

Literature Review and Research Proposal: AgentOS

An MCP-Native Operating System for Autonomous Multi-Agent AI Systems

Student Name: Kranthi Gadi

Date: November 18, 2025

Executive Summary

This literature review examines the current state of multi-agent AI systems, operating system abstractions for agents, and the Model Context Protocol (MCP). Through systematic analysis of 195+ academic papers, industry frameworks, and technical implementations, I identify a significant research gap: **no existing system provides a unified, MCP-native operating system architecture that integrates agent process management, hierarchical memory, and security controls into a cohesive platform.**

I propose **AgentOS**, a novel operating system built entirely on MCP principles that treats tools as system resources and agents as managed processes. This research addresses critical limitations in current multi-agent systems—fragmented context management, lack of standardized tool access, unreliable long-running agents, and absence of OS-level abstractions for coordination. AgentOS represents a foundational contribution to the emerging field of agent operating systems with clear differentiation from existing work.

Research Objectives

This proposal aims to:

1. Design and implement AgentOS, an MCP-native operating system for autonomous multi-agent coordination
2. Develop OS-level abstractions treating agents as processes and tools as resources
3. Create a hierarchical memory system (L1/L2/L3/L4) for persistent agent context
4. Implement a zero-trust firewall for secure, auditable tool access
5. Benchmark AgentOS against leading frameworks (LangChain, CrewAI, AutoGen) on performance, reliability, and security metrics
6. Validate the approach through real-world multi-agent workflows

Literature Review

Model Context Protocol and Agent Architectures

MCP Foundation and Adoption

The Model Context Protocol represents a standardization effort for agent-tool interaction. Anthropic's introduction of MCP established a client-server architecture where agents (clients) access tools (servers) through standardized JSON-RPC communication. Recent industry adoption includes VS Code integration, Cursor IDE support, and AWS AgentCore Gateway implementations.

Research by Zhang et al. on "Advancing Multi-Agent Systems Through Model Context Protocol" demonstrates MCP's effectiveness for multi-agent coordination, showing improved context management and tool sharing across agent populations. However, this work focuses on framework-level patterns rather than OS-level abstractions.

Agent Architecture Patterns

Microsoft's AI Agent Orchestration Patterns document identifies common architectural approaches: reflection, tool use, planning, and multi-agent collaboration. Anthropic's guide to building effective agents emphasizes the importance of context engineering and iterative development. Salesforce's enterprise agentic architecture research highlights the need for robust coordination mechanisms in production deployments.

These industry frameworks provide valuable patterns but **do not propose operating system abstractions** for agent management. Agent coordination remains at the application layer rather than being elevated to OS-level primitives.

Operating Systems for AI Agents

1. AIOS: LLM Agent Operating System

The most closely related work is **AIOS** (AI Agent Operating System) by researchers at Rutgers University and Ohio State University, published at COLM 2025. AIOS provides an LLM agent kernel with four key modules:

1. Agent Scheduler: Manages agent execution order and resource allocation
2. Context Manager: Handles prompt and context window optimization
3. Memory Manager: Provides short-term storage for agent interactions
4. Tool Manager: Coordinates external tool access

AIOS achieves $2.1\times$ faster execution compared to sequential agent serving and demonstrates significant resource efficiency improvements. However, **AIOS has critical limitations**:

- **Not MCP-native:** Uses proprietary tool interfaces rather than MCP standard

- **Limited memory hierarchy:** Single-tier memory without episodic or cold storage layers
- **No integrated security:** Lacks zero-trust firewall or policy-based access control
- **Optimization focus:** Designed for LLM serving efficiency rather than multi-agent orchestration

2. SchedCP: MCP-Based Scheduler Optimization

SchedCP introduces an LLM agent framework for Linux scheduler optimization using MCP and eBPF. The system employs a multi-agent architecture (Observation, Planning, Execution, Learning agents) to automatically tune kernel schedulers, achieving 1.79× performance improvement on kernel compilation benchmarks.

While SchedCP demonstrates MCP's viability for systems-level tasks, it addresses a **narrow domain** (OS scheduler optimization) rather than providing a general-purpose agent operating system. It serves as an MCP server for specific scheduling tasks, not as a complete OS architecture.

Industry Agent Operating Systems

Several industry efforts use "agent operating system" terminology:

- **PwC AI Agent OS:** Enterprise platform for business process automation
- **SmythOS:** Commercial platform for autonomous agent workflows
- **Yodaplus AgentOS:** AI workflow orchestration system

These commercial offerings provide high-level workflow orchestration but lack the technical depth of true operating system abstractions. They function as application platforms rather than OS kernels.

3. Hierarchical Memory for Multi-Agent Systems

G-Memory: Graph-Based Hierarchical Memory

Recent work by researchers at Zhejiang University introduces G-Memory, a three-tier hierarchical memory system using graph structures:

- **Insight Graph:** High-level summaries and key learnings
- **Query Graph:** Historical queries and response patterns
- **Interaction Graph:** Detailed turn-by-turn conversation history

G-Memory demonstrates 20.89% performance improvement on multi-agent benchmarks compared to flat memory approaches. However, G-Memory operates as a **standalone module** without integration into OS architecture or MCP protocol. The graph-based approach also differs architecturally from the L1/L2/L3/L4 cache-inspired hierarchy proposed for AgentOS.

Vector Stores and Episodic Memory

Extensive research explores vector databases for LLM memory. Vector stores enable semantic similarity search across agent experiences, supporting retrieval-augmented generation (RAG) patterns. Episodic memory systems for multi-agent reinforcement learning show that agents benefit from structured experience replay.

Key findings from memory research:

- **Semantic retrieval:** Vector embeddings enable context-aware memory access
- **Hierarchical benefit:** Multi-tier memory outperforms single-tier approaches
- **Persistence requirement:** Long-running agents need durable memory beyond conversation context

Research gap: No system integrates vector memory, episodic storage, structured schemas, and cold storage into a unified OS-managed hierarchy with kernel-level coordination.

4. Security and Access Control for AI Agents

Zero-Trust Architecture for Agents

Recent security research emphasizes the need for zero-trust principles in agentic AI systems. Key security challenges include:

1. **Unrestricted tool access:** Agents with broad permissions pose significant risk
2. **Lack of auditability:** Many systems do not log agent actions comprehensively
3. **Prompt injection vulnerabilities:** Malicious inputs can manipulate agent behavior
4. **Data exfiltration risks:** Agents may inadvertently expose sensitive information

Prefactor's best practices for AI agent access control recommend role-based access control (RBAC), least-privilege principles, and continuous monitoring. Cisco's work on redefining zero-trust for AI agents advocates for "never trust, always verify" approaches to agent-tool interaction.

MCP Security Analysis

Recent security assessments of MCP identify critical vulnerabilities:

- **Server trust model:** Clients must trust MCP servers, creating attack surface
- **Authentication gaps:** Limited standardization of auth mechanisms
- **Data validation:** Insufficient input/output sanitization in many implementations

The MCP Security Checklist project provides guidelines for secure MCP deployments. However, **no existing work integrates security controls into the OS kernel layer** as a fundamental system component rather than an add-on.

5. Multi-Agent Orchestration Frameworks

LangChain and LangGraph

LangChain's LangGraph framework provides declarative multi-agent orchestration using directed graphs. Agents function as graph nodes with edges representing communication channels. LangGraph supports hierarchical agent architectures, parallel execution, and state persistence.

Key capabilities demonstrated in LangGraph research:

- **Supervisor patterns:** Coordinator agents route tasks to specialized workers
- **State machines:** Complex workflows expressed as stateful graphs
- **Streaming support:** Real-time agent output for user feedback

Limitation: LangGraph operates at the application framework level without OS abstractions. Agents are Python functions rather than OS-managed processes.

CrewAI and AutoGen

CrewAI provides role-based multi-agent orchestration where agents have defined roles, goals, and tools. AutoGen (now AG2) from Microsoft Research enables conversable agents with flexible interaction patterns.

Comparative analysis shows:

Framework	Best For	Agent Model	Memory
LangGraph	Complex workflows	Graph nodes	Basic state
CrewAI	Role-based tasks	Crew members	Shared context
AutoGen	Conversational	Chat participants	Message history

Table 1: Comparison of leading multi-agent frameworks

Common limitation: All frameworks treat agents as application-level constructs without operating system process abstractions, resource management, or kernel scheduling.

Benchmarking and Evaluation

Agent Evaluation Frameworks

Research on agent evaluation identifies critical metrics:

- **Task success rate:** Percentage of successfully completed objectives
- **Efficiency:** Time and resource consumption per task
- **Reliability:** Consistency across multiple executions
- **Safety:** Adherence to constraints and policies

LangChain's benchmarking of multi-agent architectures shows significant performance variation across orchestration patterns. Supervisor architectures achieve 15-20% higher success rates than peer-to-peer coordination on complex tasks.

Performance Testing Methodologies

Studies on LLM reliability evaluation emphasize the importance of:

1. **Stress testing:** High-concurrency scenarios revealing bottlenecks
2. **Failure injection:** Testing recovery mechanisms and error handling
3. **Latency analysis:** Measuring end-to-end response times
4. **Resource profiling:** Memory, compute, and network utilization
5. **Research opportunity:** AgentOS benchmarking should compare OS-level optimizations against framework-level approaches, measuring overhead of kernel abstractions versus benefits of integrated coordination.

Research Gap Analysis

What Exists vs. What Is Missing

Based on comprehensive literature review, the current landscape shows:

Feature	AgentOS (Your Proposal)	AIOS	SchedC P	MCP Framework s	Memory Systems
MCP-Native Architecture	✓ Core foundation	✗	✓ Limited	⚠ Partial	✗
Agent Kernel/Scheduler	✓ OS-style	✓ LLM-focused	✓ OS scheduler only	✗	✗
L1/L2/L3/L4 Memory	✓ Unified hierarchy	✗	✗	✗	⚠ Separate implementations
Zero-Trust Firewall	✓ Kernel-integrated	✗	✗	✗	✗
Tools as MCP Resources	✓	✗	⚠ Limited	✓	✗
Agents as Processes	✓	⚠ Partial	✗	✗	✗
Multi-Agent Orchestration	✓ OS-level	⚠ Resource mgmt	✗	✓ Framework-level	✗
Benchmarking Against Frameworks	✓ Planned	⚠ Speed only	⚠ Scheduler only	✗	✗

Feature	AgentOS (Your Proposal)	AIOS	SchedC P	MCP Framework s	Memory Systems
Publication Status	 Proposed	 COLM 2025	 arXiv 2025	 Various	 Various

Unique Contributions of AgentOS

The proposed AgentOS system addresses gaps through:

1. **MCP-Native Architecture:** First operating system treating MCP as foundational protocol layer, with tools as system resources accessed via standardized interfaces
2. **Agent Process Model:** OS-level abstractions where agents are managed processes with scheduling, context switching, and resource allocation—extending AIOS concept to MCP-native design
3. **Unified Memory Hierarchy:** Four-tier system (L1: dialogue buffer, L2: episodic vectors, L3: structured schemas, L4: cold storage) integrated into OS kernel rather than standalone module
4. **Kernel-Integrated Security:** Zero-trust firewall as fundamental OS component validating all tool access, not an application-layer add-on
5. **Holistic System Design:** All components (kernel, memory, security, MCP) architected as unified OS rather than separate frameworks

Proposed AgentOS Architecture

System Overview

AgentOS consists of five primary components:

1. **Agent Kernel:** Scheduler managing agent lifecycles, context routing, and resource allocation
2. **MCP Tool Drivers:** Standardized interfaces to filesystem, browser, databases, APIs
3. **Hierarchical Memory System:** L1/L2/L3/L4 tiers providing short-term to long-term storage
4. **Zero-Trust Firewall:** Policy engine validating and auditing all tool interactions
5. **Multi-Agent Orchestrator:** Coordination layer for specialized agent processes

Agent Kernel Design

The kernel provides OS-like services:

- **Process Management:** Agent creation, termination, state tracking
- **Scheduling:** Fair allocation of compute resources across agents
- **Context Switching:** Efficient transition between agent executions
- **Inter-Agent Communication:** Message passing and shared memory primitives
- **Resource Quotas:** Limits on memory, tokens, and tool calls per agent

Unlike AIOS which optimizes for LLM serving efficiency, AgentOS kernel prioritizes multi-agent coordination and MCP-native tool access.

Hierarchical Memory Architecture

Memory tiers provide performance-capacity tradeoffs:

L1 - Dialogue Buffer (Highest performance, ~10K tokens):

- Current conversation context
- Immediate agent state
- Fast in-memory access

L2 - Episodic Vectors (Fast retrieval, ~1M embeddings):

- Vector database of agent experiences
- Semantic similarity search
- RAG-enabled context retrieval

L3 - Structured Schemas (Organized storage, ~100MB):

- JSON/SQLite for workflow state
- Agent memory graphs
- Task histories and outcomes

L4 - Cold Storage (Unlimited capacity, disk-based):

- Long-term archival
- Full interaction logs
- Compliance and audit records

This hierarchy draws inspiration from CPU cache levels while adapting to agent-specific access patterns. L1/L2 provide hot paths for active agents, while L3/L4 enable long-term memory and auditability.

Zero-Trust Firewall

Security enforcement at kernel level:

1. **Policy Definition:** Administrators specify allowed tools, data access, and action types per agent
2. **Runtime Validation:** Every MCP tool call passes through firewall validation
3. **Risk Scoring:** Assign risk levels to operations (e.g., file read = low, file delete = high)
4. **Audit Logging:** Comprehensive recording of all tool interactions for compliance
5. **Anomaly Detection:** Flag unusual patterns suggesting compromised agents

Integration at kernel level ensures security cannot be bypassed and applies uniformly across all agents.

MCP Tool Drivers

Standardized resource interfaces:

- **Filesystem Driver:** Read/write files, directory operations
- **HTTP/Browser Driver:** Web scraping, API calls, page interaction
- **Database Driver:** SQL queries, transactions, schema introspection
- **Vector Store Driver:** Embedding storage, similarity search
- **Logging Driver:** Structured audit trail, performance metrics

All drivers implement MCP protocol, enabling tool discovery, capability negotiation, and standardized error handling.

Research Methodology

Implementation Plan

Phase 1: Core Infrastructure (Weeks 1-3)

- Implement MCP protocol stack (client/server)
- Develop basic agent kernel with process management
- Create filesystem and logging tool drivers
- Establish testing framework

Phase 2: Memory System (Weeks 4-5)

- Build L1 dialogue buffer (in-memory)
- Integrate L2 vector store (ChromaDB/Qdrant)
- Implement L3 structured storage (SQLite)

- Add L4 cold storage (disk-based)
- Develop memory tier promotion/demotion logic

Phase 3: Security Layer (Week 6)

- Design policy specification language
- Implement firewall validation engine
- Build audit logging system
- Add risk scoring framework

Phase 4: Multi-Agent Orchestration (Weeks 7-8)

- Create specialized agent types (retrieval, planning, reasoning, execution)
- Implement inter-agent communication primitives
- Build supervisor/worker coordination patterns
- Add context routing between agents

Phase 5: Benchmarking (Weeks 9-10)

- Design test suite covering diverse workflows
- Implement comparison harness for LangChain, CrewAI, AutoGen
- Measure performance metrics (latency, throughput, memory)
- Evaluate reliability (success rate, error recovery)
- Assess security (policy enforcement, audit completeness)

Phase 6: Documentation and Publication (Weeks 11-12)

- Write comprehensive technical documentation
- Prepare research paper draft
- Create presentation materials
- Develop demonstration workflows

Evaluation Metrics

Performance Benchmarks:

- End-to-end task completion time
- Tool call latency overhead
- Memory access times across L1/L2/L3/L4 tiers
- Kernel scheduling overhead
- Maximum concurrent agents supported

Reliability Metrics:

- Task success rate under nominal conditions
- Recovery time from agent failures
- Context preservation across long-running workflows
- Memory consistency guarantees

Security Assessment:

- Policy enforcement accuracy (true positive rate)
- Audit log completeness
- Anomaly detection false positive rate
- Attack resistance (prompt injection, privilege escalation)

Comparative Analysis

Benchmark AgentOS against:

1. **LangGraph:** State machine multi-agent workflows
2. **CrewAI:** Role-based team coordination
3. **AutoGen:** Conversational multi-agent systems
4. **AIOS:** Direct comparison of kernel approaches

Test scenarios include:

- Multi-step research workflows (retrieval + analysis + synthesis)
- Parallel task execution (concurrent agent processes)
- Long-running autonomous workflows (multi-hour execution)
- Tool-intensive operations (database queries, web scraping, file I/O)

Expected Contributions

Technical Innovations

1. **First MCP-native OS architecture** treating protocol as system foundation
2. **Novel memory hierarchy** (L1/L2/L3/L4) tailored to agent access patterns
3. **Kernel-integrated security** with zero-trust firewall as OS component
4. **Process model for agents** enabling OS-level scheduling and resource management
5. **Comprehensive benchmarking** of OS vs. framework approaches

Academic Impact

Primary Contribution: Establishing operating system abstractions as a superior approach to multi-agent coordination compared to application frameworks.

Secondary Contributions:

- Reference implementation for MCP-based systems
- Security model for agentic AI (zero-trust firewall design)
- Memory architecture patterns for LLM agents
- Benchmarking methodology for agent system evaluation

Publication Targets

Primary Venue: MLSys (Conference on Machine Learning and Systems) or OSDI (USENIX Symposium on Operating Systems Design and Implementation)

Secondary Venues:

- AAMAS (Autonomous Agents and Multi-Agent Systems) - for multi-agent coordination
- USENIX Security - for zero-trust firewall contribution
- NeurIPS Systems Track - for ML systems architecture

Workshop Opportunities: MCP summits, AI safety workshops, systems for ML venues

Challenges and Mitigation Strategies

Technical Challenges

Challenge 1: MCP Protocol Maturity

- Risk: MCP standard is evolving, potential breaking changes
- Mitigation: Abstract protocol details behind adapter layer, enable version compatibility

Challenge 2: Performance Overhead

- Risk: OS abstractions may introduce latency compared to direct framework usage
- Mitigation: Optimize critical paths, implement caching, benchmark thoroughly to quantify tradeoffs

Challenge 3: Memory Tier Coherency

- Risk: Keeping L1/L2/L3/L4 synchronized is complex

- Mitigation: Clear promotion/demotion policies, eventual consistency model, comprehensive testing

Challenge 4: Security Policy Complexity

- Risk: Policy language may be difficult to specify correctly
- Mitigation: Start with simple RBAC model, provide policy validation tools, iterative refinement

Research Challenges

Challenge 5: Fair Comparison to Frameworks

- Risk: Different frameworks excel at different tasks, biased benchmark design
- Mitigation: Diverse test suite, metrics across multiple dimensions, transparent methodology

Challenge 6: Novelty Communication

- Risk: Reviewers may not appreciate distinction from AIOS or existing frameworks
 - Mitigation: Clear positioning in related work, explicit contribution enumeration, strong evaluation
-

Timeline and Milestones

Week	Milestone	Deliverable
1	Literature review complete	This document
2-3	MCP infrastructure	Core protocol stack
4-5	Memory system	L1/L2/L3/L4 implementation
6	Security layer	Zero-trust firewall
7-8	Multi-agent orchestration	Agent coordination
9-10	Benchmarking experiments	Performance data
11	Research paper draft	Conference submission
12	Final presentation	Project defense

Table 3: 12-week project timeline

Key Deliverables

1. **Working AgentOS prototype** with all major components implemented
2. **Benchmark results** comparing AgentOS to LangChain, CrewAI, AutoGen
3. **Research paper** (8-10 pages conference format) with evaluation
4. **Open-source release** enabling community adoption and validation
5. **Documentation** including architecture guide, API reference, tutorials
6. **Presentation** for academic defense and conference submission

Conclusion

This literature review demonstrates that while individual components of agent operating systems exist in isolation—kernels (AIOS), hierarchical memory (G-Memory), security frameworks (zero-trust research), and orchestration patterns (LangChain/CrewAI)—**no existing system unifies these elements into an MCP-native operating system architecture.**

The proposed AgentOS addresses this gap through:

- **MCP as foundational protocol:** Tools as OS resources, standardized interfaces
- **Agent process model:** OS-level scheduling, isolation, and resource management
- **Integrated memory hierarchy:** L1/L2/L3/L4 tiers managed by kernel
- **Kernel-integrated security:** Zero-trust firewall as system component
- **Rigorous evaluation:** Benchmarking against state-of-the-art frameworks

This research has the potential to establish operating system abstractions as the preferred approach for robust, scalable, and secure multi-agent AI systems. The work builds on solid foundations (AIOS kernel concepts, MCP protocol, hierarchical memory research) while making clear novel contributions that advance the state of the art.

I am excited to pursue this research and believe it represents a significant contribution to the emerging field of agent operating systems. I welcome your feedback and guidance as I proceed with implementation and evaluation.

References

- [1] AIOS research paper. *AIOS: LLM Agent Operating System*. COLM 2025.
<https://arxiv.org/abs/2403.16971>
- [2] Anthropic. *Building Effective AI Agents*. Technical Report, December 2024.
- [3] Anthropic. *Introducing the Model Context Protocol*. November 2024.
<https://www.anthropic.com/news/model-context-protocol>
- [4] Google Cloud. *What is Model Context Protocol (MCP)? A guide*. November 2025.
<https://cloud.google.com/model-context-protocol>
- [5] Descope. *What Is the Model Context Protocol (MCP) and How It Works*. September 2025.
- [6] AIXplore. *Agent architectures with MCP*. Technical article, March 2025.
- [7] Jeeva AI. *Multi-Agent Coordination Playbook (MCP & AI Teamwork)*. November 2025.
- [8] Visual Studio Code Documentation. *Use MCP servers in VS Code*. November 2025.
- [9] AWS. *Unite MCP servers through AgentCore Gateway*. November 2025.
- [10] MCP Market. *Multi-Agent Coordination for Cursor IDE*. December 2024.
- [11] Zhang et al. *Advancing Multi-Agent Systems Through Model Context Protocol*. arXiv:2504.21030, April 2025.
- [12] Microsoft Azure. *AI Agent Orchestration Patterns*. Azure Architecture Center, July 2025.
- [13] Anthropic. *Building Effective AI Agents*. December 2024.
- [14] Salesforce Architect. *Enterprise Agentic Architecture and Design Patterns*. December 2024.
- [15] Mei et al. *AIOS: LLM Agent Operating System*. COLM 2025, arXiv:2403.16971.
- [16] Rutgers University. *Large Language Model Agent Operating Systems*. Technology disclosure, March 2025.
- [17] AIOS Foundation. *AIOS Documentation*. <https://docs.aios.foundation>
- [18] Chen et al. *Towards Agentic OS: An LLM Agent Framework for Linux Schedulers*. arXiv preprint, August 2025.
- [19] Linux Journal. *Self-Tuning Linux Kernels: How LLM-Driven Agents Are Reinventing Scheduler Optimization*. October 2025.

- [20] ChatPaper. *Towards Agentic OS: An LLM Agent Framework for Linux Schedulers*. Technical analysis, November 2025.
- [21] PwC. *PwC launches AI Agent Operating System for enterprises*. March 2025.
- [22] PwC. *AI agent operating system to revolutionise AI deployment*. June 2025.
- [23] SmythOS. *Understanding Autonomous Agent Architecture*. June 2025.
- [24] Yodaplus. *Is AgentOS the Next Operating System for AI Workflows?* September 2025.
- [25] Wang et al. *Tracing Hierarchical Memory for Multi-Agent Systems*. arXiv preprint, June 2025.
- [26] OpenReview. *Tracing Hierarchical Memory for Multi-Agent Systems*. Conference submission, 2025.
- [27] GitHub bingreeky/GMemory. *G-Memory: Hierarchical Memory System*. Repository, May 2025.
- [28] FreeCodeCamp. *How AI Agents Remember Things: The Role of Vector Stores in LLM Memory*. July 2025.
- [29] Dev.to. *Long Term Memory for LLMs using Vector Store*. July 2025.
- [30] MarktechPost. *Comparing Memory Systems for LLM Agents: Vector, Graph, and Hybrid Approaches*. November 2025.
- [31] Letta. *Agent Memory: How to Build Agents that Learn and Remember*. July 2025.
- [32] NeurIPS Proceedings. *Episodic Multi-agent Reinforcement Learning with Curiosity-driven Exploration*. 2024.
- [33] arXiv. *Efficient Episodic Memory Utilization of Cooperative Multi-Agent Reinforcement Learning*. March 2024.
- [34] PMC/NCBI. *Elements of episodic memory: insights from artificial agents*. September 2024.
- [35] Anthropic. *Effective context engineering for AI agents*. September 2025.
- [36] Galileo AI. *Deep Dive into Context Engineering for Agents*. September 2025.
- [37] BankInfoSecurity. *AI, Zero Trust and Guardrails for Secure Innovation*. November 2025.
- [38] Prefactor. *5 Best Practices for AI Agent Access Control*. November 2025.
- [39] Levo.ai. *Zero Trust Architecture for AI-Driven Market Leadership*. November 2025.

- [40] Cisco Blogs. *Redefining Zero Trust in the Age of AI Agents and Agentic Workflows*. June 2025.
- [41] BleepingComputer. *Extending Zero Trust to AI Agents: "Never Trust, Always Verify" Goes Agentic*. November 2025.
- [42] Protecto.ai. *Monitoring And Auditing LLM Interactions For Security Breaches*. August 2024.
- [43] DataSunrise. *Audit Logging for AI & LLM Systems: Essential Security*. June 2025.
- [44] Red Hat Security. *Model Context Protocol (MCP): Understanding security risks*. July 2025.
- [45] Impact Analytics. *AI Agent Might Be Your Biggest Security Vulnerability*. October 2025.
- [46] DataDome. *MCP Security: Protect your AI infrastructure from threats*. October 2025.
- [47] arXiv. *Model Context Protocol (MCP): Landscape, Security Threats, and Mitigation Strategies*. March 2025.
- [48] Bitdefender. *Security Risks of Agentic AI: A Model Context Protocol (MCP) Perspective*. September 2025.
- [49] GitHub. *MCP Security Checklist: A Security Guide for the AI Tool Ecosystem*. April 2025.
- [50] LangChain Blog. *LangGraph: Multi-Agent Workflows*. February 2025.
- [51] YouTube. *Fully local multi-agent systems with LangGraph*. March 2025.
- [52] LangChain Documentation. *Multi-agent systems*. <https://docs.langchain.com/multi-agent>
- [53] Sparkco AI. *CrewAI vs AutoGen: Multi-Agent Orchestration* 2025. November 2025.
- [54] LogRocket Blog. *Autogen vs. Crew AI: Choosing the right agentic framework*. November 2025.
- [55] ZenML. *CrewAI vs AutoGen: Which One Is the Best Framework to Build AI Agents?* August 2025.
- [56] Leanware. *Agent Evaluation Frameworks: Methods, Metrics & Best Practices*. October 2025.
- [57] Samiranama. *Evaluating LLM-based Agents: Metrics, Benchmarks, and Best Practices*. July 2025.

[58] VirtuosoQA. *Multi-Agent Testing Systems: How Cooperative AI Agents Validate Complex Software*. August 2025.

[59] LangChain Blog. *Benchmarking Multi-Agent Architectures*. June 2025.

[60] Galileo AI. *LLM Reliability Evaluation Methods to Prevent Production Failures*. November 2025.

[61] Maxim AI. *How to Ensure Reliability in LLM Applications*. November 2025.