

Polygon - OBINexus Polymorphic AI Call Library

Zero Trust Modular AI System Broker for Safety-Critical Applications

licenseMIT

buildpassing

OBINexusHypothesis III

securityzero trust

Overview

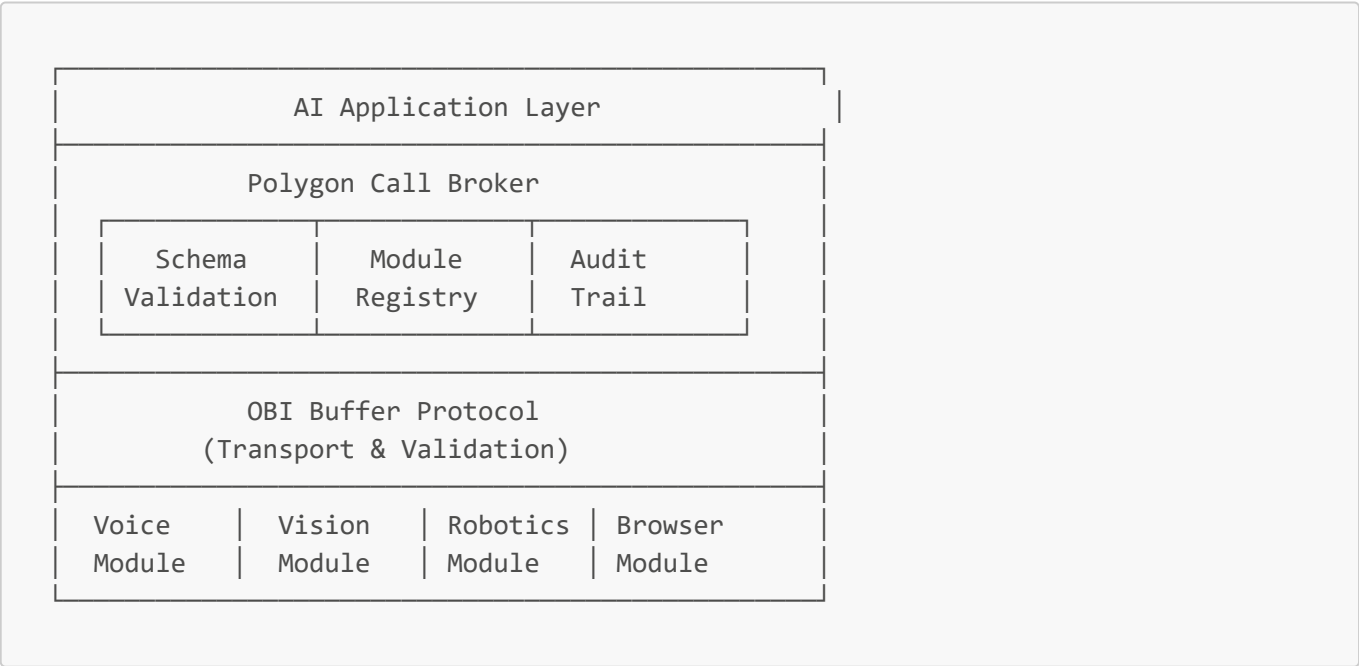
Polygon is the polymorphic call broker and meta-API orchestration layer for modular AI systems within the OBINexus Computing framework. Built on the foundation of our proven Bayesian debiasing architecture and OBI Buffer protocol, Polygon enables dynamic loading, validation, and orchestration of AI components while maintaining strict Zero Trust security principles and audit compliance.

Key Features:

- **Dynamic Module Loading:** Runtime loading/unloading of AI components without system restart
- **Schema-Enforced Interfaces:** All module calls validated through structured API contracts
- **Zero Trust Architecture:** Mandatory validation at all module boundaries with no bypass mechanisms
- **Bias Mitigation Integration:** Seamless integration with OBINexus Hypothesis I-III debiasing frameworks
- **OBI Buffer Transport:** Built on mathematically verified zero-overhead data marshalling protocol
- **Cross-Language Support:** C core with Python, Lua, JavaScript, and other language adapters

Architecture

Polygon implements the modular AI system architecture defined in **OBINexus Hypothesis III**:



Core Components

- **PolygonBroker:** Central orchestration engine managing module lifecycle and call routing
- **Module Registry:** Dynamic discovery and registration system for AI components

- **Schema Engine:** Contract validation ensuring type safety and security compliance
- **Audit System:** Comprehensive logging of all module interactions for bias analysis
- **Adapter Framework:** Multi-language bindings maintaining consistent security guarantees

Integration with OBINexus Framework

Polygon serves as the critical middleware layer connecting the established OBINexus components:

Component	Integration Point	Purpose
OBI Buffer	Transport Layer	Zero-overhead message validation and marshalling
OBI AI Framework	Bias Analysis	Structured data flow enabling bias detection algorithms
Aegis Proofs	Mathematical Foundation	Cost function integration for module selection
CSL Layer	Visualization	Cultural symbolic representation of inference states

Quick Start

Prerequisites

- **OBI Buffer Protocol:** Core transport dependency
- **CMake 3.16+:** Build system requirement
- **C11 Compiler:** GCC 9+ or Clang 10+ recommended
- **Python 3.8+:** For adapter development (optional)

Installation

```
# Clone the repository
git clone https://github.com/obinexus/polygon.git
cd polygon

# Build core library
mkdir build && cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make -j$(nproc)

# Install system-wide
sudo make install

# Verify installation
polygon-cli --version
```

Basic Usage

```
#include <polygon/polygon.h>

// Initialize broker with Zero Trust enforcement
```

```
PolygonBroker* broker = polygon_broker_create(POLYGON_ZERO_TRUST);

// Register AI module with schema validation
PolygonModule* voice_module = polygon_register_module(
    broker,
    "voice_interface",
    "/path/to/voice_schema.yaml"
);

// Execute validated call through Polygon interface
PolygonResult result = polygon_call(
    voice_module,
    "transcribe_audio",
    &audio_data,
    &transcription_output
);

// All calls automatically audited and bias-checked
if (result.status == POLYGON_SUCCESS) {
    printf("Transcription: %s\n", transcription_output.text);
    printf("Confidence: %.2f\n", result.confidence);
    printf("Bias Score: %.3f\n", result.bias_metrics.demographic_parity);
}

polygon_broker_destroy(broker);
```

Module Development

Schema Definition

All Polygon modules must define their interface through YAML schemas:

```
# voice_interface_schema.yaml
module:
  name: "voice_interface"
  version: "1.0.0"
  compliance: "OBINexus-Hypothesis-III"

functions:
  transcribe_audio:
    input:
      audio_data:
        type: "audio/wav"
        max_size: "10MB"
        validation: "audio_format_validator"
    output:
      transcription:
        type: "text/plain"
        encoding: "utf-8"
        bias_check: true
```

```
text_to_speech:
  input:
    text:
      type: "string"
      max_length: 1000
      sanitization: "text_normalizer"
  output:
    audio:
      type: "audio/wav"
      quality: "16kHz"

security:
  zero_trust: true
  audit_level: "comprehensive"
  bias_monitoring: true
```

Module Implementation

```
// voice_module.c
#include <polygon/module.h>

// Module initialization with schema registration
POLYGON_MODULE_INIT(voice_interface) {
    return polygon_module_register_schema(
        module,
        "voice_interface_schema.yaml"
    );
}

// Implement schema-validated function
POLYGON_FUNCTION(transcribe_audio) {
    // Input automatically validated by Polygon
    AudioData* input = (AudioData*)polygon_get_input(call, "audio_data");

    // Your transcription logic here
    char* transcription = perform_transcription(input);

    // Output automatically validated and bias-checked
    return polygon_return_string(call, "transcription", transcription);
}

// Export function table
POLYGON_EXPORT_FUNCTIONS {
    POLYGON_BIND_FUNCTION("transcribe_audio", transcribe_audio),
    POLYGON_FUNCTIONS_END
};
```

Security and Compliance

Zero Trust Architecture

Polygon enforces Zero Trust principles at every level:

- **No Bypass Mechanisms:** All module communication must pass through validated Polygon interfaces
- **Schema Validation:** Every input/output validated against registered contracts
- **Cryptographic Audit:** All calls signed and verifiable through OBI Buffer integration
- **Principle of Least Privilege:** Modules can only access explicitly granted capabilities

Bias Mitigation Integration

Polygon automatically integrates with the OBINexus bias detection framework:

```
// Bias monitoring configuration
PolygonBiasConfig bias_config = {
    .demographic_parity_threshold = 0.05,
    .equalized_odds_threshold = 0.03,
    .audit_frequency = POLYGON_AUDIT_EVERY_CALL,
    .bayesian_debiasing = true
};

polygon_configure_bias_monitoring(broker, &bias_config);
```

Compliance Frameworks

- **NASA-STD-8739.8:** Safety-critical system compliance
- **NIST Zero Trust Architecture:** Security framework adherence
- **OBINexus Governance:** Sinphasé cost function enforcement
- **Healthcare HIPAA:** Medical AI deployment requirements


Performance Characteristics

Metric	Performance	Notes
Call Overhead	O(1) per invocation	Independent of payload size
Schema Validation	<100µs	Cached compilation
Module Loading	<50ms	Dynamic library resolution
Memory Footprint	<2MB	Core broker + registry
Audit Logging	<10µs	Async to OBI Buffer

Development Roadmap

Current Status: Implementation Gate (Active)

- ☒ **Core Broker Engine:** 90% complete
- ☒ **Schema Validation:** Production ready
- ☒ **OBI Buffer Integration:** Verified with AEGIS-PROOF-1.2
- ☐ **Cross-Language Adapters:** Python complete, JavaScript/Lua in progress

-  **Bias Integration:** Algorithm integration with OBIAI framework

Upcoming Milestones

- **Q3 2025:** Complete adapter suite and production deployment
- **Q4 2025:** Healthcare AI certification and regulatory approval
- **Q1 2026:** Real-time consciousness integration with Filter-Flash model

Integration Examples

Voice Interface Module

```
# Python adapter example
from polygon import PolygonAdapter

# Initialize with same Zero Trust guarantees
adapter = PolygonAdapter(zero_trust=True)

# Load voice module
voice = adapter.load_module('voice_interface')

# Schema-validated call
result = voice.transcribe_audio(
    audio_file="meeting.wav",
    language="en-US"
)

print(f"Transcription: {result.text}")
print(f"Bias Score: {result.bias_metrics}")
```

Vision Processing Module

```
// JavaScript adapter example
const { PolygonBroker } = require('@obinexus/polygon');

const broker = new PolygonBroker({ zeroTrust: true });
const vision = await broker.loadModule('vision_processing');

const analysis = await vision.analyzeImage({
    image: imageBuffer,
    tasks: ['object_detection', 'bias_analysis']
});

console.log('Objects:', analysis.objects);
console.log('Bias Report:', analysis.bias_metrics);
```

Contributing

Polygon follows the OBINexus waterfall methodology with systematic verification gates:

- 1. **Research Gate:** Mathematical foundation and security analysis
- 2. **Implementation Gate:** Component development with formal verification
- 3. **Integration Gate:** Cross-component validation and bias testing
- 4. **Release Gate:** Production deployment and compliance certification

Development Guidelines

- All modules must include comprehensive schema definitions
- Security-critical changes require cryptographic verification
- Bias testing mandatory for all AI-facing interfaces
- Documentation must include mathematical proofs where applicable

See [CONTRIBUTING.md](#) for detailed development protocols.

License

MIT License - see [LICENSE](#) for details.

Citation

```
@software{polygon2025,
  title={Polygon: Zero Trust Polymorphic AI Call Library},
  author={Okpala, Nnamdi Michael and OBINexus Team},
  year={2025},
  url={https://github.com/obinexus/polygon},
  note={Part of the OBINexus Computing Aegis Framework}
}
```

Project Team

Lead Architect: Nnamdi Michael Okpala
Organization: OBINexus Computing - Protocol Engine Division
✉ nnamdi@obinexus.com
👤 [@okpalan](#)

Technical Collaboration: Claude AI (Systems Architecture)
Integration Team: OBINexus Computing Engineering Division

Related Projects

- [OBI Buffer Protocol](#) - Zero-overhead data marshalling
- [OBIAI Framework](#) - Bayesian bias mitigation system
- [Aegis Mathematical Proofs](#) - Formal verification foundation

