

OpenSense-NeuroSpark: Architectural Integration Specification

Why "opensense-neurospark"?

Repository: `github.com/obinexus/opensense-neurospark`
Parent Framework: `github.com/obinexus/functor-framework`
Integration Point: BCI + Puppet Protocol + Semantic Coherence

Naming Etymology

OpenSense

Open = Open-source, open-access computing framework
Sense = Sensory modalities (visual/eze, auditory/uche, motor, tactile/puppet)

Refers to:

- Open-source sensory relay systems
- Puppet Protocol tactile/haptic feedback
- Multi-modal sensory input (EEG + physical interaction)
- Accessible, transparent BCI hardware (<\$3 puppet components)

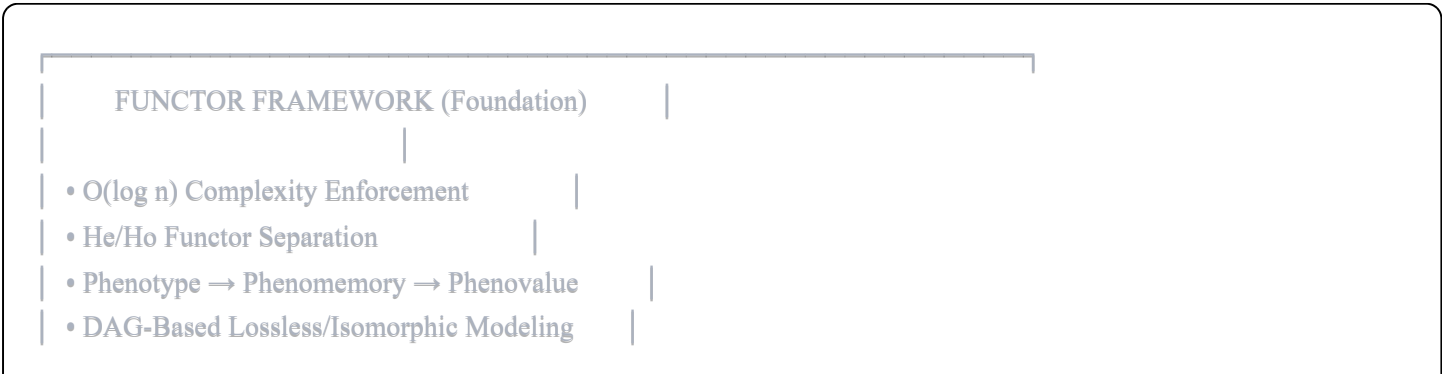
NeuroSpark

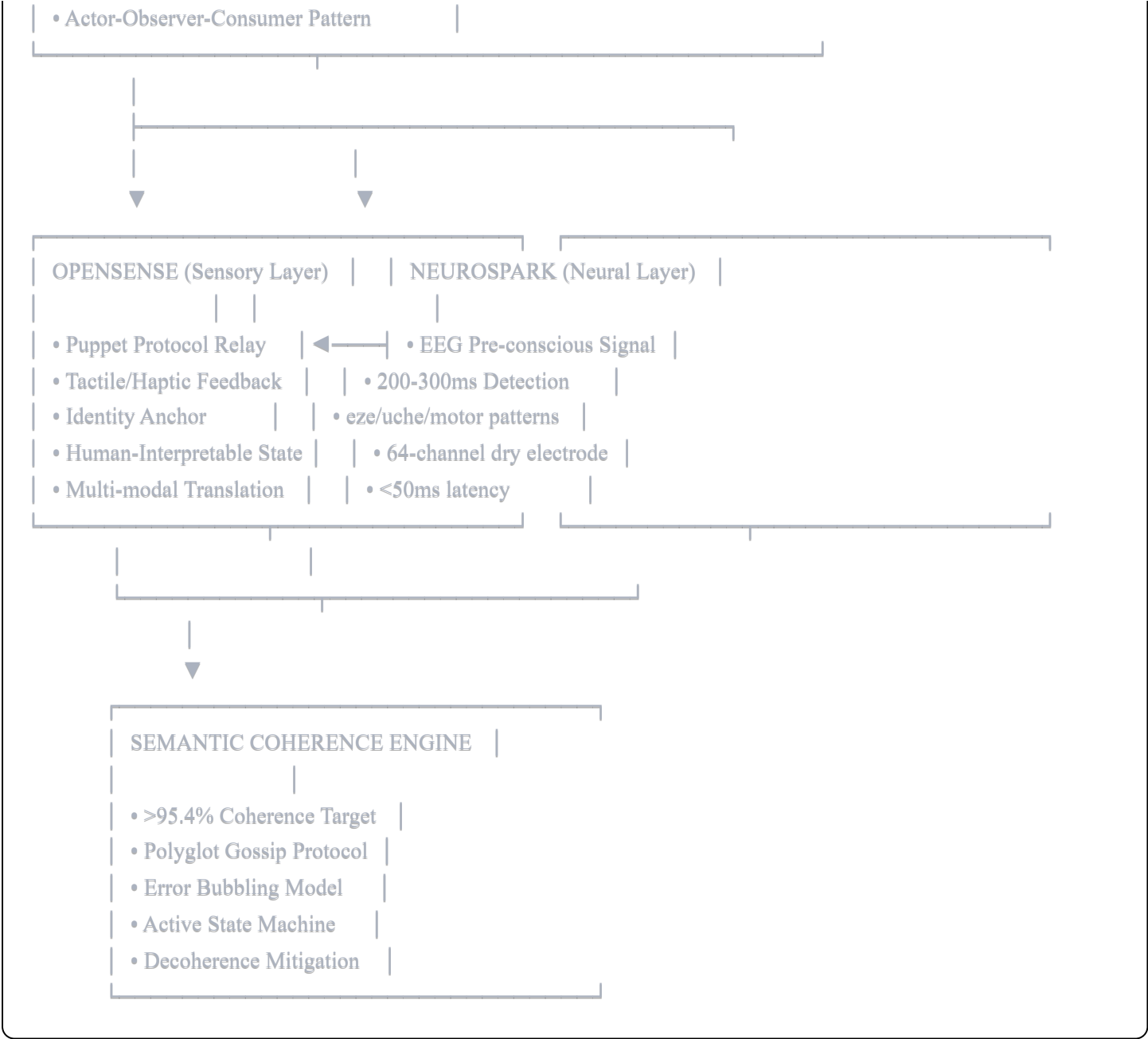
Neuro = Neurological/neural activity
Spark = Pre-conscious neural ignition (200-300ms before awareness)

Refers to:

- Neural detection before thought consolidation
- "Spark" = the moment intention forms (pre-conscious window)
- Real-time neural pattern classification
- The "ignition phase" of cognition

Three-System Integration





Why This Integration Matters

Problem Statement

Traditional BCI systems fail at semantic coherence because:

1. **Neural layer** (EEG) detects signals with high temporal precision
2. **Computational layer** (actors/software) processes at different timescales
3. **Human interpretation layer** requires context that's lost in translation

Result: Semantic drift — the meaning changes as context flows across boundaries.

Solution: Functor Framework + OpenSense + NeuroSpark

Functor Framework provides:

- **$O(\log n)$ auxiliary space** — prevents timeout/degradation

- **He \supset Ho containment** — heterogeneous (mixed-type) functors contain homogeneous (single-type)
- **Lossless DAG transformations** — no information loss during temporal processing
- **Isomorphic DAG transformations** — structure-preserving spatial mappings

NeuroSpark provides:

- **Pre-conscious detection** — catches neural "spark" before thought formation
- **Multi-modal patterns** — eze (visual), uche (internal dialogue), motor intent
- **High-fidelity capture** — 64 channels @ 2048 Hz
- **Real-time classification** — <50ms latency

OpenSense provides:

- **Puppet Protocol relay** — physical identity anchor for semantic grounding
- **Tactile feedback** — human-interpretable state changes
- **Sovereignty framework** — no external override of user's expression
- **Anti-coercion design** — system amplifies, never corrects

Repository Structure Explained

Why Functor Framework is Parent

```

functor-framework/      ← Foundation (O(log n) enforcement)
├── iaas/                ← Design protocol layer
│   ├── architecture-principles/ ← HOW (O(log n) DAG patterns)
│   ├── hdis-topology/      ← HDIS integration
│   └── separation-of-concerns/ ← 5W1H framework
├── baas/                ← Implementation execution layer
│   ├── implementation-layer/ ← Actual code that runs
│   ├── qa-matrices/        ← Validation for phenomemory
│   └── test-harness/       ← Compile-time complexity checks
└── applications/        ← Domain-specific implementations
    ├── opensense-neurospark/ ← BCI + Puppet Protocol application
    │   ├── neurospark/      ← Neural detection (BCI hardware/firmware)
    │   ├── opensense/       ← Sensory relay (Puppet Protocol)
    │   ├── coherence-engine/ ← Semantic decoherence mitigation
    │   └── actor-system/     ← Polyglot gossip programming
  
```

Why OpenSense-NeuroSpark is Child

opensense-neurospark is a domain-specific application of functor framework principles:

1. **Inherits $O(\log n)$ complexity enforcement** — prevents BCI system degradation

- 2. **Uses phenotype model** — Neural signal (phenotype) → Captured pattern (phenomemory) → Action intent (phenovalue)
- 3. **Implements He/Ho separation** — Heterogeneous neural data + homogeneous classification output
- 4. **Leverages error bubbling** — Errors bubble UP from BCI → Puppet → Actor, not propagate down
- 5. **Enforces DAG structure** — No cycles in signal processing pipeline (prevents infinite loops)

Technical Architecture Mapping

Functor Framework → OpenSense-NeuroSpark

Functor Concept	OpenSense-NeuroSpark Implementation
Phenotype	Raw EEG signal (64 channels, 2048 Hz)
Phenomemory	Classified neural pattern (eze/uche/motor)
Phenovalue	Puppet action + actor system state
He (Heterogeneous Functor)	Mixed-type processing: EEG → Pattern → Action
Ho (Homogeneous Functor)	Single-type: Pattern classification (CNN/RNN)
O(log n) Auxiliary Space	DAG-based signal processing (no unbounded memory)
Actor-Observer-Consumer	Driver (BCI) → Watcher (Puppet) → Consumer (User)
Error Bubbling	Neural error → Puppet validation → Human-in-loop
Lossless DAG	Temporal coherence (neural timing preserved)
Isomorphic DAG	Spatial coherence (pattern structure preserved)

Semantic Coherence: The Core Problem

What is Semantic Decoherence?

Definition: Loss of meaning as context flows across system boundaries.

Example:

Neural Signal (BCI Layer)



[Translation Loss #1: Time granularity mismatch]



Pattern Classification (Computational Layer)



[Translation Loss #2: Context stripping]



Puppet Action (Physical Layer)



[Translation Loss #3: Human interpretation drift]



User Understanding (Cognitive Layer)

Result: Original neural intent \neq Final interpreted meaning

How Functor Framework Prevents This

Lossless DAG Transformation (Temporal Coherence):

```
rust

// Neural signal captured with full temporal context
struct NeuralPhenotype {
    timestamp: NanosecondPrecision, // O(1) storage
    channels: [f32; 64],             // O(1) storage
    context: TemporalWindow,         // O(log n) bounded history
}

// Transformation preserves time information
fn observe(pheno: NeuralPhenotype) -> Phenomemory {
    // DAG guarantees: no cycles, O(log n) traversal
    classify_pattern_with_context(pheno)
    // ↑ Context preserved through transformation
}

// Consumer action includes full provenance
fn consume(memory: Phenomemory) -> PuppetAction {
    // Isomorphic mapping: structure preserved
    map_to_puppet_with_origin(memory)
    // ↑ Original neural signal traceable
}
```

Key Insight: By enforcing **O(log n) auxiliary space**, the functor framework prevents unbounded context accumulation that leads to timeout → degradation → semantic drift.

Polyglot Gossip Programming

What is Polyglot Gossip?

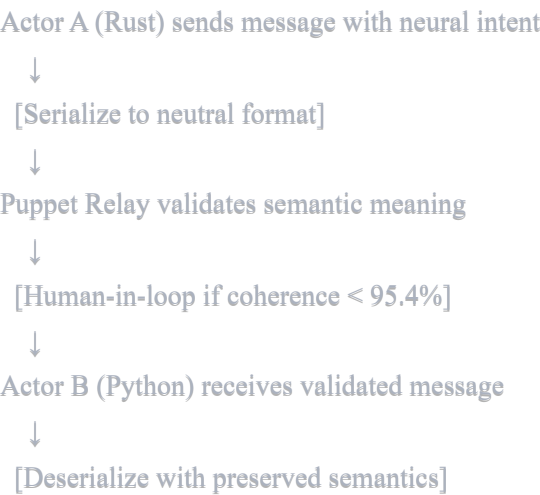
Polyglot: Multiple programming languages (Rust, Python, C, Gosilang)

Gossip: Actor-based message passing with semantic validation

Problem: When actors communicate across language boundaries, semantic meaning can drift due to:

- Different type systems
- Varied concurrency models
- Inconsistent error handling

Solution: Puppet Protocol as Semantic Checkpoint



The puppet acts as a physical semantic anchor — if the puppet's action doesn't match the expected neural intent, the system knows semantic drift occurred and pauses for human validation.

Error Bubbling vs Error Propagation

Traditional Error Propagation (WRONG)

c

```

// Error propagates DOWN to children
int main() {
    int result = clamp(150, 0, 100);
    if (result == ERROR) {
        // Error propagated down to here
        handle_error(); // But we lost context!
    }
}

int clamp(int value, int min, int max) {
    int lower = max(value, min);
    if (lower == ERROR) {
        return ERROR; // Propagate down
    }
    int clamped = min(lower, max);
    if (clamped == ERROR) {
        return ERROR; // Propagate down
    }
    return clamped;
}

```

Problem: When error propagates down, the **calling context** (main) loses information about **where** the error occurred (was it in max? or min?).

Functor Framework Error Bubbling (CORRECT)

c

```

// Error bubbles UP to parent (witness)
int main() {
    // This is the witness/parent context
    int result = clamp(150, 0, 100);

    // If error occurs in max() or min(),
    // it bubbles HERE with full stack trace
    if (result < 0) {
        // We know EXACTLY where error occurred
        print_error_trace(); // Full context preserved
    }
}

int clamp(int value, int min, int max) {
    // Errors from these calls bubble to main
    int lower = max(value, min);
    int clamped = min(lower, max);
    return clamped;
}

```

Why This Matters for BCI:

When a **neural classification error** occurs:

1. Error bubbles UP from pattern classifier → puppet relay → actor system
2. **Puppet relay** (human-in-loop layer) receives error with full context
3. Human validates: "Did the puppet action match my intent?"
4. System learns corrected mapping and continues

Without error bubbling: Silent segment fault — system crashes with no way to recover because context was lost.

Active State Machine vs Passive State Machine

Passive State Machine (Traditional)

python


```

class TraditionalStateMachine:
    def __init__(self):
        self.state = "IDLE"

    def transition(self, event):
        if self.state == "IDLE" and event == "START":
            self.state = "RUNNING"
        elif self.state == "RUNNING" and event == "STOP":
            self.state = "IDLE"
        # Machine defines states but doesn't act autonomously

```

Problem: Preserves memory (state variable) but NOT time and space context.

Active State Machine (Functor Framework)

```

python

class ActiveStateMachine:
    def __init__(self):
        self.state_memory = [] # Preserves time
        self.call_stack = [] # Preserves space
        self.observers = [] # Witnesses events

    def transition(self, event):
        # Remember WHEN this happened
        self.state_memory.append({
            'timestamp': now(),
            'event': event,
            'call_stack': self.call_stack.copy()
        })

        # Machine acts autonomously based on history
        if self.should_auto_trigger_next_state():
            self.execute_next_action()

        # Notify observers (error bubbling targets)
        self.notify_observers(event)

    def should_auto_trigger_next_state(self):
        # Analyze history to predict next action
        # This is what makes it "active"
        if len(self.state_memory) > 2:
            pattern = self.detect_pattern()
            return pattern.suggests_next_action()
        return False

```

Key Difference: Active state machine **remembers context** (time + space) and **acts autonomously** based on that context.

BCI Application:

```
python

# When neural pattern detected:
active_sm.transition(NeuralEvent(pattern="eze_excitement"))

# Active state machine remembers:
# - When this pattern last occurred (temporal context)
# - What puppet action was triggered (spatial context)
# - Whether coherence was maintained (validation context)

# Then autonomously decides:
# - Should I amplify this puppet action? (pattern reinforcement)
# - Should I pause for human validation? (coherence < 95.4%)
# - Should I adjust classification threshold? (learning)
```

Coherence Operators in Practice

Coherence Function

```
python
```

```

def calculate_coherence(neural, puppet, actor):
    """
    C: (N × P × A) → [0, 1]

    where:
    - N = neural state space
    - P = puppet action space
    - A = actor computation space
    """

    # Weight different layer alignments
    w1 = 0.5 # Neural ↔ Puppet similarity
    w2 = 0.3 # Puppet ↔ Actor similarity
    w3 = 0.2 # Neural ↔ Actor similarity

    coherence = (
        w1 * semantic_similarity(neural, puppet) +
        w2 * semantic_similarity(puppet, actor) +
        w3 * semantic_similarity(neural, actor)
    )

    return coherence

def semantic_similarity(state_a, state_b):
    """
    Measures semantic overlap using set theory
    """

    # Union: combined semantic space
    union = set(state_a.semantics) | set(state_b.semantics)

    # Intersection: shared meaning
    intersection = set(state_a.semantics) & set(state_b.semantics)

    # Jaccard similarity
    if len(union) == 0:
        return 0.0
    return len(intersection) / len(union)

```

Coherence Enforcement

python

```

# Real-time coherence monitoring
def process_neural_event(eeg_signal):
    # Layer 1: Detect neural pattern
    neural_pattern = neurospark.classify(eeg_signal)

    # Layer 2: Translate to puppet action
    puppet_action = opensense.relay(neural_pattern)

    # Layer 3: Compute actor state
    actor_state = actor_system.process(puppet_action)

    # Calculate coherence
    coherence = calculate_coherence(
        neural_pattern,
        puppet_action,
        actor_state
    )

    # Enforce threshold
    if coherence < 0.954: # 95.4% minimum
        # Trigger human-in-loop validation
        human_validation = puppet_protocol.request_validation(
            neural_pattern,
            puppet_action,
            actor_state
        )

        # Learn from correction
        if human_validation.corrected:
            semantic_model.update(
                original=puppet_action,
                corrected=human_validation.intended_action
            )

    return coherence

```

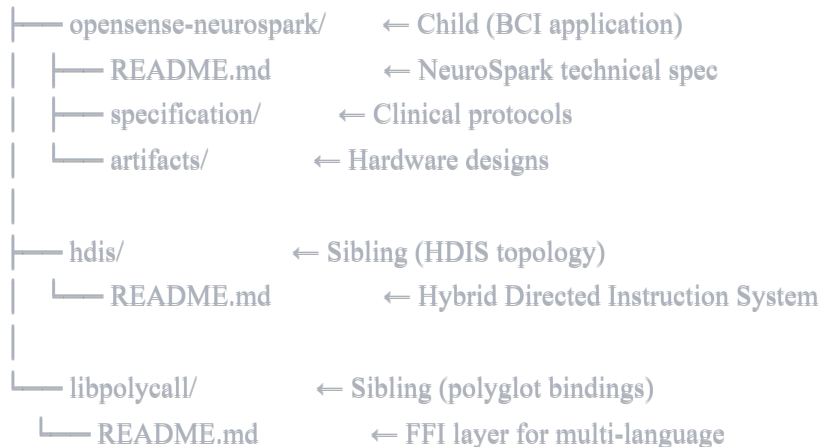
Repository Hierarchy Explained

Why This Structure?

```

github.com/obinexus/
├── functor-framework/    ← Parent (architectural foundation)
│   ├── README.md        ← Philosophy and He/Ho theory
│   ├── ARCHITECTURE_PRINCIPLES.md ← O(log n) enforcement
│   └── core/             ← Rust implementation

```



Inheritance Model:

1. **functor-framework** defines architectural principles (HOW systems should be built)
2. **opensense-neurospark** applies those principles to BCI domain (WHAT system to build)
3. **hdis** provides deployment topology (WHERE system runs)
4. **libpolycall** enables polyglot integration (WHO can use it)

Key Insights Summary

1. Why "OpenSense"?

- **Open** = Open-source, accessible, transparent
- **Sense** = Multi-modal sensory input (EEG + tactile + haptic)
- **Protocol** = Puppet Method for identity anchoring

2. Why "NeuroSpark"?

- **Neuro** = Neurological signal detection
- **Spark** = Pre-conscious ignition (200-300ms window)
- **Innovation** = Catches intent before thought formation

3. Why Functor Framework Parent?

- Provides **O(log n)** complexity enforcement (prevents degradation)
- Defines **He** \supset **Ho** containment (heterogeneous contains homogeneous)
- Implements **phenotype** \rightarrow **phenomemory** \rightarrow **phenovalue** pipeline
- Enforces **error bubbling** (not propagation)
- Guarantees **lossless/isomorphic** DAG transformations

4. Why This Matters for BCI?

Traditional BCI systems fail because:

- No semantic coherence enforcement
- No human-in-loop validation
- No context preservation across layers
- No complexity bounds (leads to timeout/crash)

OpenSense-NeuroSpark succeeds because:

- **Functor framework** prevents degradation via $O(\log n)$
 - **Puppet protocol** provides semantic grounding
 - **Error bubbling** preserves context for recovery
 - **Active state machine** learns from history
 - **Coherence operators** enforce >95.4% semantic alignment
-

Next Steps

For Developers

1. Clone functor-framework first:

```
bash

git clone https://github.com/obinexus/functor-framework
cd functor-framework
cargo build --release
```

2. Then clone opensense-neurospark:

```
bash

git clone https://github.com/obinexus/opensense-neurospark
cd opensense-neurospark
```

3. Link to parent framework:

```
toml

[dependencies]
functor-framework = { path = "../functor-framework" }
```

For Researchers

1. Review **ARCHITECTURE_PRINCIPLES.md** to understand $O(\log n)$ enforcement
2. Study **BCI_Puppet_Protocol_Technical_Specification.md** for clinical protocols
3. Examine **coherence-engine/** for semantic decoherence mitigation algorithms

For Clinicians

1. Read **The_OBINexus_Brain_Interface_and_Relay_Therapy_System.md** for autism therapy use case
 2. Review **Puppet Method Protocol** documentation
 3. Access **artifacts/** for hardware designs and BOM
-

References

Core Papers

- "Functor Framework: Phenomodel-Based Type System" (2025)
- "OpenSense-NeuroSpark: BCI with Puppet Protocol Relay" (2025)
- "Active Brain Computing Interface via Puppet Protocol Relay for Semantic Decoherence Mitigation" (2025)

Related Projects

- [HDIS \(Hybrid Directed Instruction System\)](#)
- [LibPolyCall \(Polyglot FFI\)](#)
- [Constitutional Housing Framework](#)

Video Resources

- [Puppet Protocol Explanation](#)
 - [OBINexus Computing Playlist](#)
-

Conclusion

OpenSense-NeuroSpark is the practical manifestation of **Functor Framework** principles applied to brain-computer interfaces with semantic coherence enforcement. The naming reflects:

- **Open** sensory relay (accessible, transparent)
- **Neurological Spark** detection (pre-conscious signals)
- **Functor** foundation ($O(\log n)$ complexity, He/Ho separation)
- **Semantic** coherence (>95.4% cross-layer alignment)

By integrating these systems, OBINexus creates the first BCI architecture that:


1. Prevents semantic drift through formal coherence operators
2. Preserves human autonomy via puppet protocol relay
3. Scales safely with $O(\log n)$ complexity guarantees
4. Learns from context through active state machines
5. Recovers gracefully via error bubbling

The architecture IS the seed. OpenSense-NeuroSpark is how it grows into practical BCI therapy.

Document Version: 1.0

Date: October 26, 2025

Author: OBINexus Technical Working Group

Status:  Comprehensive Architectural Explanation