Cahier des charges du contrôle qualité des surfaces brûlées issues de la chaîne des feux de l'OEIL

Thomas Avron

2024-03-28

Table des matières

1 Introduction

1.1 Contexte du projet

Les feux de forêts représentent un enjeu environnemental majeur en Nouvelle-Calédonie, où les incendies constituent une des premières causes de destruction des milieux naturels. Pour mesure l'ampleur de cette pression et aider à mettre en place des mesures de gestion adaptées, il faut avoir une connaissance fine de la façon dont les écosystèmes sont impactés. C'est dans ce but que l'OEIL, Observatoire de l'Environnement en Nouvelle-Calédonie, avec l'aide de ses partenaires, a développé des outils de surveillance et d'analyse de l'étendue de l'impact environnemental des incendies, sur la base d'images satellitaires.

La haute fréquence de revisite des satellites Sentinel 2 dont la résolution spatial est de 10 mètres (taille du pixel) et l'évolution des méthodes d'analyse des images satellitaires issues de Sentinel 2 ont amené l'OEIL et ses partenaires à construire une chaîne des feux, méthode de détection des surfaces brûlées potentielles, qui s'appuie sur l'analyse des images de Sentinel

2. La fréquence de revisite de 5 jours de ce satellite amène à avoir des données régulièrement, bien que celles-ci ne soient pas forcément exploitables (couvert nuageux).

La chaîne de traitement Détection des surfaces brûlées utilise les données issues de 15 tuiles Sentinel 2 qui couvrent la Nouvelle-Calédonie (la Grande-Terre, les îles Loyauté et les autres îles et îlots proches de la Grande-Terre). Elle intègre les tuiles Sentinel de niveau 2A.

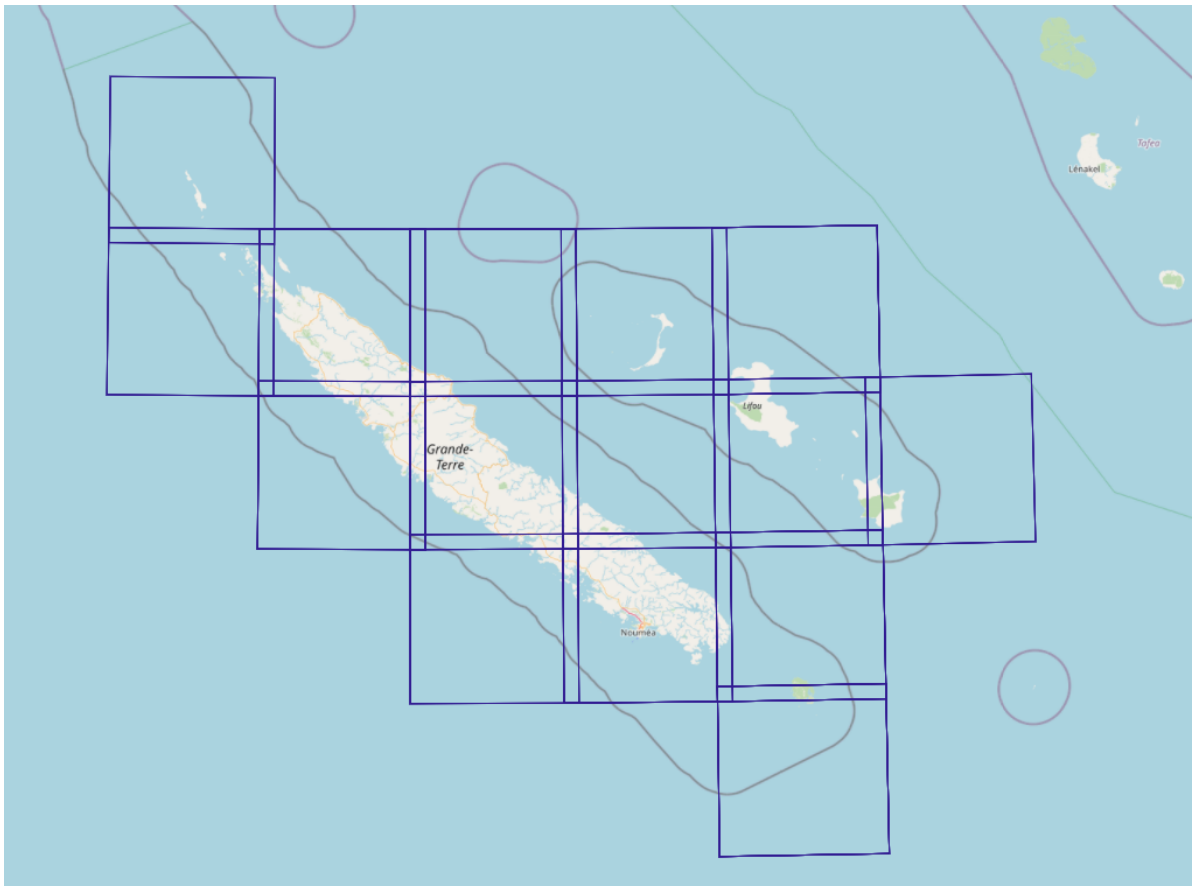

Figure 1: Figure 1 : Tuiles Sentinel 2 sur la NC

La chaîne des feux est une classification supervisée multi-classe effectuée à partir de plusieurs modèles de classification entraînés sur un set d'apprentissage conçu spécifiquement dans ce but. Elle fournit en sortie des données vectorisées de surfaces potentiellement brûlées.

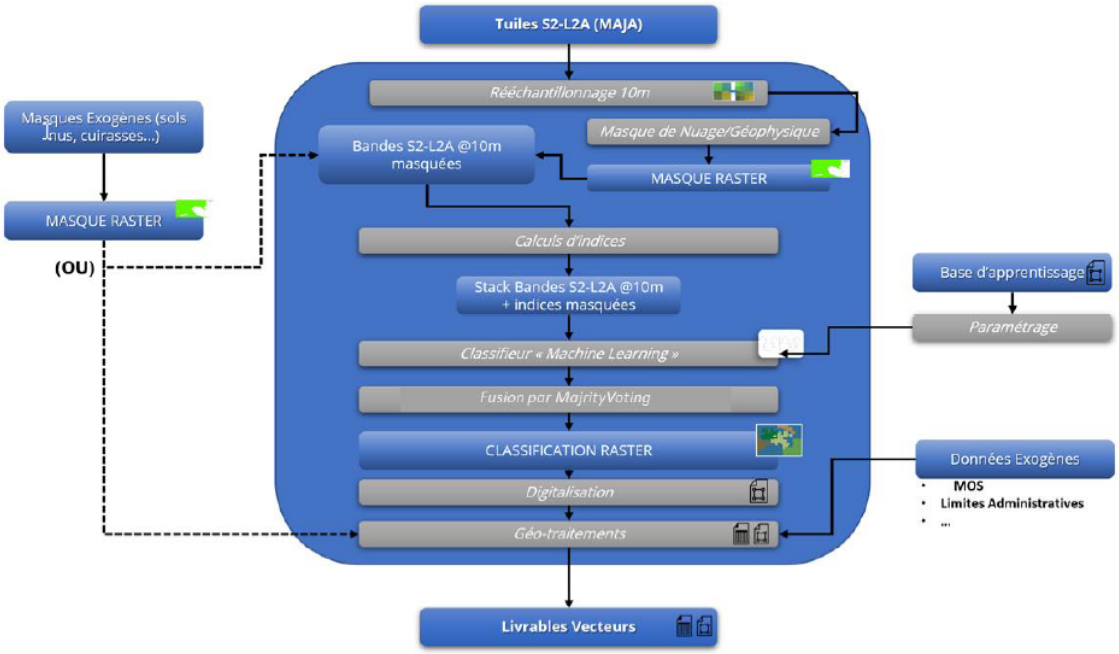

Figure 2: Figure 2 : Algorithmie globale de la chaîne des feux

Cependant suite à la qualification par photo -interprétation des surfaces brûlées potentielles issues de cette chaîne des feux, le besoin de contrôler les données brutes issues de la chaîne de traitement est apparu. Pour cela un projet d'amélioration du dispositif de détection des surfaces incendiées a été réalisé. Une partie de ce projet portait sur la qualification des surfaces détectées selon des indicateurs pertinents, le catalogage de données exogènes, une classification bivariée (surface effectivement brûlée ou non brûlée) et une étude de la validité de l'approche. Ce projet comportait une étude de faisabilité du contrôle qualité des surfaces brûlées en se basant sur des indicateurs issues de la littérature scientifique et sur le benchmarking d'outils disponible pour ce contrôle qualité et une phase de Preuve de Concept (POC) de l'utilisation de ces indicateurs via l'outil retenu par suite du benchmarking de l'étude de faisabilité.

Le présent cahier des charges a pour objet la mise en conformité de production d'algorithmes Python destinés au contrôle qualité des surfaces brûlées potentielles détectées par la chaîne des feux, et l'intégration de ces algorithmes dans le système informatique de l'OEIL. Les algorithmes à mettre en conformité et à intégrer ont été développés au cours du POC du projet d'amélioration du dispositif de détection des surfaces incendiées.

1.2 Objectifs du cahier des charges

Ce cahier des charges vise à définir les spécifications techniques et fonctionnelles des algorithmes Python à développer et à intégrer pour le contrôle qualité des surfaces brûlées. Il decrit le POC réalisé qui servira de base de développement. Sur la description fonctionnelle des snippets de code réalisés durant le POC, sont détaillés les spécifications fonctionnelles des algorithmes Python à développer et à intégrer, ainsi que les objectifs d'intégration dans l'infrastructure existante, avec les contraintes techniques associées, les attendus en termes de performance et les méthodes pour répondre à ces attendus. L'architecture logicielle de l'algorithme est rappelée, puis sont décrits les tests et validations à effectuer, la méthodologie attendue de gestion de projet, un planning et des éléments de réponse à l'appel d'offre du présent cahier des charges.

1.3 Description du POC

Sur la base de la spécification STAC et des outils Python (Pystac et Stackstac) associés, une solution de qualification des surfaces brûlées issues de la chaîne des feux a été retenue.

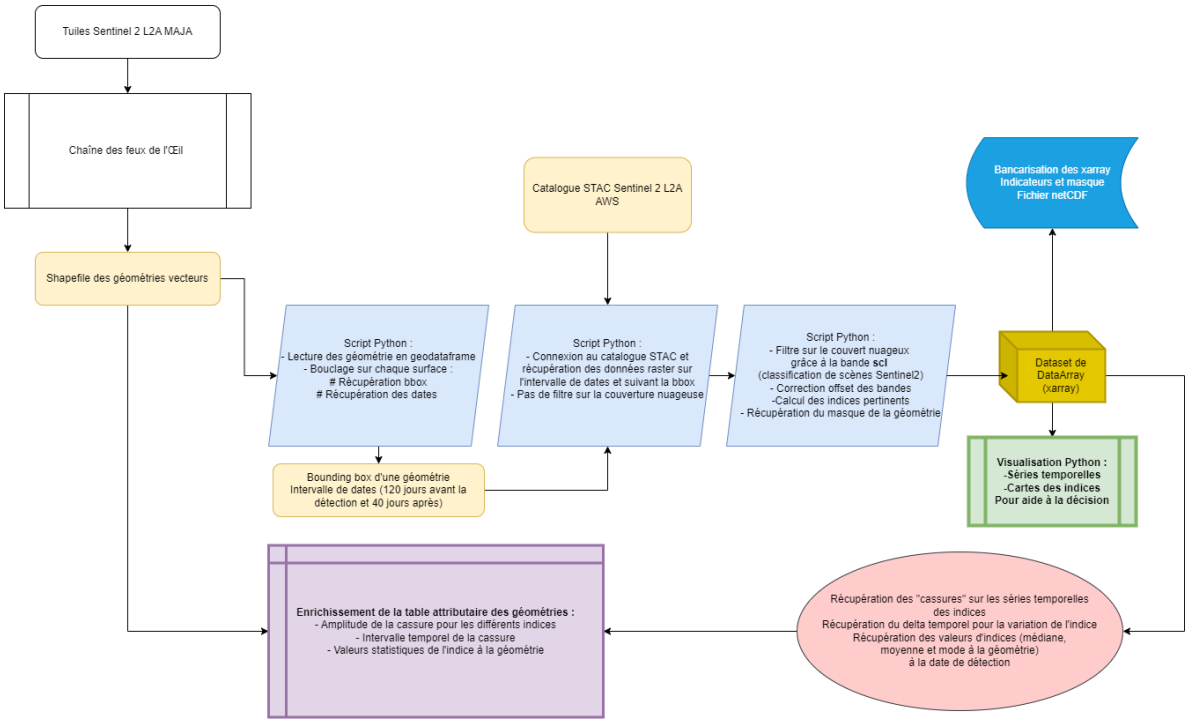

Figure 3: Figure 3 : Schéma des flux de données du POC

Un script Python a été développé avec comme données d'entrées les données vecteurs issues

de la chaîne des feux. Il s'agit de retravailler les fonctions et le script pour rendre le tout compatible à un environnement de production. Le POC a été réalisé sur une station de travail seule, aux performances de calcul limitées (processeur i7 8ième génération à 2,4 GHz et 16 Go de RAM). L'environnement de production dans lequel le script s'inscrira nécessite donc d'adapter les fonctions pour les faire fonctionner sur un cluster de calcul. Mais aussi de gérer les accès et les "credentials" pour récupérer les données d'entrées de l'algorithme et sauvegarder les outputs.

Le cahier des charges fournit la liste des librairies nécessaires à l'éxecution des fonctions développées dans le POC, puis décrit chaque fonction et détaille certaines évolutions nécessaires de ces fonctions pour s'adapter à un environnement de production. Puis la fonction principale `main` du POC est détaillée et la bancarisation des données en netCDF est décrite.

1.3.1 Librairies

Voici les librairies utilisées dans le script du PoC :

```
from pystac_client import Client as psc
import stackstac
import matplotlib.pyplot as plt
import geopandas as gpd
import pandas as pd
import datetime
import numpy

import rioxarray
import rasterio.features
import xarray
from shapely.geometry import mapping
from pyproj import CRS
```

Outre ces librairies fonctionnelles, les librairies suivantes seront nécessaires :

- `dotenv` pour la partie gestion de l'environnement qui permet de développer en s'appuyant sur un fichier de configuration qui passent des variables d'environnement pour gerer les services externes, les credentials…
- `intake` pour le chargement et l'accès aux données
- `dask` pour la parallélisation.

1.3.2 Adaptation de certaines fonctions aux données d'entrées

En input du POC, nous avons un geodataframe qui contient toutes les surfaces brulées à traiter. Voici les fonctions qui réalisent les prétraitements :

```
local_path_to_BA = r"C:\Users\Administrateur\OneDrive\Documents" + \
r"\APID\CLIENTS\OEIL" + \
r"SISSPEO\datas\sentinel_surfaces_detectees_sept_oct_2023.gpkg"
BurnedArea_data = gpd.read_file(local_path_to_BA)


def preprocess_geometries_df(gdf):
  """
  On rajoute une colonne date_ au format datetime et on passe
  de multipolygones à polygon
  On récupère la liste de dates des geométries de surfaces brûlées
  avec un intervalle de temps
  de 120 jours avant la première détection de surface brûlée et
  le geodataframe corrigé (passé de multipolygones à polygones)
  """
  gdf['date_']= pd.to_datetime(gdf['date'], format='%Y-%m-%d').dt.date
  gdf = gdf.explode()
  datemin = (min(gdf['date_']) \
    - datetime.timedelta(days=120)).strftime('%Y-%m-%d')
  datemax = (max(gdf['date_']) \
    + datetime.timedelta(days=60)).strftime('%Y-%m-%d')
  dates = f"{datemin}/{datemax}"
  print(f"Interval temporel {dates}")
  return (gdf, dates)
```

Cette fonction part du geodataframe pour en récupérer la date. La fonction suivante construit une stack de données à partir d'un géodataframe filtré sur une surface brûlée unique.

```
def construct_stack(gdf_filtrer, cc, URL, collection, dates):
  """
  Fonction de construction de la stack à partir
  d'un geodataframe filtré sur une surface brûlée.
  """
  bbox = gdf_filtrer["geometry"].to_crs(4326).total_bounds
  print(f'Emprise spatiale de la géométrie sélectionnée : {bbox}')
  client = psc.open(URL)
  search = client.search(
```

```
    collections=[collection],
    bbox=bbox,
    datetime=dates,
    query={"eo:cloud_cover": {"lt": cc}}

    )

  print(f"{search.matched()} scenes Sentinel-2 L2A trouvées dans
  l'intervalle temporel ayant - de {cc}% de couverture nuageuse")
  items = search.item_collection()
  stack = stackstac.stack(
    items,
    bounds_latlon=[bbox[0], bbox[1],  bbox[2],  bbox[3]],

    gdal_env=stackstac.DEFAULT_GDAL_ENV.updated(
      {'GDAL_HTTP_MAX_RETRY': 3,
      'GDAL_HTTP_RETRY_DELAY': 5,
      }),
    epsg=4326
    ).rename({'x': 'lon', 'y': 'lat'})

  print("stack.crs", stack.crs)

  return stack
```

A l'issue de l'appel de ces fonctions, nous récupérons une `stack` qui contient toutes les scènes associées à une bounding box d'une géométrie et sur un intervalle de date que nous avons pris par défaut à 120 jours avant la date de détection de la surface brûlée par la chaîne des feux, et 60 jours après.

1.3.3 Parallélisation de certaines fonctions

Il faudra donc parraléliser certaines fonctions, notamment les fonctions qui se sont avérées lourdes en temps de calcul sur la station de travail du POC, comme la fonction de sélection des scènes avec un couvert nuageux non restrictif :

```
def select_scenes_without_cc(gdf_filter, stack):
  """
  On utilise la Scene Classification de Sentinel pour vérifier que
  la zone brulée est visible sur l'image.
  """
```

```
data_times = pd.to_datetime(stack['time']).date
dates_burnedarea = gdf_filter['date_'].values
images_to_keep = []

for i, time in enumerate(data_times):
  if time in dates_burnedarea:
    images_to_keep.append(i)
    print(f"on conserve automatiquement l'image {i}")
    continue

  scl_data = stack.isel(time = i).sel(band = "scl")

  mask = (scl_data>=4) & (scl_data<=7)
  filtered_data = scl_data.where(mask)

  percentage = filtered_data.count() / scl_data.count() *100

  if percentage > 95:
    print(f"on prend l'image sufisamment peu couverte {i}")
    images_to_keep.append(i)

data_to_keep = stack.isel(time=images_to_keep)

print(f"Nombre d'images après filtrage :{len(images_to_keep)}")

return data_to_keep
```

Cette fonction de sélection des scènes Sentinel 2 sans couvert nuageux devra être parralélisée avec `dask`. **Description du cluster de calcul par l'OEIL**

1.3.4 Calcul des indices à la volée

Les fonctions suivantes calculent les indicateurs pertinents pour caractériser les surfaces brûlées, en s'appuyant sur les bandes disponibles des images Sentinel 2 L2A. Un offset est appliqué à ces bandes. Il est à prévoir une correction de l'offset selon la formule **L2A_SRi = (L2A_DNi + BOA_ADD_OFFSETi) / QUANTIFICATION_VALUEi**. Lors du POC, ces corrections ont été faites manuellement mais il faut, pour assurer la robustesse du script, accéder à l'offset dans les métadonnées des DataArray. Une investigation est à prévoir pour corriger cet offset et trouver la valeur de quantification associée.

Une piste est d'utiliser les snippets de code suivants :

```
def snippet_indication():
  data_indices = sentinel_stack.to_dataset(dim='band')
  for i in range(len(data_indices.coords['band'].values[()])):
    # On boucle sur les 32 bandes de la stack
    name = data_indices.coords["raster:bands"][i].common_name.values[()]
    # On récupère le nom de la bande pour corriger la bande par la suite
    offset = data_indices.coords["raster:bands"][i].values[()][0]["offset"]
    # On récupère l'offset
    # définir si la quantification_value est bien la coordonnée Scale

    # Faire le calcul
    # data_indices[name] = data_indices[name] - offset / scale ???
```

```
def correction_offset (data_indices):

  data_indices["red"] = data_indices["red"]+0.1
  data_indices["nir"] = data_indices["nir"]+0.1
  data_indices["swir22"] = data_indices["swir22"]+0.1
  data_indices["swir16"] = data_indices["swir16"]+0.1
  data_indices["green"] = data_indices["green"]+0.1
  data_indices["blue"] = data_indices["blue"]+0.1
  data_indices["nir08"] = data_indices["nir08"]+0.1
  data_indices["rededge2"] = data_indices["rededge2"]+0.1
  data_indices["rededge3"] = data_indices["rededge3"]+0.1

  return data_indices
```

Une fois que les bandes nécessaires aux calculs des indicateurs ont été corrigées de l'offset, nous calculons les indices. Nous mettons ici le calcul pour 5 indices : le `ndvi`, le `nbr`, le `nbr+`, le `bais2` et le `badi`.

```
def calcul_indices (data_to_keep):
  """
  On calcule les indices à partir de la stack data_to_keep
  """
  data_indices = data_to_keep.sel(band=["blue","rededge2", "rededge3", "green",
   "red", "nir", "nir08", "rededge1","swir16",
    "swir22", "scl"]).to_dataset(dim='band')

  for elt in data_indices.data_vars :
    data_indices[elt] = data_indices[elt] + 0.1 #Correction de l'offset
    # à supprimer si la fonction de correction est appliquée avant.
```

```
  data_indices['ndvi'] = ((data_indices['nir'] - data_indices['red'])/\
    (data_indices['nir'] + data_indices['red']))

  data_indices['nbr'] = ((data_indices['nir'] - data_indices['swir22'])/\
    (data_indices['nir'] + data_indices['swir22']))

  data_indices['nbr+'] = ((data_indices['swir22'] - \
    data_indices['nir08'] - data_indices['green'] \
    - data_indices['blue'])/(data_indices['swir22'] + \
        data_indices['nir08'] + data_indices['green'] \
        + data_indices['blue']))

  data_indices['bais2'] = (1-(numpy.sqrt((data_indices['rededge2'] \
    * data_indices['rededge3'] \
    * data_indices['nir08'])/data_indices['red']))*\
        ((data_indices['swir22'] -\
         data_indices['nir08'] )/ numpy.sqrt((data_indices['swir22']\
             + data_indices['nir08'] ))+1))

  data_indices['f1'] = ((data_indices['swir22'] + data_indices['swir16']) - \
    (data_indices['nir'] + data_indices['nir08'])) /\
        (numpy.sqrt((data_indices['swir22'] + data_indices['swir16']) +\
             (data_indices['nir'] + data_indices['nir08'])))

  data_indices['f2'] = (2 - numpy.sqrt((data_indices['rededge2']\
    *data_indices['rededge3']*(data_indices['nir'] + data_indices['nir08']))/\
        (data_indices['red']+data_indices['rededge1'])))

  data_indices['badi'] = data_indices['f1']*data_indices['f2']

  return data_indices
```

Les indices ainsi calculés, l'étape suivante est de calculer le masque de la surface brûlée.

1.3.5 Calcul du masque de la surface

```
def create_mask(dataset, gdf):
  """
  On utilise le geodataframe filtré sur la géométrie voulue
  pour créer un masque de la geométrie et le rajouter au dataset
```

```
  """
  ShapeMask = rasterio.features.geometry_mask(gdf['geometry'].\
    to_crs(4326).apply(mapping),
                                              out_shape =(len(dataset.lat),
                                              len(dataset.lon)),
                                              transform = dataset.transform,
                                              invert = True)
  ShapeMask = xarray.DataArray(ShapeMask, dims = ("lat","lon"))
  dataset['mask'] = ShapeMask

  return dataset
```

Une fois le dataset complet, toutes les données à bancariser dans un netCDF sont en mémoire et doivent être enregistrées sur disque.

1.3.6 Première partie du programme `main`

Le programme `main` du POC avait la forme suivante :

```
def main():
  gdf, dates = preprocess_geometries_df(BurnedArea_data)

  # Il faudra remplacer cette étape par
  # la fonction encapsulante qui permettra de boucler
  # sur les surfaces brûlées du géodataframe
  gdf_filter = gdf[gdf["surface_id"]==359594]
  # Fin de l'étape à remplacer

  # Ces autres fonctions seront donc appellées dans une boucle.
  sentinel_stack = construct_stack(gdf_filter, 100, api_url,
  collection, dates)
  data_to_keep = select_scenes_without_cc(gdf_filter, sentinel_stack)
  data_indices = calcul_indices (data_to_keep)
  dataset = create_mask(data_indices, gdf_filter)
  return 0
```

Mais il faudra aussi inclure dans chaque tour de boucle la bancarisation en netCDF :

```
def sauvegarder_netcdf (data_indices, gdf_filtered):
  dataset_save = data_indices.drop([c \
    for c in data_indices.coords if not (c in ['time', 'lat', 'lon'])])
```

```
dataset_save = dataset_save.drop_vars([v \
  for v in dataset_save.data_vars if not (v in ['ndvi', 'nbr',
  'nbr+', 'bais2', 'mask'])])
dataset_save.attrs['spec'] = str(dataset_save.attrs['spec'])
dataset_save.to_netcdf(f"{str(gdf_filtered["surface_id"].iloc[0])}
_{str(gdf_filtered["nom"].iloc[0])}_
{str(gdf_filtered["date"].iloc[0])}.nc")
return dataset_save
```

Dans cette fonction, sont supprimées les coordonnées et les variables du xarray sans utilité. Ensuite, le cast de l'attribut specifications qui est de type RasterSpec en string est effectué, car l'attribut spécifications doit être de type `string` pour sauvegarder le fichier en netCDF.

Une fois ceci fait un appel à la méthode `to_netcdf` d'un dataset xarray est faite pour sauvegarder le dataset xarray en un fichier.

Dans les test de validité, il sera nécessaire de vérifier que le cast de l'attribut spécifications `RasterSpec` n'engendre pas d'erreur par la suite (à la réutilisation des fichiers netCDF).

Le POC décrit plus haut fournit donc la base de code pour l'intégration dans le système de l'OEIL, en production.

2 Objectifs

2.1 Objectifs généraux du développement de l'algorithme

Le POC réalisé a permis de développer les aspects fonctionnels de l'algorithme voulu. Ce dernier doit, à partir d'un `shapefile` des géométries vecteurs issues de la chaîne des feux de l'OEIL :

- Calculer les indicateurs pertinents pour caractériser les surfaces brûlées
- Bancariser les données
- Enrichir la table attributaire des géométries en entrée de données issues des séries temporelles des indices, ainsi que des valeurs d'indices à l'échelle de la surface brûlée.

Les objectifs du développement sont donc d'adapter les snippets de code issus du POC pour les rendre fiables et robustes en production et respecter les contraintes liées à l'infrastructure existante ainsi que les objectifs de performance.

2.2 Objectifs spécifiques liés à l'intégration dans l'infrastructure existante

Les scripts du PoC sont fournis, il est nécessaire de les intégrer de manière fonctionnelle dans un script qui puisse être appelé automatiquement afin de réaliser les traitements avec le minimum d'intervention humaine.

Les bonnes pratiques de l'OEIL seront à respecter dans la structure du code, par exemple avec l'utilisation de `dotenv` pour gérer les chemins d'accès vers les données stockées en local ou gérer les "credentials". Le déploiement devra se faire sous `Docker` pour la contenairisation.

2.3 Objectifs de performance

Certaines fonctions du POC prennaient un temps relativement contraignant à s'éxecuter sur la station de travail seule. Il est demandé d'intégrer aux fonctions "lourdes" en termes de temps de calcul une parallélisation et de distribution avec `Dask`.

3 Spécifications fonctionnelles

3.1 Description détaillée des fonctionnalités de l'algorithme

L'algorithme doit avoir plusieurs fonctionnalités :

3.1.1 Fonctionnalités de gestion des données en entrée

L'algorithme doit :

- Pouvoir lire un fichier shapefile ou un geopackage des surfaces brûlées et les charger en mémoire en géodataframe.
- Pouvoir boucler sur chaque surface pour réaliser les traitements afférents
- Pouvoir récupérer la bounding box pour chaque géométrie de surface brûlée potentielle
- Pouvoir récupérer la date de détection de la surface brûlée pour construire un intervalle de date autour.

3.1.2 Fonctionnalités de récupération des données de catalogue STAC

L'algorithme doit :

- Pouvoir se connecter à un catalogue STAC
- Pouvoir charger les données raster sur l'intervalle de date construit en première partie de l'algorithme et selon la bounding box calculée à partir du filtrage d'une surface brûlée du géodataframe

3.1.3 Fonctionnalités de traitement des données

L'algorithme doit :

- Filtrer les données raster récupérées selon le couvert nuageux grâce à la bande **scl** de classification de scènes Sentinel 2
- Récupérer les information d'offset et de valeur de quantification pour les différentes bandes
- Sélectionner les bandes d'intérêt pour calculer les différents indicateurs
- Corriger les offset et obtenir la réflectance de surface pour les différentes bandes sélectionnées
- Calculer à la volée les indicateurs pertinents sur l'intervalle de date considéré
- Récupérer le masque des géométries et le bancariser dans le DataArray xarray.

3.1.4 Fonctionnalités d'utilisation des indicateurs et de regroupement à l'échelle de la surface brûlée

L'algorithme doit :

- Pouvoir récupérer dans les séries temporelles des indicateurs les "cassures" d'indicateurs
- Pouvoir récupérer le delta temporel pour la "cassure" d'indicateur considérée maximale sur l'intervalle de temps minimal
- Pouvoir récupérer les valeurs d'indices (médiane, moyenne et mode à la géométrie) pour chaque géométrie à la date de détection
- Pouvoir bancariser toutes les données récupérées ainsi en les enregistrant dans la table attributaire de la géométrie considérée

3.1.5 Fonctionnalité de bancarisation de la donnée source

L'algorithme doit :

- Pouvoir bancariser les xarray des indicateurs et du masque de la géométrie dans un fichier netCDF réutilisable

3.2 Exigences en termes d'entrées et de sorties

L'input des géométries vecteurs de l'algorithme doit être récupéré dans la base de donnée de l'OEIL en respectant les bonnes pratiques de l'OEIL. Il doit y avoir possiblité de traiter plusieurs fichiers en entrée de manière séquentielle.

La connexion à un catalogue STAC doit être robuste, le nombre de scènes obtenues lors de la connexion au catalogue STAC doit être rendu disponible dans des logs. De même le nombre

de scènes restantes par géométries après filtrage du couvert nuageux doit être rendu disponible dans des logs. La présence ou l'absence de la scène à la date d'acquisition de la surface brûlée par la chaîne des feux doit être indiquée dans les logs.

L'enrichissement de la table attributaire des géométries devra être discutée avec l'OEIL pour la gestion de l'information. La bancarisation des DataArray en netCDF devra se faire selon une nomenclature discutée avec l'OEIL.

3.3 Gestion des erreurs et des exceptions

Les erreurs potentielles tel que l'absence de données raster sources dans le catalogue STAC source ou encore la situation exceptionnelle ou aucune scène n'est exploitable dans l'intervalle des dates considérées doivent être gérées et ne pas interrompre l'exécution de l'algorithme. Ce genre d'exceptions qui impliquent l'absence d'enrichissement possible de la table attributaire de la géométrie traitée doit être signalée dans les logs. Un rapport en fin de traitement d'un fichier shapefile ou geopackage des géométries vecteurs des surfaces brûlées doit faire remonter toutes les erreurs ou exceptions potentielles.

3.4 Exigences de performance

L'exécution de l'algorithme doit se faire dans un laps de temps suffisamment court pour traiter l'intégralité des surfaces brûlées sur un mois en un temps de l'ordre de l'heure. Cela signifie que le temps de traitement d'une surface brûlée pour bancarisation des xarray et enrichissement de la table attributaire doit être de l'ordre de la dizaine de secondes. En terme d'utilisation des ressources, l'algorithme doit pouvoir s'exécuter sans surcharger la RAM disponible.

4 Spécifications techniques

4.1 Langage de Programmation

Le langage choisi pour l'algorithme est Python. Les librairies utilisées dans le POC fournissent l'intégralité des outils pour répondre aux spécifications fonctionnelles. Les bonnes pratiques de Python (PEP) doivent être respectées.

4.2 Utilisation de la parallélisation et de la distribution

L'utilisation de `Dask` est souhaitée pour répondre aux exigences de performance. Le cluster de calcul disponible à l'OEIL est **à remplir par l'OEIL**.

4.3 Intégration dans l'architecture existante

à remplir par l'OEIL

4.4 Utilisation de solution de containerisation

`Docker` est la solution retenue pour la gestion des conteneurs.

4.5 Utlisation de Pystac et Stackstac

Pour la manipulation des données géospatiales, il est demandé d'utiliser Pystac et Stackstac qui permettent de se connecter à un catalogue STAC et de réaliser des géotraitements sur les données de manière efficace. Le POC réalisé par l'OEIL et ses partenaires fournit les snippets de code nécessaire pour la réalisation des fonctionnalités techniques avec ces librairies.

5 Architecture

5.1 Description de l'architecture logicielle de l'algorithme

L'algorithme doit s'inspirer des scripts développés pour le POC afin de réaliser :

- La lecture et les prétraitements sur le fichier des géométries vecteurs des surfaces brûlées potentielles en entrée
- Les connexions au catalogue STAC pour chaque surface brûlée
- Les géotraitements nécessaires
- La bancarisation des informations obtenues et des données sources

Une proposition est à faire pour la décomposition en modules et en composants si nécessaire.

6 Tests et validation

Une stratégie de tests doit être proposée pour vérifier le bon fonctionnement de l'algorithme. Des tests unitaires, d'intégration et des tests systèmes doivent être réalisé pour assurer la réponse optimale au présent cahier des charges.

7 Gestion de projet

7.1 Méthodologie de développement

Il est demandé de respecter la méthodologie de gestion de projet “Agile”. Sur la base du présent cahier des charges, une phase de lancement de projet permettra de :

- Préparer conjointement avec l’OEIL la “product backlog”
- Valider le “sprint backlog” avec l’OEIL
- Préparer les environnements de développement
- Etablir la gouvernance de projet

Puis sur chaque Sprint est demandé la livraison :

- Le détail des développements livrés
- Les scripts de construction et de déploiement
- Le code des développement
- Les spécifications afférentes au développement
- Les tests unitaires et fonctionnels
- Le rapport d’exécutiond es tests unitaires

Un gestionnaire de version est fourni par l’OEIL (Azure DevOps). Une documentation utilisateur est demandée si nécessaire.

Une Validation d’Aptitude au Bon Fonctionnement sera opérée par l’OEIL pour recette définitive de l’algorithme.

8 Planning

à remplir par l’OEIL

9 Réponse à l’appel d’offre

9.1 Délais de réponse

La date limite de réception des offres est fixée au ** à remplir par l’OEIL **. L’offre sera addressée par voie électronique à l’adresse : hugo.roussaffa@oeil.nc et contact@oeil.nc (prévoir un accusé de réception). Les dossiers parvenus après la date et l’heure limite de réception des offres pourront ne pas être retenus.

9.2 Contacts

Toute demande de précision de quelque nature que ce soit sur la présente consultation se fera par mail.

- Pour les aspects fonctionnels : hugo.roussaffa@oeil.nc
- Pour les aspects techniques : hugo.roussaffa@oeil.nc

Les réponses aux question pourront être diffusées à l'ensemble des postulants si cela est opportun.

9.3 Contenu de l'offre

Il est attendu pour la porposition des offres une proposition technnique et financière détaillée qui doit intégrer :

- La présentation de la structure répondant
- Les CV de chaque intervenant
- L'expérience passée
- Une méthodologie de travail et une réponse technique
- Une ventilation des coûts
- Un calendrier détaillé incluant les délais de livraison du code considéré
- Tout autre élément jugé pertinent pour le présent appel d'offres

9.4 Critères de choix

à remplir par l'OEIL