Supplementary materials

APPENDIX 1

Quantum-Inspired Air Quality Monitoring System Simulation codes in Python

Here is a Python simulation that models proposed quantum-inspired air quality monitoring scenario. This implementation uses classical computing to simulate the quantum entanglement concept between input and output pixels, with error modelling and continuous refinement.

```
importnumpyas np
fromscipy.interpolateimportgriddata
importmatplotlib.pyplotasplt
fromsklearn.gaussian_processimportGaussianProcessRegressor
fromsklearn.gaussian_process.kernelsimportRBF,ConstantKernel
import time

classQuantumInspiredAirQuality:
def__init__(self,num_stations=10,grid_size=100):
self.num_stations=num_stations
self.grid_size=grid_size
self.station_locations=None
self.true_air_quality=None
self.entangled_pixels=None
self.quantum_estimate=None
self.error_model=None
self.initialize_system()

definitialize_system(self):
# Randomly place monitoring stations
np.random.seed(42)
self.station_locations=np.random.rand(self.num_stations,2)*self.grid_size

# Generate synthetic true air quality field (unknown in real scenario)
      x =np.linspace(0,self.grid_size,self.grid_size)
      y =np.linspace(0,self.grid_size,self.grid_size)
xx,yy=np.meshgrid(x, y)

# Create synthetic pollution sources
      sources =[(20,20,150),(70,80,200),(30,60,180)]
self.true_air_quality=np.zeros((self.grid_size,self.grid_size))
forsx,sy, intensity in sources:
dist=np.sqrt((xx-sx)**2+(yy-sy)**2)
self.true_air_quality+= intensity *np.exp(-dist/20)

# Normalize to 0-300 AQI range
self.true_air_quality=(self.true_air_quality/self.true_air_quality.max())*300

# Initialize entangled pixels (simulating quantum entanglement concept)
self.initialize_entanglement()

# Initialize error model
```

```
self.initialize_error_model()

definitialize_entanglement(self):
"""Simulate quantum entanglement between station pixels and output grid pixels"""
# For each station, create entanglement with nearby pixels
self.entangled_pixels={}

fori,(x, y)inenumerate(self.station_locations):
# Determine affected area around each station (simulating entanglement range)
        radius =self.grid_size/5
grid_x,grid_y=np.meshgrid(np.arange(self.grid_size),np.arange(self.grid_size))
distances =np.sqrt((grid_x- x)**2+(grid_y- y)**2)
influence =np.exp(-distances/radius)

# Store entanglement weights
self.entangled_pixels[i]={
'station_pos':(x, y),
'influence_map': influence,
'current_value':self.sample_true_air_quality(x, y)
}

# Initialize quantum estimate with interpolated values
self.update_quantum_estimate()

definitialize_error_model(self):
"""Initialize Gaussian Process model for error estimation"""
# Start with some initial error observations
    X =self.station_locations
    y =np.zeros(len(X))# Initially assume no error

    kernel =ConstantKernel(1.0)* RBF(length_scale=1.0)
self.error_model=GaussianProcessRegressor(kernel=kernel, alpha=0.1)
self.error_model.fit(X, y)

defsample_true_air_quality(self, x, y):
"""Sample the true air quality at a point (for simulation purposes)"""
    xi =int(np.clip(x,0, self.grid_size-1))
yi=int(np.clip(y,0, self.grid_size-1))
returnself.true_air_quality[yi, xi]

defupdate_station_value(self,station_idx,new_value):
"""Update a station's reading and propagate through entangled pixels"""
ifstation_idx>=self.num_stations:
raiseValueError("Invalid station index")

# Update the station's value
self.entangled_pixels[station_idx]['current_value']=new_value

# Update quantum estimate
self.update_quantum_estimate()
```

```
# Update error model when we have ground truth (from mobile sensor)
returnself.quantum_estimate

defupdate_quantum_estimate(self):
"""Update the quantum-inspired estimate by combining entangled pixels"""
self.quantum_estimate=np.zeros((self.grid_size,self.grid_size))
total_influence=np.zeros((self.grid_size,self.grid_size))

# Combine all entangled pixels
for station inself.entangled_pixels.values():
        value = station['current_value']
        influence = station['influence_map']
self.quantum_estimate+= value * influence
total_influence+= influence

# Normalize by total influence
self.quantum_estimate=np.divide(self.quantum_estimate,total_influence,
                                out=np.zeros_like(self.quantum_estimate),
                                where=total_influence!=0)

# For areas with no influence, use simple interpolation
ifnp.any(total_influence==0):
        points =np.array([v['station_pos']for v inself.entangled_pixels.values()])
        values =np.array([v['current_value']for v inself.entangled_pixels.values()])
grid_x,grid_y=np.meshgrid(np.arange(self.grid_size),np.arange(self.grid_size))

interp_values=griddata(points, values,(grid_x,grid_y), method='linear')
no_data_mask=(total_influence==0)
self.quantum_estimate[no_data_mask]=interp_values[no_data_mask]

defadd_ground_truth_measurement(self, x, y):
"""Add a ground truth measurement from mobile sensor and update error model"""
true_value=self.sample_true_air_quality(x, y)
estimated_value=self.quantum_estimate[int(y),int(x)]
    error =true_value-estimated_value

# Update error model
    X =self.error_model.X_train_
y =self.error_model.y_train_

    X =np.vstack([X,[x, y]])
    y =np.append(y, error)

self.error_model.fit(X, y)

returntrue_value,estimated_value, error

defpredict_error(self, x, y):
"""Predict the error at a given location"""
returnself.error_model.predict(np.array([[x, y]]))[0]
```

```
defget_corrected_estimate(self):
"""Get quantum estimate corrected with error model predictions"""
grid_x,grid_y=np.meshgrid(np.arange(self.grid_size),np.arange(self.grid_size))
        points =np.column_stack([grid_x.ravel(),grid_y.ravel()])

# Predict error across entire grid
        errors =self.error_model.predict(points).reshape(self.grid_size,self.grid_size)

returnself.quantum_estimate+ errors

defvisualize(self):
"""Visualize the current state"""
plt.figure(figsize=(18,12))

# True air quality
plt.subplot(2,3,1)
plt.imshow(self.true_air_quality,cmap='RdYlGn_r',vmin=0,vmax=300)
plt.scatter(self.station_locations[:,0],self.station_locations[:,1], c='black', marker='^')
plt.title("True Air Quality (Unknown)")
plt.colorbar(label='AQI')

# Quantum estimate
plt.subplot(2,3,2)
plt.imshow(self.quantum_estimate,cmap='RdYlGn_r',vmin=0,vmax=300)
plt.title("Quantum-Inspired Estimate")
plt.colorbar(label='AQI')

# Error
        error =self.true_air_quality-self.quantum_estimate
plt.subplot(2,3,3)
plt.imshow(error,cmap='bwr',vmin=-100,vmax=100)
plt.title("Estimation Error")
plt.colorbar(label='Error (True - Estimate)')

# Entanglement influence
plt.subplot(2,3,4)
        influence =np.zeros((self.grid_size,self.grid_size))
for station inself.entangled_pixels.values():
            influence += station['influence_map']
plt.imshow(influence,cmap='viridis')
plt.title("Total Entanglement Influence")
plt.colorbar()

# Corrected estimate
        corrected =self.get_corrected_estimate()
plt.subplot(2,3,5)
plt.imshow(corrected,cmap='RdYlGn_r',vmin=0,vmax=300)
plt.title("Corrected Estimate (with Error Model)")
plt.colorbar(label='AQI')

# Corrected error
```

```
corrected_error=self.true_air_quality- corrected
plt.subplot(2,3,6)
plt.imshow(corrected_error,cmap='bwr',vmin=-100,vmax=100)
plt.title("Corrected Estimation Error")
plt.colorbar(label='Error (True - Corrected)')

plt.tight_layout()
plt.show()


# Simulation demo
if __name__=="__main__":
print("Initializing Quantum-Inspired Air Quality Monitoring System...")
   system =QuantumInspiredAirQuality(num_stations=15,grid_size=100)

# Initial visualization
system.visualize()

# Simulate real-time updates
print("\nSimulating real-time updates...")
foriinrange(5):
print(f"\nUpdate cycle {i+1}:")

# Randomly update some stations
forstation_idxinnp.random.choice(system.num_stations, size=3, replace=False):
        x, y =system.entangled_pixels[station_idx]['station_pos']
true_value=system.sample_true_air_quality(x, y)
# Add some noise to simulate real measurements
measured_value=true_value+np.random.normal(0,5)
system.update_station_value(station_idx,measured_value)
print(f"  Updated station {station_idx} at ({x:.1f}, {y:.1f}) with value: {measured_value:.1f} AQI")

# Take some mobile measurements (ground truth)
for _ inrange(2):
        x, y =np.random.rand(2)*system.grid_size
true_val,estimated_val, error =system.add_ground_truth_measurement(x, y)
print(f"  Mobile sensor at ({x:.1f}, {y:.1f}): True={true_val:.1f}, Estimated={estimated_val:.1f},
Error={error:.1f}")

# Visualize after updates
system.visualize()
time.sleep(1)

print("\nSimulation complete. Final results:")
final_error=np.mean(np.abs(system.true_air_quality-system.quantum_estimate))
final_corrected_error=np.mean(np.abs(system.true_air_quality-system.get_corrected_estimate()))
print(f"  Mean absolute error (raw quantum estimate): {final_error:.2f}")
print(f"  Mean absolute error (corrected estimate): {final_corrected_error:.2f}")
```

Note that each monitoring station is "entangled" with nearby pixels through an influence map that decays with distance. It updates to station values immediately affect their entangled pixels. Indeed station values can be updated and the quantum estimate recalculated instantly and thus there is no need for full spatial analysis pipeline. It also uses Gaussian Process Regression to model the error between quantum estimates and ground truth. Mobile sensor measurements improve the error model over time. In addition using the Performance Metrics, it is possible to track estimation error compared to the "true" air quality (which would be unknown in reality) and shows improvement from raw quantum estimates to error-corrected estimates.