Algorithm Design Brief – Quantum Solvers for 1D Burgers' Equation

1. Framework Overview

This section introduces the two quantum algorithms implemented and compared for solving the 1D Burgers' equation:

- **HSE (Hydrodynamic Schrödinger Equation)**: This method uses quantum circuits with Trotterized evolution steps to simulate fluid dynamics. It applies **Rx** and **CZ** gates in the evolution process, which are typical for simulating quantum systems.

- **QTN (Quantum Tensor Network)**: This quantum-inspired classical approach uses **Matrix Product States (MPS)**to compress and evolve the velocity field. It evolves the velocity using simple 2-site gate approximations, making it more scalable compared to HSE for larger grid sizes.

The goal is to solve the **1D viscous Burgers' equation**, which models convective and diffusive dynamics in a fluid system.

2. PDE to Quantum Mapping

In this section, the **Burgers' equation** is mapped onto the quantum algorithms. It includes:

- **Initial Condition**: A shock tube is used to initialize the velocity field.
  - $u(x,0)=1$ if $x<0.5$, otherwise 0.
- **Discretization**:
  - **Space domain**: The grid is divided into N points, with the velocity $u(x)$ represented at each point.
  - **Time steps**: Time is discretized into T steps with step size Δt.
- **HSE Mapping**:
  - The velocity u is mapped to a quantum wavefunction ψ where $\psi=\sqrt{u}$.

- Evolution is performed under a discretized diffusion Hamiltonian $H \propto -\nabla^2$, which is implemented via quantum gates.

- **QTN Mapping**:
  - The velocity field is encoded in **MPS format** (rank-3 tensors).
  - Evolution is done by repeatedly applying 2-site gates, which approximates the evolution via Trotterization.

3. Gate Decomposition (HSE)

Here, the breakdown of quantum gates applied at each time step is provided:

- **Rx(θ)**: This gate is applied to each qubit, evolving the state with the parameter $\theta = -dt$, where dt is the time step.
- **CZ(i, i+1)**: This two-qubit gate captures the interaction between adjacent qubits, simulating the quantum diffusion process.
- Trotter steps are repeated T times, depending on the number of time steps in the simulation.

4. Resource Estimates

This section provides a rough estimate of the quantum resources required for the simulation:

Grid Size (N)	Qubits	2Q Gate Depth	Runtime (sim)
8	3	~6	0.01 s
16	4	~12	0.09 s
32	5	~20	~0.4 s

- **Qubits**: The number of qubits required depends on the grid size N. The qubit count is calculated as $\lceil \log_2(N) \rceil$.
- **2Q Gate Depth**: This is the total number of two-qubit gates used in the evolution process.
- **Runtime**: The estimated runtime to simulate the system for a given grid size.

5. Summary of Tradeoffs

A comparison of the **HSE** and **QTN** methods is given in this section, summarizing the strengths and weaknesses of each approach:

Feature	HSE (Quantum Circuit)	QTN (Tensor Network)
Native on QPU	Yes	No
Easy to scale	No (depth grows fast)	Yes (MPS truncation)
Accuracy	Medium (noise sensitive)	High (for coarse N)
T-count	Low (mostly Rx, CZ)	N/A
Gate depth	Moderate (Trotter steps)	None (classical ops)

6. Noise Mitigation

For the HSE method, simulated noise is used in the **AerSimulator** to assess the impact of noise on the simulation results. It also mentions potential future work to apply **Zero Noise Extrapolation (ZNE)** or **Clifford Data Regression** to mitigate the effects of noise in real hardware runs.

7. Real QPU Run Plan

This section discusses plans to run the quantum HSE circuit on an actual **IBM QPU** backend, specifically **ibmq_lima**. The following aspects will be captured during the run:

- **Basis state distribution**: The distribution of quantum states after evolution.
- **Raw measurement histogram**: The outcome of quantum measurements.
- **Error-mitigated output**: Potential future steps for error correction or mitigation.

8. Files Summary

A summary of the key files in the project is provided:

File	Purpose
`src/quantum_hse_solver.py`	Circuit for HSE method

`src/burgers_qtn_solver.py`	MPS-based QTN solver
`metrics/compare_all.py`	Benchmark and validation
`hardware_run/ibmq_run.py`	Real QPU execution
`results/`	Plots, CSVs, logs