Simplifying Distributed Application Troubleshooting through Observability and Contextualization of LLMs

Igor Martins Silva[1], **Juliano Araújo Wickboldt**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

Resumo. *A resolução de problemas de aplicações complexas, distribuídas, baseadas em microsserviços é uma tarefa complicada de realizar. As Large Language Models (LLMs) estão sendo amplamente utilizadas para auxiliar no processo de desenvolvimento, mas muito menos no processo de resolução de problemas nesse contexto. Soluções de observabilidade (rodando sobre Kubernetes, por exemplo) fornecem informações sobre métricas, logs e traces de execução de aplicações, porém em um volume muito grande para que possam ser facilmente interpretadas e correlacionadas pelos desenvolvedores. Existem algumas soluções baseadas em Model Context Protocol (MCP) que fornecem informações a uma LLM sobre aplicações rodando em um cluster, mas ainda de forma muito limitada a interpretação de logs e configurações, que não são específicas para esse processo de debugging. Para essa atividade torna-se crucial, integrar de forma mais ampla as informações de observabilidade (logs, métricas e traces) em um servidor MCP especificamente pensado para essa atividade. Este trabalho visa implementar uma solução para este problema no Ollama e testar diferentes LLMs e formas de integrar esses dados de séries temporais e traces de forma mais adequada para a LLM.*

Abstract. *The resolution of problems in complex, distributed, microservices-based applications is a complicated task to carry out. Large Language Modals (LLMs) are being widely used to assist in the development process, but much less in the problem-solving process in this context. Observability solutions (running on Kubernetes, for example) provide information about application metrics, logs, and execution traces, but in a volume that is too large to be easily interpreted and correlated by developers. There are some solutions based on Model Context Protocol (MCP) that provide information to a LLM about applications running in a cluster, but still in a very limited way for interpreting logs and configurations, which are not specific to this debugging process. For this activity, it becomes crucial to integrate observability information more broadly (logs, metrics, and traces) into an MCP server specifically designed for this activity. This work aims to implement this on Ollama and test different LLMs and ways to integrate these time series data and traces more appropriately for the LLM.*

1. Introduction

Complex applications are typically organized using a microservices-based architecture, constituting distributed systems that produce a series of logs, metrics, and execution traces. Observing this data is extremely important for assessing the health of the system and resolving any problems that these systems may present. To support this

process, orchestration and observability solutions such as Kubernetes (for orchestration) and Prometheus, Jaeger, and Grafana (for monitoring) have been widely used. However, these tools provide large-scale data that is often difficult to correlate and interpret manually [Sukhija and Bautista 2019].

Large Language Models (LLMs) are increasingly present in our daily lives, assisting in various tasks such as code copilots, testing support, or information research [Jiang et al. 2024]. One area that is still largely unexplored is the use of LLMs in the process of automated debugging and troubleshooting of distributed systems [De Jesus et al. 2025]. For an LLM to effectively support problem solving, it needs to be fed with rich context, integrating different sources of observability data.

In this context, the Model Context Protocol (MCP) [Anthropic 2024] emerges, an approach that allows standardizing LLMs' access to tools and data external to it. It works like a "USB port for the LLM" [Singh et al. 2025], enabling the addition of any type of program, data source, and business services (such as APIs, databases, GitHub, Google Drive) that improve contextualization and enable the LLM to produce more sophisticated answers. One of the problems is that current MCPs offer limited or paid access to auxiliary tools such as Kubernetes, in addition to lacking rich and contextualized integration between time series, traces, and logs.

With increasingly larger and interconnected systems, generating massive amounts of observability data, analyzing this data for debugging and troubleshooting can be an extremely complex and time-consuming task [Yan et al. 2024]. To solve this problem, this work proposes the creation of MCP tools integrated with LLM and Kubernetes to assist in the observability of distributed applications running on a cluster with a focus on debugging and troubleshooting. It is expected to contribute to reducing human effort in these tasks, bringing observability and generative artificial intelligence closer together in a realistic scenario, potentially impacting DevOps and the reliability of distributed systems.

2. Fundamentals

In this section, we will subdivide into microservices and distributed applications in Subsection 2.1; in Subsection 2.2, explanations about observability and troubleshooting; in Subsection 2.3, LLMs and time series analysis are explained; and finally, Subsection 2.4 contains an explanation of what MCP is and how it works.

2.1. Microservices and Distributed Applications

Microservices architecture emerged as a natural evolution from traditional monolithic systems, with the goal of increasing scalability, resilience, and flexibility in the development of complex applications. In this approach, the system is composed of several independent services, each responsible for a specific functionality, which communicate with each other through well-defined interfaces. This separation of responsibilities allows each service to be developed, deployed, and scaled in isolation. With their decoupling between components, microservices have high scalability, benefiting from asynchronous or synchronous communication between services, making them increasingly common in large software projects [Blinowski et al. 2022].

Despite all the benefits, microservice-based systems introduce significant new challenges to management, monitoring, and debugging. Fragmentation of the application into dozens or hundreds of interconnected services increases operational complexity, resulting in logs and metrics scattered across multiple sources. Making the task of gathering all the information needed to solve the problem overly difficult and laborious [John 2025].

Additionally, the use of microservices implies an increase in the volume and diversity of operational data. Each request can generate multiple log records, performance metrics, and distributed traces, composing a massive amount of data that needs to be analyzed in near real time to ensure system reliability. In this context, fault diagnosis relies heavily on observability solutions.

Identifying the source of a problem in a distributed environment requires not only centralized access to execution data, but also the ability to correlate heterogeneous information from different services and application layers. This is where fundamental challenges arise that this work seeks to address: how to integrate observability information from a distributed system in an efficient and contextualized manner, allowing language models (LLMs) to assist in the process of diagnosing and resolving failures [Thamma 2025].

2.2. Observability as a Method for Debugging and Troubleshooting Applications

The debugging process involves finding and reducing the number of bugs, or defects, in a program. A defect is usually detected because the program generates unexpected behavior. To locate the cause of a defect, it is essential to explain why this behavior is generated. The troubleshooting process, on the other hand, refers to a series of steps followed to detect, identify, analyze, correct, and prevent problems in computer systems. It involves becoming aware of a problem, characterizing it, determining its underlying causes, restoring or replacing the affected function, and taking measures to prevent future problems [Hindriks 2012]. In order to carry out these processes, it is necessary to use observability.

Observability is "the ability to measure the internal state of a system solely through its external outputs" [Usman et al. 2022]. It allows you to investigate and diagnose problems, even those that were not expected, in distributed and dynamic architectures. The external outputs are telemetry data of three main types: metrics, logs, and traces. Metrics are quantifiable values obtained through libraries or the underlying infrastructure, making it easier to identify trends, bottlenecks, and anomalous behavior. Logs provide detailed textual records of events that occur in each service, such as errors, warnings, and state changes. They are essential for in-depth analysis and post-incident diagnostics, but their high volume and unstructured format make manual interpretation complex. Finally, traces are identifiers responsible for capturing requests within a system that comprises component interactions.

Even with sophisticated tools such as Prometheus, Grafana, Elastic Stack, Datadog, OpenTelemetry, and Jaeger, which are responsible for collecting, storing, and visualizing this information. Manual observability analysis remains a complex task, especially when dealing with massively distributed environments [Mahida 2023]. In applications composed of dozens or hundreds of microservices, each request can generate a cascade of events and logs, resulting in millions of data points per minute. Interpreting and correlating this

information efficiently requires in-depth knowledge of the system architecture, business logic, and interdependencies between components [Usman et al. 2022]. In addition, errors often arise from interactions between services, making identifying the source of the problem a time-consuming process that is prone to human error.

Another important challenge lies in the fragmentation of information. Logs, metrics, and traces are collected and stored by different tools and often analyzed in isolation, making it difficult to gain an integrated understanding of system behavior. Although there is a growing movement toward standardization and unification of telemetry with initiatives such as OpenTelemetry, there are still significant barriers to semantic correlation of data, that is, understanding not only what happened, but why it happened [Boten 2022]. That is why observability is the ideal concept to assist not only humans, but mainly LLMs in this arduous task. They need this compilation of information to be able to give a more assertive response.

2.3. LLM and Time Series

LLMs are artificial intelligence (AI) models based on deep neural networks, trained on large volumes of text, which can understand natural language and reason about textual context [Kumar 2024]. With these qualities, they are increasingly being used in various aspects, including in the debugging and troubleshooting process to explain errors and attempt to solve problems related to the error.

However, the effectiveness of an LLM in diagnosing problems depends directly on the level of contextualization of the information it receives and how to interpret it. Without sufficient data such as representative logs, complete system descriptions, or event history, the model can generate inaccurate responses, make incorrect assumptions, and converge on the wrong solution [Ning et al. 2025]. Thus, debugging and troubleshooting tasks with LLMs often require multiple prompts, refinements, and successive attempts until the model is able to correctly understand the scenario being analyzed.

In addition, there is the problem of interpreting time series data [Yu et al. 2023]. These are sequences of numerical or categorical data collected and recorded over time, at regular or irregular intervals. They allow us to analyze the dynamic behavior of a system, identify trends, locate anomalies, and predict future values based on historical data. However, LLM are not very good at understanding these numerical data, making it necessary to map sections of time series to semantic tokens that describe typical behaviors (e.g., “sudden increase”, “downward trend”, “anomalous peak”), creating a bridge between the numerical and symbolic domains, allowing LLM to reason about temporal data the same way it reasons about natural language [Ye et al. 2025].

This challenge becomes even more evident in distributed and microservice-based systems, where errors can involve numerous components and communication flows [Söylemez et al. 2022]. For LLMs to truly contribute to an efficient debugging and troubleshooting process, it is necessary to ensure that they have access to structured and correlated operational data, preferably in near real time. This reduces the human effort required to manually collect and organize telemetry information, while enhancing the model’s ability to generate more accurate and automated analyzes.

Thus, although LLMs have great potential to support fault diagnosis in complex environments, there is still a fundamental limitation related to providing adequate context and analyzing time series. This gap has driven the development of new means of integration between LLMs and external systems, enabling them to automatically access logs, metrics, traces, and other relevant artifacts.

2.4. MCP

As brilliant as LLMs are, they still fail to contextualize themselves with just a few words in the prompt, always requiring much more information to respond more assertively. So, with the requirement to make LLM responses more sophisticated, improvements began to be considered so that they would have access to all the information necessary to provide the best possible response. The first implementations were simple HTTP-based request-response schemes, which were sufficient for simple tasks, but the use of Application Programming Interfaces (APIs) presented significant challenges [Singh et al. 2025], such as integration complexity, scalability issues, interoperability barriers, and security risks.

In essence, the MCP acts as an intermediate layer between the model and external systems, allowing the model to discover, query, and manipulate tools transparently. Through a standardized interface, the MCP server can expose data or functionalities from various systems, such as databases, APIs, enterprise applications, observability services, or even operating systems. Thus, the model no longer depends exclusively on the textual content of its prompts and starts using updated and contextually relevant information from reliable external sources.

Thus, MCP was created to solve this problem, based on protocols such as the Language Server Protocol (LSP), which standardized communication between Integrated Development Environments (IDEs) and language servers. It has an architecture that separates *host*, *client* and *servers*, making it possible to increase their capabilities individually. Not only did it manage to fill the gap that was missing in LLMs, establishing a unified structure that significantly simplifies interactions between AI applications and external resources, but it also provides a solid foundation for systems with AI agents capable of autonomous perception, reasoning, decision-making, and interaction with the real world. All of this makes MCP a critical component in the future development of AI systems.

2.4.1. MCP Architecture

The Protocol works bidirectionally, where the *Host* (usually an LLM or IDE) makes requests, the *client* mediates these interactions, and the *server* provides the data or performs the actions.

- **Server:** Is responsible for providing services and contextual data to the LLM, implements business logic, responding to structured requests from the client. There may be several MCP servers running simultaneously, both locally and remotely, each representing a different context domain.
- **Host:** It is the central orchestration component of the architecture. It is responsible for registering and discovering available servers, exposing their capabilities to the

client. In many scenarios, the host is the component that runs alongside the LLM (for example, within an IDE, or a tool such as Ollama). It also acts as a security and isolation layer, controlling permissions and validating operations.

- **Client:** The client represents the individual components within the host sending requests, requesting the use of a specific tool or reading data from the MCP server, maintaining a one-to-one connection with it. The client does not need to know the implementation details of the servers, only the capabilities exposed via MCP. It also receives the processed responses, which it in turn passes to the host, which transfers them to the LLMs.

2.4.2. MCP local vs. MCP remote

In local mode, the MCP server runs in the same environment as the host or client, enabling direct, low-latency communication. This configuration facilitates control over data and execution security, making it ideal for development, local debugging, and experimentation scenarios where privacy and response speed are priorities.

On the other hand, remote MCP consists of an independent server, accessed through network interfaces (such as HTTP or WebSocket), allowing LLM integration with external data sources and distributed services. This model is more suitable for large-scale applications, where multiple clients need to simultaneously access the same set of tools and contexts. However, its adoption poses additional challenges in terms of authentication, latency, and security, requiring robust authorization and encryption mechanisms.

3. Related Works

In this section, we will subdivide the topics: in Subsection 3.1, articles that address what MCP is; in Subsection 3.2, security aspects; in Subsection 3.3, benchmarks and experimental evaluations; in Subsection 3.4, LLMs and time series; and in Subsection 3.5, articles that address the issue of observability, debugging, and troubleshooting.

3.1. General Context - MCP as a emerging solution

The *Model Context Protocol* (MCP) is an innovative protocol for the era of artificial intelligence, as it proposes to provide a standardized interface for connecting LLMs to external tools, databases, and services in real time. Unlike previous mechanisms such as function calling, MCP seeks to act as a "universal port" or even a "USB for LLMs" [Singh et al. 2025], allowing language models to expand their context through dynamic integration with distributed data and systems.

Ray's survey published in 2025 presents the protocol from the perspective of communication systems, describing its layered architecture, life cycle, and applications in domains such as healthcare, cybersecurity, and cloud automation, as well as discussing issues such as latency, token overhead, and privilege escalation risks [Ray 2025]. Complementarily, Singh *et al.*. characterize MCP as a fundamental approach to standardizing integration between LLMs and external tools, highlighting its benefits and limitations [Singh et al. 2025]. [Krishnan 2025] delves deeper into the role of MCP in multi-agent system architectures, exploring how the protocol can facilitate cooperation between intelligent agents and extend context retention in distributed interactions. These works

provide a solid theoretical foundation but remain at a conceptual level, without evaluating the use of MCP in practical distributed system debugging scenarios.

3.2. Safety and Maintainability

Another important area of literature involves the analysis of security and reliability in the MCP ecosystem. [Hou et al. 2025] discuss security threats, privacy risks, and challenges at each stage of an MCP server's life cycle (creation, operation, and updating). Along the same lines, [Hasan et al. 2025] evaluate security and maintenance aspects, drawing attention to the need for robust practices to protect against attacks and strategies to ensure long-term reliability. These studies highlight legitimate concerns about large-scale adoption, but do not explore how such risks directly impact the integration of MCP with observability systems.

3.3. Benchmarks and Experimental Evaluations

With the popularization of MCP, studies focused on its practical evaluation have also emerged. [Luo et al. 2025] conduct a comparison between MCP servers and function calls, showing that, in many cases, the use of MCP does not result in significant performance gains. However, the authors point out that accuracy can be improved when the parameters that the LLM must build to interact with the server are optimized. On another front, [Song et al. 2025] propose the *MCPGAUGE* benchmark to evaluate the interaction between LLMs and MCP in four dimensions: proactivity, instruction compliance, effectiveness, and overhead. The results reveal a contrast between expectations and practice: even when instructed, LLMs rarely proactively trigger tools via MCP, and on average, performance dropped by 9.5% when integration was used. This conclusion reinforces that the potential of MCP has not yet been fully explored.

Similarly, [Mo et al. 2025] present *LiveMCPBench*, the first large-scale benchmark with over 10,000 MCP servers and 527 tools. The experiments showed that advanced models, such as *Claude-Sonnet-4*, achieve about 79% success in practical tasks, but also point to large variation between models and limitations when applied in tool-rich scenarios. These results suggest that, although promising, the use of MCP in real environments still faces significant barriers.

3.4. LLMs and Time Series

LLMs were not originally designed to handle continuous numerical data, such as time series, but recent research has shown that it is possible to reprogram these models to interpret and reason about this type of information by converting it into textual formats compatible with the models' attention architecture. This approach is demonstrated in the *TIME-LLM* work [Jin et al. 2023], where time series are transformed into linguistic "prototypes", that is textual representations that preserve semantic characteristics of the temporal variation of the data. In this method, the series are divided into windows that capture local patterns, such as rapid increases, gradual declines, or stability, and these windows are mapped to linguistic expressions (and their respective embeddings) that function as interpretable abstractions.

3.5. Observability, debugging and troubleshooting

The growing adoption of distributed architectures based on microservices has brought with it the challenge of understanding and diagnosing dynamic systems composed of multiple interdependent components. In this context, observability emerges as a fundamental requirement for enabling the inspection of the internal state of the system from external data, such as logs, metrics, and traces. A comprehensive review by [Usman et al. 2022] points out that, although advances have been made in terms of telemetry collection and visualization, effective correlation between different data sources is still limited, especially in large-scale and high-cardinality scenarios.

Furthermore, works such as those by Hindriks [Hindriks 2012] and Jonassen and Jung [Jonassen and Hung 2006] reinforce that debugging and troubleshooting activities are, in essence, cognitive processes that require the user to formulate hypotheses about the behavior of the system based on fragmented evidence. In distributed systems, this evidence is typically scattered across logs, failure events, and time series of metrics, making the debugging and troubleshooting process manually intensive, error-prone, and dependent on specialized expertise.

4. Proposal

This work proposes the development of a server based on the open-source Model Context Protocol (MCP), integrated into the observability environment of a Kubernetes cluster, with the aim of providing automated operational context to language models during the debugging and troubleshooting process. The proposal is based on the idea that the integration of log data, metrics, and traces with LLMs can significantly aid in identifying failures in distributed systems. The implementation of this thesis is visually described in Figure 1 below:

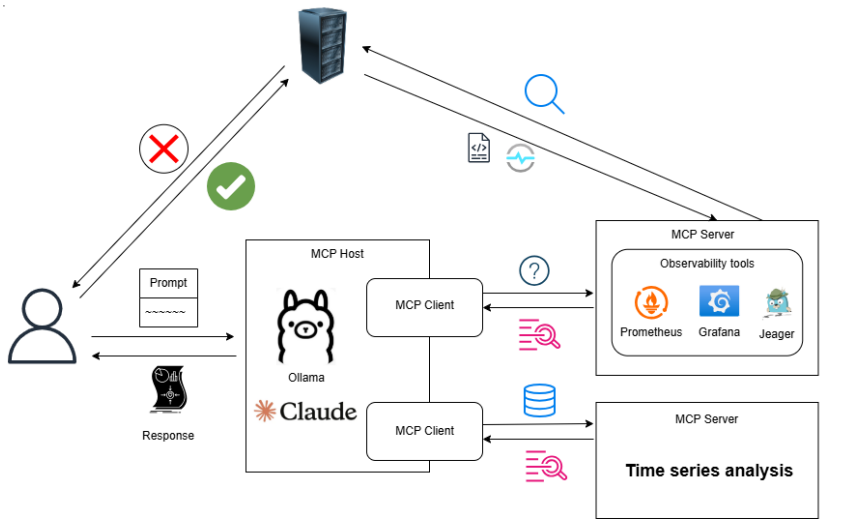

Figure 1. Illustration of the proposed solution

As can be seen in Figure 1, the server has a problem that the user will try to solve with a prompt in an LLM, either in Ollama or Claude. The MCP kicks in and consults the observability tools contained in the MCP server, which in turn will use these tools to obtain the server's logs, metrics, and traces, returning this data to the LLM. In the case

of time series data, another MCP server will be used to analyze this data. Thus, the LLM will be able to return a response with a complete analysis containing all the information necessary to troubleshoot the problem facing the cluster.

The main activities planned for the implementation of the proposal are:

- Configure the testing environment, including creating a Kubernetes cluster and installing the necessary observability tools;
- Develop an MCP server capable of collecting and integrating information from tools such as Grafana, Prometheus, and Jaeger, exposing logs, metrics, and traces via a standardized interface;
- Integrate the MCP server with the Ollama LLM, allowing AI to access the cluster's operational context and use it in the troubleshooting process;
- Implement techniques to improve the LLM's interpretation of time series in order to extract patterns and relevant information that aid in failure analysis;
- Validate the solution in a real environment by integrating the system with the cluster of the Instituto de Informática (INF) of Universidade Federal do Rio Grande do Sul (UFRGS) and conducting experiments with distributed applications.

5. Schedule

This section presents the proposed schedule for the execution of this work, from its initial conception to its final delivery, with activities organized as follows:

- **Literature review:** Study on what MCP is, microservice orchestration, observability tools, collection of articles related to MCP, time series in LLM, debugging with LLM, and MCP in distributed systems to understand the state of the art.
- **Local Kubernetes implementation:** Creation of all necessary configurations to run Kubernetes locally, creation of services and spaces; implementation of observability tools such as Grafana, Prometheus, and Jaeger.
- **Creation of the MCP server:** Creation of one or more MCP servers with observability tools so that data from the observability tools can be returned to LLM.
- **Integration of MCP with Kubernetes:** Integrate LLM and MCP into the Kubernetes server so that problems in the cluster can be debugged.
- **LLM improvements:** With successful integration, improvements will be needed in the analysis of time series data provided by observability tools.
- **Testing and analysis of results:** With the success of the previous steps, a quantitative analysis will be performed on how much the LLM responses have improved and whether it has actually helped to facilitate debugging and troubleshooting of microservices.
- **Integration with the INF cluster:** After all steps are completed, this tool will be provided to the INF to help all students and teachers with the problems they frequently encounter on their servers.

Table 1. TG1 schedule, illustrating the activities and their respective periods of implementation (Jun/25 - Dec/25).

Activities	Jun/25	Jul/25	Ago/25	Set/25	Out/25	Nov/25	Dec/25
Literature review	•	•	•	•	•	•	•
Thesis writing	•	•	•	•	•	•	•
Implementation of local Kubernetes	•	•				•	•
Creation of MCP server		•					•
Adjustments and Review (Advisor)						•	•
Delivery and presentation of TG1						•	•

Table 2. TG2 schedule, illustrating the activities and their respective periods of implementation (Jan/26 - Jul/26).

Activities	Jan/26	Feb/26	Mar/26	Abr/26	Mai/26	Jun/26	Jul/26
Literature review	•	•	•	•	•	•	•
Thesis writing	•	•	•	•	•	•	•
Integration of MCP with Kubernetes	•	•					
LLM improvements			•	•	•		
Test and analysis results				•	•	•	•
Integration with INF clustes						•	•
Delivery and presentation of TG2							•

6. Final Considerations

In this work we discussed the arduous task of troubleshooting microservice systems because observability systems generate a very large amount of logs, metrics, and traces, making it very difficult to find the root cause of the distributed system problem. This work has made the argument that LLM combined with MCP can be a very promising tool for providing solutions for microservice-based system troubleshooting.

The continuation of this project aims to implement LLM with MCP in a distributed system such as Kubernetes, specifically with a time series analysis tool so that LLM can better interpret these data. With this we intend to enable the LLM to provide more accurate responses for the developer and streamline the troubleshooting process.

References

Anthropic (2024). Introducing the model context protocol. https://www.anthropic.com/news/model-context-protocol. Acesso em: 04/10/2025.

Blinowski, G., Ojdowska, A., and Przybyłek, A. (2022). Monolithic vs. microservice architecture: A performance and scalability evaluation. *IEEE access*, 10:20357–20374.

Boten, A. (2022). *Cloud-Native Observability with OpenTelemetry: Learn to gain visibility into systems by combining tracing, metrics, and logging with OpenTelemetry*. Packt Publishing.

De Jesus, M., Sylvester, P., Clifford, W., Perez, A., and Lama, P. (2025). Llm-based multi-agent framework for troubleshooting distributed systems. In *2025 IEEE Cloud Summit*, pages 110–115. IEEE.

Hasan, M. M., Li, H., Fallahzadeh, E., Rajbahadur, G. K., Adams, B., and Hassan, A. E. (2025). Model context protocol (mcp) at first glance: Studying the security and maintainability of mcp servers. *arXiv preprint arXiv:2506.13538*.

Hindriks, K. (2012). Debugging is explaining. In *Principles and Practice of Multi-Agent Systems (PRIMA 2012)*, volume 7455, pages 31–45.

Hou, X., Zhao, Y., Wang, S., and Wang, H. (2025). Model context protocol (mcp): Landscape, security threats, and future research directions. *arXiv preprint arXiv:2503.23278*.

Jiang, J., Wang, F., Shen, J., Kim, S., and Kim, S. (2024). A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*.

Jin, M., Wang, S., Ma, L., Chu, Z., Zhang, J. Y., Shi, X., Chen, P.-Y., Liang, Y., Li, Y.-F., Pan, S., et al. (2023). Time-llm: Time series forecasting by reprogramming large language models. *arXiv preprint arXiv:2310.01728*.

John, B. (2025). Challenges and solutions in data integration for heterogeneous systems. https://www.researchgate.net/publication/389856499_Challenges_and_Solutions_in_Data_Integration_for_Heterogeneous_Systems.

Jonassen, D. H. and Hung, W. (2006). Learning to troubleshoot: A new theory-based design architecture. *Educational Psychology Review*, 18(1):77–110.

Krishnan, N. (2025). Advancing multi-agent systems through model context protocol: Architecture, implementation, and applications. *arXiv preprint arXiv:2504.21030*.

Kumar, P. (2024). Large language models (llms): survey, technical frameworks, and future challenges. *Artificial Intelligence Review*, 57(10):260.

Luo, Z., Shi, X., Lin, X., and Gao, J. (2025). Evaluation report on mcp servers. *arXiv preprint arXiv:2504.11094*.

Mahida, A. (2023). Enhancing observability in distributed systems-a comprehensive review. *Journal Of Mathematical & Computer Applications. Src/Jmca-166. Doi: Doi. Org/10.47363/Jmca/2023 (2)*, 135:2–4.

Mo, G., Zhong, W., Chen, J., Chen, X., Lu, Y., Lin, H., He, B., Han, X., and Sun, L. (2025). Livemcpbench: Can agents navigate an ocean of mcp tools? *arXiv preprint arXiv:2508.01780*.

Ning, L., Liu, L., Wu, J., Wu, N., Berlowitz, D., Prakash, S., Green, B., O'Banion, S., and Xie, J. (2025). User-llm: Efficient llm contextualization with user embeddings. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 1219–1223.

Ray, P. P. (2025). A survey on model context protocol: Architecture, state-of-the-art, challenges and future directions. *Authorea Preprints*.

Singh, A., Ehtesham, A., Kumar, S., and Khoei, T. T. (2025). A survey of the model context protocol (mcp): Standardizing context to enhance large language models (llms). *Preprints*.

Song, W., Zhong, H., Ding, Z., Xue, J., and Li, Y. (2025). Help or hurdle? rethinking model context protocol-augmented large language models. *arXiv preprint arXiv:2508.12566*.

Söylemez, M., Tekinerdogan, B., and Kolukısa Tarhan, A. (2022). Challenges and solution directions of microservice architectures: A systematic literature review. *Applied sciences*, 12(11):5507.

Sukhija, N. and Bautista, E. (2019). Towards a framework for monitoring and analyzing high performance computing environments using kubernetes and prometheus. In *2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 257–262. IEEE.

Thamma, S. R. (2025). Llmops: Transforming log analysis through ai-driven intelligence. https://medium.com/@t.sankar85/llmops-transforming-log-analysis-through-ai-driven-intelligence-6a27b2a53ded. Acesso em: 20/11/2025.

Usman, M., Ferlin, S., Brunstrom, A., and Taheri, J. (2022). A survey on observability of distributed edge & container-based microservices. *IEEE Access*, 10:86904–86919.

Yan, J., Huang, J., Fang, C., Yan, J., and Zhang, J. (2024). Better debugging: Combining static analysis and llms for explainable crashing fault localization. *arXiv preprint arXiv:2408.12070*.

Ye, W., Liu, J., Cao, D., Yang, W., and Liu, Y. (2025). When llm meets time series: Can llms perform multi-step time series reasoning and inference. *arXiv preprint arXiv:2509.01822*.

Yu, X., Chen, Z., Ling, Y., Dong, S., Liu, Z., and Lu, Y. (2023). Temporal data meets llm–explainable financial time series forecasting. *arXiv preprint arXiv:2306.11025*.