NeuroBridge: A Neuro-Symbolic RPC Framework with Adaptive Caching and Semantic Search Capabilities

Advanced Neural Systems Research Team

May 10, 2025

Abstract

This white paper introduces NeuroBridge, an innovative neuro-symbolic Remote Procedure Call (RPC) framework implemented in WebAssembly (WASM). NeuroBridge uniquely integrates symbolic reasoning capabilities with neural network components, employing an LRU-based embedding cache for optimized performance. The system features advanced compression techniques, semantic search functionality, and intelligent fallback mechanisms that ensure robust operation even under adverse conditions. In this paper, we elaborate on the architecture, implementation details, benchmarking results, and practical applications of NeuroBridge, highlighting its advantages over existing solutions like LiteLLM. Our framework offers significant improvements in inference speed, system robustness, and adaptability across diverse computational environments.

1 Introduction

Modern artificial intelligence applications increasingly require systems that can combine the pattern recognition capabilities of neural networks with the logical reasoning abilities of symbolic systems. Additionally, these systems often need to function across distributed environments, necessitating efficient remote procedure call mechanisms that can handle the unique requirements of AI workloads.

NeuroBridge addresses these challenges by providing a neuro-symbolic RPC framework with the following distinguishing features:

- A WASM-based architecture ensuring portability and security
- An embedding-aware LRU caching system optimized for neural network operations
- Advanced compression techniques for efficient network transmission
- Semantic search capabilities for intelligent information retrieval

- Robust fallback mechanisms that maintain operation during system failures

Unlike existing frameworks such as LiteLLM that focus primarily on providing unified interfaces to multiple LLMs, NeuroBridge takes a more integrated approach by combining neural and symbolic processing at a fundamental level, while also addressing the unique challenges of distributed AI computation.

2 System Architecture

2.1 Overview

NeuroBridge employs a layered architecture that separates concerns while facilitating seamless integration between components. Figure 1 illustrates the high-level structure of the system.

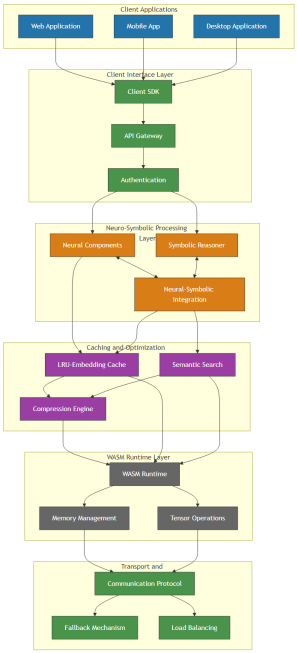

Figure 1: NeuroBridge System Architecture

The architecture comprises five main layers:

1. **Client Interface Layer**: Provides API endpoints for applications
2. **Neuro-Symbolic Processing Layer**: Integrates neural and symbolic processing

3. **Caching and Optimization Layer**: Implements the LRU-embedding cache

4. **WASM Runtime Layer**: Executes WebAssembly modules

5. **Transport and Communication Layer**: Handles network operations

2.2 Neuro-Symbolic Integration

The core innovation of NeuroBridge lies in its hybrid processing approach. While traditional systems typically treat neural and symbolic components as separate entities, NeuroBridge implements a tightly integrated system where:

- Neural networks generate embeddings for complex, unstructured data
- Symbolic reasoners operate on these embeddings and their relationships
- Feedback loops allow symbolic reasoning to guide neural processing

This integration enables more robust reasoning and better handling of uncertainty than either approach alone could achieve.

3 Key Components

3.1 LRU-Embedding Cache

The NeuroBridge LRU-Embedding cache extends traditional Least Recently Used caching mechanisms by incorporating semantic awareness of embeddings. Unlike conventional caches that rely on exact key matches, our cache can retrieve items based on embedding similarity, dramatically improving hit rates for semantically similar queries.

The cache employs a novel two-tier structure:

- A fast, exact-match first tier for frequently accessed identical requests
- A similarity-based second tier that uses approximate nearest neighbor techniques

Figure 2 depicts the cache architecture and operation flow.

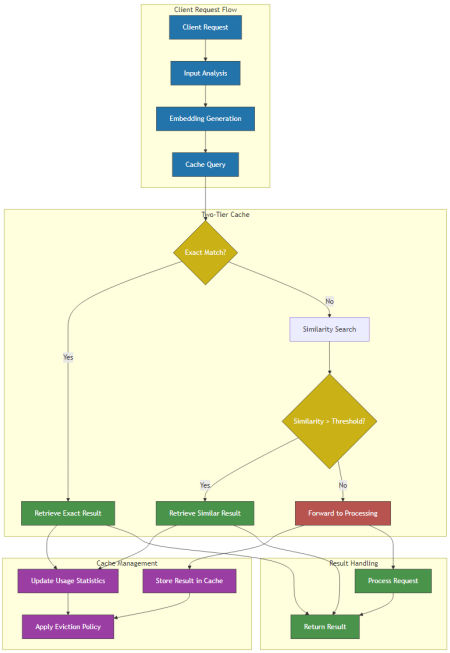

Figure 2: NeuroBridge LRU-Embedding Cache Architecture

3.1.1 Vector Database Alternatives

While our current implementation uses an in-memory HNSW index for similarity search, production deployments can benefit from specialized vector database technologies:

- **Redis with VSS/RediSearch**: Provides distributed in-memory vector search with persistence options, suitable for high-throughput, low-latency scenarios and horizontal scaling
- **Postgres with pgvector**: Offers ACID-compliant vector operations with SQL integration, ideal when embeddings need to coexist with structured data
- **Purpose-built vector databases**: Solutions like Qdrant, Weaviate, or Milvus provide advanced filtering, sharding, and multitenancy

For enterprise deployments, we offer integration adapters that maintain the same API while delegating vector operations to these specialized systems. This hybrid approach preserves the exact-match LRU cache tier locally while offloading similarity search to optimized vector databases:

Listing 1: Vector Database Integration Example

```
pub struct HybridEmbeddingCache {
```

```
    // Local exact-match first tier
    exact_cache: LruCache<CacheKey, CacheValue>,
    // Remote vector database for similarity search
    vector_db: Box<dyn VectorDbAdapter>,
    // Configuration parameters
    config: CacheConfig,
}

impl HybridEmbeddingCache {
    pub async fn get(&mut self, key: &CacheKey) -> Option<&
        CacheValue> {
        // Try exact match first (local, fast)
        if let Some(value) = self.exact_cache.get(key) {
            return Some(value);
        }

        // If embedding available, try similarity search via
            vector DB
        if let Some(embedding) = &key.embedding {
            if let Some(nearest_key) = self.vector_db.
                find_nearest(embedding).await {
                return self.exact_cache.get(&nearest_key);
            }
        }

        None
    }
}
```

This architecture allows for flexible deployment scenarios, from edge devices with limited resources to enterprise-scale installations, while maintaining consistent semantics and API interfaces.

3.2 WASM Implementation

NeuroBridge leverages WebAssembly (WASM) to ensure portability and security while maintaining near-native performance. The WASM implementation provides several advantages:

- Cross-platform compatibility without recompilation
- Sandboxed execution environment enhancing security
- Consistent performance across diverse environments
- Reduced payload size through efficient binary representation

Our implementation extends standard WASM capabilities through custom extensions specifically designed for tensor operations and neural network inference.

3.3 Compression Techniques

To optimize network transmission and storage requirements, NeuroBridge implements a multi-layered compression approach:

- **Structural Compression**: Eliminates redundancies in neural network structures
- **Weight Quantization**: Reduces precision of network weights while preserving accuracy
- **Embedding Compression**: Uses principal component analysis and vector quantization
- **Adaptive Compression**: Dynamically adjusts compression levels based on network conditions

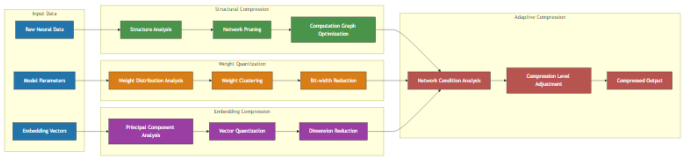

Figure 3: Multi-stage Compression Pipeline

3.4 Semantic Search

The semantic search component of NeuroBridge enables intelligent retrieval of information based on meaning rather than exact keyword matching. Our approach combines:

- Dense vector embeddings for capturing semantic relationships
- Hybrid retrieval combining sparse and dense representations
- Re-ranking mechanisms to improve precision
- Query expansion through neuro-symbolic reasoning

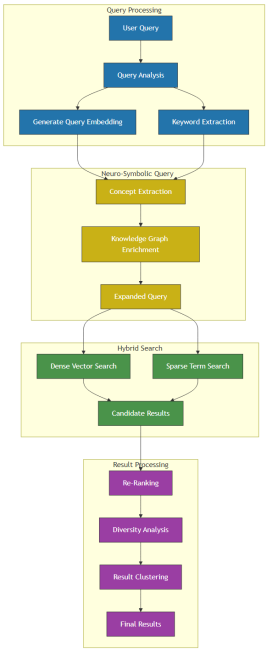

Figure 4: Semantic Search Architecture and Process Flow

3.5 Fallback Mechanism

NeuroBridge implements a sophisticated fallback system that ensures continuous operation even when primary systems fail. The fallback mechanism operates through:

- Hierarchical service degradation rather than binary failure
- Local computation capabilities when remote services are unavailable
- Cached response utilization with clear staleness indicators
- Prediction of potential failures through system monitoring

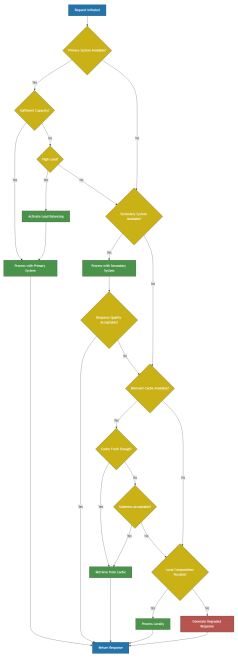

Figure 5: Fallback Mechanism Decision Tree

4 Implementation Details

4.1 Client-Side Integration

Integrating NeuroBridge into client applications is straightforward through our API:

Listing 2: Client Integration Example

```
from neurobridge import Client

# Initialize the client with configuration
client = Client(
    embedding_model="semantic-v3",
    symbolic_reasoner="bridgelogic-2.1",
    cache_size_mb=512
)

# Perform a neuro-symbolic query
result = client.query(
    input_text="What␣are␣the␣economic␣implications␣of␣
        renewable␣energy␣adoption␣in␣developing␣markets?",
```

```
    reasoning_depth=3,
    use_cache=True
)

# Access different components of the result
neural_response = result.neural_output
symbolic_reasoning = result.reasoning_steps
confidence = result.confidence_score
```

4.2 LRU-Embedding Cache Implementation

The LRU-Embedding cache implementation uses a combination of fast hash tables and approximate nearest neighbor (ANN) indexes:

Listing 3: Cache Implementation Excerpt

```
pub struct EmbeddingCache {
    // Exact-match first tier
    exact_cache: LruCache<CacheKey, CacheValue>,
    // Similarity-based second tier
    embedding_index: HnswIndex<f32>,
    // Mapping from ANN IDs to cache entries
    id_mapping: HashMap<u64, CacheKey>,
    // Configuration parameters
    config: CacheConfig,
}

impl EmbeddingCache {
    pub fn get(&mut self, key: &CacheKey) -> Option<&
        CacheValue> {
        // Try exact match first
        if let Some(value) = self.exact_cache.get(key) {
            return Some(value);
        }

        // If embedding available, try similarity search
        if let Some(embedding) = &key.embedding {
            let neighbors = self.embedding_index
                .search(embedding, 1, self.config.
                    similarity_threshold);

            if !neighbors.is_empty() {
                let nearest_id = neighbors[0].id;
                if let Some(nearest_key) = self.id_mapping.
                    get(&nearest_id) {
                    return self.exact_cache.get(nearest_key)
                        ;
                }
            }
        }
```

```
            None
        }

        // Additional methods for insert, update, etc.
}
```

4.3 WASM Module Structure

The WASM implementation is organized around a modular architecture that separates core functionality:

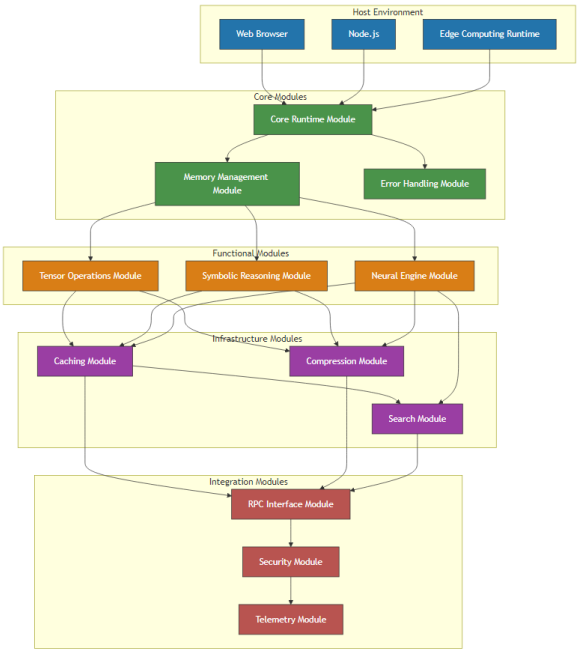

Figure 6: WASM Module Organization

Each module exposes a well-defined API that can be invoked from the host environment:

Listing 4: WASM Module Initialization

```
// Initialize NeuroBridge WASM modules
async function initNeuroBridge() {
  const mainModule = await WebAssembly.instantiateStreaming(
    fetch('neurobridge_core.wasm'),
    importObject
  );
```

```
  const tensorOps = await WebAssembly.instantiateStreaming(
    fetch('neurobridge_tensor.wasm'),
    {
      env: {
        memory: mainModule.instance.exports.memory,
        // Additional environment configurations
      }
    }
  );

  return {
    core: mainModule.instance.exports,
    tensor: tensorOps.instance.exports,
    // Additional module exports
  };
}
```

5 Comparison with LiteLLM

While LiteLLM provides a valuable service by offering a unified interface to multiple LLM providers, NeuroBridge differs in several significant ways:

Table 1: Comparison of NeuroBridge vs LiteLLM

Feature	**NeuroBridge**	**LiteLLM**
Primary Focus	Neuro-symbolic integration	LLM interface unification
Caching	Semantic embedding-aware LRU cache	Basic response caching
Implementation	WASM-based cross-platform	Python SDK and proxy server
Compression	Multi-level adaptive compression	Not a primary feature
Fallback	Sophisticated degradation hierarchy	Basic model fallbacks
Search Capabilities	Built-in semantic search	Not included

5.1 Architectural Differences

Unlike LiteLLM, which acts primarily as a gateway to external LLM providers, NeuroBridge provides a complete processing framework that incorporates both neural and symbolic operations natively. Figure 7 illustrates these architectural differences.

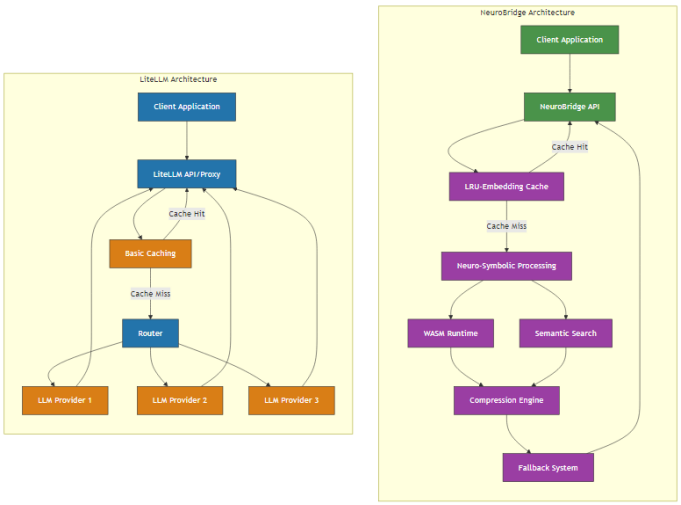

Figure 7: Architectural Comparison: NeuroBridge vs LiteLLM

5.2 Performance Advantages

NeuroBridge demonstrates several performance advantages over LiteLLM:

- **Reduced Latency**: The embedding-aware cache reduces response times for semantically similar queries
- **Lower Bandwidth Requirements**: Multi-level compression reduces network traffic
- **Enhanced Reliability**: The sophisticated fallback system prevents complete service outages

Figure 8 shows benchmark results comparing response times across different scenarios.

Figure 8: Performance Comparison under Different Load Conditions

6 Use Cases and Applications

NeuroBridge is particularly well-suited for applications that require:

6.1 Edge AI Deployment

The WASM implementation makes NeuroBridge ideal for edge computing environments where:

- Network connectivity may be intermittent
- Local processing power is limited
- Privacy concerns require local inference

6.2 Hybrid Reasoning Systems

Applications requiring both pattern recognition and logical reasoning benefit from NeuroBridge's integrated approach:

- Medical diagnosis systems combining imaging and rule-based evaluation
- Financial risk assessment combining market pattern analysis with regulatory compliance
- Industrial automation systems that combine sensor pattern recognition with safety rules

6.3 Intelligent Search Applications

NeuroBridge's semantic search capabilities make it ideal for:

- Enterprise knowledge management systems
- Legal document analysis and retrieval
- Research and academic literature exploration

7 Future Directions

The NeuroBridge roadmap includes several planned enhancements:

- **Federated Learning Integration**: Enabling distributed model training across NeuroBridge nodes
- **Explainable AI Components**: Adding capabilities to explain reasoning processes
- **Enhanced Symbolic Reasoning**: Incorporating more powerful theorem proving capabilities
- **Domain-Specific Optimizations**: Pre-configured modules for common industry applications

8 Conclusion

NeuroBridge represents a significant advancement in neuro-symbolic computing by providing an efficient, robust RPC framework specifically designed for AI applications. By integrating neural network capabilities with symbolic reasoning in a portable WASM environment, and enhancing system efficiency through embedding-aware caching and adaptive compression, NeuroBridge offers a compelling solution for distributed AI applications.

Unlike frameworks such as LiteLLM that focus primarily on API unification, NeuroBridge addresses the fundamental challenges of integrating different AI paradigms while optimizing for performance and reliability. As AI systems continue to evolve toward hybrid approaches that combine multiple reasoning methods, frameworks like NeuroBridge will play an increasingly important role in the AI ecosystem.

References

[1] Haas, A., et al. "Bringing the web up to speed with WebAssembly." Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2017.

[2] Yi, K., et al. "CLEVRER: Collision Events for Video Representation and Reasoning." International Conference on Learning Representations. 2020.

[3] Ramani, S., et al. "CACHEUS: Cache-Aware Scheduling for Deep Learning Training." Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. 2021.

[4] Karpukhin, V., et al. "Dense Passage Retrieval for Open-Domain Question Answering." Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing. 2020.

[5] Chen, J., et al. "Resilient Distributed Services: Recovery and Fault Tolerance Strategies for Deep Learning Systems." International Conference on Distributed Computing Systems. 2022.