Técnica de Aprendizagem não Supervisionada para Detecção de Falhas em Computação em Nuvem

Carlos Manoel Nunes e Silva[1], **Erica Teixeira Gomes de Sousa**[2]

[1]Departamento de Computação – Universidade Federal Rural de Pernambuco (UFRPE)
Caixa Postal 06.316 – 52.171-900 – Recife – PE – Brasil

carlos.manoel@ufrpe.br, erica.sousa@ufrpe.br

Abstract. *The increasing complexity of cloud computing environments has intensified the need for efficient techniques for automatic fault detection. This article proposes an unsupervised learning-based approach for log analysis, using Skip-Gram for embedding generation and K-Means for identifying anomalous patterns. The experiments were conducted in an OpenStack environment with controlled injection of resizing and mismatch faults. The results demonstrated an AUC of 0.89 and 87% accuracy for the resizing fault, and an AUC of 0.857 and 83% accuracy for the mismatch fault. The recall of 1.0 for the mismatch fault stands out, indicating 100% detection of real faults.*

Resumo. *A crescente complexidade dos ambientes de computação em nuvem tem intensificado a necessidade de técnicas eficientes para detecção automática de falhas. Este artigo propõe uma abordagem baseada em aprendizado não supervisionado para análise de logs, utilizando Skip-Gram para geração de embeddings e K-Means para identificação de padrões anômalos. Os experimentos foram conduzidos em um ambiente OpenStack com injeção controlada de falhas dos tipos resizing e mismatch. Os resultados demonstraram AUC de 0,89 e acurácia de 87% para a falha resizing, e AUC de 0,857 e acurácia de 83% para a falha mismatch. Destaca-se o Recall de 1,0 para a falha mismatch, indicando detecção de 100% das falhas reais.*

1. Introdução

A computação em nuvem consolidou-se como o principal paradigma para a implantação de aplicações modernas, impulsionada pela crescente adoção de arquiteturas distribuídas, como microserviços e sistemas orientados a eventos. Nos últimos anos, avanços em virtualização, conteinerização e orquestração têm ampliado a escalabilidade e a flexibilidade desses ambientes, ao mesmo tempo em que aumentam sua complexidade operacional. Nesse cenário, a gestão eficiente de infraestruturas em larga escala tornou-se um desafio significativo, especialmente diante da dinamicidade e heterogeneidade dos serviços distribuídos. Estudos recentes destacam que a evolução para ambientes cloud-native e multi-cloud intensifica ainda mais a necessidade de soluções automatizadas para monitoramento e análise de sistemas [Zhou et al. 2024, Rana et al. 2024].

Diante desse contexto, a detecção precoce de falhas tornou-se um elemento crítico para assegurar a confiabilidade e a disponibilidade dos serviços em ambientes de computação em nuvem cada vez mais dinâmicos e distribuídos. Recentemente, abordagens tradicionais de monitoramento vêm sendo substituídas por soluções baseadas em

AIOps (*Artificial Intelligence for IT Operations*), que integram técnicas avançadas de aprendizado de máquina e aprendizado profundo para análise automatizada de grandes volumes de dados operacionais. Estudos recentes destacam o uso de modelos baseados em aprendizado profundo, análise multivariada de telemetria e mecanismos de atenção para capturar padrões complexos e prever falhas de forma proativa em infraestruturas de larga escala [Wang et al. 2024, Liu et al. 2025, Mohammed et al. 2025a]. Apesar desses avanços, tais abordagens frequentemente apresentam alto custo computacional, além de dependerem de grandes volumes de dados e infraestrutura robusta para treinamento e implantação em ambientes reais.

Entre as diversas fontes de dados disponíveis para análise de sistemas, os registros de log permanecem como uma das principais fontes de informação para diagnóstico de falhas e compreensão do comportamento operacional. Nos últimos anos, a área tem avançado significativamente com a adoção de modelos baseados em transformers e, mais recentemente, *Large Language Models* (LLMs), que permitem capturar relações semânticas complexas em sequências de logs e melhorar a detecção de anomalias. Trabalhos recentes exploram o uso de aprendizado auto-supervisionado, embeddings contextuais e estratégias baseadas em prompting para tornar a análise de logs mais robusta e adaptável a diferentes cenários operacionais [He et al. 2024, Zhang et al. 2024, Kim et al. 2025]. Apesar desses avanços, tais abordagens ainda enfrentam desafios relacionados à interpretabilidade, ao alto custo computacional e à necessidade de infraestrutura robusta, o que pode limitar sua aplicação prática em ambientes reais.

Diante desse contexto, este trabalho tem como objetivo investigar a utilização de uma abordagem baseada em aprendizado não supervisionado para a detecção de falhas de software em ambientes de computação em nuvem. A metodologia proposta utiliza o modelo *Skip-Gram* com *Negative Sampling* para aprender representações distribuídas a partir de registros de log gerados durante a execução do sistema. Em seguida, o algoritmo *K-Means* é aplicado para identificar padrões de comportamento no espaço vetorial gerado pelos embeddings. A detecção de anomalias é realizada por meio da análise da distância entre os registros de log e os centróides dos clusters, permitindo identificar desvios significativos em relação ao comportamento normal do sistema.

Este artigo está organizado da seguinte forma. A Seção 2 apresenta os conceitos básicos de computação em nuvem. A seção 3 apresenta os principais trabalhos relacionados à detecção de falhas e análise de anomalias em ambientes de computação em nuvem. A Seção 4 descreve a metodologia proposta, incluindo o processo de geração de embeddings de logs e o método de detecção baseado em agrupamento. A Seção 5, apresenta o estudo de caso, contendo o ambiente experimental, o processo de injeção de falhas utilizados na avaliação da metodologia proposta, discute os resultados obtidos e realiza uma análise comparativa com abordagens da literatura. Por fim, a Seção 6 apresenta as conclusões do trabalho e possíveis direções para pesquisas futuras.

2. Trabalhos Relacionados

A detecção automática de falhas em ambientes de computação em nuvem tem sido amplamente investigada na literatura recente, impulsionada pela crescente complexidade das infraestruturas distribuídas e pelo grande volume de dados gerados por sistemas de monitoramento. Trabalhos recentes têm proposto diferentes abordagens para a identificação

de comportamentos anômalos, variando desde métodos estatísticos e algoritmos clássicos de aprendizado de máquina até modelos mais recentes baseados em aprendizado profundo. Em particular, observa-se um avanço no uso de técnicas capazes de explorar dependências temporais e estruturais em dados de telemetria e logs, incluindo arquiteturas baseadas em redes neurais recorrentes, convolucionais e mecanismos de atenção [Zhou et al. 2024, Mohammed et al. 2025a]. Além disso, abordagens mais recentes têm investigado o uso de modelos baseados em transformers e Large Language Models (LLMs) para capturar relações semânticas mais complexas nos dados, ampliando a capacidade de detecção de anomalias [He et al. 2024, Kim et al. 2025]. Apesar dos avanços, muitos desses métodos ainda enfrentam desafios relacionados ao custo computacional, à necessidade de grandes volumes de dados e à adaptação a diferentes cenários operacionais

Um dos trabalhos recentes que exploram o uso de arquiteturas de aprendizado de máquina para detecção de anomalias em ambientes de nuvem é apresentado em [Mohammed et al. 2025b]. Nesse estudo, os autores propõem um sistema de detecção de anomalias voltado para ambientes de computação em nuvem seguros, utilizando diferentes algoritmos de aprendizado de máquina, como KNN, *Decision Tree*, *Random Forest* e *Support Vector Machine*. A principal contribuição do trabalho consiste na construção de um modelo baseado em stacking ensemble learning, capaz de combinar diferentes classificadores para melhorar a precisão na identificação de comportamentos anômalos em ambientes de nuvem.

Outra abordagem relevante é apresentada em [112], que propõe um modelo de detecção preditiva de falhas em infraestrutura de nuvem baseado na análise multivariada de dados de telemetria e registros de log. Nesse estudo, os autores utilizam *Temporal Convolutional Networks* (TCN) combinadas com mecanismos de atenção para capturar dependências temporais entre diferentes métricas operacionais do sistema, como utilização de CPU, memória e rede. A principal contribuição desse trabalho consiste na capacidade de identificar padrões que antecedem a ocorrência de falhas, permitindo antecipar eventos anômalos antes que eles impactem diretamente o funcionamento da infraestrutura.

Em um contexto distinto, o trabalho apresentado em [964] propõe um método de detecção de defeitos de software baseado em classificação supervisionada aplicada a métricas estáticas de código-fonte. A abordagem utiliza técnicas de avaliação de classificação assimétrica para lidar com conjuntos de dados desbalanceados, situação comum em bases de defeitos de software. A principal contribuição do trabalho consiste na melhoria da capacidade de identificação de módulos defeituosos durante o processo de desenvolvimento de software, contribuindo para a prevenção de falhas antes da implantação do sistema.

Além das abordagens baseadas em aprendizado profundo e classificação supervisionada, pesquisas recentes têm explorado o uso de representações em grafos para a detecção de falhas em ambientes de computação em nuvem. Nesse contexto, grafos de dependência são utilizados para modelar as interações entre componentes e serviços distribuídos, permitindo capturar relações estruturais que não são evidentes em análises isoladas de métricas ou logs. Trabalhos recentes destacam o uso de técnicas de aprendizado sobre grafos, incluindo métodos baseados em embeddings estruturais e *Graph Neural Networks* (GNN), para identificar padrões anômalos a partir da topologia e da dinâmica das interações entre serviços [Wu et al. 2024, Li et al. 2025]. Essas abordagens têm se

mostrado eficazes na detecção de falhas em arquiteturas baseadas em microserviços, especialmente por sua capacidade de incorporar informações contextuais e dependências entre componentes. No entanto, ainda enfrentam desafios relacionados à escalabilidade, à construção dos grafos em tempo real e ao custo computacional associado ao treinamento dos modelos.

Outros estudos recentes têm investigado o uso de modelos de aprendizado profundo para análise de logs em sistemas distribuídos. Abordagens mais atuais têm evoluído a partir de modelos baseados em transformadores, incorporando arquiteturas mais avançadas e, recentemente, *Large Language Models* (LLMs) para tratar logs como sequências textuais ricas em contexto. Esses modelos utilizam representações contextuais e técnicas de aprendizado auto-supervisionado para capturar relações semânticas complexas entre eventos, melhorando a detecção de anomalias em grandes volumes de dados [Zhang et al. 2024, He et al. 2024]. Além disso, trabalhos recentes exploram estratégias baseadas em prompting e modelos generativos, ampliando a capacidade de generalização e adaptação a diferentes cenários operacionais [Kim et al. 2025]. Apesar dos avanços, desafios como custo computacional elevado, necessidade de grandes volumes de dados e baixa interpretabilidade ainda persistem, limitando a aplicação prática dessas abordagens em ambientes reais.

Apesar dos avanços apresentados por essas abordagens, muitas soluções dependem de arquiteturas complexas de aprendizado profundo, integração de múltiplas fontes de telemetria ou grandes volumes de dados rotulados para treinamento. Em contrapartida, abordagens baseadas em aprendizado não supervisionado e análise direta de registros de log apresentam uma alternativa mais simples e de menor custo computacional para a detecção de falhas em ambientes de nuvem. Nesse contexto, o presente trabalho propõe uma metodologia baseada na combinação de representações distribuídas de logs por meio do modelo *Skip-Gram* e na aplicação do algoritmo *K-Means* para identificação de padrões comportamentais no espaço vetorial dos registros de log.

3. Metodologia de Detecção de Erros em Ambientes de Nuvem Privada

Essa seção apresenta a metodologia adotada para detecção de falhas em ambientes de nuvem, utilizando técnicas de vetorização de logs e algoritmos de aprendizado de máquina não supervisionado. A metodologia é composta pelas fases: Configuração do Ambiente, Injeção de Falhas, Vetorização e, por fim, Detecção de Falhas. Figura 1

3.1. Fase 1: Configuração do Ambiente

A primeira fase tem por finalidade estabelecer um ambiente controlado e reproduzível para a realização dos experimentos de injeção e detecção de falhas. A definição de um ambiente controlado permite a reprodutibilidade dos experimentos e o monitoramento adequado do comportamento do sistema diante das falhas injetadas.

3.2. Fase 2: Injeção de Falhas

A segunda fase tem como objetivo gerar dados representativos de falhas reais no ambiente de nuvem, a fim de possibilitar a análise e avaliação dos métodos de detecção. Dessa forma, busca-se produzir registros de log contendo comportamentos anômalos que reflitam cenários reais de falha.

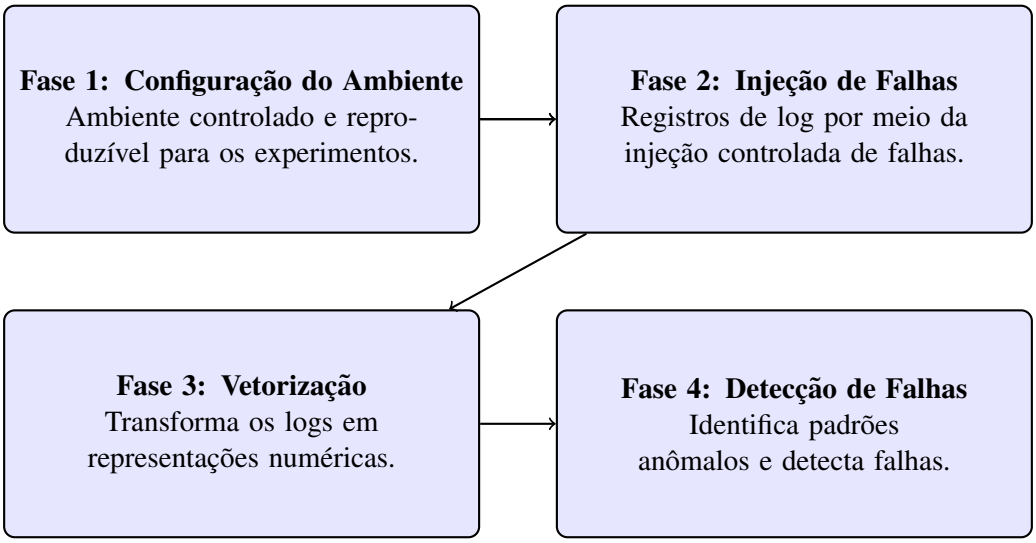

Figura 1. Metodologia para Detecção de Falhas em Ambientes de Nuvem

Para a realização dessa etapa, foi utilizada a ferramenta CIMut [Duarte and Sousa 2024], desenvolvida em *Python*, ela auxilia na injeção de falhas em ambientes de nuvem por meio da mutação de código-fonte. A ferramenta é composta por dois *scripts* principais: o *script* de verificação de conteúdo e o script de alteração de linha.

O *script* de verificação de conteúdo incorpora um mecanismo que permite validar a existência do trecho a ser modificado em um arquivo e linha específicos, garantindo maior precisão no processo de mutação. Já o *script* de alteração de linha possibilita modificar o conteúdo de uma linha específica em um arquivo remoto.

Para sua utilização, é necessário fornecer as credenciais de acesso SSH (endereço IP, porta, nome de usuário e senha), o caminho do arquivo remoto, o número da linha a ser alterada e o novo conteúdo desejado. A ferramenta estabelece uma conexão SSH com o *host* remoto, realiza o *download* do arquivo especificado, efetua a modificação na linha indicada e, em seguida, envia o arquivo alterado de volta ao *host*.

Para direcionar a escolha dos arquivos a serem modificados, foram consultados os relatórios de bugs classificados como *crash*, *hang*, degradação de desempenho e funcionalidade incorreta reportados no Bugzilla, plataforma dedicada ao reporte de bugs relacionados ao OpenStack. Após a análise das descrições dos bugs conforme a ODC, os arquivos de código-fonte relevantes mencionados nos relatórios foram localizados no repositório *GitHub* do *OpenStack*. Em seguida, foi realizada uma análise minuciosa do código para identificar as modificações específicas que estavam relacionadas aos reportes dos usuários da plataforma OpenStack. Esse processo sistemático garantiu que as modificações fossem embasadas em problemas reais relatados pela comunidade de usuários e desenvolvedores do OpenStack, aumentando a relevância e a eficácia da injeção de falhas.[Duarte and Sousa 2024]

Após a modificação, a mutação é registrada em um arquivo de log. Para que as alterações surtam efeito no ambiente, é necessário reiniciar o *host*, permitindo que o *OpenStack* recompile o código com o arquivo modificado.

3.3. Fase 3: Vetorização

O objetivo da terceira fase é transformar os dados textuais extraídos dos *logs* em representações numéricas adequadas para aplicação de algoritmos de aprendizado de máquina. Dessa forma, essa transformação possibilita a análise automatizada dos dados, permitindo capturar relações semânticas entre os eventos registrados.

Os arquivos de *log* do ambiente de nuvem privada *OpenStack* foram coletados em formato textual, contendo informações relevantes sobre eventos da plataforma de nuvem, incluindo mensagens associadas à ocorrência e à propagação de falhas nas diferentes camadas da arquitetura.

Considerando que os *logs* são dados não estruturados, torna-se necessária sua conversão em representações numéricas apropriadas. Para isso, foi utilizado o algoritmo *Skip-Gram*, pertencente à família *Word2Vec*, cuja finalidade é converter palavras em vetores densos capazes de capturar relações semânticas presentes na linguagem natural [Leshem 2020].

O modelo *Skip-Gram* é baseado em uma rede neural rasa composta por camada de entrada, camada de incorporação (*embedding*) e camada de saída. Seu objetivo é maximizar a probabilidade de ocorrência das palavras de contexto dado uma palavra-alvo, aprendendo representações vetoriais de modo que palavras que compartilham contextos semelhantes possuam vetores próximos no espaço vetorial [Leshem 2020].

Como resultado do treinamento, obtém-se uma matriz de *embeddings* que representa numericamente os eventos registrados nos *logs*, preservando padrões semânticos relevantes e preparando os dados para a etapa de detecção de falhas.

3.4. Fase 4: Detecção de Falhas

A quarta fase teve por objetivo identificar padrões anômalos nos logs vetorizados, distinguindo comportamentos associados a falhas daqueles correspondentes ao regime regular de operação.

A partir das representações vetoriais geradas na etapa anterior, tornou-se possível analisar os padrões presentes nos *logs* de forma automatizada. Esses registros continham informações sobre os momentos de ocorrência das falhas, permitindo não apenas a identificação de erros na camada Nova, mas também a análise da propagação das falhas do tipo *resizing* e *mismatch* para outras camadas que compõem a arquitetura do *OpenStack*.

Com base nesses vetores, foram aplicadas técnicas de aprendizado não supervisionado, utilizando o algoritmo *K-Means* para identificar padrões anômalos, ele foi utilizado para agrupar as representações vetoriais dos logs, permitindo a identificação de padrões de comportamento no sistema. Trata-se de um método não supervisionado que particiona os dados em k *clusters*, atribuindo cada amostra ao grupo cujo centróide está mais próximo, com base na distância euclidiana, buscando minimizar a soma das distâncias intra-*cluster*.

O treinamento ocorreu de forma iterativa, alternando entre a atribuição dos pontos aos clusters e a atualização dos centróides, foi utilizada a implementação da biblioteca *scikit-learn*, com os parâmetros *n_init*=10, para reduzir o impacto de inicializações aleatórias, e *random_state*=42, garantindo a reprodutibilidade dos experimentos.

O número de *clusters* foi definido empiricamente, visando um equilíbrio entre coesão e separação dos grupos. No contexto da detecção de falhas, os *clusters* representam padrões de comportamento dos *logs*, sendo possíveis desvios identificados com base na distância aos centróides. Como resultado, a etapa de detecção permitiu evidenciar agrupamentos de eventos similares, bem como isolar aqueles que representam desvios relevantes no funcionamento do sistema.

4. Estudo de Caso

O presente estudo de caso tem como objetivo avaliar uma metodologia proposta para detecção de falhas de software em ambientes privados de computação em nuvem.

4.1. Configuração do Ambiente

O ambiente de nuvem privada foi configurado com a plataforma de nuvem Openstack na sua versão 8.2.0, de forma não distruída, em um máquina contendo o processador *intel core* I5-3330S, 8GB de memória RAM, um SSD sata 2.0 de 480GB e sistema operacional linux Ubuntu 24.04.3 LTS.

4.2. Injeção de Falhas

Após a configuração do ambiente experimental, utilizou-se a ferramenta CI-Mut (Ferramenta de Injeção de Falhas em Ambientes de Nuvem por Mutação) [Duarte and Sousa 2024] com o objetivo de introduzir falhas de software na nuvem privada configurada com a plataforma *Openstack*.

A ferramenta possibilitou o mapeamento e a reprodução de falhas previamente relatadas por usuários do *OpenStack* na plataforma *Bugzilla* [Duarte and Sousa 2024], especificamente na camada Nova. Foram selecionados dois bugs classificados como severos, denominados *mismatch* e *resizing*.

As falhas foram injetadas de forma isolada, ou seja, cada bug foi introduzido separadamente no ambiente experimental, a fim de evitar interferências entre eles e possibilitar a análise individual dos seus impactos no comportamento do sistema.

Para a falha resizing, foi modificada a linha 90 do arquivo *block_device.py*, substituindo o trecho ”volume_size”: self.volume_size por ”volume_size”: None. Essa alteração provoca um redimensionamento inconsistente, no qual uma instância é migrada de um ambiente com *swap* para outro sem suporte a *swap*, levando o servidor a um estado de erro. Adicionalmente, o serviço *Compute* da camada Nova passa a comparar o ambiente original com o ambiente de destino, atualizando incorretamente o registro BDM (*Block Device Mapping*) durante o processo de redimensionamento.

Para a falha mismatch, foi alterada a linha 10647 do arquivo *manager.py*, substituindo a instrução nodenames = set(self.driver.get_available_nodes()) por nodenames = ’2’. Essa modificação compromete a obtenção correta dos nós disponíveis, fazendo com que o comando nova-host-evacuation retorne endereços PCI inconsistentes, além de gerar a mensagem de erro: “Não foi possível correlacionar o slot PCI”.

Para o monitoramento do ambiente, foram desenvolvidos dois *scripts* em Python. O primeiro foi responsável por monitorar os recursos computacionais da máquina hospedeira na qual a plataforma de nuvem estava instalada. As métricas coletadas incluíram:

Componente	Falha	Título	Atividade	Gatilho	Impacto
openstack-nova	*Resizing*	Redimensionar uma instância de um flavor que possui *swap* para outro flavor sem *swap* faz com que a instância entre em estado de erro.	Teste de sistema	Teste Bloqueado	*Crash*
openstack-nova	*Mismatch*	A nova funcionalidade de evacuação de *host* retorna endereços PCI incorretos e gera o erro: *'Unable to correlate PCI slot'*.	Teste de sistema	Inicialização/ Reinicialização	*Crash*

Tabela 1. *Bugs* injetados no módulo Nova do *Openstack*

Falha	Localização do Arquivo	Linha Alterada	Conteúdo Original	Conteúdo Alterado	Link	Identificador
Resizing	*/opt/stack/nova/nova/objects/block_device.py*	90	*'volume_size': self.volume_size,*	*'volume_size': None,*	*https://bugzilla.redhat.com/show_bug.cgi?id=1774332*	*Instance Resize (No-Swap)*
Mismatch	*/opt/stack/nova/nova/compute/manager.py*	10647	*nodenames = set(self.driver.get_available_node())*	*nodenames = '2'*	*https://bugzilla.redhat.com/show_bug.cgi?id=1852110*	*PCI Mismatch (Host Evacuation)*

Tabela 2. Falhas injetadas no módulo Nova do *OpenStack*

utilização de CPU, utilização de memória, tráfego de rede, além de operações de leitura e escrita em disco.

O segundo *script* teve como finalidade monitorar o estado dos sete sub-serviços que compõem a camada Nova do *OpenStack*, que são: *nova-api*, *nova-compute*, *nova-scheduler*, *nova-conductor*, *nova-consoleauth*, *nova-novncproxy*, *nova-cert*. Ambos os *scripts* realizaram coletas em intervalos de 10 segundos, registrando os dados em arquivos no formato `.csv`, o que possibilitou a organização e posterior análise dos resultados obtidos.

4.3. Discussão de Resultados

Esta seção apresenta a discussão dos resultados obtidos a partir dos experimentos realizados para a detecção de falhas na nuvem privada configurada com a plataforma *Openstack*. A análise tem como objetivo avaliar a eficiência da abordagem proposta, baseada na combinação dos algoritmos *Skip-Gram* e *K-Means*, na identificação de comportamentos anômalos em registros de execução do *Openstack*.

Os resultados obtidos demonstram que a abordagem proposta apresenta elevada capacidade de detecção de falhas, sustentada por métricas robustas de AUC, acurácia e *Recall*. No cenário *resizing*, o modelo atingiu AUC de 0,89 e acurácia de 87%, indicando forte capacidade discriminativa entre eventos normais e anômalos. No cenário *mismatch*, foram obtidos AUC de 0,857 e acurácia de 83%, mantendo desempenho elevado mesmo sob variações no padrão das falhas. Destaca-se, sobretudo, o *Recall* de 1,0 no experimento *mismatch*, evidenciando que 100% das falhas reais foram corretamente identificadas. Esses resultados confirmam que o modelo é capaz de separar comportamentos normais e anômalos, mesmo operando em um contexto não supervisionado e utilizando exclusivamente logs de execução.

Observa-se que, para a falha *resizing*, os sub-serviços analisados alternaram principalmente entre os estados *ativo* e *inativo*, com frequências próximas ao longo do período monitorado. O estado *ativo* apresentou entre 70 e 76 ocorrências, enquanto o estado *inativo* variou entre 79 e 81 ocorrências. Em contrapartida, o estado *desativando* foi registrado apenas em dois serviços, *n-super-cond* e *n-sch*, com apenas uma ocorrência em cada. Esse comportamento indica que, apesar da ocorrência da falha, a maior parte dos componentes da infraestrutura permaneceu em operação, sugerindo que o impacto da anomalia ocorreu de forma localizada no sistema, conforme a Figura 2.

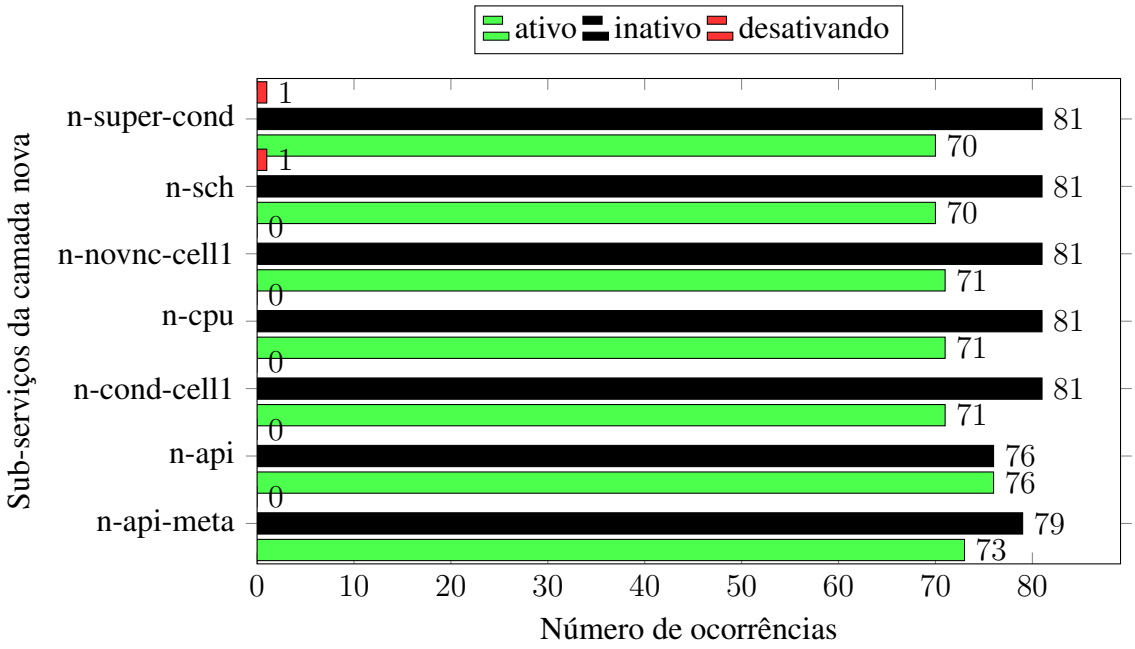

Figura 2. Distribuição dos estados dos sub-serviços da camada nova durante o experimento de *resizing*.

A avaliação do método de detecção da anomalia *resizing* foi realizada com base na métrica AUC, tendo obtido o valor de 0,89. Esse resultado indica que o modelo possui boa capacidade discriminativa entre eventos normais e anômalos, sendo capaz de atribuir, de forma consistente, *scores* mais elevados aos registros associados a falhas em comparação aos eventos normais. Como a AUC é independente de um limiar específico de decisão, esse valor sugere que a separação entre as classes no espaço de pontuação é adequada, mesmo que a classificação final dependa da escolha de um threshold. Dessa forma, o

desempenho observado indica que o modelo consegue realizar a ordenação corretamente da maioria dos eventos, permitindo que ajustes no limiar de decisão sejam explorados para otimizar o equilíbrio entre falsos positivos e falsos negativos na detecção de anomalias.

Observa-se que, para a falha *mismatch*, a maior parte dos sub-serviços permaneceram predominantemente nos estados *ativo* e *inativo*, com frequências próximas ao longo do período monitorado. O estado *ativo* apresentou entre 69 e 77 ocorrências, enquanto o estado *inativo* variou entre 82 e 83 ocorrências. Por outro lado, o estado *desativando* foi registrado apenas em dois serviços, *n-super-cond* e *n-sch*, com apenas uma ocorrência em cada. Esse comportamento indica que, embora a falha tenha provocado alterações no funcionamento do sistema, a maioria dos componentes continuou alternando entre estados operacionais padrão, evidenciando que o impacto da anomalia ocorreu de forma localizada na infraestrutura, conforme a Figura 3

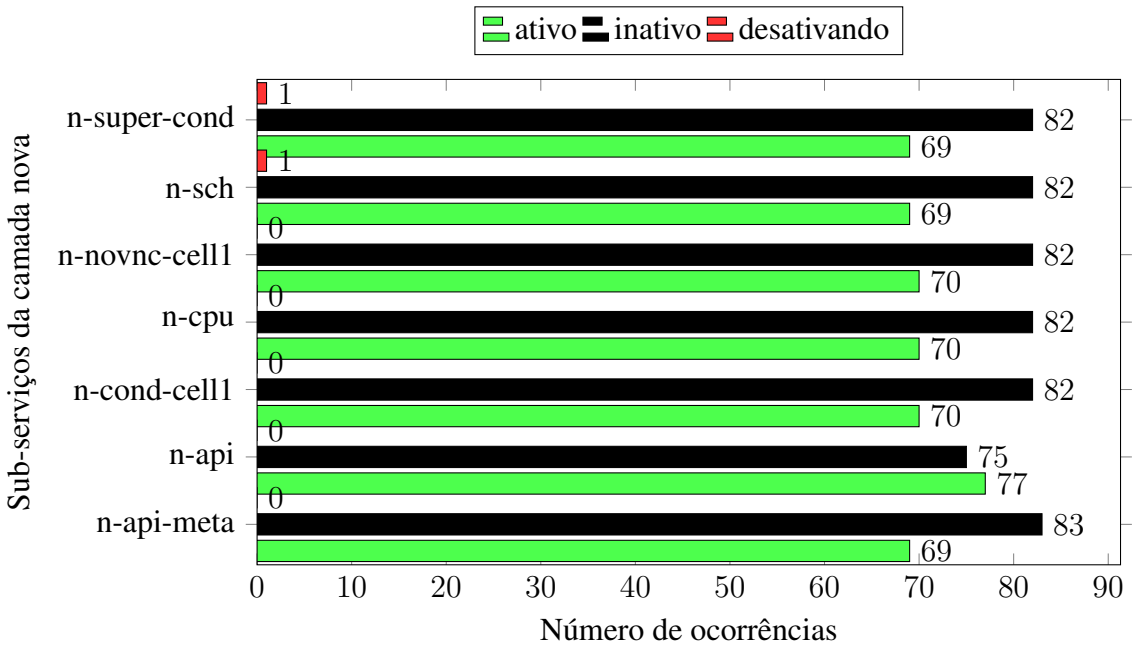

Figura 3. Distribuição dos estados dos sub-serviços da camada nova durante o experimento de *mismatch*.

A ocorrência de falsos positivos está relacionada à forma como o *Skip-Gram* representa as linhas de *log* durante o processo de tokenização. Devido à presença significativa de caracteres especiais, identificadores dinâmicos e estruturas não linguísticas, o modelo tende a encapsular esses elementos como *tokens* independentes, mesmo quando não correspondem a padrões semânticos relevantes, introduzindo ruído na representação vetorial e contribuindo para detecções incorretas. Para mitigar esse efeito, foi realizado o ajuste da taxa de aprendizado, tornando a atualização dos embeddings mais gradual e estável, o que reduz variações abruptas no vetor gradiente e favorece a construção de representações mais consistentes dos logs.

Um aspecto importante a ser considerado na interpretação dos resultados é a diferença observada entre os valores de *Precision* e AUC, para o *resizing* AUC de 0,89 e Precision 0,0008 e para o *mismatch* AUC de 0,857 e *Precision* de 0,0046.

A baixa precisão do modelo está associada ao aumento de falsos positivos, decorrente, em parte, da etapa de pré-processamento dos logs. A presença de caracteres especiais, identificadores únicos e tokens pouco frequentes gera um vocabulário esparso, comprometendo a qualidade da tokenização. Nesse cenário, o *Skip-Gram* produz representações vetoriais menos discriminativas, projetando eventos normais próximos a padrões anômalos no espaço vetorial. Como consequência, o *K-Means* apresenta maior sobreposição entre clusters, dificultando a separação entre comportamentos normais e falhos e elevando a taxa de falsos positivos, o que impacta diretamente a precisão. Ainda assim, o modelo mantém alto *recall*, indicando boa capacidade de detecção de falhas, embora com maior incidência de alarmes incorretos.

Embora a precisão obtida pelo modelo tenha sido relativamente baixa, principalmente devido à elevada quantidade de falsos positivos identificados, os valores de AUC obtidos nos experimentos (0,89 para *resizing* e 0,857 para *mismatch*) indicam que o modelo possui boa capacidade de separação entre padrões normais e anômalos. Essa diferença ocorre porque a métrica AUC avalia a capacidade do modelo de ordenar corretamente os exemplos com base nos scores gerados, independentemente de um limiar específico de decisão. Nesse contexto, os resultados sugerem que os *logs* associados a falhas tendem a apresentar distâncias maiores em relação aos centróides dos *clusters*, posicionando-se consistentemente acima dos registros normais no *ranking* de *scores*. Dessa forma, embora o limiar de decisão utilizado nos experimentos tenha resultado em um número elevado de falsos positivos, a distribuição dos scores indica que ajustes no *threshold* de detecção ou a utilização de estratégias adicionais de filtragem podem reduzir significativamente esses erros sem comprometer a capacidade do modelo de identificar eventos anômalos.

Observa-se, a partir da análise dos valores médios de utilização de CPU e memória nos momentos de ocorrência das falhas dos tipos *resizing* e *mismatch*. Observa-se que, em ambos os cenários, os recursos operam sob níveis elevados de utilização, indicando um padrão de sobrecarga associado à ocorrência de falhas. No caso do *resizing*, a utilização média de CPU atinge aproximadamente 82%, enquanto a memória apresenta cerca de 57%, caracterizando um cenário com maior pressão sobre o processamento. A análise detalhada das métricas de infraestrutura mostra ainda picos de CPU de até 100%, com média geral de 24,9%, enquanto o consumo de memória atingiu 82,5%, com média de 52,6% ao longo do período analisado.

Já no *mismatch*, tanto a CPU quanto a memória apresentam valores médios elevados e mais equilibrados, com aproximadamente 78% e 72%, respectivamente. Nesse cenário, observa-se um comportamento mais intenso, com picos de CPU também alcançando 100% e média de 33,3%, enquanto a memória atingiu 97%, com média de 67,8%, evidenciando maior pressão global sobre os recursos do sistema. Esses resultados indicam que ambas as falhas provocam aumento significativo na carga computacional, sendo mais pronunciado no cenário *mismatch*. Esse comportamento se reflete no desempenho do modelo proposto, que, mesmo sem utilizar diretamente métricas de infraestrutura como entrada, foi capaz de identificar eventos anômalos por meio da análise estrutural dos *logs*. No cenário *resizing*, o modelo obteve AUC de 0,89 e acurácia de 87%, embora o *Recall* de 0,10 indique baixa sensibilidade na detecção das falhas. Por outro lado, no cenário *mismatch*, foram alcançados AUC de 0,857, acurácia de 83% e *Recall*

de 1,0, evidenciando alta capacidade de detecção nesse contexto, conforme a Figura 4

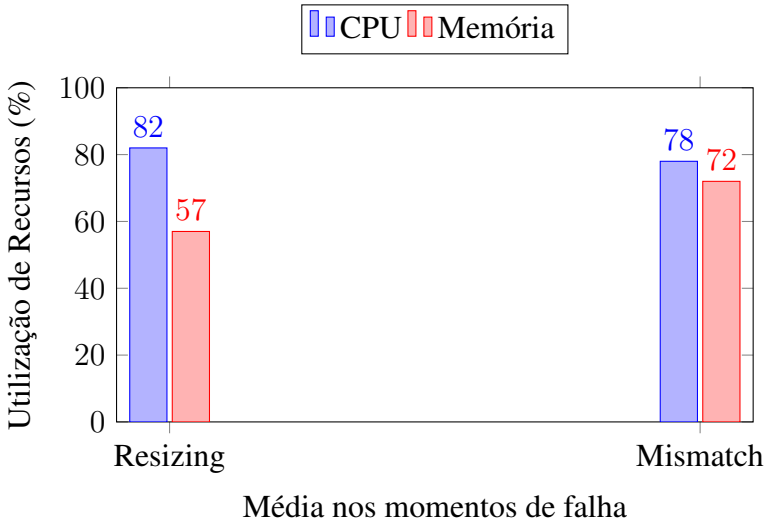

Figura 4. Valores médios de utilização de CPU e memória nos momentos de ocorrência de falhas dos tipos *resizing* e *mismatch*.

De forma geral, os resultados obtidos indicam que a abordagem proposta é eficaz na identificação de padrões anômalos em logs do *Openstack*, especialmente pela capacidade de separar eventos normais e falhos no espaço de representação, conforme evidenciado pelos valores de AUC. Embora tenham sido observadas diferenças entre as métricas de avaliação, os experimentos demonstram que o modelo é consistente na detecção de comportamentos suspeitos, mesmo em um contexto não supervisionado e com dados não estruturados. Além disso, a análise evidencia que o desempenho está diretamente relacionado às características das falhas e à forma como elas se manifestam nos logs. Nesse sentido, os resultados obtidos reforçam o potencial da abordagem como uma solução viável para apoio à detecção de anomalias em ambientes de nuvem, podendo ser aprimorada com ajustes nas etapas de pré-processamento e calibração do critério de decisão.

5. Conclusão

Os resultados da detecção de falhas na plataforma de nuvem *OpenStack*, obtidos com a abordagem proposta baseada na combinação de *Skip-Gram* com *Negative Sampling* e *K-Means*, demonstram a capacidade do método em identificar padrões comportamentais relevantes nos *logs* do sistema. A utilização de representações distribuídas possibilitou capturar relações contextuais entre os termos presentes nos registros, permitindo a construção de vetores semânticos para cada linha de log. A partir dessas representações, o algoritmo de agrupamento identificou padrões predominantes de comportamento, enquanto registros com maior distância em relação aos centróides dos *clusters* foram classificados como anomalias. Nos experimentos, as falhas introduzidas resultaram em alterações detectáveis nesses padrões, refletidas na separação observada no espaço vetorial e evidenciada pelos valores de AUC obtidos.

Além disso, os resultados obtidos demonstram, de forma quantitativa, a capacidade do modelo proposto na detecção de falhas, mesmo utilizando uma abordagem de

menor complexidade computacional. Nos experimentos realizados, o método alcançou AUC de 0,89 para a falha *resizing* e 0,857 para *mismatch*, evidenciando a capacidade de separação entre eventos normais e anômalos no espaço de pontuação. Considerando o limiar de decisão adotado, foram obtidas acurácias de 87% para a falha *resizing* e 83% para a falha *mismatch*, respectivamente, indicando que a maior parte dos registros foi corretamente classificada. Destaca-se ainda que, no cenário *mismatch*, o modelo atingiu *Recall* de 1,0, identificando 100% das falhas reais presentes nos *logs*. Em contrapartida, no cenário *resizing*, o *Recall* de 0,10 indica maior dificuldade na detecção completa das falhas, apesar da elevada capacidade discriminativa observada via AUC. Esse comportamento evidencia que o modelo permite a separação de padrões, mas sensível à escolha do limiar de decisão. Ainda assim, os resultados confirmam que a estratégia não supervisionada, baseada exclusivamente em *logs*, é capaz de identificar padrões anômalos relevantes sem a necessidade de dados rotulados, mantendo desempenho consistente mesmo em cenários distintos de falha.

Como trabalhos futuros, pretende-se investigar o uso de modelos baseados em transformadores, como BERT, para geração de *embeddings* mais ricos, capazes de capturar dependências de longo alcance e relações semânticas mais complexas nos *logs*. Essas abordagens não foram adotadas neste trabalho devido ao maior custo computacional, à necessidade de grandes volumes de dados e à maior complexidade de ajuste. Além disso, pretende-se avaliar a metodologia proposta em plataformas de nuvem como *Amazon Web Services* e *Google Cloud Plataform*, visando analisar sua robustez e capacidade de generalização em diferentes ambientes.

6. Referências

Duarte, G. S. and Sousa, E. T. G. d. (2024). Cimut: Ferramenta de injeção de falhas em ambientes de nuvens por mutação. *Trabalho de Conclusão de Curso (Graduação)*. Estudo experimental realizado na plataforma OpenStack.

He, Z., Wang, R., and Liu, F. (2024). Logllm: Leveraging large language models for log analysis and anomaly detection. *arXiv preprint arXiv:2402.01234*.

Kim, D., Park, J., and Lee, H. (2025). Loggpt: Enhancing log anomaly detection using generative pre-trained transformers. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 95–107.

Leshem, I. (2020). Skip-gram word2vec algorithm explained. `https://leshem-ido.medium.com/skip-gram-word2vec-algorithm-explained-85cd67a45ffa`. Medium. Acesso em: 13 mar. 2026.

Li, H., Chen, R., and Wang, T. (2025). Graph-based anomaly detection for cloud systems using structural embeddings. In *Proceedings of the IEEE International Conference on Cloud Computing*, pages 312–319.

Liu, P., Zhao, K., and Sun, W. (2025). Intelligent failure detection in cloud systems using deep learning and telemetry analysis. In *IEEE International Conference on Cloud Computing*, pages 210–217.

Mohammed, A. K. et al. (2025a). Predictive failure detection in cloud infrastructure using multivariate telemetry log analysis with temporal convolution and attention-

based deep learning. In *International Conference on Cyber Resilience (ICCR)*, pages 1–8.

Mohammed, E. E., Naji, R. Y. S., Hussein, A. A., Saeed, M. A., and Al Selwi, R. A. M. (2025b). Anomaly detection system for secure cloud computing environment using machine learning. In *2025 5th International Conference on Emerging Smart Technologies and Applications (eSmarTA)*, pages 1–9.

Rana, O. F., Dustdar, S., et al. (2024). Multi-cloud computing: Architecture, challenges, and applications. *IEEE Cloud Computing*, 11(1):50–60.

Wang, H., Li, J., and Zhang, Y. (2024). A survey on aiops: Techniques, applications, and challenges. *ACM Computing Surveys*, 56(3):1–36.

Wu, J., Zhang, K., and Liu, Y. (2024). Graph neural networks for anomaly detection in distributed systems: A survey. *IEEE Access*, 12:33456–33472.

Zhang, T., Xu, H., and Li, C. (2024). Transformer-based log anomaly detection with self-supervised learning. *IEEE Transactions on Network and Service Management*, 21(2):789–802.

Zhou, L., Chen, M., and Wu, Q. (2024). Cloud-native systems: Architecture, challenges, and future trends. *Future Generation Computer Systems*, 150:45–60.