Model Context Protocol (MCP): A Technical Literature Review

Abstract

This literature review examines the Model Context Protocol (MCP) and related context structuring approaches for Large Language Models (LLMs). As LLMs continue to evolve in capability and adoption, their inherent limitations in context management present significant challenges for AI system development. This review provides a comprehensive analysis of MCP's technical foundations, Anthropic's implementation, alternative context structuring methodologies, and comparative performance metrics. Through systematic examination of academic literature, technical documentation, and industry reports, we evaluate MCP's position within the broader landscape of context management solutions, its current adoption, and future research directions. The findings indicate that MCP represents a promising standardized approach to connecting AI models with external systems, offering significant advantages in terms of interoperability, security, and scalability compared to proprietary alternatives.

1. Introduction

1.1 Background and Significance

Large Language Models (LLMs) have revolutionized natural language processing, enabling sophisticated tasks such as long-form text generation, complex reasoning, and multi-turn dialogues. Despite their impressive capabilities, LLMs face inherent limitations in how they process and maintain context over extensive inputs. These models typically operate within fixed-size context windows, ranging from a few thousand to a million tokens, constraining their ability to access and process information beyond these boundaries.

The Model Context Protocol (MCP), developed and open-sourced by Anthropic in November 2024, addresses this fundamental challenge by providing a standardized protocol for connecting AI assistants to external systems where data lives (Anthropic, 2024). As described by its creators, MCP functions as a "USB-C port for AI," enabling standardized communication between LLMs and external data sources, tools, and systems.

The significance of MCP lies in its potential to transform how AI systems interact with the world. By breaking down information silos and enabling secure, bidirectional communication with external resources, MCP addresses a critical bottleneck in AI system development: the isolation of even sophisticated models from real-time data and tools.

1.2 Scope and Objectives

This literature review focuses on MCP and related context structuring approaches, providing a comprehensive analysis of their technical foundations, implementations, and performance characteristics. Our objectives include:

1. Examining the technical architecture and specifications of MCP, with particular emphasis on Anthropic's implementation
2. Analyzing alternative context structuring approaches, including architectural innovations and workflow techniques
3. Providing a comparative evaluation of different methodologies based on technical capabilities, performance metrics, and use case suitability
4. Assessing current industry adoption and applications of MCP
5. Identifying future research directions and challenges in context management for LLMs

1.3 Methodology

This review employs a systematic approach to literature selection and analysis. We examined three primary categories of sources:

1. **Academic literature**: Peer-reviewed papers, preprints, and conference proceedings related to context management in LLMs
2. **Technical documentation**: Official specifications, developer guides, and SDK documentation from AI labs and technology companies
3. **Industry reports**: Analyses, case studies, and adoption reports from industry practitioners and observers

Our analytical framework evaluates context structuring approaches based on several key dimensions: technical architecture, standardization, context length capabilities, computational efficiency, implementation complexity, security features, and use case suitability. This multifaceted approach enables a comprehensive assessment of MCP's position within the broader landscape of context management solutions.

2. The Challenge of Context Management in LLMs

2.1 Inherent Limitations of LLMs

LLMs process text through fixed-size context windows, creating several fundamental challenges:

Fixed Context Windows and Their Constraints: Traditional transformer-based LLMs have fixed context windows that limit the amount of information they can process simultaneously. While recent models have expanded these windows significantly—from a few thousand tokens in early models to hundreds of thousands or even millions in the latest iterations—they still impose hard limits on context length (KhueApps, 2025).

Computational Complexity of Self-Attention Mechanisms: The self-attention mechanism in transformer architectures has quadratic computational complexity with respect to sequence length. As Liu et al. (2025) note, "this quadratic complexity becomes prohibitively expensive for very long sequences, limiting practical context length even with substantial computational resources."

Information Retention and Relevance Filtering: Models struggle to maintain coherence and recall information from earlier parts of long contexts. They must also distinguish between relevant and irrelevant information within extensive contexts, a challenge that becomes increasingly difficult as context length grows.

2.2 Impact on AI System Development

These limitations significantly impact AI system development in several ways:

Isolation from Real-Time Data: Traditional LLMs are trained on static datasets and lack access to real-time information. This isolation limits their utility in dynamic environments where current data is essential for accurate responses.

Integration Challenges with External Systems: Without standardized protocols, integrating LLMs with external systems requires custom solutions for each combination of model and tool. This fragmentation increases development complexity and limits interoperability.

The N×M Problem in Tool Integration: As described in Anthropic's MCP documentation, the integration challenge represents an N×M problem, where N is the number of models and M is the number of tools or data sources. Without standardization, each combination requires a custom integration, leading to exponential growth in development effort as the ecosystem expands (Model Context Protocol, 2025).

2.3 Taxonomy of Context Structuring Approaches

Based on our research, context structuring approaches can be categorized into three main categories:

Architectural Innovations: These approaches modify the model architecture to handle longer contexts efficiently. Examples include sparse attention mechanisms, hierarchical processing, and memory compression techniques.

Protocol-Based Approaches: These standardized interfaces extend model capabilities through external tools and data sources. MCP is the primary example, with function calling representing a more limited alternative.

Workflow Techniques: These methodologies manage context during model operation without architectural changes. Examples include chunking and segmentation, retrieval-augmented generation, and context filtering.

Each category addresses different aspects of the context management challenge, with varying implications for implementation complexity, performance, and use case suitability.

3. Model Context Protocol: Technical Foundations

3.1 Definition and Core Architecture

The Model Context Protocol (MCP) is an open standard developed by Anthropic that serves as a standardized protocol for connecting AI assistants to external systems. At its core, MCP functions as a universal connector that enables standardized communication between LLMs and external data sources, tools, and systems (Anthropic, 2024).

The fundamental architecture of MCP follows a client-server model with three main components (Model Context Protocol, 2025):

1. **Host Applications (Host Process)**: The primary AI environment (e.g., Claude Desktop, IDEs, or other AI tools) that initiates and manages connections to external resources.
2. **MCP Clients**: Intermediary components within the host that maintain 1:1 connections with MCP servers, handling communication, security, and protocol management.
3. **MCP Servers**: Lightweight, external programs that expose specific capabilities, such as data access, tools, or prompts. These servers implement the MCP protocol primitives and can interface with databases, APIs, files, or other systems.

Figure 1: Model Context Protocol Architecture consists of three main components: Host Applications, MCP Clients, and MCP Servers in a client-server model. (Source: Model Context Protocol Documentation)

This architecture transforms the integration challenge from an N×M problem (number of models × number of tools) to a more manageable N+M setup, promoting interoperability, security, and scalability across the AI ecosystem.

3.2 Protocol Specification

MCP's protocol specification is defined in TypeScript (`schema.ts`) and exported as JSON Schema for compatibility across different programming languages (Model Context Protocol, 2025). The protocol uses JSON-RPC 2.0 as its messaging backbone, supporting three primary message types:

Requests: Initiate actions or queries from the client to the server. Each request includes a unique identifier, method name, and parameters.

Responses: Reply to requests, containing results or errors. Responses are linked to their corresponding requests through the unique identifier.

Notifications: One-way messages for status updates or events that do not require a response.

The communication lifecycle in MCP consists of three phases:

1. **Initialization**:
   - The client sends an `initialize` request with protocol version and capabilities
   - The server responds with its protocol version and capabilities
   - The client sends an `initialized` notification as acknowledgment
2. **Operation**:
   - Messages are exchanged according to negotiated capabilities
   - Context, resources, prompts, and tools are managed
3. **Shutdown**:
   - The client sends a `shutdown` request
   - The server acknowledges and releases resources
   - The client sends an `exit` notification

This structured lifecycle ensures reliable connection establishment, capability negotiation, and graceful termination.

3.3 Technical Capabilities

MCP provides three primary capabilities (Model Context Protocol, 2025):

Resources: Context and data for the user or AI model to use. Resources can include files, database records, API responses, or other structured data. The protocol standardizes how these resources are discovered, accessed, and manipulated.

Prompts: Templated messages and workflows for users. Prompts can include predefined templates, conversation starters, or guided workflows that help users interact with the AI system effectively.

Tools: Functions for the AI model to execute. Tools represent external capabilities that the model can invoke, such as calculations, API calls, or data transformations.

Additionally, clients may offer sampling capabilities to servers, enabling server-initiated agentic behaviors and recursive LLM interactions. This capability allows servers to request the model to perform specific reasoning or generation tasks as part of a larger workflow.

3.4 Security Architecture

Security is a fundamental aspect of MCP's design, with several key principles (Model Context Protocol, 2025):

User Consent and Control: Users must explicitly consent to and understand all data access and operations. The protocol includes mechanisms for requesting

and tracking user consent.

Data Privacy: Hosts must obtain explicit user consent before exposing user data to servers. The protocol includes provisions for data minimization and purpose limitation.

Tool Safety: Tools represent arbitrary code execution and must be treated with appropriate caution. The protocol includes mechanisms for tool validation and sandboxing.

LLM Sampling Controls: Users must explicitly approve any LLM sampling requests. The protocol includes controls to prevent unauthorized model invocation.

The implementation includes security features such as TLS/SSL for remote communication, authentication and authorization mechanisms, message validation, and access controls. These features ensure that MCP connections maintain confidentiality, integrity, and availability while respecting user privacy and security.

4. Anthropic's Implementation of MCP

4.1 Technical Implementation Details

Anthropic's implementation of MCP is built on a robust technical foundation that emphasizes security, flexibility, and standardization. The implementation includes several key technical components:

TypeScript Schema and JSON Schema Exports: The MCP specification is defined in TypeScript (`schema.ts`) and exported as JSON Schema for compatibility across different programming languages (Model Context Protocol, 2025). This approach ensures consistent implementation across diverse environments.

Transport Mechanisms: MCP supports multiple transport mechanisms to facilitate communication between clients and servers:

- **Stdio (Standard Input/Output)**: For local, intra-process communication
- **HTTP with Server-Sent Events (SSE)**: For remote, internet-based communication, enabling secure, scalable, and flexible deployment

SDKs and Implementation Support: Anthropic provides SDKs in multiple languages to facilitate MCP integration (Model Context Protocol, 2025): - TypeScript SDK - Python SDK - Java SDK - Kotlin SDK - C# SDK

These SDKs offer comprehensive frameworks for building MCP servers and clients, with reference implementations demonstrating core features.

4.2 Core Components and Interactions

The core components of Anthropic's MCP implementation interact through well-defined interfaces and protocols:

Protocol Layer Functionality: The protocol layer handles message framing, request/response linking, and high-level communication patterns. Key classes include `Protocol`, `Client`, and `Server`, which implement capability negotiation and session management.

Transport Layer Mechanisms: The transport layer manages actual communication between clients and servers, supporting multiple transport mechanisms and using JSON-RPC 2.0 for message exchange.

Message Format and Structure: MCP defines a structured message schema, typically JSON-based, with fields for version information, metadata, message content, and settings. This structured format ensures consistent parsing and processing across implementations.

4.3 Implementation Example

A basic example of implementing an MCP server in TypeScript demonstrates the protocol's simplicity and flexibility (Model Context Protocol, 2025):

```
import { Server } from "@modelcontextprotocol/sdk/server/index.js";
import { StdioServerTransport } from "@modelcontextprotocol/sdk/server/stdio.js";

const server = new Server({
  name: "example-server",
  version: "1.0.0"
}, {
  capabilities: {
    resources: {}
  }
});

// Handle requests
server.setRequestHandler(ListResourcesRequestSchema, async () => {
  return {
    resources: [
      {
        uri: "example://resource",
        name: "Example Resource"
      }
    ]
  };
});
```

```
// Connect transport
const transport = new StdioServerTransport();
await server.connect(transport);
```

This example illustrates the core components of an MCP server: server initialization with capabilities, request handler registration, and transport connection. The simplicity of this implementation highlights MCP's design goal of providing a standardized yet flexible protocol for AI system integration.

4.4 Security Features

Anthropic's MCP implementation includes robust security features to protect user data and system integrity:

TLS/SSL for Remote Communication: When using HTTP with SSE for remote communication, MCP employs TLS/SSL encryption to ensure confidentiality and integrity of data in transit.

Message Validation: The protocol includes comprehensive message validation to prevent malformed or malicious messages from compromising system security.

Access Controls and Permissions: MCP implements fine-grained access controls and permissions to ensure that servers can only access resources and capabilities explicitly authorized by the user.

These security features reflect Anthropic's commitment to responsible AI development and deployment, ensuring that MCP connections maintain high standards of security and privacy protection.

5. Alternative Context Structuring Approaches

5.1 Architectural Innovations

Several architectural innovations address the challenge of context management in LLMs:

Sparse Attention Mechanisms: Models like Longformer and BigBird implement sparse attention patterns that reduce the quadratic complexity of self-attention to linear or log-linear complexity (Beltagy et al., 2020). These approaches selectively attend to specific tokens rather than all tokens in the context, enabling processing of much longer sequences.

Hierarchical and Multi-scale Attention: Hierarchical models process text at multiple levels—tokens, sentences, paragraphs—allowing the model to capture both local and global dependencies. This structuring reduces the burden on the attention mechanism and preserves long-range coherence (Towards Dev, 2025).

Memory Compression Techniques: Memory compression techniques reduce the effective size of the context by compressing less relevant information. This

approach maintains important information while reducing computational requirements.

Retrieval-Augmented Models: Retrieval-Augmented Generation (RAG) extends context length by storing and retrieving relevant information from external memory or knowledge bases (Lewis et al., 2020). These techniques structure the context by dynamically incorporating external data, enabling models to maintain coherence over long documents or dialogues.

5.2 Vendor-Specific Approaches

Major AI labs have developed proprietary approaches to context structuring:

OpenAI's Context Window Extensions and Function Calling: OpenAI has focused on extending context windows through architectural optimizations and training techniques. GPT-4 models support context windows up to 32,000 tokens, with specialized versions supporting up to 1,000,000 tokens (KhueApps, 2025). Additionally, OpenAI's function calling enables models to generate structured JSON outputs that can be used to call external functions.

Meta's LLaMA Approach: Meta's LLaMA models employ several architectural innovations for context structuring:

- **Grouped Query Attention (GQA)**: Reduces computational overhead and improves inference efficiency (Meta, 2025).
- **Segmented and Hierarchical Processing**: LLaMA 4's architecture employs a mixture of experts (MoE) architecture for processing large datasets and long sequences (Geeky Gadgets, 2025).
- **Multilevel Visual and Textual Feature Preservation**: In multimodal models like LLaMA 3.2, a two-stage vision encoder preserves multiscale visual features through intermediate layer outputs (Qi, 2025).

Google's PaLM Approach: Google's Pathways Language Model (PaLM) family handles context through context setting via prompt engineering, conversation history management, and parameter optimization (Google AI, 2025).

5.3 Workflow Techniques

Several workflow techniques address context management without architectural changes:

Chunking and Segmentation Methods: Chunking involves breaking long texts into manageable segments that fit within the model's context window. Sliding window techniques process text in overlapping chunks, maintaining continuity between segments.

Retrieval-Augmented Generation (RAG): RAG combines retrieval mechanisms with generative models to enhance context management. Vector databases store embeddings of documents or chunks, enabling semantic retrieval based on query relevance.

Context Filtering and Recomposition: Context filtering techniques identify and prioritize the most relevant information within a large context. Entropy-based heuristics identify the most informative parts of the context, effectively structuring the input by removing noise and redundancy (arXiv, 2025).

Structured Context Recomposition (SCR): SCR introduces probabilistic realignment of internal representations within transformer layers. Instead of relying solely on attention, SCR dynamically adjusts hierarchical embeddings to reinforce semantically relevant information across extended sequences (Teel et al., 2024).

6. Comparative Analysis

6.1 Technical Comparison

Different context structuring approaches exhibit varying characteristics across several technical dimensions:

Approach	Protocol Architecture	Standardization	Context Length	Computational Efficiency	Implementation Complexity
MCP	Client-server	High (Open standard)	Unlimited*	High	Medium
Function Calling	Request-response	Low (Vendor-specific)	Limited	High	Low
Sparse Attention	N/A (Architectural)	Low	High	Medium	High
RAG	N/A (Workflow)	Medium	Very High	Medium	Medium
Chunking	N/A (Workflow)	Low	High	High	Low

*MCP theoretically allows unlimited context through external data access, though practical limitations exist.

Protocol Architecture and Standardization: MCP stands out for its client-server architecture and high level of standardization as an open protocol. In contrast, function calling follows a simpler request-response pattern but lacks standardization across vendors.

Context Length Capabilities: Architectural innovations like sparse attention and workflow techniques like RAG enable processing of long contexts, but MCP theoretically allows unlimited context through external data access.

Computational Efficiency: Workflow techniques like chunking offer high computational efficiency, while architectural innovations like sparse attention have medium efficiency due to their complexity.

Implementation Complexity: Function calling and chunking have low implementation complexity, making them accessible to developers with limited resources. MCP has medium complexity, balancing flexibility with ease of implementation.

6.2 Performance Metrics

Evaluating the performance of context structuring approaches presents several challenges:

Current State of MCP Performance: Early adopters such as Block and Apollo have integrated MCP into their systems, demonstrating initial practical applications. Development tool providers like Zed, Replit, Codeium, and Sourcegraph are actively collaborating with MCP to improve their platforms' data retrieval and contextual understanding capabilities (Anthropic, 2024).

Benchmarks and Evaluation Criteria: Industry experts note the absence of standardized performance metrics, which hampers objective evaluation and comparison with other data integration approaches (Renewable AI, 2025). Key metrics that would be valuable include:

- Latency: Time required to retrieve and process external data
- Throughput: Number of requests that can be handled simultaneously
- Reliability: Consistency of performance under various conditions
- Security: Resistance to unauthorized access or data leakage

Challenges in Performance Measurement: Several factors complicate performance measurement for context structuring approaches:

- Lack of standardized benchmarks
- Variability in implementation quality
- Differences in underlying hardware and infrastructure
- Diversity of use cases and requirements

6.3 Use Case Suitability

Different approaches are better suited for specific use cases:

Enterprise Data Access: MCP excels at enterprise data access scenarios, providing secure, standardized access to internal databases, document repositories, and enterprise systems. This capability allows organizations to build AI assistants that can answer questions based on internal data while maintaining security and access controls.

Development Tools and Environments: MCP enhances development environments by connecting AI models with coding tools and repositories. Companies like Zed, Replit, Codeium, and Sourcegraph are leveraging MCP to improve their AI-driven coding workflows (Anthropic, 2024).

Knowledge Management and Retrieval: RAG approaches are particularly effective for knowledge management and retrieval, enabling semantic search and retrieval-augmented generation across diverse sources.

Multi-Tool Orchestration: MCP enables complex workflows involving multiple tools and data sources, supporting sophisticated AI agents that can decompose complex tasks, select appropriate tools, and orchestrate their execution.

Desktop and Local Applications: MCP supports local AI applications that can interact with the user's environment, as demonstrated by Anthropic's Claude Desktop app (Anthropic, 2024).

6.4 Integration Challenges

Each approach presents unique integration challenges:

Technical Implementation Barriers: Architectural innovations require specialized model training and may not be compatible with all deployment environments. Protocol-based approaches like MCP depend on external infrastructure, while workflow techniques may require significant engineering effort.

Security Considerations: MCP addresses security through a comprehensive framework including user consent, data privacy, tool safety, and LLM sampling controls. However, implementing these security features correctly requires careful attention to detail.

Standardization and Compatibility Issues: While MCP aims to provide a standardized interface, compatibility across different implementations and versions remains a challenge. The evolving nature of the protocol may require ongoing adaptation as new capabilities are added.

7. Industry Adoption and Applications

7.1 Current Industry Adoption

MCP has gained traction across various industry segments:

Early Adopters: Companies like Block and Apollo have integrated MCP into their systems, demonstrating its practical utility in production environments (Anthropic, 2024).

Development Tool Providers: Zed, Replit, Codeium, and Sourcegraph are actively collaborating with MCP to improve their platforms' data retrieval and contextual understanding capabilities (Anthropic, 2024).

Enterprise System Integrations: Major companies including Microsoft, OpenAI, and Cloudflare have adopted or expressed support for MCP, indicating its potential as an industry standard (Hou et al., 2025).

7.2 Real-World Applications

MCP enables a wide range of applications across various domains:

Enterprise Data Access Implementations: Organizations are using MCP to build AI assistants that can access internal databases, document repositories, and enterprise systems securely. These applications enable employees to query company data, generate reports, and gain insights through natural language interactions.

Development Environment Integrations: IDE providers are integrating MCP to enhance their AI-driven coding workflows. These integrations enable more contextual code suggestions, bug detection, and documentation generation based on the developer's codebase and environment.

Knowledge Management Systems: MCP facilitates knowledge management applications that can query and update semantic knowledge bases, perform retrieval-augmented generation, and find relevant information across diverse sources.

Multi-Tool AI Agents: Organizations are building sophisticated AI agents that can decompose complex tasks, select appropriate tools, and orchestrate their execution to achieve user goals. These agents leverage MCP's standardized interface to interact with multiple tools and data sources seamlessly.

7.3 Emerging Standards and Best Practices

Several standards and best practices are emerging in the field of context structuring:

MCP as an Open Standard: MCP is gaining traction as a standardized interface for connecting AI models with external tools and data sources. Its open nature encourages adoption and contribution from the broader AI community.

MLCommons Taxonomy: The MLCommons taxonomy provides a standardized framework for categorizing and evaluating AI capabilities, including context handling. This taxonomy is being adopted by tools like Llama Guard 2 to support the emergence of industry standards (Meta, 2024).

Best Practices for Context Management: Industry best practices for context management include token efficiency, strategic placement of essential instructions, regular summarization and compression of lengthy histories, and the development of reusable prompt templates.

8. Future Directions and Research Opportunities

8.1 Technical Advancements

Recent and anticipated technical advancements in context structuring include:

Extended Context Windows: Models with context windows of millions of tokens, such as Claude 3.7 Sonnet (200,000 tokens) and GPT-4.1 (1,000,000 tokens), represent significant advances in context length capabilities (KhueApps, 2025).

Multimodal Context Integration: Emerging techniques focus on integrating text, images, and other modalities within a unified context representation. These approaches enable more comprehensive understanding and generation across multiple data types.

Dynamic Memory Management: Advanced approaches adaptively allocate context capacity based on information importance, optimizing the use of limited context windows for maximum effectiveness.

8.2 Research Challenges

Several research challenges present opportunities for future work:

Standardized Benchmarks Development: The development of standardized benchmarks for evaluating context structuring approaches would enable objective comparison and drive improvements across the field.

Security and Privacy Enhancements: As context structuring approaches access increasingly sensitive data, enhanced security and privacy mechanisms are needed to protect user information and prevent unauthorized access.

Scalability and Performance Optimization: Optimizing the performance and scalability of context structuring approaches, particularly for high-demand environments, represents an important area for future research.

8.3 Potential Impact on AI Development

Advances in context structuring have the potential to significantly impact AI development:

Democratization of AI Capabilities: Standardized protocols like MCP can democratize access to advanced AI capabilities, enabling smaller organizations and individual developers to build sophisticated AI applications.

Enhanced Human-AI Collaboration: Improved context management enables more natural and effective collaboration between humans and AI systems, with the AI maintaining a more comprehensive understanding of the shared context.

Integration with Emerging AI Architectures: Context structuring approaches will need to evolve to integrate with emerging AI architectures, such as multimodal models, agent-based systems, and specialized domain models.

9. Conclusion

9.1 Summary of Key Findings

The Model Context Protocol (MCP) represents a significant advancement in the integration of large language models with external data sources, tools, and systems. Developed and open-sourced by Anthropic, MCP addresses the critical challenge of connecting AI models with real-time data and tools through a standardized, secure, and flexible protocol.

Our comparative analysis reveals that MCP offers several advantages over alternative approaches:

- As an open standard, MCP promotes interoperability and ecosystem development
- Its client-server architecture provides a flexible framework for diverse applications
- Its comprehensive security features address critical concerns in AI system deployment

While architectural innovations and workflow techniques offer valuable capabilities for specific use cases, MCP's standardized approach positions it as a foundational protocol for the future of AI system integration.

9.2 Implications for AI Research and Development

MCP and related context structuring approaches have significant implications for AI research and development:

- They address a fundamental limitation of current LLMs, enabling more contextual, accurate, and capable AI applications
- They promote standardization and interoperability, reducing fragmentation in the AI ecosystem
- They enable new categories of AI applications that can interact with the world in more meaningful ways

As the field continues to evolve, we anticipate further advancements in context structuring techniques, with a focus on increased efficiency, improved coherence, and enhanced integration capabilities. These developments will enable more sophisticated AI applications across domains, from document analysis and conversational agents to enterprise systems and creative tools.

References

Anthropic. (2024, November 25). Introducing the Model Context Protocol. https://www.anthropic.com/news/model-context-protocol

arXiv. (2025). Entropy-based Context Filtering for Large Language Models. arXiv:2501.17617. https://arxiv.org/html/2501.17617v1

Beltagy, I., Peters, M. E., & Cohan, A. (2020). Longformer: The Long-Document Transformer. arXiv:2004.05150. https://arxiv.org/abs/2004.05150

Geeky Gadgets. (2025). Llama 4 AI Model Long Context Window. https://www.geeky-gadgets.com/llama-4-ai-model-long-context-window/

Google AI. (2025). PaLM API: Chat quickstart with Python. https://ai.google.dev/palm_docs/chat_quickstar

Hou, X., Zhao, Y., Wang, S., & Wang, H. (2025). Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions. arXiv:2503.23278. https://arxiv.org/abs/2503.23278

KhueApps. (2025). Explaining Context Window When Using OpenAI API. https://www.khueapps.com/blog/article/explaining-context-window-when-using-openai-api

Lewis, P., Perez, E., Piktus, A., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv:2005.11401. https://arxiv.org/abs/2005.11401

Liu, J., Zhu, D., Bai, Z., et al. (2025). A Comprehensive Survey on Long Context Language Modeling. arXiv:2503.17407. https://arxiv.org/abs/2503.17407

Medium. (2025). Model Context Protocol (MCP): Real-world Use Cases, Adoptions, and Comparison to Functional Calling. https://medium.com/@laowang_journey/model-context-protocol-mcp-real-world-use-cases-adoptions-and-comparison-to-functional-calling-9320b775845c

Meta. (2024). Introducing Meta Llama 3: The most capable openly available LLM to date. https://ai.meta.com/blog/meta-llama-3/

Meta. (2025). The Llama Family of Models: Model Architecture and Capabilities. https://www.icodeformybhasa.com/p/the-llama-family-of-models-model

Model Context Protocol. (2025). Core architecture. https://modelcontextprotocol.io/docs/concepts/architectu

Model Context Protocol. (2025). Introduction. https://modelcontextprotocol.io/introduction

Model Context Protocol. (2025, March 26). Specification. https://modelcontextprotocol.io/specification/2025-03-26/index

OpenAI. (2025). Model context protocol (MCP). https://openai.github.io/openai-agents-python/mcp/

Qi, J. (2025). Inside MLLaMA 3.2: Understanding Meta's Vision-Language Model Architecture. https://j-qi.medium.com/inside-mllama-3-2-understanding-metas-vision-language-model-architecture-ae12ad24dcbf

Renewable AI. (2025). Breaking Data Barriers: Can Anthropic's Model Context Protocol Enhance AI Performance? https://renewableai.org/news/breaking-data-barriers-can-anthropics-model-context-protocol-enhance-ai-performance/

Teel, J., Cumberbatch, J., Benington, R., & Baskerville, Q. (2024). Structured Context Recomposition for Large Language Models Using Probabilistic Layer Realignment. [Preprint]

Towards Dev. (2025). Beyond the Window: The Complete Guide to Long Context Language Models and Why They're Changing AI. https://towardsdev.com/beyond-the-window-the-complete-guide-to-long-context-language-models-and-why-theyre-changing-ai-ee28f6c2b428