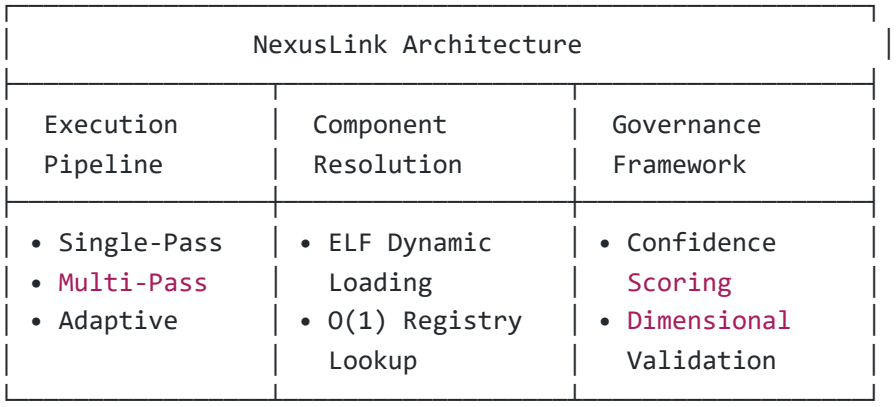
# NexusLink CLI System: Component Orchestration Framework

## 1. System Architecture Overview

NexusLink implements a deterministic component orchestration framework that enables reliable execution pipelines through hierarchical isolation protocols and dimensional homogeneity validation. The system enforces Sinphasé development constraints to maintain single-pass compilation requirements while enabling seamless transition between Human-In-The-Loop (HITL) and Human-Out-The-Loop (HOTL) execution models.




### 1.1 Key Architectural Principles

- **Deterministic Execution**: Pipeline stages execute in precisely defined sequences with predictable resource utilization patterns
- **Component Isolation**: Each component maintains strict boundaries through explicit interface contracts
- **Dimensional Homogeneity**: Components share consistent processing patterns within strategic dimensions
- **Dynamic Confidence Tracking**: Real-time monitoring of component execution metrics to validate automation readiness

## 2. Interactive Command Interface

NexusLink provides an interactive command shell for dynamic component orchestration:

```
$ nlink --interactive

****************************************
```

```
*         NexusLink CLI System           *
*          © OBINexus Computing          *
****************************************
Type 'help' for available commands, 'exit' to quit

nexus> load tokenizer
Loading component 'tokenizer'...
Successfully loaded component 'tokenizer'

nexus> load parser
Loading component 'parser'...
Successfully loaded component 'parser'

nexus> pipeline create mode=single
Created pipeline in single-pass mode with optimization enabled

nexus> pipeline add-stage tokenizer
Added stage 'tokenizer' to pipeline

nexus> pipeline add-stage parser
Added stage 'parser' to pipeline

nexus> pipeline execute
Running pipeline optimizations...
Executing pipeline...
Pipeline executed successfully in 45.23 ms with 1 iteration(s)

nexus> stats
System Statistics:
-----------------
  Components loaded: 2
  Memory usage: 0.8 MB
  Heap allocations: 73
  Peak memory: 1.2 MB
  Symbol table entries: 128
  Commands registered: 7
  Pipelines active: 1

nexus> exit
```

## 3. Scripted Pipeline Execution

For automated workflows, NexusLink supports deterministic script execution:

```
# script.nlink - Tokenizer/Parser single pass pipeline
# Load required components
load tokenizer
load parser

# Create a single-pass pipeline
```

```
pipeline create mode=single

# Add processing stages in sequence
pipeline add-stage tokenizer
pipeline add-stage parser

# Execute the pipeline
pipeline execute

# Display statistics
stats
```

Execute with:

```
$ nlink --execute script.nlink
```

Output:

```
Loading component 'tokenizer'...
Successfully loaded component 'tokenizer'
Loading component 'parser'...
Successfully loaded component 'parser'
Created pipeline in single-pass mode with optimization enabled
Added stage 'tokenizer' to pipeline
Added stage 'parser' to pipeline
Running pipeline optimizations...
Executing pipeline...
Pipeline executed successfully in 43.88 ms with 1 iteration(s)
System Statistics:
-----------------
  Components loaded: 2
  Memory usage: 0.8 MB
  Heap allocations: 73
  Peak memory: 1.2 MB
  Symbol table entries: 128
  Commands registered: 7
  Pipelines active: 1
```

# 4. Component Architecture

NexusLink implements the NS-1.0 (NLink Standard) component specification, providing:

## 4.1 Component Identity Requirements

Components maintain strict isolation boundaries while enabling dynamic composition:

```
struct ComponentIdentity {
    std::string name;                 // Unique identifier
    std::string elfPath;              // Runtime-loadable artifact path
    float confidenceScore;            // ψ: Automation readiness metric
    std::string dimensionalClass;     // D-offensive, D-defensive, D-tactical
    std::string processingPattern;    // Homogeneous pattern signature
    bool isolationRequired;           // Sinphasé isolation state
};
```

## 4.2 Directory Structure Protocol

```
PROJECT_ROOT/
├── components/           # Primary component repository
│   ├── tokenizer.elf     # HOTL-ready components
│   ├── parser.elf
│   └── analyzer.elf
├── src/                  # Source implementation for non-isolated components
│   ├── core/
│   │   ├── tokenizer/    # Original component implementation
│   │   │   ├── main.c
│   │   │   ├── utils.h
│   │   │   └── Makefile  # Independent build system
│   │   └── parser/
│   └── feature_a/
└── root-dynamic-c/       # Isolated components (exceeded cost threshold)
    └── validator-isolated-20250729/  # Component requiring isolation
        ├── src/
        ├── include/
        ├── Makefile      # Standalone build system
        └── ISOLATION_LOG.md
```

## 4.3 Available Commands

| Command | Description |
|---|---|
| `load <component>` | Loads a .elf component into memory |
| `unload <component>` | Unloads a component to free memory |
| `pipeline create mode=X` | Creates a new pipeline (single or multi) |
| `pipeline add-stage <X>` | Appends a stage (component) to current pipeline |
| `pipeline execute` | Executes pipeline with optimization |
| `pipeline export` | Outputs pipeline structure as JSON or DOT |
| `stats` | Show system performance and diagnostics |

| Command | Description |
|---|---|
| `components` | Lists all currently loaded components |
| `reset` | Unloads all components and clears state |
| `exit` | Exit the CLI |

# 5. Execution Pipeline Architecture

NexusLink implements a dual-axis pipeline execution model that maps directly to the HITL/HOTL gating framework:

## 5.1 Single-Pass Mode (Row-Oriented)

Single-pass mode executes tasks across a single execution phase (todo→doing→done), maintaining phase cohesion while traversing strategic dimensions:

```
tokenizer → parser → analyzer → validator
```

This mode optimizes for sequential data transformation where each component operates on the output of the previous stage.

## 5.2 Multi-Pass Mode (Column-Oriented)

Multi-pass mode processes a specific strategic dimension through all execution phases, maintaining dimensional integrity:

```
tokenizer(phase1) → tokenizer(phase2) → tokenizer(phase3)
parser(phase1) → parser(phase2) → parser(phase3)
```

This mode enables dimensional consistency when processing requirements span multiple execution phases.

# 6. Integration Capabilities

NexusLink integrates with diverse execution environments:

| Environment | Integration Method |
|---|---|
| Docker | `FROM obinexus/nlink:latest` |
| CI/CD | `nlink --execute pipeline.nlink` |

| Environment | Integration Method |
|---|---|
| FreeBSD | Components as shared objects |
| Web Assembly | Specialized ELF translation layer |
| Embedded | Static linking with `-lnlink_static` |

# 7. Component Evolution Framework

NexusLink implements systematic governance protocols for component evolution:

## 7.1 Automation Status Classification

Components transition between three discrete automation states:

| Confidence (ψ) | Homogeneity | Sinphasé Valid | Status |
|---|---|---|---|
| ≥ 0.8 | ✓ | ✓ | HOTL_READY |
| 0.6-0.79 | ✓ | ✓ | SUPERVISED_HOTL |
| < 0.6 or ✗ | ✗ | ✗ | HITL_REQUIRED |

## 7.2 Confidence Calculation

```
float calculateConfidence(const PipelineStage& stage) {
    // Symbol complexity from pattern-matching trie depth
    float symbolComplexity = calculateSymbolComplexity(stage.getComponentName());

    // Relationship reliability from dimensional coherence
    float relationReliability = calculateRelationshipReliability(stage);

    // Temporal stability from execution variance
    float temporalStability = calculateTemporalStability(stage);

    return (symbolComplexity * 0.3f) +
           (relationReliability * 0.4f) +
           (temporalStability * 0.3f);
}
```

# 8. Future Development Roadmap

The NexusLink roadmap includes critical advancements in component orchestration:

- **Predictive Isolation**: Implementation of Dynamic Mutation Forecast Model to anticipate threshold violations

- **Advanced Pattern Classification**: Enhanced RegexTrieClassifier with neural pattern recognition
- **Real-Time Confidence Recalibration**: Dynamic confidence scoring based on execution feedback
- **Multi-Dimensional Validation**: Extension of homogeneity validation across additional strategic dimensions

## 9. Documentation Structure

Comprehensive documentation is available in the `docs/` directory:

```
docs/
├── usage/
│   ├── basic.md            # Simple CLI usage
│   ├── pipelines.md        # Pipeline architecture & config
│   ├── scripting.md        # .nlink scripting format
│   ├── components.md       # Creating & loading ELF components
│   └── governance.md       # HITL/HOTL gating explained
├── api/
├── architecture.md
├── examples/
└── CHANGELOG.md
```

For detailed component specifications, see `docs/usage/*.md`.

## 10. Architectural Governance

NexusLink enforces Sinphasé development constraints through automated cost-based governance:

```
float calculateDynamicCost(const Pipeline& pipeline) {
    float cost = 0.0f;

    // Calculate weighted metrics
    for (const auto& metric : {
        pipeline.getIncludeDepth() * 0.15f,
        pipeline.getFunctionCallCount() * 0.20f,
        pipeline.getExternalDependencyCount() * 0.25f,
        pipeline.getComplexityScore() * 0.20f,
        pipeline.getLinkDependencyCount() * 0.20f
    }) {
        cost += metric;
    }

    // Add circular dependency penalty if detected
    if (pipeline.hasCircularDependencies()) {
        cost += 0.2f;
    }

    // Add temporal pressure component
```

```
        cost += pipeline.getTemporalPressure();

        return cost;
    }
```

When cost exceeds the 0.6 threshold, the system automatically initiates component isolation to maintain architectural integrity.