""" DARKBOT™ Basic Example

This example demonstrates the basic usage of the DARKBOT™ Resonant Field Intelligence Architecture for simple pattern recognition.

© 2025 Cato Johansen // DARKBOT™ // Artifact №369.157.248 """

import torch import numpy as np import asyncio import matplotlib.pyplot as plt from darkbot import DarkBot, DarkBotConfig

async def main(): print("DARKBOT™ Resonant Field Intelligence Architecture") print("==========")

```
# Initialize the system
print("\nInitializing DarkBot™...")
config = DarkBotConfig()
# Use smaller dimensions for this example
config.field.dimensions = 128
darkbot = DarkBot(config)

print(f"Initialized on device: {darkbot.device}")
print(f"Field dimensions: {config.field.dimensions}")
print(f"Resonance parameters: γ={config.resonance.gamma}, χ_high=
{config.resonance.chi_high}, χ_low={config.resonance.chi_low}")

# Create a set of pattern fields
print("\nCreating pattern library...")
patterns = {
    'A': torch.randn(config.field.dimensions, dtype=torch.complex64, device=darkbot.device),
    'B': torch.randn(config.field.dimensions, dtype=torch.complex64, device=darkbot.device),
    'C': torch.randn(config.field.dimensions, dtype=torch.complex64, device=darkbot.device),
    'D': torch.randn(config.field.dimensions, dtype=torch.complex64, device=darkbot.device),
    'E': torch.randn(config.field.dimensions, dtype=torch.complex64, device=darkbot.device),
}
print(f"Created {len(patterns)} patterns: {', '.join(patterns.keys())}")

# Process each pattern through the quantum system
print("\nProcessing patterns through resonant field...")
processed_patterns = {}
for key, pattern in patterns.items():
    processed = await darkbot.process_quantum(pattern)
    processed_patterns[key] = processed
    print(f"Processed pattern {key} - Field coherence:
{darkbot.calculate_coherence(processed):.4f}")

# Create a test query that's a mixture of patterns
print("\nCreating test query (mixture of patterns A and C)...")
query = 0.7 * patterns['A'] + 0.3 * patterns['C']

# Process the query
print("Processing query through resonant field...")
processed_query = await darkbot.process_quantum(query)
print(f"Query coherence: {darkbot.calculate_coherence(processed_query):.4f}")

# Perform One Draw search
print("\nPerforming One Draw search...")
result = darkbot.one_draw_search(query, list(patterns.values()))

# Find the matching pattern name
```

```
pattern_names = list(patterns.keys())
match_idx = result['match_index']
match_name = pattern_names[match_idx]

print(f"Query recognized as pattern: {match_name}")
print(f"Match confidence: {result['confidence']:.4f}")

# Show resonance profile
print("\nResonance profile:")
for i, name in enumerate(pattern_names):
    print(f"  {name}: {result['resonance_profile'][i]:.4f}")

# Create a simple visualization of the resonance profile
print("\nCreating visualization...")
plt.figure(figsize=(10, 6))

# Resonance bar chart
plt.subplot(2, 1, 1)
plt.bar(pattern_names, result['resonance_profile'])
plt.title('Resonance Profile')
plt.ylabel('Resonance')
plt.ylim([0, 1])

# Plot the field coherence history
plt.subplot(2, 1, 2)
plt.plot(darkbot.coherence_history)
plt.title('Field Coherence History')
plt.xlabel('Processing Cycle')
plt.ylabel('Coherence (χ)')
plt.axhline(y=config.resonance.chi_high, color='g', linestyle='--', label=f'High (χ=
{config.resonance.chi_high})')
plt.axhline(y=config.resonance.chi_low, color='r', linestyle='--', label=f'Low (χ=
{config.resonance.chi_low})')
plt.legend()

plt.tight_layout()

# Save or show the visualization
plt.savefig("darkbot_resonance.png")
print("Visualization saved to 'darkbot_resonance.png'")

# Optional: show the plot if in an interactive environment
try:
    plt.show()
except:
    pass
```

```
    # Benchmark the system
    print("\nRunning benchmarks...")
    benchmark_results = darkbot.benchmark_system(
        iterations=10,  # Low iterations for example
        dim=config.field.dimensions,
        batch_size=4
    )

    print("\nBenchmark Results:")
    print(f"One Draw search time: {benchmark_results['one_draw_time_ms']:.2f} ms")
    print(f"Classical search time: {benchmark_results['classical_time_ms']:.2f} ms")
    print(f"Full processing time: {benchmark_results['process_time_ms']:.2f} ms")
    print(f"Speedup factor (vs classical): {benchmark_results['speedup_factor']:.2f}x")
```

if **name** == "**main**": asyncio.run(main())