Complete Guide: Enhanced Pauli Noise Analyzer

Table of Contents

Overview

This tool simulates quantum graph states under various noise conditions, particularly focusing on weather-based noise models. It's designed for researchers studying quantum sensing, quantum communication, and noise resilience in quantum systems.

Key Features

- **15+ Graph Topologies**: Complete, ring, grid, trees, custom topologies
- **4 Weather Noise Models**: Rain, dust, turbulence, clear conditions
- **Comprehensive Metrics**: Fidelity, entanglement, sensitivity, resilience
- **Monte Carlo Simulations**: Statistical analysis with configurable trials
- **Rich Visualizations**: Heatmaps, comparison plots, topology diagrams

Installation & Setup

Prerequisites

```
python

# Required packages
pip install numpy matplotlib networkx pandas seaborn scipy tqdm
```

Basic Setup

```python
# Simply run the main script
python enhanced_pauli_noise_improved.py

# Or import specific functions
from enhanced_pauli_noise_improved import *
```

Global Configuration Parameters

All simulation parameters are controlled by global variables at the top of the script. **Modify these to customize your simulation:**

Core Simulation Parameters

```python
QUBIT_RANGES = [3, 4, 5, 6, 7, 8]  # List of qubit counts to test
NUM_TRIALS = 100                    # Monte Carlo trials per configuration
RESULTS_FOLDER = "Enhanced_Results" # Output directory name
```

Graph Topologies

```python
GRAPH_TYPES = [
    # Basic topologies
    "complete", "ring", "line", "star",

    # Advanced topologies
    "grid_2d", "binary_tree", "ladder", "wheel",
    "small_world", "scale_free", "triangular_lattice",

    # Custom topologies
    "custom_ring_with_shortcuts", "custom_double_star",
    "custom_branched_path", "custom_modular", "custom_sparse_random"
]
```

Weather Models & Intensity

```python
WEATHER_MODELS = ["rain", "dust", "turbulence", "clear"]
INTENSITY_RANGE = np.linspace(0.2, 1.5, 6)  # 6 intensity levels from 0.2 to 1.5
```

Analysis Parameters

```python
BASIC_NOISE_N_RANGE = range(1, 3)          # Number of errors (1-2)
BASIC_NOISE_P_RANGE = np.linspace(0, 1, 20)  # Probability range (20 points)
BASIC_NOISE_SAMPLES = 30                   # Samples per data point
```

Visualization Settings

```python
FIGURE_DPI = 300                    # Image quality
FIGURE_SIZE_LARGE = (16, 12)        # Large plots
FIGURE_SIZE_MEDIUM = (12, 8)        # Medium plots
FIGURE_SIZE_SMALL = (8, 6)          # Small plots
```

Core Components

1. Graph State Builder (`EnhancedGraphStateBuilder`)

Creates quantum graph states with various topologies.

```python
# Create builder for 5 qubits
builder = EnhancedGraphStateBuilder(5)

# Create different topologies
builder.create_topology("complete")     # All qubits connected
builder.create_topology("ring")         # Ring topology
builder.create_topology("grid_2d")      # 2D grid layout

# Create graph state
state = builder.create_graph_state("complete")
```

2. Noise Models

Pauli Noise Model

```python
noise = PauliNoiseModel()
noisy_state = noise.apply_noise(
    state,
    num_qubits=5,
    n_errors=2,        # Number of errors
    probability=0.3,   # Error probability
    allow_repetition=False
)
```

Weather Noise Model

```python
weather = WeatherNoiseModel("rain")
error_prob = weather.error_probability(intensity=1.2)
```

Weather types and their characteristics:

- **Rain**: Poisson-distributed errors with attenuation
- **Dust**: Weibull-distributed errors
- **Turbulence**: Log-normal distributed errors
- **Clear**: Minimal baseline errors

3. Quantum Metrics (`QuantumStateMetrics`)

```python
# Calculate fidelity between states
fidelity = QuantumStateMetrics.fidelity(original_state, noisy_state)

# Calculate entanglement measures
entanglement = QuantumStateMetrics.average_entanglement(state, graph, num_qubits)

# Calculate mutual information
mutual_info = QuantumStateMetrics.mutual_information(state, qubit_a, qubit_b, num_qubits
```

4. Complete Simulator (`GraphStateSimulator`)

```python
# Create simulator
sim = GraphStateSimulator(num_qubits=6)

# Create graph state
sim.create_graph_state("wheel")

# Apply noise
sim.apply_weather_noise(weather_model, intensity=1.0)

# Get metrics
metrics = sim.calculate_metrics()
print(f"Fidelity: {metrics['fidelity']:.3f}")
```

Quick Start Guide

Option 1: Run Everything (Recommended for first time)

```python
# This runs the complete analysis with default parameters
results, summary, performance_results, topology_df = main()
```

What this does:

- Tests all 15+ topologies across all qubit ranges
- Applies all 4 weather models at 6 intensity levels
- Runs 100 Monte Carlo trials per configuration
- Generates ~20 visualization files
- Creates comprehensive analysis reports

Expected runtime: 10-30 minutes depending on your system

Option 2: Run Individual Components

Basic Noise Analysis

python

```
run_basic_noise_analysis(
    num_qubits=5,           # Number of qubits
    save_results=True       # Save plots
)
```

Custom Simulation

python

```
simulator = MonteCarloSimulator()
results = simulator.run_simulation(
    graph_types=["complete", "ring", "star"],  # Specific topologies
    qubit_ranges=[4, 5, 6],                    # Specific qubit counts
    weather_models=["rain", "clear"],          # Specific weather
    num_trials=50                              # Fewer trials for speed
)
```

Performance Comparison

python

```
performance_results = run_performance_comparison()
```

Advanced Usage

Creating Custom Topologies

Method 1: Edge List

python

```
builder = EnhancedGraphStateBuilder(6)

# Define custom edges
custom_edges = [(0,1), (1,2), (2,3), (3,0), (0,4), (4,5)]
builder.create_topology("custom", custom_edges=custom_edges)
```

Method 2: Adjacency Matrix

python

```python
adj_matrix = np.array([
    [0, 1, 0, 1],
    [1, 0, 1, 0],
    [0, 1, 0, 1],
    [1, 0, 1, 0]
])
builder.create_custom_topology_from_adjacency_matrix(adj_matrix)
```

Method 3: Custom Function

python

```python
def star_plus_ring(n):
    """Star topology with additional ring around outer nodes"""
    edges = [(0, i) for i in range(1, n)]  # Star
    edges += [(i, (i % (n-1)) + 1) for i in range(1, n)]  # Ring
    return edges

builder.create_custom_topology_from_function(star_plus_ring)
```

Method 4: Random Topology

python

```python
builder.create_random_topology(
    edge_probability=0.4,  # 40% chance for each edge
    min_edges=5,           # Minimum number of edges
    max_edges=15           # Maximum number of edges
)
```

Custom Weather Models

python

```python
# Create custom weather with specific parameters
custom_weather = WeatherNoiseModel("rain", params={
    'lambda_param': 3.0,    # Higher error rate
    'attenuation': 0.5      # Lower attenuation
})

# Use in simulation
sim.apply_weather_noise(custom_weather, intensity=2.0)
```

Focused Analysis

python

```python
# Analyze specific aspects
def analyze_topology_scalability():
    """Analyze how topologies scale with qubit count"""
    results = []

    for topology in ["complete", "ring", "star"]:
        for num_qubits in range(3, 10):
            sim = GraphStateSimulator(num_qubits)
            sim.create_graph_state(topology)

            # Test with standard noise
            weather = WeatherNoiseModel("dust")
            sim.apply_weather_noise(weather, intensity=1.0)

            metrics = sim.calculate_metrics()
            results.append({
                'topology': topology,
                'qubits': num_qubits,
                'fidelity': metrics['fidelity'],
                'entanglement': metrics['avg_entanglement_noisy']
            })

    return pd.DataFrame(results)

# Run the analysis
scalability_data = analyze_topology_scalability()
```

Customization Guide

Modifying Simulation Parameters

For Faster Testing

python

```python
# Reduce scope for quick tests
QUBIT_RANGES = [4, 5]                    # Test fewer qubit counts
NUM_TRIALS = 20                          # Fewer trials
GRAPH_TYPES = ["complete", "ring", "star"]  # Fewer topologies
INTENSITY_RANGE = np.linspace(0.5, 1.0, 3)  # Fewer intensity levels
```

For Comprehensive Analysis

python

```python
# Increase scope for thorough analysis
QUBIT_RANGES = [3, 4, 5, 6, 7, 8, 9, 10]   # More qubit counts
NUM_TRIALS = 200                           # More trials
INTENSITY_RANGE = np.linspace(0.1, 2.0, 10) # More intensity levels
```

For Specific Research Focus

python

```python
# Example: Focus on communication applications
GRAPH_TYPES = ["complete", "ring", "small_world", "scale_free"]
WEATHER_MODELS = ["turbulence", "clear"]  # Relevant for atmospheric communication
QUBIT_RANGES = [6, 8, 10, 12]             # Practical system sizes
```

Adding New Graph Types

python

```python
# Add to TOPOLOGY_BUILDERS in EnhancedGraphStateBuilder
@staticmethod
def _create_honeycomb(n: int) -> List[Tuple[int, int]]:
    """Create honeycomb lattice topology"""
    # Implementation here
    edges = []
    # ... honeycomb logic ...
    return edges

# Then add to the dictionary
TOPOLOGY_BUILDERS["honeycomb"] = lambda n: EnhancedGraphStateBuilder._create_honeycomb(n
```

Custom Metrics

python

```python
def custom_analysis(simulator):
    """Add your custom analysis"""
    # Access the quantum states
    original = simulator.original_state
    noisy = simulator.noisy_state

    # Calculate custom metrics
    custom_metric = your_calculation(original, noisy)

    return custom_metric
```

Output Files & Results

Generated Files

The simulation creates a timestamped results folder with:

```
Enhanced_Results_YYYYMMDD_HHMMSS/
├── all_topologies_N_qubits_timestamp.png      # Topology visualizations
├── topology_comparison_matrix_timestamp.png   # Properties comparison
├── topology_properties_timestamp.csv          # Raw topology data
├── weather_simulation_results_timestamp.csv   # Main simulation results
├── sensitivity_heatmap_timestamp.png          # Sensitivity analysis
├── fidelity_by_weather_timestamp.png          # Weather impact
├── fidelity_by_topology_timestamp.png         # Topology comparison
├── resilience_score_timestamp.png             # Resilience analysis
├── entanglement_change_timestamp.png          # Entanglement dynamics
├── mutual_info_timestamp.png                  # Information measures
├── performance_comparison_timestamp.png       # Category comparison
├── basic_noise_analysis_timestamp.png         # Basic noise plots
├── custom_bowtie_topology_timestamp.png       # Custom topology examples
└── simulation_summary_timestamp.txt           # Text summary
```

Understanding Results

Main Results DataFrame

python

```python
# Key columns in results CSV:
- graph_type: Topology used
- num_qubits: System size
- weather_type: Noise model
- intensity: Weather intensity
- trial: Monte Carlo trial number
- fidelity: State fidelity (0-1, higher better)
- sensitivity: Change sensitivity (higher = better sensor)
- resilience: High fidelity probability (higher = better communication)
- avg_entanglement_*: Entanglement measures
- graph_density: Topology connectivity
- is_connected: Graph connectivity (0/1)
```

Summary Statistics

```python
# Best configurations for different applications:
summary['best_sensor_graph']      # Highest sensitivity
summary['most_resilient_graph']   # Highest resilience
summary['optimal_qubit_count']    # Optimal sizes for different uses
summary['weather_impact']         # Weather severity ranking
```

Interpreting Plots

Topology Comparison Matrix

- **Density**: Higher = more connected
- **Clustering**: Higher = more local connectivity
- **Connectivity**: Binary connected/disconnected

Fidelity Plots

- **Y-axis**: Fidelity (1.0 = perfect, 0.0 = completely corrupted)
- **X-axis**: Weather intensity
- **Higher lines**: More noise-resistant topologies

Sensitivity Heatmap

- **Bright colors**: High sensitivity (good for sensing)
- **Dark colors**: Low sensitivity
- **Trade-off**: High sensitivity often means low resilience

Resilience Scores

- **Height**: Probability of maintaining high fidelity
- **Higher bars**: Better for communication applications

Troubleshooting

Common Issues

Memory Errors

python

```python
# Reduce simulation scope
QUBIT_RANGES = [3, 4, 5]  # Fewer qubits
NUM_TRIALS = 50           # Fewer trials
```

Long Runtime

python

```python
# Quick test configuration
GRAPH_TYPES = ["complete", "ring"]
WEATHER_MODELS = ["clear", "rain"]
INTENSITY_RANGE = np.linspace(0.5, 1.0, 3)
```

Import Errors

bash

```bash
# Install missing packages
pip install networkx pandas seaborn scipy tqdm matplotlib numpy
```

Plot Display Issues

python

```python
# If plots don't show, add:
import matplotlib
matplotlib.use('Agg')  # For headless systems
# Or
plt.ioff()  # Turn off interactive mode
```

Error Messages

"Graph not created yet"

python

```python
# Always create graph state before applying noise
sim = GraphStateSimulator(5)
sim.create_graph_state("complete")  # Required before noise
sim.apply_weather_noise(weather_model)
```

"Unknown graph type"

```python
# Check GRAPH_TYPES list for valid options
print(GRAPH_TYPES)  # See available topologies
```

"Could not create topology"

- Some topologies may fail for certain qubit counts
- The code automatically falls back to complete graph
- Check console warnings for details

Performance Tips

Speed Optimization

1. Reduce Simulation Scope

```python
# For development/testing
QUBIT_RANGES = [4, 5]                    # 2 instead of 6 sizes
NUM_TRIALS = 25                          # 25 instead of 100 trials
GRAPH_TYPES = ["complete", "ring", "star"]  # 3 instead of 15+ types
```

2. Parallel Processing

```python
# Modify the tqdm loop for parallel processing
from multiprocessing import Pool

def run_single_trial(params):
    # Extract parameters and run single trial
    return result

# Use with Pool for parallel execution
```

3. Selective Analysis

```python
# Run only what you need
run_basic_noise_analysis()              # Just basic analysis
# OR
run_performance_comparison()            # Just performance comparison
# Instead of main() which runs everything
```

Memory Optimization

1. Process in Batches

```python
# For large simulations, process topology groups separately
for topology_batch in [["complete", "ring"], ["star", "line"]]:
    simulator = MonteCarloSimulator()
    results = simulator.run_simulation(graph_types=topology_batch)
    # Process results before next batch
```

2. Clear Variables

```python
# Clear large arrays when done
del simulator.results_df
import gc; gc.collect()
```

Analysis Tips

1. Focus on Key Metrics

- **Fidelity**: Overall state preservation
- **Sensitivity**: Ability to detect changes (sensing applications)
- **Resilience**: Probability of high fidelity (communication applications)

2. Compare Relative Performance

- Don't focus on absolute values
- Compare trends across topologies and conditions
- Look for robust performers across different scenarios

3. Statistical Significance

- Default 100 trials provides good statistics
- Increase NUM_TRIALS for more precise results
- Watch for error bars in plots

Summary

This tool provides a comprehensive framework for analyzing quantum graph states under realistic noise conditions. The key to effective use is:

1. **Start Simple**: Run with reduced parameters first
2. **Understand Outputs**: Focus on relative comparisons
3. **Customize Gradually**: Modify parameters for your specific research
4. **Interpret Physically**: Connect results to quantum physics principles

The default configuration provides a good balance of comprehensiveness and computational feasibility. Adjust the global parameters based on your computational resources and research focus.