Flood Mapping and Impact Assessment

Sarwan Shah

I. INTRODUCTION

This paper presents an innovative remote-sensing based application using Google Earth Engine for flood mapping and impact assessment . By integrating multi-temporal satellite imagery, hydrological modeling, and land cover classification datasets, our approach enables near real-time monitoring of flood extent across a specified region. In addition, it provides statistics on the flood's impact, analyzing its extent in relation to agricultural land and urban settlements. We use the 2022 flood of Pakistan, which submerged an estimated one-third of the country, as a benchmark and running example for the testing of our application. This application provides a scalable and easy-to-use solution for non-technical users to quickly map floods and assess the extent of their impact, helping improve the resilience of communities in flood-prone areas by enabling proactive risk management and enabling informed decision making.

In the following sections, we detail the methods used for this mapping and impact assessment, followed by a section where we present the results of our application before concluding with highlights on future work.

II. METHODOLOGY

To map the flood, the approach recommended by the UN-Spider was adopted [6]. This is summarized in figure 1.

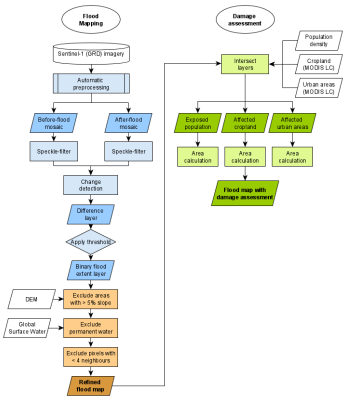

Fig. 1. Methodology

A. Flood Mapping

1) Satellite Sensor : The primary sensor used for mapping the flood was the C-band synthetic aperture radar (SAR) ground range detected (GRD) sensor on the sentinel-1 satellite [1]. The specifications for this sensor are provided in table I. This sensor was natural choice as it works on radar principles by sending microwave pulses to the ground and using the reflected pulses to sense information, unlike optical sensors such as Landsat, which rely on surface & TOA (top of the atmosphere) reflectance, giving the former the advantage of sensing easily in cloud cover conditions, which maybe a likely scenario during a flood event.

Parameter	Value
Spatial Res.	10m, 25m, 40m
Temporal Res.	6 - 12 days
Type	Microwave-based
Instrument Modes	IW, EW, SM
Passes	DESCENDING & ASCENDING
Bands	VV, VH, HH, HV

TABLE I
SENTINEL-1 SAR SPECIFICATIONS.

The aforementioned sensor captures images in 4 polarization settings: VV (vertical Verticaal), VH (Vertical Horizontal), HH, HV, and in three different modes: Interferometric Wide swath (IW), Extra Wide swath (EW), and Strip-map (SM). For the purpose of flood mapping, a ratio of VV & VH polarization was used as it provided a balance between vegetation penetration, back-scattering from water bodies, sensitivity to volume scattering, and a contrast between water bodies and other surfaces. Meanwhile the IW mode provided a good balance between resolution (10m) and coverage area.

Additionally, the image collection used for the mapping was captured in the 'DESCENDING' orbit mode of the sensor, and thus, it is worth noting that this collection may not have images to support mapping for some regions.

2) Identifying Flooded Regions: First, sentinel-1 data using the specifications highlighted in the previous section was filtered and clipped based on a user-defined region of interest (roi). Following this, two time-frames were defined that would capture the state of the roi before and after the flooding event. These time-frames were then used to filter and form two (before and after) image collections from sentinel-1 data, the difference of which would provided a snapshot of the landscape.

In optical data when we want an image from a collection we use the median reducer instead of a mean value to get a snapshot of roi, because presence of clouds and shadows can throw off the mean-value but since sentinel-1 data is not effected by clouds, we directly mosaicked the collections without having to apply any filtering for clouds and this provided us the required snapshots of the roi, as seen in figure 2.

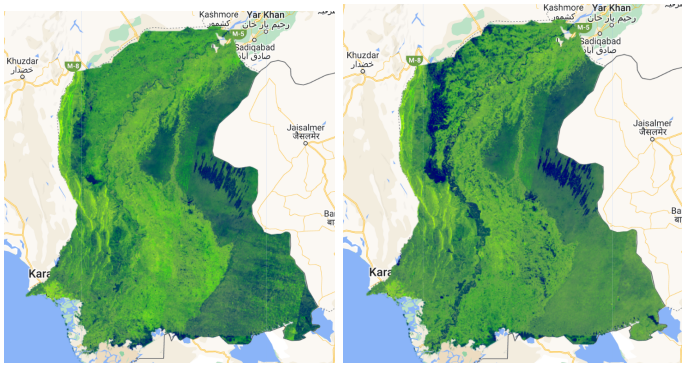

Fig. 2. Before and after flood event mosaicked images - Sindh, Pakistan 2022.

Additionally, a speckle filter was applied to two mosaicked images to remove grainy noise. After this, to achieve an initial map of the flooded area, the difference of the two mosaicked images was taken by dividing one with the other. This is because the image values were converted to decibels by the speckle filter, and moreover, division was more suitable because many changes in SAR images were of multiplicative nature, and division captured that difference, in-addition to having the effect of normalization. Following this, a user-defined thresholding mask was applied on the difference image to identify the flooded areas, as seen in figure 3.

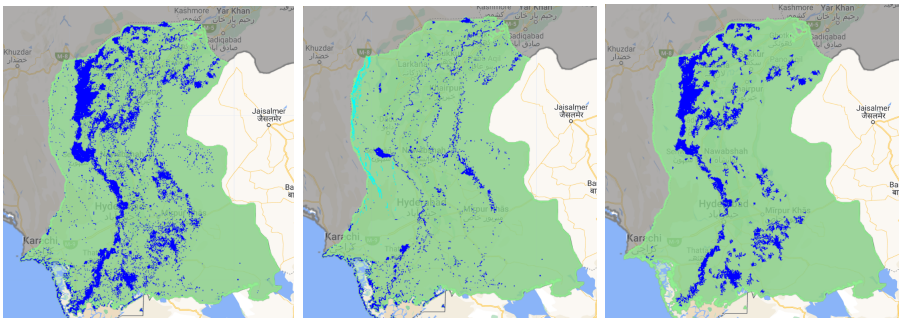

Fig. 3. 1. Initial estimate of flooded region. 2. Permanent water bodies (blue) & slopes (cyan). 3. Final estimate of flooded region

3) Improving Results: To further improve the accuracy of the flood map, a global surface water dataset from the joint research center (JRC) of the European commission that has information about the presence, change, and seasonality of water bodies on the earth's surface was used to identify existing water bodies and their seasonal variation, which was then used to mask out water presence that was abnormal [3]. Additionally, a hydro-sheds dataset from WWF (World Wide Fund for Nature) was used identify terrain with significant slope based on a user-defined threshold, allowing to mask out locations

where flood water could not possibly stand [4]. Lastly, an isolation filter was applied to mask out pixels that had neighbors less than a user specified threshold, considering they represent noise than actually flooded regions. The final estimate of the flood region can be seen in figure 3

B. Impact Mapping & Assessment

1) Satellite Sensor: A MODIS satellite based dataset was used to identify urban and crop land cover distribution in the the region of interest as it has explicit bands that provide this information [2]. The dataset provides an yearly aggregate snapshot of the roi with a spatial resolution of 500m, where the year in our case was chosen was based on the before & after flood event time-frames given by the user. The specifications for the MODIS satellite are given in table II.

Parameter	Value
Spatial Res.	250m, 500m, 100m
Temporal Res.	2 days
Type	Optical
Bands	36

TABLE II

MODIS SPECIFICATIONS.

2) Identifying Urban & Crop Land Cover: To identify areas of the land cover and crop the band's 12 (Crop), 13 (Urban), and 14 (Crop) were extracted from the dataset for the specified year. These bands are explicitly specified to highlight urban and crop land cover, and thus, we clipped and extracted over a region of interest. Additionally, flooded region was masked with the extracted urban and crop land cover to highlight the urban and crop land cover that was impacted by the flood. This is highlighted in figure 4.

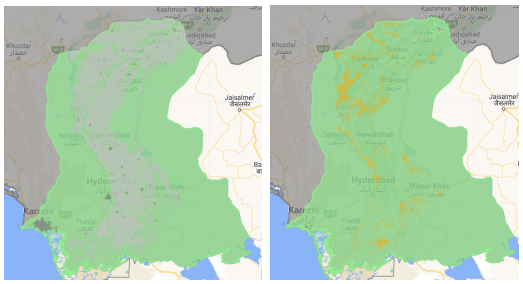

Fig. 4. 1. Urban & crop land cover. 2. Flood impact urban & crop land cover

3) Getting Statistics: Using the various categories of regions identified within the roi, such as the urban and crop regions impacted by the flood, statistics in terms of their area and percentage area were calculated using basic arithmetic and the google earth engine reduceRegion method. These statistics for the 2020 Pakistan floods over the state of Sindh were as specified in table III.

Parameter	Value
Total State Area	13489642 hectares (ha)
Crop Area.	4160490 ha
Urban Area.	106543 ha
Flooded Area	2538551 ha
Crop Flooded Area	1273704 ha
Urban Flooded Area	25222 ha
% Flooded Area.	18.8
% Urban + Crop Flooded Area	30.4

TABLE III

STATISTICS FOR IMPACT ASSESSMENT

C. UI Interfacing

Lastly, the aforementioned methodology was binding together using a UI to make the application dynamic, allowing the user to conduct flood mapping for potentially any region of interest, as seen in figure 5. Datasets from the Food and Agriculture Organization (FAO) of the United Nations were used to retrieve a list of countries, their states, and their respective geographic boundaries [5]. These were binded into UI allowing the user to make selections and the code to automatically fetch the associated boundaries. The code was also divided into modular files and functions that could be passed parameters in most cases, allowing easy integration with UI. The input to these parameters was exposed to the user via a panel interface on UI where the user could input them before running analysis. A list of these parameters that can be controlled by the user and their functions is given in table IV. Additionally, a legend for the flood map and statistics calculated were also exposed to the user using a panel on UI after they had been computed google earth engine.

Parameter	Description	Default Value
Country	country of interest for analysis hectares (ha)	None
State	state of interest for analysis	None
Before Flood Period Start Date	marks the start of the before flood event time-frame	2022-05-15
Before Flood Period End Date	marks the end of the before flood event time-frame	2022-07-15
After Flood Period Start Date	marks the start of the after flood event time-frame	2022-07-15
After Flood Period End Date	marks the end of the after flood event time-frame	2022-07-15
Flooded Area Pixel Threshold	controls sensitivity of flood detection. Adjusted by trail and error based on roi	1.05
Surface Water Seasonal Presence Threshold	controls sensitivity of seasonal surface water presence (0-12)	6
Sloped Area Pixel Threshold	controls sensitivity of sloped surface presence	10
Scale	controls area scale used for calculation of statistics	500
tileScale	controls distribution of computation amongst nodes (1-16). Adjust based on roi	8

TABLE IV
CONTROLLABLE PARAMETERS

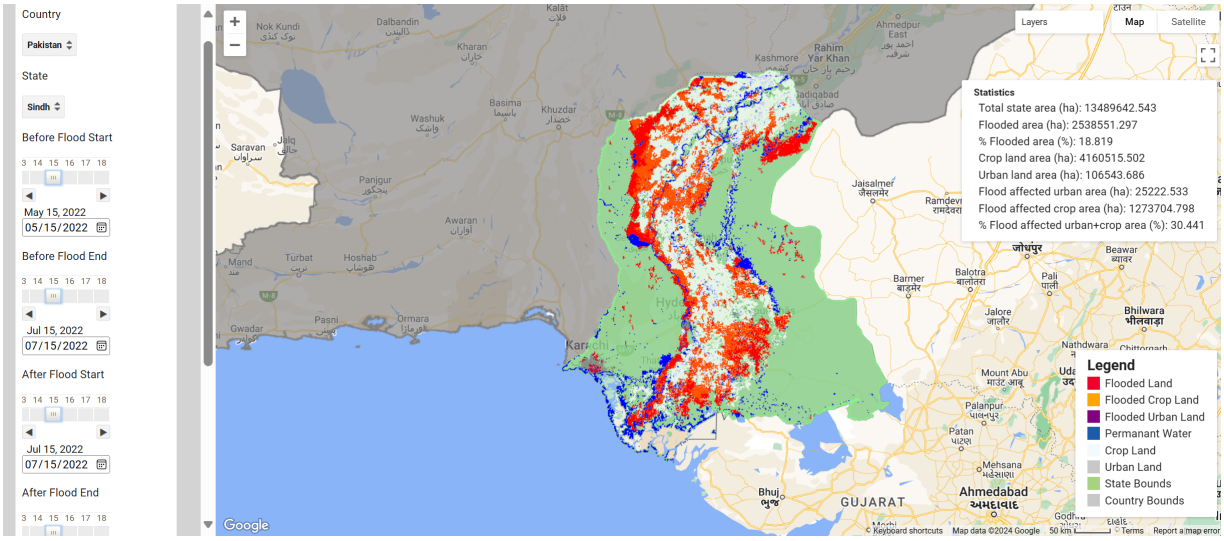

Fig. 5. Flood Mapping Application UI

III. CHALLENGES & FUTURE WORK

Integrating the asynchronous nature of Google Earth Engine (GEE) with the user interface (UI) posed a significant challenge in the application due coordination of asynchronous tasks with synchronous user interactions. Additionally, addressing edge cases where data might not be available or other issues might exist, as well as exposing additional parameters for user control are key areas for future improvement of this application

IV. RESULTS & CONCLUSION

Based on figure 5 and the results demonstrated throughout the methodology section, we see that the presented flood mapping application integrates Sentinel-1 SAR and MODIS satellite data to accurately identify flooded regions and assess their impact on urban and crop land cover in near real-time. In-fact our result of approximately 30% urban & crop land being submerged in the region is close to the officially provided estimates.

By leveraging SAR's cloud-penetrating ability and MODIS's land cover data, the application offers a comprehensive approach to flood monitoring and impact assessment. By employing polarization settings, interferometric modes, and refinements like speckle filtering, it enhances flood extent delineation. Integration of global surface water and terrain datasets ensures precise identification of flooded regions. In the impact assessment phase, MODIS data evaluates flood impact on urban and crop land cover, providing insightful statistics for informed decision-making. The user-friendly interface facilitates easy region definition and parameter adjustment, allowing stakeholders to gain valuable insights quickly. Overall, the application contributes to proactive risk management in flood-prone regions, empowering decision-makers with timely information for effective response strategies.

REFERENCES

[1] European Space Agency (ESA). (Year of data acquisition). Sentinel-1 SAR Ground Range Detected (GRD) data [Data set]. ESA Copernicus Open Access Hub. https://scihub.copernicus.eu/dhus//home

[2] Friedl, M., Sulla-Menashe, D. (2022). MODIS/Terra+Aqua Land Cover Type Yearly L3 Global 500m SIN Grid V061 [Data set]. NASA EOSDIS Land Processes Distributed Active Archive Center. Accessed 2024-05-02 from https://doi.org/10.5067/MODIS/MCD12Q1.061

[3] Jean-Francois Pekel, Andrew Cottam, Noel Gorelick, Alan S. Belward, High-resolution mapping of global surface water and its long-term changes. Nature 540, 418-422 (2016). (doi:10.1038/nature20584)

[4] Lehner, B., Verdin, K., Jarvis, A. (2008): New global hydrography derived from spaceborne elevation data. Eos, Transactions, AGU, 89(10): 93-94.

[5] Food and Agriculture Organization of the United Nations (FAO). (2015). Global Administrative Unit Layers (GAUL) 2015 [Data set]. FAO GeoNetwork. https://data.apps.fao.org/catalog/dataset/global-administrative-unit-layers-gaul-2015

[6] UN-SPIDER. (n.d.). Satellite Image of Flood Mapping. Retrieved May 1, 2024, from https://un-spider.org/advisory-support/recommended-practices/recommended-practice-google-earth-engine-flood-mapping/in-detail

APPENDIX I
CODE

A. FM_UI

```
   // ============================ IMPORTS ================================== //
var main = require('users/sarwanshah1996/SES598-RemoteSensing:FM_Main');
var map = ui.Map();


// Load FAO/GAUL dataset for countries and states
var countries = ee.FeatureCollection("FAO/GAUL/2015/level0");
var states = ee.FeatureCollection("FAO/GAUL/2015/level1");

// Extract country names
var countryNames = countries.aggregate_array("ADM0_NAME").distinct().sort();

// Extract state names for each country
var statesByCountry = {};
countryNames.evaluate(function(countryNamesList) {
  countryNamesList.forEach(function(countryName) {
    var countryStates = states.filter(ee.Filter.eq("ADM0_NAME",
        countryName)).aggregate_array("ADM1_NAME").distinct();
    statesByCountry[countryName] = countryStates;
  });
});

// Define dropdown menus for country and state
var countryInput = ui.Select({

  items: countryNames.getInfo(),
  placeholder: 'Select country'
});

var stateInput = ui.Select({
  placeholder: 'Select state'
});

// Handle country selection
countryInput.onChange(function(country) {
  var countryStates = statesByCountry[country];
  stateInput.items().reset(countryStates.getInfo());
  stateInput.setPlaceholder('Select state');
});


var beforeFloodStartInput = ui.Panel([
  ui.Label('Before Flood Start'),
  ui.DateSlider({
    start: '2000-01-01',
    value: '2022-05-15'
  })
]);

var beforeFloodEndInput = ui.Panel([
  ui.Label('Before Flood End'),
  ui.DateSlider({
    start: '2000-01-01',
    value: '2022-07-15'
  })
]);

var afterFloodStartInput = ui.Panel([
  ui.Label('After Flood Start'),
  ui.DateSlider({
    start: '2000-01-01',
    value: '2022-07-15'
  })
```

```
]);

var afterFloodEndInput = ui.Panel([
  ui.Label('After Flood End'),
  ui.DateSlider({
    start: '2000-01-01',
    value: '2022-09-15'
  })
])

var scaleInput = ui.Textbox('Scale', '500');
var tileScaleInput = ui.Textbox('Tile Scale', '8');

// Define run button
var runButton = ui.Button({
  label: 'Run Analysis',
  onClick: function() {
    main.runAnalysis(
      countryInput.getValue(),
      stateInput.getValue(),
      new
          Date(beforeFloodStartInput.widgets().get(1).getValue()[0]).toISOString().split('T')[0],
      new Date(beforeFloodEndInput.widgets().get(1).getValue()[0]).toISOString().split('T')[0],
      new
          Date(afterFloodStartInput.widgets().get(1).getValue()[0]).toISOString().split('T')[0],
      new Date(afterFloodEndInput.widgets().get(1).getValue()[0]).toISOString().split('T')[0],
      parseInt(scaleInput.getValue()),
      parseInt(tileScaleInput.getValue()),
      map
    );
  }
});


// Define dropdown menus for country and state
var countryInputPanel = ui.Panel([
  ui.Label('Country'), // Add a label for country input
  countryInput
]);

var stateInputPanel = ui.Panel([
  ui.Label('State'), // Add a label for state input
  stateInput
]);

var scaleInputPanel = ui.Panel([
  ui.Label('Scale'), // Add a label for scale input
  scaleInput
]);

var tileScaleInputPanel = ui.Panel([
  ui.Label('Tile Scale'), // Add a label for tile scale input
  tileScaleInput
]);


// Define the panel layout with center-aligned elements
var inputsPanel = ui.Panel({
  widgets: [
    countryInputPanel,
    stateInputPanel,
    beforeFloodStartInput,
    beforeFloodEndInput,
    afterFloodStartInput,
    afterFloodEndInput,
    scaleInputPanel,
    tileScaleInputPanel,
    runButton
```

```
  ],
  layout: ui.Panel.Layout.flow('vertical'),
  style: {
    width: '280px',
    backgroundColor: 'lightgrey', // Center-align the elements
    textAlign: 'center',
    padding: '30px',
    border: '10px',
  }
});


// Add the panel to the map
ui.root.widgets().reset([inputsPanel, map]);


// Define a dictionary which will be used to make legend and visualize image on map
var dict = {
  "names": [
    "Flooded Land",
    "Flooded Crop Land",
    "Flooded Urban Land",
    "Permanant Water",
    "Crop Land",
    "Urban Land",
    "State Bounds",
    "Country Bounds",
  ],
  "colors":[
    "#ED022A",
    "#FFA500",
    "#800080",
    "#1A5BAB",
    "#F2FAFF",
    "#C8C8C8",
    "#A7D282",
    "#C8C8C8",
  ],
  };

// Create a panel to hold the legend widget
var legend = ui.Panel({
  style: {
    position: 'bottom-right',
    padding: '8px 15px'
  }
});

// Create and add the legend title.
var legendTitle = ui.Label({
  value: 'Legend',
  style: {
    fontWeight: 'bold',
    fontSize: '18px',
    margin: '0 0 4px 0',
    padding: '0'
  }
});
legend.add(legendTitle);

var loading = ui.Label('Loading legend...', {margin: '2px 0 4px 0'});
legend.add(loading);

// Creates and styles 1 row of the legend.
var makeRow = function(color, name) {
  // Create the label that is actually the colored box.
  var colorBox = ui.Label({
    style: {
```

```
    backgroundColor: color,
    // Use padding to give the box height and width.
    padding: '8px',
    margin: '0 0 4px 0'
   }
 });

 // Create the label filled with the description text.
 var description = ui.Label({
  value: name,
  style: {margin: '0 0 4px 6px'}
 });

 return ui.Panel({
  widgets: [colorBox, description],
  layout: ui.Panel.Layout.Flow('horizontal')
 });
};

// Get the list of palette colors and class names from the image.
var palette = dict['colors'];
var names = dict['names'];
loading.style().set('shown', false);

for (var i = 0; i < names.length; i++) {
 legend.add(makeRow(palette[i], names[i]));
}

map.add(legend);

// ======================
```

B. *FM_Main*

```
   // ============================ IMPORTS ================================== //

var speckleFilter = require('users/sarwanshah1996/SES598-RemoteSensing:FM_SpeckleFilter');
var region = require('users/sarwanshah1996/SES598-RemoteSensing:FM_FetchRegion');

exports.runAnalysis = function(country, state,
                       beforeStart, beforeEnd,
                       afterStart, afterEnd,
                       scale, tileScale, Map) {


// ============================ DEFINING THE REGION ================================ //
// ============================== AND TIMEFRAME ================================== //


// Call the function and retrieve the boundaries
var boundaries = region.getBoundaries(country, state);
var countryBoundary = boundaries.countryBoundary;
var stateBoundary = boundaries.stateBoundary;

Map.addLayer(countryBoundary, {color: 'grey', fillOpacity: 0, outlineOpacity: 1}, 'Country
    Boundary');
Map.addLayer(stateBoundary, {color: 'lightgreen', fillOpacity: 0, outlineOpacity: 1}, 'State
    Boundary');
Map.centerObject(stateBoundary, 7); // Center the map on the state boundary

var roi = stateBoundary;
var beforeFloodStart = beforeStart;
var beforeFloodEnd = beforeEnd;
var afterFloodStart = afterStart;
var afterFloodEnd = afterEnd;
var dateMODIS = beforeFloodStart.slice(0,4) + '_01_01';
```

```
// ============================ IMPORTING REQUIRED DATASETS
    ================================== //

// Dataset from the joint research center (JRC) of the European
// commission that has information about the presence, change,
// seasonability of water bodies on the earth's surface. We make
// use of this to identify what are existing water bodies and
// their seasonal variation against what is an abnormal presence of
// whatever bodies. Meanwhile the hydrosheds datasets allows us to see
// terrain with slope, allowing to eliminate locations where flood water
// cannot stand.
var gsw = ee.Image("JRC/GSW1_2/GlobalSurfaceWater").clip(roi);
var hydrosheds = ee.Image("WWF/HydroSHEDS/03VFDEM").clip(roi);

// This MODIS based dataset provides information on urban and crop
// land cover distribution that will help us access the impact of
// the flood in terms of its affect on urban and rural populations.
var landCover = ee.Image('MODIS/061/MCD12Q1/' + '2022_01_01').clip(roi)
                  .select('LC_Type1');

// Fetching sentinel-1 (C-band Synthetic Aperture Radar Ground Range Detected)
// SAR GRD data. This data is based on a microwaves and can penetrated clouds.
// It has a spatial resolution of 10m and a temporal resolution of 10 - 12 days
// over south asia.

// It captures data while ascending and descending in orbit across a certain location
// and the data is sensed in two polarization modes, vertical (V) and horizontal (H).
// This enabled four bands: HH, HV, VV, VH

//Additionally. the sensor captures data in 3 modes:
// 1. Interferometric Width Swath (IW) --> Most commonly used
// 2. Extra Wide Swath (EW) --> low resolution, much larger area
// 3. Strip-map (SM) --> high resolution, smaller area

// GEE already performs pre-processing on sentinel 1 data for
// noise removel and radiometric calibration
var sentinel = ee.ImageCollection('COPERNICUS/S1_GRD')
          .filter(ee.Filter.eq('instrumentMode','IW'))
          .filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VH'))
          .filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VV'))
          .filter(ee.Filter.eq('orbitProperties_pass', 'DESCENDING'))
          .filter(ee.Filter.eq('resolution_meters',10))
          .filterBounds(roi)
          .select(['VH', 'VV'])


var addRatioBand = function(image) {
  var ratioBand = image.select('VV').divide(image.select('VH')).rename('VV/VH');
  return image.addBands(ratioBand)
}

// ============================ DRAWING OUT/MAPPING THE FLOOD
    ================================== //

// Gathering images from before the flood and after it
var beforeFloodImages = sentinel.filterDate(beforeFloodStart, beforeFloodEnd)
var afterFloodImages = sentinel.filterDate(afterFloodStart, afterFloodEnd)

// In optical data when we want an image from a collection
// we use the median reducer instead of a mean value, because
// presence of clouds and shadows can throw off the mean-value
// but since sentinel data is not effected by clouds, we can take the mean
// or alternatively, mosaic it without having to apply any filtering for clouds
var beforeFlood = beforeFloodImages.mosaic().clip(roi);
var afterFlood = afterFloodImages.mosaic().clip(roi);

beforeFlood = addRatioBand(beforeFlood);
afterFlood = addRatioBand(afterFlood);
```

```
// The sentinel-1 radar data inherently suffers from grainy noise
// that reduces the quality of features that can derived from the data
// and can result in misleading features as well. A refined lee speckle filter is
// thus applied to image collections, which is an industry standard to
// address this issue. Note: the return images are in decibels

var beforeFloodFiltered = speckleFilter.apply(beforeFlood);
var afterFloodFiltered = speckleFilter.apply(afterFlood);


// Taking the difference between the before flood and
// after flood images by dividing them since they are in decibels.
// Division is suitable because many changes in SAR images are of
// multiplicative nature, and division captures that difference, in-addition
// to having the effect of normalization. Results in a single band image.
var difference = afterFloodFiltered.divide(beforeFloodFiltered);


var floodThres = 1.05;
var flooded = difference.gt(floodThres).rename('floodwater').selfMask();

// Using the global surface water dataset to identify & remove
// permenant/semi-perm surface water from the scene based
// on seasonality. Seasonality band ranges from 0 (never has water)
// to 12 (if has water all 12 months)
var permWater = gsw.select('seasonality').gte(6)
var flooded = flooded.where(permWater, 0).selfMask()
Map.addLayer(permWater.selfMask(), {min:0, max:1, palette: ['blue']}, 'Permanent Water')

// Similarly, using the HydroSHEDS DEM dataset we identify
// and remove areas that have a slope of greater than 10m
// as we would not expect flood water to hold along such slopes
var slopeThres = 10;
var slope = ee.Algorithms.Terrain(hydrosheds).select('slope');
var flooded = flooded.updateMask(slope.lt(slopeThres));
var slopedTerrain = slope.gte(slopeThres).selfMask()

// Remove isolated pixels that maybe just noise or
// very small water bodies
var isolationThres = 4;
var connections = flooded.connectedPixelCount(25)
var flooded = flooded.updateMask(connections.gt(isolationThres))
var noisyTerrain = connections.lte(isolationThres).selfMask()


// ============================ DRAWING OUT/MAPPING URBAN AND CROP LAND
    ================================== //

// Filter urban (13) and cropland (12) classes
var urbanLand = landCover.eq(13);
var cropLand = landCover.eq(12).or(landCover.eq(14));

var urbanCropLand = cropLand.where(urbanLand, 2);
urbanCropLand = urbanCropLand.updateMask(urbanCropLand.neq(0));

var urbanCropLandVis = {
  min: 1, // No urban/crop land (black)
  max: 2, // Urban/crop land (white)
  palette: ['white', 'grey'], // Black for non-urban/crop land, white for urban/crop land
};

Map.addLayer(urbanCropLand, urbanCropLandVis, 'Urban and Crop Land', true, 0.75);


// FINAL FLOOD MAP
Map.addLayer(flooded, {min:0, max:1, palette: ['red']}, 'Flooded Land');
```

```
// ================== IDENTIFYING URBAN AND LAND COVER EFFECTED BY FLOOD
    ====================//

// Masking the urban and crop land cover layers with the flooded area
var floodedUrbanMasked = flooded.updateMask(urbanLand);
var floodedCropMasked = flooded.updateMask(cropLand);

// Visualize the affected urban and crop land
Map.addLayer(floodedUrbanMasked, {min:0, max:1, palette: ['purple']}, 'Flooded Urban Land',
    true, 0.5);
Map.addLayer(floodedCropMasked, {min:0, max:1, palette: ['orange']}, 'Flooded Crop Land',
    true, 0.5);

// ==================== PERFORMING STATISTICAL ANALYSIS ON DATA
    ============================== //

// Calculate Affected Area
var stats = flooded.multiply(ee.Image.pixelArea()).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: roi,
  scale: scale,
  maxPixels: 1e10,
  tileScale: tileScale
});

var totalArea = roi.geometry().area().divide(10000);
var floodedArea = ee.Number(stats.get('floodwater')).divide(10000);
var percFloodedArea = ee.Number(100).multiply(
  floodedArea.divide(totalArea)
);

// Calculate the area of cropland and urban land
var areaCropLand = urbanCropLand.multiply(ee.Image.pixelArea()).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: roi,
  scale: scale,
  maxPixels: 1e12,
  tileScale: tileScale
});

var areaUrbanLand = urbanLand.multiply(ee.Image.pixelArea()).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: roi,
  scale: scale,
  maxPixels: 1e12,
  tileScale: tileScale
});

// Convert area to hectares
var areaCropLandHectares = ee.Number(areaCropLand.get('LC_Type1')).divide(10000);
var areaUrbanLandHectares = ee.Number(areaUrbanLand.get('LC_Type1')).divide(10000);

// Calculate the affected area of urban and crop land
var statsUrban = floodedUrbanMasked.multiply(ee.Image.pixelArea()).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: roi,
  scale: scale,
  maxPixels: 1e10,
  tileScale: tileScale
});

var statsCrop = floodedCropMasked.multiply(ee.Image.pixelArea()).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: roi,
  scale: scale,
  maxPixels: 1e10,
  tileScale: tileScale
```

```
});

// Convert area to hectares
var totalUrbanCropArea = areaCropLandHectares.add(areaUrbanLandHectares);
var floodedUrbanArea = ee.Number(statsUrban.get('floodwater')).divide(10000);
var floodedCropArea = ee.Number(statsCrop.get('floodwater')).divide(10000);
var totalFloodedUrbanCropArea = floodedUrbanArea.add(floodedCropArea);
var percEffectedArea = ee.Number(100).multiply(
  totalFloodedUrbanCropArea.divide(totalUrbanCropArea)
);

// Create an empty ee.List to store stats and values
var statsList = ee.List(['Total state area (ha)', 'Flooded area (ha)', '% Flooded area
    (%)']);
var valuesList = ee.List([totalArea.format('%.3f'), floodedArea.format('%.3f'),
    percFloodedArea.format('%.3f')]);

// Add statistics and values to the lists
statsList = statsList.add('Crop land area (ha)');
valuesList = valuesList.add(areaCropLandHectares.format('%.3f'));

statsList = statsList.add('Urban land area (ha)');
valuesList = valuesList.add(areaUrbanLandHectares.format('%.3f'));

statsList = statsList.add('Flood affected urban area (ha)');
valuesList = valuesList.add(floodedUrbanArea.format('%.3f'));

statsList = statsList.add('Flood affected crop area (ha)');
valuesList = valuesList.add(floodedCropArea.format('%.3f'));

statsList = statsList.add('% Flood affected urban+crop area (%)');
valuesList = valuesList.add(percEffectedArea.format('%.3f'));

// Create a dictionary from lists
var dict = ee.Dictionary({
  stats: statsList,
  values: valuesList
});


var statsPanel = ui.Panel({
  style: {
    position: 'top-right',
    padding: '8px 15px'
  }
});

// Define the function to create a row in the statsPanel
var makeRow = function(stat, val) {
  // Create the label filled with the description text.
  var description = ui.Label({
    value: stat + ": " + val,
    style: {margin: '0 0 4px 6px'}
  });

  // Return a panel containing the description label
  return ui.Panel({
    widgets: [description],
    layout: ui.Panel.Layout.Flow('horizontal')
  });
};

// Define a function to update the statsPanel with statistics from the dictionary
var updateStatsPanel = function(dict) {
  // Clear existing widgets from the statsPanel
  statsPanel.clear();

  // Add the legend title
```

```
  var statsTitle = ui.Label({
    value: 'Statistics',
    style: {
      fontWeight: 'bold',
      fontSize: '12px',
      margin: '0 0 4px 0',
      padding: '0'
    }
  });
  statsPanel.add(statsTitle);

  // Get the lists of statistics and values from the dictionary
  var statsList = dict.get('stats').getInfo();
  var valuesList = dict.get('values').getInfo();

  // Create UI elements for each statistic and value pair
  for (var i = 0; i < statsList.length; i++) {
    var stat = statsList[i];
    var val = valuesList[i];
    var row = makeRow(stat, val);
    statsPanel.add(row);
  }

  // Update the statsPanel on the UI
  Map.widgets().remove(statsPanel); // Remove existing statsPanel from the map
  Map.add(statsPanel); // Add updated statsPanel to the map
};

// Call the updateStatsPanel function to initially display the statistics
updateStatsPanel(dict);

}
```

C. FM_FetchRegion

```
exports.getBoundaries = function(country, state) {
  // Dataset from the Food and Agriculture Organization (FOA)
  // of the United Nations, which provides us with global boundaries
  // on a country (level 0) and state (level 1) level.
  var countries = ee.FeatureCollection("FAO/GAUL/2015/level0");
  var states = ee.FeatureCollection("FAO/GAUL/2015/level1");

  // Filter by country & state
  var countryBoundary = countries.filter(ee.Filter.eq('ADM0_NAME', country));
  var stateBoundary = states.filter(ee.Filter.and(
    ee.Filter.eq('ADM0_NAME', country),
    ee.Filter.eq('ADM1_NAME', state)
  ));

  return { countryBoundary: countryBoundary, stateBoundary: stateBoundary };
}
```

D. FM_SpeckleFilter

```
    //############################
// Speckle Filtering Functions
//############################

exports.apply = function(img) {
  return ee.Image(toDB(RefinedLee(toNatural(img))));
}

// Function to convert from dB
```

```
function toNatural(img) {
  return ee.Image(10.0).pow(img.select(0).divide(10.0));
}

//Function to convert to dB
function toDB(img) {
  return ee.Image(img).log10().multiply(10.0);
}

//Apllying a Refined Lee Speckle filter as coded in the SNAP 3.0 S1TBX:

//https://github.com/senbox-org/s1tbx/blob/master/s1tbx-op-sar-processing/src/main/java/org/esa/s1tbx/sa
//Adapted by Guido Lemoine

// by Guido Lemoine
function RefinedLee(img) {
  // img must be in natural units, i.e. not in dB!
  // Set up 3x3 kernels
  var weights3 = ee.List.repeat(ee.List.repeat(1,3),3);
  var kernel3 = ee.Kernel.fixed(3,3, weights3, 1, 1, false);

  var mean3 = img.reduceNeighborhood(ee.Reducer.mean(), kernel3);
  var variance3 = img.reduceNeighborhood(ee.Reducer.variance(), kernel3);

  // Use a sample of the 3x3 windows inside a 7x7 windows to determine gradients and
      directions
  var sample_weights = ee.List([[0,0,0,0,0,0,0], [0,1,0,1,0,1,0],[0,0,0,0,0,0,0],
      [0,1,0,1,0,1,0], [0,0,0,0,0,0,0], [0,1,0,1,0,1,0],[0,0,0,0,0,0,0]]);

  var sample_kernel = ee.Kernel.fixed(7,7, sample_weights, 3,3, false);

  // Calculate mean and variance for the sampled windows and store as 9 bands
  var sample_mean = mean3.neighborhoodToBands(sample_kernel);
  var sample_var = variance3.neighborhoodToBands(sample_kernel);

  // Determine the 4 gradients for the sampled windows
  var gradients = sample_mean.select(1).subtract(sample_mean.select(7)).abs();
  gradients = gradients.addBands(sample_mean.select(6).subtract(sample_mean.select(2)).abs());
  gradients = gradients.addBands(sample_mean.select(3).subtract(sample_mean.select(5)).abs());
  gradients = gradients.addBands(sample_mean.select(0).subtract(sample_mean.select(8)).abs());

  // And find the maximum gradient amongst gradient bands
  var max_gradient = gradients.reduce(ee.Reducer.max());

  // Create a mask for band pixels that are the maximum gradient
  var gradmask = gradients.eq(max_gradient);

  // duplicate gradmask bands: each gradient represents 2 directions
  gradmask = gradmask.addBands(gradmask);

  // Determine the 8 directions
  var directions =
      sample_mean.select(1).subtract(sample_mean.select(4)).gt(sample_mean.select(4).subtract(sample_mean
  directions =
      directions.addBands(sample_mean.select(6).subtract(sample_mean.select(4)).gt(sample_mean.select(4).
  directions =
      directions.addBands(sample_mean.select(3).subtract(sample_mean.select(4)).gt(sample_mean.select(4).
  directions =
      directions.addBands(sample_mean.select(0).subtract(sample_mean.select(4)).gt(sample_mean.select(4).
  // The next 4 are the not() of the previous 4
  directions = directions.addBands(directions.select(0).not().multiply(5));
  directions = directions.addBands(directions.select(1).not().multiply(6));
  directions = directions.addBands(directions.select(2).not().multiply(7));
  directions = directions.addBands(directions.select(3).not().multiply(8));

  // Mask all values that are not 1-8
  directions = directions.updateMask(gradmask);
```

```
  // "collapse" the stack into a singe band image (due to masking, each pixel has just one
      value (1-8) in it's directional band, and is otherwise masked)
  directions = directions.reduce(ee.Reducer.sum());

  //var pal = ['ffffff','ff0000','ffff00', '00ff00', '00ffff', '0000ff', 'ff00ff', '000000'];
  //Map.addLayer(directions.reduce(ee.Reducer.sum()), {min:1, max:8, palette: pal},
      'Directions', false);

  var sample_stats = sample_var.divide(sample_mean.multiply(sample_mean));

  // Calculate localNoiseVariance
  var sigmaV =
      sample_stats.toArray().arraySort().arraySlice(0,0,5).arrayReduce(ee.Reducer.mean(),
      [0]);

  // Set up the 7*7 kernels for directional statistics
  var rect_weights =
      ee.List.repeat(ee.List.repeat(0,7),3).cat(ee.List.repeat(ee.List.repeat(1,7),4));

  var diag_weights = ee.List([[1,0,0,0,0,0,0], [1,1,0,0,0,0,0], [1,1,1,0,0,0,0],
    [1,1,1,1,0,0,0], [1,1,1,1,1,0,0], [1,1,1,1,1,1,0], [1,1,1,1,1,1,1]]);

  var rect_kernel = ee.Kernel.fixed(7,7, rect_weights, 3, 3, false);
  var diag_kernel = ee.Kernel.fixed(7,7, diag_weights, 3, 3, false);

  // Create stacks for mean and variance using the original kernels. Mask with relevant
      direction.
  var dir_mean = img.reduceNeighborhood(ee.Reducer.mean(),
      rect_kernel).updateMask(directions.eq(1));
  var dir_var = img.reduceNeighborhood(ee.Reducer.variance(),
      rect_kernel).updateMask(directions.eq(1));

  dir_mean = dir_mean.addBands(img.reduceNeighborhood(ee.Reducer.mean(),
      diag_kernel).updateMask(directions.eq(2)));
  dir_var = dir_var.addBands(img.reduceNeighborhood(ee.Reducer.variance(),
      diag_kernel).updateMask(directions.eq(2)));

  // and add the bands for rotated kernels
  for (var i=1; i<4; i++) {
    dir_mean = dir_mean.addBands(img.reduceNeighborhood(ee.Reducer.mean(),
        rect_kernel.rotate(i)).updateMask(directions.eq(2*i+1)));
    dir_var = dir_var.addBands(img.reduceNeighborhood(ee.Reducer.variance(),
        rect_kernel.rotate(i)).updateMask(directions.eq(2*i+1)));
    dir_mean = dir_mean.addBands(img.reduceNeighborhood(ee.Reducer.mean(),
        diag_kernel.rotate(i)).updateMask(directions.eq(2*i+2)));
    dir_var = dir_var.addBands(img.reduceNeighborhood(ee.Reducer.variance(),
        diag_kernel.rotate(i)).updateMask(directions.eq(2*i+2)));
  }

  // "collapse" the stack into a single band image (due to masking, each pixel has just one
      value in it's directional band, and is otherwise masked)
  dir_mean = dir_mean.reduce(ee.Reducer.sum());
  dir_var = dir_var.reduce(ee.Reducer.sum());

  // A finally generate the filtered value
  var varX =
      dir_var.subtract(dir_mean.multiply(dir_mean).multiply(sigmaV)).divide(sigmaV.add(1.0));

  var b = varX.divide(dir_var);

  var result = dir_mean.add(b.multiply(img.subtract(dir_mean)));
  return(result.arrayFlatten([['sum']]));
}
```