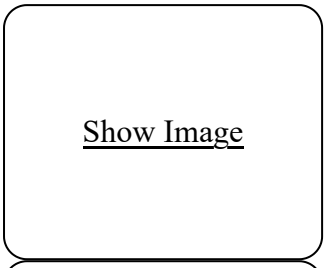
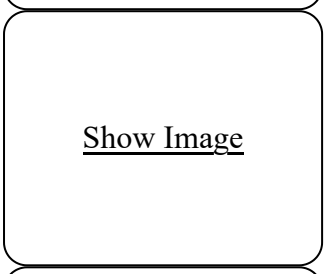
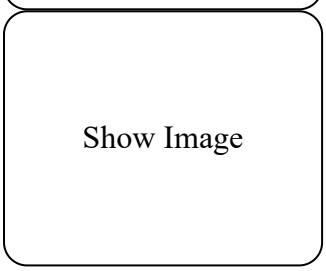
OmniNexus: Fragmergent Adaptive Systems

A research platform for oscillator-driven hybrid architectures with coherence-based mode switching and phase-coupled reinforcement learning.







🌟 What is OmniNexus?

OmniNexus is a novel adaptive system that operates at the **"edge of chaos"** - the fragmergent zone where systems exhibit optimal adaptability. It combines:

- **Multi-harmonic oscillators** for rhythmic phase generation
- **Reinforcement learning** with phase-coupled policy updates
- **Dual-mode architecture** (optical/digital) with intelligent switching
- **Procedural world generation** driven by agent performance
- **Real-time analysis** tools (phase space, Fourier, multi-agent comparison)

Key Innovation: Fragmergent Systems

Fragmergent = *Fragile* + *Emergent*

The system maintains dynamic equilibrium between order and chaos, enabling:

- Adaptive mode switching based on coherence metrics
- Energy-efficient operation through intelligent regeneration decisions
- Natural, biologically-inspired learning rhythms

🚀 Quick Start

Python Implementation

```bash
# Install dependencies
pip install -r core/python/requirements.txt

# Run basic demo
python core/python/v10_research_grade/demo.py

# Run with visualization
python core/python/v10_research_grade/demo.py --visualize
```

Web Interactive Explorer

```bash
# Open in browser
open core/web/index.html

# Or serve locally
python -m http.server 8000
# Navigate to http://localhost:8000/core/web/
```

📊 Repository Structure

```
omninexus-fragmergent-system/
├── core/
│   ├── python/          # Python implementations
│   │   ├── v9_proof_of_concept/    # Original implementation
│   │   └── v10_research_grade/     # Enhanced version (recommended)
│   └── web/             # Interactive browser-based explorer
│
├── docs/                # Comprehensive documentation
│   ├── 01_CONCEPT.md
│   ├── 02_ARCHITECTURE.md
│   ├── 03_MATHEMATICS.md
│   ├── 04_APPLICATIONS.md
│   └── 05_PATENT_ANALYSIS.md
│
├── tests/               # Test suite and validation
```

```
├── examples/          # Usage examples
└── research/          # Theoretical foundations
```

🎯 Core Components

1. Fragmergent Oscillator

Multi-harmonic phase generator with coherence tracking:

```python
oscillator = FragmergentOscillator(
    base_freq=0.1,
    harmonic_layers=3,
    noise=0.05
)
phi = oscillator.step()
coherence = oscillator.get_phase_coherence()
```

2. Optical World

FFT-based procedural world with adaptive complexity:

```python
world = OpticalWorld(size=(128, 128), complexity=0.001)
world.generate()
region = world.sample_region(pos=(64, 64), size=5)
```

3. Avatar

Energy-constrained agent with smart navigation:

```python
avatar = Avatar(world_size=(128, 128))
avatar.move('up', world_size)
avatar.interact(region)
```

4. RL Agent

Phase-coupled policy learning with adaptive rates:

```python
```

```
agent = RLAgent(learning_rate=0.015)
should_regenerate = agent.decide(phi, coherence)
agent.update(reward, phi)
```

5. OmniNexus Core

Orchestrator integrating all components:

```
python

nexus = OmniNexus(world_size=(128, 128))
state = nexus.run_cycle(smart_navigation=True)
```

📈 Version Evolution

v9 → v10: What Changed?

Feature	v9 (Proof of Concept)	v10 (Research Grade)
Oscillator	Single harmonic	Multi-harmonic (1-7 layers)
Coherence	Not tracked	Dynamic metric with history
Navigation	Random walk	Smart resource-seeking
Learning Rate	Fixed	Adaptive based on variance
Reward Function	3-component	5-component with richness
Visualization	Matplotlib static	Comprehensive dashboard

Recommendation: Use **v10** for new projects. v9 is preserved for reference and educational purposes.

🔬 Research Applications

Phase Space Analysis

Detect attractors, limit cycles, and chaotic behavior:

```
python
```

```
from core.python.v10_research_grade.analysis import analyze_phase_space

attractors = analyze_phase_space(agent.phiHistory, agent.policyHistory)
```

Fourier Analysis

Identify dominant frequencies and harmonics:

```
python

spectrum = analyze_fourier(agent.phiHistory)
dominant_freqs = find_peaks(spectrum, n=5)
```

Multi-Agent Competition

Compare different strategies:

```
python

agents = [
    Agent(id=0, preset='stable'),
    Agent(id=1, preset='chaotic'),
    Agent(id=2, preset='resonant')
]
winner = run_competition(agents, steps=500)
```

🎨 Presets Library

Six scientifically designed configurations:

Preset	Frequency	Harmonics	Noise	Learning Rate	Best For
Stable	0.05	2	0.01	0.005	Predictability, convergence
Chaotic	0.35	5	0.15	0.03	Exploration, adaptation
Resonant	0.15	3	0.05	0.015	Balanced performance
Exploration	0.20	4	0.10	0.04	High variability tasks
Minimal	0.08	1	0.0	0.01	Baseline comparison
Quantum	0.12	6	0.08	0.02	Multi-frequency interference

💡 Practical Applications

1. Robotics

- **Energy-efficient path planning**: Switch between full replanning (optical) and path following (digital)
- **Adaptive exploration**: Balance exploration vs. exploitation based on coherence

2. Gaming & Metaverse

- **Procedural worlds**: Regenerate content when agent performance indicates staleness
- **Adaptive difficulty**: Adjust challenge based on player learning curve

3. Trading & Finance

- **Regime detection**: Switch strategies when market coherence changes
- **Portfolio rebalancing**: Optical mode = full reoptimization, Digital = incremental adjustments

4. Smart Cities

- **Traffic optimization**: Recalculate routes (optical) vs. follow current plan (digital)
- **Energy grids**: Balance load redistribution frequency with efficiency

5. IoT & Sensor Networks

- **Power management**: Low-power sensing (digital) with periodic high-power analysis (optical)
- **Anomaly detection**: Coherence drops trigger deeper investigation

---

📖 Documentation

Comprehensive guides available in `/docs`:

1. **CONCEPT.md** - System overview and philosophy
2. **ARCHITECTURE.md** - Technical design and components
3. **MATHEMATICS.md** - Theoretical foundations
4. **APPLICATIONS.md** - Use cases and examples
5. **PATENT_ANALYSIS.md** - IP strategy and novelty

🧪 Testing & Validation

Run the complete test suite:

```bash
cd tests/
pytest -v
```

Validation outputs demonstrate:

- ✅ Single agent convergence
- ✅ Multi-agent competition
- ✅ Phase space trajectory analysis
- ✅ Fourier spectrum detection
- ✅ Long-term stability (300+ steps)

🤝 Contributing

We welcome contributions! See CONTRIBUTING.md for guidelines.

Areas of interest:

- New preset configurations
- Application-specific adaptations
- Performance optimizations
- Additional analysis tools
- Real-world case studies

📜 License

This project is licensed under the MIT License - see LICENSE file.

Patent Notice: Certain novel aspects of this system are subject to pending patent applications. See PATENT_STRATEGY.md for details.

📚 Citation

If you use OmniNexus in your research, please cite:

```bibtex
```

```
@software{omninexus2025,
  title = {OmniNexus: Fragmergent Adaptive Systems with Oscillator-Driven Hybrid Architecture},
  author = {Lucian Coman},
  year = {2025},
  url = {https://github.com/yourusername/omninexus}
}
```

🔗 Links

- **Documentation**: Full Docs
- **Live Demo**: Interactive Explorer
- **Examples**: Usage Examples
- **Research**: Theoretical Background

🌟 Acknowledgments

Built on foundational concepts from:

- Complex Systems Theory
- Reinforcement Learning
- Signal Processing (FFT)
- Procedural Generation
- Biologically-Inspired Computing

📞 Contact

Author: Lucian Coman
Location: Römerswil, Switzerland
Specialization: AI Consciousness Architectures & Post-Transformer Neural Systems

For collaborations, licensing inquiries, or research partnerships, please open an issue or contact directly.

Version: 10.0.0
Last Updated: January 2025
Status: Active Research & Development