Uma análise comparativa entre Skip-Gram e o CBOW com a utilização de métodos não supervisionados para detecção de anomalias em ambientes de nuvem

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

[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

Resumo. *O trabalho compara métodos não supervisionados para detecção de falhas em logs do OpenStack utilizando embeddings. A combinação Skip-Gram + K-Means apresentou o melhor desempenho (AUC até 0,89, acurácia de 87% e recall de 1,0), enquanto Skip-Gram + Isolation Forest não detectou as falhas avaliadas. Já as abordagens baseadas em CBOW obtiveram resultados inferiores, com AUC entre 0,46 e 0,73. Os resultados indicam que a qualidade dos embeddings é determinante para a detecção de anomalias.*

1. Introdução

A crescente adoção de infraestruturas em nuvem dinâmicas e distribuídas tem ampliado os desafios de observabilidade, monitoramento e diagnóstico de falhas devido à presença de microsserviços, elasticidade de recursos e arquiteturas heterogêneas [Chen et al. 2024, Li et al. 2024a].

Nesse contexto, a detecção automática de anomalias tornou-se essencial, motivando o uso de técnicas de aprendizado de máquina, séries temporais e modelos de linguagem aplicados à análise de logs de sistema. Apesar dos avanços, essas abordagens ainda enfrentam limitações relacionadas ao custo computacional, à latência, à dependência de grandes volumes de dados e à capacidade de generalização [Qin et al. 2024, Zhang et al. 2024, Liu et al. 2023, Jiang et al. 2024].

Diante desse contexto, este trabalho propõe uma análise comparativa para detecção de falhas em nuvem, combinando modelos de *embeddings Skip-Gram* com *K-Means*, *CBOW* com *K-Means* e *CBOW* com *Isolation Forest*. A detecção baseia-se na análise das representações no espaço vetorial, considerando proximidade e isolamento para identificar desvios no comportamento do sistema.

2. Trabalhos Relacionados

A detecção de falhas em ambientes de computação em nuvem tem sido estudada sob diferentes abordagens, incluindo métodos estatísticos, não supervisionados e baseados em grafos, além de técnicas tradicionais de aprendizado de máquina. Essas estratégias buscam identificar padrões anômalos em dados de telemetria e logs, mas ainda enfrentam desafios relacionados à heterogeneidade dos dados, alta dimensionalidade e adaptação a ambientes dinâmicos [Qin et al. 2024, Li et al. 2024b].

LSTMs, Transformers e técnicas de Aprendizado Contrastivo têm sido utilizados para capturar dependências temporais e gerar representações robustas de logs,

superando métodos tradicionais na detecção de falhas, mesmo com poucos dados de treinamento [Xu et al. 2026, Zhang et al. 2026, Wang et al. 2025]. Essas abordagens integram métricas heterogêneas e reduzem falsos positivos em ambientes dinâmicos [Liu et al. 2026].

Apesar dos avanços, muitas abordagens apresentam alto custo computacional, dependência de grandes volumes de dados e complexidade de implantação. Nesse contexto, métodos não supervisionados baseados em logs surgem como alternativas mais simples. Assim, este trabalho propõe uma análise comparativa do uso de *Skip-Gram* e *CBOW* combinados com *K-Means* e *Isolation Forest*, respectivamente, para detecção de anomalias.

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

Esta seção apresenta a metodologia proposta para detecção de falhas em ambientes de nuvem, baseada na vetorização de logs e no uso de aprendizado não supervisionado. O processo é composto por quatro fases: configuração do ambiente, injeção de falhas, vetorização dos logs e detecção de anomalias, conforme ilustrado na Figura 1.

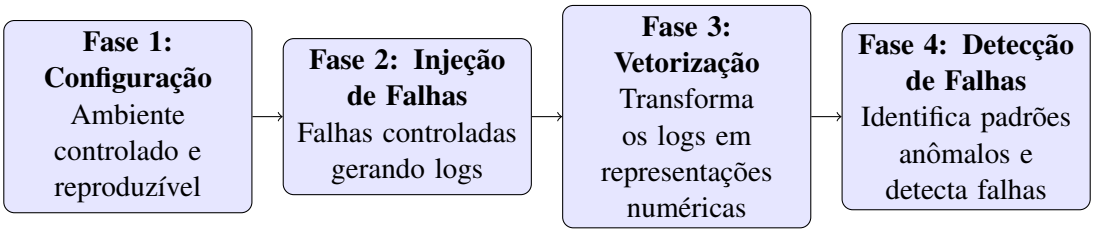

Figura 1. Etapas da metodologia proposta

3.1. Fase 1: Configuração do Ambiente

Inicialmente, foi estabelecido um ambiente controlado e reproduzível, permitindo a execução dos experimentos e o monitoramento do comportamento do sistema diante das falhas injetadas.

3.2. Fase 2: Injeção de Falhas

A geração de dados foi realizada por meio da injeção controlada de falhas em um ambiente *OpenStack*, utilizando a ferramenta CIMut [Duarte and Sousa 2024]. A abordagem baseia-se na mutação de código-fonte em arquivos remotos via SSH, permitindo reproduzir falhas reais a partir de relatórios do Bugzilla. Após as modificações, o ambiente é reiniciado para que as alterações entrem em vigor, e os eventos gerados são registrados em logs para posterior análise.

3.3. Fase 3: Vetorização

Os logs são convertidos em representações numéricas por meio de embeddings gerados pelos modelos Skip-Gram e CBOW, treinados com Negative Sampling. Após a construção do vocabulário e cálculo da frequência das palavras, ambos utilizam dimensão 200, taxa de aprendizagem de 0,001, 10 amostras negativas, lotes de 512 pares e 3 épocas. O Skip-Gram emprega janela de contexto 8, enquanto o CBOW utiliza janela dinâmica de até 2 palavras e descarta termos com frequência inferior a duas ocorrências. Os vetores são então agregados por média aritmética e normalizados, gerando as representações dos logs para detecção de falhas.

3.4. Fase 4: Detecção de Falhas

A detecção de falhas é realizada a partir dos *embeddings* gerados, utilizando algoritmos não supervisionados: *K-Means* e *Isolation Forest*. O *K-Means* identifica anomalias com base na distância aos centróides dos clusters, enquanto o *Isolation Forest* baseia-se no princípio de isolamento de instâncias.

Ambos os métodos são aplicados sobre *embeddings* gerados por *Skip-Gram* e *CBOW*, permitindo avaliar o impacto das representações vetoriais. A implementação foi realizada com a biblioteca *scikit-learn* na versão 1.6.1.

4. Estudo de Caso

O estudo de caso avalia a metodologia proposta por meio da injeção de falhas na plataforma *OpenStack*. Logs e métricas são representados por embeddings gerados pelos modelos *Skip-Gram* e *CBOW*, combinados aos algoritmos *K-Means* e *Isolation Forest* para identificação de padrões anômalos. Por fim, as abordagens são comparadas quanto ao desempenho na detecção de falhas.

4.1. Configuração do Ambiente

O ambiente de nuvem privada foi implementado com o *OpenStack* 8.2.0, instalado via *DevStack* em uma arquitetura não distribuída. A configuração incluiu os serviços *Keystone*, *Glance*, *Nova*, *Placement*, *Neutron*, *Cinder* e *Horizon*, sem cargas de trabalho ativas durante os experimentos. A plataforma foi executada em um *host* com processador *Intel Core i5-3330S*, 8 GB de memória RAM, SSD SATA de 480 GB e Ubuntu 24.04.3 LTS.

4.2. Injeção de Falhas

A obtenção de dados foi realizada por meio da injeção controlada de falhas em um ambiente *OpenStack*, utilizando a ferramenta CIMut [Duarte and Sousa 2024]. A abordagem baseia-se na mutação de código-fonte em arquivos remotos via SSH, na qual linhas específicas são validadas e modificadas para refletir falhas reais reportadas no *Bugzilla*. Após a alteração e reenvio do arquivo ao *host*, o ambiente é reiniciado para aplicação das mudanças, e os eventos gerados são registrados em logs para posterior análise.

4.3. Discussão de Resultados

Esta seção analisa os resultados da detecção de falhas na plataforma *OpenStack* utilizando as combinações *Skip-Gram* e *CBOW* com os algoritmos *K-Means* e *Isolation Forest*.

A combinação do *Skip-Gram* com *K-Means* apresentou o melhor desempenho, alcançando AUC de 0,89 e acurácia de 87% em *resizing*, e AUC de 0,857 com acurácia de 83% em *mismatch*, além de *recall* igual a 1,0. Esses resultados indicam que os *embeddings* gerados pelo *Skip-Gram* favorecem a separação entre eventos normais e anômalos. Em contraste, o *Skip-Gram* com *Isolation Forest* não detectou as falhas avaliadas, pois os eventos anômalos formaram agrupamentos no espaço vetorial, deixando de ser tratados como observações isoladas.

As abordagens baseadas em *CBOW* apresentaram desempenho inferior, refletindo a menor capacidade do modelo em capturar diferenças sutis nos logs. Com *K-Means*, foram obtidos AUC de 0,59 e *recall* de 0,92 para *resizing*, e AUC de 0,46 com *recall* de 0,47

para *mismatch*. Já o *CBOW* com *Isolation Forest* apresentou desempenho intermediário, alcançando AUC de 0,73 e *recall* de 0,78 em *resizing*, e AUC de 0,63 com *recall* de 1,0 em *mismatch*, embora com limitada separação entre as classes.

A Figura 2 sintetiza os resultados, evidenciando o desempenho superior de *Skip-Gram* com *K-Means*, especialmente na falha *mismatch*.

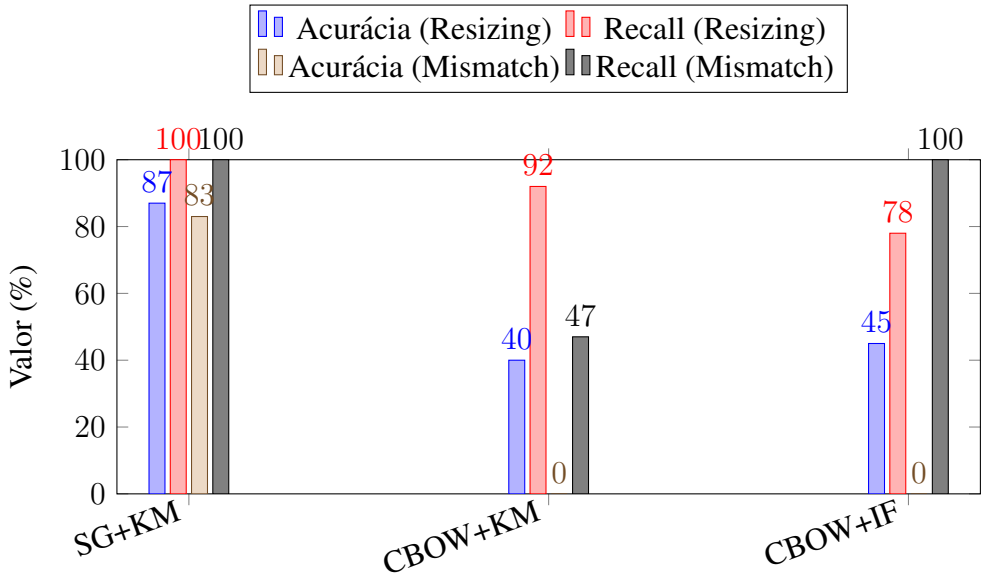

Figura 2. Comparação de acurácia e recall para as falhas resizing e mismatch

As falhas de *resizing* e *mismatch* apresentaram comportamento anômalo principalmente em CPU, memória e rede. A CPU exibiu elevada variabilidade (desvios padrão de 27,72% e 29,21%) e picos de utilização de até 100%. O cenário de *mismatch* registrou maiores médias de CPU (33,33% versus 24,93%) e memória (67,81% versus 52,64%), além de maior dispersão da memória (17,23% versus 7,51%). O disco permaneceu estável, enquanto a rede apresentou elevada dispersão e forte assimetria. A correlação entre CPU e memória foi fraca (r = 0,11 e r = 0,137), e os intervalos de confiança de 95% corroboraram a maior variabilidade observada nos recursos computacionais e de rede. Esses resultados indicam uma associação entre a ocorrência das falhas e alterações significativas no consumo de recursos, evidenciada pelo aumento da variabilidade, pelos picos de utilização e pela dispersão das métricas operacionais durante os eventos de falha. Embora não permitam estabelecer causalidade, tais evidências sugerem que o monitoramento de recursos pode fornecer indícios relevantes da ocorrência de anomalias. Além disso, como essas alterações tendem a ser refletidas nos logs do sistema, elas contribuem para a formação de padrões distintos que podem ser explorados por métodos de detecção de falhas.

Os resultados evidenciam que a qualidade dos *embeddings* é determinante para a detecção de anomalias. O Skip-Gram gerou representações mais discriminativas que o CBOW, favorecendo a separação entre eventos normais e anômalos. Consequentemente, a combinação Skip-Gram + K-Means apresentou o melhor desempenho, indicando que a eficácia da detecção depende tanto do algoritmo quanto da representação vetorial utilizada.

5. Conclusão

Este trabalho investigou a detecção de falhas em logs do OpenStack utilizando abordagens não supervisionadas baseadas em embeddings. Os resultados mostram que a qualidade da representação vetorial é determinante para a detecção de anomalias, destacando-se a combinação Skip-Gram + K-Means (AUC de 0,89 e 0,857; acurácia de 87% e 83% em resizing e mismatch, respectivamente). Já Skip-Gram + Isolation Forest e as abordagens baseadas em CBOW apresentaram desempenho inferior.

De forma geral, a qualidade dos *embeddings* mostrou-se determinante. Como trabalhos futuros, propõe-se o uso de *FastText* e *BERT*, parsing estruturado de logs, limiares adaptativos baseados nos escores de anomalia e comparar os resultados com abordagens, como: *LogBERT*, *LogAnomaly*, *DeepLog*, *Autoencoders* e *One-Class SVM*.

6. Referências

Chen, M., Zhang, Q., and Xu, L. (2024). Observability in cloud-native systems: Trends and challenges. *IEEE Communications Surveys & Tutorials*, 26(1):123–145.

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.

Jiang, Z., Liu, K., and Zhang, Y. (2024). Exploring large language models for log analysis: Opportunities and challenges. *arXiv preprint arXiv:2402.01234*.

Li, J., Wang, R., and Zhao, P. (2024a). Managing complexity in cloud computing environments: A survey. *Journal of Systems and Software*, 210:111234.

Li, Y., Zhang, H., and Chen, X. (2024b). Graph-based anomaly detection in distributed systems: A survey. *ACM Computing Surveys*, 56(5):1–34.

Liu, P., Zhu, J., He, S., and Lyu, M. (2023). Logbert: Log anomaly detection via bert. In *Proceedings of the International Conference on Software Engineering (ICSE)*.

Liu, X., Garcia, M., and Patel, R. (2026). Multimodal anomaly detection: Integrating heterogeneous metrics in dynamic cloud architectures. *Journal of Network and Systems Management*, 34(1):89–105.

Qin, Y., Wang, Z., and Chen, L. (2024). Time-series anomaly detection in cloud systems: A review. *IEEE Transactions on Network and Service Management*, 21(1):120–135.

Wang, S., Li, Y., and Kim, T. (2025). Contrastive learning for robust anomaly representation in cloud environments. *ACM Transactions on Autonomous and Adaptive Systems*, 20(3):1–25.

Xu, Y., Chen, J., and Wang, H. (2026). Log-driven anomaly detection in microservices using long-range transformer architectures. *IEEE Transactions on Cloud Computing*, 14(2):452–467.

Zhang, C., Liu, Y., and Zhao, M. (2024). A survey on deep learning-based anomaly detection. *ACM Computing Surveys*, 56(3):1–36.

Zhang, L., Zhao, Q., and Liu, B. (2026). Deep sequence modeling for multi-variate telemetry anomaly detection. In *Proceedings of the 2026 IEEE International Conference on Cloud Computing (CLOUD)*, pages 112–120. IEEE.