Aplicação de *Machine Learning* à Predição de Tempo de Execução em FaaS com o *Framework* Orama

Leonardo Rebouças de Carvalho[1], **Geraldo Pereira Rocha Filho**[1,2], **Aleteia Araujo**[1]

[1]Depto de Ciência da Computação - Universidade de Brasilia
Campus Darcy Ribeiro – Brasilia – DF – Brazil

[2]Depto de Ciências Exatas e Tecnológicas, Universidade Estadual do Sudoeste da Bahia
Vitória da Conquista – BA – Brazil.

leouesb@gmail.com, geraldo.rocha@uesb.edu.br, aleteia@unb.br

Resumo. *Um dos principais desafios em Function-as-a-Service (FaaS) é a imprevisibilidade do tempo de execução das funções, o que pode causar aumento de custos e degradação de desempenho em aplicações distribuídas entre provedores de nuvem. Este artigo apresenta um preditor baseado em Machine Learning (ML) integrado ao Framework Orama, que combina métricas estáticas de código (medidas de complexidade de Halstead) e dados empíricos de desempenho para estimar o tempo de execução diretamente a partir do código-fonte. Foram avaliadas três arquiteturas de redes neurais (Dense, LSTM e BLSTM), sendo a BLSTM a que apresentou maior precisão.*

1. Introdução

A computação *Serverless* [Nupponen and Taibi 2020] tornou-se um paradigma central para aplicações baseadas em microsserviços, permitindo execução sob demanda de funções via *Function-as-a-Service* (FaaS) com escalabilidade automática e sem necessidade de gerenciar infraestrutura. Porém, a imprevisibilidade do tempo de execução continua sendo um desafio, especialmente em ambientes *multicloud*, onde pequenas variações podem afetar a experiência do usuário e custos.

O desempenho das funções é influenciado por fatores como complexidade do código, ambiente de execução, memória, políticas de escalonamento e características específicas de provedores como AWS Lambda, Google Cloud Functions, Azure Functions e Alibaba Function Compute. Ferramentas tradicionais de *benchmarking* apresentam limitações de reprodutibilidade e suporte a múltiplos provedores, dificultando comparações e decisões de implantação.

Para contornar essas limitações, este trabalho estende o *Framework* Orama [Carvalho et al. 2024][1] com um preditor de tempo de execução baseado em *Machine Learning* (ML), combinando métricas estáticas de código, principalmente as de *Halstead*, com dados empíricos de *benchmarks*. Três arquiteturas de redes neurais (Dense, LSTM e BLSTM) foram avaliadas, destacando-se a BLSTM ($R^2 = 0,91$ e MSE 20% menor). O preditor foi integrado ao Orama via APIs e interface gráfica, permitindo comparações entre provedores, planejamento de implantações e otimização de custo e desempenho em aplicações *Serverless*.

[1]https://github.com/unb-faas/orama

O restante deste artigo está organizado da seguinte forma: a Seção 2 apresenta os principais conceitos por trás da ferramenta de predição do Orama; a Seção 3 discute os trabalhos relacionados; a Seção 4 descreve a metodologia utilizada para a geração do conjunto de dados e o treinamento dos modelos; a Seção 5 apresenta os resultados experimentais; e a Seção 6 conclui o artigo e indica direções para trabalhos futuros.

2. Fundamentação

A avaliação de desempenho em ambientes FaaS enfrenta desafios como instrumentação, testes automatizados, coleta de métricas e análise estatística em múltiplos provedores. Ferramentas tradicionais de *benchmarking* apresentam limitações de reprodutibilidade, granularidade e suporte *multicloud*. Para contornar isso, o *Framework* Orama foi desenvolvido como uma infraestrutura modular e escalável, integrando módulos de provisão de funções, orquestração, coleta de métricas, análise estatística e geração de relatórios, com suporte a AWS Lambda, Google Cloud Functions, Azure Functions e Alibaba Function Compute. Os dados coletados serviram de base para treinar os modelos de predição de tempo de execução deste trabalho.

O ML é usado no Orama para estimar o tempo de execução a partir do código-fonte, com aprendizado supervisionado e regressão. Foram testadas três arquiteturas de redes neurais, tais como: Dense; LSTM e BLSTM, sendo que LSTM e BLSTM exploram dependências sequenciais e contextos bidirecionais nas métricas do código. O pré-processamento incluiu tratamento de valores ausentes, codificação de variáveis categóricas, normalização, remoção de outliers e análise de correlação para garantir consistência e qualidade dos dados.

A extração de métricas de código-fonte é essencial para transformar software em dados estruturados para ML. As métricas lexicais de *Halstead* destacam-se por analisar o código estaticamente, capturando tamanho de vocabulário, volume e esforço cognitivo, oferecendo visão mais detalhada que complexidade ciclomática ou contagem de linhas. Sua natureza estática e agnóstica a linguagens torna-as ideais para aplicações escaláveis, como no Orama, estabelecendo a base para preditores precisos e robustos de desempenho em FaaS.

3. Trabalhos relacionados

Diversos estudos exploram a predição de tempo de execução em FaaS, como o SLOPE [Tomaras et al. 2023], que usa redes neurais para estimar instâncias, mas depende de contêineres; o FaaStest [Horovitz et al. 2019], que otimiza custo e desempenho sem analisar código-fonte; o TrIMS [Dakkak et al. 2019] e o FuncMem [Pandey and Kwon 2024], que reduzem latência e melhoram throughput, mas sem métricas de código; e o ML-FaaS [Filippini et al. 2025], que prevê sobrecarga com precisão, porém sem considerar complexidade do código.

Embora eficazes em otimização de recursos, essas soluções dependem de métricas de infraestrutura ou são limitadas a plataformas específicas. O *Framework* Orama [Carvalho et al. 2024] se diferencia ao unir dados de execução com métricas estáticas de código, permitindo prever tempos de execução em múltiplos provedores e oferecendo suporte prático para decisões de implantação e planejamento *multicloud* em cenários *Serverless*.

4. Metodologia

A construção do preditor de tempo de execução para FaaS no *Framework* Orama iniciou-se com a definição e normalização de um conjunto de casos de uso multiplataforma. Essa padronização garante a comparabilidade entre plataformas heterogêneas, reduz vieses causados por diferenças de nomenclatura ou recursos nativos e permite a reutilização automatizada de experimentos. Foram definidos três tipos de casos de uso: "Calculator", "API for Object Storage" e "API for DBaaS". Todos foram implementados usando o Orama. Para cada implementação, foram extraídas métricas de complexidade de código com base na família de métricas de *Halstead*, por meio do serviço interno *Halsteader*, integrado ao Orama. As métricas utilizadas foram: comprimento, vocabulário, dificuldade, volume e esforço estimado.

Somente métricas de código não capturam variações causadas por infraestrutura, *cold starts*, políticas de escalonamento ou limitações específicas de cada provedor. Para incorporar esses fatores, foram reutilizados resultados empíricos de execuções de *benchmarking* realizadas anteriormente com o Orama, publicadas em estudos prévios, abrangendo tempos de execução sob diferentes níveis de concorrência, tamanhos de carga e configurações regionais. O processo de construção do *dataset* utilizado no treinamento do preditor é composto por quatro macrofases: Experimentos → Resultados Consolidados → Extração de Métricas *Halstead* → *Dataset* final. Os experimentos foram executados de forma controlada e reproduzível nos quatro provedores; os resultados consolidados incluem metadados contextuais (região, carga, número de invocações) e tempos medidos; e, por fim, as métricas de *Halstead* são integradas em um *dataset* analítico tabular, pronto para as etapas de pré-processamento e modelagem.

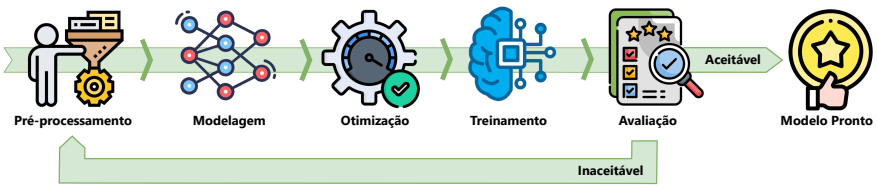

Figura 1. Processo de treinamento do modelo.

A Figura 1 apresenta o *pipeline* de ML usado para construir o preditor, composto pelas etapas: Pré-processamento → Modelagem → Otimização → Treinamento → Avaliação → Decisão. Durante o pré-processamento, são realizadas tarefas como remoção de *outliers*, imputação de valores ausentes, codificação categórica de provedores e normalização de atributos numéricos (incluindo as métricas de *Halstead*). Na modelagem, são avaliadas diferentes famílias de modelos de regressão — Dense, LSTM e BLSTM. A otimização aplica busca multiobjetivo de hiperparâmetros, considerando erro quadrático médio e robustez entre provedores. Após o treinamento, os modelos candidatos são avaliados e, caso atendam aos critérios de desempenho, são congelados e versionados; caso contrário, o ciclo retorna ao pré-processamento para ajustes em engenharia de atributos, filtragem ou balanceamento de amostras.

A arquitetura atualizada do *Framework* Orama incorpora dois novos serviços: (i) *Halsteader*, responsável pela análise estática de código e geração das métricas de *Hals-*

tead; e (ii) *Predictor*, que utiliza o modelo treinado para estimar tempos de execução esperados por provedor e nível de concorrência. Embora o backend ofereça APIs unificadas para automação de experimentos e consulta de predições, identificou-se a necessidade de aprimorar a interface gráfica que permita ao usuário enviar o código-fonte de uma função FaaS, definir parâmetros de carga e obter estimativas comparativas de tempo de execução entre AWS, Google Cloud, Azure e Alibaba Cloud. Essa funcionalidade é essencial para estudos de portabilidade, análise custo-desempenho e planejamento *multicloud*.

5. Resultados

Esta seção apresenta os resultados da avaliação dos modelos de predição de tempo de execução integrados ao *Framework* Orama. O desempenho de três arquiteturas de redes neurais (Dense, LSTM e BLSTM) foi comparado utilizando MSE e R^2. Essas métricas foram calculadas por meio de *cross-validation* no conjunto de validação do *dataset*. Os três modelos apresentaram convergência durante o treinamento, sendo que as arquiteturas Dense e BLSTM demonstraram reduções de perda mais estáveis e rápidas.

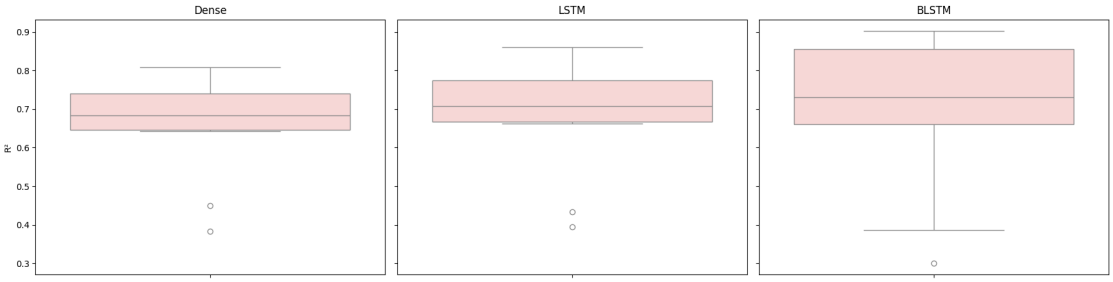

Figura 2. R^2 Boxplot for Dense, LSTM, and BLSTM models.

A Figura 2 apresenta *boxplots* dos valores de R^2 para cada modelo. O modelo BLSTM obteve a menor mediana de MSE e o maior R^2, indicando superior precisão e capacidade de generalização entre casos de uso e provedores de nuvem. O modelo Dense apresentou desempenho próximo, enquanto o LSTM mostrou erros de predição ligeiramente maiores e maior variabilidade, provavelmente devido à sua sensibilidade ao comprimento e à complexidade da sequência de entrada.

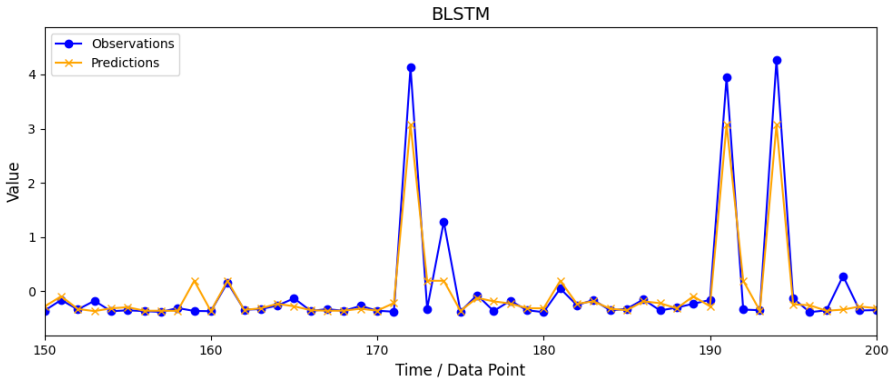

Figura 3. Observations vs. Predictions in BLSTM model.

A Figura 3 apresenta os tempos observados versus os previstos para o modelo com melhor desempenho (BLSTM). Os pontos de dados alinham-se de forma próxima à dia-

gonal, confirmando forte concordância preditiva. Pequenas variações são observáveis em casos extremos, com alta complexidade ou combinações incomuns de métricas de *Halstead* e comportamentos de provedores; entretanto, o preditor mantém robustez ao longo do *dataset*. Os resultados confirmam a eficácia do uso de métricas lexicais de código para prever tempos de execução em FaaS. Entre os modelos avaliados, a arquitetura BLSTM destacou-se pela maior precisão e capacidade de generalização, sendo adotada como base para o novo componente *Predictor* no *Framework* Orama.

6. Conclusão e trabalhos futuros

Este trabalho apresentou um preditor de tempo de execução para funções *Serverless* integrado ao *Framework* Orama, combinando métricas de *Halstead* e dados de *benchmarking* para estimar desempenho diretamente do código-fonte. Entre os modelos testados, o BLSTM se destacou ($R^2 = 0,91$, MSE 20% menor que as *baselines*), e o preditor já está disponível via APIs e interface gráfica, permitindo estimativas em tempo real por provedor.

Como próximos passos, planeja-se expandir o suporte a mais linguagens e provedores, incluir predição de custos e explorar arquiteturas avançadas, como BERT e LLaMA, para melhorar a interpretação do código. Além disso, a ampliação do dataset e o aprimoramento da interface gráfica devem tornar a ferramenta mais versátil e acessível.

Referências

Carvalho, L. R. d., Kamienski, B., and Araujo, A. (2024). Main FaaS Providers Behavior Under High Concurrency: An Evaluation with Orama Framework Distributed Architecture. *SN Computer Science*, 5(5):541.

Dakkak, A., Li, C., Garcia de Gonzalo, S., Xiong, J., and Hwu, W.-m. (2019). Trims: Transparent and isolated model sharing for low latency deep learning inference in function-as-a-service. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 372–382.

Filippini, F., Cavenaghi, L., Calmi, N., Savi, M., and Ciavotta, M. (2025). Ml-based performance modeling in edge faas systems. In *European Conference on Service-Oriented and Cloud Computing*, pages 112–127. Springer.

Horovitz, S., Amos, R., Baruch, O., Cohen, T., Oyar, T., and Deri, A. (2019). Faastest - machine learning based cost and performance faas optimization. In Coppola, M., Carlini, E., D'Agostino, D., Altmann, J., and Bañares, J. Á., editors, *Economics of Grids, Clouds, Systems, and Services*, pages 171–186, Cham. Springer International Publishing.

Nupponen, J. and Taibi, D. (2020). Serverless: What it is, what to do and what not to do. In *2020 IEEE ICSA-C*, pages 49–50.

Pandey, M. and Kwon, Y.-W. (2024). Funcmem: reducing cold start latency in serverless computing through memory prediction and adaptive task execution. In *Proceedings of the 39th ACM/SIGAPP symposium on applied computing*, pages 131–138.

Tomaras, D., Tsenos, M., and Kalogeraki, V. (2023). Prediction-driven resource provisioning for serverless container runtimes. In *2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pages 1–6.