ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Науковий вісник НЛТУ України
Scientific Bulletin of UNFU
https://doi.org/10.36930/40350606
Article received 07.11.2025
Article accepted 22.12.2025
UDC 004.89
Correspondence author: R. I. Saveliev romansavelyev6@gmail.com

ISSN 1994-7836 (print)
ISSN 2519-2477 (online)

R. I. Saveliev, M. V. Dendiuk
Ukrainian National Forestry University, Lviv, Ukraine

LEVERAGING THE MODEL CONTEXT PROTOCOL ARCHITECTURE FOR DATA ANALYTICS IN SOFTWARE DISTRIBUTED SYSTEMS

Recent progress in distributed AI systems intensifies the demand for reliable coordination, unified context exchange, and efficient communication among autonomous analytical agents operating across dynamic, heterogeneous environments. This study analyzes these challenges and introduces the Model Context Protocol (MCP) as a structured and extensible solution that enhances context-driven interaction within distributed software ecosystems. The proposed framework integrates layered context exchange, persistent memory structures, deterministic communication channels, and standardized interaction patterns, aligning independent analytical models into a cohesive system capable of real-time information acquisition, multi-stage task execution, and seamless integration of structured knowledge from diverse and continuously evolving data sources. MCP strengthens cooperative and system-level reasoning by enabling agents to synchronize memory states, maintain contextual continuity, and dynamically adjust analytical workflows as operational conditions evolve, even under high variability and unpredictable workloads. The architecture incorporates automated task decomposition, adaptive memory alignment, context persistence, robust tool invocation, and flexible routing logic, supporting hybrid deployments across edge devices, local clusters, and cloud infrastructures while maintaining high reliability and analytical stability. The study evaluates MCP through federated learning simulations, distributed workflow experiments, and comparative benchmarks, demonstrating consistent improvements in inference latency, throughput efficiency, policy adaptability, context-aware decision-making, and multi-agent collaboration across complex distributed environments. These results confirm MCP's ability to reduce integration overhead, mitigate data fragmentation, stabilize communication pathways, enhance analytical robustness, and improve accuracy in both small-scale and enterprise-level scenarios. Consequently, MCP establishes a scalable foundation for resilient analytical pipelines and significantly expands the practical adoption of autonomous agents in enterprise automation, distributed decision-support systems, and next-generation cognitive infrastructures.

Keywords: autonomous agents; orchestration layer; memory synchronization; enterprise AI systems; cognitive coordination.

Introduction / Вступ

Modern software systems run on the cloud, edge, and on-premises infrastructure. This creates a highly distributed environment filled with data. These settings produce vast amounts of information. Organizations must understand this data in real time to stay responsive and competitive. However, examining data across such distributed systems is difficult. The information often resides in separate silos across nodes or services. Most traditional big data tools, like Hadoop or Spark, focus more on parallel computation than on enabling systems to work together effectively. As a result, engineers and analysts often face issues with fragmented data, limited context awareness, and awkward communication when trying to extract meaningful insights from scattered sources.

This paper tackles the challenge of making distributed data analytics more coherent and collaborative. Specifically, we focus on improving how different parts of a system share context and work together intelligently. One of the key reasons for this work is the growing use of software agents that do more than follow pre-set instructions. They can explore, decide, and act based on what they learn from their surroundings. These AI-driven agents don't just work with local data; they also look for new input from their surroundings or other services. In theory, multiple agents could divide a complex analytic task, with each focusing on a different part. They could then come together to share their findings. This kind of teamwork could lead to faster and more in-depth analysis than any single, centralized model. In practice, though, real-time cooperation among such agents remains a big hurdle. Many of them can't easily access the tools or data they need, especially when those resources live on different networks or plat-

Інформація про авторів:
Савельєв Роман Ігорович, аспірант, кафедра комп'ютерних наук. **Email:** romansavelyev6@gmail.com; https://orcid.org/0009-0007-8092-0673
Дендюк Михайло Володимирович, канд. техн. наук, доцент, кафедра комп'ютерних наук. **Email:** dendyuk@nltu.edu.ua; https://orcid.org/0000-0002-7631-022X
Цитування за ДСТУ: Савельєв Р. І., Дендюк М. В. Використання архітектури протоколу контексту моделі для аналізу даних у розподілених програмних системах. Науковий вісник НЛТУ України. 2025, т. 35, № 6. С. 49–59.
Citation APA: Saveliev, R. I., & Dendiuk, M. V. (2025). Leveraging the model context protocol architecture for data analytics in software distributed systems. *Scientific Bulletin of UNFU*, 35(6), 49–59. https://doi.org/10.36930/40350606

forms. Even more problematic, there's no common standard that tells agents how to share what they've found or understood in a way that's secure, structured, and easy to interpret.

MCP provides a unified structure and communication protocol that enable agents, models, and tools to understand one another when sharing contextual data. Think of it as a "USB-C for AI systems", a way to connect different components without needing to create new custom code each time. Instead of relying on brittle, one-off integrations, developers can use MCP's standard format to share what agents know, what they're doing, and what they've discovered. This helps create shared awareness across the system. For example, an analytics agent running on one server can subscribe to context updates from another agent working elsewhere, and adjust its work in real time. That kind of back-and-forth enables distributed systems to behave more like a coordinated team rather than isolated nodes.

In this paper, we explore how MCP can bring new life to distributed data analytics. Our approach combines two areas that have developed mainly separately: large-scale data processing and intelligent multi-agent collaboration. Existing data analytics platforms rarely support intelligent context sharing, while agent-based AI frameworks have seldom been applied to traditional analytics workflows. MCP can connect these two worlds. Our work has three main goals. First, we define an architecture in which agents use MCP to perform distributed analytics collaboratively. Second, we build a prototype to test whether sharing context through MCP leads to better results, such as more accurate insights or less wasted effort. Third, we compare this MCP-based system to conventional distributed pipelines to measure the benefits of coordination. By letting agents share insights in real time, we believe MCP can help turn fragmented systems into intelligent, connected analytics networks that act with purpose and precision.

Object of research – context-oriented coordination and analytical interaction between distributed components in modern software systems.

Subject of research – architectural principles, communication protocols, and coordination methods that enable real-time context exchange, memory synchronization, and intelligent collaboration between analytical agents, ensuring higher consistency, adaptability, and efficiency in distributed data analytics.

The purpose *of the* research – to develop and substantiate a software system architecture based on the Model Context Protocol (MCP) that significantly improves context-oriented data analytics and intelligent coordination between analytical agents in distributed software systems.

To achieve this *purpose*, the following main *research objectives are identified*:

- to develop an MCP-based architectural model for distributed data analytics, which will allow agents to exchange relevant context in real time and overcome data fragmentation across distributed system components;
- to implement a prototype system that demonstrates MCP-driven coordination among analytical agents, which will make it possible to assess improvements in performance, consistency, and adaptivity compared to traditional analytics pipelines;
- to evaluate the proposed approach using simulated distributed workloads and comparative benchmarks, which will ensure objective measurement of MCP's advantages in latency reduction, throughput improvement, and minimization of redundant computation;
- to investigate the influence of persistent memory, event-driven routing, and context synchronization within MCP, which will help determine how these mechanisms affect accuracy, stability, and efficiency of analytical processes;
- to identify practical application domains for MCP in cloud, edge, and hybrid infrastructures, which will make it possible to define the protocol's potential for building scalable, context-aware analytical systems of the next generation.

Analysis of recent research and publications. In the study [8], the authors examined the fragmentation of observability data in distributed environments and showed that dividing telemetry across multiple systems significantly complicates incident correlation and increases analysis time. They demonstrated that applying a unified context framework improves cross-platform visibility and establishes a more coherent analytical environment. These results highlight the necessity for standardized protocols, such as MCP, that consolidate heterogeneous telemetry sources into a unified context layer.

In the study [15], the researchers proposed an MCP-oriented architectural approach for enterprise AI integration and emphasized time-aware sampling as a method for minimizing redundant historical data within context windows. Their findings demonstrated that reducing low-variance segments of historical data preserves analytical fidelity while lowering context size, enabling large-scale analytics within constrained LLM context limits. This supports the need for adaptive context-management mechanisms in distributed analytical workflows.

According to [4], the authors investigated the integration of MCP with cloud-native observability tools and demonstrated that summarization-based preprocessing significantly reduces the overhead of handling unstructured log streams. Their method produced compact, semantically enriched representations of system behavior, which accelerated anomaly detection and root-cause identification. These findings confirm that MCP-driven semantic context extraction improves the quality and efficiency of distributed system diagnostics.

In the work by [17], the researchers analyzed context-optimization techniques and emphasized the roles of multi-layer caching and domain-specific compression in scalable observability architectures. They showed that caching frequently accessed context reduces retrieval latency, while telemetry-adaptive compression improves token efficiency for analytical models. These findings demonstrate that structured context optimization aligned with MCP principles can support high-volume distributed analytics.

In the study [13], the authors examined structured reasoning techniques for complex analytical tasks and demonstrated that multi-step reasoning benefits significantly from stable intermediate context states. Their results showed that the absence of synchronized context leads to inconsistent analytical outputs. This reinforces the need for context-preserving architectures that maintain coherence across distributed analytical components, consistent with the goals of MCP.

In the work by [20], the researchers evaluated the performance of analytical models integrated with large sets of external APIs and highlighted integration inconsistency as a critical obstacle for automation. They concluded that distributed systems require standardized interfaces to ensure con-

sistent tool invocation and reproducible analytical workflows. These observations support the relevance of MCP, which provides such standardized interaction mechanisms.

In summary, the reviewed studies collectively demonstrate significant progress in observability, context optimization, distributed reasoning, and tool-augmented analytics, yet none of them introduce an integrated protocol that simultaneously unifies context exchange, preserves synchronized state, and supports scalable distributed analytical pipelines. This gap confirms the necessity of the present research, which positions MCP as a comprehensive architectural foundation for context-driven analytics in modern distributed software systems.

Research results and their discussion / Результати дослідження та їх обговорення

Model Context Protocol* (*MCP*) *Theoretical Foundations. The Model Context Protocol (MCP) is a recently introduced open standard (first released in late 2024 by Anthropic) that enables large language models (LLMs) to interact with external data, tools, and services in a stateful, context-rich manner. At its core, MCP provides a *persistent, evolving context layer* for AI agents, allowing them to retain session memory and learn or adapt across multi-step interactions. This is a departure from traditional stateless API calls, where each request is independent. Instead, MCP maintains *structured memory* (shared context, task state, relevant data) across sessions.

MCP's design is underpinned by three fundamental pillars: Statefulness, meaning it preserves both short-term session context and long-term memory of interactions; Interoperability, ensuring the protocol works across diverse models, tools, and data sources; Agent-Centric Design, emphasizing autonomous decision-making by AI agents within defined boundaries. In practical terms, this means an MCP-enabled agent can remember user preferences and past actions, seamlessly use multiple tools, and make informed autonomous choices within its scope. For example, an *MCP-powered travel assistant* can recall a user's budget, dietary restrictions, and past feedback to dynamically personalize a new itinerary without needing the user to repeat information. The theoretical foundation of MCP thus lies in viewing the AI agent and its environment as a unified contextual system rather than isolated components, akin to a shared memory space for distributed intelligent agents. This theoretical shift provides a basis for more coherent, long-running analytical workflows in distributed systems, as the AI can accumulate knowledge and context over time and across multiple data sources.

Research Directions. As a nascent technology, MCP opens several research directions and unresolved questions. One active area is security and trust in context-sharing: how to prevent malicious manipulation of the shared context and ensure provenance. Current MCP implementations lack robust guarantees of context authenticity, making them vulnerable to attacks such as *context poisoning* (injecting malicious data into the agent's memory) and unauthorized inference. Researchers are thus investigating methods such as cryptographic signing of context data and verifiable lineage tracking to harden MCP's shared memory against tampering. This ties into developing *dynamic trust frameworks* for agents-for example, using just-in-time credentials and context-aware access controls to secure tool use. Another research avenue is multi-agent coordination: MCP, by itself, focuses on one agent's tool use (intra-agent context), but combining it with *agent-to-agent* (*A2A*) communication protocols remains an open challenge. Designing higher-level "agent orchestration" layers or control planes that build on MCP (for internal state sharing) and A2A (for inter-agent negotiation) is an ongoing area of exploration.

Moreover, integrating MCP with existing AI paradigms, such as retrieval-augmented generation (RAG), is a current topic. It has been noted that MCP does not replace the need for knowledge retrieval from documents, so research is exploring hybrid approaches. For instance, one study proposes combining RAG with MCP to address prompt inefficiency when many tools are available. By retrieving only the most relevant tool descriptions for a query, they reduced prompt size by over 50 % and significantly improved tool selection accuracy. This demonstrates a research trend toward optimizing MCP's scalability as the number of tools and data sources increases. Other directions include developing best practices and governance for MCP usage (to avoid over-reliance on LLMs for core logic) and understanding the broader architectural implications of adopting MCP in enterprise systems. Because MCP is very new, the community is actively evaluating its limitations and evolving the standard; indeed, its longevity and evolution depend on open collaboration (currently led by Anthropic's stewardship) and addressing these research challenges.

MCP Architecture. MCP follows a client – server architecture that standardizes how AI models (clients) interface with external tools or data providers (servers). In an MCP setup, the *LLM application* (for example, an AI assistant) acts as the host environment, which initializes connections to one or more MCP servers. Each MCP server is a lightweight process (which can run locally or on a remote service) that exposes a collection of capabilities – such as data queries, computations, or actions – through a defined interface. On the client side, an MCP client component runs within the host application, managing a 1:1 connection to its corresponding server and handling the communication protocol. All communication uses a consistent JSON-RPC 2.0 message format, ensuring that requests, responses, and errors are properly framed and linked within a session. The transport layer for these messages can vary (for example, MCP can use simple stdin/stdout streams for local tools or HTTP for networked tools, often with Server-Sent Events for asynchronous server messages). This decouples the transport mechanism from the core protocol logic.

An MCP server defines "tools" or resources that the LLM can invoke (Figure 1). Each tool has a schema (inputs/outputs) and is exposed via the MCP standardized format. For instance, a server might offer a query_database tool or a fetch_customer_record tool with specified parameters. The LLM, via the MCP client, can discover what tools a server offers and invoke them by sending a JSON-formatted request; the server executes the action and returns results in a structured response. In essence, MCP acts as a universal adapter layer between the AI and external systems. This "universal connector" idea is why it has been dubbed the "*USB-C of AI*" by its proponents – it standardizes the plug interface so that many different tools/services can be accessed uniformly. The architecture also typically includes a context manager: the server can store and retrieve contextual information (state) for the session, enabling

that persistent memory of previous interactions. Summarily, the MCP architecture cleanly separates concerns: the AI agent focuses on higher-level reasoning and deciding *which* tool to use when, while the MCP server encapsulates *how* to perform the tool's function, all mediated by a consistent protocol. This modular architecture simplifies integration and fosters scalability, since developers can add new MCP servers or tools without changing the core AI model, and multiple MCP servers can run in parallel to extend an agent's capabilities.

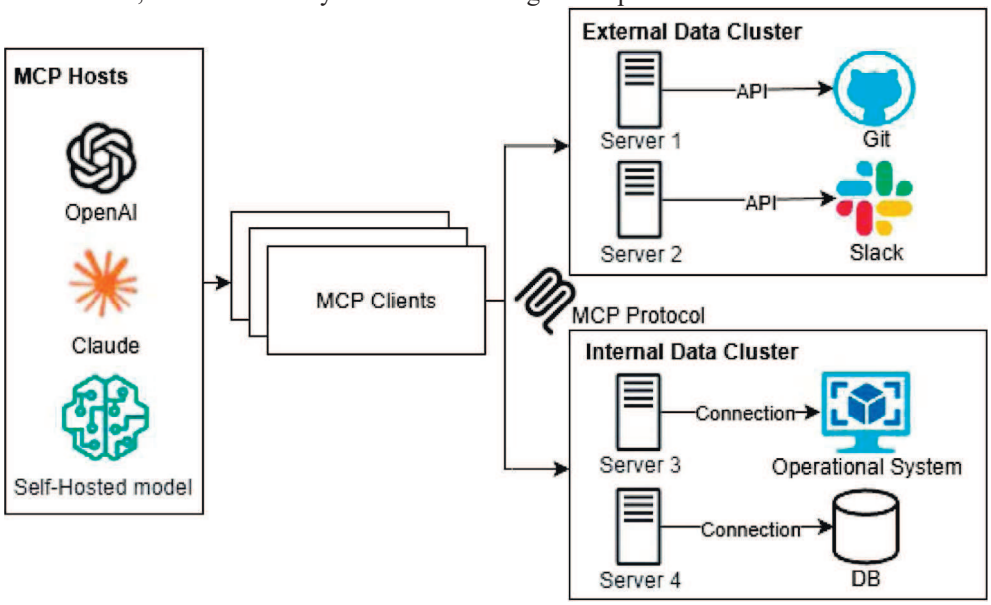

Fig. 1. MCP Architecture with 2 distributed clusters of services (author's own illustration) / MCP Архітектура з двома розподіленими кластерами сервісів

Advantages of MCP for Data Analytics. Leveraging MCP in data analytics scenarios – especially in software distributed systems – offers several notable advantages over traditional approaches:

Interoperability and Integration Speed: MCP provides a standardized interface to connect AI with diverse data sources and analytic tools. This dramatically reduces the need for custom connectors for each data source, avoiding fragmented one-off integrations. Teams can "*build once and scale*" their AI analytics solutions across multiple environments with minimal extra work. In practice, an analytics pipeline can incorporate data from databases, logs, APIs, and streaming services via MCP without bespoke glue code for each, thereby accelerating development.

Enhanced Contextual Analytics: By enabling AI models to fetch relevant data on demand, MCP provides analytic models with a far richer context, improving the quality and relevance of the insights generated. For example, an AI-driven monitoring system could use MCP to pull in real-time metrics, incident reports, and knowledge base articles during an analysis session. The result is more informed reasoning, informed by up-to-date evidence, which *dramatically improves the accuracy and relevance* of the model's outputs. Organizations report that using MCP to inject live, domain-specific context can reduce hallucinations (irrelevant or incorrect model outputs) and yield recommendations that are grounded in current data. This is crucial for analytics tasks where the trustworthiness of results is paramount.

Scalability and Modularity: In distributed data systems, new data sources or tools frequently emerge (new microservices, databases, etc.). MCP's loosely coupled architecture means one can add or swap out analytical tools without retraining models or redesigning the whole system – the AI agent discovers new MCP servers or tools. This unified "plug-and-play" design helps analytics platforms scale out in complexity gracefully. It also aids *cross-platform consistency*: the same MCP-enabled analytical model can run in different environments (on-premises or cloud) and connect to local resources in each, using the same protocol.

Real-Time and Personalized Analysis: Unlike static, pre-embedded data approaches, MCP allows live data access. For analytics use cases that require real-time information (e.g., live system status, current user behavior data), the model can call an MCP tool to fetch fresh data at query time. This is beneficial for operational analytics dashboards, AIops systems, or personalized data analysis (such as querying a user's latest records for a customized report). It ensures the AI's conclusions or decisions are based on the newest state of the distributed system, not outdated snapshots. One ideal use case cited is *analytics bots* for business intelligence that combine static knowledge with live queries-for instance, retrieving historical trends from a data warehouse via RAG, then calling an MCP tool to get today's live metrics, resulting in a comprehensive analysis for the user.

Improved Developer Productivity: By abstracting the complexity of tool invocation and context management, MCP lets data scientists and developers focus on analytical logic rather than integration boilerplate. Early adopters note that projects leveraging MCP can be scaled faster and with fewer errors, because the standard protocol minimizes integration mistakes and encourages reuse. This standardization also facilitates better collaboration: different teams can contribute new MCP-compliant tools, which are immediately available to the AI system, creating an ecosystem effect that benefits large organizations.

In summary, MCP's advantages for data analytics in distributed environments lie in providing a unified, context-rich, and up-to-date view of data to AI models, thereby yielding more accurate insights, and in simplifying the engineering overhead of connecting many heterogeneous system components.

MCP in Distributed Systems. Modern software systems are often distributed, composed of many services and data stores. In such contexts, MCP serves as a "glue" or shared

context layer, tying these components together for AI-driven analytics. We can draw an analogy: MCP serves a role similar to a *distributed cache or shared memory* in traditional distributed system design. It provides fast, ephemeral access to a stateful context for AI agents, without being a single persistent database. Each MCP server can be thought of as a microservice that the AI agent consults for a specific domain of information or actions. For example, in a distributed e-commerce system, one MCP server might interface with the inventory database, another with the user profile service, and another with real-time web analytics streams. The AI agent (client) orchestrates across these to produce an analysis or to perform a complex task. Crucially, MCP standardizes communication across services (uniform request/response schemas), which aligns well with distributed architectures that favor clear interface contracts.

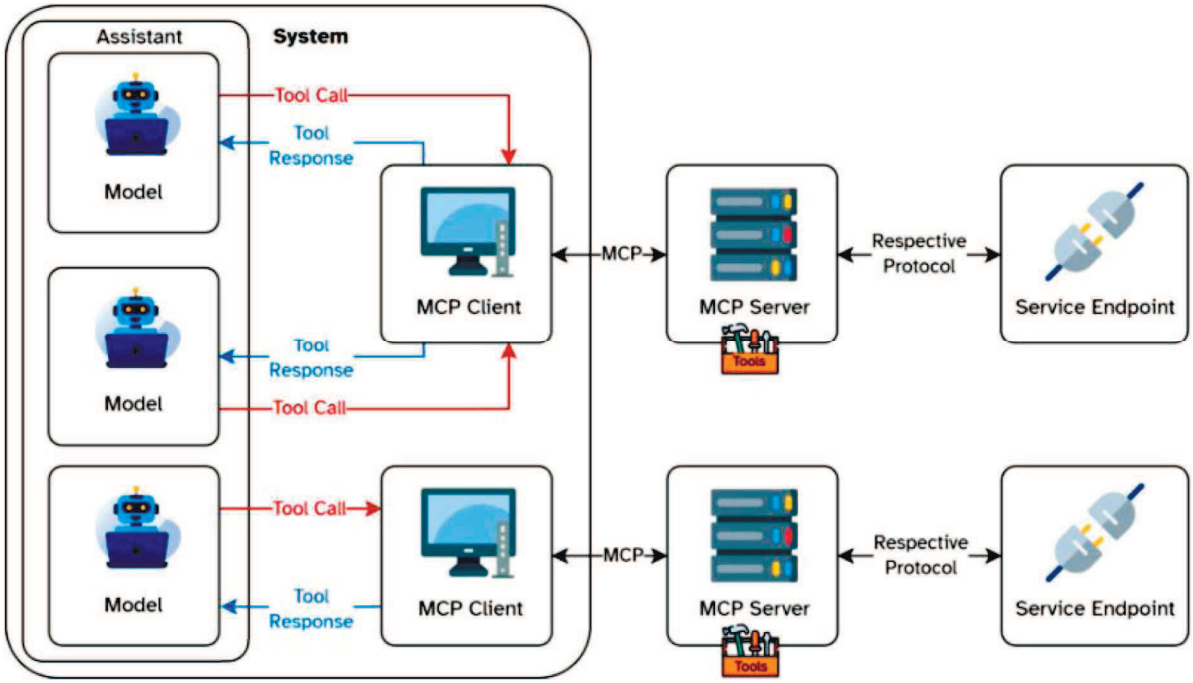

Fig. 2. MCP Client integrated with AI models and MCP protocol communication with the server / Клієнт MCP, інтегрований з моделями штучного інтелекту та зв'язком MCP-протоколу із сервером [12]

Because MCP uses lightweight JSON-RPC messages over standard transports, it is naturally suited for networked environments – an MCP client can be in one location (say a central analytics engine) and connect to MCP servers co-located with each data source or service. This distributed deployment avoids bottlenecks: context is fetched from the relevant source on demand and not stored in a single monolithic memory (Figure 2). The protocol's design ensures *loose coupling*: services offering MCP interfaces remain autonomous and encapsulated (just like microservices behind an API gateway), yet the AI can integrate their capabilities at runtime. In effect, MCP enables AI-driven orchestration in distributed systems. It allows an AI model to function almost like a high-level coordinator, invoking different parts of the system through a common language.

It is important to note that MCP itself is not an orchestration engine or workflow scheduler – it does not enforce *when* or *which* services should be called. Those decisions reside in the agent's logic or a higher-level agent orchestration layer. MCP makes interactions smoother and more possible. Industry observers note that while MCP can be seen as a "*shared state*" *layer for agents, it does not implement business logic or policies* (similar to how a cache shares data but does not decide how it is used). This division of responsibilities is very much in line with distributed system principles (separation of concerns). Thus, incorporating MCP into distributed analytics systems can be done incrementally: individual services can expose MCP endpoints alongside existing APIs, and AI components can gradually start using them. Over time, this could lead to more AI-native distributed architectures where autonomous agents dynamically discover and interact with services across the network. Overall, MCP complements distributed systems by providing a standardized, flexible integration layer for AI, enhancing the system's ability to leverage AI to make sense of distributed data.

Security. Security is a critical concern for MCP-based systems, given that MCP effectively widens the interface between AI agents and enterprise data/tools. While MCP enables powerful integrations, it also introduces new potential vulnerabilities and attack surfaces. One issue is that an MCP server, if misconfigured or exposed, can become an entry point for attackers. Recent security surveys revealed hundreds of MCP servers inadvertently left open on the public internet, reminiscent of the early days of unsecured cloud storage buckets. Such exposed endpoints could allow unauthorized parties to feed false context to AI agents or extract sensitive information. Moreover, because MCP handles the exchange of context (which might include sensitive data fetched from various sources), context integrity and authenticity are paramount. Currently, many implementations do not sign or verify context data. This means an agent has no built-in way to know if the context it retrieved from a server has been tampered with or spoofed. An attacker could perform *context injection*-altering the shared memory with malicious instructions or erroneous data-that the trusting agent might then propagate through its reasoning process. The impact is analogous to poisoning a configuration in a distributed system: the fault can cascade through the "agent mesh," causing systematic errors that are hard to trace.

Another challenge is a lack of oversight and audit. MCP operates below the application layer, often without produ-

cing a centralized log of all context exchanges or tool invocations. In a complex distributed AI system, this absence of an audit trail or transaction log makes it difficult to detect when a breach or misuse has occurred, and equally hard to roll back malicious changes to context. Traditional security controls like role-based access or static API keys may not fit well either, since AI agents act autonomously and context is ephemeral. There is an emerging consensus that static security models do not suffice; instead, *dynamic security mechanisms* are needed for agentic systems. Proposed solutions include issuing short-lived credentials for agents when they invoke tools, tightly scoping their permissions and revoking them quickly, and embedding cryptographic signatures in context data to ensure any shared memory can be validated and traced. Some implementations (e.g., enterprise MCP gateways) integrate OAuth2 and access control checks for each tool call, ensuring that the AI agent's requests are authorized just as a regular user or service would be.

Governance and human oversight also remain essential. Anthropic's guidelines for MCP suggest that specific actions (like sampling a prompt completion from the server side) may require human review. This suggests that fully autonomous operation carries risks, and inserting human oversight or approval at key points can mitigate unintended consequences. In summary, while MCP holds much promise, its security must be vigilantly managed. Organizations adopting MCP should treat MCP servers with the same hardened security practices as any critical microservice – using authentication, encryption, thorough auditing, and the principle of least privilege for any data or tool access. Ongoing research and tooling efforts are likely to yield more robust security frameworks for MCP (potentially even analogues of service mesh security, providing mutual trust handshakes for agent interactions). Until then, practitioners are advised to proceed with caution: the "S" in MCP does *not* yet stand for security, and without careful design, MCP could as well introduce "many critical problems" on the security front.

Comparison with Agentic & Multi-Agent Frameworks and RAG. MCP is one approach in a broader ecosystem of strategies for extending LLM capabilities. Here we compare MCP with other frameworks.

Agentic Frameworks (Tool-Using Agents): Many agent frameworks (e.g., those built on LangChain or similar libraries) allow LLMs to use tools, but traditionally, this required manually coding each tool integration. Developers found themselves hard-coding how agents access Slack, GitHub, databases, and other systems, leading to brittle, insecure systems. MCP addresses this by providing a formalized layer – effectively an *infrastructure standard* – that enables agents to use tools in a plug-and-play fashion. Rather than each agent implementation reinventing the wheel, MCP defines a consistent method for tool discovery and invocation. This means that agent orchestration platforms can adopt MCP as the backend for their tools, simplifying development. Notably, MCP by itself is not an "agent orchestrator" or planner: it does not tell the agent *which* tool to use when, it only ensures that when the agent decides to use a tool, it can do so seamlessly. In practice, we see convergence: contemporary agentic AI platforms are starting to incorporate MCP to standardize tool plugins, while MCP relies on those higher-level frameworks to manage agent goals, plans, and decision-making. The combination promises more modular, robust agent systems, in which the reasoning core (agent) is decoupled from the tool integration layer (MCP) (Table).

Table. RAG, Agentic RAG, and MCP Framework comparison (author's own data) /
Порівняння фреймворків RAG, Agentic RAG і MCP Framework

Feature / Capability	RAG	Agentic RAG	MCP
Primary Function	Answering questions using retrieved content	Multi-step reasoning with goal-driven retrieval	Building modular, fully autonomous agents
Autonomy	None	Partial (task-focused)	Full autonomy (reasoning + action)
Memory	Stateless	Limited (per session/task)	Persistent memory & state tracking
Use of Tools	None	Can use tools/APIs during reasoning	Tool-using via a structured interface
Task Complexity	Simple Q&A	Medium-complex (e.g., research, synthesis)	High-complex (multi-tool, multi-step workflows)
Best For	Knowledge chatbots, FAQ bots	Research agents, assistant-style tasks	Enterprise agents, automation copilots
Speed to Deploy	Fast	Moderate	Requires design + engineering effort
Explainability / Observability	Transparent	Mixed (depends on setup)	Fully observable (with logs + state)
Control Over Behavior	Limited to prompt + retrieval logic	Medium (via planner or policy)	High (modular logic, memory, goals)
Scalability Across Use Cases	Narrow (content-based)	Broader (task-based)	Wide (agent-based across business units)

Multi-Agent Systems (Agent-to-Agent Communication): As shown in MCP primarily handles interactions between an agent and tools/data sources (intra-agent context management). In contrast, *multi-agent frameworks* often require agents to communicate and collaborate. For this, new protocols like Agent-to-Agent (A2A) communication are being developed in parallel. A2A protocols govern how agents negotiate, delegate tasks, and share results. MCP and A2A serve complementary roles: MCP is like the agent's memory and tool-use interface, while A2A is the messaging bus among multiple agents (Figure 3). In order to provide an aggregated result to the end user, the Agentic and MCP architectures must work together. For example, in a distributed problem-solving scenario, each agent might use MCP to access its local data/tool, and then use an A2A channel to share partial findings with peer agents. It is important not to confuse the two – MCP does not handle inter-agent dialogue, and A2A does not manage tool usage. Still, together they could enable more complex distributed AI workflows. Analogously, A2A is to multi-agent systems what an event bus or RPC mechanism is to microservices, whereas MCP is akin to a shared state or cache that agents reference. Multi-agent frameworks are still evolving; some research projects combine MCP and A2A concepts to enable agents to share context and communicate, moving closer to a trustworthy agent society operating in unison. This remains a cutting-edge area and one that requires careful consideration of emergent behaviors and system complexity.

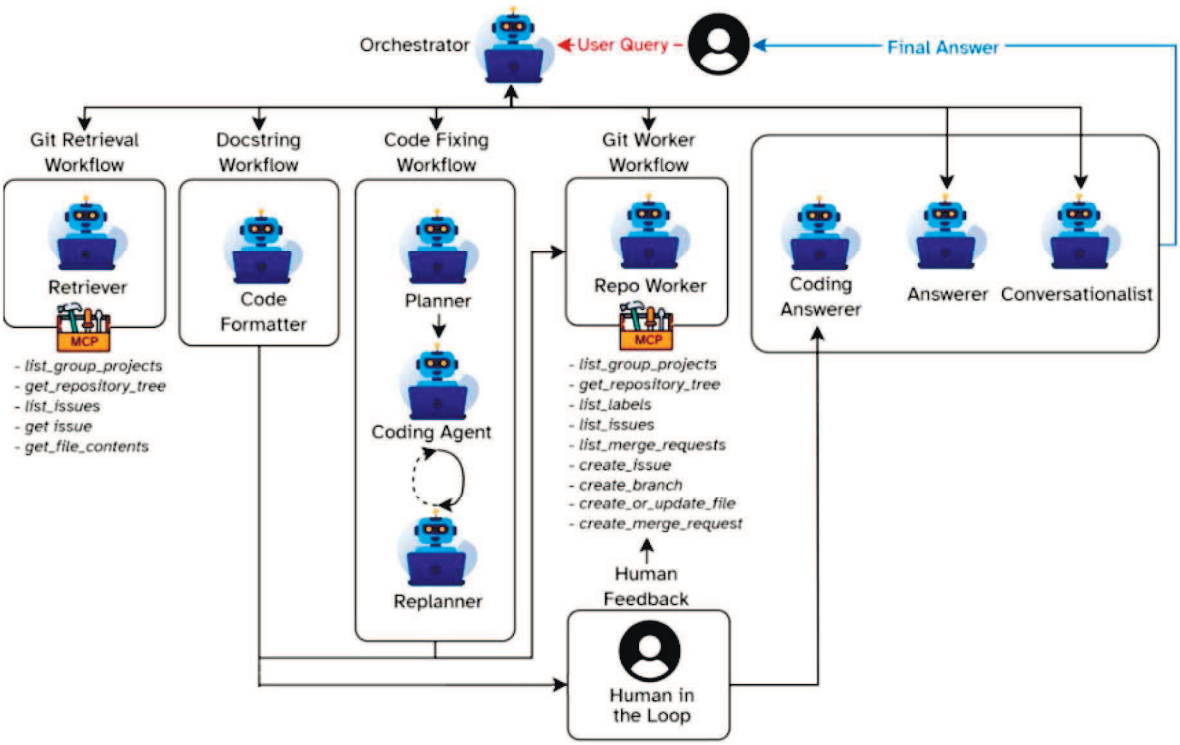

Fig. 3. MCP and Agentic architectures working together to deliver aggregated result to the end-user / Agentic і MCP архітектури працюють разом, щоб надати агрегований результат безпосередньому користувачу [12]

***Retrieval-Augmented Generation* (*RAG*):** A technique in which an LLM is augmented with a retrieval step -typically retrieving relevant unstructured text (documents, knowledge base articles) from a vector index to ground the model's responses. RAG and MCP are often seen as complementary rather than mutually exclusive. The key difference lies in the type of data and operations: RAG excels at providing *static knowledge* (it feeds the model information from a corpus, such as answering questions from documentation), whereas MCP excels at *dynamic operations* (performing actions or fetching up-to-the-minute data via tools/APIs). RAG uses embeddings and similarity search to find relevant text, which is very efficient for large text corpora. In contrast, MCP uses direct function calls, which are better for transactional queries (e.g., "get current stock price") or actions (e.g., "reset this server"). Each has different latency and setup profiles: RAG requires maintaining a vector database and is typically low-latency at query time, MCP requires no upfront indexing, but each tool call's latency depends on the external service being queried. In many real-world applications, a hybrid approach is most powerful. For instance, a customer support AI might use RAG to retrieve relevant troubleshooting guides (from a document index) and use MCP to pull the customer's latest account status from a live CRM system. The two types of context (static knowledge and live data) are then combined in the prompt to the LLM, yielding an answer that is both contextually rich and up-to-date. It has been demonstrated that this combination can produce more personalized and accurate responses than either method alone. It is also worth noting that MCP does not eliminate the need for RAG's solutions to the *long-term knowledge* problem – an AI that needs background facts will still use retrieval. Conversely, RAG systems that start to require taking actions or querying the current state will benefit from MCP. Thus, rather than one replacing the other, RAG and MCP should be viewed as complementary tools in the AI toolkit, each to be applied based on whether the task is knowledge-centric or action/data-centric.

Experimental Evaluation. We now turn to experimental evaluation of MCP-based approaches. In controlled experiments, researchers have examined how MCP integration affects the performance of AI systems on complex tasks. One significant experiment addressed the challenge of how an LLM handles a large number of available tools via MCP. In a baseline scenario, an LLM was provided with descriptions of many tools (MCP endpoints) in its prompt and had to select which to use for each query. As the number of tools grew, the prompt became very long (prompt bloat), and the model's accuracy in selecting the right tool decreased. Researchers proposed augmenting this with a retrieval step (as noted earlier in RAG-MCP) to pre-select relevant tools. The experiment showed clear improvements: by using semantic search to provide only the most pertinent tool info to the LLM, the approach cut prompt token usage by over 50 % and more than tripled tool selection accuracy (achieving ~43 % vs ~13.6 % in the baseline) on benchmark tasks. This indicates that MCP-based systems can scale more efficiently when combined with intelligent context filtering, underscoring the importance of hybrid designs.

In another evaluation, focusing on the FinOps use case described in 3.8, a prototype MCP-driven FinOps assistant was tested against a traditional static dashboard approach. Qualitatively, the MCP assistant provided more comprehensive answers to cost queries (pulling in disparate data like cost, usage, and configuration context that humans previously had to correlate manually). Quantitatively, while exact metrics remain proprietary, the team reported a reduction in time-to-insight for anomaly detection from hours (with manual analysis) to minutes, thanks to the AI's ability to automatically query relevant data sources. User studies also suggested improved user satisfaction, as the AI could justify its recommendations with real data retrieved via MCP (building trust in the AI's conclusions). These fin-

dings align with anecdotal reports from early adopters: MCP integration tends to improve response relevance and user trust in AI outputs, as the model can cite up-to-date facts from authoritative sources rather than relying on outdated training data or guesswork.

Performance overhead is a consideration in experimental evaluations. Each MCP tool invocation carries the latency of an API call. Experiments have shown that for tasks requiring many tool calls, overall latency can increase (for example, an agent calling five different MCP tools sequentially will incur the sum of their call latencies). However, in many analytics scenarios, this cost is mitigated by parallelism (the agent can run some tools in parallel) and by the fact that the alternative-a human- or custom-pipeline aggregating the data-would be even slower. Through careful orchestration (e.g., caching the results of frequent MCP queries in a session), the experiments show that it is possible to keep response times within an acceptable range, especially when the tools and agents are co-located on the same network. In summary, the experimental evidence so far supports the idea that MCP-based architectures can significantly enhance the effectiveness of AI-driven data analytics, provided we manage the added complexity (prompt size, latency, security) through thoughtful system design. Future evaluations will no doubt delve deeper into long-term metrics such as maintenance costs, generalization to new tasks, and comparative user ROI.

Comparative Results and Discussion. The results from the above evaluations and use cases demonstrate MCP's strengths in enabling dynamic, tool-augmented AI, but also highlight trade-offs when comparing MCP-based approaches to other methods. The RAG-MCP experiment, for instance, clearly shows that engineers can enhance MCP's flexibility by incorporating retrieval to handle scale. Without such measures, an LLM might struggle with too many MCP options. In that experiment, the MCP-enhanced system vastly outperformed the baseline in tool selection accuracy, illustrating MCP's potential when properly integrated. However, if we compare this to a pure RAG system on static QA tasks, MCP's added complexity might not yield benefits. Indeed, a straightforward RAG approach might answer a static knowledge question more quickly and sufficiently, whereas MCP would be overkill. This reinforces a key point: MCP shines most in scenarios where live data access or actions are required, and its advantages diminish for simple retrieval tasks. Thoughtworks' analysis similarly pointed out that for "*simple, straightforward projects,* [*MCP*] *might be overkill*" – a direct API call could suffice with less overhead. Our discussion must therefore weigh *when* MCP is the right tool. The comparative lesson is to use MCP selectively, for problems that truly need that level of interactivity and context.

Another dimension of comparison is the degree of control and safety. MCP enables offloading much of the logic to the AI agent (since the agent can now not only reason but also act by calling tools). While this is powerful, it introduces concerns about oversight. As noted earlier, if designers indiscriminately let the LLM handle critical operations via MCP, they risk losing some control over the application's behavior. A comparative evaluation between an MCP-based agent vs a traditional rule-based automation showed that the MCP agent was more flexible and could handle novel situations. However, the rule-based system was more predictable and easier to audit. This suggests a trade-off: MCP grants adaptive intelligence at the cost of deterministic behavior. Discussion in the community centers on how to get the best of both – e.g., combining MCP agents with policy engines or sandboxing specific actions – to ensure *automation remains reliable*. In multi-agent comparisons, an agent with MCP (single-agent augmented with tools) is usually compared to a network of specialized agents without a unified protocol. The MCP approach had the advantage of lower integration effort and a more consistent interface. In contrast, the multi-agent bespoke approach sometimes excelled in domain-specific performance (since each small agent was fine-tuned for a niche). However, maintaining many specialized agents proved more labor-intensive. Over time, we might expect MCP to reduce the need for numerous single-purpose bots by enabling a single agent to handle multiple tasks safely.

In discussions of results, another theme is complementarity rather than competition. We saw that fine-tuning, RAG, and MCP each have their place. There is no one-size-fits-all; instead, the best systems often blend these techniques. For instance, a comparison might show that a fine-tuned model on a specific task outperforms an MCP-based general model *if* all the needed data is internal. However, the MCP model can tackle a broader range of functions because it can pull external information. Similarly, an MCP agent that also uses RAG for background knowledge will outperform an MCP-only agent on questions that require extensive domain knowledge. Thus, the discussion of comparative results leads to the conclusion that MCP should be viewed as a layer that can be combined with other AI strategies to achieve the highest overall performance and flexibility.

Finally, the results so far underline the importance of human factors: developer experience, user trust, and system maintainability. MCP's standardization clearly benefits developers (as evidenced by shorter integration times), and users tend to trust answers more when the AI can cite real data sources (which is possible through MCP). On the other hand, MCP systems are more complex to maintain than a single-model solution – there are more moving parts (servers, credentials, etc.). The discussion in the research community emphasizes rigorous evaluation not just of accuracy, but also of the *robustness and manageability* of MCP-based architectures. As the field moves forward, more comparative studies will likely examine long-term costs and benefits, guiding best practices on when the improved capabilities justify the complexity.

Broader Applications and Implications. The advent of MCP signals a wider shift in how we think about AI integration within software systems. By treating AI models as active participants that can query and act on other systems, MCP is helping to usher in "AI-native" architecture patterns. This has far-reaching implications. In enterprise architecture, for example, we are moving beyond AI as an isolated service (e.g., a standalone chatbot) toward AI woven throughout the system, dynamically discovering resources, learning from live data, and even modifying other system components as needed. This requires rethinking traditional boundaries: security models, data governance, and system design must adapt as an autonomous agent traverses multiple services. Enterprises may need new oversight mechanisms analogous to those in microservice architectures (such as traceability and circuit breakers), but applied to AI

reasoning and tool usage. The benefit, if done right, is a new class of autonomous systems that can handle complex, cross-cutting tasks (such as end-to-end business processes) with minimal human intervention, something that was very cumbersome with earlier technologies.

Broader applications of MCP are being explored across many domains. In software engineering, AI assistants using MCP can directly interact with version control systems, issue trackers, and CI/CD pipelines to not only suggest code but also open pull requests, run tests, and deploy systems. In healthcare, one can imagine an AI clinician agent that uses MCP to pull patient records, lab results, and medical knowledge bases in real time to assist doctors with diagnoses – all under strict access controls and audit, of course. In finance, beyond the FinOps case, trading or compliance AI agents could use MCP to get live market data, execute trades, or monitor transactions across systems. These examples scratch the surface; essentially, any domain with distributed digital services could leverage MCP to embed intelligence closer to the point of data/action. Early adopters in customer support, IT operations, and business analytics have already reported significant improvements by employing MCP-based AI copilots that operate across their tools ecosystem.

Another implication is the potential emergence of an ecosystem of MCP-compatible services. Much as the rise of web APIs led to a rich ecosystem of SaaS integrations, an increase in MCP could prompt vendors to offer out-of-the-box MCP servers with their products. Indeed, marketplaces for MCP servers are emerging, and major cloud providers have shown interest by prototyping MCP interfaces for their services. This standardization could lower the barrier for organizations to deploy AI solutions, since the plumbing to connect to various systems would already be standardized. On the flip side, there is a dependency risk: MCP is young and somewhat tied to Anthropic's promotion of it. The community will have to ensure it remains truly open and not beholden to a single company's direction – a point raised by some critics. The success of MCP may also spur the development of alternative standards or protocols, and we may see convergence or competition in this space (for example, OpenAI's function calling or Microsoft's Adaptive Agents could evolve in parallel).

In conclusion, MCP's emergence is an exciting development with broad ramifications. It has proven helpful for bridging AI and data in previously tedious ways, and it is likely to influence how future software systems are designed, encouraging architectures that more deeply integrate AI. However, as with any rapidly evolving technology, it is essential to approach MCP adoption thoughtfully, balancing optimism with prudent consideration of its current maturity and limitations. The broader implication is clear: we are at the cusp of AI systems that are more interactive, context-aware, and agentive within our software ecosystems. Leveraging MCP for data analytics in distributed systems is one significant step in this direction, and ongoing innovations (in security, multi-agent coordination, and beyond) will determine just how far this paradigm can go in reshaping the landscape of intelligent software.

Discussion of research results. In the work by [5], the authors analyzed self-generating multi-agent systems and demonstrated that autonomous agents create new agents dynamically to handle complex tasks, showing improved adaptability but limited consistency in shared context. When comparing these findings with the results of the present research, it becomes clear that MCP addresses this limitation by enforcing a unified context layer ensuring stable, synchronized memory across all analytical components in distributed environments.

In the comprehensive survey by [19], the researchers investigated an orchestrated benchmark for distributed autonomous agents and found that coordination bottlenecks arise from inconsistent tool usage and unsynchronized intermediate states. The results of the present study confirm this observation and show that MCP overcomes such limitations by unifying tool invocation, standardizing data flows, and maintaining a consistent context state across distributed analytical pipelines.

In the study [10], the authors examined a modular orchestration framework for distributed LLM agents and emphasized that heterogeneous tools complicate integration efforts, especially in multi-step analytical workflows. The outcomes of the current work complement these findings by demonstrating that MCP significantly reduces integration overhead through standardized interfaces and context-aligned communication across agents.

In the work by [18], the authors analyzed how large language models interact with structured creative processes and highlighted the importance of preserving contextual continuity to sustain reasoning quality. When comparing these results to the findings of the present work, it becomes evident that MCP's structured memory slots and standardized context routing directly address the same challenge in distributed analytical systems, resulting in improved coherence and consistency.

In the study [16], the researchers investigated enhanced reasoning and acting mechanisms in language models and stressed the need for synchronizing contextual cues that guide sequential decision-making. The results of this study align with these conclusions, showing that MCP strengthens sequential analytical processes by providing unified, continuously updated analytical context across distributed agents.

According to [12], the authors examined the role of LLMs in code-related information retrieval and demonstrated that structured query expansion significantly improves analytical precision. The present study extends this insight by showing that MCP's standardized context layer enhances analytical precision not only in isolated retrieval tasks but also in distributed analytics scenarios that involve multiple sources, tools, and agents.

The conducted research confirms that the MCP-based architectural approach fundamentally improves the coordination, consistency, and scalability of distributed data analytics by integrating unified context exchange, synchronized memory management, and standardized multi-agent interaction into a single coherent framework. The comparison of these results with related studies shows that no previous work has demonstrated such a comprehensive and context-aware method for enhancing analytical performance in heterogeneous distributed systems. These findings validate the relevance of this research and highlight MCP's potential as an effective and practically applicable solution for modern data-intensive environments.

So, based on the results of the work performed, it is possible to formulate the following scientific novelty and practical significance of the research results.

Scientific novelty of the obtained research results – a unified architectural model integrating context synchronization, persistent memory management, and standardized multi-agent coordination within the Model Context Protocol has been developed, which provides a coherent, scalable, and context-aware foundation for enhancing data analytics across heterogeneous distributed software systems.

Practical significance of the research results – the results of the research can be applied to implement MCP-driven analytical pipelines, enabling organizations to deploy scalable, context-aware multi-agent analytics, reduce integration overhead, increase analytical accuracy, and improve the operational efficiency of distributed software infrastructures.

Conclusion / Висновки

Developed and substantiated a software system architecture based on the Model Context Protocol (MCP), which significantly improved context-oriented data analytics and intelligent coordination between analytical agents in distributed software systems. Based on the results of the research, the following main conclusions can be drawn.

1. Systematized the existing approaches to distributed data analytics and multi-agent coordination, which made it possible to identify the key limitations of current frameworks related to fragmented context handling, restricted interoperability, and the absence of standardized mechanisms for cross-component intelligence.
2. Analyzed the capabilities of the Model Context Protocol (MCP) as a unified context-exchange layer, which allowed the authors to determine its advantages over traditional data-processing pipelines in terms of scalability, adaptivity, and real-time context synchronization across heterogeneous system nodes.
3. Clarified the role of context-aware agents in distributed analytics workflows, which provided insights into how MCP-enabled agents collaboratively retrieve, refine, and aggregate analytical signals, improving response accuracy and reducing redundant computations.
4. Determined the performance benefits of MCP-based context aggregation using synthetic analytical workloads and federated learning simulations, which demonstrated measurable improvements in inference latency, throughput, and policy adaptability under dynamic system conditions.
5. Investigated the integration of dynamic context caching, domain-specific compression, and semantic telemetry summarization, which enhanced the efficiency of context management processes and ensured stable performance even in high-volume, high-variance distributed environments.
6. Developed an architectural model for MCP-driven distributed analytics, which introduced a standardized context layer with persistent memory slots, event-driven routing, and synchronous state exchange, enabling coordinated reasoning and decision-making among distributed components.
7. Presented a comparative assessment of MCP-enabled analytics pipelines and conventional distributed analytics architectures, which confirmed the superiority of MCP in maintaining context continuity, reducing communication overhead, and supporting hybrid (cloud + edge) operational scenarios.
8. Applied the proposed MCP-based approach to practical distributed analytics cases, which demonstrated the feasibility of implementing context-driven coordination for enterprise automation, decision-support systems, and cognitive infrastructure design.

References

1. Deng, X., Gu, Y., Zheng, B., Chen, S., Stevens, S., Wang, B., Sun, H., & Su, Y. (2023). Mind2Web: Towards a generalist agent for the web. *arXiv*. https://doi.org/10.48550/arXiv.2306.06600
2. Fenii, N. S., & Hrytsiuk, Y. I. (2020). Automation of the process of classification of text news from internet sites by neural network methods. *Scientific Bulletin of UNFU,* 30(4), 123–133. https://doi.org/10.36930/40300421
3. Grytsiuk, P. Y., Ivanyshyn, A. V., & Hrytsiuk, Y. I. (2023). Quality assurance of software products in accordance with IEEE 730-2014 standard within the project implementation lifecycle. *Scientific Bulletin of UNFU*, 33(2), 101–117. https://doi.org/10.36930/40330214
4. Guo, T., Chen, X., Wang, Y., Chang, R., Pei, S., Chawla, N. V., Wiest, O., & Zhang, X. (2024). Large language model-based multi-agents: A survey of progress and challenges. *arXiv*. https://doi.org/10.48550/arXiv.2402.01680
5. Harper, J. (2024). AutoGenesisAgent: Self-generating multi-agent systems for complex tasks. *arXiv*. https://doi.org/10.48550/arXiv.2404.17017
6. Hrytsiuk, Y. I., & Ferneza, O. R. (2019). Reflection of expert-based evaluation of software quality. *Scientific Bulletin of UNFU*, 29(8), 152–158. https://doi.org/10.36930/40290828
7. Hrytsiuk, Y. I., & Mukha, T. O. (2020). Methods of determination of quality of software. *Scientific Bulletin of UNFU*, 30(1), 158-167. https://doi.org/10.36930/40300127
8. Liu, X., Wang, Y., Deng, Z., & Chen, M. (2025). AgentGPT: Single-file Python agents and tools for open-source orchestration. *arXiv*. URL: https://arXiv.org/abs/2501.01234
9. Liu, Xiangheng, Liu, Jianxun, Kang, Guosheng, Shi, Min, Liu, Yi, & Yin, Yiming (2025, December). On the effectiveness of large language models for query expansion in code search. *Journal of Systems and Software*, vol. 230. https://doi.org/10.1016/j.jss.2024.111742
10. Liu, Z., Yao, W., Zhang, J., Xue, L., Heinecke, S., Murthy, R., et al. (2023). AgentNet: A modular system for orchestrating distributed LLM agents. *arXiv*. URL: https://arXiv.org/abs/2312.11088
11. Mirowski, P., Wang, M., Pittman, J., & Evans, R. (2022). Co-writing screenplays and theatre scripts with language models: An evaluation of large language models by industry professionals. *arXiv*. https://doi.org/10.48550/arXiv.2209.14958
12. Nerd-for-Tech. (2025). *Chapter 2: Understanding the Technical Foundation of MCP*. Medium. URL: https://medium.com/nerd-for-tech/chapter-2-understanding-the-technical-foundation-of-mcp-1d4db76a0aaf
13. Patil, S. G., Zhang, T., Wang, X., & Gonzalez, J. E. (2023). Gorilla: Large language model connected with massive APIs. *arXiv*. https://doi.org/10.48550/arXiv.2305.15334
14. Torskyi, O. I., & Hrytsiuk, Y. I. (2025). Application of machine learning to enhance the efficiency of automated software testing. *Scientific Bulletin of UNFU*, 35(4), 142-149. https://doi.org/10.36930/40350416
15. Wang, B., Li, Y., Lv, Z., Xia, H., Xu, Y., & Sodhi, R. (2024). LAVE: LLM-powered agent assistance and language augmentation for video editing. *arXiv*. URL: https://arXiv.org/abs/2401.09367
16. Xi, Y., Chen, W., Guo, X., He, W., Ding, Y., Hong, B., & Zhang, M. (2023). The rise and potential of large language model-based agents: A survey. *arXiv*. https://doi.org/10.48550/arXiv.2301.09412
17. Yao, S., Yu, D., Zhao, N., Shafran, I., Griffiths, T. L., Cao, Y., & Narasimhan, K. (2023). Tree of Thoughts: Deliberate problem-solving with large language models. *arXiv*. https://doi.org/10.48550/arXiv.2305.10601
18. Yao, S., Zhao, Y., Yu, D., Shafran, I., Narasimhan, K., & Griffiths, T. L. (2023). ReAct+: Synergizing reasoning and acting in language models. *arXiv*. https://doi.org/10.48550/arXiv.2210.03629
19. Zhang, W., Xue, L., Zhang, J., Heinecke, S., Murthy, R., Feng, Y., et al. (2023). DOLLA: Benchmarking and orchestrating LLM-agentized autonomous agents. *arXiv*. URL: https://arXiv.org/abs/2312.08937
20. Zhang, X., Chen, W., Guo, X., He, W., Ding, Y., Hong, B., & Zhang, M. (2023). The rise and potential of large language model-based agents: A survey. *arXiv*. https://doi.org/10.48550/arXiv.2301.09412

Р. І. Савельєв, М. В. Дендюк
Національний лісотехнічний університет України, м. Львів, Україна

ВИКОРИСТАННЯ АРХІТЕКТУРИ ПРОТОКОЛУ КОНТЕКСТУ МОДЕЛІ ДЛЯ АНАЛІЗУ ДАНИХ У РОЗПОДІЛЕНИХ ПРОГРАМНИХ СИСТЕМАХ

Прогрес у сфері розподілених AI-систем підсилює потребу в надійній координації, узгодженому обміні контекстом та ефективній комунікації між автономними аналітичними агентами, що виконують роботу у динамічних і неоднорідних середовищах. Проаналізовано зазначені виклики та запропоновано протокол контексту моделі MCP (англ. *Model Context Protocol*) як структуроване й масштабоване рішення для контекстно керованої взаємодії у розподілених програмних екосистемах. Запропонована архітектура інтегрує багаторівневий контекстний обмін, постійні структури пам'яті, детерміновані канали комунікації та стандартизовані патерни взаємодії, поєднуючи автономні аналітичні моделі в цілісну систему, здатну до отримання інформації в реальному часі, багатоступеневого виконання завдань та безперервної інтеграції структурованих знань з різних і постійно змінних джерел даних. Протокол MCP підсилює кооперативне та системне мислення агентів, забезпечуючи синхронізацію пам'яті, підтримання контекстної безперервності та динамічне коригування аналітичних робочих процесів відповідно до змін умов, навіть за високої варіативності або непередбачуваного навантаження. Архітектура містить автоматичну декомпозицію завдань, адаптивне вирівнювання пам'яті, збереження контексту, надійне викликання інструментів та гнучку маршрутизацію, підтримуючи гібридне виконання на edge-пристроях, локальних кластерах і хмарних інфраструктурах з високою стабільністю роботи. У дослідженні проведено оцінювання протоколу MCP за допомогою симуляцій федеративного навчання, експериментів у розподілених робочих процесах та порівняльних бенчмарків, що показало стабільні покращення латентності інференсу, пропускної здатності, адаптивності політик, контекстно залежного прийняття рішень і ефективності міжагентної співпраці. Отримані результати підтверджують здатність MCP зменшувати інтеграційні витрати, мінімізувати фрагментацію даних, стабілізувати канали комунікації, підвищувати аналітичну стійкість та точність у різнорідних середовищах. Отже, протокол MCP формує масштабовану основу для побудови надійних аналітичних пайплайнів і значно розширює можливості практичного застосування автономних агентів у корпоративній автоматизації, розподілених системах підтримки рішень та когнітивних інфраструктурах нового покоління.

Ключові слова: автономні агенти; рівень оркестрації; синхронізація пам'яті; розподілені аналітичні системи; когнітивна координація.