# Modified field equations preserving CPT symmetry

class TemporalField:

def \_\_init\_\_(self):

self.W = None # Temporal flow field

self.S = None # Entropy field

def compute\_temporal\_evolution(self, state, params):

"""

Compute temporal evolution preserving microscopic reversibility

while generating macroscopic arrow of time

"""

# Modified field equation

dW\_dt = -self.grad(self.P\_t)/self.rho\_t + self.nu\_t \* self.laplacian(self.W)

# Entropy coupling term

S\_coupling = self.compute\_entropy\_coupling(state)

# CPT-symmetric quantum term

Q\_term = self.compute\_quantum\_term(state)

# Combined evolution respecting symmetries

total\_evolution = dW\_dt + S\_coupling + Q\_term

return total\_evolution

def compute\_entropy\_coupling(self, state):

"""

Compute entropy coupling that preserves microscopic reversibility

but leads to entropy increase at macroscopic scales

"""

local\_entropy = self.compute\_local\_entropy(state)

entropy\_gradient = self.compute\_entropy\_gradient(local\_entropy)

# Scale-dependent coupling

g\_r = 1.0 / (1.0 + (self.r/self.r\_c)\*\*self.n)

return g\_r \* entropy\_gradient

def compute\_quantum\_term(self, state):

"""

Compute quantum term preserving CPT symmetry

"""

psi = state.wavefunction

H = self.hamiltonian

# CPT-symmetric quantum evolution

quantum\_evolution = -1j \* (H @ psi - psi @ H)/self.hbar

return quantum\_evolution

# Usage

field = TemporalField()

evolved\_state = field.compute\_temporal\_evolution(initial\_state, params)