

GeoFlood Enhancement for Robust Flood Inundation Mapping in Flat Terrain Zones

Marwa Wahba ¹, Ayman G. Awadallah ^{1,*}, Nabil A. AwadAllah ¹ and Maysara Ghaith ^{2,3,*}

¹ Department of Civil Engineering, Fayoum University, Fayoum 63514, Egypt; mw1146@fayoum.edu.eg (M.W.)

² Department of Irrigation and Hydraulic, Faculty of Engineering, Cairo University, Giza 12613, Egypt

³ INTERFACE Institute for Multi-Hazard Systemic Risk Studies, McMaster University, Hamilton, ON L8S 4L8, Canada

* Correspondence: aga00@fayoum.edu.eg (A.G.A.); ghaithm@mcmaster.ca or maysaraghaith@cu.edu.eg (M.G.)

The primary purpose of this supplementary material is to describe how the original GeoFlood framework, presented in Figure 2 of the main manuscript, was enhanced through the integration of the CPOP approach. While the overall structure of the GeoFlood framework was preserved, several targeted adjustments were implemented to enable the application of the CPOP method. Both the original and modified workflows are implemented in Python.

The original GeoFlood framework was developed by Zheng et al.[1], and the source code and accompanying guidance materials are publicly available at: <https://github.com/passaH2O/GeoFlood>. Zheng et al. [1] recommended using an Anaconda environment to run the GeoFlood modules; however, any Python-compatible environment may be used, including execution via the Windows command prompt. Required dependencies, such as GRASS GIS and TauDEM, can be installed either prior to setting up the Python environment or configured within it.

The GitHub repository outlines the original GeoFlood workflow as a sequence of twenty steps. The modified CPOP-GeoFlood framework follows the same overall structure and numbering. All files and steps 1 to 12 remain unchanged from the original implementation, while modifications were introduced from Step 13 onward, and the corresponding new files are provided as supplementary data and also available at <https://github.com/marwawahba/Geoflood-Enhancement.git>. The main objective of this document is to clearly identify and explain these code-level changes, enabling straightforward replication of the methodology presented in the main manuscript.

In addition, some modifications were introduced to support uncertainty analysis conducted prior to identifying CPOP as the preferred segmentation method. Not all presented modifications are strictly required for implementing the CPOP-GeoFlood framework; however, they are included here for completeness and to provide guidance for users interested in extending or adapting the methodology.

Step 13 Streamline Segmentation Modification

Two streamline segmentation approaches are examined in this study. The first approach follows the original GeoFlood methodology and is based on equal-length streamline segmentation. Multiple segment lengths are tested to evaluate how segmentation length choice influences slope estimation and the sensitivity of GeoFlood results.

The second approach applies the CPOP algorithm, which does not depend on predefined or fixed segment lengths. Instead, CPOP automatically identifies optimal streamline breakpoints, producing segments of unequal length that are determined by terrain-driven changes in slope. This data-adaptive segmentation enables a more robust and physically meaningful representation of longitudinal slope variability along the stream network.

Equal Length Streamline Segmentation

In the original GeoFlood framework, Step 13 (Streamline_Segmentation.py) divides the extracted flowline into equal-length segments based on a user-defined segment length. The default segmentation length is 1000 m, as shown in Figure SA1. This procedure produces a single output corresponding to the specified segmentation length.

Instead of applying the original code multiple times according to different segment lengths, the updated code automatically processes all possible segmentation lengths in a single run, following the same idea of breaking the flowline into equal-length segments. Therefore, the updated code applies an alternative loop, as shown in Figure SA2, which tests a full range of split distance values from 100 m up to the total length of the streamline with 50-meter increments. These values can be modified by the user based on the case study and sensitivity or uncertainty analysis.

For each segmentation length, the updated code generates a separate output file and stores it in an individual directory. All segmentation-specific directories are organized under a newly created parent folder named ChannelSegments, as shown in Figure SA3.

It should be noted that although some subsequent GeoFlood processing steps do not require changes to the methodology, they must be updated to reference the appropriate input files in the ChannelSegments directory to ensure consistency with the new segmentation workflow.

```
def main():
    geofloodHomeDir, projectName, DEM_name, chunk_status, input_fn, output_fn, hr_status = cfg_finder()
    geofloodResultsDir = os.path.join(geofloodHomeDir, output_fn,
                                      "GIS", projectName)
    Name_path = os.path.join(geofloodResultsDir, DEM_name)
    in_shp = Name_path + "_channelNetwork.shp"
    out_shp = Name_path + "_channelSegment.shp"
    split_distance = 1000
    network_split(in_shp, out_shp, split_distance)
    #segment_shp = gpd.read_file(out_shp)
    #print(type(segment_shp))
    #segment_shp = segmenet_shp[segment_shp.Length !=0]
    #segment_shp.to_file(out_shp, driver="ESRI Shapefile")
```

Figure SA1: Original GeoFlood default streamline segmentation.

```
def main():
    geofloodHomeDir, projectName, DEM_name, chunk_status, input_fn, output_fn, hr_status = cfg_f
    geofloodResultsDir = os.path.join(geofloodHomeDir, output_fn, "GIS", projectName)
    Name_path = os.path.join(geofloodResultsDir, DEM_name)
    in_shp = Name_path + "_channelNetwork.shp"

    # Base folder
    base_out_folder = os.path.join(geofloodResultsDir, "ChannelSegments")
    os.makedirs(base_out_folder, exist_ok=True)

    for d in range(100, 1801, 50):
        # folder for each segmentstion length
        out_folder = os.path.join(base_out_folder, f"d_{d}")
        os.makedirs(out_folder, exist_ok=True)

        # same file name
        out_shp = os.path.join(out_folder, f"{DEM_name}_channelSegment.shp")

        print(f"Processing split distance = {d} ...")
        network_split(in_shp, out_shp, d)

    print("All segmentations completed successfully.")
```

Figure SA2: Updated code for streamline segmentation loop.

d_100	1/21/2026 1:30 PM	File folder
d_150	1/21/2026 1:30 PM	File folder
d_200	1/21/2026 1:30 PM	File folder
d_250	1/21/2026 1:30 PM	File folder
d_300	1/21/2026 1:30 PM	File folder
d_350	1/21/2026 1:30 PM	File folder
d_400	1/21/2026 1:30 PM	File folder
d_450	1/21/2026 1:30 PM	File folder
d_500	1/21/2026 1:31 PM	File folder
d_550	1/21/2026 1:31 PM	File folder
d_600	1/21/2026 1:31 PM	File folder
d_650	1/21/2026 1:31 PM	File folder
d_700	1/21/2026 1:31 PM	File folder
d_750	1/21/2026 1:31 PM	File folder
d_800	1/21/2026 1:31 PM	File folder
d_850	1/21/2026 1:31 PM	File folder
d_900	1/21/2026 1:31 PM	File folder
d_950	1/21/2026 1:31 PM	File folder
d_1000	1/21/2026 1:31 PM	File folder
d_1050	1/21/2026 1:31 PM	File folder
d_1100	1/21/2026 1:31 PM	File folder
d_1150	1/21/2026 1:31 PM	File folder
d_1200	1/21/2026 1:31 PM	File folder

Figure SA3: Subfolders creation based on different segmentation lengths.

Unequal-length Streamline segmentation

The alternative approach of segmentation code was applied to break the original flowline into unequal-length segments based on the CPOP method obtained from Fearnhead et al. [2]. The original CPOP source code, along with detailed instructions and guidance materials, is publicly available at <https://github.com/bastienlc/CPop>. The CPOP method depends mainly on two parameters (Beta and Scale), which the user can initially determine based on the description in the above link and could be adjusted based on the model output.

The original CPOP code obtained from GitHub was updated to ensure full compatibility with the GeoFlood workflow. The CPOP GeoFlood code begins with the user definition of the flowline folder paths, as shown in Figure SA4. CPOP then takes sample elevation values along the flowline at a fixed 10-meter interval (user-defined value) as shown in Figure SA5. The next step, the CPOP method extracts the elevation profile and detects optimal changepoints along the streamline and reconstructs piecewise-linear segments between these detected points with their calculated slopes. The code also produces a diagnostic plot showing the elevation profile, identified changepoints, and resulting segmentation as shown in Figure SA6. This visualization enables the user to assess whether further adjustment of the Beta and Scale parameters is required. Beta parameter controls the number of changepoints detection along the flowline, higher values produce fewer and smoother segments; meanwhile, lower values produce more segments according to the sensitivity of the DEM. In contrast, the Scale parameter is used in the cost function to stabilize and balance the optimization process without altering the overall characteristics of the elevation signal.

The code finally generates the output file with unequal segmentation lengths and their associated slopes based on the detected change points. The output file is stored in a new dedicated subfolder (d_No_smoothing_CPOP_realSlope) within the Channel_Segments directory.

```

from src import CPOP, LogCost, continuous_pieewise_linear_approximation, reconstruct_segmentation

def median_abs_deviation(arr):
    arr = np.asarray(arr)
    med = np.median(arr)
    return np.median(np.abs(arr - med))

flowline_path = r"C:\Users\user\Desktop\GeoFlood-master\GeoInputs\GIS\my_project\Flowline.shp"
dem_path = r"C:\Users\user\Desktop\GeoFlood-master\GeoInputs\GIS\my_project\dem.tif"
output_folder = r"C:\Users\user\Desktop\GeoFlood-master\GeoOutputs\GIS\my_project\ChannelSegments\d_NO_smoothing_CPOP_realSlope"
os.makedirs(output_folder, exist_ok=True)

flow_gdf = gpd.read_file(flowline_path)
line = flow_gdf.geometry.iloc[0]
dem = rasterio.open(dem_path)

```

Figure SA4: CPOP flowlines identification.

```

# -----
def sample_line(line, step=10):
    distances = np.arange(0, line.length + 1e-6, step) # include end
    points = [line.interpolate(d) for d in distances]
    return points, distances

points, distances = sample_line(line, step=10)
coords = [(p.x, p.y) for p in points]
elev = np.array([val[0] for val in dem.sample(coords)])

y_vals = elev.copy()

line_length = line.length
if line_length < 500:
    beta = 2
    scale = 1
elif line_length < 2000:
    beta = 2
    scale = 1
else:
    beta = 10
    scale = 1

```

Figure SA5: CPOP elevation sampling.

```

# -----
def sample_line(line, step=10):
    distances = np.arange(0, line.length + 1e-6, step) # include end
    points = [line.interpolate(d) for d in distances]
    return points, distances

points, distances = sample_line(line, step=10)
coords = [(p.x, p.y) for p in points]
elev = np.array([val[0] for val in dem.sample(coords)])

y_vals = elev.copy()

line_length = line.length
if line_length < 500:
    beta = 2
    scale = 1
elif line_length < 2000:
    beta = 2
    scale = 1
else:
    beta = 10
    scale = 1

```

Figure SA6: CPOP parameterization.

Data Management (Folder Data Code)

An additional data-management step is required to accommodate the updated segmentation workflow. Although this step is currently implemented as a separate post-processing code, it can be automated in future versions of the code to be within the equal segmentation code. In the original GeoFlood equal-length segmentation procedure, the number of stream segments is calculated by dividing the total stream length by the user-defined segmentation length. This value

is then rounded up to the nearest integer, and a revised segment length is computed by dividing the total stream length by the resulting integer number of segments.

This rounding procedure may produce redundancy in the final segment lengths. For example, if the total stream length is 1780 m and the requested segmentation length is 1100 m, the calculated number of segments is 1.61, which is rounded up to 2, resulting in an effective segment length of 890 m. Similarly, a requested segmentation length of 1000 m yields a calculated value of 1.78, which is also rounded up to 2, producing the same effective segment length of 890 m. As a result, multiple user-defined segmentation lengths can lead to identical segmentation outputs.

After updating the code to process all segmentation lengths in a single run while maintaining consistency with the original GeoFlood equations, an additional script is applied to identify and remove redundant outputs. This script scans the generated output folders, detects repeated effective segment lengths, and retains only unique segment length configurations for subsequent processing steps of GeoFlood.

This process is done based on a CSV file generated containing the folder names and their corresponding effective segment lengths. The data are then sorted and filtered based on segment length, and duplicate entries are removed by retaining only a single representative folder for each unique segmentation. Corresponding folders with identical names are then created in the inundation directory to ensure consistency in subsequent GeoFlood framework steps.

When the unequal length segmentation based on the CPOP method is applied, this additional filtering step is not required, as only one output is generated. However, the Folder Data code in this case is executed, which will keep the extracted segmentation folder (d_No_smoothing_CPOP_realSlope) preserved as is, and only a corresponding folder with the same name will be created within the inundation directory for subsequent GeoFlood processing steps.

Step 14 GRASS GIS Catchment Delineation

The GRASS GIS Catchment Delineation script was updated to operate on all segmentation output files generated from Step 13. This modification in the original code makes it process multiple extracted files sequentially instead of only a single input file.

As shown in Figure SA7, the only required change to the original GeoFlood Step 14 implementation is an update to the directory path. Once this path is modified to point to the parent directory containing the segmentation subfolders, the script automatically iterates through all available subfolders and applies the catchment delineation procedure to each segmentation scenario without further manual intervention.

```
def main():
    # Paths
    geofloodHomeDir = r"C:\Users\user\Desktop\GeoFlood-master"
    DEM_name = "dem"
    projectName = "my_project"

    geofloodResultsDir = os.path.join(geofloodHomeDir, "GeoOutputs", "GIS", projectName)
    Name_path = os.path.join(geofloodResultsDir, DEM_name)
    fdrfn = Name_path + '_fdr.tif'

    # Folder with multiple subfolders containing shapefiles
    channel_segments_folder = os.path.join(geofloodResultsDir, "ChannelSegments")

    # Loop over each subfolder
    for subfolder in os.listdir(channel_segments_folder):
        folder_path = os.path.join(channel_segments_folder, subfolder)
        if os.path.isdir(folder_path):
            segshp = os.path.join(folder_path, "{}_channelSegment.shp".format(DEM_name))
            if not os.path.exists(segshp):
                print("Shapefile not found: {}, skipping...".format(segshp))
                continue
            segcatfn = os.path.join(folder_path, "{}_segmentCatchment.tif".format(DEM_name))
            print("Processing {} -> {}".format(segshp, segcatfn))
            segment_catchment_delineation(fdrfn, segshp, segcatfn)

    print("All segmentations completed successfully.")
```

Figure SA7: GRASS GIS catchment delineation updated script.

Step 15 River Attribute Estimation

Step 15 consists of multiple scripts designed to support different slope estimation approaches. When applying the original GeoFlood methodology with equal-length segmentations, the user can execute the River_Attribute_Estimation-

main script, which follows the default slope calculation procedure for each segmentation scenario (base case scenario). Alternatively, the user can apply other slope estimators, including mean slope, median slope, and Theil-Sen trend estimation, with the option to consider either all calculated slopes or only positive slopes, as shown in Figure SA8. Alternatively, the user can use the CPOP method for unequal stream segmentation. All those methods were tested in this study against the base case scenario, but the primary focus of this study, based on results, focuses primarily on two slope estimation approaches: the Theil-Sen trend with slope ranges and the CPOP-based slope method, as they provide comparable results to the original base case scenario.

The Theil-Sen trend with ranges is applied using elevation sampling at each 10-m interval (this can be manually adjusted by the user). Within each stream segment, all pairwise slopes are computed, and their median is taken as a robust slope estimate. A final slope value is then selected by comparing the estimated slope against a user-defined, physically reasonable slope range. This filtering step allows unrealistic values caused by DEM noise or local irregularities to be excluded, based on the user's experience and the observed longitudinal behavior of the stream, as shown in Figure SA9. Meanwhile, river attribute estimation using the CPOP approach does not involve additional slope calculations at this step. Instead, the script directly reads the segment slopes previously computed by the CPOP algorithm in Step 13 from the corresponding output folder and assigns them to the river attributes used in subsequent GeoFlood processing steps.

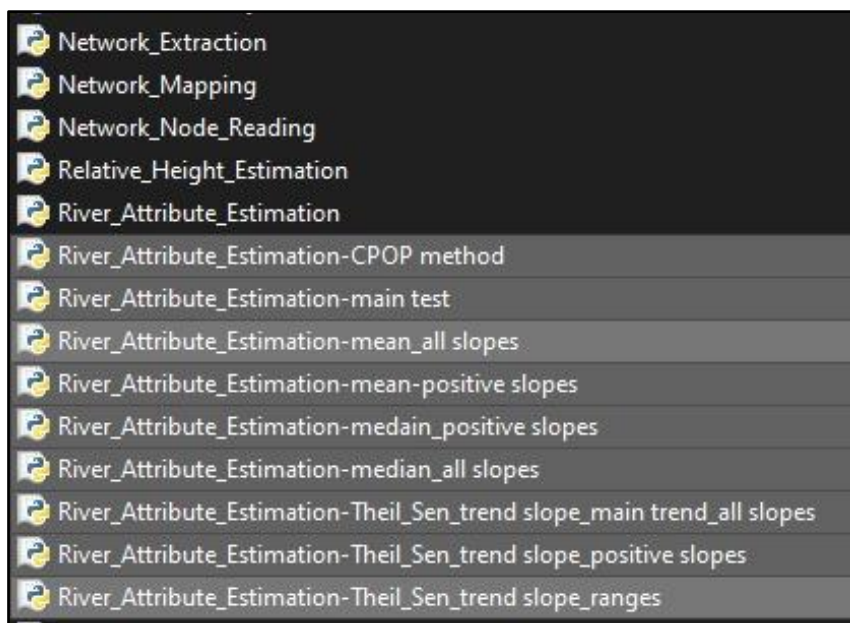


Figure SA8: River attribute slope estimation scripts.

```

slope = compute_segment_slope(m_array, z_array, step=10)

cumulative_length += length

if cumulative_length < 1:
    min_slope = 0.0006
    max_slope = 0.0014
elif 1 <= cumulative_length <= 2:
    min_slope = 0.0024
    max_slope = 0.0056
elif 2 <= cumulative_length <= 3.8:
    min_slope = 0.0036
    max_slope = 0.0084
elif 3.8 <= cumulative_length <= 6.6:
    min_slope = 0.00078
    max_slope = 0.00182
else:
    min_slope = 0.0048
    max_slope = 0.0112

if slope < min_slope:
    slope = min_slope
elif slope > max_slope:
    slope = max_slope

```

Figure SA9: Theil-Sen trend slope estimation with ranges.

Step 16 Network Mapping

The network mapping script was updated to operate on all files generated from the previous Step. The only modification to the original implementation is the extension of the workflow to process multiple extracted input files rather than a single file. No changes were made to the underlying network mapping logic; the update enables automated iteration over all segmentation outputs.

Step 17 Hydraulic Property Base Table

Similarly, the Hydraulic Property Base Table script was updated to operate on all files generated in the previous step. The sole modification to the original implementation is the extension of the script to process multiple extracted input files rather than a single file. The underlying hydraulic property calculations remain unchanged; the update enables automated iteration across all segmentation outputs.

Step 18 Hydraulic Property Full Table (More Channel Geometries)

The Hydraulic Property Full Table script was updated to operate on all files generated in the previous step. The only modification to the original implementation is the extension of the workflow to process multiple extracted input files instead of a single file. All hydraulic property calculations and underlying logic remain unchanged; the update solely enables automated iteration across the full set of segmentation outputs.

Step 19 National Water Model Forecast

In the original GeoFlood framework, discharge values are provided using outputs from the National Water Model Forecast. To support the updated multi-segmentation workflow, the discharge processing script was modified to operate on all files generated in the previous step, extending its functionality from a single input file to multiple files.

The updated script is renamed (Forecast_Table_different_Q), which incorporates two main modifications. First, the input data format was changed from NetCDF to CSV to improve flexibility and ease of use. The location of the CSV input file can be manually specified by the user, as shown in Figure SA10. Second, the structure of the discharge input data was revised. Instead of assigning a single discharge value per stream, as in the original GeoFlood implementation, the updated CSV file associates each stream ID (COMID) with multiple discharge values corresponding to different return

periods. This modification enables direct evaluation of multiple flood scenarios within the same workflow, as shown in Figure SA11.

```
comid_var[:] = df['COMID'].values
hydroid_var[:] = df['HYDROID'].values
q_var[:] = df['Q'].values
h_var[:] = df['H'].values

rootgrp.close()
print(f"NetCDF saved: {nc_output}")

# Main
if __name__ == '__main__':
    input_file = r'C:\Users\user\Desktop\GeoFlood-master\GeoInputs\Net\my_project\Q_values.xlsx'

    geofloodHomeDir, projectName, DEM_name, chunk_status, input_fn, output_fn, hr_status = cfg_finder()

    base_dir = os.path.join(geofloodHomeDir, output_fn, "GIS", projectName, "ChannelSegments")

    print("\nProcessing all folders in ChannelSegments...\n")

    for folder in os.listdir(base_dir):
        folder_path = os.path.join(base_dir, folder)
        if not os.path.isdir(folder_path):
            continue
```

Figure SA10: CSV discharge file location.

	A	B	C	D	E	F
1	COMID	Q_5yr	Q_10yr	Q_25yr	Q_50yr	Q_100yr
2	1	22.63	40.89	74.71	110.69	154.87
3						
4						
5						
6						
7						
8						
9						
10						

Figure SA11: CSV flow data.

Step 20 Inundation Map

The Inundation Map script was updated to operate on all files generated in the previous step, extending its functionality from processing a single input file to handling multiple extracted files. In addition, the updated script enables the extraction of flood inundation maps for multiple discharge scenarios in a single run, building on the workflow established in Step 19.

The revised script is renamed to (flood_maps_YRS). Users can specify input and output paths as shown in Figure SA12. Internal modifications to the inundation mapping code based on TauDEM are shown in Figure SA13. Extracted inundation files are automatically named according to their respective return periods, as shown in Figure SA14 ensuring organized outputs.

```
import os
import subprocess

# Paths
channel_segments_dir = r"C:\Users\user\Desktop\GeoFlood-master\GeoOutputs\GIS\my_project\ChannelSegments"
inunmap_base = r"C:\Users\user\Desktop\GeoFlood-master\GeoOutputs\Inundation\my_project"
hand_file = r"C:\Users\user\Desktop\GeoFlood-master\GeoOutputs\GIS\my_project\dem_hand.tif"

def runInunMap(hand_file, catch_file, forecast_nc, map_output):
    """
    Run InunMap to generate flood inundation map.
    """
```

Figure SA12: Inundation map paths modification.

```

import os
import subprocess

# Paths
channel_segments_dir = r"C:\Users\user\Desktop\GeoFlood-master\GeoOutputs\GIS\my_project\ChannelSegments"
inunmap_base = r"C:\Users\user\Desktop\GeoFlood-master\GeoOutputs\Inundation\my_project"
hand_file = r"C:\Users\user\Desktop\GeoFlood-master\GeoOutputs\GIS\my_project\dem_hand.tif"

def runInunMap(hand_file, catch_file, forecast_nc, map_output):
    """
    Run InunMap to generate flood inundation map.
    """
    cmd = [
        "mpiexec", "-n", "4",
        r"C:\Users\user\Desktop\GeoFlood-master\TauDEM\InunMap",
        "-hand", hand_file,
        "-catch", catch_file,
        "-forecast", forecast_nc,
        "-mapfile", map_output
    ]
    print(f"\nRunning InunMap on {forecast_nc} ...")
    subprocess.run(cmd, check=True)
    print(f"Created inundation map: {map_output}")

```

Figure SA13: TauDEM tool path identification modification.

```

        r"C:\Users\user\Desktop\GeoFlood-master\TauDEM\InunMap",
        "-hand", hand_file,
        "-catch", catch_file,
        "-forecast", forecast_nc,
        "-mapfile", map_output
    ]
    print(f"\nRunning InunMap on {forecast_nc} ...")
    subprocess.run(cmd, check=True)
    print(f"Created inundation map: {map_output}")

# Forecast years
forecast_years = [5, 10, 25, 50, 100]

for folder_name in os.listdir(channel_segments_dir):
    folder_path = os.path.join(channel_segments_dir, folder_name)
    if not os.path.isdir(folder_path):
        continue

```

Figure SA14: Inundation maps extraction for different return periods.

Step 21 Comparison between Inundation Maps

This additional script, called (comparison_1m_buffer_csv_res), is used to evaluate the performance of the GeoFlood-generated inundation maps against HEC-RAS inundation maps using four metrics: True Negatives (TN), True Positives (TP), False Negatives (FN), and False Positives (FP).

To use the script, the user first specifies the folder containing the extracted GeoFlood inundation maps, as shown in Figure SA15. A new folder named "Ras" should then be created within the same path, containing the corresponding HEC-RAS inundation maps with appropriately matching filenames, as shown in Figure SA16.

The script begins by generating a 1 m buffer around all inundation maps, as shown in Figure SA17 (the user can manually change the buffer value). For each subfolder, the script produces a CSV file summarizing the performance metrics for the corresponding return periods, as shown in Figure SA18. Additionally, a combined CSV file is generated in the parent folder, consolidating all comparison results across all subfolders for a convenient overview and analysis, as shown in Figure SA19.

```

# -----
# Base settings (adjust paths if needed)
# -----
base_dir = r"C:\Users\user\Desktop\GeoFlood-master\GeoOutputs\Inundation\my_project"
ref_dir = os.path.join(base_dir, "Ras")
splits_summary_file = os.path.join(base_dir, "splits_summary.csv")

ref_files = ["5_yrs.tif", "10_yrs.tif", "25_yrs.tif", "50_yrs.tif", "100_yrs.tif"]
comp_files = [
    "dem_NWH_inunmap_5yr.tif",
    "dem_NWH_inunmap_10yr.tif",
    "dem_NWH_inunmap_25yr.tif",
    "dem_NWH_inunmap_50yr.tif",
    "dem_NWH_inunmap_100yr.tif"
]

splits_df = pd.read_csv(splits_summary_file)

out_merged_dir = os.path.join(base_dir, "merged_per_recurrence")
os.makedirs(out_merged_dir, exist_ok=True)
out_boundaries_dir = os.path.join(base_dir, "boundaries_per_recurrence")
os.makedirs(out_boundaries_dir, exist_ok=True)

```

Figure SA15: Inundation maps location identification.



Figure SA16: HEC-RAS Inundation Maps for Different Return Periods.

```

final_poly = unary_union(polygons).buffer(0)
buffered_poly = final_poly.buffer(1)

out_shp = os.path.join(out_boundaries_dir, f"combined_boundary_{ref_file.replace('.tif','')}_buffer5.shp")
schema = {"geometry": "Polygon", "properties": {}}
if crs is None:
    with fiona.open(out_shp, "w", driver="ESRI Shapefile", schema=schema) as dst:
        dst.write({"geometry": mapping(buffered_poly), "properties": {}})
else:
    with fiona.open(out_shp, "w", driver="ESRI Shapefile", crs=from_epsg(crs.to_epsg()), schema=schema) as dst:
        dst.write({"geometry": mapping(buffered_poly), "properties": {}})

print("Saved boundary shapefile:", out_shp)

```

Figure SA17: Buffer zone definition.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
Ref_File	Comp_File	TP	TN	FP	FN	Accuracy	Precision	Recall	F1	MCC	FM	CSI	MError	Bias	ROC	RFN	RTN	REP	RTP	Number	cSplit_length
d_300	dem_NWH	211815	50921	7882	9343	0.938474	0.964123	0.957754	0.960928	0.816368	0.960933	0.924795	0.061526	0.993394	0.957754	0.042246	0.865959	0.035877	0.957754	6	296.18
d_300	dem_NWH	234824	71216	6750	8735	0.951839	0.972058	0.964136	0.968081	0.870151	0.968089	0.938136	0.048161	0.99185	0.964136	0.035864	0.913424	0.027942	0.964136	6	296.18
d_300	dem_NWH	256504	99947	12493	8949	0.943259	0.953557	0.966288	0.959988	0.863241	0.959901	0.922856	0.056741	1.013351	0.966288	0.033712	0.888892	0.046443	0.966288	6	296.18
d_300	dem_NWH	273131	121732	15725	9152	0.940732	0.945561	0.967579	0.956443	0.864335	0.956507	0.916522	0.059268	1.023285	0.967579	0.032421	0.885601	0.054439	0.967579	6	296.18
d_300	dem_NWH	289832	139573	20982	9872	0.932964	0.932493	0.967061	0.949463	0.851273	0.94962	0.903788	0.067036	1.03707	0.967061	0.032939	0.869316	0.067507	0.967061	6	296.18

Figure SA18: CSV sample corresponding to each scenario (subfolder).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
	Ref_File	Comp_File	TP	TN	FP	FN	Accuracy	Precision	Recall	F1	MCC	FM	CSI	MError	Bias	ROC	RFN	RTN	REP	RTP	Number_o	Split_len
2	d_100	dem_NWH	210629	43520	15283	10529	0.907801	0.93235	0.952392	0.942264	0.714662	0.942317	0.890831	0.092199	1.021496	0.952392	0.047608	0.740098	0.06765	0.952392	18	98.73
3	d_100	dem_NWH	233913	63373	14593	9646	0.924612	0.941277	0.960396	0.95074	0.790995	0.950788	0.906106	0.075388	1.020311	0.960396	0.039604	0.812829	0.058723	0.960396	18	98.73
4	d_100	dem_NWH	255079	91819	20621	10374	0.917979	0.925205	0.960592	0.942724	0.800275	0.942893	0.891654	0.082021	1.038602	0.960592	0.03908	0.816604	0.074795	0.960592	18	98.73
5	d_100	dem_NWH	271289	112105	25352	10994	0.913408	0.914536	0.961053	0.937218	0.800553	0.937506	0.881853	0.086592	1.050864	0.961053	0.038947	0.815564	0.085464	0.961053	18	98.73
6	d_100	dem_NWH	287862	128106	32449	11842	0.903769	0.898695	0.960488	0.928565	0.785737	0.929078	0.866655	0.096231	1.068758	0.960488	0.039512	0.797895	0.101305	0.960488	18	98.73
7	d_1000	dem_NWH	211484	52570	6233	9674	0.943181	0.971371	0.964758	0.963755	0.832928	0.963785	0.930046	0.058819	0.984441	0.956758	0.043742	0.894002	0.078629	0.956758	2	888.55
8	d_1000	dem_NWH	234012	71836	6130	9547	0.951262	0.974473	0.960802	0.967589	0.869581	0.967614	0.937214	0.048758	0.985971	0.960802	0.039198	0.921376	0.025527	0.960802	2	888.55
9	d_1000	dem_NWH	258400	100793	11647	7053	0.950515	0.95687	0.97343	0.965079	0.880593	0.965115	0.932515	0.049485	1.017306	0.97343	0.02657	0.896416	0.04313	0.97343	2	888.55
10	d_1000	dem_NWH	275219	122953	14504	7064	0.948816	0.949938	0.974975	0.962294	0.82436	0.962376	0.927328	0.051384	1.026357	0.974975	0.025025	0.894483	0.050062	0.974975	2	888.55
11	d_1000	dem_NWH	292604	141149	19406	7100	0.942411	0.937803	0.97631	0.956669	0.872506	0.956863	0.916938	0.057589	1.041061	0.97631	0.02369	0.879133	0.062197	0.97631	2	888.55
12	d_150	dem_NWH	209053	47107	11696	12105	0.914985	0.947017	0.945265	0.94614	0.74447	0.946141	0.897786	0.085015	0.998151	0.945265	0.054735	0.801099	0.052983	0.945265	12	148.09
13	d_150	dem_NWH	233859	68909	9057	9700	0.941662	0.962716	0.960174	0.961443	0.841661	0.961444	0.925749	0.058338	0.99736	0.960174	0.039826	0.883834	0.037284	0.960174	12	148.09
14	d_150	dem_NWH	254904	96456	15984	10549	0.929787	0.940994	0.96026	0.95053	0.830168	0.950578	0.905723	0.070213	1.020474	0.96026	0.03974	0.857844	0.050006	0.96026	12	148.09
15	d_150	dem_NWH	277337	114934	22523	10946	0.920263	0.923355	0.961223	0.941909	0.81665	0.942099	0.890196	0.079737	1.041012	0.961223	0.038777	0.836145	0.076645	0.961223	12	148.09
16	d_150	dem_NWH	287798	129134	31421	11906	0.905864	0.901569	0.960274	0.929996	0.790425	0.930459	0.869152	0.094136	1.065114	0.960274	0.039726	0.804298	0.099431	0.960274	12	148.09
17	d_200	dem_NWH	212652	22995	35808	8506	0.841714	0.85588	0.961539	0.905638	0.544407	0.907173	0.827549	0.158286	1.12345	0.961539	0.038461	0.391051	0.14412	0.961539	9	197.46
18	d_200	dem_NWH	235011	25603	52363	7648	0.813355	0.818357	0.968599	0.887162	0.418009	0.800314	0.797207	0.186645	1.18359	0.968599	0.031401	0.328387	0.181643	0.968599	9	197.46
19	d_200	dem_NWH	257939	29063	83377	7514	0.75948	0.755719	0.971694	0.850205	0.355893	0.856929	0.73944	0.24052	1.285787	0.971694	0.028306	0.258476	0.244281	0.971694	9	197.46
20	d_200	dem_NWH	274497	35413	102044	7786	0.738338	0.728996	0.972418	0.833294	0.355304	0.841955	0.714228	0.261662	1.333913	0.972418	0.027582	0.25763	0.271004	0.972418	9	197.46
21	d_200	dem_NWH	291746	43787	116768	7958	0.729009	0.714164	0.973447	0.823888	0.371412	0.833787	0.700518	0.270991	1.363058	0.973447	0.026553	0.272723	0.285836	0.973447	9	197.46
22	d_2000	dem_NWH	209793	54095	4708	11365	0.942588	0.978051	0.948611	0.963106	0.835876	0.963219	0.928838	0.057412	0.960989	0.948611	0.051389	0.919936	0.021946	0.948611	1	1777.1
23	d_2000	dem_NWH	232104	73645	4321	11455	0.950934	0.981724	0.952968	0.967132	0.871967	0.967239	0.936356	0.049066	0.970709	0.952968	0.047032	0.944578	0.018276	0.952968	1	1777.1
24	d_2000	dem_NWH	257541	104275	8165	7912	0.957456	0.969271	0.970194	0.969732	0.898151	0.969732	0.941243	0.042544	1.000953	0.970194	0.029806	0.927383	0.030729	0.970194	1	1777.1
25	d_2000	dem_NWH	274397	126809	10648	7886	0.955844	0.962644	0.972063	0.967331	0.899336	0.967343	0.936729	0.044156	1.009785	0.972063	0.027937	0.922536	0.037356	0.972063	1	1777.1
26	d_2000	dem_NWH	291815	146081	13574	7889	0.953368	0.955552	0.973677	0.964529	0.896839	0.964572	0.931489	0.046632	1.018969	0.973677	0.026323	0.915456	0.044448	0.973677	1	1777.1
27	d_250	dem_NWH	208362	53813	4990	12796	0.93647	0.976611	0.942141	0.959067	0.820806	0.959221	0.921352	0.06353	0.964704	0.942141	0.057859	0.91514	0.023389	0.942141	8	222.14
28	d_250	dem_NWH	229668	72773	5193	13891	0.940645	0.977889	0.942967	0.960111	0.846472	0.960269	0.923281	0.059355	0.964288	0.942967	0.057033	0.933394	0.022111	0.942967	8	222.14
29	d_250	dem_NWH	254143	103742	8698	11310	0.947054	0.966908	0.957394	0.962127	0.874302	0.962139	0.927018	0.052946	0.99016	0.957394	0.042606	0.922643	0.033092	0.957394	8	222.14

Figure SA19: Consolidated CSV file.

Remarks

The scripts accompanying this study are provided as supplementary material and will also be made available on GitHub, along with the same detailed documentation as in this document. This will ensure accessibility for the research community, facilitate reproducibility, and encourage further development and refinement of the adopted methodology. By enabling broader use and adaptation, these resources support the creation of more robust and resilient models that contribute to sustainable development and improved decision-making.

Appendix A. REFERENCES

- Zheng, X.; Maidment, D. R.; Tarboton, D. G.; Liu, Y. Y.; Passalacqua, P. GeoFlood: Large-Scale Flood Inundation Mapping Based on High-Resolution Terrain Analysis. *Water Resour. Res.* **2018**, *54* (12), 10,013–10,033. <https://doi.org/10.1029/2018WR023457>.
- Fearnhead, P.; Maidstone, R.; Letchford, A. Detecting Changes in Slope with an L0 Penalty. *J. Comput. Graph. Stat.* **2019**, *28* (2), 265–275. <https://doi.org/10.1080/10618600.2018.1512868>.