LUASCRIPT Architecture Specification

System Architecture Overview

LUASCRIPT is architected as a multi-layered system designed to deliver JavaScript familiarity with Mojo-like performance characteristics.

Layer 1: Core Language Engine

Parser & Lexer

- **JavaScript-compatible syntax parsing**
- **Ternary literal support** (-1, 0, +1 values)
- **C inline code recognition**
- **Gaussian CSS syntax integration**

Transpiler

- **JavaScript → Lua transpilation**
- **Performance optimization passes**
- **C code generation pipeline**
- **Memory management insertion**

Runtime System

- **LuaJIT execution engine**
- **JavaScript compatibility layer**
- **C FFI integration**
- **Ternary arithmetic operations**

Layer 2: Performance Optimization

Memory Management

- **Automatic garbage collection** (default)
- **Manual memory control** (opt-in)
- **Memory pool allocation**
- **Leak detection and prevention**

Compilation Pipeline

- **Multi-stage optimization**
- **SIMD instruction generation**
- **Hardware-specific tuning**
- **Parallel execution planning**

C Integration

- **Foreign Function Interface**
- **Inline C code compilation**

- **Library binding generation**
- **System call optimization**

Layer 3: Agentic IDE System

Core IDE Framework

```
LUASCRIPT IDE (Self-Hosted)
├── Editor Engine (LUASCRIPT)
├── Parser Integration (Native)
├── AI Agent System (LUASCRIPT + C)
└── Extension Framework (LUASCRIPT)
```

AI Agent Architecture

- **Code Analysis Agents**: Understand LUASCRIPT semantics
- **Optimization Agents**: Suggest performance improvements
- **Refactoring Agents**: Autonomous code restructuring
- **Generation Agents**: Write LUASCRIPT code from specifications

Self-Modification Capabilities

- **IDE can update its own source code**
- **Agents can improve agent algorithms**
- **Continuous self-optimization**
- **Version control integration for changes**

Layer 4: Ternary Computing System

Data Types

```
// Balanced ternary primitives
trit: -1 | 0 | 1
tryte: trit[6]  // 6 trits = 729 values
tword: trit[27] // 27 trits = large range
```

Operations

- **Ternary arithmetic**: Addition, multiplication in base-3
- **Ternary logic**: AND, OR, NOT operations
- **Conversion functions**: Binary ↔ Ternary
- **Quantum-ready algorithms**: Superposition-like states

Applications

- **Signed number representation** (more natural)
- **Fuzzy logic systems**
- **Quantum algorithm simulation**
- **Novel data structures**

Layer 5: CSS Evolution Pipeline

Gaussian CSS Engine

```css
/* Gaussian distribution-based styling */
.element {
    gaussian-margin: normal(10px, 2px);
    gaussian-opacity: bell-curve(0.8, 0.1);
    transition: gaussian-ease(300ms);
}
```

GSS Specification

- **Mathematical function library**
- **Distribution-based layouts**
- **Probabilistic responsive design**
- **Performance-optimized rendering**

AGSS (Agentic) System

- **AI design agents**
- **Context-aware styling**
- **Adaptive layout optimization**
- **Generative design systems**

Integration Architecture

Component Interaction

```
┌─────────────────┐    ┌─────────────────┐
│   JavaScript    │───▶│   LUASCRIPT     │
│   Developer     │    │   Parser        │
└─────────────────┘    └─────────────────┘
                                │
                                ▼
┌─────────────────┐    ┌─────────────────┐
│   Agentic IDE   │◀───│   Transpiler    │
│   (Self-Hosted) │    │   Engine        │
└─────────────────┘    └─────────────────┘
                                │
                                ▼
┌─────────────────┐    ┌─────────────────┐
│   C Libraries   │◀───│   LuaJIT        │
│   & System      │    │   Runtime       │
└─────────────────┘    └─────────────────┘
```

Data Flow

1. **Source Code**: JavaScript-like LUASCRIPT syntax
2. **Parsing**: AST generation with ternary and C support
3. **Optimization**: Multi-pass performance enhancement
4. **Transpilation**: Lua code generation with C integration
5. **Execution**: LuaJIT runtime with native performance
6. **IDE Feedback**: AI agents analyze and suggest improvements

Performance Targets

Execution Speed

- **Numeric computation**: 90% of C performance
- **String operations**: 80% of C performance
- **System calls**: 95% of C performance
- **Memory allocation**: 85% of C performance

Development Experience

- **Compilation time**: < 100ms for typical modules
- **IDE responsiveness**: < 50ms for code analysis
- **Agent suggestions**: < 200ms for optimization hints
- **Hot reload**: < 10ms for code changes

Scalability Architecture

Horizontal Scaling

- **Multi-core execution**: Automatic parallelization
- **Distributed computing**: Built-in clustering support
- **GPU acceleration**: CUDA/OpenCL integration
- **Cloud deployment**: Container-ready runtime

Vertical Scaling

- **Memory efficiency**: Minimal runtime overhead
- **CPU optimization**: Hardware-specific tuning
- **I/O performance**: Async/await with native speed
- **Resource management**: Intelligent allocation strategies

This architecture specification serves as the technical blueprint for achieving the grand LUASCRIPT vision while maintaining practical implementation pathways.