C.1 Computational Framework for Indigenous Territory Land Cover Analysis

This appendix provides detailed documentation of the computational framework developed to analyze historical land cover change within indigenous territories in Brazil from 1985 to 2023. The analysis leverages MapBiomas Collection 8 data to quantify land cover transitions, persistence patterns, and change dynamics that support the policy analysis presented in the phd project proposal.

https://github.com/leandromet/nlp_project_cuda/blob/main/raster_proc/brazil_self_color_table_polygon.py - MIT license

C.1.1 Methodological Overview

The analysis utilizes a custom Python-based geospatial processing framework that handles the computational challenges of analyzing high-resolution (30m) land cover data across large regions, while maintaining fine-scale detail necessary for policy-relevant insights. The processing pipeline includes:

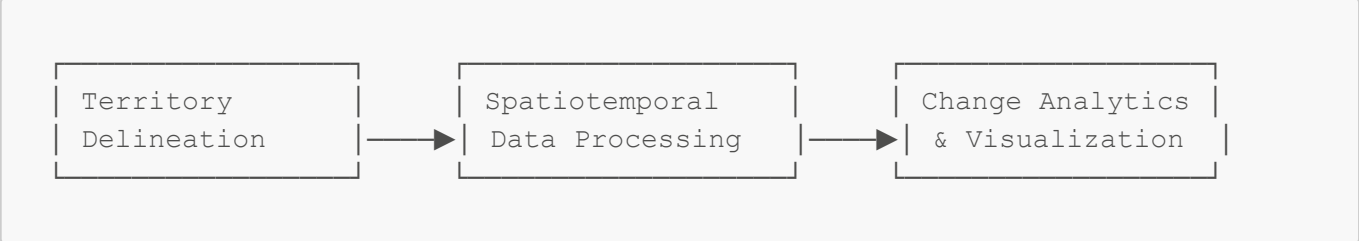
1. **Polygon-based extraction** from continental-scale data
2. **Temporal change detection** across 39 annual land cover maps
3. **Transition matrix calculation** for all land cover class combinations
4. **Statistical analysis** of land cover persistence and change frequency
5. **Visualization generation** for both spatial patterns and transition flows

C.1.2 Data Sources and Structure

The implementation works with the following data inputs:

- **Raster Data**: MapBiomas Collection 8 (1985-2023) in Virtual Raster (VRT) format
    - 39 annual layers with 30m resolution
    - EPSG:4326 coordinate reference system
    - 35+ land cover classes following the MapBiomas classification system
- **Vector Data**: Indigenous territory boundaries in GeoJSON format
    - Includes formal boundaries of indigenous territories in Brazil
    - Properties include official territory name (`terrai_nom`) and other attributes

C.1.3 Processing Architecture



The processing is optimized through:

- **Tile-based parallel processing** to enable efficient computation

- **Chunked array storage** using Zarr for memory efficiency
- **Window-aligned reading** for optimal I/O performance
- **Robust error handling** with automatic recovery mechanisms
- **In-memory caching** of frequently accessed data

C.2 Technical Implementation Details

C.2.1 Land Cover Change Detection

The change detection algorithm identifies pixels that change land cover class between consecutive years:

```
def calculate_changes_tile(args):
    """Calculate changes for a single tile with robust error handling."""
    # Read first year data
    prev_data = robust_read(src, 1, window)

    for year_idx in range(2, src.count + 1):
        # Read current year data
        current_data = robust_read(src, year_idx, window)

        # Identify changed pixels
        changed = prev_data != current_data
        changes += changed.astype('uint8')

        # Update transition matrix for statistics
        if np.any(changed):
            from_vals = prev_data[changed]
            to_vals = current_data[changed]

            for from_val, to_val in zip(from_vals, to_vals):
                transition_matrix[from_val, to_val] += 1

        prev_data = current_data
```

C.2.2 Class Persistence Analysis

The persistence analysis identifies pixels that maintained the same land cover class throughout the entire study period:

```
def calculate_class_persistence(src, window):
    """Calculate which pixels remained in the same class for all years."""
    # Read first year as reference
    reference = robust_read(src, 1, window)
    persistence = np.ones_like(reference, dtype=bool)

    # Compare against all subsequent years
    for year_idx in range(2, src.count + 1):
        current = robust_read(src, year_idx, window)
        persistence &= (reference == current)
```

```
    # Return both the persistence mask and the reference classes
    return persistence, reference
```

C.2.3 Geospatial Processing with Indigenous Territory Boundaries

The analysis uses polygon boundaries to clip and analyze only the relevant areas:

```
# Transform polygons to pixel coordinates relative to the window
transform_window = src.window_transform(full_window)

# Rasterize polygon boundaries for masking
mask = rasterize(
    all_polygons,
    out_shape=mask_shape,
    transform=transform_window,
    fill=0,
    default_value=1,
    dtype=np.uint8
)

# Apply mask to changes data
full_changes[~mask_bool] = FILL_VALUE
```

C.3 Analytical Outputs and Visualizations

C.3.1 Land Cover Maps

The framework produces georeferenced land cover maps, compatible with GIS software through world files (PNGW):

```
def create_landcover_map(root, output_dir):
    """Create land cover visualization with PNGW for GIS compatibility."""
    # Create colormap from MAPBIOMAS colors
    cmap = ListedColormap([COLOR_MAP.get(i, '#ffffff') for i in
range(max(COLOR_MAP.keys()) + 1)])

    # Generate visualization
    plt.figure(figsize=(12, 20))
    plt.imshow(last_year, cmap=cmap, vmin=0, vmax=max(COLOR_MAP.keys()))

    # Create PNGW world file for georeferencing
    pngw_path = output_path + 'w'
    with open(pngw_path, 'w') as pngw_file:
        pngw_file.write(f"{transform.a}\n")  # Pixel size in x-direction
        pngw_file.write(f"{transform.b}\n")  # Rotation
        pngw_file.write(f"{transform.d}\n")  # Rotation
        pngw_file.write(f"{transform.e}\n")  # Pixel size in y-direction
        pngw_file.write(f"{transform.c}\n")  # X-coordinate of the upper-
```

```
left
        pngw_file.write(f"{transform.f}\n")  # Y-coordinate of the upper-
left
```

C.3.2 Transition Flow Visualization

The framework generates Sankey diagrams to visualize land cover class transitions during specified time periods:

```
def create_sankey(transition_matrix, classes, title, output_html,
output_csv):
    """Create a Sankey diagram showing land cover flows between classes."""
    # Calculate flow metrics
    total_flow = np.sum(transition_matrix)
    out_flows = np.sum(transition_matrix, axis=1)
    in_flows = np.sum(transition_matrix, axis=0)

    # Position nodes proportionally to flow magnitude
    left_y = calc_positions(out_flows)
    right_y = calc_positions(in_flows)

    # Create interactive Sankey diagram with Plotly
    fig = go.Figure(go.Sankey(
        arrangement="fixed",
        node=dict(
            pad=10,
            thickness=10,
            line=dict(color="black", width=0.3),
            label=node_labels,
            color=node_colors,
            x=node_x,
            y=node_y
        ),
        link=dict(
            source=sources,
            target=targets,
            value=values,
            color=link_colors
        )
    ))
```

C.3.3 Persistence Analytics

The framework quantifies and visualizes the stability of land cover within indigenous territories:

```
def create_persistence_visualization(root, output_dir):
    """Create stacked bar chart showing persistent vs changed pixels by
class."""
    # Calculate changed counts (initial - persistent)
```

```
    changed_counts = initial_counts - persistence_counts

    # Create stacked bar chart showing stability by class
    plt.figure(figsize=(14, 10))
    bars_changed = plt.bar(labels, changed, color=colors, alpha=0.6,
label='Changed')
    bars_persistent = plt.bar(labels, persistent, bottom=changed,
color=colors, alpha=1.0)
```

C.4 Policy Analysis Integration

This computational framework supports policy analysis in multiple ways:

1. **Quantitative Impact Assessment**: Provides precise measurements of policy impacts on land cover change rates and patterns within indigenous territories
2. **Temporal Alignment with Policy Implementation**: Change analysis can be temporally aligned with specific policy implementation dates
3. **Comparative Analysis**: Enables comparisons of land cover change between protected and unprotected areas
4. **Evidence-Based Evaluation**: Generates empirical evidence on effectiveness of conservation and indigenous rights policies
5. **Visual Communication**: Creates compelling visualizations for policy communication to diverse stakeholders

C.5 Technical Limitations

Several technical considerations should be noted:

1. **Classification Accuracy**: Analysis inherits any misclassifications present in the source MapBiomas data
2. **Temporal Resolution**: Annual snapshots may miss seasonal or sub-annual changes
3. **Spatial Resolution**: 30m resolution may not capture fine-scale degradation processes
4. **Edge Effects**: Boundaries between land cover classes may show classification instability
5. **Processing Requirements**: Full analysis requires substantial computational resources (32GB+ RAM recommended)

This appendix documents the technical implementation that underpins the policy analysis presented in the project proposal. The computational framework enables robust, repeatable, and transparent analysis of land cover change in indigenous territories, providing an empirical foundation for policy evaluation.