Maurício Gallera de Almeida

Processamento de dados e Modelagem de Séries Temporais para Detecção de Anomalias em Arquiteturas de Microsserviço

São Carlos, SP

2025

Maurício Gallera de Almeida

Processamento de dados e Modelagem de Séries Temporais para Detecção de Anomalias em Arquiteturas de Microsserviço

Trabalho de Conclusão de Curso apresentado ao curso de Engenharia de Computação da Universidade Federal de São Carlos, como requisito parcial para a obtenção do título de Bacharel em Engenharia de Computação.

Universidade Federal de São Carlos – UFSCar

Departamento de Computação

Programa de Graduação

Orientador: Daniel Lucrédio

São Carlos, SP

2025

Almeida, Maurício Gallera De

Processamento de dados e Modelagem de Séries Temporais para Detecção de Anomalias em Arquiteturas de Microsserviço / Maurício Gallera De Almeida -- 2025.
60f.

TCC (Graduação) - Universidade Federal de São Carlos, campus São Carlos, São Carlos
Orientador (a): Daniel Lucrédio
Banca Examinadora: Alan Demétrius Baria Valejo, Daniel Lucrédio, Valter Vieira de Camargo
Bibliografia

1. Processamento de Dados. 2. Aprendizado Profundo. 3. Microsserviços. I. Almeida, Maurício Gallera De. II. Título.

Ficha catalográfica desenvolvida pela Secretaria Geral de Informática (SIn)

DADOS FORNECIDOS PELO AUTOR

Bibliotecário responsável: Arildo Martins - CRB/8 7180

Dedico este trabalho aos meus pais, Sidney e Rosa, que, sob muito sol, me permitiram chegar até aqui, na sombra.

Agradecimentos

Gostaria de agradecer, primeiramente, aos meus pais. Desde criança, recebi todo apoio e incentivo. Do simples livro de literatura comprado para incentivar a leitura até o apoio de minha permanência fora de minha cidade natal, nunca houve medição de esforços. Somente com esse apoio é que, sem problemas que me impedissem e com tempo em sobra, me restou estudar. E estudei motivado e sabendo do potencial que havia nisso, motivo pelo qual cheguei até aqui.

Quero agradecer também minhas irmãs que, de longe e em seus próprios caminhos, tenho certeza que torceram por mim e se alegraram com cada pequena vitória.

Não posso deixar de fora também meu professor orientador, Dr. Daniel Lucrédio, com quem fiz uma pesquisa e também este trabalho. Sempre muito solícito, me orientou ao longo de toda jornada, tendo papel crucial na superação de todos desafios. Vale agradecer ainda, todos professores que contribuíram com minha formação, do Ensino Fundamental a Graduação. Valorizo mais ainda os muitos que contribuiram no despertar do meu interesse genuíno em ciência, matemática e, principalmente, computação.

Por fim, agradeço a todos que fizeram da minha graduação uma jornada mais leve e divertida. São eles meus amigos, que compartilharam desafios comigos, e minha namorada, Mariana, que pacientemente me ouviu em momentos difíceis e torceu para o meu sucesso.

"Tenho o privilégio de não saber quase tudo,
e isso explica o resto."
(Manoel de Barros.)

Resumo

A arquitetura de microsserviços tem ganhado ampla adoção devido à sua escalabilidade e flexibilidade, mas também impõe desafios relevantes de monitoramento e detecção de falhas. Análises manuais e métodos estatísticos simples frequentemente resultam em falsos positivos ou negativos, prejudicando a identificação precisa de anomalias. Este trabalho propõe uma metodologia completa para detecção de anomalias em ambientes de microsserviços por meio de processamento de métricas de séries temporais e aplicação de Aprendizado Profundo. Como não houve acesso a dados reais, foi desenvolvida uma base sintética verossímil contendo métricas de latência HTTP, *lag Kafka* e uso de CPU, reproduzindo problemas reais como frequências distintas de coleta, lacunas temporais, outliers e distribuição descentralizada das métricas. Em seguida, elaborou-se um pipeline de pré-processamento envolvendo padronização temporal, transformação de formato, imputação de dados faltantes e enriquecimento informacional, dentre outros. Anomalias simuladas — representando crashes comuns em produção — foram posteriormente inseridas no conjunto de dados. Por fim, um modelo inicial baseado em CNN 1D foi treinado e avaliado para a tarefa de classificação binária entre comportamento normal e anômalo. Os resultados obtidos apresentaram bons valores de precisão, recall e F1-score, indicando o potencial da abordagem, ainda que limitada pelo uso de dados artificiais. O estudo demonstra a viabilidade de aplicar técnicas de aprendizado profundo para detecção de anomalias em sistemas distribuídos e estabelece bases para trabalhos futuros com dados reais e modelos mais refinados.

Palavras-chave: detecção de anomalias. microsserviços. séries temporais. aprendizado profundo. redes neurais. pré-processamento de dados.

Abstract

Microservices architecture has gained widespread adoption due to its scalability and flexibility, but it also poses significant challenges for monitoring and fault detection. Manual analysis and simple statistical methods often result in false positives or negatives, hindering the accurate identification of anomalies. This work proposes a comprehensive methodology for anomaly detection in microservices environments through time series metric processing and the application of deep learning. As there was no access to real data, a plausible synthetic database was developed containing HTTP latency, Kafka lag, and CPU usage metrics, reproducing real problems such as different collection frequencies, temporal gaps, outliers, and decentralized distribution of metrics. Next, a preprocessing pipeline was developed involving temporal standardization, format transformation, missing data imputation, and information enrichment, among others. Simulated anomalies—representing common crashes in production—were then inserted into the dataset. Finally, an initial model based on 1D CNN was trained and evaluated for the task of binary classification between normal and anomalous behavior. The results obtained showed good precision, recall, and F1-score values, indicating the potential of the approach, although limited by the use of artificial data. The study demonstrates the feasibility of applying deep learning techniques for anomaly detection in distributed systems and lays the groundwork for future work with real data and more refined models.

Keywords: anomaly detection. microservices. time series. deep learning. neural networks. data preprocessing.

Lista de ilustrações

Lista de quadros

Lista de tabelas

Lista de abreviaturas e siglas

CNN	Convolutional Neural Networks

Sumário

1 Introdução

A arquitetura de microsserviços tem sido amplamente adotada por desenvolvedores e equipes de software devido aos benefícios alinhados às práticas da Metodologia Ágil. No entanto, essa abordagem também impõe desafios significativos, como vulnerabilidades de segurança, alta complexidade de gerenciamento, implantação e orquestração, dificuldade de manter uma visão global da aplicação e das interações entre serviços, além de possíveis falhas e inconsistências de dados. Além desses, destaca-se a dificuldade em alcançar um método eficiente para detecção rápida e precisa de falhas e problemas reais em produção usando das métricas do serviço — esse problema compõe o foco desta monografia.

Segundo Lewis e Fowler (2014), uma arquitetura de microsserviços é composta por pequenos componentes independentes, cada um sendo um serviço autônomo com sua função específica, mas juntos formando uma aplicação maior. A independência entre serviços permite maior isolamento de falhas, paralelismo de desenvolvimento em times que são responsáveis por serviços diferentes, velocidade maior de implantação, dentre outras vantagens, como possibilidade de adequação da tecnologia usada de acordo com o serviço e sua responsabilidade.

Apesar das vantagens, a independência entre os serviços e funções de um sistema traz também diversas dificuldades adicionais. Entre elas, destaca-se a detecção de problemas reais em serviços em produção — como falhas introduzidas por novos códigos, ataques cibernéticos, lentidão de resposta devido ao aumento de demanda ou ao provisionamento insuficiente de recursos —, um desafio crítico com impacto direto na estabilidade da arquitetura. Nobre, Pires e Reis (2023) afirmam que essa independência, ao elevar a complexidade de gerenciamento e o entendimento global da aplicação, torna mais lenta a identificação e a mitigação manual de falhas, afetando a experiência do usuário, a qualidade do serviço e, em casos extremos, levando à interrupção do funcionamento. Por outro lado, um sistema de detecção de anomalias mal calibrado pode gerar falsos alarmes, resultando em desperdício de tempo de engenheiros de software na investigação de problemas inexistentes.

Sistemas automáticos de detecção de anomalias, responsáveis por gerar alarmes, são amplamente utilizados por empresas que buscam identificar rapidamente alterações no comportamento normal de seus serviços, sobretudo aqueles essenciais ao funcionamento da aplicação como um todo. No entanto, é comum que tais sistemas sejam desenvolvidos a partir de análises manuais realizadas por engenheiros de software, que procuram definir o que é considerado comportamento normal com base nas principais métricas do serviço, ou ainda que empreguem métodos estatísticos simples. Essa prática amplia a probabilidade

de ocorrência de falsos positivos — quando parâmetros excessivamente sensíveis geram alarmes indevidos, causando desperdício de tempo de trabalho — ou de falsos negativos, que atrasam a detecção de problemas reais capazes de afetar os usuários da aplicação, quando os parâmetros são definidos de forma demasiadamente rígida. Parte desse problema decorre do método empregado na detecção de falhas: análises manuais e métodos estatísticos simples tendem a ser insuficientes para compreender padrões complexos em dados diversos e inter-relacionados de maneira não evidente. Ademais, o comportamento normal das métricas de um serviço pode seguir padrões temporais extensos, variando ao longo do dia, da semana ou até do mês, configurando uma série temporal complexa que contrasta com as análises de janelas curtas geralmente associadas a métodos mais simples. Outro fator agravante é que, tanto em abordagens simples quanto em mais sofisticadas, os dados disponíveis frequentemente não são devidamente tratados nem enriquecidos com informações adicionais. Com isso, perde-se o potencial de extração de novos padrões, e problemas como ausência de dados, diferentes frequências de coleta, descentralização no armazenamento e análise das métricas, entre outros, dificultam a compreensão do comportamento típico usado como base para identificação de comportamentos anômalos quando eles de fato ocorrem.

Neste contexto, este trabalho propõe o desenvolvimento de uma metodologia voltada a contribuir para o processo de detecção de anomalias em ambientes de microsserviços. Para isso, foram desenvolvidos algoritmos capazes de processar métricas dos serviços obtidas por meio de logs da plataforma Grafana e armazenadas em banco de dados sem tratamento prévio ou enriquecimento informacional. Assim, busca-se transformar métricas dispersas e não tratadas, originalmente dispostas em tabelas separadas, em um único conjunto de dados estruturado, com formatos mais adequados e informações mais ricas para aplicação em algoritmos de Aprendizado de Máquina — em especial, aqueles voltados à detecção de anomalias em séries temporais.

O pipeline desenvolvido enfrenta diversos desafios, como ausência de dados, diferentes frequências de coleta entre métricas e múltiplos registros para uma mesma métrica, considerando ambiente com *sharding*, dentre outros que serão detalhados posteriormente. Após o tratamento dos dados, realizou-se a aplicação em um algoritmo de Aprendizado de Máquina escolhido, a Rede Neural Convolucional (CNN) de 1 dimensão, considerando que os dados variam apenas no tempo.

Cabe destacar que uma dificuldade recorrente em trabalhos dessa natureza é o acesso a dados reais, geralmente não disponibilizados por empresas. Por essa razão, este estudo simulou dados representativos de um ambiente de microsserviços, replicando seu formato e comportamento observados em bases reais — às quais não se teve permissão de uso ou citação —, porém sem empregar diretamente esses dados. Considerando essa simulação, não é objetivo deste trabalho um alto grau de refinamento do modelo usado e

garantia de acurácia do mesmo na tarefa de detecção de anomalias, aspectos que podem ser aprimorados em pesquisas futuras com o uso de dados reais. No entanto, as contribuições deste trabalham permitem que pesquisadores e profissionais interessados em aprimorar suas soluções de monitoramento usem dos conhecimentos produzidos aqui para tal fim usando dados reais e fazendo esta etapa de refinamento fino de um modelo para obtenção de resultados.

Este trabalho é organizado em cinco capítulos. Neste primeiro, apresentou-se a contextualização da proposta de trabalho e seus fins. No Capítulo 2, serão apresentados a fundamentação teórica e trabalhos relacionados. O Capítulo 3 apresenta a metodologia usada ao longo do desenvolvimento do trabalho, enquanto o Capítulo 4 descreve os resultados obtidos. Ao fim, no Capítulo 5, pode-se encontrar a conclusão e os trabalhos futuros propostos a partir deste.

2 Fundamentação teórica

Este Capítulo contém os principais conceitos envolvidos para entendimento deste trabalho, na Seção 2.1, bem como alguns trabalhos relacionados, na Seção 2.2.

2.1 Principais conceitos

Este trabalho simula dados de métricas de uma arquitetura de microsserviços para obtê-los de maneira não tratada e enriquecidos de informações, no mesmo formato como se observou em uma base de dados real anônima, mas sem utilizar dessa. Métricas de latência de resposta em endpoints HTTP, quantidade de mensagens em fila para consumo e uso de CPU por instância foram as métricas observadas. Usou-se tais dados então para aplicá-los em um pipeline que os processa e enriquece de informações para uso em um algoritmo de aprendizado e detecção de anomalias, com objetivo de melhorar o processo de identificação de problemas reais em produção. O algoritmo utilizado para tal foi a rede neural CNN 1D. Todos os conceitos citados aqui são brevemente definidos neste capítulo para compreensão do trabalho.

2.1.1 Arquitetura de Microserviços, Comunicação Síncrona e Assíncrona e Sharding

De acordo com Lewis e Folwer (2014), arquitetura de microsserviços é baseada na divisão de uma aplicação em diferentes componentes, sendo estes independentes entre si. Cada microsserviço tem uma responsabilidade bem definida dentro da arquitetura, geralmente associada a uma lógica específica do domínio envolvido. Isso permite isolamento de falhas, velocidade de desenvolvimento e implantação – considerando possibilidade de desacoplamento de tais tarefas entre times de desenvolvimento responsáveis por serviços ou domínios diferentes -, escalabilidade independente de cada serviço, possibilidade de adequação de tecnologias usadas para cada microsserviço, considerando sua função e responsabilidades, dentre outras vantagens.

Uma vez independentes, os serviços precisam de alguma maneira de se comunicar. Neste sentido, comunicação entre serviços é o conjunto de mecanismos que permite que diferentes partes de um sistema distribuído troquem informações e coordenem ações. Na comunicação síncrona, feita via requisições HTTP, um serviço envia uma requisição diretamente a outro e aguarda sua resposta antes de continuar o processamento. Esse modelo é simples e direto, mas gera acoplamento temporal entre os serviços: se o serviço chamado estiver lento ou indisponível, o chamador também é afetado. Já na comunicação

assíncrona, feita via sistemas de mensageria, os serviços não interagem diretamente; em vez disso, produzem e consomem mensagens através de um intermediário — o broker de mensagens. Esse modelo desacopla o tempo de envio e processamento, permitindo que os serviços continuem funcionando mesmo que outros estejam temporariamente indisponíveis, além de facilitar escalabilidade e processamento distribuído.

Dentro do contexto de mensageria, uma tecnologia amplamente utilizada é o Apache Kafka[1], que funciona como uma plataforma de streaming distribuída baseada em logs particionados. Em *Kafka*, as mensagens são organizadas em tópicos e armazenadas de forma imutável, permitindo que múltiplos consumidores leiam dados em ritmos diferentes sem interferir entre si. Os tópicos são divididos em partições distribuídas entre diversos brokers, garantindo alta disponibilidade e escalabilidade horizontal. Além disso, *Kafka* fornece fortes garantias de durabilidade, suporte a alto throughput e capacidade de reprocessamento de mensagens, o que o torna particularmente adequado para sistemas de grande volume de dados, processamento de eventos e arquiteturas orientadas a eventos.

Arquitetura de sharding é uma abordagem de distribuição de dados ou processamento em que um sistema é dividido em múltiplas partes menores, chamadas shards. Cada shard funciona de forma relativamente independente, armazenando ou processando apenas o subconjunto que lhe pertence, o que permite distribuir carga, aumentar a escalabilidade horizontal e reduzir pontos únicos de gargalo. Assim como em microsserviços, essa separação melhora a tolerância a falhas — pois a indisponibilidade de um shard não necessariamente compromete o sistema inteiro — e possibilita otimizações localizadas, como escolher tecnologias, padrões de armazenamento ou estratégias de indexação mais adequadas para cada fragmento. Essa arquitetura é especialmente útil em sistemas de grande volume de dados ou alta taxa de requisições, onde a distribuição eficiente de carga é essencial. Essa estratégia pode ser combinada com microsserviços de maneira que, enquanto cada serviço é responsável por uma função do domínio da aplicação, cada shard contém réplicas dos serviços para distribuir a carga entre eles, potencializando ao máximo a escalabilidade horizontal.

A Figura 1 ilustra os conceitos definidos. No exemplo, os microsserviços de 0 a M dividem os domínios e formam a aplicação como um todo. A comunicação entre os serviços é ilustrada de maneira síncrona com a chamada HTTP entre os serviços 0 e 1 e de maneira assíncrona via a produção e consumo de mensagens usando um *Broker Kafka* como intermediário. Além disso, é notável que essa lógica é replicada em N shards, isto é, a aplicação como um todo é repetida N vezes para distribuir os fluxos dos usuários, sendo que cada shard fica responsável por uma parte da demanda. A política de particionamento neste caso de exemplo é a divisão justa entre todos shards, mas este fator depende da

[1] APACHE KAFKA. *Apache Kafka Documentation. Disponível em: https://kafka.apache.org/ . Acesso em: 06 dez. 2025.*

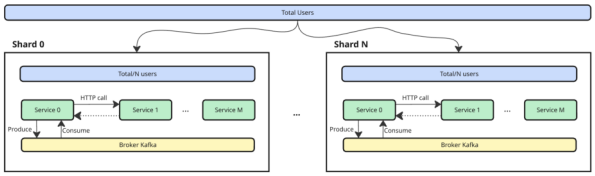

Figura 1 – Exemplo de Arquitetura de Microsserviços com Sharding e Comunicação Síncrona e Assíncrona

aplicação, empresa, dentre outros fatores.

2.1.2 Métricas associadas a ambiente de produção

Em ambiente de produção, pode-se usar de *logs* produzidos para obtenção de diferentes métricas do serviço em tempo de execução. Dentre eles, três podem ser destacados como indicadores claros de bom funcionamento do serviço: quantidade de *lag kafka*, tempo de latência em requisições HTTP e uso de CPU. O primeiro, *lag kafka*, é usado em serviços que contenham comunicação assíncrona para contabilizar o número de mensagens que o serviço tem pendente a processar, sendo tal contabilização feita em cada tópico de consumo no momento da coleta do dado. Tempo de latência de HTTP, por sua vez, mede quanto tempo, em segundos, cada endpoint disponibilizado pelo serviço demora para responder ao cliente. Por fim, uso de CPU mede qual o percentual de uso de cada instância de execução no momento da coleta, sendo o número de instâncias ativas variável e, por si só, outro indicador de desempenho do microsserviço.

2.1.3 Anomalias e Crashes e sua relação com anomalias de variável e instância

No contexto de métricas de execução do serviço, pode-se entender que existe um limite aceitável para o comportamento de tais métricas, isto é, comportamentos que não afetam demasiadamente a aplicação na qual o serviço está inserido como um componente. Pode-se entender, por exemplo, que a latência de resposta de um endpoint crítico à aplicação não pode ser superior a um determinado limite superior, de forma que, se superar tal valor, prejudicará fluxos do sistema. Como exemplos, pode-se pensar no processamento de requisições de autenticação, a consulta a informações essenciais para a continuidade de uma transação, ou ainda a comunicação entre serviços responsáveis por atualizações em tempo real. Em tais cenários, latências elevadas podem provocar retentativas excessivas, *timeouts*, bloqueio de usuários durante operações importantes ou mesmo degradação na experiência geral da aplicação. Para fins deste trabalho, tais comportamentos das métricas fora do limite aceitável serão chamados de anomalias.

Nesse sentido, de maneira mais ampla, é comum que problemas críticos, que afetam a funcionalidade da aplicação, a qualidade do serviço e a experiência do usuário, causem

um ou, mais comumente, diversos comportamentos anômalos nas métricas dos serviços. Momentos como estes são geralmente chamados de *crashes* e demandem atuação de diversas pessoas para rápida mitigação do problema e reestabelecimento do comportamento normal do sistema. Alguns exemplos possíveis de problemas e os comportamentos anômalos relacionados observados nas métricas dos serviços são: mau provisionamento ou baixa eficiência no algoritmo de escalonamento de instâncias de execução causando alto uso de CPU e alta latência em *endpoints* HTTP; indisponibilidade ou alta latência em algum endpoint externo usado pelo serviço observado em algum fluxo ou algoritmos e buscas em banco de dados ineficientes causando alta latência em endpoints específicos; mau provisionamento e configuração de escalonamento de instâncias, ineficiênia algoritmica ou de leitura em banco de dados ou erros em mensagens específicas bloqueando o consumo causando acúmulo de mensagens em lag e possivelmente alto uso de CPU.

Neste contexto, as anomalias de variável referem-se a desvios significativos observados em uma ou mais variáveis específicas, independentemente do comportamento global da instância à qual pertencem. Nesse caso, uma métrica individual assume valores inesperados ou incoerentes em relação ao seu padrão histórico ou estatístico, como picos abruptos, quedas acentuadas ou variações fora dos limites esperados. Para fins deste trabalho, especificamente, definiu-se que anomalias deste tipo se relacionam ao termo e a definição de anomalias feita acima. Por outro lado, as anomalias de instância dizem respeito ao comportamento anômalo do conjunto de variáveis de uma observação como um todo. Uma instância é considerada anômala quando a combinação entre suas variáveis foge dos padrões normais aprendidos pelo modelo, mesmo que, isoladamente, cada variável apresente valores dentro de faixas aceitáveis. Esse tipo de anomalia está associado a relações atípicas entre atributos, sendo mais difícil de detectar por métodos univariados. Neste trabalho, esse tipo de instância é referido como crash.

2.1.4 Alarmes

Com o objetivo de obter observabilidade assertiva do comportamento do serviço e rápida detecção e atuação em caso de *crashes*, uma prática comum é a definição de alarmes que notificam pessoas responsáveis em caso de anomalias de métricas escolhidas que possam indicar comportamento prejudicial à aplicação. Tais alarmes são definidos, geralmente, usando um valor limite, como por exemplo, o máximo aceitável de lag kafka em um tópico específico, bem como uma janela de tolerância, de modo que o alarme é ativado caso o valor observado seja maior que o limite por um intervalo de tempo maior que a janela de tolerância. A dificuldade associada a essa abordagem é a definição de qual limiar usar, bem como qual janela de tolerância. Isto se explica por vários motivos. Um deles é a grande quantidade de variáveis observáveis que se relacionam entre si de maneira não óbvia ou facilmente detectáveis na maioria do tempo. Além de haver diferentes

métricas observáveis se relacionando, há ainda variabilidade dentro das próprias métricas, considerando a existência de múltiplos endpoints, tópicos de consumo e até mesmo réplicas do sistema em arquitetura de sharding, como é o caso da arquitetura simulada neste trabalho. Ademais, existe ainda padrões temporais dificilmente capturados pela percepção humana, considerando a possibilidade de necessidade de observação de grandes intervalos de tempo para compreensão dos mesmos. Por fim, há também a presença de ruído estocástico e picos nas métricas que são considerados normais, isto é, que embora sejam maiores que o comportamento normal da aplicação, são curtos e não afetam a aplicação significativamente, podendo representar, por exemplo, um estado de transição do sistema quando o mesmo está escalando o número de instâncias ativas.

Os fatores descritos acima podem explicar o porquê, ao definir limiares e janelas de tolerância manualmente a partir de observação dos dados ou de métodos estatísticos muito simples, é altamente comum que tais alarmes fiquem muito sensíveis, gerando falsos positivos, causando interrupção do trabalho de pessoas em suas tarefas previstas para verificar problemas inexistentes. Por outro lado, pode acontecer que, para evitar tal problema, limiares e janelas temporais muito rígidas sejam definidos, causando o efeito contrário: o escondimento de problemas reais.

2.1.5 Séries temporais

Segundo Chatfield (2003), uma série temporal é uma coleção de observações feita sequencialmente ao longo do tempo. O conceito é definido pois, neste trabalho, os dados das métricas observadas formam séries temporais discretas – suas observações são feitas apenas em instantes específicos do tempo. Definir os dados como tal é importante pois, ainda segundo o autor, a principal característica de interesse das séries temporais é que observações sucessivas geralmente não são independentes, de modo que análises feitas devem considerar a ordem dos dados e, como resultado, observações futuras podem ser previstas com base em observações do passado. Tal predição não é exata nos dados deste trabalho, motivo pelo qual as séries temporais são ditas estocásticas.

2.1.6 Pré-processamento de dados

Antes da aplicação de dados existentes em qualquer análise, principalmente em algoritmos de Aprendizado de Máquina, faz-se necessário uma etapa prévia em que os dados são preparados e enriquecidos para tal, constituindo duas fases iniciais que Silva (2011) define como Preparação dos Dados – na qual dados brutos são tratados para formatos viáveis ao uso – e Engenharia de Atributos – quando características são extraídas dos dados. Segundo García, Luengo e Herrera (2015), bases de dados reais são altamente influenciadas negativamente por fatores como ruído, ausência ou inconsistência de dados, dados inúteis em dimensões exarcebadas, dentre outros, causando dados de baixa qualidade

que causam análises de baixa qualidade. Logo, esta etapa de pré-processamento é crucial para o aumento da qualidade do resultado final obtido e frequentemente constitui a parte que mais demanda tempo em projetos de Aprendizado de Máquina, podendo consumir até 80% do tempo total de projeto, segundo uma pesquisa com cientistas de dados feita pela revista Forbes[2].

2.1.7 Aprendizado Profundo e Redes Neurais Convolucionais (CNN)

Segundo LeCun, Bengio, Hinton (2015), Aprendizado de Representações é um conjunto de métodos que permitem uma máquina descobrir automaticamente representações necessárias para detectar ou classifcar quando alimentada por dados. Método de Aprendizado Profundo, por sua vez, são Aprendizados de Representação, mas com múltiplas camadas de representação, o que se obtém compondo módulos simples não lineares, responsáveis pela representação em cada camada, desde a entrada do dado até a saída do resultado, permitindo amplo aprendizado de padrões cada vez mais complexos em cada camada mais profunda. Dessa forma, ao se utilizar de transformações o suficiente, pode-se aprender padrões altamente complexos. Tais camadas são chamadas de Redes Neurais e são projetadas com base no cérebro humano. Ainda segundo os autores, as Redes Neurais provaram-se eficientes na descoberta de estruturas complexas em dados multi-dimensionais, como é o caso dos dados deste trabalho.

Dentro de Aprendizado Profundo, Redes Neurais Convolucionais (CNN) formam uma categoria de modelo de aprendizado amplamente utilizado. Yamashita e outros (2018) definem que CNN é um tipo de modelo de Aprendizado Profundo para processamento de dados em formato de grade, como sinais temporais ou imagens, e é construído para aprender automaticamente padrões de baixo e alto nível de complexidade nesses dados, sendo capaz de compreender padrões especiais nos mesmos. No artigo, os autores explicam que o algoritmo tem por chave principal uma camada de convolução, composta por filtros com operações matemáticas, como a convolução, uma operação linear especializada. Esses filtros, aplicados sobre todos os dados camada a camada, são capazes de extrair características simples dos dados em uma camada e alimentar a próxima, que fica responsável então por extrair informações mais complexas e alimentar uma próxima camada, até que a camada de saída seja alcançada.

No contexto deste trabalho, com aplicação de séries temporais, a CNN de 1 dimensão (1D) temporal tem por vantagens uma simplicidade de definição e uso e uma maior eficiência em relação a outros modelos, demandando menos recursos computacionais, devido a paralelização que acontece em seu treinamento. Porém, sua aplicação tem algumas limitações: sua arquitetura gera excelente desempenho na extração de padrões locais, mas o

[2] Disponível em <https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/#2ea5ca046f63>

desempenho é menor quando aplicada a séries temporais longas. Em particular, o tamanho finito dos filtros convolucionais e das janelas de convolução faz com que o modelo tenha dificuldade em capturar dependências temporais de longo prazo. O aumento de camadas para tentar suprir esse problema pode aumentar o custo computacional e ainda não resolver de maneira esse problema. Por isso, diante dessas limitações, surgiram evoluções naturais voltadas especificamente ao modelamento de dependências sequenciais, como as Redes Neurais Recorrentes (RNN) e, principalmente, suas variantes mais robustas, como Long Short-Term Memory (LSTM) e Gated Recurrent Unit (GRU). Esses modelos incorporam mecanismos de memória e portas de controle, permitindo reter ou descartar informações ao longo do tempo, o que os torna mais adequados para séries temporais extensas e com dependências de longo alcance.

2.1.8 Avaliação de resultados em modelos de Aprendizado de Máquina

Na avaliação de resultados em modelos de Aprendizado de Máquina, especialmente em problemas de classificação, utilizam-se métricas estatísticas que permitem analisar de forma quantitativa o desempenho do modelo. Entre as principais métricas destacam-se a precisão (precision), o recall e o F1-score, todas derivadas da matriz de confusão, que organiza as predições do modelo em quatro categorias: verdadeiros positivos, falsos positivos, verdadeiros negativos e falsos negativos. A precisão indica a proporção de predições corretas de uma classe em relação ao total de predições daquela classe realizadas pelo modelo, refletindo o grau de confiabilidade das classificações feitas. O recall, por sua vez, mede a capacidade do modelo em identificar corretamente todas as instâncias de uma classe existentes no conjunto de dados, sendo especialmente relevante em cenários onde a não detecção de um evento é crítica. O F1-score consiste na média harmônica entre precisão e recall, fornecendo uma métrica única que equilibra ambos os aspectos e é particularmente útil quando há desbalanceamento entre classes. A matriz de confusão, portanto, serve como base para o cálculo dessas métricas e como ferramenta analítica fundamental para compreender os tipos de erro cometidos pelo modelo e orientar ajustes e comparações entre diferentes abordagens.

2.2 Trabalhos Relacionados

Em um estudo que focou no pré-processamento de dados para aplicação em modelos de Aprendizado de Máquina Supervisionado, Batista (2003) destaca a importância e o tempo gasto em algoritmos de pré-processamento antes da aplicação em modelos de Aprendizado de Máquina. Por isso, seu trabalho contribui com a criação de um framework de desenvolvimento e avaliação de métodos de tratamento de dados comuns e automatizáveis, tendo por resultado métodos automáticos para tratamento de valores desconhecidos e conjuntos de dados com classes desbalanceadas.

Dentro do escopo de monitoramento de ambientes de microsserviços, Ying et al. (2020) defendem que a observabilidade desempenha um papel fundamental nesse tipo de arquitetura e pode ser realizada em diferentes níveis, como hardware, rede, sistema e aplicação. Os autores propõem ainda um esquema de monitoramento baseado em três plataformas: Logstash, responsável pela coleta e filtragem dos dados; Elasticsearch, utilizado para armazenamento e análise; e Kibana, empregado para visualização das informações de monitoramento.

Sobre séries temporais e detecção de anomalias de maneira mais genérica, isto é, fora do contexto de arquitetura de microsserviços, Choi et al. (2021) realizam uma revisão dos principais modelos de Aprendizado Profundo e os avaliam e comparam utilizando diferentes conjuntos de dados reais, como o WADI, que representa dados de uma rede de distribuição de água sujeita a falhas e ataques, e o MSL, composto por séries temporais de telemetria do rover Curiosity usadas para identificar anomalias em sistemas espaciais. Como resultado, obtiveram-se evidências suficientemente satisfatórias em mais de um modelo para concluir que técnicas baseadas em Aprendizado Profundo apresentam impacto significativo na detecção de anomalias em dados com padrões complexos, como séries temporais. Sobre CNN, especificamente, os autores destacam que essas arquiteturas demonstram desempenho particularmente robusto quando as anomalias se manifestam como padrões locais de curta duração, e que são capazes de processar janelas de dados de forma altamente paralelizada, tornando-as adequadas para aplicações com grandes volumes de informação. Ademais, observam que CNNs alcançam resultados competitivos quando o sinal temporal é transformado em representações bidimensionais, como matrizes de recorrência ou espectrogramas, permitindo a captura eficiente de relações estruturais entre variáveis. Por fim, os autores apontam que modelos híbridos que combinam convoluções com mecanismos temporais, como os ConvLSTMs, tendem a superar abordagens puramente convolucionais, ao incorporar simultaneamente dependências locais e de longo prazo, ampliando ainda mais sua aplicabilidade em cenários reais.

Em um estudo que aborda detecção de anomalias em ambientes de microsserviços, Nobre, Pires e Reis (2023) investigam o uso de um modelo de aprendizado de máquina supervisionado, especificamente um Multi-Layer Perceptron (MLP), para detecção de anomalias em arquiteturas de microsserviços. Para obtenção dos dados a aplicar no modelo, os autores desenvolvem uma infraestrutura de microsserviços própria, incluindo um módulo de injeção de falhas que simula anomalias, além da construção de um conjunto de dados de monitoramento. O modelo MLP é então treinado e avaliado para identificar essas anomalias. Os resultados mostram um bom desempenho, especialmente para anomalias em nível de serviço, como tempo de resposta do mesmo. O modelo apresentou métricas superiores em acurácia, precisão, revocação e F1-score. O estudo destaca ainda o potencial desse tipo de abordagem para automatizar o monitoramento e a gestão de sistemas distribuídos, reforçando a eficácia de métodos supervisionados na detecção de anomalias.

Em outro trabalho de mesmo objetivo, Hrusto, Engström e Runeson (2022) propõem uma solução de detecção baseada em métodos de Aprendizado de Máquina profundo aplicada a métricas multivariadas coletadas do monitoramento do sistema. Os autores destacam o desafio de trabalhar com séries temporais de alta dimensionalidade e, principalmente, com o desbalanceamento entre dados anômalos e não-anômalos e a ausência de verdade absoluta nos rótulos de anomalias, considerando que pode haver anomalias mascaradas na base de dados. Para mitigar os problemas apresentados, o trabalho introduz um ciclo de realimentação contínua entre operações e desenvolvimento: anomalias detectadas pelo modelo são repassadas aos desenvolvedores, que fornecem rótulos utilizados posteriormente para otimização iterativa do modelo. Além disso, o estudo apresenta diretrizes para seleção de métodos de *deep learning* aplicáveis a séries temporais multivariadas e um plano de implantação em nuvem para avaliar e atualizar continuamente o detector. A abordagem reforça o potencial de integração entre monitoramento, DevOps e técnicas de *deep learning* para aprimorar a detecção de falhas em arquiteturas distribuídas.

Em outro estudo semelhante, mas com abordagem diferente, Xu et al (2018) usam de algoritmos de Aprendizado de Máquina não supervisionado para detecção de anomalias. No estudo, os autores destacam a importância de monitorar indicadores de perfomance chave, como acessos de páginas e números de usuários onlines, em grandes aplicações para detecção e mitigação rápida de anomalias. Neste estudo, dados de uma empresa de internet global são usados e tratados como séries temporais, considerando que as métricas usadas também variam com padrões ao longo do tempo. Além disso, o trabalho destaca a dificuldade de se obter dados de tais métricas rotulados e, por isso, utiliza de um algoritmo não supervisionado de Aprendizado de Máquina nomeado Donut e baseado em *Autoencoder* Variacional. Como resultado, o modelo tem melhor desempenho que alguns modelos supervisionados e tem seu melhor F-score variando entre 0.75 e 0.9.

3 Metologia

Este Capítulo apresenta os principais passos realizados neste trabalho, sendo cada passo organizado em uma Seção entre as Seções 3.1 e 3.4.

3.1 Observação de dados reais e simulação de dados para obtenção de uma base de dados

Com objetivo de obter um conjunto de dados para aplicar os algoritmos de pré-processamento, foi necessário a criação de um conjunto de dados brutos. Para isso, o primeiro passo foi a observação de uma base de dados reais a qual não se teve permissão de uso direto ou citação, mas sim de observação e análise exploratória após contato com uma empresa. A organização em questão, que não será identificada, mantém uma aplicação real com arquitetura de microsserviços e sharding e os dados observados foram dados brutos de métricas de um desses serviços, sendo ele um de alto impacto na aplicação. Tais dados se encontram espalhados em diversas tabelas, uma para cada métrica, das quais se escolheu três para observação e reprodução: uma do tempo de resposta por *endpoint* HTTP, outra para a taxa de uso de CPU por instâncias ativas e outra com a quantidade de mensagens em fila para consumo em um sistema de mensageria *Kafka*.

A partir de observação visual e uma análise exploratória nesta base de dados, foram desenvolvidos diversos algoritmos que aplicam métodos arbitrários, aleatórios e configuráveis para geração de dados sintéticos verossímeis. É importante ressaltar que, com esta decisão, era sabido que parte da fidelidade de padrões temporais e de relações entre variáveis reais seria perdida, mas o principal objetivo – que era a obtenção de dados brutos em formatos semelhantes, com distribuições de probabilidade, amplitudes e atributos estatísticos diferentes mas possíveis de se encontrar em aplicações reais – seria preservado.

Para construção dos algoritmos, utilizou-se, para cada métrica, uma distribuição de probabilidade específica para obtenção de dados numéricos que se assemelhasse minimamente ao comportamento observado na base de dados reais. Dessa forma, decidiu-se que usaria-se da distribuição log-normal para latência HTTP e gamma tanto para uso de CPU quanto para *lag* de mensageria *Kafka*. Além disso, houveram etapas manuais em que, aleatoriamente, introduziu-se ruído nos dados obtidos. Vale mencionar também que os algoritmos desenvolvidos nesta etapa preservaram problemas reais observados nos dados reais, como descentralização das métricas obtidas em várias tabelas, ausência de dados e diferentes frequências de coletas entre métricas diferentes e dentro das próprias métricas, que contém discrepâncias de até um minuto entre o instante de uma coleta e o instante

posterior, por exemplo.

Por fim, cabe mencionar que nesta etapa inicial apenas os dados considerados como comportamento "normal" foram inseridos, de forma que dados que representam problemas na aplicação foram inseridos depois, por facilidade e convenção.

3.2 Desenvolvimento e aplicação de algoritmos de tratamento e enriquecimento de dados

Nesta etapa observou-se os dados brutos de entrada métrica a métrica – cada um em uma tabela separada – e desenvolveu-se um *pipeline* formado de diversas funções para tratar problema a problema observado, com a preocupação de desenvolver funções genéricas que possam resolver questões em comum entre diferentes métricas. Outro foco da etapa foi o enriquecimento informacional da tabela de uso de CPU.

Neste sentido, alguns dos problemas notados e que precisaram de desenvolvimento de funções para solução foram:

- Informações temporais inúteis: informações de segundos e milissegundos na coluna que marca o instante de coleta dos dados não são necessários e dificultam a visualização. Logo, decidiu-se fazer algoritmos que removessem tais partes e deixasse apenas ano, mês, dia, hora e segundo;

- Tabelas em formato longo: no processo de coleta de dados, para o mesmo instante de coleta teórico, eram feitas diversas coletas reais, uma para cada combinação de shard com endpoint ou tópico nos casos de latência HTTP e lag *Kafka* e uma para cada instância ativa na tabela de CPU. Logo, a base de dados continha o mesmo instante de coleta, com baixa diferença nos segundos, em várias linhas, de forma que as tabelas se encontravam em formato longo. Para os objetivos deste trabalho – a aplicação em algoritmos de Aprendizado de Máquina -, é necessário que tais dados estejam em formato largo, de forma que cada linha seja apenas um instante de tempo e hajam várias colunas, uma para cada combinação possível de shard com endpoint e tópico e uma para uso de CPU. Para tal problema, desenvolveu-se funções de transformação de formato largo em longo respeitando as particularidades do formato de cada tabela.

- Diferença temporal entre instantes da mesma coleta: considerando as diversas coletas para o mesmo instante de coleta teórico, já explicada, haviam dados que pertenciam ao mesmo instante teórico de coleta mas que, na prática, tinham diferença de um minuto em seus instantes reais de coleta. Para transformação em formato largo, isso deve ser corrigido para que se possa agrupar dados do mesmo instante sob um mesmo

instante de coleta. A solução escolhida foi percorrer todas tabelas e padronizar os minutos usando do maior minuto entre todas coletas do mesmo instante;

- Saltos temporais aleatórios em tabela de *lag Kafka*: na base de dados reais, apenas é persistido no banco de dados o valor da coleta se, naquele instante, existe pelo menos uma mensagem em fila de consumo, causando saltos temporais quando não há mensagens a consumir na fila. Escolheu-se tratar isso fazendo um algoritmo que percorre a tabela do começo ao fim e, ao notar salto temporal, preenche um instante temporal com zero;

- Diferente frequência de coleta para cada métrica: nas bases de entrada, a latência HTTP tem frequência de coleta de dez minutos, o uso de CPU 15 minutos e *lag kafka* 30 minutos. Para união das tabelas em uma única de formato largo, é necessário unir os dados sob a mesma base temporal. Para isso, definiu-se como base de tempo a menor granularidade e diferentes estratégias foram usadas para imputação nos instantes de tempo não existentes nas tabelas de uso de CPU e *lag* de *Kafka*: Para o primeiro, considerando a existência de picos locais e breves, escolheu-se a estratégia de preencher com o valor mais próximo. Para *lag Kafka*, considerando picos mais longos, os valores foram preenchidos com o uso de interpolação temporal;

- Dados faltantes: em alguns instantes de tempo não haviam coletas para as métrica. Tal problema foi indiretamente resolvido ao unificar a base temporal das tabelas;

- Frequência entre coletas diferente da frequência teórica: embora haja uma frequência de coleta teórica para cada métrica, exposta acima, na base de dados a diferença instantes de coleta podiam variar em alguns minutos a mais ou a menos. A padronização da base temporal com imputação de dados resolveu este problema;

- Tabela de uso de CPU informacionalmente pobre: havia apenas uma linha com o máximo de uso de CPU por instância ativa e seu instante de tempo. Para trasformar em formato longo, deveria-se ecsolher qual valor usar. Ao invés de escolher um único valor, desenvolveu-se uma função que, ao iterar sob a tabela, introduziu três colunas novas: máximo uso de CPU entre os três valores possíveis, média do uso máximo de CPU e o número de instâncias ativas por instante de tempo de coleta.

- Descentralização: as métricas continham cada uma sua tabela com dados brutos. Após tratamento de cada uma individualmente, foi atestada a unificação dos dados em uma tabela única para uso da mesma como entrada no algoritmo de Aprendizado Profundo.

3.3 Inserção de crashes simulados

Uma vez com uma tabela centralizada e com dados tratados e enriquecidos que representassem o comportamento normal da aplicação, o próximo passo foi a inserção de problemas reais – neste trabalho chamados de *crashes* – simulados. A inserção a posteriori foi decidida com objetivo de facilitar a inserção de tais valores de maneira sequencial no dataset final, já em formato largo e enriquecido. Usando desta estratégia, foi possível gerar um grande dataset inicial de comportamento considerado normal, como descrito na Seção 2.1, para posteriormente, nesta fase, iterar sobre os valores sobreescrevendo, apenas nos instantes desejados escolhidos como crash, com valores anormais.

Para criação dos crashes, usou-se de conhecimento indiretamente adquirido com base na observação da base de dados reais para arquitetar, conceitualmente, três diferentes tipos de comportamento anômalos nas métricas dos serviços. Isso foi feito com base em possíveis problemas que podem afetar aplicações baseadas em arquitetura de microsserviços. Considerando que este trabalho foca na detecção de anomalias com base no comportamento das métricas, de forma que a detecção da causa do mesmo seria o passo seguinte e responsabilidade da pessoa alertada, cita-se os problemas abaixo com foco principal nas métricas afetadas, de forma que apenas uma breve explicação de algunas causas possíveis é feita, mas com ressalva de que podem haver outras causas para tais comportamentos das métricas. Podem haver, também, outros problemas que podem causar outros comportamentos anômalos nas métricas, mas não considerados por este estudo. Neste trabalho, os comportamento anômalos considerados e suas possíveis causas são:

- Métricas de elevado uso de CPU e alta latência em um ou mais endpoints: Pode indicar insuficiente provisionamento de recursos para o serviço se o serviço já está com alto número de instâncias ativas para atender a demanda. Caso haja um baixo número de instâncias, pode indicar má configuração do algoritmo de escalonamento de instâncias. De qualquer forma, costuma ser evidenciado em momentos de maior demanda e afeta diversos fluxos, indicando problemas na saúde do serviço.

- Métricas indicando elevado *lag* de mensagens *Kafka* em um ou mais tópicos: Pode ser causado por mau provisionamento de recursos em relação à demanda do serviço, lentidão no processamento das mensagens, indicando ineficiência de código ou leitura do banco de dados, ou problemas no escalonamento. Comumente, quando lidando com escalonamento horizontal de instâncias, pode acontecer um processo de má configuração do gatilho de escalonamento por taxa de uso da CPU que pode afetar o consumo das mensagens. Isto ocorre quando o limiar que ativa o escalonamento é definido próximo do uso médio de CPU de forma que, em um instante o uso de CPU supera o limiar normal e as instâncias escalam, fazendo com que poucos instantes depois o uso fique abaixo do limiar, fazendo com que instâncias sejam inativadas.

O problema é que, cada vez que instâncias são criadas ou destruídas, acontece um rebalanceamento das mensagens a consumir entre as instâncias, de forma que o consumo é parado. Quando tal processo acontece de maneira cíclica, o serviço pode interromper o consumo de mensagens por horas. Tal comportamento foi reproduzido neste trabalho.

- Métricas indicando elevada latência em *endpoint* crítico: Considerando que o problema é observado em apenas um endpoint, pode ser causado por ineficiência interna no mesmo, como algoritmo de alta latência no processamento que pode ser paralelizado ou refatorado, buscas lentas em banco de dados causando baixa taxa de *I/O*, ou dependência externa de algum outro endpoint externo lento ou que passa por problemas, gerando um efeito em cascata.

Com base em tais definições de crashes, manual e arbitrariamente escolheu-se instantes de inserção dos mesmos entre os dados já gerados, com o cuidado de evitar padrões temporais entre um crash e momentos de normalidade na aplicação dentro dos dados, para que o algoritmo não aprenda padrões inexistentes na intercalação de dias normais e com ocorrências de anomalias. No momento de inserção de cada *crash*, então, construiu-se algoritmos que sorteiam a duração do mesmo – podendo variar de 7 a 21 horas, arbitrariamente – e depois inserem dados simulados respeitando os comportamentos anômalos previamente definidos, respeitando a seguinte estratégia:

- Para crashes que envolvam alta latência em um único endpoint, aleatoriamente definiu-se quais shards seriam afetados, considerando que o comportamento pode ou não estar presente em mais de um shard. Posteriormente, usando o ínicio do crash definido e sua duração sorteada, iterou-se sobre os dados normais previamente gerados e sorteou-se, para cada instante dentro da janela de duração do crash, um multiplicador que varia arbitrariamente entre 3 e 5. Neste instante, então, a estratégia arbitrária de inserção de crash escolhida foi sobreescrever o valor normal pela soma entre ele e a multiplicação dele pelo multiplicador aleatório.
- Para crashes que envolvam alto uso de CPU e latência, foi arbitrariamente definido que multiplicaria-se as métricas de máximo uso de CPU e média do uso máximo de CPU pelo triplo delas, enquanto que para latência o mesmo processo já explicado seria realizado, isto é, somar o valor normal com a multiplicação dele por um multiplicador aleatório entre 3 e 5. Tal processo foi feito para cada instante dentro de um mesmo crash. Além disso, entre a geração de um crash e outro deste tipo, foi sorteada, com a mesma probabilidade de ocorrer, se o número de instâncias ativas de execução seria sorteado próximo do número máximo de instâncias possíveis ou próximo do número mínimo. Tal feito, além de trazer variabilidade aos dados de crashes gerados para

desafiar o modelo, reproduz cenários possíveis e observados em bases reais, em que o provisionamento de recursos pode ser ineficiente, indicando que mesmo com alta escabilidade horizontal o uso de CPU está alto, ou o contrário, que mesmo com alto uso de CPU o número de instâncias está baixo, indicando possibilidade de melhoria na política de escalonamento.

- Para crashes que envolvam alto *lag Kafka*, foi definido que seriam inseridos crashes com o comportamento de alto número de mensagens pendentes para consumo para um determinado tópico sorteado a cada crash e em um ou mais shards, o que também é decidido aleatoriamente. Além disso, decidiu-se adicionar em tais crashes o comportamento de oscilação do uso de CPU e o do número de instâncias ativas, considerando a má configuração do algoritmo de escalonamento horizonal, como explicado anteriormente. Tal definição se deve à alta frequência de ocorrência desse tipo de crash nos dados reais observados. Para combinar tais efeitos em um crash, desenvolveu-se um algoritmo arbitrário em que, no primeiro momento de crash é feito a multiplicação dos indicadores de CPU desse passo por 3, enquanto que o *lag* de *kafka* é somado a ele mesmo multiplicado por um multiplicador aleatório entre 3 e 5. Depois disso, itera-se sobre a duração do crash e em cada passo de inserção de anomalia, é feito um sorteio do número de instâncias ativas dentro do intervalo de 4 a 6 instâncias. Se o número de instâncias do passo atual for maior que o passo anterior, o lag atual assume o resultado da subtração entre o valor do passo anterior e ele mesmo multiplicado por uma taxa de variação, definida aleatoriamente em cada passo entre 10% e 20%, enquanto que os indicadores de CPU assumem um valor aleatório entre 5% e o próprio valor do passo anterior. Isso é feito seguindo a lógica que, se o número de instâncias aumentou, o uso de CPU e o *lag kafka* deve diminuir levemente. Ao contrário, se o número de instâncias diminui, o lag e o uso de CPU aumentam levemente, então o processo inverso é feito: com as mesmas taxas é feito a soma dos valores anterior de lag e CPU pela taxa de variação. Por fim, se o número de instâncias é o igual ao passo anterior, os mesmos indicadores de CPU e lag são mantidos. Pela frequência de ocorrência na base de dados reais, este crash teve maior número de ocorrências em relação aos outros.

Vale mencionar que durante esta etapa a coluna de rótulo foi adicionada. Antes da execução dos algoritmos de geração de crashes, inseriu-se uma coluna com todos valores False chamada ”is_anomaly“ em todas linhas. Depois, durante cada momento de crash inserido, sobreescreveu-se a linha em questão com True.

3.4 Aplicação em modelo CNN 1D

Após obtenção do conjunto de dados pré-processados, rotulados e com crashes inseridos, procedeu-se à aplicação de um modelo de aprendizado profundo baseado em uma rede neural convolucional unidimensional (CNN 1D) com foco na detecção temporal de anomalias. O modelo foi escolhido por ter simplicidade e custo computacional menor em relação a outros modelos, sendo visto como uma boa escolha para uma versão inicial de sistema de detecção de anomalias.

Esta etapa teve por principal objetivo testar a viabilidade de aplicação dos dados processados obtidos em etapas anteriores em um algoritmo de Aprendizado Profundo para detecção de anomalias, ou seja, se havia compatibilidade entre os dados esperados por um algoritmo deste e os dados obtidos e se, com uso deles, haveria um treinamento funcional e com potencial de gerar resultados. Vale ressaltar, porém, que as definições necessárias de tamanhos de janela, número de épocas de treinamento do modelo, suas camadas, dentre outras definições, não formam o principal objetivo deste trabalho e, portanto, são escolhas experimentais e arbitrárias definidas com base em leitura de documentação e sugestão de IA generativa, sem maior refinamento ou avaliação. Em resumo, o foco está na validação dos dados obtidos e no potencial da abordagem de todo projeto e não na otimização fina do modelo e seu desempenho, que pode-se entender como próxima etapa em um trabalho futuro.

Inicialmente, foi criada uma função que divide o conjunto de dados inteiro em sequências de um tamanho de janela definido como 128. Tal configuração representa o número de passos consecutivos que serão usados de entrada no modelo em cada etapa e sua definição escolhida representa cerca de 21 horas de dados, podendo capturar padrões de um dia quase completo.

Na próxima etapa, dividu-se os dados em dados de treino e validação, com taxa de 30% para treino, e usou-se do método *StandardScaler*, que retorna uma distribuição de média 0 e desvio-padrão 1, para normalização dos dados de treinamento e avaliação – separados para evitar vazamento de dados, em que dados de avaliação são usadas na normalização de dados de treinamento, enviesando o modelo. Com os dados normalizados, usou-se da função desenvolvida para dividir os dados de treinamento e validação em sequências de janelas de tamanho de 128 passos.

O passo seguinte foi a definição do modelo. Novamente, sua definição e refinamento não compõem o foco do trabalho e portanto, um modelo inicial experimental obtido com base em documentações simples e suporte de IA generativa teve as seguintes características:

- Camada de convolução 1D *Conv1D* inicial com 64 filtros, kernel de tamanho 3 e ativação ReLU, recebendo como entrada as janelas temporais criadas com o conjunto

de dados e sendo responsável por identificar padrões locais iniciais na sequência temporal usando os filtros.

- Camada de *MaxPooling1D* com fator de redução igual a 2: reduz a dimensionalidade extraindo apenas os valores mais relevantes, diminuindo ruído e custo computacional para passagem à camada seguinte.
- Segunda camada *Conv1D* com 128 filtros e kernel de tamanho 3: aprofunda a extração de padrões, aprendendo características mais complexas da série temporal.
- Camada *Flatten*: apenas para transformar os mapas de características em um vetor unidimensional.
- Camada *Dropout* com taxa de 0,3 compõe uma camada para redução sobreajuste.
- Camada densa intermediária com 64 neurônios e ativação ReLU para combinar todas as características extraídas e aprender relações mais abstratas entre elas.
- Camada densa de saída com 1 neurônio e ativação sigmoide para classificação binária.
- Modelo compilado com otimizador Adam e função de perda *binary crossentropy*: o otimizador ajusta os pesos, enquanto a função de perda mede o erro em tarefas de classificação binária.

Com o modelo definido, foi feito o treinamento do mesmo. Para isso, usou-se de 30 épocas e tamanho de batch de 128 janelas, arbitrariamente. Por fim, terminado o treinamento, usou-se dos dados separados anteriormente para avaliação para avaliar o desempenho do modelo. Foi obtido então o valor da função de perda calculada, a acurácia e uma tabela de reporte de classificação, com medidas mais robustas: precisão, recall e f1-score para ambos valores negativos e positivos, bem como a média deles. Todas essas medidas foram escolhidas por representarem, juntas, uma visão global do desempenho do modelo, considerando diferentes aspectos a serem analisados. Para análise visual, também foi criada uma matriz de confusão.

4 Resultados

Este capítulo apresenta os resultados obtidos ao longo de todo estudo, começando pela geração dos dados de comportamento não anômalo similares a dados brutos de aplicações reais, na Seção 2.1, a aplicação dos algoritmos desenvolvidos para tratar os problemas presentes em tais dados, na Seção 2.2, a criação e inserção de anomalias nessa base intermediária, na Seção 2.3 e, por fim, a aplicação da base de dados final em um modelo de Rede Neural CNN 1D.

4.1 Base sintética de dados de comportamento normal

Após construção dos algoritmos de geração de dados aleatórios que respeitem os formatos de dados brutos desejados para servir de entrada ao trabalho, obteve-se três tabelas separadas, cujas características e formatos serão exibibidos nas Seções 4.1.1, 4.1.2 e 4.1.3 para Latência HTTP, *Lag* de mensagens *Kafka* e Uso de CPU, respectivamente. Para todos, definiu-se um ambiente com *sharding* de três shards: s1, s2 e s3, para aumentar a semelhança com a base real de dados obsevada.

4.1.1 Latência HTTP

Conforme mencionado anteriormente, os dados de latência HTTP seguiam uma distribuição log-normal. Com as definições arbitrárias da média como 0.5 segundo e o desvio-padrão como 0.7 segundo, a simulação de tais dados gerou dados entre 0.06 e 30 segundos, sendo este um outlier. A distribuição dos dados sintéticos gerados pode ser vista na Figura-2.

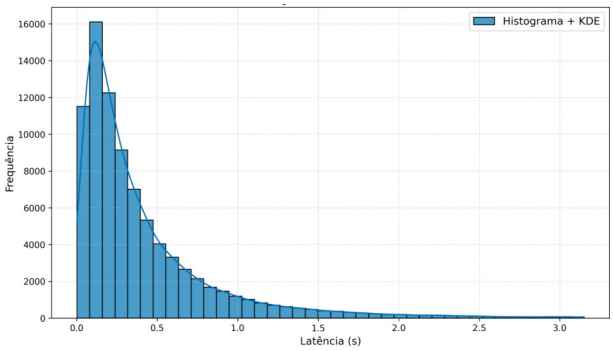

Figura 2 – Histograma e Estimativa de Densidade de Kernel da distribuição dos dados sintéticos de Latência HTTP

Em relação ao formato e as características dos dados, a Tabela 1 exibe um exemplo dos dados obtidos e partes de dois instantes de coleta com um salto de 9 minutos entre

eles. Possíveis de observar no exemplo da Tabela, as principais características dos dados sintéticos gerados são:

- Quatro endpoints fictícios para fins de simulação: '/api/fake-endpoint', '/api/i-am-simulating/endpoint', '/api/admin/delete-tcc' e '/api/wow'. Considerando os três shards, isto faz com que cada instante de coleta tenha doze linhas, compostas do valor da latência dentro de cada combinação endpoint e shard possível.
- Variação de um minuto aleatoriamente presente dentro das doze linhas que compõem um instante de coleta, pois a coleta a partir de um certo momento pode ter ocorrido alguns poucos segundos após o momento da coleta anterior, o que em alguns casos causa a troca do minuto da coleta embora elas façam parte do mesmo instante teórico de coleta.
- Frequência de coleta teórica e observada de 10 minutos, mas coletas geradas podem ser observadas com saltos de tempo dentro do intervalo [8, 10] entre uma coleta e outra, o que foi definido aleatoriamente em cada coleta.
- Ruído estocástico adicionado.

Tabela 1 – Exemplo de dados sintéticos de Latência HTTP

queried-at	**path**	**value**	**shard**
2025-12-14T03:55:34:48Z	/api/fake-endpoint	0.1324	s1
2025-12-14T03:55:34:48Z	/api/i-am-simulating/endpoint	0.2405	s1
2025-12-14T03:55:34:48Z	/api/admin/delete-tcc	0.7922	s1
2025-12-14T03:55:34:48Z	/api/wow	0.5882	s1
2025-12-14T03:56:24:31Z	/api/fake-endpoint	0.2907	s2
2025-12-14T03:56:24:31Z	/api/i-am-simulating/endpoint	0.2425	s2
...	...	...	...
2025-12-14T04:05:15:50Z	/api/fake-endpoint	0.7922	s1
2025-12-14T04:05:15:50Z	/api/i-am-simulating/endpoint	0.5882	s1
...	...	...	...

Fonte: Autor.

4.1.2 Kafka Lag

Para os dados de lag em mensageria, escolheu-se simular os mesmos usando uma distribuição *gamma* e, arbitrariamente, definiu-se uma média 12000 e um desvio padrão 21000. Como resultado, gerou-se uma distruição entre 0 e 234730, que pode ser vista na Figura-3.

As principais características dos dados gerados de *Lag Kafka* são:

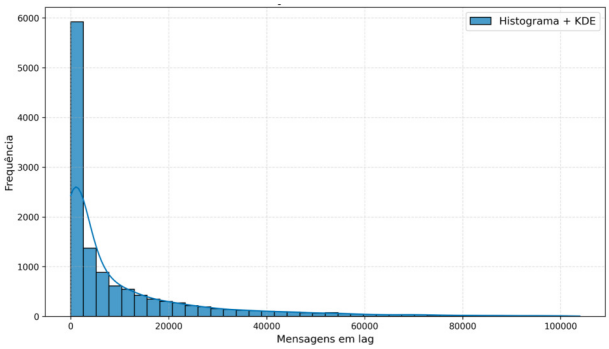

Figura 3 – Histograma e Estimativa de Densidade de Kernel da distribuição dos dados sintéticos de *Lag Kafka*

- Dois tópicos fictícios para fins de simulação: ”IMPORTANT-TOPIC“ e ”POST-TCC-IN-AVA2-UFSCAR“. Considerando os três shards, isto faz com que cada instante de coleta possa ter até seis linhas, compostas do valor do lag observado dentro de cada combinação de tópico e shard possível.

- Saltos temporais entre coletas: replicado da base de dados reais, inseriu-se um comportamento em que, se em determinada sequência de tempo não havia mensagens pendentes pro consumo, tais instantes não foram persistidos na base de dados. Isso causou saltos temporais não previsíveis para combinações de tópicos e shards aleatórios ao longo do conjunto de dados gerados.

- Ruído estocástico adicionado.

Um exemplo do formato dos dados obtidos pode ser visto no exemplo de Tabela 2. Nele, observa-se dois instantes de coletas seguidos, separados por 30 minutos. Nota-se, porém, que no primeiro o tópico ”IMPORTANT-TOPIC“ não tem valor definido no shard s2, enquanto que na segunda coleta o tópico também não tem valor no shard 1. Isso é mostrado na Tabela para exemplificar o comportamento inserido de saltos temporais: na primeira coleta já foi definido anteriormente que a ausência de lag no tópico no shard está acontecendo por algum tempo aleatoriamente definido, de forma que em tais instantes não haverá dados deste tópico e shard. Na segunda coleta o exemplo é a simulação de um instante em que aleatoriamente definiu-se que todo lag foi drenado e, portanto, interrompe-se o persistir de informação para o mesmo tópico, mas em outro shard. Isso exemplifica sobreposição de saltos temporais de tópico e shards que acontecem na base de dados e dificultam a criação de algoritmos, considerando que de uma coleta para outra não existe padrão do número de linhas esperado para transformação da tabela de formato longo em largo.

Tabela 2 – Exemplo de dados sintéticos de Kafka Lag

queried-at	**topic**	**value**	**shard**
2025-10-16T03:30:34:48Z	IMPORTANT-TOPIC	30	s1
2025-10-16T03:30:34:48Z	POST-TCC-IN-AVA2-UFSCAR	2010	s1
2025-10-16T03:30:34:48Z	POST-TCC-IN-AVA2-UFSCAR	12017	s2
2025-10-16T03:30:34:48Z	IMPORTANT-TOPIC	5882	s3
2025-10-16T03:30:34:48Z	POST-TCC-IN-AVA2-UFSCAR	2907	s3
2025-10-16T04:00:09:48Z	POST-TCC-IN-AVA2-UFSCAR	2425	s1
2025-10-16T04:00:09:48Z	POST-TCC-IN-AVA2-UFSCAR	7922	s2
2025-10-16T04:00:09:48Z	IMPORTANT-TOPIC	3190	s3
2025-10-16T04:00:09:48Z	POST-TCC-IN-AVA2-UFSCAR	2003	s3
...	...	...	...

Fonte: Autor.

4.1.3 Uso de CPU

Para dados de uso de CPU, a distribuição gamma foi usada com média 0.2 e desvio-padrão 0.1. Com isso, a distribuição da Figura-4 foi obtida.

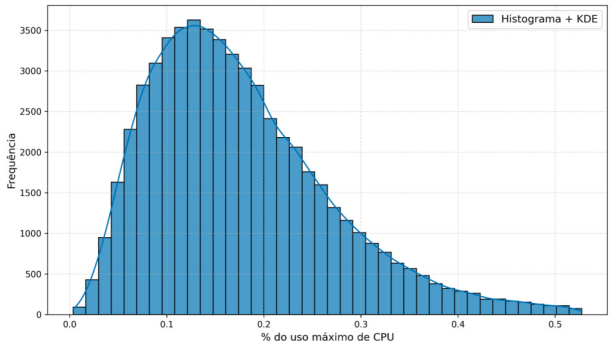

Figura 4 – Histograma e Estimativa de Densidade de Kernel da distribuição dos dados sintéticos de Taxa de Uso Máximo de CPU

As principais características dos dados gerados de taxa de uso de CPU são:

- Número de linhas, que representa o número de instâncias ativas dentro de um mesmo instante de coleta teórico e seu uso de CPU, variando com base em sorteio no intervalo [2, 6]. O sorteio seguiu uma distribuição normal, onde 4 é mais frequente de ocorrer e 2 e 6 são as caudas.
- Variação de um minuto aleatoriamente presente no instante de coleta dentro do número variável de possíveis linhas que uma mesma coleta teórica possui.
- Outliers aleatórios inseridos: sabe-se que em dados de CPU existem algumas coletas com valores acima do normal pois a medição foi feita muito próxima a criação da instância, de forma que sua taxa de uso está elevada. Por isso, inseriu-se em alguns

momentos aleatórios outliers como sendo a soma da taxa de uso de CPU no instante pela multiplicação do desvio padrão e um multiplicador aleatório entre 3 e 5.

- Frequência de coleta teórica e observada de 15 minutos, mas coletas geradas podem ser observadas com saltos de tempo dentro do intervalo [13, 15] entre uma coleta e outra, o que foi definido aleatoriamente em cada coleta.

A Tabela 3 exibe um exemplo de dados de uso de CPU obtidos. Nela são mostrados dois instantes de coleta separados por 13 minutos. No primeiro, o serviço tinha apenas 2 instâncias ativas em todos shards, enquanto na segunda o número aumentou para 4 instâncias apenas no shard s2, indicando escalonamento horizonal em tal shard.

Tabela 3 – Exemplo de dados sintéticos de Uso de CPU

queried-at	**value**	**shard**
2025-09-06T00:00:34:48Z	0.3	s1
2025-10-16T03:00:34:48Z	0.12	s1
2025-10-16T00:00:34:48Z	0.69	s2
2025-10-16T00:00:34:48Z	0.59	s2
2025-10-16T00:00:34:48Z	0.81	s3
2025-10-16T00:00:34:48Z	0.12	s3
2025-10-16T00:13:41:01Z	0.11	s1
2025-10-16T00:13:41:01Z	0.9	s1
2025-10-16T00:13:41:01Z	0.02	s2
2025-10-16T00:13:41:01Z	0.19	s2
2025-10-16T00:13:41:01Z	0.16	s2
2025-10-16T00:13:41:01Z	0.22	s2
2025-10-16T00:13:41:01Z	0.15	s3
2025-10-16T00:13:41:01Z	0.32	s3
...	...	...

Fonte: Autor.

4.2 Aplicação de algoritmos de pré-processamento e enriquecimento de dados

Após aplicação de todas tabelas individualmente nas devidas funções de pré-processamento de dados e redefinição de base temporal, um algoritmo de união foi usado para centralizar todas informações em única tabela de formato largo e de dados tratados. Sobre ela, as principais observações são listadas abaixo:.

- Coluna queried_at com instante de coleta de 10 em 10 minutos padronizado para todas coletas e métricas. Apenas a data, hora e minuto foram mantidos.

- Colunas adicionadas para uso de CPU: máxima taxa de uso de CPU entre instâncias ativas no momento da coleta, média de todas taxas e número de instâncias ativas.
- 7264 pontos de dados amostrados de 10 em 10 minutos, tendo a primeira coleta simulada no dia 2025-10-24 e a última no dia 2025-12-14.
- 27 colunas de entrada ao modelo: 3 para CPU (métricas de uso médio de CPU, máximo uso de CPU e número de instâncias ativas), 4 para latência HTTP (número de endpoints) e 2 para *lag Kafka* (número de tópicos), sendo cada uma dessas replicadas em 3 colunas, considerando o ambiente simulado com 3 shards.

No Quadro 1 é mostrado um exemplo dos dados centralizados obtidos e em formato largo. O Quadro é exibido de maneira transposta apenas por uma questão de espaço.

Quadro 1 – Exemplo de dados pré-processados e unificados

queried-at	2025-10-24T00:00
n-pods-s1	4
max-cpu-usage-s3	0.74
mean-max-cpu-usage-s1	0.43
…	…
/api/admin/delete-tcc-s1	0.23
/api/admin/delete-tcc-s2	0.19
…	…
value-IMPORTANT-TOPIC-s1	2301
value-IMPORTANT-TOPIC-s2	1024
…	…

Fonte: Autor.

4.3 Inserção de dados sintéticos de comportamento anômalo

Após a etapa de geração de crashes sintéticos para inserção direta na base de dados, a tabela final de entrada ao modelo foi obtida. Para isso, foi inserida uma coluna ”is_anomaly“ que indica se o momento em questão faz parte ou não de um crash inserido.

Em relação aos crashes, foi feita a geração de: 10 crashes de alto *lag* de mensagens *Kafka* combinado a oscilação de CPU e instâncias ativas; 6 crashes de alta latência HTTP e uso de CPU em um ou mais endpoint e shard da base de dados; e 4 crashes de alta latência HTTP – de maneira isolada – em um único endpoint. Como resultado, considerando esse número de crashes e a duração aleatória sorteada durante a geração dos mesmos, a base de dados final ficou com 1716 momentos anômalos, compondo aproximadamente 24% dos dados. Escolheu-se por não aumentar a duração dos crashes e equilibrar mais a proporção para desafiar o modelo ao replicar, em menor proporção, um desafio inerente à tarefas de

classificação deste tipo, que é o desequilíbrio entre a quantidade de dados positivos em relação a dados negativos.

4.4 Aplicação da base final de dados no algoritmo CNN 1D e avaliação

Por fim, a base de dados obtida foi aplicada no modelo CNN 1D definido para treinamento, com 70% das instâncias, e avaliação, com 30%.

Para avaliação do resultado, foram consideradas as medidas de precisão, recall e f1-score, mostradas na Tabela 4. Usando o f1-score como medida principal, pode-se destacar que nas predições de dados positivos o resultado foi de 0.97, enquanto para dados negativos foi de 0.81, aproximadamente.

Para análise visual, a Figura-5 exibe a matriz de confusão, que indica maioria no acerto de verdadeiros negativos, influência do desbalanceamento dos dados. Por outro lado, o algoritmo também foi capaz de prever uma quantidade considerável de verdadeiros positivos.

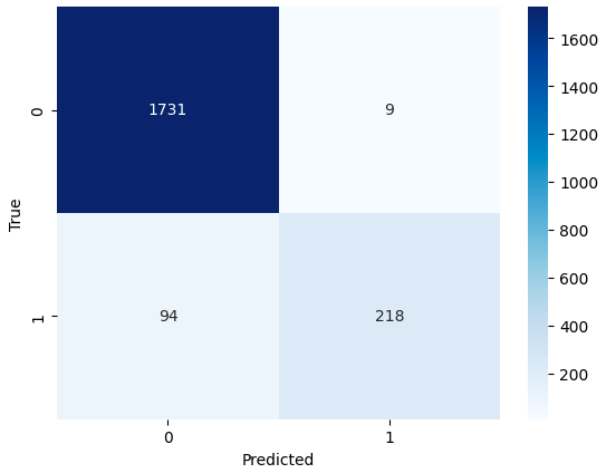

Figura 5 – Matriz de Confusão

Tabela 4 – Resultados da Avaliação do Modelo

	precisão	**recall**	**f1-score**
False	0.9485	0.9948	0.9711
True	0.9604	0.6987	0.8089
Média	0.9544	0.8468	0.8900

Fonte: Autor.

5 Conclusão

Após realização do estudo, é possível afirmar que os objetivos propostos por este trabalho foram satisfatoriamente alcançados.

Na ausência de uma base de dados real para que se possa utilizar no trabalho, criou-se uma base artificial suficientemente verossimil a uma base de dados real em relação à distribuição de dados numéricos, formato dos mesmos e problemas reais que ela apresenta. Criou-se também dados artificiais que representam anomalias comuns em ambientes de microsserviços para inserção dos mesmos nos dados de entrada. Embora a metodologia tenha sido arbitrária em todas escolhas feitas para geração dos dados e anomalias, pode-se dizer que a base de dados final satisfez os objetivos desejados, permitindo a realização do estudo.

Posteriormente, desenvolveu-se com sucesso algoritmos para tratamentos de dados e preperação dos mesmos para aplicação em modelos de Aprendizado de Máquina. Assim, diversos problemas gerais de pré-processamento de dados para projeto de Aprendizado de Máquina, bem como problemas específicos de uma base de dados reais de aplicação de microsserviços foram tratados em funções parametrizáveis que permitem, no futuro, fácil adequação para uso em diferentes dados e em diferentes maneiras.

Por fim, aplicou-se os dados sintéticos gerados em um algoritmo CNN 1D. Como resultado, obteve-se bons valores globais de precisão, recall e F1-Score, sendo todos acima de 85%. Porém, considerando o forte desbalanceamento entre as classes e a importância da classe positiva, a interpretação das métricas deve ir além da análise dos resultados de maneira global, dando maior peso àquelas que refletem o desempenho na classe minoritária, especialmente o recall, uma vez que em sistemas de detecção de anomalias, é importante que verdadeiros positivos não sejam mascarados e passem despercebidos, o que significaria, neste domínio, deixar crashes acontecerem sem serem notificados rapidamente pelo modelo.

A precisão indica a proporção de previsões positivas que de fato correspondem a instâncias positivas reais. Observa-se uma precisão elevada para ambas as classes, com destaque para a classe True (0.9604). Isso indica que, quando o modelo prediz a ocorrência da classe positiva, ele raramente está errado, ou seja, há poucos falsos positivos (confirmado pelos apenas 9 casos na matriz de confusão). Isto é relevante pois evita que profissionais sejam atrapalhados com a notificação de problemas que na verdade não existem, evitando que sua produtividade seja afetada durante a carga horário de trabalho ou os atrapalhe na vida pessoal, com alarmes fora do horário de trabalho. Porém, em cenários desbalanceados, isso sugere que o modelo é conservador ao sinalizar a classe minoritária, evitando alarmes falsos e, portanto, alta precisão isoladamente não garante boa cobertura da classe positiva,

o que reforça a necessidade de analisar o recall.

O recall mede a capacidade do modelo de identificar corretamente todas as instâncias positivas reais. Para a classe True, o valor obtido (0.987) indica que aproximadamente 70% dos casos positivos foram corretamente detectados, enquanto cerca de 30% foram perdidos (94 falsos negativos). Em problemas como deste trabalho, quando a classe positiva representa eventos críticos como crashes, falsos negativos são particularmente prejudiciais. Assim, apesar de o recall obtido ser razoável, ele evidencia uma limitação importante do modelo, que ainda deixa de identificar uma parcela relevante dos eventos de interesse. Dessa forma, seria necessário o refinamento do modelo para que esta medida tivesse um valor maior em um trabalho futuro, mesmo que isso significasse, por exemplo, perda de precisão, pois é mais importante, pensando no funcionamento do sistema, permitir falsos positivos de vez em quando do que deixar problemas reais despercebidos. Por outro lado, o recall extremamente alto da classe False (0.9948) mostra que o modelo está fortemente enviesado para reconhecer corretamente a classe majoritária, o que é um comportamento esperado nesse tipo de distribuição.

Esses resultados, porém, tem significância limitada. Considerando o uso de dados sintéticos, é provável que os crashes adicionados sejam muito destoantes do comportamento normal simulado, facilitando a identificação de crashes pelo modelo, o que justifica seu desempenho supostamente elevado mesmo sem muito refinamento. Em dados reais, os padrões dos dados são mais complexos ao longo do tempo, a distinção entre dados anômalos e normais é mais difícil de se diferenciar, há outros tipos de crashes não trabalhados neste estudo e podem haver falsos negativos nos dados. Tais fatores desafiariam mais o modelo. Ainda assim, o resultado obtido ilustra o potencial que o método desenvolvido tem para detecção de anomalias de maneira mais eficiente. Dessa forma, com refinamento, uso de dados reais e realimentação do modelo ao longo do tempo, resultados mais significativos poderiam ser obtidos.

Dessa maneira, este trabalho fornece como contribuição um pipeline de ponta a ponta, isto é, do pré-processamento a escolha de um modelo e seu treinamento e avaliação, que permite a identificação mais eficiente de anomalias em uma base real de dados. Além disso, os conceitos trabalhados, bem como as escolhas feitas e os algoritmos desenvolvidos, formam uma base valiosa de conhecimento que permite a replicação deste processo em outras bases de dados, com outras características, para obtenção dos mesmo fim: melhor identificação de anomalias reais quando comparado a métodos tradicionais de alarmes.

Outra contribuição deste trabalho é a mudança de perspectiva em relação a percepção de problemas reais em uma aplicação de microsserviços. Enquanto a empresa na qual se observou os dados e outras que mantém aplicações de microsserviços usam de uma granularidade mais fina, na qual se observam diversos alarmes diferentes a nível de anomalia de variável, este trabalho busca um método de detecção em granularidade

mais grossa de anomalia de instância. Assim, observando o sistema como um todo, a detecção de um problema real pode ser automatizada por um modelo de Aprendizado de Máquina, o que é mais simples do que o trabalho manual de análise de um operador humano do sistema de vários alarmes separados, um para cada métrica. Tal fator não garante, necessariamente, o descarte de alarmes de anomalias a nível de serviço. Para isso, seria necessário um refinamento grande e obtenção de resultados muito bons do modelo de Aprendizado Profundo, em especial o recall, para evitar a perda de verdadeiros positivos. Outra abordagem poderia ser uma abordagem mista entre as duas granularidades, removendo apenas alguns alarmes a nível de variável e adicionando uma camada extra de proteção com o sistema desenvolvido por Aprendizado Profundo.

Por fim, vale ressaltar que o objetivo do trabalho foi a detecção mais eficiente de anomalias. As ações tomadas após isso fogem do escopo do estudo. Porém, trabalhos futuros podem abordar essa temática.

5.1 Trabalhos Futuros

Apesar dos resultados satisfatórios, o seguinte trabalho tem possíveis próximas oportunidades de contribuição. Em um primeiro momento, uma próxima contribuição clara seria realizar um trabalho com foco inteiro na última etapa deste trabalho, que foi o algoritmo de Aprendizado de Máquina usado. Poderia-se refinar o modelo CNN 1D utilizado ou usar outro mais complexo, como o *Long Short-Term Memory* (LSTM), para se obter resultados melhores na predição de *crashes*. Vale mencionar, porém, que seria preferível o uso de dados reais em um eventual próximo trabalho com tal fim, para aumentar a confiabilidade do resultado obtido, uma vez que os mesmos tendem a ter padrões mais complexos a serem compreendidos. Nesse sentido, pode-se citar duas dificuldades inerentes à realização deste próximo trabalho: a ausência de bases de dados reais disponíveis para uso e, mesmo que se encontre uma base real, um possível desbalanceamento entre dados de métricas de dias normais, em alto volume, e dados de dias com anomalias, raros, causando dificuldade de aplicação e generalização dos dados no modelo. Em relação ao segundo, poderia-se fazer um trabalho que conseguisse, com o apoio manual dos times de desenvolvimento responsáveis pelos serviços, criar alguma solução que permitisse obter mais dados de anomalias reais e feedback das previsões dos modelos, realimentando o mesmo e o melhorando com o tempo.

Outra possibilidade de trabalho seria desenvolver uma maneira de avaliar melhor o potencial impacto que o método de detecção de anomalias por Aprendizado Profundo tem em relação a métodos tradicionais de alarmes. Para isso, poderia-se desenvolver algoritmos que simulam a detecção de anomalias com uso de um limiar e janela de tolerância, como é atualmente feito na organização responsável pela base de dados reais que inspirou este

trabalho. A partir daí, com os dados e algoritmos gerados neste trabalho, poderia-se verificar qual dos dois métodos identifica mais crashes reais e menos falsos positivos. Tal processo poderia ser iterativo e usado para melhorar o modelo de Redes Neurais usado.

Em um trabalho de abordagem diferente, poderia-se também mudar a estratégia de algoritmo e fazer-se o uso de um algoritmo de aprendizado não supervisionado para detecção automática de anomalias sem rótulos para posterior comparação com modelo supervisionado. Isso se justifica pois, em tal ambiente de detecção de anomalias, um problema real não tratado neste trabalho é a presença de falsos negativos e falsos positivos nos dados rotulados. Isso se deve ao fato de que, no sistema atual, a classificação do que é anomalia ou não é feito de maneira manual por engenheiros, o que pode causar momentos no conjunto de dados em que anomalias reais são escondidas como momentos de normalidade, ou vice-versa, em que momentos de normalidade com um pouco de variação foram reportados como anomalias por excesso de cuidado. Tais fatores podem influenciar negativamente no comportamento de algoritmos supervisionados, motivo pelo qual seria interessante a comparação com modelos não supervisionados.

Embora fuja do escopo deste trabalho, o que é feito após detecção da anomalia pode ser tema de trabalhos futuros. Pode-se pensar, por exemplo, na adição de funcionalidades extras que permitam ao sistema criado não só detectar as anomalias, mas também já reportá-la com algum diagnóstico inicial, podendo ser quais métricas estão fora do convencional, mensagens de erros obtidas de logs ou qualquer informação útil que possa dar dicas sobre como proceder.

Referências

APACHE KAFKA. Apache Kafka Documentation. Disponível em: <https://kafka.apache.org/> Acesso em: 06 dez. 2025.

BATISTA, Gustavo Enrique de Almeida Prado Alves. Pré-processamento de dados em aprendizado de máquina supervisionado. 2003. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2003.

CHATFIELD, Chris. The Analysis of Time Series: An Introduction. 6. ed. Boca Raton: Chapman & Hall/CRC, 2003.

CHOI, K.; YI, J.; PARK, C.; YOON, S. Deep Learning for Anomaly Detection in Time-Series Data: Review, Analysis, and Guidelines. IEEE Access, v. 9, p. 120043-120065, 2021. DOI: 10.1109/ACCESS.2021.3107975.

HRUSTO, A.; ENGSTRÖM, E.; RUNESON, P. Optimization of anomaly detection in a microservice system through continuous feedback from development. In: IEEE/ACM 10th INTERNATIONAL WORKSHOP ON SOFTWARE ENGINEERING FOR SYSTEMS-OF-SYSTEMS AND SOFTWARE ECOSYSTEMS (SESoS), 2022, Pittsburgh. Anais... Piscataway: IEEE, 2022. p. 13–20.

James Lewis and Martin Fowler. 2014. Microservices - a definition of this new architectural term. https://martinfowler.com/articles/microservices.html. Acesso em: 2025-12-02.

LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. Deep learning. Nature, v. 521, n. 7553, p. 436–444, 2015. DOI: https://doi.org/10.1038/nature14539. Acesso em: 2025-12-02.

NOBRE, J.; PIRES, E. J. S.; REIS, A. Anomaly Detection in Microservice-Based Systems. Applied Sciences, v. 13, n. 13, p. 7891, 2023. DOI: https://doi.org/10.3390/app13137891. Acesso em: 2025-12-02.

XU, Haowen; CHEN, Wenxiao; ZHAO, Nengwen; LI, Zeyan; BU, Jiahao; LI, Zhihan; LIU, Ying; ZHAO, Youjian; PEI, Dan; FENG, Yang; CHEN, Jie; WANG, Zhaogang; QIAO, Honglin. Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications. In: Proceedings of the 2018 World Wide Web Conference on World Wide Web, 2018, p. 187–196. DOI: 10.1145/3178876.3185996. Acesso em 2025-12-02.

YAMASHITA, Rikiya; NISHIO, Mizuho; DO, Richard Kinh Gian; TOGASHI, Kaori. Convolutional neural networks: an overview and application in radiology. Insights into Imaging, v. 9, n. 4, p. 611–629, 2018. DOI: 10.1007/s13244-018-0639-9. Acesso em:

2025-12-02. .