UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO DE CIÊNCIAS, TECNOLOGIAS E SAÚDE DO CAMPUS ARARANGUÁ
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

Ricardo Santacattarina

Implementação de uma API em arquitetura serverless no ambiente da AWS para predição de batimetria em estuários através da aplicação de métodos de aprendizado de máquina em imagens multiespectrais de satélites

Araranguá
2021

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Santacattarina, Ricardo
Implementação de uma API em arquitetura serverless no ambiente da AWS para predição de batimetria em estuários através da aplicação de métodos de aprendizado de máquina em imagens multiespectrais de satélites / Ricardo Santacattarina ; orientador, Antonio Carlos Sobieranski, coorientador, Antonio Henrique da Fontoura Klein, 2021.
27 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Araranguá, Graduação em Engenharia de Computação, Araranguá, 2021.

Inclui referências.

1. Engenharia de Computação. 2. Batimetria derivada por satélite. 3. Arquitetura serverless. 4. Implementação de API. I. Sobieranski, Antonio Carlos. II. Klein, Antonio Henrique da Fontoura. III. Universidade Federal de Santa Catarina. Graduação em Engenharia de Computação. IV. Título.

Documento assinado digitalmente
Fabricio de Oliveira Ourique
Data: 21/05/2021 13:38:16-0300
CPF: 916.167.860-00
Verifique as assinaturas em https://v.ufsc.br

Prof. Fabrício De Oliveira Ourique, Dr.
Coordenador do Curso

Banca Examinadora:

Documento assinado digitalmente
Antonio Carlos Sobieranski
Data: 21/05/2021 12:25:52-0300
CPF: 005.305.809-77
Verifique as assinaturas em https://v.ufsc.br

Prof. Antônio Carlos Sobieranski, Dr.
Orientador

Documento assinado digitalmente
Antonio Henrique da Fontoura Klein
Data: 21/05/2021 13:56:58-0300
CPF: 696.210.710-20
Verifique as assinaturas em https://v.ufsc.br

Prof. Antônio Henrique da Fontoura Klein, Dr.
Co-Orientador

Documento assinado digitalmente
Anderson Luiz Fernandes Perez
Data: 21/05/2021 13:03:39-0300
CPF: 020.803.459-58
Verifique as assinaturas em https://v.ufsc.br

Prof. Anderson Luiz Fernandes Perez, Dr.
Avaliador

Luís Pedro Melo de Almeida

Prof. Luis Pedro Melo de Almeida, Dr.
Avaliador

Prof. Fábio Rodrigues De La Rocha, Dr.
Avaliador Suplente

Implementação de uma API em arquitetura *serverless* no ambiente da AWS para predição de batimetria em estuários através da aplicação de métodos de aprendizado de máquina em imagens multiespectrais de satélites

Serverless API implementation on AWS for estuary bathymetry prediction through machine learning techniques using multispectral satellite imagery

Ricardo Santacattarina [*] Vinicius Camozzato Vaz [†]
Luis Pedro Melo de Almeida [‡] Antonio Henrique da Fontoura Klein [§]
Antônio Carlos Sobieranski [¶]

2021, Maio

Resumo

O processo de batimetria tem como objetivo medir a profundidade da coluna d'água e é de suma importância para eficiência e segurança da navegação, mapeamento de tipo de perfis de costa, como também para a construção civil em regiões costeiras. Geralmente é feito através da medição *in situ*, com o uso de equipamentos como ecobatímetros, porém estes métodos são limitados pela capacidade de navegação na área de interesse e pelo custo, oportunizando alternativas como a batimetria derivada por satélite. O presente trabalho apresenta as especificações e etapas de implementação de uma API no ambiente da *Amazon Web Services* (AWS) utilizando da arquitetura *serverless*, possibilitando treinamento e utilização de modelos de aprendizado de máquina capazes de estimar a profundidade da coluna d'água a partir dos valores das bandas de reflectância obtidas por imagens de satélite. Adicionalmente, o presente trabalho expõe as vantagens relacionadas à computação *serverless*, permitindo mitigar os desafios de esforço operacional e computacional, efetivando a integração de diferentes etapas para a solução proposta, como processamento de dados, técnicas de aprendizado de máquina e aquisição de dados de sensoriamento remoto. Resultados experimentais da implementação em ambiente *serverless* da abordagem proposta neste trabalho

[*] ricardo.santacattarina@gmail.com
[†] vinicvaz95@gmail.com
[‡] melolp@gmail.com
[§] antonio.klein@ufsc.br
[¶] a.sobieranski@ufsc.br

demonstraram a efetividade, facilidade de uso e gestão e escalabilidade, do modelo de batimetria por imagens de satélite no contexto da Baía da Babitonga para um sistema externo.

Palavras-chaves: *serverless*, API, batimetria derivada por satélite, sensoriamento remoto, aprendizado de máquina.

Implementação de uma API em arquitetura *serverless* no ambiente da AWS para predição de batimetria em estuários através da aplicação de métodos de aprendizado de máquina em imagens multiespectrais de satélites

Serverless API implementation on AWS for estuary bathymetry prediction through machine learning techniques using multispectral satellite imagery

Ricardo Santacattarina [*] Vinicius Camozzato Vaz [†]
Luis Pedro Melo de Almeida [‡] Antonio Henrique da Fontoura Klein [§]
Antônio Carlos Sobieranski [¶]

2021, Maio

Abstract

The bathymetry process aims to measure the depth of the water column over bodies of water and is of crucial importance for navigation efficiency and safety, mapping coast profiles' types, as well as for civil construction in coastal regions. It is usually done by *in situ* measurement, with the use of equipment such as echo sounders, however these methods are limited by the navigation capacity in the area of interest and operational cost, allowing alternatives such as satellite-derived bathymetry. This work presents the steps of implementing an API in the Amazon Web Services (AWS) environment using the serverless architecture, enabling training and usage of machine learning models capable of estimating the depth of the water column based on values of the reflectance bands acquired from satellite images. It also exposes the advantages related to serverless computing, overcoming the challenges of operational and computational effort, accomplishing the integration of different steps for the proposed solution, such as data processing, machine learning techniques and remote sensing data acquisition. Experimental results of the implementation in a serverless environment of the proposed approach in this work demonstrated the effectiveness, ease of use, management and

[*] ricardo.santacattarina@gmail.com
[†] vinicvaz95@gmail.com
[‡] melolp@gmail.com
[§] antonio.klein@ufsc.br
[¶] a.sobieranski@ufsc.br

scalability of the bathymetry model by satellite images in the context of the Babitonga Bay for an external system.

Key-words: serverless, API, satellite derived bathymetry, remote sensing, machine learning.

1 Introdução

Batimetria é caracterizada pelo estudo da profundidade do fundo de oceanos, rios e lagos em relação à superfície do espelho d'água. É uma técnica bastante empregada e de suma importância para eficiência e segurança da navegação, mapeamento de tipo de perfis de costa, assim como para a construção civil em regiões costeiras (GAGG, 2016). O processo de realização da batimetria geralmente é realizada através da utilização de ecobatímetros, aparelhos que calculam a distância do sensor de acordo com o tempo transcorrido da transmissão de um pulso sonoro, até o recebimento do mesmo após ter refletido no fundo da coluna d'água (FERREIRA, 2016). No entanto, o uso de ecobatímetros é limitado pela capacidade de navegação na área de interesse e pelo elevado custo de operação, assim dando lugar às alternativas modernas de análise de imagem, como a Batimetria Derivada por Satélite (BDS).

A BDS, como o próprio nome sugere, é uma modalidade de obtenção da batimetria que faz uso de dados de sensoriamento remoto para a estimativa de profundidade com base em imagens. A profundidade da coluna d'água pode ser estimada através da relação entre as diferentes bandas de reflectâncias, possibilitando desta forma uma estimativa bastante aproximada para a obtenção da profundidade local (PHILPOT, 1989). Várias técnicas e abordagens computacionais numéricas podem ser utilizadas para estimativa da profundidade a partir das imagens obtidas. Dentre estas, técnicas de aprendizado de máquina podem ser empregadas para buscar correlações entre os padrões de reflectância com medições realizadas in-loco por técnicas convencionais (tal como ecobatímetro), e ajustar os valores de relação de reflectância obtidos com a profundidade do local, conforme as características específicas do mesmo (transparência da água, índice de material particulado em suspensão, dentre outros). Desta forma, é possível determinar um modelo computacional capaz de reconhecer padrões existentes nas bandas multi-espectrais, e estimar a profundidade em relação a padrões de entrada, mesmo em dados que não participaram do conjunto de treinamento.

Na literatura é possível verificar que grande parte das técnicas de aprendizado de máquina existentes para BDS são baseadas em alguma estratégia de ajuste de curva em relação a um dado de entrada, caracterizadas por regressões lineares, quadráticas ou de ordem maior (polinomial) (FILIPPI, 2020) (GEYMAN; MALOOF, 2019). Outras técnicas ainda se utilizam de estratégias de modelos de regressão baseados em grupos e análise de *clusters*, obtendo a predição final a partir da média ponderada de diferentes modelos específicos (GEYMAN; MALOOF, 2019). Ainda, existem as classes de métodos baseadas em modelos conexionistas, tais como modelos gerados por redes neurais artificiais, que podem atingir precisões superiores, porém devido à sua natureza, também apresentam maior custo computacional de treinamento (KE et al., 2017). No entanto, dependendo da complexidade da característica local a ser analisada, o dado de entrada pode se apresentar de forma não-linear, podendo assim ser de complexa caracterização pelas técnicas anteriormente mencionadas, necessitando de classificadores ou preditores mais robustos para correlacionar os padrões complexos de reflectância observados. O que geralmente se observa é que um

dado modelo pode não ser facilmente generalizável a outros padrões de entrada, visto a alta dependência da informação de translucidez e transparência da água, restringindo a classificação às especifidades do conjunto de treinamento local utilizado (SAGAWA et al., 2019).

No entanto, para aplicações genéricas e de propósitos científicos como no caso do BDS, observa-se um problema de ordem operacional e computacional: todo o processo de preparação e aquisição dos dados, definição do modelo de predição, treinamento e execução, torna o processo custoso computacionalmente e operacionalmente. É necessária infraestrutura de *hardware* adequada para a elaboração e execução dos modelos anteriormente mencionados de forma eficiente. Com a arquitetura *serverless*, esta etapa da implementação de serviços torna-se bastante facilitada, servindo de alternativa às opções convencionais de instalação e gerência de um próprio servidor ou mesmo locação de servidor na nuvem. Na arquitetura *serverless*, o provedor é responsável pela configuração e alocação de recursos, que, ao invés de inferir em custos periódicos pela aquisição ou locação de um servidor próprio, acarretará em cobrança pelo que somente for de fato utilizado (ADZIC; CHATLEY, 2017)(SHAFIEI; KHONSARI; MOUSAVI, 2019).

Este artigo utiliza-se dos resultados experimentais de estudos realizados sobre técnicas de aprendizado de máquina aplicadas na solução da BDS, para a implementação de uma API (*Application Programming Interface*) em arquitetura *serverless*. A motivação para a implementação em uma arquitetura *serverless* é fornecer uma camada de comunicação para o CASSIE, um sistema web de código aberto que possibilita a análise e mapeamento da linha de costa com imagens de sensoriamento remoto (ALMEIDA et al., 2021). A camada de integração, foco principal deste trabalho, considera a especificação e implementação de uma API em uma arquitetura *serverless* para treinamento e uso do modelo de predição de BDS, anteriormente elaborado nos estudos. O modelo de predição mencionado utiliza a estratégia *LightGradient Boosting Machine* (LGBM), um *framework* de aprendizado de máquina baseado em árvores de decisão, e obteve uma precisão de r^2 de 0,94 para um conjunto de entrada de dados da Baía da Babitonga. Resultados experimentais da abordagem computacional proposta demonstraram que a especificação e implementação do modelo preditivo de BDS foi possível em uma arquitetura *serverless*, fazendo uso de vantagens tais como infraestrutura, operacionalização, custos e escalabilidade. Através desta API, são fornecidos suplementos para a integração com o CASSIE, assim como para permitir a realização de BDS em todas as regiões do globo, possibilitando a escalabilidade e uso do sistema.

O restante deste artigo está organizado da seguinte forma: Na Seção 2 são apresentados os trabalhos correlatos na área de implementações de aplicações envolvendo sensoriamento remoto e/ou em arquitetura *serverless*. A Seção 3 apresenta a fundamentação teórica básica necessária para melhor compreensão dos aspectos de aprendizado de máquina e da arquitetura *serverless*. A Seção 4 descreve a metodologia empregada para o desenvolvimento da solução proposta, demonstrando fluxo computacional, arquitetura e algoritmos utilizados. A Seção 5 discute os resultados obtidos e validação, finalizando com a Seção 6, com as conclusões e discussões obtidas acerca do trabalho desenvolvido, assim como possíveis trabalhos futuros.

2 Trabalhos Correlatos

Uma revisão na literatura revelou que não foi possível encontrar até a presente data trabalhos correlatos que abordem um modelo preditivo de BDS com enfoque a detalhes de implementação e uso de arquiteturas *serverless*. Diante desta carência da literatura e possibilidade de uma comparação com a proposta apresentada neste artigo, a presente seção será categorizada em 3 classes distintas: (i) modelos de BDS ou técnicas de aprendizado de máquinas próximos do contexto de aplicação supracitado; (ii) implementação de arquiteturas em nuvem ou *serverless* para áreas de georreferenciamento ou para propósitos de *workflows* científicos; (iii) arquiteturas *serverless* com enfoque em propósitos mais gerais e comerciais, sendo estas consideradas as etapas subsequentes do processo de maturação de software em estágios mais avançados, tal qual proposto neste trabalho.

Quanto aos trabalhos que descrevem (i) técnicas de BDS, na literatura podem ser destacadas:

- (GEYMAN; MALOOF, 2019): Análise de diferentes métodos para estimar a profundidade da água em diferentes regiões dos Bancos das Bahamas, utilizando imagens do satélite RapidEye[1] e dados de medição da batimetria do local. Com a presença de diferentes tipos de fundos, foi feito o uso de uma técnica de regressão baseada em grupos, calculando o valor de predição através da média ponderada dos diferentes modelos para cada grupo, obtendo o melhor r^2 dentre os algoritmos propostos, de 0,90.

- (SAGAWA et al., 2019): Desenvolvimento de método com aprendizado de máquina e imagens de satélite multi-temporais para estimativa de profundidade de águas costeiras cristalinas. Concluíram que sistemas de análise de imagens de satélite baseado em computação na nuvem possibilitarão um incremento na pesquisa em diversos campos que requerem aprendizado de máquina e/ou análise de série temporal de imagens. Também notam que para águas menos cristalinas, talvez outro modelo deva ser construído.

- (FILIPPI, 2020): Trabalho realizado com os dados de batimetria da Baía da Babitonga e imagens obtidas do Sentinel 2 com mesma data da medição. Analisou diferentes relações de bandas, usando regressão polinomial para ajustar a profundidade de acordo com os valores de batimetria medidos, na melhor relação das bandas, restringindo a aplicação às regiões de baixa concentração de material particulado em suspensão e profundidade inferior à 20 metros, foi obtido um r^2 de 0,68.

- (CAMOZZATO et al., 2021): Etapa preliminar da abordagem computacional apresentada neste trabalho, aborda a implementação, análise e validação do uso de métodos de aprendizado de máquina para a realização do processo de BDS, utilizando os dados da Baia da Babitonga. Os resultados experimentais para o melhor modelo apresentaram um r^2 de 0,78, melhorando até 0,948 através de etapas de otimização de hiper-parâmetros e *feature-engineering*. A especificação e implementação da API em *serverless* descrita neste trabalho utilizará este preditor como suplemento.

Ainda na primeira categoria de trabalhos correlatos, mais especificamente no sentido de *frameworks* genéricos de aprendizado de máquina próximos ao contexto explorado

[1] <https://earth.esa.int/eogateway/missions/rapideye>

neste trabalho, vários possuem direcionamento para a área de sensoriamento remoto. Nos trabalhos Das, Pant e Bebortta (2020) e Maxwell, Warner e Fang (2018), uma série de aspectos de técnicas de aprendizado de máquina são descritas para sensoriamento remoto tais como a escolha de algoritmos, requerimentos de dados para treinamento, parâmetros e otimizações, dentre outros. No entanto, detalhes de implementação em termos de complexidade de aplicação em uma arquitetura em nuvem ou *serverless* não são descritos neste *survey*.

Quanto aos trabalhos dedicados à (ii) aplicação de alguma arquitetura em nuvem ou *serverless* para áreas de georreferenciamento ou para propósitos de *workflows* científicos, os mais relevantes podem ser destacados a seguir:

- Em Carreira (2018), uma arquitetura geral para *frameworks* de aprendizado de máquina em ambientes *serverless* é proposta. Foram destacados alguns pontos sobre as dificuldades de operar neste ambiente, como a falta de suporte de carga de trabalho em placas de vídeo, falta de armazenamento compartilhado de pouca latência e alta vazão, como também pouca memória local. Também é comentado sobre a necessidade do processamento realizado ser leve e de alta performance para o uso eficiente dos recursos limitados das funções Lambda.
- Em Bebortta et al. (2020), é implementado um *framework* para o uso de bibliotecas Python a fim de realizar o processamento de dados geoespaciais. São avaliados dois casos de estudo: um para o conjunto de dados *Mineral Resources Data System*, referentes às informações de densidade de recursos minerais pelo globo, e outro para o conjunto de dados *Fairfax Forecast Households*, referente à previsão domiciliar. Ambos os casos de estudo mencionados utilizaram arquitetura *serverless* para reduzir as limitações de tempo e processamento.
- Em Malawski et al. (2020), é avaliada a aplicabilidade da arquitetura *serverless* para a execução de fluxos de trabalhos científicos. Um protótipo foi desenvolvido nos ambientes Google Cloud Functions e AWS Lambda, usando da *workflow engine HyperFlow*. Levantando dados de *benchmark* através do fluxo científico Montage, que é muito usado para esta finalidade, foi concluído que houve boa escalabilidade e baixas despesas na abordagem *serverless*.
- Em Cardoso (2020) é realizado o desenvolvimento de um *framework* para coleta e processamento de dados de geolocalização em tempo real. Testando-o na implementação de uma aplicação de monitoramento de frota de veículos, no ambiente *serverless* Google Cloud Functions.

No que diz respeito a última categoria de trabalhos, é possível verificar as (iii) arquiteturas *serverless* com enfoque em propósitos mais gerais e comerciais:

- (ADZIC; CHATLEY, 2017): Descrevem a arquitetura *serverless*, tendo como principal característica o pagamento por uso ao invés de por capacidade reservada. Comparam os custos em relação às alternativas, como aluguel de servidores e contêineres. Também analisam dois casos da indústria, mostrando como fazer a migração para Lambda reduziu custos de hospedagem na ordem de 66% e 95%.

- (SHAFIEI; KHONSARI; MOUSAVI, 2019): Abordam as características da arquitetura *serverless*, entrando mais a fundo em questões técnicas como memória compartilhada, segurança e privacidade. Citam diversos trabalhos sobre casos de implementações em *serverless*, com aprendizado de máquina presente em poucos casos.
- (EISMANN et al., 2021): Analisam 89 casos de uso, levantando estatísticas sobre características como linguagem utilizada, tamanho das funções, tempo de execução, entre outros. Contextos de aplicação variam de categorias, mas sensoriamento remoto se destaca entre os casos de estudo científicos analisados. De acordo com a pesquisa, os maiores motivadores para utilização de *serverless* foram: custo (33%), redução de esforço operacional (também diretamente relacionado ao custo) (24%) e escalabilidade oferecida (24%). Entretanto, a manutenção (2%) não é um fator tão decisivo.

Como pode ser observado, os trabalhos correlatos acima descrevem técnicas bastante pontuais para a solução de BDS isoladamente, mas sem mencionar detalhes de implementação em arquiteturas alternativas, ou trabalhos que apresentam arquiteturas para *frameworks* genéricos, mas sem possuir a sua aplicação com foco direto em BDS. Neste direcionamento, o presente trabalho se caracteriza como o primeiro a explorar a integração entre uma solução BDS desenvolvida com bom grau de eficácia, em uma arquitetura *serverless* apresentando os benefícios anteriormente descritos, como estratégia de infraestrutura.

3 Fundamentação Teórica

3.1 Aprendizado de Máquina

Aprendizado de máquina é o estudo sobre algoritmos que permitem aos sistemas aprenderem automaticamente e melhorarem com experiência sem serem necessariamente programados (SAMUEL, 1959). O processo de aprendizagem pode ser categorizado como supervisionado, onde se sabe o resultado esperado para determinados dados de entrada durante o treinamento; não supervisionado, onde não há a relação entre os dados de entrada e a saída esperada, mas o sistema é modelado para encontrar tal relação; e por reforço, onde os modelos são recompensados por atingirem bons resultados e penalizados por errarem. Os problemas a serem solucionados podem ser classificados em regressão (encontrar o valor) e classificação (encontrar a categoria) (HASTIE; TIBSHIRANI; FRIEDMAN, 2001).

Entre as técnicas disponíveis de modelos de aprendizado de máquina, uma que apresenta resultados expressivos e boa capacidade de generalização dos problemas são as árvores de decisão. Estas consistem em nodos internos e nodos de folhas, que servem para testar e detectar padrões nos dados de entrada, se ajustando conforme o treinamento para que o resultado final seja de acordo com o esperado (NAVADA et al., 2011). Neste contexto, nota-se o *framework* LGBM, que consiste em algoritmos de aprendizado baseados em árvores de decisão utilizando *gradient boosting*, uma técnica de aprendizado de máquina que produz um modelo de predição a partir da agregação de distintos modelos menores (KE et al., 2017).

3.2 Sensoriamento Remoto

Google Earth Engine (GEE) é uma plataforma em nuvem para análise geoespacial em nível global, que disponibiliza a alta capacidade computacional da Google na solução de

problemas de alto impacto social, como desmatamento, seca, desastres, segurança alimentar, gestão da água, monitoramento do clima e proteção ambiental. Foi projetado para permitir que não apenas cientistas capacitados possam utilizá-lo, mas também pessoas que carecem de conhecimentos técnicos, disponibilizando diversas operações de manipulação de dados com processamento direto da nuvem e ampla documentação (GORELICK et al., 2017) (GOOGLE..., 2021).

Permite a aquisição de imagens de diversas missões de sensoriamento remoto, dentre elas a missão Sentinel-2, do Programa Copernicus. Esta missão tem como objetivo prover observações ópticas multi-espectrais de alta resolução da superfície do globo terrestre, podendo ser utilizada para aplicações como mapas de cobertura da terra, mapas de detecção de mudança de terreno, análise de variáveis geográficas, entre outros. A missão Sentinel-2 captura 13 bandas espectrais, conforme demonstrado na Tabela 1, variando de visível e infravermelho próximo até infravermelho de ondas curtas, e resolução espacial variando de 10 metros à 60 metros por pixel dependendo da banda espectral. É esperado que cada satélite da missão tenha pouco mais de 7 anos como expectativa de vida, ao longo de 15 anos do andamento da missão, que teve seu início em 2013 (DRUSCH et al., 2012).

Nome	**Resolução (metros por pixel)**
1 - Aerosol Costeiro	60
2 - Azul	10
3 - Verde	10
4 - Vermelho	10
5 - Red Edge 1	20
6 - Red Edge 2	20
7 - Red Edge 3	20
8 - NIR	10
8A - Narrow NIR	20
9 - Vapor D'água	60
10 - Cirrus	60
11 - SWIR1	20
12 - SWIR2	20

Tabela 1 – Resoluções das bandas obtidas pelos satélites da missão Sentinel-2

3.3 Tecnologias e *Frameworks*

Python é uma linguagem de programação interpretada, orientada a objetos e de alto nível com semântica dinâmica, possuindo disponibilidade de uma ampla variedade de bibliotecas e *frameworks*, como também é suportada no ambiente AWS Lambda. Bibliotecas relevantes que foram utilizadas ao longo do desenvolvimento deste trabalho e relacionadas ao processo de BDS ou operações no ambiente *serverless* estão listadas abaixo:

- Scikit-learn (2021): biblioteca de aprendizado de máquina com suporte a técnicas supervisionadas e não supervisionadas. Provendo diversas ferramentas para processamento de dados, treinamento, seleção e análise de modelos, entre outros.
- LightGBM... (2021): *framework* de algoritmos de aprendizado baseados em árvores de decisão utilizando *gradient boosting*.

- Numpy (2021): biblioteca em Python que permite a execução de diversas operações matemáticas, como operações de álgebra linear, operações de estatística, ordenação, entre muitos outros.
- Pandas (2021): biblioteca para análise e manipulação de conjuntos de dados, geralmente em formato de tabela de dados, ou quadro de dados, conhecidos como *dataframes*.
- Matplotlib (2021): biblioteca para visualização de dados e estatística descritiva, usada na geração de métricas dos modelos.
- Tifffile (2021): biblioteca para manipulação de metadados e valores de imagem contidos em arquivos de extensão tif.
- Boto3 (2021): biblioteca que permite criação, configuração e gerenciamento de serviços AWS.
- Joblib (2021): conjunto de ferramentas para operações eficientes de leitura e escrita de objetos em disco.
- PyMySQL (2021): interface para conexão do Python com servidor de banco de dados MySQL.
- SQLAlchemy (2021): conjunto de ferramentas para operações SQL em Python.
- Zipfile (2021): módulo que prove ferramentas para criação, escrita, leitura e manipulação de arquivos ZIP.
- utm (2021): Conversão bidirecional entre coordenadas do padrão UTM (distância horizontal, vertical, código e letra da zona) e WGS84 (latitude e longitude em ângulo).
- GEE... (2021): módulo que permite a utilização da plataforma GEE através de uma API.
- Flask (2021): *framework* de *Web Server Gateway Interface*, especificação que descreve como servidores se comunicam com aplicações.
- Requests (2021): biblioteca para envio de requisições HTTP.

3.4 *Serverless*

Arquitetura *serverless*, ou computação *serverless*, também conhecida como *Function as a Service* (FaaS) e é uma categoria de serviço de computação em nuvem que oferece uma plataforma para que clientes desenvolvam e gerem aplicações a partir da definição de funções. Sua principal distinção é a abstração dos servidores nas aplicações, removendo esta responsabilidade dos desenvolvedores, e dando a origem ao nome *serverless*, que significa "sem servidor". A lógica da aplicação é definida pelo cliente: quando o usuário final ou outro sistema requisitar a execução, uma cópia da função é executada pelo provedor, que envia o resultado da execução de volta para o usuário (ADZIC; CHATLEY, 2017)(SHAFIEI; KHONSARI; MOUSAVI, 2019).

As principais características de um serviço *serverless* são descritas por (SHAFIEI; KHONSARI; MOUSAVI, 2019) como:

- Omissão das informações sobre o ambiente de execução para o cliente, como máquina virtual, contêiner, sistema operacional, entre outros, tornando-as transparentes para o cliente;
- Escalabilidade automática como responsabilidade do provedor, onde recursos devem estar disponíveis de acordo com a demanda;
- Somente os recursos utilizados pelo cliente devem ser cobrados pelo provedor;
- Capacidade do provedor de executar a requisição assim que ela for recebida, ou o quanto antes possível, com tempo de execução máximo limitado;
- O provedor deve estar ciente das definições e dependências das funções adicionadas pelo cliente, como também quaisquer informações relacionadas ao estado e ambiente de execução.

Os principais benefícios da arquitetura *serverless* estão relacionados à implementação de aplicações sem a preocupação com infraestrutura e escalabilidade. O provedor é o responsável por alocar os recursos necessários para a execução de novas requisições, removendo o limite de chamadas que uma aplicação poderia atender por meios de infraestrutura convencionais, como alocação de recursos em nuvem (máquinas virtuais e/ou contêineres). Máquinas dedicadas à execução de uma única aplicação geralmente mantém um baixo nível de utilização média, subutilizando a mesma e não dimensionando a relação entre custo de aquisição e aproveitamento. No ambiente de compartilhamento de recursos, o uso é otimizado, reduzindo gastos em infraestrutura e esforço operacional. Outro ponto relacionado à redução de gastos é a influência que a cobrança por processamento afeta a qualidade do código produzido, motivando otimização e eficiência (SHAFIEI; KHONSARI; MOUSAVI, 2019).

A AWS Lambda foi a primeira plataforma de computação *serverless* disponível por um provedor de grande porte, seguida pela Google Cloud Functions[2] e Microsoft Azure Functions[3], juntamente com as soluções baseadas em código aberto, como a IBM Cloud Functions[4] (baseada no OpenWhisk[5]) e Oracle Cloud Fn[6] (baseada no Fn Project[7]). Amazon Web Services (AWS) é uma subsidiária da Amazon que fornece serviços de computação em nuvem sob demanda com base no pagamento conforme uso, possuindo servidores por todo o globo. O AWS Lambda é um serviço de computação *serverless* para praticamente qualquer tipo de aplicação ou serviço de *back-end*. Funções podem ser configuradas a partir do envio do código em formato ZIP ou de imagem de contêiner, e o Lambda se responsabiliza por alocar os recursos necessários para a execução das funções com base na solicitação ou evento de entrada, para qualquer dimensão de tráfego (AWS. . . , 2021a).

Atualmente, o preço para a região da América do Sul (São Paulo), que é o mesmo para a maioria das regiões disponíveis, é de 0,0000166667 USD por cada GB de memória por segundo e 0,20 USD por 1 milhão de solicitações, de acordo com a página oficial[8].

[2] <https://cloud.google.com/functions>
[3] <https://azure.microsoft.com/en-us/services/functions/>
[4] <https://cloud.ibm.com/functions/>
[5] <https://openwhisk.apache.org/>
[6] <https://www.oracle.com/cloud-native/functions/>
[7] <https://fnproject.io/>
[8] <https://aws.amazon.com/lambda/pricing/>

É possível alocar à função Lambda valores de memória entre 128 MB e 10.240 MB, em incrementos de 1 MB. Na Tabela 2 são apresentados valores para alguns casos simulados para efeitos de comparação em função do número de requisições e memória alocada.

Número de requisições	**Tempo de execução (segundos)**	**Memória alocada (MB)**	**Preço (USD)**
1	1	1024	0,0000167
50	300	2048	0,501
3000	900	4096	180,36

Tabela 2 – Simulações para diferentes cenários de uso do serviço AWS Lambda

Fonte: Próprio autor

A cobrança só é feita referente ao uso além dos 400 mil GB-s disponíveis mensalmente sem custo, ou seja, os dois primeiros casos apresentados seriam de graça e o terceiro seria de 173.68 USD.

4 Metodologia

Uma vez que a abordagem computacional proposta para a solução do problema da Batimetria Derivada de Satélite envolve a elaboração de um modelo preditor (CAMOZZATO et al., 2021) e a sua respectiva implementação em um ambiente *serverless*, o que compete ao presente trabalho é sumarizado através do diagrama da Figura 1, onde 6 etapas características podem ser categorizadas. A partir do ponto 1, de levantamento de requisitos, realizado através de reuniões com coordenadores e integrantes do projeto responsável pelo CASSIE, foram definidas as funcionalidades necessárias na API e como a integração poderia ser feita, juntamente com as regras de negócio. Com base nos requisitos definidos na etapa 1, várias etapas subsequentes podem ser realizadas de forma paralela, como sendo: 2 - modelagem e criação do banco de dados; 3 - definição dos fluxos de execução; e 4 - preparação das camadas do Lambda. A evolução destas etapas permitiu a continuação do desenvolvimento, com o passo 5, de criação das funções lambda, e 6, configuração da API, concluindo a implementação. As etapas anteriormente mencionadas serão descritas em detalhes nas próximas subseções.

4.1 Levantamento dos requisitos

Esta etapa apresenta as funcionalidades necessárias, permitindo que fosse definido como o sistema da API deveria ser desenvolvido. Os requisitos principais podem ser sumarizados em:

- Criação de modelos: criar modelos capazes de realizar a predição de BDS a partir de dados de batimetria e informações da região de interesse.
- Listagem dos modelos: filtrar modelos por endereço de email de quem o criou, retornando a lista dos modelos correspondentes e suas informações.
- Criação de predições: criar predições para regiões de interesse a partir de modelos previamente treinados.

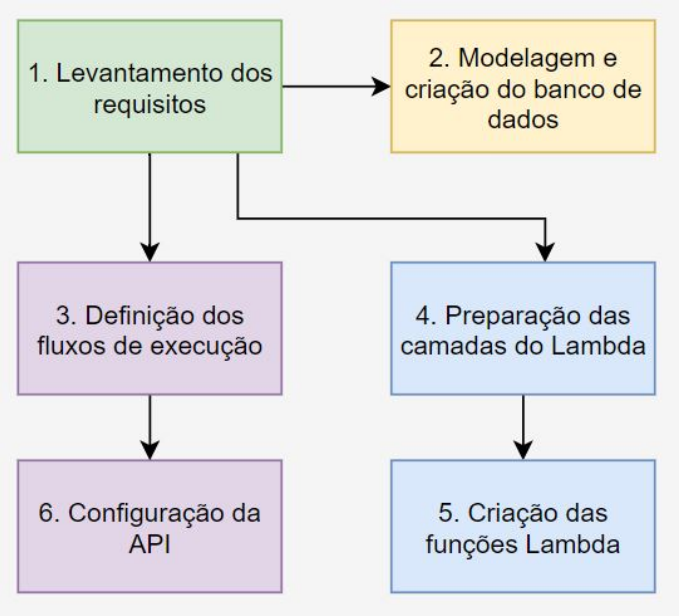

Figura 1 – Diagrama de fluxo e etapas da metodologia. Uma vez definido o modelo preditor, a presente proposta para concepção em um ambiente *serverless* pode ser definida em 6 principais etapas.

Fonte: Próprio autor

- Listagem de predições: filtrar predições feitas pelo endereço de email de quem a requisitou, retornando a lista de predições correspondentes e suas informações.
- Métricas do modelo: métricas do modelo serão salvas no banco de dados, e gráfico de comparação e histograma de resíduo deverão ser gerados e armazenados. Estas informações poderão ser acessadas pelos usuários que criaram os modelos.
- Informações detalhadas de predição: possibilitar acesso às informações detalhadas de uma predição.

Para as requisições relacionadas à criação de modelos e criação de predições, a aplicação utilizando da API será responsável pela criação de um objeto de imagem GEE a ser serializado e enviado, como também pelo envio do arquivo de dados de batimetria para o conjunto de dados de treinamento.

Nota-se que na requisição de criação de predições, também é possível que o modelo seja usado para gerar a predição na mesma área que foi treinado, porém com outra informação temporal (data), visto que o objeto de imagem GEE enviado para a requisição contém tanto informações espaciais como temporais.

4.2 Modelagem e criação do banco de dados

Para o armazenamento dos dados foi utilizado o *Amazon Simple Storage Service* (Amazon S3) e *Amazon Relational Database Service* (Amazon RDS). O primeiro sendo um serviço de armazenamento de arquivos, possibilitando vários casos de uso, como data lakes, sites, aplicativos para dispositivos móveis, backup e restauração, arquivamento (AWS. . . ,

2021c). O segundo é o serviço de banco de dados relacionais na nuvem, permitindo seis mecanismos de banco de dados comuns, Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database e SQL Server (AWS. . . , 2021b).

Para modelagem do banco de dados, os módulos Flask e SQLAlchemy foram utilizados, gerando um banco de dados MySQL no ambiente Amazon RDS. MySQL é um sistema de gerenciamento de banco de dados relacionais de código aberto (MYSQL, 2021). O modelo do banco de dados relacional, Figura 2, apresenta as entidades *Model* (Modelo) e *Prediction* (Predição) e suas relações, onde um Modelo pode estar associado a diversas Predições e uma Predição deve possuir um Modelo associado.

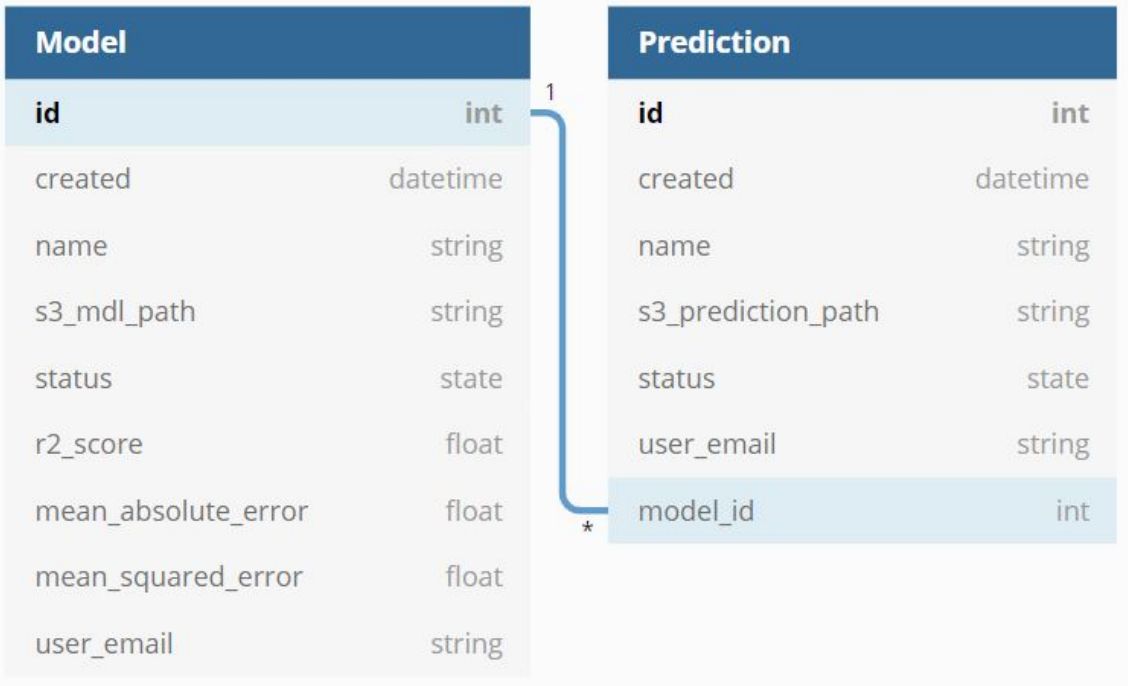

Figura 2 – Modelo do banco de dados, apresentando as definições e relações das entidades *Model* e *Prediction* para a abordagem BDS proposta.

Fonte: Próprio autor

Os atributos das entidades foram definidos conforme a necessidade da aplicação. Para o caso da entidade Modelo:

- *id*: identificador único, em formato de número inteiro.
- *name* e *created*: nome e data de criação, em formatos de texto e data, respectivamente, para que o usuário possa diferenciar entre os modelos criados.
- *s3_mdl_path*: caminho do diretório no Amazon S3, em formato de texto, onde arquivos relacionados ao modelo serão mantidos, como imagens de gráficos de métricas (comparação e histograma), objeto serializado do modelo treinado, dados da batimetria e dados relacionados à predição realizada no conjunto de testes.
- *status*: estado indicando se o modelo está em criação ou ativo (disponível após treinamento).

- *r2_score, mean_absolute_error, mean_squared_error*: métricas do modelo, sendo elas coeficiente de determinação (R^2), erro médio absoluto e erro médio quadrado.
- *user_email*: email do usuário, utilizado para filtrar quais modelos pertencem à ele.

De forma similar, para a entidade Predição os seguintes campos foram definidos:

- *id*: identificador único, em formato de número inteiro.
- *name e created*: Nome e data de criação, em formatos de texto e data, respectivamente, para que o usuário possa diferenciar entre as predições realizadas.
- *s3_prediction_path*: caminho para o dataframe armazenado no Amazon S3, contendo as informações de cada ponto da região selecionada para a realização da batimetria.
- *status*: estado indicando se a predição está em andamento ou finalizada.
- *user_email*: email do usuário, utilizado para filtrar quais predições pertencem à ele.

Os arquivos de conjuntos de dados de predições, objetos serializados de modelos treinados, imagens de gráficos de métricas e dados de batimetria são mantidos no ambiente Amazon S3, designado para esta finalidade. Cada modelo possui um diretório, nomeado a partir de um identificador único, e nele estão contidos os arquivos relacionados ao modelo, descritos acima.

4.3 Definição dos fluxos de execução

O AWS Step Functions é um orquestrador de funções sem servidor que facilita o sequenciamento de funções do AWS Lambda e vários serviços da AWS. Fluxos de trabalho podem ser criados por meio de uma interface visual, determinando sequências de etapas a serem executadas, respeitando a lógica de negócios definida, com a saída de uma etapa servindo de entrada para a próxima. Os fluxos A, de criação de modelo, e B, de utilização, apresentados na Figura 3 descrevem as etapas elaboradas para criação e utilização dos modelos, especificando a execução das funções Lambda, no ambiente AWS Step Functions (AWS. . . , 2021d).

O fluxo de criação de modelos, Fluxo A (Figura 3), inicia na etapa *DataAcquisition*, com a aquisição dos dados de treinamento: a partir das imagens obtidas através da API do GEE, em conjunto com os dados de batimetria enviados pelo usuário, um *dataframe* é gerado e armazenado no Amazon S3. Seguindo para a etapa *TrainModel*, que realiza o treinamento do modelo, salvando-o em formato de arquivo de objeto serializado, como também gera e armazena no banco de dados as informações de métricas. Outras métricas também são geradas, na etapa *CreateValidationCharts*, que é feita logo após o treinamento, imagens de gráficos de comparação e histograma de resíduos são geradas e armazenadas. Chegando à etapa de decisão *CheckStatus*, responsável por verificar se a criação do modelo ocorreu com sucesso, neste caso, inicia a etapa *CreateModelDone*, que salva no banco de dados as informações de estado e métricas do modelo, caso a criação tenha falhado, a etapa *AbortCreatePrediction* é responsável por remover registros e arquivos relacionados ao modelo, criados pelas etapas anteriores.

O fluxo de criação de predição, Fluxo B (Figura 3), descreve as etapas para a utilização de um modelo previamente treinado. Começando pela etapa *AcquisitionAndPrediction*, responsável por adquirir os dados e montar um *dataframe* dos pontos considerados

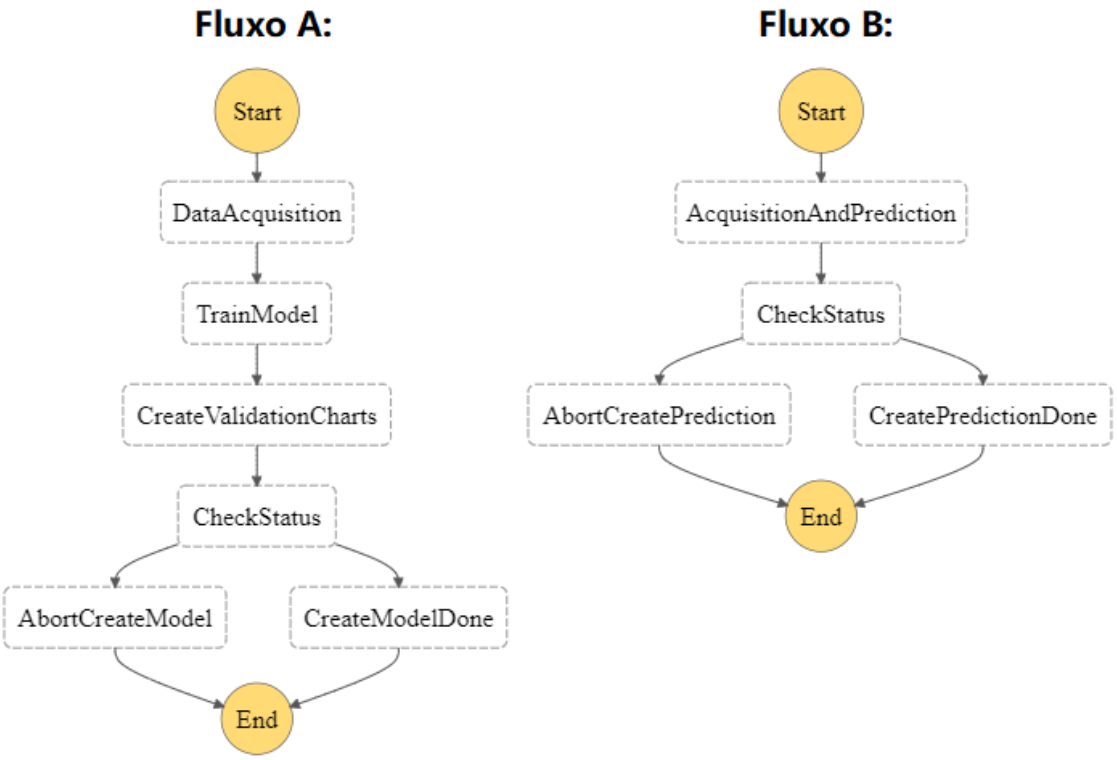

Figura 3 – Fluxos definidos no ambiente AWS Step Funcion, onde o fluxo A descreve o processo de criação do modelo e o fluxo B descreve sua utilização.

Fonte: Próprio autor

região de água, possibilitando a aplicação do modelo para gerar a predição. Novamente, uma etapa de decisão é utilizada para selecionar qual o próximo passo, em *CheckStatus* é verificado se a predição foi feita com sucesso, seguindo para a etapa *CreatePredictionDone*, salvando no banco de dados em caso de sucesso, ou para *AbortCreatePrediction*, apagando registros relacionados à predição em caso de falha.

Cada fluxo funciona como uma máquina de estados, a definição é feita com a Linguagem de estados da Amazon, nomeando cada estado, descrevendo suas condições e comportamentos. Os estados individuais podem tomar decisões com base em seus dados de entrada, executar ações e transmitir os dados de saída para outros estados. Os fluxos descritos aqui operam com estados do tipo *task* ou *choice*, o primeiro tipo executa uma tarefa, que neste contexto se resume à execução de funções Lambda, de acordo com o identificador do recurso informado, o segundo tipo está relacionado com a tomada de decisão, escolhendo para qual estado seguir. Também existem outros tipos, como *pass*, que repassa os dados de entrada para saída, *map*, que itera dinamicamente sobre as etapas, *wait*, que introduz um atraso na operação, *parallel*, que ramifica o prosseguimento dos estados, entre outros.

A utilização do AWS Step Functions é motivada por dois fatores. Um sendo a possibilidade de retornar uma resposta logo ao receber uma requisição que demoraria a ser concluída, e o outro pela modularização das etapas de processamento, evitando a dificuldade das restrições de tempo de execução das funções Lambda.

4.4 Preparação das camadas do Lambda

As camadas do Lambda permitem o compartilhamento de código entre as funções, podendo conter bibliotecas e outras dependências. Servem para reduzir o tamanho do código compactado a ser enviado para o Lambda. Cada função pode fazer uso de cinco camadas ao mesmo tempo, com o tamanho total das camadas e dos arquivos de código descompactados não podendo exceder 250 MB.

Foram criadas camadas para os seguintes módulos e bibliotecas: LGBM, GEE API, Pandas, Requests, PyMySQL, SQLAlchemy, Scikit-learn, TiffFile, utm, Matplotlib e Joblib. Algumas camadas contém mais do que uma biblioteca, permitindo que cada função Lambda faça o uso das bibliotecas necessárias, mas sem ultrapassar o limite descrito anteriormente. Outra maneira de resolver esta restrição é fazer uso de imagens de contêiner, contendo o código das funções e todas as dependências necessárias.

4.5 Funções Lambda

A configuração das funções Lambdas criadas permite que elas sejam executadas nos seguintes casos: as funções relacionadas à requisições, que possuem *endpoints* (pontos de acesso disponíveis), são executadas conforme chamadas são feitas para a API; já as funções relacionadas com os fluxos da AWS Step Functions são executadas de acordo com o andamento do mesmo.

4.5.1 Funções da API

Uma API é uma interface que define as interações possíveis entre aplicações. As funções descritas nesta seção foram configuradas para estarem disponíveis à requisições externas, relacionadas ao retorno de informações contidas no banco, ou ao início de fluxos referentes ao treino, ou uso de modelos.

***StartModelTraining*:**

Função do ponto de acesso relacionada ao início do fluxo de criação e treinamento de modelos. Cria um diretório no Amazon S3 para armazenar os arquivos relacionados e também um registro no banco de dados para o modelo, contendo as informações do corpo da requisição, como nome, endereço de email, data de criação, estado e caminho do diretório recém criado. Adquire o identificador único do registro e salva no corpo da requisição, para as próximas etapas utilizarem, e retorna resposta para a requisição após dar início ao fluxo de criação do modelo.

***GetModels*:**

Esta função permite que requisições com a intenção de listar os modelos criados por um usuário sejam realizadas. Ela se conecta com o banco, filtrando a tabela de modelos pelo endereço de e-mail informado por parâmetro da requisição e retorna um objeto contendo as informações de cada modelo.

***GeneratePrediction*:**

Possibilita a requisição de predição para a região desejada. Recebe no corpo da requisição o objeto de imagem da GEE serializado, identificador único do modelo a ser utilizado, nome da predição e endereço de email do usuário. Verifica se já existe uma predição com este nome para este usuário, retornando mensagem de erro, se não procede com a criação da predição no banco de dados, salvando as informações de nome da predição,

data de criação, estado, endereço de email, caminho do diretório do modelo e identificador único do modelo.

Inicia o fluxo de criação da predição, adicionando o identificador único da predição recém criada para o corpo da requisição, passando-a para a próxima etapa, depois, retorna resposta avisando sobre o início do processo.

***GetAllUserPredictions*:**

A partir do endereço de email informado por parâmetro da requisição, uma busca no banco de dados filtrando por requisições de endereço de email correspondente é realizada. Registros compatíveis são retornados no corpo da resposta.

***GetDetailedPredictionInfo*:**

Retorna informações detalhadas da predição associada ao identificador único informado na requisição. Com estas informações é possível acessar o arquivo do *dataframe* contido no Amazon S3, contendo a coordenada e profundidade estimada para cada um dos pontos da região de interesse.

4.5.2 Funções do fluxo de criação de modelos

Nesta seção são apresentadas as funções relacionadas ao fluxo de criação de modelos, que se resume à aquisição dos dados, treino do modelo e geração de métricas.

***DataAcquisition*:**

Função da etapa responsável por adquirir os dados necessários para o treinamento, salvando o arquivo do *dataframe* gerado na Amazon S3. Este contendo as coordenadas e profundidade dos pontos de batimetria, como também respectivos valores de reflectância para cada uma das bandas. Do corpo da requisição são adquiridos o objeto de imagem GEE serializado e caminho do arquivo de dados da batimetria armazenado no Amazon S3. Deserializando o objeto de imagem permite que este seja utilizado para fazer requisições à GEE através de sua API, primeiro um filtro de mediana com janela móvel de 150m por 150m é aplicado, depois é adquirido o endereço de descarregamento das imagens, especificando parâmetros para a resolução da imagem e para o sistema de referência de coordenadas EPSG:4326 (WGS84, coordenadas em ângulos de latitude e longitude, a partir do equador e meridiano de Greenwich).

Fazendo uma requisição a este endereço, um arquivo de formato ZIP contendo as imagens correspondentes às bandas da região de interesse é armazenado localmente. Após ser descompactado, as imagens de extensão tif podem ser carregadas na memória. Informações de coordenada inicial e escala são obtidas através da leitura dos metadados dos arquivos de imagens (GEOTIFF..., 1995), usados posteriormente para relacionar cada ponto de batimetria ao pixel correspondente.

O *dataframe* é criado a partir dos dados de batimetria, e todos os pontos de coordenadas são convertidos em coordenadas de imagem.Valores inválidos de coordenada de pixel são filtrados, como valores negativos ou acima do tamanho da imagem (horizontalmente ou verticalmente). Com a informação de coordenada de imagem, valores das reflectâncias das bandas são inseridos no *dataframe*.

Devido à conversão, muitos pontos de batimetria podem pertencer à um mesmo pixel. Para uniformizar este aspecto, através do agrupamento destes pontos é possível fazer uma análise de variância, removendo os pontos que possuírem variância acima de

0,05. Também utilizando do agrupamento, valores de profundidade para cada pixel são calculados a partir da média de profundidade dos pontos de batimetria naquele pixel.

Após a realização destas operações o *dataframe* é armazenado na Amazon S3 e o caminho para o arquivo é adicionado no corpo da requisição, passada para a próxima etapa, de treinamento.

***TrainModel*:**

Função responsável pela etapa de criação, treinamento e armazenamento do objeto do modelo de predição. Os dados de treinamento armazenados previamente em um *dataframe* são carregados, a partir do caminho do arquivo informado no corpo da requisição. Este conjunto de dados então é dividido aleatoriamente em conjunto de treinamento, contendo 70% dos dados, e conjunto de validação, contendo 30% dos dados.

O processo de treinamento é iniciado, com os dados do conjunto de treinamento, e após finalizado, informações básicas de métricas são extraídas a partir de predições feitas sobre o conjunto de validação, sendo elas: R^2, erro médio absoluto e erro médio quadrado. Os dados de profundidade do conjunto de validação e predições correspondentes são armazenados no corpo da requisição, para serem utilizados na etapa de criação de gráficos de métricas. Esta etapa se encerra com o armazenamento do arquivo do objeto de modelo treinado na Amazon S3, dando continuidade para a próxima etapa.

***CreateValidationCharts*:**

Fazendo uso dos dados de profundidade do conjunto de validação e predições correspondentes, gráficos de histograma de resíduos e de comparação são gerados e armazenados no diretório do modelo na Amazon S3. Estes gráficos são discutidos em mais detalhes na Seção 5

***CreateModelDone*:**

Esta função somente é executada caso o modelo tenha sido criado com sucesso, verificado na etapa *CheckStatus*. Armazena no banco de dados que a criação do modelo está finalizada (alterando seu estado) e insere as métricas geradas anteriormente. Por fim, a função apresenta como retorno o aviso de treinamento de modelo finalizado.

***AbortCreateModel*:**

Esta função somente é executada caso a criação do modelo tenha falhado, verificado na etapa *CheckStatus*. Remove o registro do modelo em questão do banco de dados e apaga o diretório relacionado ao modelo na Amazon S3. Retornando resposta de erro, treinamento de modelo abortado.

4.5.3 Funções do fluxo de predição:

Nesta seção são apresentadas as funções relacionadas ao fluxo de utilização de um modelo, gerando uma predição.

***AcquisitionAndPrediction*:**

Função responsável pela aquisição dos dados de imagens da GEE e aplicação do modelo para obtenção da predição de profundidade. O processo de aquisição dos dados é idêntico ao descrito anteriormente (na função *DataAcquisition*). Com as imagens carregadas em memória, uma máscara da região de água é criada a partir do *normalized difference water index* (NDWI), gerado utilizando das bandas 8 e 3. O índice NDWI é obtido através

dos valores de reflectância de duas bandas, calculando o valor da divisão da diferença das bandas pela soma das mesmas.

Com a máscara criada é possível filtrar regiões consideradas água como sendo aquelas com um valor de índice superior a algum determinado *threshold*. Um *dataframe* é criado contendo todos os pontos desta região, como também os valores de reflectância das bandas e coordenadas. Os valores de coordenadas não são utilizados pelo modelo, mas são mantidos no *dataframe* para que posteriormente o usuário possa utilizá-los, como, por exemplo, na geração de mapas de profundidade.

O objeto do modelo previamente treinado é carregado em memória, permitindo que seja utilizado para a geração das predições. Os valores de profundidade gerados são adicionados ao *dataframe*, que então é armazenado na Amazon S3.

***CreatePredictionDone*:**

A execução desta função depende de um estado, verificado na etapa *CheckStatus*. É executada somente se a criação da predição obteve êxito. Atualiza o registro da predição, mudando o estado para finalizado e adiciona o caminho para o arquivo do *dataframe* criado. Retornando resposta de sucesso na operação.

***AbortCreatePrediction*:**

A execução desta função depende de um estado, verificado na etapa *CheckStatus*. É executada somente em caso de falha no processo de predição. Removendo o registro da predição do banco de dados e retornando a resposta de erro.

4.6 Configuração da API

A comunicação entre aplicações pode ser feita através de APIs, sendo estas, interfaces definidas e documentadas. As APIs agem como a porta de entrada para aplicativos acessarem dados, lógica de negócios ou funcionalidade de serviços. A configuração dos *endpoints* possibilita o recebimento destas requisições, que geralmente contém um cabeçalho, contendo informações sobre a própria requisição, e um corpo, contendo os dados enviados.

A API implementada neste trabalho foi desenvolvida utilizando o Amazon API Gateway. Este consiste em um serviço que permite a criação, publicação, gerenciamento e monitoramento de APIs com facilidade em qualquer escala (AMAZON. . . , 2021). Além disso, administra todas as tarefas envolvidas no recebimento e processamento das requisições, incluindo o suporte a centenas de milhares de chamadas simultâneas, gerenciamento de tráfego, suporte a *Cross-Origin Resource Sharing* (CORS) e controle de autorização e acesso. Estas facilidades fazem o uso do Amazon API Gateway extremamente vantajoso pois com ele o utilizador consegue escalar APIs facilmente trazendo agilidade no desenvolvimento de soluções. O Amazon API Gateway permite a criação de dois tipos de APIs, sendo elas:

- REST: *Representational State Transfer* (Transferência Representacional de Estado) é uma abstração de arquitetura Web. Este tipo de API utiliza o protocolo HTTP de comunicação, sendo que a arquitetura REST define o modo que o protocolo deve ser utilizado e não o protocolo em si. Uma API deste tipo deve realizar a comunicação cliente/servidor de forma *stateless*, ou seja, o servidor não armazena estados e cada requisição é independente. Diversos formatos de dados nas requisições são aceitos por esta arquitetura, porém o mais comum é o *Javascript Object Notation* (JSON).

- Websocket: Este tipo de API permite que aplicativos realizem comunicação bidirecional em tempo real. Diferentemente da API REST que utiliza o protocolo HTTP em toda sua comunicação, Websocket utiliza HTTP apenas no *handshake*[9] e depois mantém a comunicação através do protocolo TCP (Transmission Control Protocol).

A arquitetura escolhida para o desenvolvimento deste trabalho foi a REST, com dois tipos de requisições HTTP utilizadas para comunicação: GET para requisições de dados e POST para envio de dados.

No Amazon API Gateway é possível criar recursos, que consistem nos *endpoints* da API e para cada recurso é possível adicionar métodos HTTP. Na criação dos métodos existem diversas opções de integração com outros serviços. Neste trabalho foi utilizado a integração com função Lambda através de *proxy*. Ao utilizar o *proxy*, toda requisição é passada para a função Lambda e esta fica responsável por analisar a requisição e determinar o tipo de resposta que deve ser retornada.

Além de configurar o tipo de integração, é necessário configurar o CORS para cada *endpoint*. O CORS é um mecanismo presente nos navegadores que permite o compartilhamento seletivo de recursos entre diferentes origens fazendo uso de *headers* HTTP. Com isto consegue-se permitir que aplicações de diferentes domínios acessem um recurso específico. Em estágio de desenvolvimento da aplicação deste trabalho, o CORS estava configurando permitindo acesso a partir de qualquer instância, ou seja, qualquer usuário poderia acessar a API através de sua URL. Para produção, o CORS deve ser configurado permitindo acesso apenas pelo endereço da aplicação *frontend* do CASSIE.

Todas as funções apresentadas na subseção 4.5.1 tem como gatilho de execução o recebimento de requisições HTTP para as rotas expostas pelo Amazon API Gateway descritas na Tabela 3.

5 Resultados Obtidos e Validação

Foram realizados testes com dezenas de chamadas simultâneas no ambiente da AWS Lambda, executando o fluxo de criação de modelo, abrangendo a aquisição dos dados a partir da GEE, o treinamento do modelo e a geração de gráficos de métricas. O tempo médio de execução do fluxo, que inclui o tempo da comunicação com o banco de dados e também de transferência com o ambiente de armazenamento de arquivos Amazon S3, é apresentado na Tabela 4, em função da quantidade de memória alocada para as funções.

Os tempos para valores de memória superiores a 2048 MB, na Tabela 4, possuem alta variância devido ao processo de aquisição dos dados através da API da GEE não possuir uma duração constante, como também o tempo para AWS realizar a alocação de recursos com maior capacidade, sugerindo que o ganho em processamento não valha a pena considerando este ponto. Havendo uma diferença de até aproximadamente 30 segundos entre os tempos de execução para o caso de 2048 MB de memória, onde o menor tempo de execução foi de 81,4 segundos, enquanto o maior foi de 108,8 segundos.

Executando este mesmo fluxo localmente, sem comunicação com quaisquer entidades externas além da GEE (para a aquisição dos dados das imagens), o tempo médio de execução foi de aproximadamente 75 segundos. O ambiente local possui as seguintes características:

[9] Primeira etapa da comunicação. No handshake detalhes da conexão são negociados e a conexão é aceita ou recusada.

StartModelTraining	
Rota:	/train-model
Tipo de requisição:	POST
Corpo da requisição:	caminho do arquivo de dados de batimetria, endereço de email, nome do modelo e objeto de imagem GEE serializado
Resposta:	estado e mensagem referente ao sucesso ou erro da inicialização do processo
GetModels	
Rota:	/create-prediction
Tipo de requisição:	GET
Parâmetro adicional no URL da requisição:	endereço de email
Resposta:	dicionários de informações de modelos
GeneratePrediction	
Rota:	/create-prediction
Tipo de requisição:	POST
Corpo da requisição:	JSON contendo endereço de email, objeto de imagem e id do modelo a ser utilizado
Resposta:	informações de estado e caminho do arquivo de predição
GetAllUserPredictions	
Rota:	/my-predictions
Tipo de requisição:	GET
Parâmetro adicional no URL da requisição:	endereço de email
Resposta:	JSON de informações de predições
GetDetailedPredictionInfo	
Rota:	/my-predictions
Tipo de requisição:	GET
Parâmetro adicional no URL da requisição:	id da previsão
Resposta:	JSON contendo de informações específicas da previsão

Tabela 3 – Descrição das rotas expostas no Amazon API Gateway

Fonte: Próprio autor

Memória alocada (MB)	**Tempo médio de execução (s)**
512	153,59
1024	123,73
2048	94,29
4096	91,54
8192	93,36

Tabela 4 – Tabela de tempo de execução em função da memória alocada para as funções Lambda.

Fonte: Próprio autor

processador Intel I5 6600K, com frequência de clock de 3.5GHz, 16 GB de memória RAM e sistema operacional Windows 10 Pro. Enquanto no ambiente da AWS Lambda foram alocados 512, 1024, 2048, 4096 e 8192 MB de memória RAM, porém, sem especificação exata sobre o processamento, que escala conforme a memória alocada. Considerando a escalabilidade proporcionada pela plataforma de computação *serverless* em questão, a diferença no tempo de execução se torna desprezível em configurações superiores à 2048 MB. Os modelos treinados nos diferentes ambientes apresentaram uma diferença de menos de 1% no valor de R^2.

Outro ponto de suma importância é a limitação encontrada durante o processo de aquisição de imagens através da GEE. Um limite de tamanho é atingido, restringindo a resolução das imagens de acordo com o tamanho da área selecionada. Mesmo com esta restrição, o modelo atingiu um R^2 acima de 0,8, porém, inferior ao R^2 de 0,94 obtido nos estudos utilizando imagens com a maior resolução disponível, de 10 metros por pixel. Possíveis soluções para este obstáculo são discutidas na próxima seção (6).

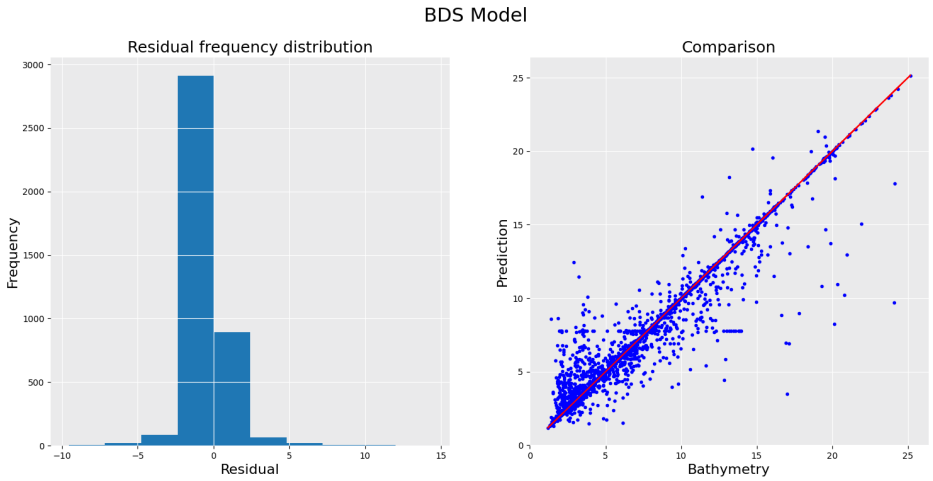

Figura 4 – Histograma de resíduos e gráfico de comparação do modelo gerado na nuvem

Fonte: Próprio autor

Os gráficos gerados na etapa de treinamento do modelo (histograma de resíduos e comparação) são apresentados na Figura 4, utilizando imagens obtidas através da GEE para toda a área da Baia da Babitonga, com resolução para 40 metros por pixel. Na Figura 5 estão os dados relacionados ao modelo gerado localmente para a mesma região, porém utilizando imagens de outras fontes, com resolução de 10 metros por pixel. Nota-se uma diferença considerável entre os modelos, reforçando que a aquisição das imagens é um ponto crítico para o desempenho.

Os histogramas de resíduos (Figuras 4 e 5) apresentam o número de ocorrências (eixo vertical) dos valores de diferença entre a profundidade predita e a medida (eixo horizontal), agrupados em intervalos de 2,5 metros. Os gráficos de comparações (Figuras 4 e 5) relacionam os valores de predição (eixo vertical) com os medidos (eixo horizontal), pontos de valores similares em ambos os eixos se aproximam da linha em vermelho, que representa o modelo ideal com uma relação de um para um entre a estimativa e valor medido.

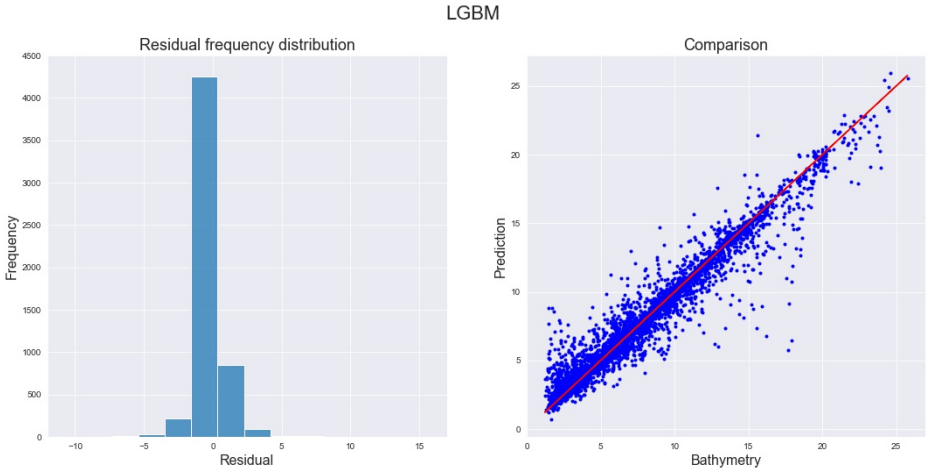

Figura 5 – Histograma de resíduos e gráfico de comparação do modelo gerado localmente

Fonte: Próprio autor

6 Conclusões e Trabalhos Futuros

Este trabalho apresentou uma abordagem no desenvolvimento de aplicações através do paradigma de computação *serverless*. A implementação descrita abordou diversos aspectos, como aquisição de dados de sensoriamento remoto, treinamento e uso de modelos de aprendizado de máquina, reforçando as vantagens e a viabilidade da arquitetura *serverless* para efetivação de *workflows* científicos. O presente artigo utilizou como preditor o modelo descrito em (CAMOZZATO et al., 2021), e o mesmo foi descrito e especificado através de uma API para utilização em uma arquitetura *serverless*.

O próximo passo para este trabalho consiste no acompanhamento do processo de integração com o CASSIE, realizando possíveis modificações que se mostrem necessárias. Permitindo assim que a aplicação seja utilizada para a realização de BDS em regiões por todo o globo.

Visando a melhoria da aplicação nota-se que um dos pontos a serem trabalhados futuramente é o processo de aquisição das imagens, alterando-o para que seja feito de tal forma que a maior resolução possível seja adquirida parcialmente em várias requisições, separando a região de interesse em diversas regiões menores. Alternativamente, é possível também fazer uso de algum serviço pago para a aquisição de imagens de alta resolução.

Outra possível melhoria seria a utilização de algum *framework* para a construção e implantação da aplicação, provendo agilidade ao desenvolvimento. Como o *Serverless Framework*[10], que se responsabiliza pela configuração de camadas, como também pelas configurações de memória, tempo limite, eventos, entre outros.

[10] <https://www.serverless.com/>

Referências

ADZIC, G.; CHATLEY, R. Serverless computing: Economic and architectural impact. In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2017. (ESEC/FSE 2017), p. 884–889. ISBN 9781450351058. Disponível em: <https://doi.org/10.1145/3106237.3117767>.

ALMEIDA, L. P. et al. Coastal analyst system from space imagery engine (cassie): Shoreline management module. *Environmental Modelling & Software*, v. 140, p. 105033, 2021. ISSN 1364-8152. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1364815221000761>.

AMAZON API Gateway. 2021. <https://aws.amazon.com/pt/api-gateway/>. Accessed: 2021-4-01.

AWS Lambda. 2021. <https://aws.amazon.com/pt/lambda/>. Accessed: 2021-4-01.

AWS RDS. 2021. <https://aws.amazon.com/pt/rds/>. Accessed: 2021-4-01.

AWS S3. 2021. <https://aws.amazon.com/pt/s3/>. Accessed: 2021-4-01.

AWS Step Functions. 2021. <https://aws.amazon.com/pt/step-functions>. Accessed: 2021-4-01.

BEBORTTA, S. et al. Geospatial serverless computing: Architectures, tools and future directions. *ISPRS International Journal of Geo-Information*, v. 9, n. 5, 2020. ISSN 2220-9964. Disponível em: <https://www.mdpi.com/2220-9964/9/5/311>.

BOTO3. 2021. <https://boto3.amazonaws.com/v1/documentation/api/latest/index.html>. Accessed: 2021-4-01.

CAMOZZATO, V. de V. et al. Obtenção de batimetria em estuários através da aplicação de métodos de aprendizado de máquina em imagens multiespectrais de satélites. 2021.

CARDOSO, D. S. D. *Framework for collecting and processing georeferencing data.* Dissertação (Mestrado), 2020. Disponível em: <https://hdl.handle.net/10216/127192>.

CARREIRA, J. A case for serverless machine learning. In: . [S.l.: s.n.], 2018.

DAS, S. K.; PANT, M.; BEBORTTA, S. Geospatial data analytics: A machine learning perspective. 2020. Disponível em: <https://dx.doi.org/10.2139/ssrn.3599656>.

DRUSCH, M. et al. Sentinel-2: Esa's optical high-resolution mission for gmes operational services. *Remote Sensing of Environment*, v. 120, p. 25–36, 2012. ISSN 0034-4257. The Sentinel Missions - New Opportunities for Science. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0034425712000636>.

EISMANN, S. et al. *A Review of Serverless Use Cases and their Characteristics*. 2021.

FERREIRA, I. O. *Coleta, processamento e análise de dados batimétricos visando a representação computacional do relevo submerso utilizando interpoladores determinísticos e probabilísticos*. Dissertação (Mestrado), 2016. Disponível em: <https://locus.ufv.br//handle/123456789/3801>.

FILIPPI, B. *Obtenção de batimetria em estuários através de imagens de satélite: estudo de caso na Baía da Babitonga, SC*. 2020. Disponível em: <https://repositorio.ufsc.br/handle/123456789/209992>.

FLASK. 2021. <https://palletsprojects.com/p/flask/>. Accessed: 2021-4-01.

GAGG, G. *Apostila de levantamentos hidrográficos - noções gerais*. [S.l.], 2016. Disponível em: <http://hdl.handle.net/10183/157210>.

GEE API Documentation. 2021. <https://developers.google.com/earth-engine/api_docs>. Accessed: 2021-4-01.

GEOTIFF Format Specification GeoTIFF Revision 1.0. [S.l.], 1995.

GEYMAN, E. C.; MALOOF, A. C. A simple method for extracting water depth from multispectral satellite imagery in regions of variable bottom type. *Earth and Space Science*, v. 6, n. 3, p. 527–537, 2019. Disponível em: <https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2018EA000539>.

GOOGLE Earth Engine. 2021. <https://earthengine.google.com/>. Accessed: 2021-4-01.

GORELICK, N. et al. Google earth engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment*, v. 202, p. 18–27, 2017. ISSN 0034-4257. Big Remotely Sensed Data: tools, applications and experiences. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0034425717302900>.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The Elements of Statistical Learning*. New York, NY, USA: Springer New York Inc., 2001. (Springer Series in Statistics).

JOBLIB. 2021. <https://joblib.readthedocs.io/en/latest/>. Accessed: 2021-4-01.

KE, G. et al. Lightgbm: A highly efficient gradient boosting decision tree. In: GUYON, I. et al. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017. v. 30. Disponível em: <https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf>.

LIGHTGBM Documentation. 2021. <https://lightgbm.readthedocs.io/en/latest/>. Accessed: 2021-4-01.

MALAWSKI, M. et al. Serverless execution of scientific workflows: Experiments with hyperflow, aws lambda and google cloud functions. *Future Generation Computer Systems*, v. 110, p. 502–514, 2020. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X1730047X>.

MATPLOTLIB. 2021. <https://matplotlib.org/>. Accessed: 2021-4-01.

MAXWELL, A. E.; WARNER, T. A.; FANG, F. Implementation of machine-learning classification in remote sensing: an applied review. *International Journal of Remote Sensing*, Taylor & Francis, v. 39, n. 9, p. 2784–2817, 2018. Disponível em: <https://doi.org/10.1080/01431161.2018.1433343>.

MYSQL. 2021. <https://www.mysql.com/>. Accessed: 2021-4-01.

NAVADA, A. et al. Overview of use of decision tree algorithms in machine learning. In: *2011 IEEE Control and System Graduate Research Colloquium*. [S.l.: s.n.], 2011. p. 37–42.

NUMPY. 2021. <https://numpy.org/>. Accessed: 2021-4-01.

PANDAS. 2021. <https://pandas.pydata.org/>. Accessed: 2021-4-01.

PHILPOT, W. Bathymetric mapping with passive multispectral imagery. *Applied optics*, v. 28, p. 1569–78, 04 1989.

PYMYSQL. 2021. <https://pypi.org/project/PyMySQL/>. Accessed: 2021-4-01.

REQUESTS. 2021. <https://docs.python-requests.org/en/master/>. Accessed: 2021-4-01.

SAGAWA, T. et al. Satellite derived bathymetry using machine learning and multi-temporal satellite images. *Remote Sensing*, v. 11, n. 10, 2019. ISSN 2072-4292. Disponível em: <https://www.mdpi.com/2072-4292/11/10/1155>.

SAMUEL, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, v. 3, n. 3, p. 210–229, 1959.

SCIKIT-LEARN. 2021. <https://scikit-learn.org/stable/>. Accessed: 2021-4-01.

SHAFIEI, H.; KHONSARI, A.; MOUSAVI, P. *Serverless Computing: A Survey of Opportunities, Challenges and Applications*. 2019.

SQLALCHEMY. 2021. <https://www.sqlalchemy.org/>. Accessed: 2021-4-01.

TIFFFILE. 2021. <https://pypi.org/project/tifffile/>. Accessed: 2021-4-01.

UTM. 2021. <https://pypi.org/project/utm/0.4.2/>. Accessed: 2021-4-01.

ZIPFILE. 2021. <https://docs.python.org/3/library/zipfile.html>. Accessed: 2021-4-01.