Contextualização de observabilidade de Kubernetes usando Large Language Models e Model Context Protocol

Igor Martins Silva[1], **Juliano Araújo Wickboldt**[2]

[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 microserviços é uma tarefa complicada de realizar. As Largue Language Modals(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. 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. Largue 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. The goal is to more broadly integrate observability information (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. Introdução

Aplicações complexas são costumeiramente organizadas através de uma arquitetura baseada em microsserviços, constituindo sistemas distribuídos que produzem uma série de *logs*, métricas e traços de execução. Observar esses dados é de extrema importância para aferir a saúde do sistema e para resolver problemas que esses sistemas podem apresentar.

Para apoiar esse processo, soluções de orquestração e observabilidade, como Kubernetes (na orquestração), Prometheus, Jaeger e Grafana (no monitoramento), têm sido amplamente utilizadas. No entanto, essas ferramentas fornecem dados em grande escala, que muitas vezes são difíceis de correlacionar e interpretar manualmente.

Nesse contexto, as Large Language Models (LLMs) estão cada vez mais presentes no nosso cotidiano, auxiliando em diversas tarefas como, por exemplo, copilotos de código, suporte para testes ou pesquisa de informações. Um terreno ainda pouco explorado é o da utilização de LLMs no processo de *debugging* e *troubleshooting* automatizado de sistemas distribuídos. Para que uma LLM possa efetivamente apoiar na resolução de problemas, ela precisa ser alimentada com um contexto rico, integrando diferentes fontes de dados de observabilidade.

Neste contexto, surge então o Model Context Protocol (MCP) [Anthropic 2024], uma abordagem que permite padronizar o acesso das LLMs a ferramentas e dados externas a ela. Ele funciona como uma entrada USB [Singh et al. 2025] para a LLM, possibilitando adicionar qualquer tipo de programa, fonte de dados e serviços de negócios (como APIs, bancos de dados, GitHub, Google Drive) que melhore a contextualização e proporcione a LLM produzir respostas mais sofisticadas. Um dos problemas é que os MCPs atuais oferecem acesso limitado ou pago para auxiliar ferramentas como o Kubernetes, além de faltar integração rica e contextualizada entre séries temporais, traces e logs.

Com sistemas cada vez maiores e interligados, gerando um volume massivo de dados de observabilidade, analisar esses dados para conseguir fazer *debugging* e *troubleshooting* pode ser uma tarefa extremamente complexa e demorada de ser feita. Para resolver esse problema, este artigo propõe a criação de ferramentas MCP integradas com LLM e Kubernetes para auxiliar na observabilidade das aplicações distribuídas executando sobre um cluster com foco em *debugging* e *troubleshooting*. Espera-se contribuir para reduzir o esforço humano nessas tarefas, aproximando a observabilidade e a inteligência artificial generativa em um cenário realista, potencialmente impactando DevOps, SRE e a confiabilidade de sistemas distribuídos.

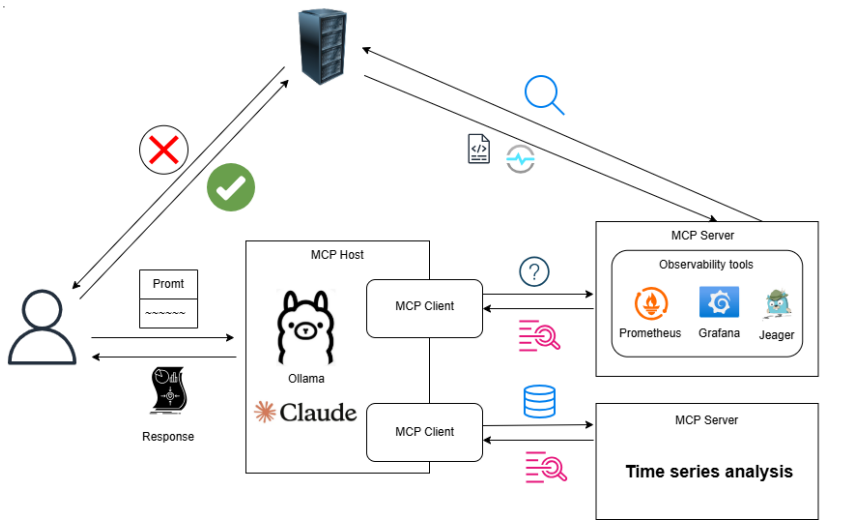

Figure 1. Ilustração da solução proposta.

Como pode se perceber na Figure 1 o servidor tem um problema no qual o usuário vai tentar resolver com um *promt* em uma LLM, seja no Ollama ou o Claude. O MCP entra em ação e consulta as ferramentas de observabilidade contidas no MCP *server* que por usa vez vai ultilizar essa ferramentas para obder os logs, métricas e *traces* do servidor, devolvendo para a LLM esses dados. E no caso dos dados de séries temporais será ultiliza um outro MCP server para fazer a análise desses dados. Assim, a LLM conseguirá repassar uma resposta com uma análise completeta com todas as informações necessárias para fazer o *debugging* e *troubleshooting* do problema que o cluster enfrenta.

2. Fundamentação

2.1. Microsservisos e aplicações distribuídas

A arquitetura de microsserviços surgiu como uma evolução natural em relação aos sistemas monolíticos tradicionais, com o objetivo de aumentar a escalabilidade, a resiliência e a flexibilidade no desenvolvimento de aplicações complexas. Nessa abordagem, o sistema é composto por diversos serviços independentes, cada um responsável por uma funcionalidade específica, que se comunicam entre si por meio de interfaces bem definidas. Essa separação de responsabilidades permite que cada serviço seja desenvolvido, implantado e escalado de forma isolada, com o seu desacoplamento entre os componentes, os microsserviços possuem uma alta escalabilidade, se beneficiando da comunicação assíncrona ou síncrona entre os serviços, fazendo com que ele se torne cada vez mais comum entre grandes projetos de softwares.

Apesar de todos os benefícios, os sistemas baseados em microsserviços introduzem novos desafios significativos de gerenciamento, monitoramento e depuração. A fragmentação da aplicação em dezenas ou centenas de serviços interconectados aumenta a complexidade operacional no qual acarreta em logs e métricas dispersos em várias fontes, tornando a tarefa de juntar todas as informações necessárias para resolver o problema demasiadamente difícil e trabalhosa.

Adicionalmente, o uso de microsserviços implica em um aumento no volume e diversidade de dados operacionais. Cada requisição pode gerar múltiplos registros de log, métricas de desempenho e *traces* distribuídos, compondo uma quantidade massiva de dados que precisa ser analisada em tempo quase real para garantir a confiabilidade do sistema. Nesse contexto, o diagnóstico de falhas depende fortemente de soluções de *observabilidade*.

A identificação da origem de um problema em um ambiente distribuído requer não apenas acesso centralizado aos dados de execução, mas também a capacidade de correlacionar informações heterogêneas provenientes de diferentes serviços e camadas da aplicação. É nesse ponto que surgem desafios fundamentais que este trabalho busca abordar: como integrar as informações de observabilidade de um sistema distribuído de forma eficiente e contextualizada, permitindo que modelos de linguagem (LLMs) possam auxiliar no processo de diagnóstico e resolução de falhas.

2.2. Observabilidade

O processo de *debugging* [Hindriks 2012] é encontrar e reduzir o número de bugs, ou defeitos, em um programa. Um defeito normalmente é detectado porque o programa gera um comportamento inesperado. Para localizar a causa de um defeito, é essencial

explicar por que esse comportamento é gerado. Já o processo de *troubleshooting* refere-se a uma série de etapas seguidas para detectar, identificar, analisar, corrigir e prevenir problemas em sistemas de computador. Envolve tomar conhecimento de um problema, caracterizá-lo, determinar suas causas subjacentes, restaurar ou substituir a função afetada e tomar medidas para evitar problemas futuros. E para se fazer esses processos é necessário se utilizar da *observabilidade*.

Observabilidade é "a capacidade de medir o estado interno de um sistema apenas por meio de suas saídas externas" [Usman et al. 2022]. Ela permite investigar e diagnosticar problemas, mesmo aqueles que não eram esperados, em arquiteturas distribuídas e dinâmicas. As saídas externas são dados de telemetria de três tipos principais: métricas, logs e *traces*. Métricas são valores quantificáveis obtidos por meio de bibliotecas ou da infraestrutura subjacente, que permite identificar tendências, gargalos e comportamentos anômalos com maior facilidade. Os logs oferecem registros textuais detalhados de eventos que ocorrem em cada serviço, como erros, avisos e alterações de estado. Eles são essenciais para análises profundas e diagnósticos pós-incidente, mas seu volume elevado e formato não estruturado tornam a interpretação manual complexa. Por fim, *traces* são identificadores responsáveis por capturar solicitações dentro de um sistema que compreende interações de componentes.

Mesmo com ferramentas sofisticadas, como Prometheus, Grafana, Elastic Stack, OpenTelemetry e Jaeger, que são responsáveis por coletar, armazenar e visualizar essas informações. A análise manual de observabilidade continua sendo uma tarefa complexa, sobretudo quando se lida com ambientes massivamente distribuídos. Em aplicações compostas por dezenas ou centenas de microsserviços, cada requisição pode gerar uma cascata de eventos e registros, resultando em milhões de pontos de dados por minuto. Interpretar e correlacionar essas informações de maneira eficiente exige conhecimento profundo da arquitetura do sistema, da lógica de negócio e das interdependências entre componentes. Além disso, erros frequentemente surgem de interações entre serviços, o que torna a identificação da origem do problema um processo demorado e sujeito a falhas humanas.

Outro desafio importante está na fragmentação da informação. Logs, métricas e *traces* são coletados e armazenados por ferramentas distintas e muitas vezes analisados de forma isolada, dificultando o entendimento integrado do comportamento do sistema. Embora exista um movimento crescente em direção à padronização e unificação da telemetria com iniciativas como o OpenTelemetry, ainda há barreiras significativas quanto à correlação semântica dos dados — ou seja, entender não apenas o que aconteceu, mas o porquê aconteceu. Por isso a obeservabilidade é o conceito ideal para auxiliar não só humanos, mas principalmente as LLMs nessa árdua tarefa. Pois elas precisam desse compilado de informações para poder dar uma resposta mais assertiva.

2.3. LLM e séries temporais

LLMs são modelos de inteligência artificila(IA) baseados em redes neurais profundas, treinados em grandes volumes de texto, que conseguem compreender linguagem natural e raciocinar sobre o contexto textual. Com essas qualidades elas veem sendo utilizadas cada vez mais em diversos aspectos, inclusive no processo de *debugging* e *troubleshooting* para explicar os erros e tentar solucionar os problemas relacionados ao erro.

Contudo, a eficácia de uma LLM no diagnóstico de problemas depende diretamente do nível de contextualização das informações que ela recebe e de como os interpretar. Sem dados suficientes como logs representativos, descrições completas do sistema ou histórico de eventos, o modelo pode gerar respostas imprecisas, assumir hipóteses incorretas convergindo sempre para uma solução errada. Assim, tarefas de *debugging* e *troubleshooting* com LLMs frequentemente exigem múltiplos *prompts*, refinamentos e tentativas sucessivas até que o modelo seja capaz de compreender corretamente o cenário analisado.

Além disso, há o problema de interpretar os dados de séries temporais. Que são sequências de dados numéricos ou categóricos coletados e registrados ao longo do tempo, em intervalos regulares ou irregulares. Elas permitem analisar o comportamento dinâmico de um sistema, identificar tendências, localizar anomalias e prever valores futuros com base em seu histórico. Mas as LLMs não são tão boas em entender estes dados numéricos, sendo necessário mapear trechos de séries temporais para tokens semânticos que descrevem comportamentos típicos (por exemplo: “aumento repentino”, “tendência de queda”, “pico anômalo”), cria-se uma ponte entre o domínio numérico e o simbólico, permitindo que a LLM raciocine sobre dados temporais da mesma forma que raciocina sobre linguagem natural.

Esse desafio fica ainda maior mais evidente em sistemas distribuídos e baseados em microsserviços, onde erros podem envolver inúmeros componentes e fluxos de comunicação. Para que as LLMs possam realmente contribuir para um processo de *debugging* e *troubleshooting* eficiente, é necessário garantir que elas tenham acesso a dados operacionais estruturados e correlacionados, preferencialmente em tempo próximo ao real. Dessa forma, reduz-se o esforço humano em coletar e organizar manualmente informações de telemetria, enquanto se potencializa a capacidade do modelo em gerar análises mais assertivas e automatizadas.

Desse modo, embora as LLMs apresentem grande potencial para apoiar o diagnóstico de falhas em ambientes complexos, ainda há uma limitação fundamental relacionada ao fornecimento de contexto adequado e a análise de séries temporais. Essa lacuna tem impulsionado o desenvolvimento de novos meios de integração entre LLMs e sistemas externos, possibilitando que elas acessem logs, métricas, traces e outros artefatos relevantes de forma automática.

2.4. MCP

Por mais brilhante que seja as LLMs, elas ainda falham em se contextualizar com poucas palavras no *prompt*, sendo sempre necessário dar o muito mais informações para que ela responda mais assertivamente. Então com o requisito de deixar mais sofisticada a resposta das LLMs, começou-se a pensar em melhorias para que ela tivesse acesso a todas informações necessárias para fornecer a melhor resposta possível. As primeiras implementações eram esquemas simples de solicitação-resposta baseados em HTTP, o que era suficiente para tarefas simples, mas o uso de Application Programming Interfaces(APIs) mostrou desafios significativos[Singh et al. 2025] como complexidade de integração, problemas de escalabilidade, barreiras de interoperabilidade e riscos de segurança.

Em essência, o MCP atua como uma camada intermediária entre o modelo e os sistemas externos, permitindo que o modelo descubra, consulte e manipule ferramentas de maneira transparente. Através de uma interface padronizada, o servidor MCP pode expor

dados ou funcionalidades de diversos sistemas, como bancos de dados, APIs, aplicações corporativas, serviços de observabilidade ou até mesmo sistemas operacionais. Assim, o modelo deixa de depender exclusivamente do conteúdo textual de seus *prompts* e passa a utilizar informações atualizadas e contextualmente relevantes, provenientes de fontes externas confiáveis.

Assim surgiu o MCP para resolver esse problema, baseado em protocolos como o Language Server Protocol (LSP) que padronizou a comunicação entre entre Integrated Development Environment(IDE) e servidores de linguagens. Ele possui uma arquitetura que separa *host*, *client* e *servers*, possibilitando incrementar as suas capacidades isoladamente. Ele não só conseguiu preencher o gap que faltava nas LLMs, estabelecendo uma estrutura unificada que simplifica significativamente as interações entre aplicações de IA e recursos externos, mas como também fornece uma base sólida para sistemas com agentes de inteligencia artificial(IA) capazes de ter percepção autonoma, raciocínio, tomada de decisão e interação com o mundo real. Tudo isso faz com que o MCP se torne um componente crítico na trajetória de desenvolvimento futuro de sistemas de IA.

2.4.1. Arquitetura do MCP

O Protocolo funciona de forma bidirecional, onde o *Host*(geralmente uma LLM ou IDE) faz requisições, o *client* media essas interações e o *server* fornece os dados ou executa as ações.

A. Server

O Server é responsável por fornecer serviços e dados contextuais para a LLM, implementa a lógica de negocio, respondendo as requisições estruturadas do *client*. Pode existir diversos servidores MCP rodando simultaneamente, tanto localmente como remotamente, cada um representando um domínio de contexto diferente.

B. Host

É o componente da orquestração central da arquitetura. É responsável por registrar e descobrir os servidores disponíveis, expondo suas capacidades ao *client*. Em muitos cenários, o *host* é o componente que executa junto com a LLM (por exemplo, dentro de uma IDE, ou de uma ferramenta como o Ollama). Atua também como camada de segurança e isolamento, controlando permissões e validando as operações.

C. Client

Já o *client* representa os componentes individuas dentro do host enviando requisições, solicitando o uso de determinada ferramenta ou a leitura de dados do MCP server, mantendo uma conexão um para um com ele. O *client* não precisa conhecer detalhes de implementação dos servidores, apenas as capacidades expostas via MCP. E também recebe as respostas processadas que por sua vez passa para o host que as transfere para as LLMs.

2.4.2. MCP local vs. MCP remoto

No modo local o servidor MCP é executado no mesmo ambiente do host ou do client, permitindo uma comunicação direta e de baixa latência. Essa configuração

favorece o controle sobre os dados e a segurança da execução, sendo ideal para cenários de desenvolvimento, depuração local e experimentação, nos quais a privacidade e a velocidade de resposta são prioritárias.

Por outro lado, o MCP remoto consiste em um servidor independente, acessado por meio de interfaces de rede (como HTTP ou WebSocket), permitindo a integração da LLM com fontes externas de dados e serviços distribuídos. Esse modelo é mais adequado para aplicações em larga escala, em que múltiplos clientes precisam acessar simultaneamente o mesmo conjunto de ferramentas e contextos. Entretanto, sua adoção impõe desafios adicionais de autenticação, latência e segurança, exigindo mecanismos robustos de autorização e criptografia.

3. Trabalhos Relacionados

Nessa seção vamos subdividir os temas: na 3.1, artigos que abordem sobre o que é o MCP; na 3.2, aspectos de segurança; 3.3, benchmarks e avaliações experimentais; 3.4, LLMs e séries temporais; 3.5, artigos que abordem a questão da observabilidade, *debugging* e *troubleshooting*.

3.1. Contexto Geral - MCP como uma solução emergente

O *Model Context Protocol* (MCP) é um protocolo inovador para a era da inteligência artificial, por sua proposta de fornecer uma interface padronizada para conectar LLMs a ferramentas externas, bancos de dados e serviços em tempo real. Diferentemente de mecanismos prévios como *function calling*, o MCP busca atuar como uma "porta universal" ou mesmo um "USB para LLMs" [Singh et al. 2025], permitindo que modelos de linguagem ampliem seu contexto por meio da integração dinâmica com dados e sistemas distribuídos.

O *survey* de [Ray 2025] apresenta o protocolo sob a ótica de sistemas de comunicação, descrevendo sua arquitetura em camadas, ciclo de vida e aplicações em domínios como saúde, cibersegurança e automação em nuvem, além de discutir problemas como latência, sobrecarga de *tokens* e riscos de escalonamento de privilégios. De forma complementar, [Singh et al. 2025] caracterizam o MCP como uma abordagem fundamental para padronizar a integração entre LLMs e ferramentas externas, destacando seus benefícios e limitações. Já [Krishnan 2025] aprofunda o papel do MCP em arquiteturas de sistemas multiagentes, explorando como o protocolo pode facilitar a cooperação entre agentes inteligentes e ampliar a retenção de contexto em interações distribuídas. Esses trabalhos fornecem uma base teórica sólida, mas permanecem em nível conceitual, sem avaliar o uso do MCP em cenários práticos de depuração de sistemas distribuídos.

3.2. Segurança e Manutenibilidade

Outro eixo importante da literatura envolve a análise de segurança e confiabilidade no ecossistema MCP. [Hou et al. 2025] discutem ameaças de segurança, riscos de privacidade e os desafios em cada fase do ciclo de vida de um servidor MCP (criação, operação e atualização). Nessa mesma linha, [Hasan et al. 2025] avaliam aspectos de segurança e manutenção, chamando a atenção para a necessidade de práticas robustas de proteção contra ataques e de estratégias que garantam a confiabilidade a longo prazo. Esses estudos evidenciam preocupações legítimas quanto à adoção em larga escala, mas

não exploram como tais riscos impactam diretamente a integração do MCP com sistemas de observabilidade.

3.3. Benchmarks e Avaliações Experimentais

Com a popularização do MCP, surgiram também estudos voltados à sua avaliação prática. [Luo et al. 2025] conduzem uma comparação entre servidores MCP e chamadas de função, mostrando que, em muitos casos, o uso de MCP não resulta em ganhos expressivos de desempenho. Entretanto, os autores apontam que a precisão pode ser melhorada quando há otimização dos parâmetros que o LLM deve construir para interagir com o servidor. Em outra frente, [Song et al. 2025] propõem o *benchmark* MCPGAUGE para avaliar a interação entre LLMs e MCP em quatro dimensões: proatividade, obediência a instruções, efetividade e sobrecarga. Os resultados revelam um contraste entre expectativas e prática: mesmo quando instruídas, as LLMs raramente acionam ferramentas via MCP de forma proativa, e em média a performace caiu 9,5% quando a integração foi utilizada. Essa conclusão reforça que o potencial do MCP ainda não está plenamente explorado.

No mesmo sentido, [Mo et al. 2025] apresentam o *LiveMCPBench*, primeiro *benchmark* em larga escala, com mais de 10.000 servidores MCP e 527 ferramentas. Os experimentos mostraram que modelos avançados, como *Claude-Sonnet-4*, alcançam cerca de 79% de sucesso em tarefas práticas, mas também apontam grande variação entre modelos e limitações quando aplicados em cenários ricos em ferramentas. Esses resultados sugerem que, embora promissor, o uso do MCP em ambientes reais ainda enfrenta barreiras significativas.

3.4. LLMs e Séries Temporais

LLMs não foram originalmente projetadas para lidar com dados numéricos contínuos, como séries temporais, mas pesquisas recentes têm mostrado que é possível reprogramar esses modelos para interpretar e raciocinar sobre esse tipo de informação ao convertê-la para formatos textuais compatíveis com a arquitetura de atenção dos modelos. Essa abordagem é demonstrada no trabalho TIME-LLM [Jin et al. 2023], onde séries temporais são transformadas em "prototypes" linguísticos — representações textuais que preservam características semânticas da variação temporal dos dados. Nesse método, as séries são divididas em janelas que capturam padrões locais, como aumentos rápidos, quedas graduais ou estabilidade, e essas janelas são mapeadas para expressões linguísticas (e seus respectivos embeddings) que funcionam como abstrações interpretáveis.

3.5. Observabilidade, *debugging* e *troubleshooting*

A crescente adoção de arquiteturas distribuídas baseadas em microsserviços trouxe consigo o desafio de compreender e diagnosticar sistemas dinâmicos compostos por múltiplos componentes interdependentes. Nesse contexto, a observabilidade surge como um requisito fundamental para possibilitar a inspeção do estado interno do sistema a partir de dados externos, como logs, métricas e traces. Uma revisão abrangente de [Usman et al. 2022] aponta que, embora avanços tenham sido feitos em termos de coleta e visualização de telemetria, a correlação eficaz entre diferentes fontes de dados ainda é limitada, sobretudo em cenários de grande escala e alta cardinalidade.

Complementarmente, trabalhos como [Hindriks 2012] e [Jonassen and Hung 2006] reforçam que atividades de debugging e troubleshooting são, em essência, processos cognitivos que exigem do usuário a formulação de

hipóteses sobre o comportamento do sistema, com base em evidências fragmentadas. Em sistemas distribuídos, essas evidências são tipicamente dispersas ao longo de logs, eventos de falha e séries temporais de métricas, tornando o processo de textitdebugging e *troubleshooting* manualmente intensivo, susceptível a erros e dependente de expertise especializada.

4. Proposta

Este trabalho propõe o desenvolvimento de um servidor baseado no Model Context Protocol (MCP), de código aberto, integrado ao ambiente de observabilidade de um cluster Kubernetes, com o objetivo de fornecer contexto operacional automatizado a modelos de linguagem durante o processo de *debugging* e *troubleshooting*. A proposta se fundamenta na ideia de que a integração entre dados de logs, métricas e traces com LLMs pode auxiliar significativamente na identificação de falhas em sistemas distribuídos. As principais atividades previstas para a implementação da proposta são:

- Configurar o ambiente de testes, incluindo a criação de um cluster Kubernetes e a instalação das ferramentas de observabilidade necessárias;
- Desenvolver um servidor MCP capaz de coletar e integrar informações de ferramentas como Grafana, Prometheus e Jaeger, expondo logs, métricas e traces via uma interface padronizada;
- Integrar o servidor MCP com a LLM Ollama, permitindo que a IA acesse o contexto operacional do cluster e possa utilizá-lo no processo de *debugging*;
- Implementar técnicas para aprimorar a interpretação de séries temporais por parte da LLM, de modo a extrair padrões e informações relevantes que auxiliem na análise de falhas;
- Validar a solução em ambiente real, integrando o sistema ao cluster do Instituto de Informática (INF) da Universidade Federal do Rio Grande do Sul (UFRGS) e realizando experimentos com aplicações distribuídas.

5. Cronograma

Essa seção apresenta o cronograma proposto para a execução desse trabalho, desde sua concepção inicial até a sua entrega final, com as atividades organizadas da seguinte forma:

- **Revisão bibliográfica:** Estudo sobre o que é MCP, sobre orquestração de microsserviços, ferramentas de observabilidade, coleta de artigos relacionados com MCP, series temporais em LLM, debbuging com com LLM, e MCP em sistemas distribuídos para saber qual o estado-da-arte,
- **Implementação do Kubernetes local:** Criação de todas as configurações necessárias para rodar o kubernetes localmente, criação de serviços e spaces; implementação de ferramentas de observabilidade como, por exemplo, Grafana, Prometheus e Jaeger.
- **Criação do server MCP:** Criação de um ou mais servidores MCP com as ferramentas de observabilidade para que possa retornar para a LLM os dados das ferramentas de observabilidade.
- **Integração do MCP com Kubernetes:** Integrar o LLM e MCP no servidor Kubernetes para que possa ser feito o debbuging de problemas no cluster.

- **Melhorias da LLM:** Com a integração bem sucedida, será necessário melhorias na analise dos dados de series temporais fornecidos pelas ferramentas de observabilidade
- **Testes e analises de resultados:** Com o exito das etapas anteriores será feito uma análise quantitativa sobre o quanto melhorou as respostas da LLm e se realmente ajudou a facilitar o debbugin e troubleshoting dos microsserviços
- **Integração com o cluster do INF:** Após todas as etapas será fornecido para o INF essa ferramenta, visando ajudar todos os alunos e professores com os problemas que eles enfrentam frequentemente no seus servidores.

Table 1. Cronograma TG1, ilustrando as atividades e seu respectivo período de realização (Jun/25 - Dez/25).

Atividades	Jun/25	Jul/25	Ago/25	Set/25	Out/25	Nov/25	Dez/25
Revisão bibliográfica	•	•	•	•	•	•	•
Escrita da tese	•	•	•	•	•	•	•
Implementação do Kubernetes local	•	•				•	•
Criação do server MCP		•					•
Ajustes e Revisão (Orientador)						•	•
Entrega e apresentação do TG1						•	•

Table 2. Cronograma TG2, ilustrando as atividades e seu respectivo período de realização (Jan/26 - Jul/26).

Atividades	Jan/26	Fev/26	Mar/26	Abr/26	Mai/26	Jun/26	Jul/26
Revisão bibliográfica	•	•	•	•	•	•	•
Escrita da tese	•	•	•	•	•	•	•
Integração do MCP com Kubernetes	•	•					
Melhorias da LLM			•	•	•		
Testes e analises de resultados				•	•	•	•
Integração com o cluster do INF						•	•
Entrega e apresentação do trabalho final (TG2)							•

References

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

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. 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*.

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*.

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*.

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

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*.

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*.

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.