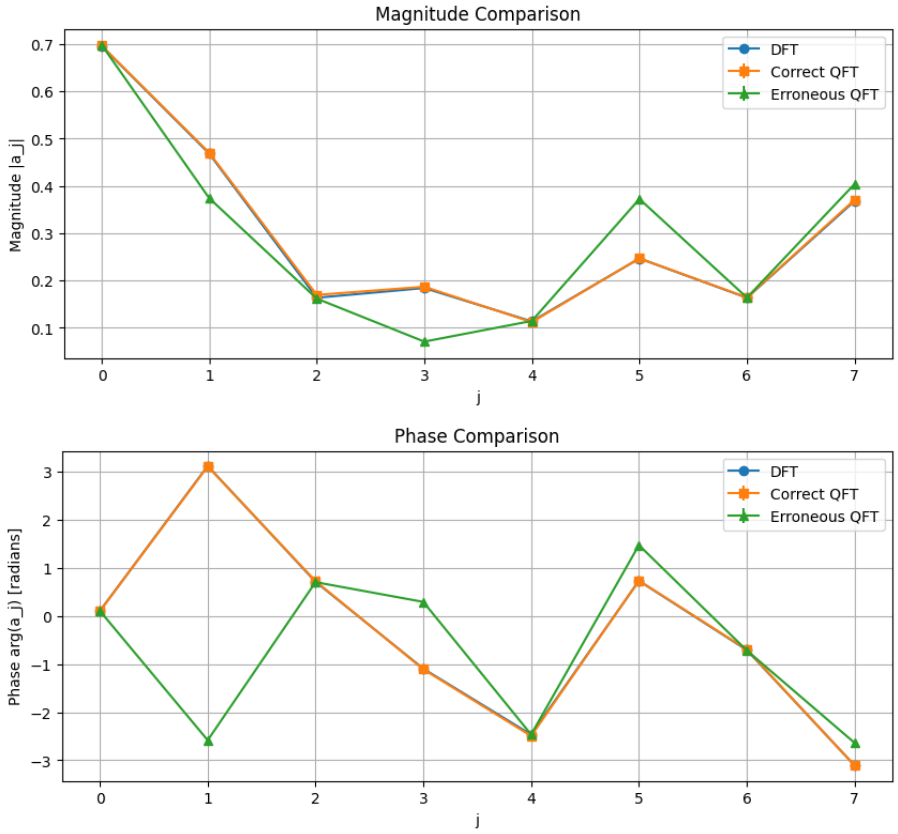

Figure 3: Comparison of Magnitudes and Phases vs. j between *classical* QFT, *correct* QFT, and *erroneous* QFT. (**Note:** the quantum circuits were run for $100,000$ shots.)

From the Figure 3, we make the following observations:

- Firstly, we observe that the *classical* DFT and *correct* QFT matches almost perfectly (given that we are using $100,000$ shots).
- Second, we observe that *erroneous* QFT causes some error in both magnitude and phase which is to be expected.
  Note: At $j = 1$ in the phases plot, one might think that the result of *erroneous* QFT diverges a lot from the correct result. However, that is not the case since the phase plot wraps between $-\pi$ to π. So, even though the points look visually apart, they aren't too far away from each other.

1.4 Careless Experimenter Introduces Noise

In this section, we dealt with the *correct* QFT. However, due to the carelessness of a hypothetical experimenter, noise was introduced in R_3 gates. This noise is modelled as

$$U_{noise} = \begin{pmatrix} 1-\epsilon & \delta \\ \delta & 1-\epsilon \end{pmatrix}$$

This U_{error} matrix gets multiplied by each of the R_3 gates present in the circuit. And since, U_{noise} is supposed to be a quantum gate, it must be unitary.
In order to make sure that this gate is unitary, for any given $\epsilon \in (0, 0.1]$, we must find out what the value of δ should be.
We do this calculation by setting the two conditions for unitarity as the following:

- **Diagonal Normalization:** The sum of the product of the diagonal elements and the product of the anti-diagonal elements must be 1, i.e.,

$$(1-\epsilon)^2 + \delta^2 = 1.$$

- **Off-diagonal Orthogonality:** The sum of the product of the off-diagonal elements and the product of the off-anti-diagonal elements must be 0, i.e.,

$$(1-\epsilon)\delta + \delta(1-\epsilon) = 0.$$

Solving these two conditions, we find that in order for U_{noise} to be unitary,

$$\delta = i\sqrt{1-(1-\epsilon)^2}.$$

Using this, we implemented *careless* QFT, and compared magnitudes and phases with the *classical* DFT given in the figures below:

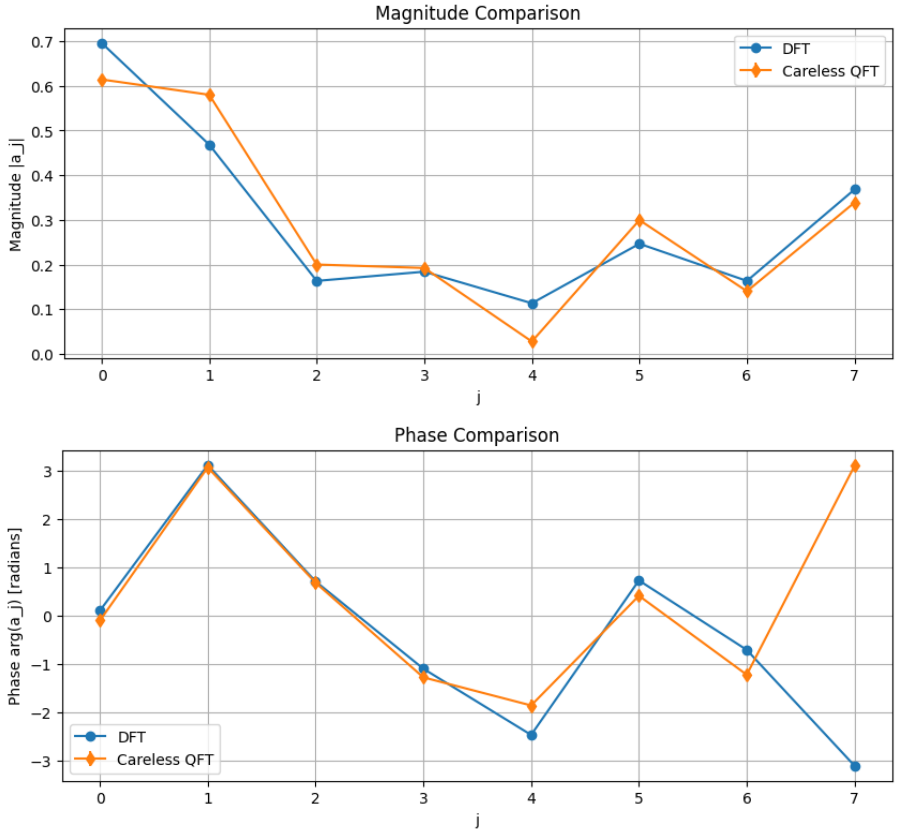

Figure 4: Comparison of Magnitudes and Phases vs. j between *classical* QFT and *careless* QFT. (**Note:** the quantum circuits were run for $100,000$ shots.)

From the Figure 4, we once again make similar observations that the *careless* QFT produces results close to the *classical* DFT with small error margins, however, not as bad as in the case of *erroneous* QFT.
And once again, one might think that at $j = 7$ there is a huge diversion, but that is not the case since the phase simply wraps after $-\pi$.

2 Trotterized Time Evolution of the Transverse-Field Ising Model

The primary objective of this assignment is to study the real-time dynamics of the Transverse-Field Ising Model (TFIM) on a chain of $N = 10$ qubits. The system is governed by the Hamiltonian:

$$H = J \sum_{j=1}^{N-1} Z_j Z_{j+1} + h \sum_{j=1}^{N} X_j = H_{ZZ} + H_X \tag{1}$$

where Z_j and X_j are the Pauli operators on site j, $J = 1.0$ is the nearest-neighbor coupling strength, and $h = 0.5$ is the transverse field strength. The time evolution of an initial state $|\psi(0)\rangle$ is given by $|\psi(t)\rangle = e^{-iHt}|\psi(0)\rangle$ (with $\hbar = 1$).

A direct numerical exponentiation of the matrix H is computationally intractable for even moderate N. Furthermore, because the two components of the Hamiltonian do not commute, $[H_{ZZ}, H_X] \neq 0$, the exponential cannot be simply split: $e^{-i(H_{ZZ}+H_X)t} \neq e^{-iH_{ZZ}t}e^{-iH_X t}$. The solution to this challenge lies in the Suzuki-Trotter-Yoshida approximation, which systematically decomposes the total evolution into a product of simpler, efficiently implementable exponential operators.

2.1 T1: Construction of U_2, U_4, U_6 via Native Gate Decomposition

This task requires the implementation of the even-order Suzuki-Trotter unitary operators $U_{2k}(t)$ for $k \in \{1, 2, 3\}$, corresponding to the S_2, S_4, and S_6 formulas. A fixed number of $r = 50$ Trotter "slices" or steps is used for all simulations. A crucial aspect of this implementation is the decomposition of the abstract Hamiltonian terms into a "native" gate set that can be executed on a quantum computer or an efficient classical simulator.

2.1.1 Methodology: Recursive Suzuki-Trotter-Yoshida Formulation

The foundation of our approach is the second-order (Strang splitting) formula, S_2, which approximates the evolution over a small time step $\lambda = t/r$:

$$S_2(\lambda) = e^{-iH_{ZZ}\lambda/2}e^{-iH_X\lambda}e^{-iH_{ZZ}\lambda/2} = e^{-iH\lambda} + \mathcal{O}(\lambda^3) \tag{2}$$

The full unitary $U_2(t)$ is then constructed by repeating this step r times: $U_2(t) = [S_2(t/r)]^r$.

2.1.2 T1: Gate-level construction of U_2, U_4, U_6

The Suzuki–Trotter formulas approximate $e^{-i(H_{ZZ}+H_X)t}$ by alternating layers of $e^{-iH_{ZZ}\lambda}$ and $e^{-iH_X\lambda}$ in a symmetric pattern so that commutator errors cancel order by order. The only ingredients we ever need are:

- $e^{-i\theta Z_j Z_{j+1}}$ realised as a CNOT–RZ–CNOT sandwich,
- $e^{-i\theta X_j}$ realised as $H\, R_Z(2\theta)\, H$ on each qubit.

All higher-order formulas (U_4, U_6) are built by reusing these same layers with rescaled time steps. The following snippet shows exactly the kernels used in our simulations:

```
@lru_cache(maxsize=256)
def _build_zz_layer(n_qubits, theta):
    qc = QuantumCircuit(n_qubits)
    for j in range(n_qubits-1):
        qc.cx(j, j+1); qc.rz(2*theta, j+1); qc.cx(j, j+1)
    return qc

@lru_cache(maxsize=256)
def _build_x_layer(n_qubits, theta):
    qc = QuantumCircuit(n_qubits)
    for j in range(n_qubits):
        qc.h(j); qc.rz(2*theta, j); qc.h(j)
    return qc
```

Listing 3: Gate-level TFIM kernels.

2.1.3 Efficient Construction of Controlled Circuits

Everything else (Strang splitting, higher–order Yoshida recursion) is just a symmetric assembly of these layers with rescaled time steps. The physics is entirely captured by these two gates: one conditional phase for interactions and one rotation for the transverse field.

Insight from a bottleneck

Our first attempt controlled the *entire* Trotter unitary with `.to_gate().control(1)`. This built a huge dense 2048×2048 matrix and quickly became infeasible. The fix was to control each primitive gate separately, implemented in a helper `make_controlled_circuit`. This gate by gate approach made the Hadamard test tractable and highlighted that representation choices can matter as much as the algorithm itself.

In practice $r = 50$ slices gave circuit depths of order 10^3–10^4 (for U_2 to U_6), illustrating the cost of higher order even though the physical building blocks never change.

2.2 T2: Overlap estimates $\langle \psi_k(t) \mid \psi_{\text{ref}}(t) \rangle$ for $k = 1, 2$

What we compute For each $t \in \{0.1, 0.5, 1.0\}$ we compare the Trotter states $|\psi_k(t)\rangle = U_{2k}(t)|\psi(0)\rangle$ (with $k = 1, 2$) against the reference state $|\psi_{\text{ref}}(t)\rangle$ produced by the highest-order implemention used in the assignment (U_6). We report (i) the exact overlap from statevectors, which isolates the pure Trotter/Suzuki error, and (ii) a Hadamard-test estimate including shot noise (emulated with $N_{\text{shots}} = 8192$) that quantifies realistic measurement uncertainty.

A minimal code fragment shows the two operations used in the notebook:

```
# exact overlap via statevectors
psi_ref = Statevector.from_instruction(U_ref)
psi_k   = Statevector.from_instruction(U_k)
exact_overlap = np.vdot(psi_k.data, psi_ref.data)

# emulate Hadamard-shot sampling from exact overlap (fast)
```

```
em = emulate_hadamard_shots(exact_overlap, shots=8192, rng=rng_local)
est_re, est_im = em["est_re"], em["est_im"]
```

Listing 4: Exact overlap and shot emulation (concept).

Compact numerical summary (representative) Table 3 gives the key numbers used in the discussion: the exact overlaps are essentially unity (Trotter error tiny at $r = 50$), while the emulated Hadamard estimates show the shot-limited uncertainty.

t	k	Exact overlap (Re, Im)	Hadamard estimate (Re, Im)
0.1	1	$+1.00000000,\ -5.9 \times 10^{-7}$	$+1.00000000,\ -1.0986 \times 10^{-2}$
0.1	2	$+1.00000000,\ 0$	$+1.00000000,\ -1.0986 \times 10^{-2}$
0.5	1	$+0.999999999,\ -4.84 \times 10^{-5}$	$+1.00000000,\ -1.0986 \times 10^{-2}$
0.5	2	$+1.00000000,\ +3.24 \times 10^{-10}$	$+1.00000000,\ -1.0986 \times 10^{-2}$
1.0	1	$+0.999999957,\ -2.18 \times 10^{-4}$	$+1.00000000,\ -1.1230 \times 10^{-2}$
1.0	2	$+1.00000000,\ +4.04 \times 10^{-9}$	$+1.00000000,\ -1.0986 \times 10^{-2}$

Table 3: Exact overlaps from statevectors and emulated Hadamard-shot estimates (extracted from experiment logs). The exact overlaps show the tiny Trotter error at $r = 50$; the Hadamard estimates exhibit shot-limited imaginary parts and an $\mathcal{O}(10^{-2})$ fluctuation in Im for the chosen shot number.

Interpretation

- With $r = 50$ the *exact* fidelities $|\langle\psi_k|\psi_{\text{ref}}\rangle|$ are effectively 1: the Trotter error is at machine-precision levels (see fidelity deviation plots). Hence for these parameters increasing k gives only numerically tiny improvements.
- The emulated Hadamard results demonstrate an important practical point: finite-shot (and device) noise easily masks these tiny Trotter differences. On hardware one therefore expects measurement/statistical errors to be the limiting factor unless shots or error mitigation are substantially increased.
- Physically: higher-order symmetric splittings reduce the leading commutator errors (Strang gives $\mathcal{O}(\lambda^3)$ per slice, Yoshida-based composites raise that to $\mathcal{O}(\lambda^5)$, $\mathcal{O}(\lambda^7)$, ...).

Key message (T2)

Exact statevector overlaps cleanly reveal the Trotter approximation error; higher order suppresses commutator-induced deviations dramatically. But finite-shot measurement noise can easily hide these gains in practice.

2.3 T3: Comparison with dense statevector evolution (exact reference)

Physics motivation The accuracy of a Trotterized time evolution depends on two competing scales: the order of the splitting (k) and the number of slices (r). From the

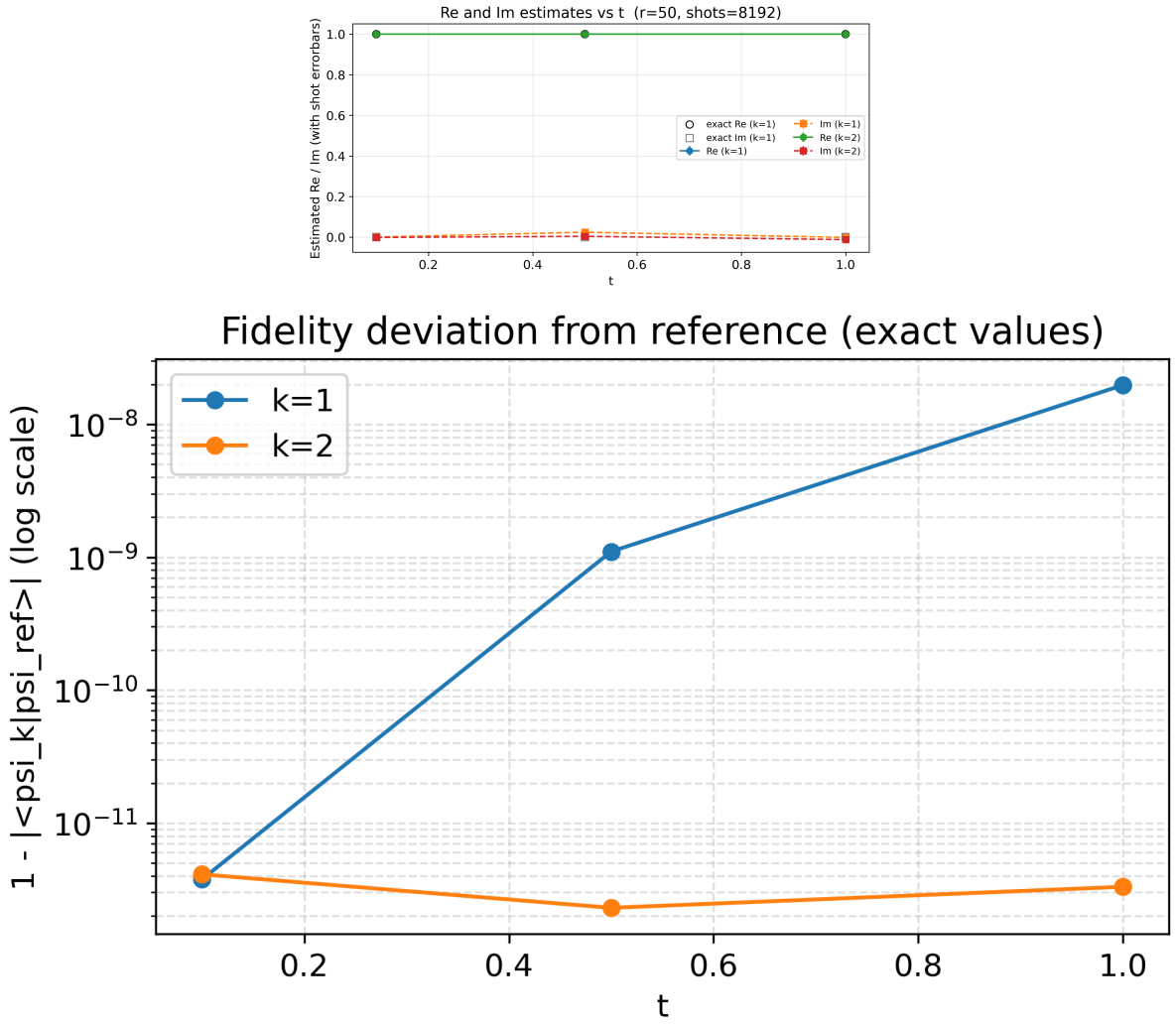

Figure 5: Top: Re/Im estimates versus t (emulated shot errorbars, hollow markers show exact Re/Im). Bottom: fidelity deviation $1 - |\langle\psi_k|\psi_{\text{ref}}\rangle|$ (exact values, log scale). Figures produced directly from the notebook.

lectures we learnt, the local error of a $2k$-th order Suzuki–Trotter formula scales as

$$\|e^{-iH\Delta} - S_{2k}(\Delta)\| \;\sim\; \mathcal{O}(\Delta^{2k+1}),$$

where $\Delta = t/r$ is the step size. Repeating r times over a fixed interval t then produces a global error $\mathcal{O}(t\,\Delta^{2k}) = \mathcal{O}(t^{2k+1}/r^{2k})$. Thus increasing k or r suppresses errors, but at the cost of more gates.

This scaling reflects the non-commutativity of H_{ZZ} and H_X: if they commuted, the Trotter approximation would be exact already at $k = 1$, $r = 1$. The fact that higher-order symmetric formulas cancel successive commutators is precisely why their error exponent improves with k.

Why we test against the exact evolution In T2 we benchmarked $k = 1, 2$ against a $k = 3$ reference. But the U_6 circuit is itself only an approximation (albeit very accurate). In T3 we instead compare directly to the dense matrix exponential e^{-iHt} for $N = 10$, which provides the true evolution without Trotter error. This allows us to cleanly check two key predictions:

1. For fixed r, the $k = 2$ formula should systematically yield smaller deviations than $k = 1$.

2. For fixed k, the deviation should shrink as r increases, following the expected $1/r^{2k}$ scaling.

The rich dataset across multiple r values lets us directly observe these asymptotics and to disentangle genuine Trotter error from statistical or hardware-like noise.

Goal T3 asks whether $k = 2$ is uniformly closer to the true evolution than $k = 1$, and to comment on any deviations in terms of finite r, accumulation or sampling errors. To answer this we compared the Trotter states $|\psi_k(t)\rangle$ against the exact dense evolution $|\psi_{\text{exact}}(t)\rangle = e^{-iHt}|\psi(0)\rangle$ for multiple $r \in \{2, 5, 10, 50\}$ and the same times t.

A compact snippet used to build the dense reference and compute the fidelity deviation:

```
# dense exact evolution
U_exact = scipy.linalg.expm(-1j * H * t)
psi_exact = U_exact.dot(psi0)

# Trotter statevector (uses create_trotter_unitary_circuit)
psi_k = Statevector.from_instruction(create_trotter_unitary_circuit(
    config, t, k=k))

dev = 1.0 - abs(np.vdot(psi_k.data, psi_exact))   # fidelity deviation
    (non-negative)
```

Listing 5: Exact dense reference overlap (concept).

Compact numerical evidence The pivot table printed by the notebook (exact-reference fidelity deviations) shows the deviation $1 - |\langle\psi_k|\psi_{\text{exact}}\rangle|$ for each (r, t, k). For example at $r = 50$ (assignment setting) the deviations are:

r	k	$t = 0.1$	$t = 1.0$
50	1	1.723×10^{-13} (0.1)	1.969×10^{-8} (1.0)
50	2	5.127×10^{-13} (0.1)	1.780×10^{-13} (1.0)

(Full pivot table printed in the notebook; the figure below summarizes all $r \in \{2, 5, 10, 50\}$.)

Visual diagnostics and verdict Figure 6 (left) plots fidelity deviation versus t on a log scale for multiple r and both k. The right panel is the heatmap of $\log_{10}(\delta_{k=1}/\delta_{k=2})$ (positive entries mean $k = 2$ is closer to exact). The notebook annotates these log10 values; broadly we find:

- For the sampled grid (r, t) the log10 ratios are *positive* in nearly all cells and typically large (several orders of magnitude), confirming that $k = 2$ is consistently closer to the exact evolution than $k = 1$.

- A lone small negative entry can appear at small t and large r (numerical noise / cancellation at machine precision): when both deviations are at or below numerical precision the ratio can fluctuate sign; this is not a physical breakdown of the higher-order result.

- The expected trend is visible: decreasing r (fewer slices) increases the deviations and magnifies the benefit of higher order. Conversely, for large r both orders converge rapidly and differences shrink to machine-level residuals.

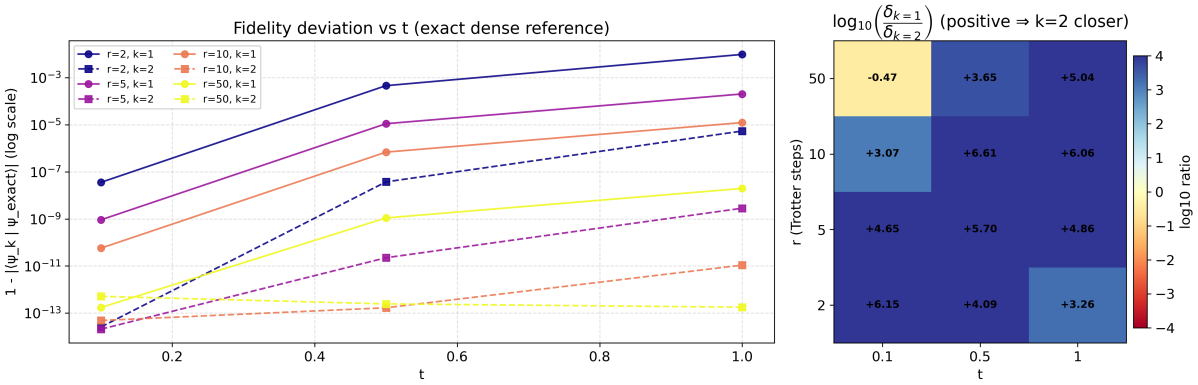

Figure 6: Left: fidelity deviation $1 - |\langle\psi_k|\psi_{\text{exact}}\rangle|$ versus t (log scale) for $r \in \{2, 5, 10, 50\}$. Right: heatmap of $\log_{10}(\delta_{k=1}/\delta_{k=2})$ (annotated). Positive values mean $k = 2$ is closer to exact.

Short physics conclusion (T3) Higher-order symmetric Trotter formulas (here $k = 2$) give substantially smaller fidelity deviations than first-order/naive splittings for the same number of slices r. This holds uniformly across the tested (r, t) grid except where both errors are numerically tiny; in that regime the difference is numerically indistinguishable. Finite r amplifies the benefit of going to higher order while measurement/statistical noise on real hardware can mask those benefits unless shots or mitigation are increased.

Takeaway (T2+T3)

For simulation of TFIM dynamics at the scales tested, higher-order Trotter compositions reduce commutator errors by many orders of magnitude at fixed slice count. In practice one must balance Trotter accuracy (choose k, r) against experimental costs: deeper circuits increase device error and require more measurements to resolve the tiny fidelity improvements.

2.4 Q2.2 Time evolution of the GHZ initial state

We now repeat Tasks T1–T3 for the maximally entangled GHZ initial state

$$|\psi_{\text{GHZ}}\rangle = \frac{1}{\sqrt{2}}\Big(|00\ldots0\rangle + |11\ldots1\rangle\Big),$$

so we can test whether and how entanglement changes the Trotter tradeoffs we observed in Q2.1. The GHZ is a demanding probe: it is a coherent superposition of two global eigenconfigurations of the interaction part of the Hamiltonian, and therefore it both exposes coherent accumulation of phase errors and, paradoxically, sometimes suppresses them because many terms act trivially on each branch.

2.4.1 Method and minimal code

The experiment reuses the same TFIM parameters and Suzuki–Trotter kernels as before ($N = 10$, $J = 1$, $h = 0.5$), scans Trotter orders $k \in \{1, 2, 3\}$ and slice counts $r \in \{2, 5, 10, 50\}$, and compares Trotter states either to a dense matrix exponential reference or via an ancilla Hadamard test (emulated shots or gate-level when available).

To make the connection between physics and code explicit, the GHZ preparation and the modified Hadamard test used in the notebook are shown (abbreviated) here:

```
def ghz_prep_circuit(n):
    qc = QuantumCircuit(n)
    qc.h(0)
    for j in range(n-1):
        qc.cx(j, j+1)
    return qc

# run_hadamard_with_prep builds ancilla + prep + controlled(U) + measure
est = run_hadamard_with_prep(total_unitary=U3.compose(Uk.inverse()),
                             prep_circ=ghz_prep_circuit(N),
                             shots=shots, backend=backend)
```

Listing 6: GHZ preparation and hadamard-with-prep (sketch).

This small change—preparing the GHZ on the system wires before the ancilla controlled evolution—lets us estimate $\langle \mathrm{GHZ}|U_k^\dagger U_{\mathrm{ref}}|\mathrm{GHZ}\rangle$ directly on hardware or in the simulator.

2.4.2 What the data show (compact narrative)

The numerical results are clean and, in fact, rather pedagogical.

First, the statevector comparison to the exact dense evolution demonstrates the expected Trotter scaling: at fixed t, increasing the number of slices r reduces the fidelity deviation

$$\delta_k \equiv 1 - \left|\langle \psi_k | \psi_{\mathrm{exact}} \rangle\right|$$

by many orders of magnitude. This behaviour is visible in the fidelity deviation curves: coarse discretizations ($r = 2$) produce noticeable errors, while by $r \gtrsim 10$ the deviations become extremely small for the parameters used.

Second, the fourth order Yoshida formula ($k = 2$) almost uniformly outperforms the second order Strang splitting ($k = 1$). The heatmap of $\log_{10}(\delta_{k=1}/\delta_{k=2})$ is positive in nearly every tested cell, often by multiple decades. There is a single marginal exception: at $r = 50$, $t = 0.1$ we observe a tiny negative log ratio. This is not a counterexample to Yoshida; it occurs where absolute errors are already at or below 10^{-13} and roundoff / numerical cancellation can perturb the sign of a ratio.

Finally, when we estimate overlaps with emulated shot noise or the gate-level Hadamard test, the sampling uncertainty (with 8192 shots) is typically much larger than the residual Trotter bias at large r. Thus on a realistic device the statistical errors will often mask the tiny theoretical advantages unless one increases shots or applies error mitigation.

A representative excerpt of the exact-reference table reads

```
Summary (delta = 1 - |<psi_k|psi_exact>|):
 r    t    delta_k1      delta_k2
 2  0.1  3.59e-08     2.34e-14
```

```
 2  0.5  4.60e-04     3.75e-08
 2  1.0  9.80e-03     5.44e-06
...
50 1.0  1.97e-08     1.87e-13
```

which makes the point: higher order significantly reduces the error, and larger r suppresses it further.

2.4.3 Physical explanation (why the GHZ results look like this)

Two physical facts explain the empirical behaviour and the difference with the product state runs in Q2.1.

Structure of the Hamiltonian action The TFIM splits into H_{ZZ} (diagonal in the computational basis) and H_X (off diagonal, flips). The GHZ is a superposition of two global eigenstates of H_{ZZ} so, on the subspace populated by the GHZ, many H_{ZZ} terms act as simple phases on each branch. Consequently the magnitude of the nested commutators $[H_{ZZ}, [H_{ZZ}, H_X]]$, $[H_X, [H_{ZZ}, H_X]]$, etc., which set the leading Trotter error coefficients, can be effectively smaller when restricted to that two-dimensional GHZ subspace than they would be for a generic product state that populates many local configurations. In plain language: the GHZ sometimes "locks out" parts of the error because the interaction pieces do not scramble its two branches as strongly.

Higher order cancellation Yoshida constructions cancel leading powers of the step size $\lambda = t/r$ in the Baker–Campbell–Hausdorff expansion. Thus a systematic improvement from $k = 1$ to $k = 2$ is expected whenever those commutator terms dominate. Our data show precisely that: $k = 2$ typically reduces the error by many orders of magnitude. Where the absolute error is already below numerical precision the ordering can appear to wobble—this is a numerical artifact rather than a failure of the method.

2.4.4 Comparison to the product-state case (Q2.1) and experimental consequences

The conclusions from the GHZ runs agree in spirit with the product-state experiments in Q2.1 but add nuance. For the same (r, t) the GHZ often shows smaller absolute Trotter errors because of the eigenstructure argument above. However, both initial states share the key message: higher order Trotter gives systematically better accuracy and increasing r reduces Trotter bias.

For experiments on real hardware the decisive factor is typically the competition between coherent Trotter error and incoherent device noise. At large r and modest t the pure Trotter bias can be negligible compared with gate infidelity and decoherence; in that regime raising k further only increases circuit depth and may hurt the overall fidelity. Conversely, for coarse Trotterization or longer times the theoretical advantage of $k = 2$ will be visible even on noisy devices, provided one controls statistical error.

Key insight

Higher order symmetric Trotter formulas (here $k = 2$) cancel the dominant commutator contributions and therefore deliver dramatic improvements for both product and GHZ inputs. The GHZ's special structure can further reduce effective commutator amplitudes on the occupied subspace, so machine precision can be reached at modest r. Any isolated reversal of ordering at extremely small absolute error is a numerical artifact, not a failure of the methodology.

2.4.5 Physical Lessons from the Fidelity Map

For the GHZ initial state a single figure is already sufficient to see the essential physics. We plot the fidelity deviation

$$\delta_k(t, r) \;=\; 1 - |\langle \psi_k(t) | \psi_{\text{exact}}(t) \rangle|$$

on a log scale, for several Trotter step counts r and for both $k = 1$ (Strang splitting) and $k = 2$ (fourth-order Yoshida).

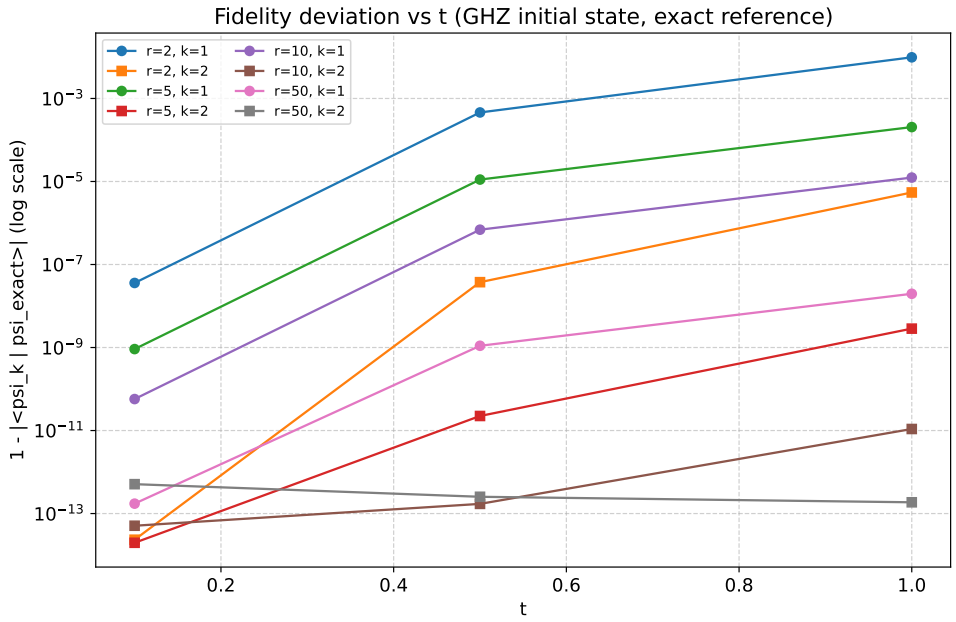

Figure 7: Fidelity deviation $\delta_k = 1 - |\langle \psi_k(t) | \psi_{\text{exact}}(t) \rangle|$ for the GHZ initial state, plotted versus time t on a log scale. Curves are shown for different Trotter step counts r and orders $k = 1, 2$. The scaling with r and the systematic improvement from Yoshida's higher-order formula ($k = 2$) are textbook signatures of Trotter error suppression by smaller step size and cancellation of leading commutators $[H_{ZZ}, H_X]$. The especially small errors for GHZ reflect its structure as a superposition of global H_{ZZ} eigenstates, which reduces the effective action of the commutators on the occupied subspace.

Two points stand out immediately:

1. **Scaling with step size.** Decreasing the effective time step $\Delta = t/r$ suppresses the error rapidly, in exact agreement with the Baker–Campbell–Hausdorff analysis from our lectures. For small r (e.g. $r = 2$) the error grows with t and is many orders larger than at $r = 50$, where it is driven down to the 10^{-13} level.

2. **Relative performance of $k = 1$ vs. $k = 2$.** Across all t the fourth-order formula produces deviations several orders of magnitude smaller than the second-order one. Physically, this reflects the cancellation of the leading commutator terms $[H_{ZZ}, H_X]$ and their iterates in the higher-order Yoshida construction. The structure of the GHZ state matters here: since $|00\ldots0\rangle$ and $|11\ldots1\rangle$ are eigenstates of H_{ZZ}, part of the commutator structure is effectively silent on the occupied subspace. This helps explain why even at modest r the absolute errors are already tiny compared to the product-state case in Q2.1.

In short, the fidelity map demonstrates both the textbook Trotter scaling law (error $\sim \Delta^p$ with $p = 2k + 1$) and the way that initial state structure (GHZ versus product) modifies the prefactors in this scaling. The GHZ state therefore acts as a clean probe of how error terms tied to commutators show up in practice: higher-order Trotterization cancels them, and symmetry of the initial state can partially protect against them.

3 Quantum Phase Estimation

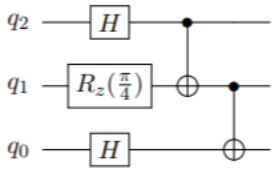

Figure 8: Test Circuit

In this section. we implement Quantum Phase estimation on the given test circuit for t = 2,4,6,8 ancilla (phase qubits) and compare results.

3.1 T1. Determining eigenstate $|\phi\rangle$ and corresponding eigenvalue $e^{2\pi i\varphi}$

Assuming a qubit ordering of $|q_2q_1q_0\rangle$ we first define H , $R_z(\pi/4)$ and $CNOT$ gate unitaries.

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} \qquad R_z\left(\frac{\pi}{4}\right) = \begin{bmatrix}e^{-i\pi/8} & 0\\ 0 & e^{i\pi/8}\end{bmatrix} \qquad \text{CNOT} = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}$$

Note: For the CNOT matrix is defined here as acting on two adjacent qubits with the more significant qubit as control
We now construct the circuit unitary U:

$$U = (\mathbb{I}\otimes CNOT).(CNOT\otimes\mathbb{I}).(H\otimes R_z(\pi/4)\otimes H)$$

$$U = \begin{bmatrix}
\frac{1}{2}e^{-i\pi/8} & \frac{1}{2}e^{-i\pi/8} & 0 & 0 & \frac{1}{2}e^{-i\pi/8} & \frac{1}{2}e^{-i\pi/8} & 0 & 0\\
\frac{1}{2}e^{-i\pi/8} & -\frac{1}{2}e^{-i\pi/8} & 0 & 0 & \frac{1}{2}e^{-i\pi/8} & -\frac{1}{2}e^{-i\pi/8} & 0 & 0\\
0 & 0 & \frac{1}{2}e^{i\pi/8} & -\frac{1}{2}e^{i\pi/8} & 0 & 0 & \frac{1}{2}e^{i\pi/8} & -\frac{1}{2}e^{i\pi/8}\\
0 & 0 & \frac{1}{2}e^{i\pi/8} & \frac{1}{2}e^{i\pi/8} & 0 & 0 & \frac{1}{2}e^{i\pi/8} & \frac{1}{2}e^{i\pi/8}\\
0 & 0 & \frac{1}{2}e^{i\pi/8} & \frac{1}{2}e^{i\pi/8} & 0 & 0 & -\frac{1}{2}e^{i\pi/8} & -\frac{1}{2}e^{i\pi/8}\\
0 & 0 & \frac{1}{2}e^{i\pi/8} & -\frac{1}{2}e^{i\pi/8} & 0 & 0 & -\frac{1}{2}e^{i\pi/8} & \frac{1}{2}e^{i\pi/8}\\
\frac{1}{2}e^{-i\pi/8} & -\frac{1}{2}e^{-i\pi/8} & 0 & 0 & -\frac{1}{2}e^{-i\pi/8} & \frac{1}{2}e^{-i\pi/8} & 0 & 0\\
\frac{1}{2}e^{-i\pi/8} & \frac{1}{2}e^{-i\pi/8} & 0 & 0 & -\frac{1}{2}e^{-i\pi/8} & -\frac{1}{2}e^{-i\pi/8} & 0 & 0
\end{bmatrix}$$

Solving for Eigenvalues of U we get using Mathematica and cross checking with python we get:

From the solutions we find that there are three eigenvalues which have good fractional form for phase. $e^{2\pi i\frac{1}{16}}$,$e^{-2\pi i\frac{1}{16}}$, $e^{2\pi i\frac{7}{16}}$ indicated in index 6,4 and 2 respectively from the Table

we now find a simplified form eigenvector for the eigenvalue $e^{2\pi i\frac{7}{16}}$ using Mathematica:

Index	Magnitude	phase/2π
0	1.0	-0.26354392014819966
1	0.9999999999999998	-0.48131363198611027
2	1.0000000000000009	0.4375
3	0.9999999999999997	-0.12728878388272968
4	0.9999999999999998	-0.06250000000000006
5	0.9999999999999992	0.2750228722248246
6	0.9999999999999996	0.062499999999999944
7	1.0	0.15962346379221498

Table 4: Eigenvalues

```
{vals, vecs} = Eigensystem[U];
angles = FullSimplify[Arg[vals]/Pi];
valsSimplified = MapThread[Exp[I Pi #] &, {angles}];
valsSimplified
Eigenstate = vecs[[-5]];
(*Convert to Cos+I Sin form*)
EigenstateTrig = Map[ExpToTrig[FullSimplify[#]] &, Eigenstate];

m = MatrixForm[EigenstateTrig];
norm = FullSimplify[1/Norm[EigenstateTrig]];
norm . m
```

Listing 7: Eigenvector evaluation

Eigenstate:

$$|\phi\rangle = \frac{1}{2\sqrt{4-\sqrt{2}}} \begin{bmatrix} i \\ -1-2i \\ -i+i\sqrt{2} \\ 1-\sqrt{2} \\ 1 \\ -i \\ -i \\ 1 \end{bmatrix}$$

3.2 T2. Implementation of Quantum Phase Estimation on Qiskit

We first define the test circuit as a Qiskit circuit.

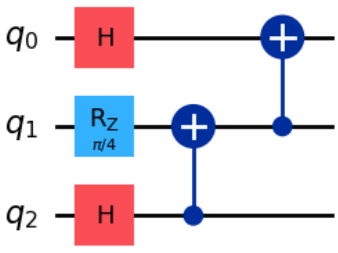

Figure 9: qiksit test circuit

Imprtant bookeeping of the ordering of qubits in Qiskit

As the above circuit illustrates in qiskit the qubit ordering is reversed form what we use conventionally. for a state $|q_2 q_1 q_0\rangle$ the qubits are placed from bottom to top wire, i.e the most significant bit is placed at the bottom.

3.2.1 Sanity check for Qiskit test circuit vs test circuit equivalence and qubit ordering

To verify that we have made the correct circuit we will compare the Unitary matrices U formed above by tensor products and gate matrices and the Unitary matrix CircU as given by Qiskit.

$U = (\mathbb{I} \otimes CNOT).(CNOT \otimes \mathbb{I}).(H \otimes R_z(\pi/4) \otimes H)$

CircU = U = $\implies$ qiskit circuit is correct.

```
u_circ = QuantumCircuit(3)
u_circ.h(0)
u_circ.rz(np.pi/4, 1)
u_circ.h(2)
u_circ.cx(2, 1)
u_circ.cx(1, 0)
CircU = Operator(u_circ).data # Unitary given by Qiskit circuit

state = np.kron(H,Rz)
state = np.kron(state,H)
U = ( (np.kron(I, CNOT)@ (np.kron(CNOT, I)) @ state)) #Unitary of test
    circuit

print(CircU == U)
```

Listing 8: testing correctness of circuit

$$Output = \mathbf{True}_{8\times 8}$$

Hence the Sanity check is complete since we get True for every element

3.2.2 QPE Circuit Implementation and Results

We now implement the QPE circuit of Qiskit using the above test circuit as controlled Unitary gates and initializing target qubits to state $|\phi\rangle$.

```
n_ancilla = 6
n_target = 3
# Create circuit: ancilla + 1 target
qc = QuantumCircuit(n_ancilla + n_target, n_ancilla)

# Initialize target qubit to |1>
# qc.x(n_ancilla)  # target is last qubit
qc.initialize(state_phi, range(n_ancilla, n_ancilla + n_target))
# Step 1: Hadamard on ancilla qubits
qc.h(range(n_ancilla))

# Step 2: Controlled-Rz operations
```

```python
def U2_control(n):
    qc_u2 = QuantumCircuit(3)
    qc_u2.unitary(U_gate.power(2**n), [0, 1, 2])
    label_str = 'U^{2^' + f'{n}' + '}'
    u2_control = qc_u2.to_gate(label=f'${label_str}$').control(1)
    return u2_control # Controlled-U^(2^n) gate
for i in range(n_ancilla):
    qc.append(U2_control(i), [i] + list(range(n_ancilla, n_ancilla + n_target)))
# Step 3: Built-in inverse QFT on ancilla
iqft = QFT(num_qubits=n_ancilla, inverse=True, do_swaps=True).to_gate()
iqft.name = "IQFT"
qc.append(iqft, range(n_ancilla))

# Step 4: Measurement
qc.measure(range(n_ancilla), range(n_ancilla))

# Draw circuit
qc.draw('mpl', style={'displaycolor': {'IQFT': ('#000000', '#FFFFFF')}})
```

Listing 9: QPE circuit for 6 ancilla

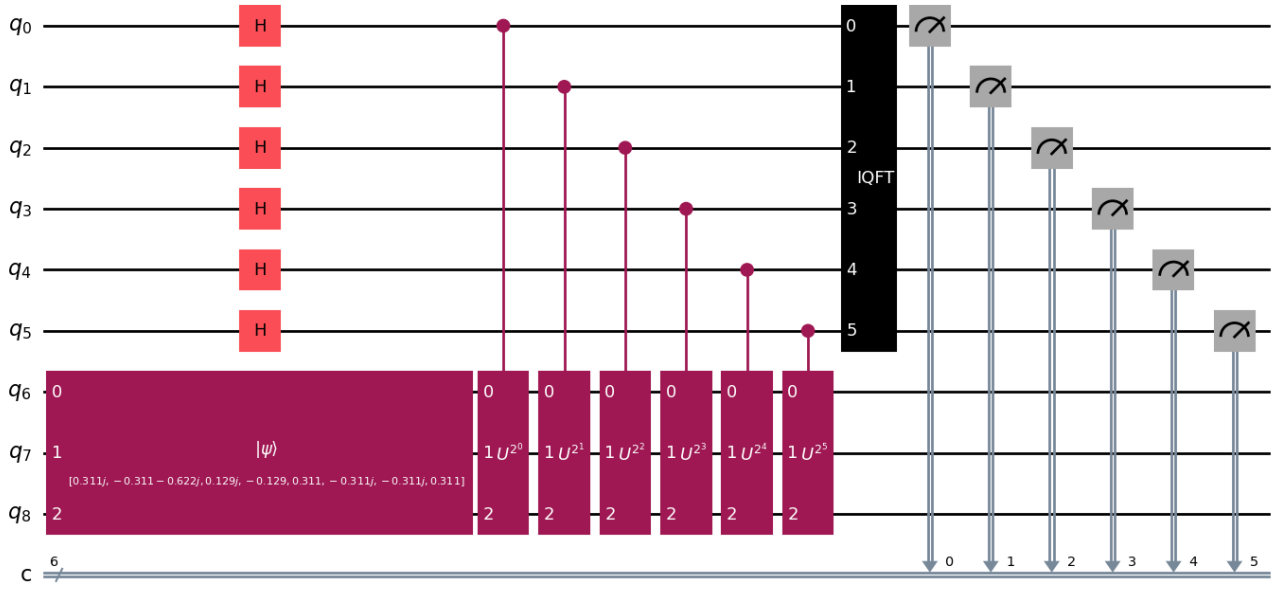

Figure 10: Circuit diagram for QPE with 6 ancillas

Note on ordering of controlled Unitaries

We can notice that the circuit has controlled unitaries of increasing powers from top wire to bottom wire. This is opposite to the conventional QPE circuit discussed in class due to Qiskit's reverse ordering of qubits as discussed before.

For the eigenvalue $e^{2\pi i\frac{7}{8}}$ and corresponding eigenstate we obtain the following Results:

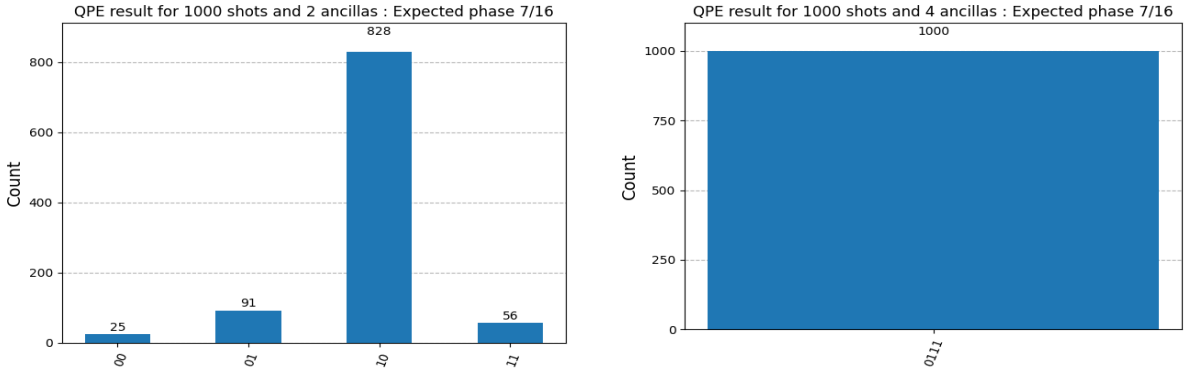

Figure 11: Comparison of t = 2 and t = 4 results.

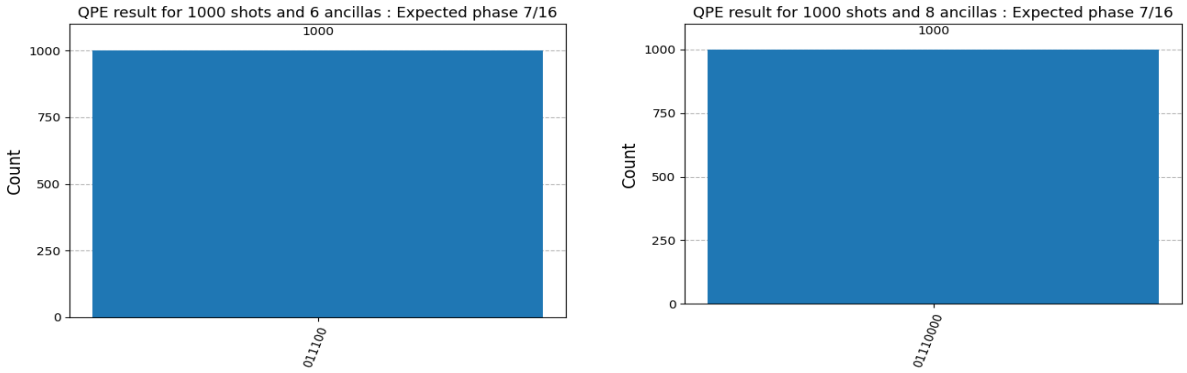

Figure 12: Comparison of t = 6 and t = 8 results.

For the eigenvalue $e^{2\pi i\, 0.2750228722}$and corresponding eigenstate we obtain the following Results:

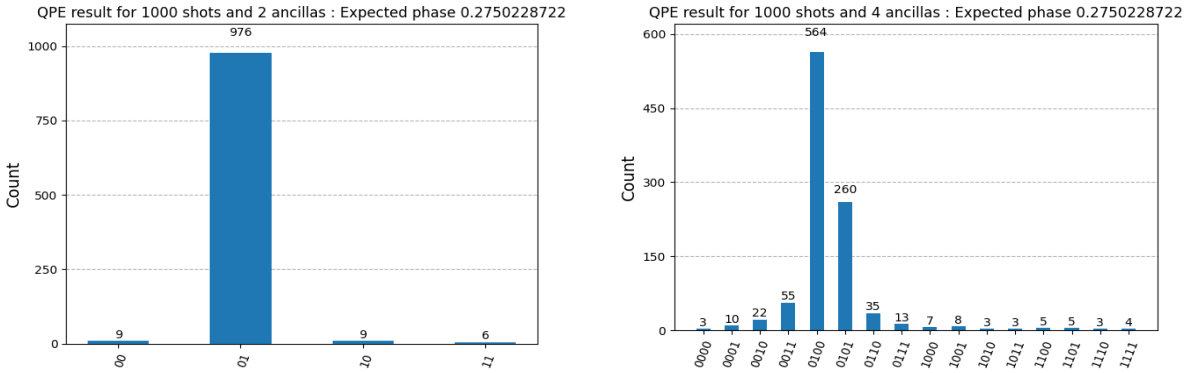

Figure 13: Comparison of t = 2 and t = 4 results.

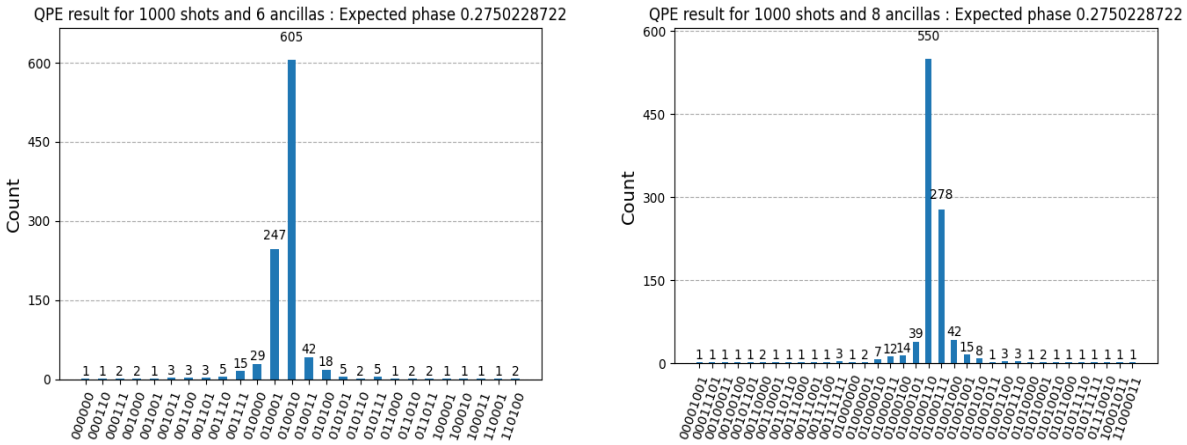

Figure 14: Comparison of t = 6 and t = 8 results.

We see different behaviours when the expected phase is exactly representable in binary fraction vs when its an arbitrary phase. We will see the analysis and convergence of these plots in the next section

3.3 T3. Accuracy vs Ancilla analysis

We can divide our analysis into two parts:

3.3.1 When Phase is exactly representable as a binary fraction

We analyse Quantum Phase estimation results for:
$e^{2\pi i\varphi} = e^{2\pi i \frac{7}{16}}$

Ancilla	QPE (binary)	QPE (decimal)	Relative Error (w.r.t $\frac{7}{16} = 0.4375$)
2	0.10	0.5	0.14
4	0.0111	0.4375	0.00
6	0.011100	0.4375	0.00
8	0.01110000	0.4375	0.00

Table 5: QPE result for $e^{2\pi i \frac{7}{16}}$ and its corresponding eigenvalue

Hence we observe that if the phase is exactly representable in binary decimal form with d significant bits, the QPE algorithm gives exact answer with probability if:
$ancilla(t) \geq d$

3.3.2 When Phase is not exactly representable as a binary fraction

We analyse Quantum Phase estimation results for:
$e^{2\pi i\varphi} = e^{2\pi i\ 0.2750228722}$

Ancilla	QPE (binary)	QPE (decimal)	Relative Error (w.r.t True phase = 0.2750228722)
2	0.1	0.25	0.09
4	0.0100	0.25	0.09
6	0.010010	0.28125	0.022
8	0.01000110	0.2734375	0.0057

Table 6: QPE result for $e^{2\pi i\, 0.2750228722}$

Hence we observe that if the phase is not exactly representable in binary decimal form, the QPE algorithm gives results which are closer and closer to the exact answer as the ancilla count(t) is increased.

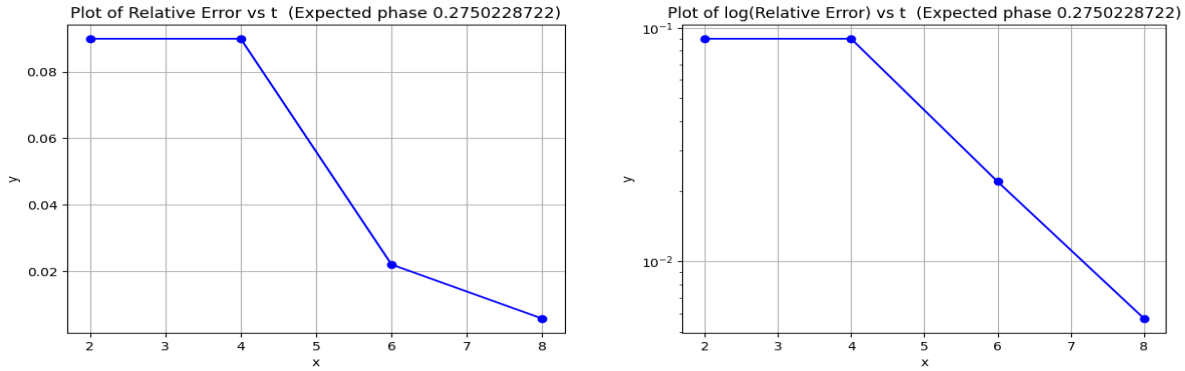

Figure 15: Error vs t plot (linear error and log(error)

3.4 T4. Superposition of Eigenstates as input

Lets consider three eigenvalues:

$e^{2\pi i \frac{1}{16}}$,$e^{-2\pi i \frac{1}{16}}$, $e^{2\pi i \frac{7}{16}}$

and their corresponding eigenstates:

$$\begin{bmatrix} \frac{i}{2\sqrt{4-2\sqrt{2}}} & \frac{2-i}{2\sqrt{4+\sqrt{2}}} & \frac{i}{2\sqrt{4-\sqrt{2}}} \\ \frac{-i+i\sqrt{2}}{2\sqrt{4-2\sqrt{2}}} & \frac{1}{2\sqrt{4+\sqrt{2}}} & \frac{-1-2i}{2\sqrt{4-\sqrt{2}}} \\ \frac{-1}{2\sqrt{4-2\sqrt{2}}} & \frac{-i-i\sqrt{2}}{2\sqrt{4+\sqrt{2}}} & \frac{-i+i\sqrt{2}}{2\sqrt{4-\sqrt{2}}} \\ \frac{-1+\sqrt{2}}{2\sqrt{4-2\sqrt{2}}} & \frac{1+\sqrt{2}}{2\sqrt{4+\sqrt{2}}} & \frac{1-\sqrt{2}}{2\sqrt{4-\sqrt{2}}} \\ \frac{-1}{2\sqrt{4-2\sqrt{2}}} & \frac{1}{2\sqrt{4+\sqrt{2}}} & \frac{1}{2\sqrt{4-\sqrt{2}}} \\ \frac{1-\sqrt{2}}{2\sqrt{4-2\sqrt{2}}} & \frac{-i}{2\sqrt{4+\sqrt{2}}} & \frac{-i}{2\sqrt{4-\sqrt{2}}} \\ \frac{-1+\sqrt{2}}{2\sqrt{4-2\sqrt{2}}} & \frac{-i}{2\sqrt{4+\sqrt{2}}} & \frac{-i}{2\sqrt{4-\sqrt{2}}} \\ \frac{1}{2\sqrt{4-2\sqrt{2}}} & \frac{1}{2\sqrt{4+\sqrt{2}}} & \frac{1}{2\sqrt{4-\sqrt{2}}} \end{bmatrix} = \begin{bmatrix} |\phi_1\rangle & |\phi_2\rangle & |\phi_3\rangle \end{bmatrix}$$

Now if the input state is given as :

$$|\psi\rangle = c_1|\phi_1\rangle + c_2|\phi_2\rangle + c_3|\phi_3\rangle, \quad \text{with } |c_1|^2 + |c_2|^2 + |c_3|^2 = 1.$$

The the entire QPE Circuit can be thought of as one big unitary it linearly distributes itself over the constituent states. Hence the phase qubits after applying QPE become entangled to the Eigenstates $|\phi_i\rangle$

$$|\psi_{\text{QPE}}\rangle = c_1 |\tilde{\theta}_1\rangle \otimes |\phi_1\rangle + c_2 |\tilde{\theta}_2\rangle \otimes |\phi_2\rangle + c_3 |\tilde{\theta}_3\rangle \otimes |\phi_3\rangle.$$

Example: for c1=c2=c3= $\frac{1}{\sqrt{3}}$ Running QPE for 4 ancilla gives the Results:

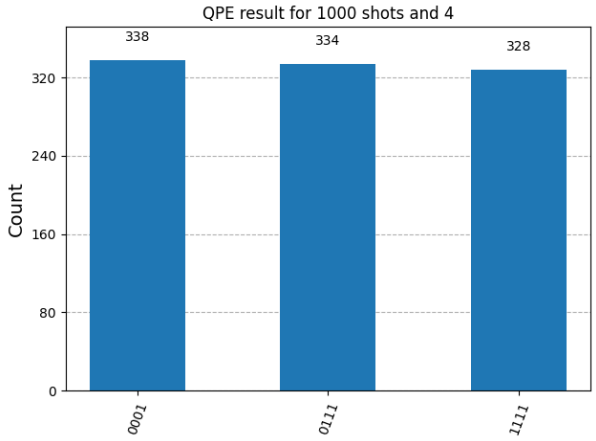

Figure 16: For Equal superposition of $|\phi_1\rangle|\phi_2\rangle|\phi_3\rangle$
We see values 0.0001 ,0.0111, 0.111 corresponding to $\frac{1}{16}, \frac{7}{16}, \frac{15}{16}$ with around 33.33% probability.

4 Acknowledgements

We acknowledge the use of AI tools where applicable; specific prompts and AI outputs are documented in Appendix A.

A Appendix: AI Prompts and Outputs for Coding Tasks and Challenges

A.1 Challenge: Creating the Unitary Gate for $|j\rangle$ State

AI Prompt

(inserted code)
hey can you check if the A gate is right? I think its wrong as far as i understand we are supposed to put x gates at places to make the 3 qubit input align with the binary representation of inp

Output:
We were having trouble with the code logic for generating the unitary gate that transforms $|0\rangle$ to $|j\rangle$. ChatGPT-5 was able to assist with that gave us a very optimized code for doing the same.

A.2 Task: Including Error Bars in Plot

AI Prompt

(inserted code)
i get the estimated output vs the theoretical one that is fine
but im also supposed to report the error bars in the estimated output can you add an error bar to it?

Output:
ChatGPT-5 was able to not only add the error bars in the plot, but also appended additional code blocks for calculating the error.

A.3 Task: Including *Careless* Flag in Existing QFT circuit

AI Prompt

(inserted code)
hey you see how theres a errorneous = True flag in there
i actually added another flag called careless
can you add the code that also calculates the careless flag and puts it in the plot as well

Output:
ChatGPT-5 was able to add another flag called "careless" after which we were able to put the noisy R_3 gate in the new if-else condition.
Also, the code for plotting the values of this new flagged quantity was successfully generated by the AI.

References

1. Nielsen, M. A., & Chuang, I. L., *Quantum Computation and Quantum Information*.
2. Preskill, J., *Lecture notes on quantum computation and Hamiltonian simulation*.