SMART INDIA HACKATHON

PROBLEM STATEMENT – PS1735:

- ➔ On-device semantic segmentation of WMS services with geospatial data export
- ➔ Description: *Develop a mobile or desktop (qgis plugin) or web application that uses on-device GPU/NPU for interactive semantic segmentation on images loaded using WMS service. Challenge: To ensure the system is user-friendly and accessible, even for non-technical users. To utilize the computational power of GPUs/NPUs to enhance the performance and responsiveness of the system and reduce reliance on server-side GPU compute. Usage: Useful for assisting on-screen digitization for various remote sensing applications. Users: WebGIS application developers and the end users of these applications. Available Solutions (if yes, reasons for not using them): Individual components are available, comprehensive and proven solution does not exist. Desired Outcome: 1. The tool should be compatible with and OGC compatible WMS service. 2. It should provide data export in geospatial format of user selected features (geojson/kml). 3. It should make maximum utilisation of on-device GPUs/NPUs available in modern desktop/mobile devices.*

(left) a satellite image and (right) the semantic classes in the image.

OVERVIEW:

Development of an application with following features:

1. Shows Maps and Satellite Images: Like Google Maps, but the images come from a WMS service (which is a special map server).
2. Uses AI to Recognize Objects: The AI can look at the map and highlight areas (like fields, buildings, roads) using something called semantic segmentation.
3. Works Fast on Devices: The app should use the power of your phone's or computer's processor (GPU/NPU) to quickly do the AI work without needing a big server.
4. Export Data: After the AI finds those areas, you can save that information in a format like GeoJSON or KML, which is useful for mapping apps.

WORKFLOW:

1. **Technology Stack**
    a. *Web*: React.js with WebGL (Three.js or TensorFlow.js for GPU processing)
    b. *Mobile:* React Native (TensorFlow Lite or ONNX for mobile GPU/NPU inference)
    c. *Desktop/QGIS Plugin:* Python with PyQt and OpenCV (for QGIS Plugin development)
    d. Serverless functions (for minimal processing)
    e. Geospatial libraries (GeoPandas, Shapely, GDAL)
    f. Semantic Segmentation Models:
        i. Pre-trained models like DeepLabV3, UNet, or MobileNet for segmentation.
        ii. Model deployment on mobile using TensorFlow Lite or Core ML for iOS.
    *g. GIS Libraries**:*
        i. Geospatial Data Handling: GDAL/OGR, Fiona (for GeoJSON/KML exports).
        ii. Map Rendering: OpenLayers or Leaflet.js (for WMS integration).

****Geographic Information System**. It's a system that helps us capture, store, analyze, and display data related to positions on Earth's surface. In simple terms, GIS is used for working with maps and location data.

Examples of GIS include:

- **Google Maps**: It shows where things are located and how to navigate from one place to another.
- **Weather Maps**: Showing temperature, rainfall, and other weather data over different regions.

What is a WMS Service?

WMS stands for Web Map Service. It's like an online library of maps. When you ask it for a map of a certain area (like your city), it gives you that map as an image. You can zoom in, pan around, and even stack different layers (like roads, buildings, vegetation) on top of each other.

How Does a WMS Work?

- Imagine you want to see a map of your neighborhood.

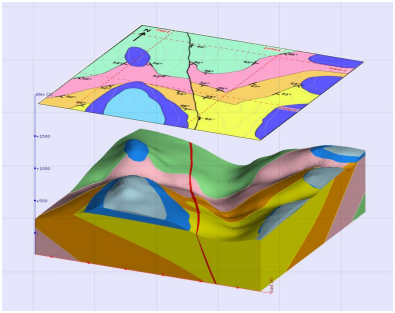
- The WMS service has a huge collection of map images. When you ask it for a specific location, it sends you the right map tiles (small map pieces) for that area.

- You can think of it like asking a librarian for a book. You tell them what you want (location, zoom level), and they give you the exact pages (map tiles) you need.

2. **WMS Service Integration**
    a. Using OpenLayers (web) or Leaflet to load WMS layers from an OGC-compliant server.
    b. Implementing pan, zoom, and interactive layers on the map.
    c. Extract the required tile images from the WMS service for segmentation

3. **On-Device Semantic Segmentation**
    a. Loading the model onto the device using TensorFlow.js (web) or TensorFlow Lite (mobile).
    b. For desktop, leveraging PyTorch or TensorFlow with CUDA if available.
    c. Processing images in real-time using WebGL or device-native libraries.

4. **User-Friendly Interface**
    a. Development of an intuitive UI for non-technical users:
    b. Interactive Segmentation: Allowing users to click or draw polygons on areas they want to segment.
    c. Live Feedback: Displaying segmented outputs in real-time as users interact with the map.
    d. Using WebGL shaders or GPU-accelerated libraries for smooth interaction and visualization.



5. **Data Export (GeoJSON/KML)**
    a. After segmenting, converting the selected features into GeoJSON or KML using GDAL/OGR (Python) or JavaScript libraries (turf.js for GeoJSON).
    b. Allowance for easy export and integration with other GIS tools.

6. **Optimizing GPU/NPU Utilization**
    a. Using TensorFlow Lite's GPU delegate on mobile devices.
    b. For web, using TensorFlow.js with WebGL backend for GPU acceleration.
    c. On desktop, enabling CUDA (if available) for PyTorch or TensorFlow.

7. **Accessibility and Usability**
    a. Provide simple toggle switches for different layers and segmentation modes.
    b. Include tooltips, guided steps, and help sections for non-experts.
    c. Make the installation/deployment straightforward for QGIS plugin or web applications.

8. **Testing**

a. Ensure compatibility with various WMS standards.
b. Optimize model size and inference time to avoid lag in real-time applications.
c. Test across multiple devices (desktop GPUs, mobile NPUs) to ensure smooth performance.

ML Semantic Segementaion:

Think of it like this: You have a puzzle (the map image), and the AI is the person coloring different pieces based on what they represent (roads, trees, etc.).

1. Getting the Location and Map Data:

- The WMS service gives you map images for any location you pick.

- the app asks the WMS service for the area you're interested in (like your city or neighborhood) and gets that image.

2. Applying Semantic Segmentation:

- Semantic segmentation is a fancy word for "teaching the AI to recognize different things in the image."

- We can use a pre-trained model (like DeepLabV3 or UNet). These models already know how to recognize common things in maps or satellite images.

3. How Does This Work on user Devices?

- The application loads the map image and passes it through the AI.

- The AI uses the GPU/NPU to quickly highlight those different objects in real time.

4. Saving the Data:

- Once the AI has colored in the objects, it can be saved as GeoJSON or KML.

- These formats are like special map files that other apps can use to understand what's been highlighted.

1. Getting the Map: Using a WMS service (like Google maps or OLA maps) to fetch map images for a location.

2. AI model: Using a pre-built segmentation model and train it further if needed. (fine-tuning)

3. Processing on Device: Making sure the app uses the GPU/NPU so it runs fast, even on a phone.

4. Exporting Data: Adding a feature to save the highlighted areas in GeoJSON or KML.

1. Choosing a WMS Provider:

- There are many WMS servers available, both public and private. Some popular ones are:

  - OpenStreetMap WMS

  - NASA Earthdata

  - Sentinel Hub WMS (for satellite imagery)

- These servers store different types of maps, like road maps, satellite images, terrain maps, etc.

2. Getting the WMS URL:

- Each WMS service has a URL that you use to ask for map data. For example, OpenStreetMap might have a URL like:

https://wms.openstreetmap.org/

https://ows.terrestris.de/osm/service?

https://gibs.earthdata.nasa.gov/wms/epsg4326/best/wms.cgi

- We'll need this URL to connect your app to the service.

3. Requesting a Map for a Specific Location:

- WMS works using a system called *GetMap requests.* This is how client asks the server for a specific piece of the map.

  - Bounding Box (BBOX): The coordinates of the area you want to see.

  - Layers: What you want to show (roads, terrain, satellite view, etc.).

  -Projection: The map's coordinate system (usually something like EPSG:4326).

  - Format: The image type (e.g., PNG, JPEG).

https://wms.example.com/wms?service=WMS&version=1.1.1&request=GetMap&layers=example-layer&bbox=-180,-90,180,90&width=800&height=600&srs=EPSG:4326&format=image/png

- This URL gives you a map image of the world (based on the coordinates).

4. Display the Map in the App:

- Using libraries like *Leaflet.js* (for web apps) or *OpenLayers* to load and displaying these WMS images as interactive maps.

- These libraries handle things like zooming, panning, and stacking multiple layers.

1. **Connecting to the WMS service:** Using the service URL.
2. **Requesting the map tiles:** Based on the user's location and zoom level.
3. **Rendering the map:** Using a library to display the images.

```
const map = L.map('map').setView([51.505, -0.09],
13);
L.tileLayer.wms('https://wms.example.com/wms', {
    layers: 'example-layer',
    format: 'image/png',
    transparent: true,
    attribution: "Map data © Example"
}).addTo(map);
```

***This code connects to a WMS service, fetches map tiles, and displays them on a web map.*

Overlaying Segmentation Results on the Map

- After running the segmentation, we get output like colored masks or polygons that highlight different areas (e.g., buildings, roads, vegetation).

- The app needs to take this segmentation output and overlay it on top of the map. This can be done by converting the segmented regions into g*eospatial polygons.*

- For instance, a segmented road can be converted into a polygon and drawn on the map using a mapping library like *Leaflet.js* (for web) or *Mapbox* (for mobile).

I

Allowing Users to Interact with the Segmentation

- Letting users refine or modify the segmented areas. For instance, users might click on a polygon to see details or manually adjust the boundaries.

- The app can have options like:

- *Selecting Segmented Regions:* Users can click on specific areas to select them.

- *Editing or Refining Segments:* Users can adjust the boundaries or labels if the AI made a mistake.

Exporting the Segmentation Data (GeoJSON/KML)

- Once the segmentation is done and refined, you should allow users to export the data in geospatial formats:

- *GeoJSON:* Widely used format for storing geographic data as JSON.

- *KML (Keyhole Markup Language):* Commonly used in applications like Google Earth.

*** This is crucial because it lets users save the segmented areas and use them in other GIS tools.*

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [
            [-122.4, 37.8],
            [-122.5, 37.7],
            [-122.6, 37.8],
            [-122.4, 37.8]
          ]
        ]
      },
      "properties": {
        "class": "road"
      }
    }
  ]
}
```

How to do this:

- Converting the segmented polygons into GeoJSON or KML using libraries like:

- **turf.js** (for GeoJSON in JavaScript)

- **GDAL/OGR** (for KML in Python)

** The coordinates are the vertices of the polygon, and "class" specifies the segmented object.

Optimizing Performance

- Making sure that the app runs smoothly, especially when dealing with large map areas or complex segmentations:

- *Memory Management:* Ensuring that the app efficiently handles memory when dealing with large images.

- *GPU/NPU Utilization:* Continuously optimizing the usage of on-device GPUs/NPUs to maintain high-speed performance.

- *Real-time Feedback:* Keeping the interaction fast and responsive by optimizing the segmentation model (like using TensorFlow Lite on mobile).

Making the App User-Friendly

- Addition of a clear and intuitive UI:

- *Layer Control:* Lets users toggle between different layers (e.g., satellite view, segmented view).

- *Zoom and Pan:* Ensuring that the users can easily explore the map.

- *Tooltips and Help:* Including instructions or guides for non-technical users.

Examples of UI features:

- A sidebar to control layers.

- Buttons to export the segmentation as GeoJSON/KML.

- A toolbox for drawing, editing, or selecting polygons.