Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

IKG
Institute of Cartography
and Geoinformation

Leveraging Diffusion Models for Urban Change Detection and Classification from Historical Maps

Master's Thesis
Autumn Semester 2024

Chunyang Gao
gaochu@ethz.ch

Supervisors:
Sidi Wu, Dr. Yizi Chen
Prof. Dr. Lorenz Hurni

March 9, 2025

Acknowledgements

I sincerely express my deepest gratitude to my advisors, Sidi Wu and Dr. Yizi Chen, for their invaluable guidance, continuous support, and insightful advice throughout my master's thesis. Additionally, I am profoundly grateful to Prof. Dr. Lorenz Hurni for granting me the opportunity to conduct this research at the Chair of Cartography, Institute of Cartography and Geoinformation at ETH Zurich. His support has provided me with an inspiring environment to explore and develop the project.

Abstract

Urban change detection is crucial for understanding historical urban transformations and informing future planning decisions. This thesis explores a novel approach that manipulates vector data to simulate urban changes and employs ControlNet-guided Stable Diffusion to generate synthetic Old National-style map tiles. A systematic classification of urban changes based on geospatial features—including buildings, roads, railways, rivers, lakes, streams, and forests—is proposed and implemented through vector data manipulation. The manipulated vector data served as conditioning inputs to generate pre- and post-change map tiles, accompanied by change masks for change detection. Change detection models, including ChangeFormer and MambaBCD, are trained on the generated dataset and tested on real historical maps. The results demonstrate that these models effectively detect building and forest changes, with ChangeFormer performing well in overall change detection and MambaBCD achieving strong performance in building change detection. The findings highlight the potential of leveraging diffusion models for historical map analysis and change detection. By simulating changes in vector data and generating synthetic maps, this approach provides a scalable solution for training change detection models in scenarios where annotated historical data is scarce. Future research could focus on refining the classification of urban change types, extending change detection beyond binary classification to include semantic change analysis, and developing conflict detection rules to improve the realism of vector manipulation.

Contents

List of Figures

Chapter 1

Introduction

This chapter introduces the research by first outlining the motivation and the current direction in urban change detection. It then presents the challenges associated with analyzing historical maps and defines the research problem. The objectives of the thesis are then listed, followed by a brief overview of the content covered in this report.

1.1 Motivation

Our current era, often referred to as "The Century of the City" (Johnson and Peirce, 2008; Reba and Seto, 2020), is characterized by rapid urbanization in the 21st century, following significant population growth and urban transformations in recent centuries. These dynamics have drawn the attention of researchers, policymakers, and practitioners from various fields, all of whom are keen to understand how urban areas are changing—whether through growth, shrinkage, densification, or restructuring (Reba and Seto, 2020). This understanding is essential for urban planners, policymakers, and stakeholders who aim to create sustainable and livable cities.

Recently, diffusion models have gained prominence as a powerful tool in generative AI, capable of generating high-quality images based on various prompts, such as text descriptions, edge maps, and semantic inputs. These models provide significant potential for applications in historical map analysis by enabling the creation of synthetic maps that replicate historical styles. Using manipulated vector representations as segmentation map, diffusion models can effectively generate pre- and post-change historical maps in specified styles according to the vector maps, such as road construction, building expansions, and land use transformations.

This project aims to simulate a wide range of urban change scenarios, leveraging diffusion models to generate synthetic data that can bootstrap urban change detection and classification tasks on historical maps.

1.2 Problem statement

To comprehensively address the complexities of contemporary urbanization and its long-term impacts, advancements in technology have played a pivotal role in reshaping how we study and monitor urban change. Remote sensing, in particular, has revolutionized the way we characterize and track transformations in urban land. Data and information about changes in urban land structure, form, and extent—once accessible only through labor-intensive ground surveys and traditional mapping techniques—are now routinely collected and produced through a constellation of government and commercial airborne and satellite sensors (Reba and Seto, 2020). By providing frequent, large-scale, and detailed observations, remote sensing imagery has become a crucial data source for many studies aiming to uncover and analyze urban change. Researchers have widely used remote sensing imagery to monitor changes within the time frame of available satellite records, offering valuable insights into contemporary urban dynamics.

However, remote sensing has only seen widespread application in recent decades or centuries, leaving significant gaps when investigating longer historical periods. Historical maps, in contrast, serve as valuable complementary data for studies that require information about past urban conditions and transformations. These maps offer an exceptional window into understanding human activities and city development over time, often capturing the spatial configuration and evolution of urban areas before modern satellite technologies existed. Many historical maps are publicly accessible, providing an essential resource for long-term studies of urban change and enabling researchers to analyze urban dynamics spanning several centuries (Z. Li et al., 2021). Through the analysis of historical maps, it is possible to reveal urban changes, offering insights into the patterns, processes, and impacts of urban growth from the distant past. However, when performing change detection using historical maps, the absence of corresponding change masks poses a significant challenge for training models to accurately detect changes. Unlike modern remote sensing datasets, which often include well-annotated change masks for supervised learning, historical maps typically lack such labeled data. This limitation hinders the development of reliable models for identifying and quantifying urban transformations over time. To overcome this obstacle, a comprehensive dataset containing pairs of pre- and post-change historical maps along with their corresponding change masks is essential.

An ideal solution is to simulate urban changes and save change masks at the same time. But directly copying and pasting a feature from one image onto another can lead to inconsistencies between the foreground and background, resulting in noticeable artifacts or mismatches. These inconsistencies may arise due to differences in color tones, spatial alignment, or texture variations between the two images. As a result, such operations can introduce unwanted noise, potentially affecting the model performance. Ensuring seamless integration of features while maintaining contextual coherence is therefore crucial for generating high-

quality simulated changes.

On the other hand, in the field of urban planning, studies on urban change primarily focus on macro-level transformations within urban areas, distinguishing urban from non-urban regions. Urban expansion is typically categorized into three main types: infilling (development within existing urban boundaries), edge-expansion (growth at the periphery), and outlying (isolated developments beyond the urban fringe), as shown in Figure 1.1 (C. Sun et al., 2013). In contrast, change detection emphasizes micro-level transformations of geospatial objects (Tian et al., 2022), capturing fine-grained modifications in individual features such as buildings, roads, and lakes. When performing change detection on historical maps, a comprehensive definition of geospatial object change categories is essential to simulate different types of urban transformations. This classification serves as a foundation for robust change detection models, ensuring a structured and interpretable analysis of historical urban evolution.

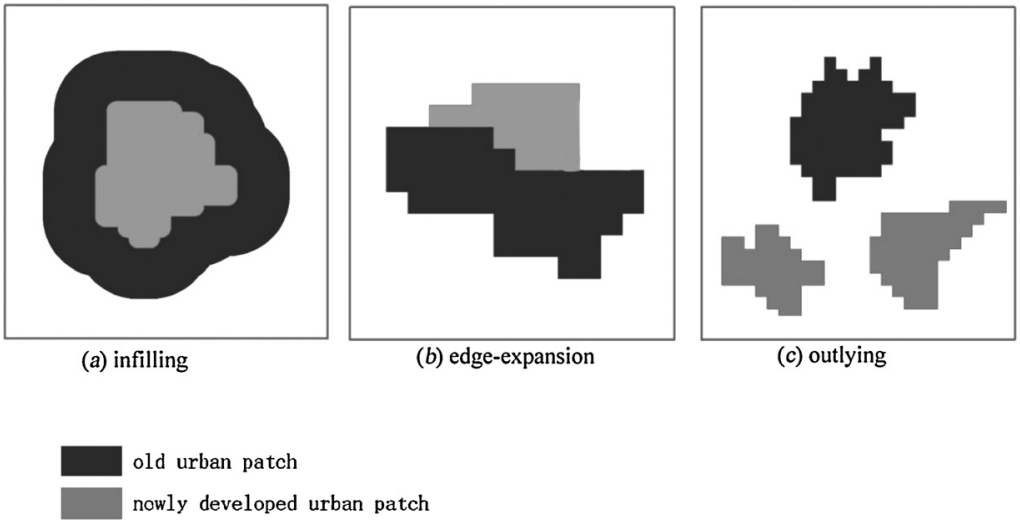


Figure 1.1: Three types of urban growth

1.3 Objectives

Recognizing the absence of a comprehensive definition for geospatial object changes in historical maps, the thesis explores different types of changes to guide vector-based manipulation.

Furthermore, Claudio Affolter from the Institute of Cartography and Geoinformation of ETH Zurich has explored how stable diffusion can be controlled using text prompts in combination with vector data. He trained a specialized Stable Diffusion model for map tile generation where users can select a specific map style and provide their own vector data to control the layout (Affolter, 2024).

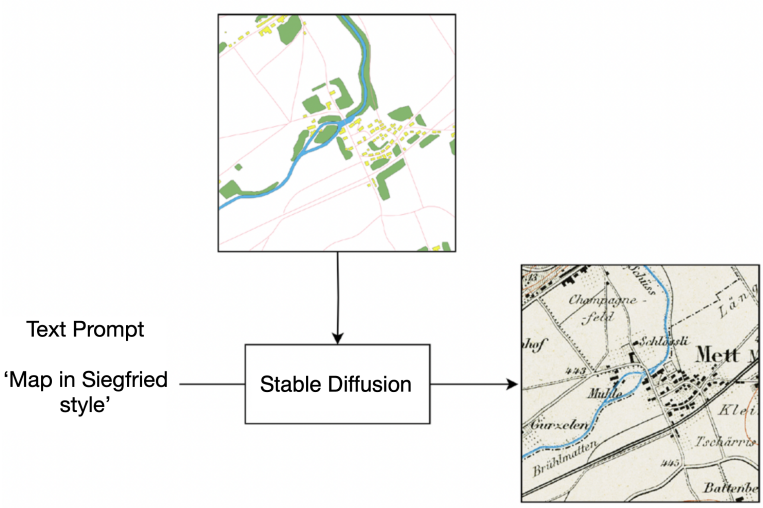


Figure 1.2: Overall idea: A text prompt specifying the map style, here Siegfried style, and vector data defining the spatial composition and semantic layout are provided as input to Stable Diffusion.

As shown in Figure 1.2, the style of the map is specified as Siegfried style, which corresponds to the style used for the Topographic Atlas of Switzerland produced between 1870 and 1926. Given the vector data that defines the spatial composition and semantic layout of features located in the map tile, the model can generate a map tile in Siegfried style automatically.

The diffusion models he trained can be employed to generate plausible maps given the pre- and post-change vector input. Combined with the change masks saved in the manipulation phase, the thesis trains the change detection model for prediction in real historical maps.

1.4 Content overview

This thesis first introduces the relevant theory and related work. It then presents the methodology, covering vector-based change simulation, synthetic map generation, and change detection model training workflows. Key components are explained in depth, including the associated implementation. Finally, the results are analyzed, and the implications, limitations, and potential future improvements are discussed.

Chapter 2

Theory and Related Work

This chapter provides an overview of the current research on urban change classification and simulation. Subsequently, methods for map generation are introduced. In the final section, urban change detection techniques are presented.

2.1 Urban change type

In the field of urban planning, classifying urban growth types often involves the distinction between urban land and non-urban land (Wilson et al., 2003). It is important to quantify and categorize urban growth in a way that is useful and meaningful to land use decision-makers at the municipal, regional and state levels (Wilson et al., 2003). Infilling, expansion, and outlying growth are the three main types of urban growth, each representing a distinct pattern of urban development. Edge-expansion means the newly developed urban area spreading out from the fringe of existing urban patches. Outlying growth is characterized by a change from non-urban to urban land occurring beyond existing urban areas. Expansion refers to the newly developed urban area spreading out from the fringe of existing urban patches.

Many studies have proposed quantitative methods to identify urban growth types. Wilson et al. (2003) develop a geospatial model to quantify, describe and map urban growth. They use two dates of satellite-derived land cover data to create a change map according to the land cover type change. The change map is then used to identify five classes of growth: infill, expansion, isolated, linear branch, and clustered branch. Similarly, C. Sun et al. (2013) introduce an integrated approach of remote sensing and GIS to identify three urban growth types: infilling growth, outlying growth and edge-expansion growth. It converts land cover maps to urban area and non-urban area first and divides the urban patches in each period into newly developed urban patches and old urban patches. Urban growth types are defined according to the ratio between the length of common edge and patch perimeter of patches (Xu et al., 2007), as shown in Figure 1.1. J. Chen et al. (2016) integrates the convex hull analysis and common

edge analysis at double scales, and develop a comprehensive matrix analysis to distinguish the different types of urban land expansion.

Visual interpretation is also used as a method to identify urban areas. Camagni et al. (2002) defines different typologies of urban expansion—namely, infilling, extension, linear development, sprawl, and large-scale projects to explore whether different patterns of urban expansion could be associated with specific environmental costs. The urban expansion of each commune is identified using land consumption maps. M. Li et al. (2016) systematically explored urban sprawl in Chinese coastal cities with a visual interpretation method to identify urban built-up areas from 1979 to 2013.

Apart from the above methods, D. Wang et al. (2022) use vector building data and road intersection data for comparative validation based on urban expansion curve method to identify the physical urban area using the meso-city scale.

Additionally, without classifying urban growth types, Y. Sun et al. (2024) derive 2-D morphological patterns from town ground plans in 1918, 1969, 2000, and 2021 to represent urban landscape fractions (buildings, lands, and waters) using historical maps and remote sensing images.

In the other classification scheme of urban type, emphasis is put on urban morphology. Fang et al. (2024) develop UrbanClassifier, a deep learning-based model for automated urban fabric typology and temporal analysis by considering features from multiple spatial scales and view points. Six distinct urban fabric typologies are identified and labelled: linear development, open block, gated compound, medieval region, irregular grid and orthogonal grid. Figure 2.1 illustrates the six generic urban fabric typologies. Vanderhaegen and Canters (2017) focus on the potential of urban metrics describing the presence and the configuration of built-up and open space areas for mapping distinct types of urban form and function at city block level. They identify six land-use/urban from classes: Commercial/Industrial/Services, Mixed, Continuous residential without frontal setback, Continuous residential with frontal setback, Semi-detached residential and Detached residential. Moosavi (2022) trains a deep convolutional auto-encoder, that automatically learns the hierarchical structures of urban forms and represents them via dense and comparable vectors.

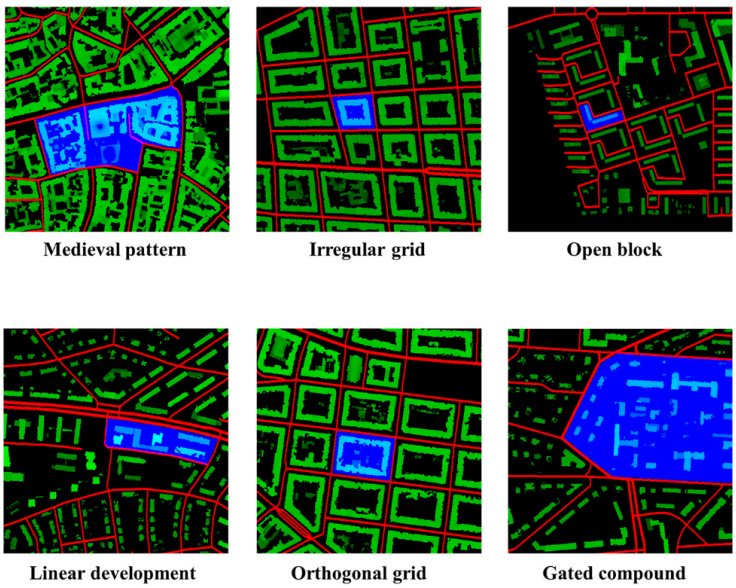


Figure 2.1: Six urban fabric typologies

Given the above studies, these methods are not suitable for defining urban change in historical maps. Firstly, the three types of urban growth—infilling, expansion, and outlying growth—all involve the classification of urban and non-urban areas in land use maps. The seamless land cover representation provided by remote sensing imagery is well-suited for generating land use maps. However, the situation is entirely different for historical maps.

1. Historical maps discretely depict various geospatial objects, such as lakes, roads, and buildings, while the remaining areas serve as background. This makes it difficult to distinguish urban and non-urban areas.

2. Classifying urban morphology requires additional information, such as the delineation of land parcels.

3. By visually comparing maps from different periods, it is observed that urban change tends to manifest as changes in individual geospatial objects rather than patterns that significantly alter the overall layout.

2.2 Urban change simulation

To better understand and simulate urban change, researchers have proposed various models and methods to capture urban expansion patterns and update geographic information systems (GIS). X. Li and Gong (2016) summarize three urban growth models, including land use/transportation models (LUTs), cellular

automata (CA), and agent-based models (ABMs). Rui and Ban (2011) develop a dynamic urban growth model that integrates road network expansion with land use changes, using space syntax metrics to simulate traffic flow and guide development. Rui et al. (2013) propose a growing model for self-organized urban street networks. The model involves a competition among new centers with different values of attraction radius and a local optimal principle of both geometrical and topological factors. Rui and Ban (2010) employ a multi-agent system (MAS) to simulate urban sprawl in the Greater Toronto Area (GTA) from 1985 to 2005, defining three types of agents—new and existing residents, developers, and the government—to model their interactions in land development.

Given the problem that simulating real-world urban land-use change using cellular automata (CA) mpdels can be challenging due to high degree of subjectivity involved in CA model parameterisation. J. Wang et al. (2022) used the U-Net deep learning algorithm to capture historical urban development and simulate future patterns. It successfully captured neighborhood effects whereby new urban areas are more likely to arise near existing urban areas.

Updating geographic information systems (GIS) also requires urban change information (H. Chen et al., 2024). Doucette et al. (2009) present a methodology for updating vector data using automated feature extraction and registration techniques, ensuring spatial accuracy with newly acquired imagery. Ceresola et al. (2005) propose a semi-automatic approach for updating GIS vector maps by detecting changes between historical and current aerial images. Their method integrates edge detection, color segmentation, and adaptive edge linking to identify and extract new building footprints, effectively reducing manual effort in urban map updating.

2.3 Map generation

A variety of approaches have been explored for generating map representations. Shen et al. (2020) propose to apply Generative Adversarial Network (GAN) in creating urban design plans, helping designers automatically generate the predicted details of buildings configuration with a given condition of cities. Highly related to the project, Z. Li et al. (2021) use a style transfer model to convert contemporary map images into historical style and place text labels upon them to automatically generate an unlimited amount of annotated historical map images for training text detection models. The model is shown in Figure 2.2.

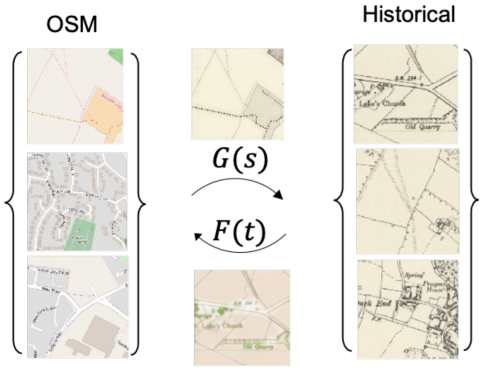


Figure 2.2: Illustration of CycleGAN. Generator G learns to convert OSM images to historical style and another generator F learns to convert the historical map images to the OSM style.

Yang et al. (n.d.) propose a conditional diffusion model-based approach for the generation of historical urban fabric, enabling the controlled synthesis of urban patterns by adjusting multiple parameters such as street layout and building density. This data-driven method effectively simulates and reconstructs historical urban evolution.

This thesis is based on Claudio Affolter's work (Affolter, 2024), "From Vector Features to Stylized Maps – Exploration of Stable Diffusion Applied to Maps," investigates how to leverage image diffusion models to generate map tiles with specific styles. Although models like Stable Diffusion excel at generating high-quality images, they face significant limitations when generating complex layouts, such as maps, particularly in controlling spatial composition and semantic layout. He proposes a method that combines text prompts with vector data, using the ControlNet architecture to control the Stable Diffusion model and generate map tiles with predefined styles.

ControlNet is a neural network architecture to add spatial conditioning controls to large, pretrained text-to-image diffusion models. It locks the production-ready large difussion models, and reuses their deep and robust encoding layers pretrained with billions of images as a strong backbone to learn a diverse set of conditional controls (Zhang et al., 2023).

He selects three map styles—Swisstopo (Federal Office of Topography swisstopo, 2024b), Siegfried (Federal Office of Topography swisstopo, 2024c), and Old National (Federal Office of Topography swisstopo, 2024a) and collects corresponding raster maps and vector data. Due to the incompleteness of historical vector data (Siegfried and Old National map), modern vector data was used as a substitute. Affolter then created a triple-structured dataset for each map style, consisting of raster map tiles, vector map tiles, and text prompts to train the ControlNet. He trained three specialized models (one for each style) and a combined

model (supporting all three styles). Model performance is evaluated using both quantitative and qualitative methods, including metrics such as Mean Squared Error (MSE) and Mean Intersection over Union (MIoU). The experimental results show that the Swisstopo-style model performs the best, generating map tiles that closely matched the target style with minimal post-processing. In contrast, the Old National and Siegfried-style models face challenges, particularly in generating background regions and textures. Through post-processing and augmentation techniques (e.g., generating multiple versions of tiles and selecting the best results), the output quality of these models are improved. The combined model shows mixed performance when generating multiple map styles, with some textures (e.g., rivers and forests) deteriorating in quality, while background regions improved. The results are illustrated in Figure 2.3, 2.4 and2.5.

Figure 2.3: Performance of the Swisstopo style model. Left: Control image. Center: Output map in Swisstopo style. Right: Output map in Swisstopo style after post-processing.

Figure 2.4: Performance of the Old National style model. Left: Control image. Center: Output map in Old National style. Right: Output map in Old National style after post-processing.

Figure 2.5: Performance of the Siegfried style model. Left: Control image. Center: Output map in Siegfried style. Right: Augmented output map in Siegfried style.

2.4 Urban change detection

With the rapid expansion of urban areas, urban change detection (UCD) has become a crucial and effective method for capturing changes in geospatial objects, facilitating dynamic urban analysis (Tian et al., 2022). Current change detection methods can be categorized into binary change detection and semantic change detection. Binary change detection is used to specify where the change occurs, and to determine whether the object changes. While semantic change detection also involves the change directions, to provide "from-to" information.

Many studies have utilized historical maps for change detection. Liu et al. (2024) introduce MapChange—a novel semantic change detection framework that integrates temporally invariant historical maps with a triplet network architecture, effectively mitigating temporal inconsistencies in urban change detection and demonstrating superior performance on the Hi-UCD and HRSCD datasets. Deng et al. (2024) propose CMPANet, a deep learning model for cross-modal change detection (CMCD) between historical land use maps and current remote sensing images. The approach leverages Vision Transformer (ViT) for feature encoding and incorporates an Image-Map Alignment Module (IMAM) and a Cross-Modal Co-Interaction Module (CCMAT) to enhance cross-modal feature alignment and fusion, achieving state-of-the-art performance on the EVLab-CMCD and HRSCD datasets.

To address the lack of data for semantic change detection, Tian et al. (2022) introduce Hi-UCD, a large-scale ultra-high-resolution remote sensing dataset designed for urban semantic change detection. The study benchmarks various deep learning methods, highlighting the importance of unchanged samples in change detection and demonstrating that HRNet excels in high-resolution feature extraction. Additionally, it proposes a new semantic consistency evaluation metric to enhance accuracy in semantic change detection.

With regard to vector change, Armenakis et al. (2002) propose an interactive and semi-automated change detection framework for topographic database updating, integrating existing vector data with satellite imagery. The approach leverages non-intersection analysis and buffer correction to identify spatial changes while reducing false detections, enhancing the efficiency of large-scale geospatial data revision.

In the field of change detection, Convolutional Neural Networks (CNNs) and Transformers are widely used. However, the recently proposed ChangeMamba (2.6), which leverages the Mamba architecture based on state space models (SSMs), outperforms current CNN- and Transformer-based approaches (H. Chen et al., 2024). Depending on the size and depth of the encoder network, ChangeMamba architectures are available in Tiny, Small, and Base versions.

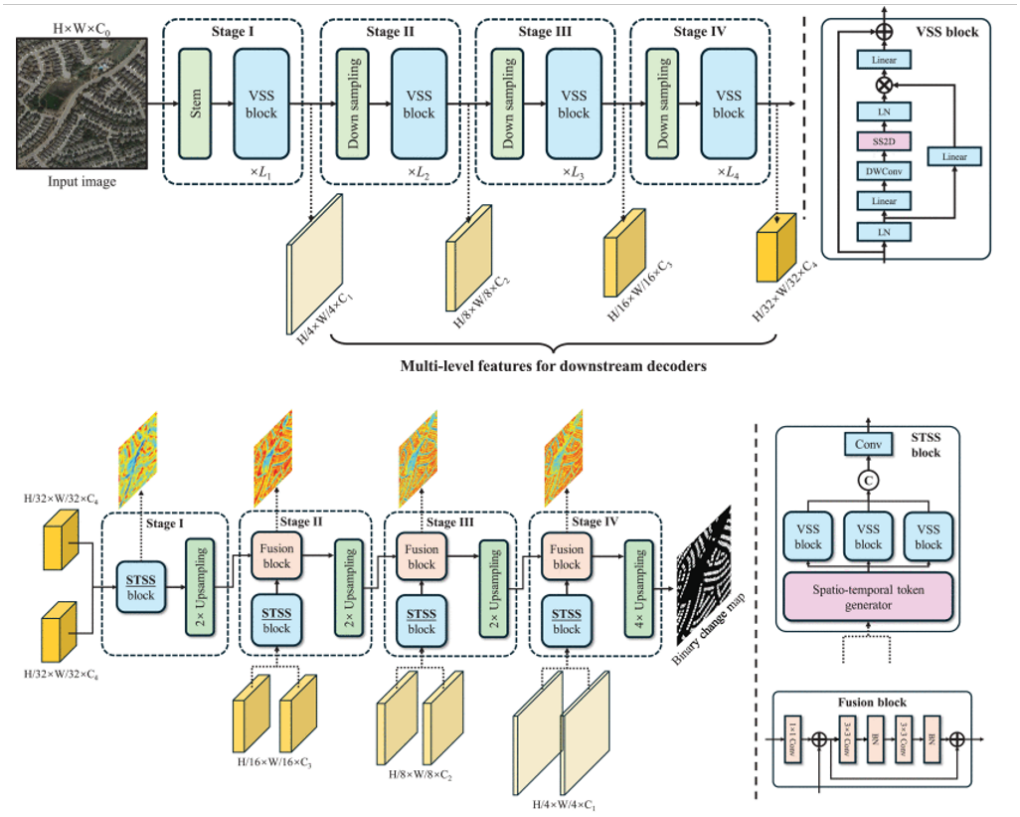


Figure 2.6: Change encoder and decoder in ChangeMamba

In this project, ChangeFormer (Figure 2.7) and ChangeMamba (Tiny version due to computational resource limitations, and MambaBCD architecture for binary change detection) are employed for binary change detection in historical maps. ChangeFormer unifies hierarchically structured transformer encoder with Multi-Layer Perception (MLP) decoder in a Siamese network architecture to ef-

ficiently render multi-scale long-range details required for accurate CD (Bandara and Patel, 2022).

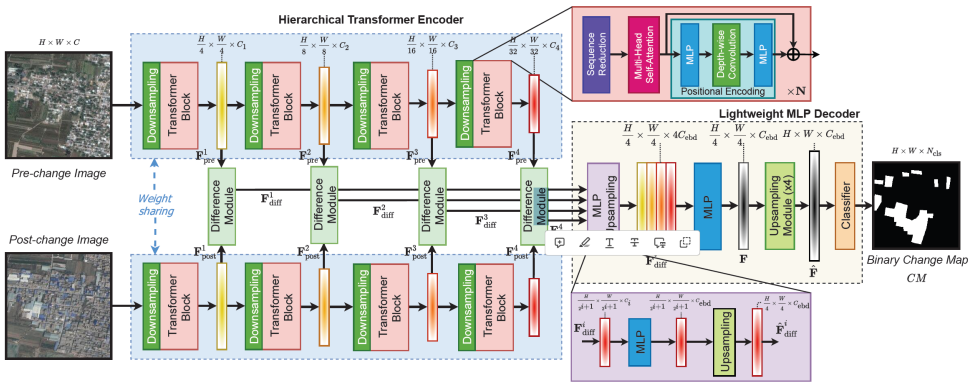


Figure 2.7: ChangeFormer architecture for change detection

Chapter 3

Methodology

This chapter outlines the methodology employed in this study. First, the workflow for simulating urban changes and generating synthetic historical maps is introduced. Then, the process of leveraging the diffusion model and conducting change detection is described. Finally, the evaluation method is presented.

3.1 Urban change type definition

Through visual interpretation and supporting evidence from previous studies, Table 3.1 presents the features manipulated in this study and their corresponding change categories, while Section 3.2 provides the implementation details of the manipulation process.

Urban change types are categorized based on key geospatial features in Old National maps. The classification focuses on modifications in buildings, transportation infrastructure (roads and railways), water bodies (rivers, lakes, and streams), and natural features like forests. These changes are further divided into addition, deletion, expansion, shrinkage, and morphological alterations, reflecting different urban transformation patterns over time.

Buildings undergo three primary changes: addition (new buildings constructed), deletion (removal of existing buildings), and morphology changes (modifications in shape or structure). Roads and railways experience expansion (addition of new segments, extension in length, or increase in width) and shrinkage (removal, reduction in length, or narrowing). Similarly, rivers, lakes, and streams exhibit expansion (new water bodies formed or existing ones growing in size) and shrinkage (water bodies disappearing or reducing in area). Forested regions also experience expansion and shrinkage, representing deforestation or afforestation processes.

By systematically defining these urban change types, this study establishes a structured framework for manipulating historical map vector data. This classification ensures that changes are represented in a detailed and meaningful manner,

forming the foundation for generating synthetic map pairs. These pairs, accompanied by corresponding change masks, provide essential training data for the change detection model, enabling more accurate identification and classification of urban transformations.

Feature class	Change	Sub-change
Building	Addition Deletion Morphology	
Road, railway	Expansion	Addition Longer Wider
	Shrinkage	Deletion Shorter Narrower
River, lake	Expansion	Addition Larger
	Shrinkage	Deletion Smaller
Stream	Expansion	Addition Longer
	Shrinkage	Deletion Shorter
Forest	Expansion Shrinkage	

Table 3.1: Summary of manipulated features and their corresponding change types.

3.2 Urban change manipulation

3.2.1 Building manipulation

Building layer manipulation involves three key operations: random translation of buildings, deletion of buildings, and generation of a change mask. Since no conflict detection is performed, overlapping buildings can be considered as morphological changes. The following code snippets belonging to `building_change.py` show how building layer is manipulated. Key components of the implementation are explained below.

```
import geopandas as gpd
import random
import pandas as pd
from shapely.affinity import translate
```

```
5
6  def apply_random_affine_transform(geometry):
7      """Apply a translation transformation to buildings"""
8
9      translation_x = random.uniform(-500, 500)
10     translation_y = random.uniform(-500, 500)
11
12     transformed_geometry = translate(geometry, xoff=translation_x,
       yoff=translation_y)
13
14     return transformed_geometry
15
16 def process_building_layer(map_path, output_path, mask_path):
17     """Apply translation transformations to buildings and save the
       mask"""
18     gdf = gpd.read_file(map_path, layer="T48_DKM25_GEBAEUDE")
19     random.seed(42)
20
21     # Randomly select 10% of buildings for transformation
22     selected_buildings = gdf.sample(frac=0.1)
23     transformed_buildings = selected_buildings.copy()
24     transformed_buildings['geometry'] = transformed_buildings['
       geometry'].apply(apply_random_affine_transform)
25
26     # Randomly delete 5% of buildings and add them to the mask
27     fraction_to_delete = 0.05
28     buildings_to_delete = gdf.sample(frac=fraction_to_delete)
29
30     gdf = gdf.drop(buildings_to_delete.index)
31     gdf = pd.concat([gdf, transformed_buildings], ignore_index=True
       )
32
33     # Generate a mask containing only the changed areas
34     original_gdf = gpd.read_file(map_path, layer="
       T48_DKM25_GEBAEUDE")
35     mask = gpd.overlay(original_gdf, gdf, how='symmetric_difference
       ')
36
37     # Save the modified building layer and the mask layer
38     gdf.to_file(output_path, layer="T48_DKM25_GEBAEUDE", driver="
       GPKG")
39     mask.to_file(mask_path, layer="T48_DKM25_GEBAEUDE_mask", driver
       ="GPKG")
40
41     print(f"Modified building data saved to {output_path}")
42     print(f"Mask of all changes saved to {mask_path}")
```

The function `process_building_layer()` reads the building layer from a GeoPackage file using `geopandas.read_file()`, specifically selecting the layer `"T48_DKM25_GEBAEUDE"`, which contains building geometries.

To simulate urban modifications such as building relocations, addition or shifts in historical maps, 10% of the buildings are randomly selected.

These buildings undergo an affine translation transformation, where they are shifted by a random distance between -500 and 500 units in both the x and y directions. This transformation is implemented in the function `apply_random_affine_transform()`, using `shapely.affinity.translate()`. Another aspect of urban change is the removal of buildings due to factors such as urban redevelopment or demolition. To simulate this, 5% of the buildings are randomly selected and removed from the dataset.

A change mask is created to capture all modifications (both deletions and translations). This is done using `geopandas.overlay()` with the `symmetric_difference` method, which identifies the differences between the original dataset and the modified dataset. The result is a new geospatial layer containing only the areas where changes have occurred.

The modified dataset, which incorporates the translated and remaining buildings while excluding the deleted ones, is saved as a new GeoPackage file (`output_path`). Additionally, the change mask, which records all building modifications, is stored separately (`mask_path`).

3.2.2 Road, railway manipualtion

Road layer manipulation involves three key operations: expansion of roads, deletion of roads, and generation of a change mask. The following code snippets belonging to `road_change.py` show how road layer is manipulated. During road expansion, roads extend outward from their nodes, allowing the simulation of both road addition and elongation. The same principle applies to road deletion and shortening. However, changes in road width, such as widening or narrowing, introduce additional complexity and are therefore not simulated. Key components of the implementation are explained below.

```
import geopandas as gpd
import random
import numpy as np
import pandas as pd
from shapely.geometry import LineString, MultiLineString,
    GeometryCollection
import networkx as nx
from shapely.ops import split

def process_road_layer(map_path, output_path, mask_path):
    """Expand the road network and save the mask"""
    gdf = gpd.read_file(map_path, layer="T45_DKM25_STRASSE")

    # Step 1: Randomly delete 1% of the roads
    gdf, deleted_roads = random_delete_roads(gdf)
    print(f"Deleted {len(deleted_roads)} road segments.")

    # Step 2: Build graph and proceed with road expansion
    G = build_graph(gdf)
```

```
19
20      sindex = gdf.sindex
21
22      # Step 3: Expand road network
23      updated_gdf, new_features = expand_road_network(G, gdf, sindex)
24
25      # Step 4: Combine deleted and new roads into the mask
26      mask = pd.concat([deleted_roads, new_features], ignore_index=
        True)
27
28      # Step 5: Save updated GeoPackage and mask
29      updated_gdf.to_file(output_path, layer="T45_DKM25_STRASSE",
        driver="GPKG")
30      mask.to_file(mask_path, layer="T45_DKM25_STRASSE_mask", driver=
        "GPKG")
31
32      print(f"New road segments saved to {output_path}")
33      print(f"Mask of deleted and new roads saved to {mask_path}")
```

The function `process_road_layer()` reads the road network layer from a GeoPackage file using `geopandas.read_file()`, specifically selecting the layer `"T45_DKM25_STRASSE"`, which contains road geometries.

```
1  def random_delete_roads(gdf, delete_percentage=0.01):
2      """Randomly delete roads and save as GeoDataFrame"""
3      num_to_delete = int(len(gdf) * delete_percentage)
4      deleted_roads = gdf.sample(n=num_to_delete, random_state=42)
5      remaining_roads = gdf.drop(deleted_roads.index)
6      return remaining_roads.reset_index(drop=True), deleted_roads.
       reset_index(drop=True)
```

To simulate road removal, 1% of the existing road segments are randomly selected and removed from the dataset. This step reflects real-world scenarios where roads are decommissioned or repurposed.

```
1  def build_graph(gdf):
2      """Build a road network graph"""
3      G = nx.Graph()
4      for _, row in gdf.iterrows():
5          geometry = row['geometry']
6          if isinstance(geometry, (LineString, MultiLineString)):
7              for line in geometry.geoms if isinstance(geometry,
       MultiLineString) else [geometry]:
8                  start_point = (line.coords[0][0], line.coords
       [0][1])
9                  end_point = (line.coords[-1][0], line.coords
       [-1][1])
10                 G.add_edge(start_point, end_point, weight=line.
       length)
11     return G
```

After deletion, the remaining roads are used to construct a road network graph using `networkx`. Each road segment is represented as an edge, and inter-

sections are represented as nodes. This graph structure allows for controlled road expansion in the next step.

```
def generate_curved_road(start_point, min_length=100, max_length=500, segments=10, max_angle_change=15):
    """Generate a curved road"""
    length = random.uniform(min_length, max_length)
    segment_length = length / segments
    angle = np.random.uniform(0, 360)
    points = [start_point]

    for _ in range(segments):
        angle += np.random.uniform(-max_angle_change, max_angle_change)
        new_x = points[-1][0] + segment_length * np.cos(np.radians(angle))
        new_y = points[-1][1] + segment_length * np.sin(np.radians(angle))
        points.append((new_x, new_y))

    return LineString(points)

def expand_road_network(G, gdf, sindex, iterations=300):
    """Expand the road network and check for conflicts"""
    new_roads = []
    objektart_options = ['10m_Strasse', '3m_Strasse', '4m_Strasse', '6m_Strasse', '8m_Strasse', '2m_Weg','1m_Weg','Ausfahrt','Autobahn','Autostrasse','Einfahrt']

    for _ in range(iterations):
        start_point = random.choice(list(G.nodes))
        new_road = generate_curved_road(start_point)
        possible_matches = gdf.iloc[list(sindex.intersection(new_road.bounds))]

        for _, row in possible_matches.iterrows():
            if new_road.intersects(row['geometry']):
                intersection_point = new_road.intersection(row['geometry'])
                split_roads = split(new_road, intersection_point) if not intersection_point.is_empty else [new_road]

                if isinstance(split_roads, (MultiLineString, GeometryCollection)):
                    for geom in split_roads.geoms:
                        if isinstance(geom, LineString):
                            G.add_edge((geom.coords[0][0], geom.coords[0][1]), (geom.coords[-1][0], geom.coords[-1][1]), weight=geom.length)
                            new_roads.append(geom)
                elif isinstance(split_roads, LineString):
                    G.add_edge((split_roads.coords[0][0], split_roads.coords[0][1]), (split_roads.coords[-1][0], split_roads.coords[-1][1]), weight=split_roads.length)
```

```
                    new_roads.append(split_roads)
                break

    new_roads_gdf = gpd.GeoDataFrame({'OBJEKTART': [random.choice(
    objektart_options) for _ in new_roads], 'geometry': new_roads},
     crs=gdf.crs)
    return pd.concat([gdf, new_roads_gdf], ignore_index=True),
    new_roads_gdf
```

To simulate road network expansion, new roads are generated from randomly selected starting points within the existing network. The function `generate_curved_road()` creates curved roads by iteratively adjusting their direction within a defined angle range, ensuring a more natural road development pattern. A spatial index is used to check for conflicts with existing roads, ensuring that newly generated roads integrate seamlessly. If intersections occur, the new roads are split at these points. Each new segment is assigned a road type from predefined categories to reflect real-world road attributes.

A change mask is created to capture all modifications, including both removed and newly added roads. The mask is generated by concatenating the deleted road segments and the newly added roads. This ensures that all road network changes are explicitly recorded in the mask layer, providing a clear reference for change detection.

The modified dataset, incorporating both deleted and newly added roads, is saved as a new GeoPackage file (`output_path`). Additionally, the change mask is stored separately (`mask_path`) to be used later in change detection tasks.

Railway manipulation follows the same workflow as road manipulation, involving random deletion of segments, expansion of the network, and generation of a change mask. The only difference lies in the attribute fields: railway symbols and road symbols are stored under different field names in the dataset. This distinction is accounted for when assigning attributes to newly generated railway segments, ensuring consistency with the existing data structure.

3.2.3 River, lake manipulation

Lake layer manipulation involves four primary operations: deletion, translation, expansion, and shrinkage of lakes. To avoid unintended impacts on other features, only a small portion of lakes and rivers were selected for modification, as large-scale changes could potentially affect the surrounding landscape. The following code snippets belonging to `lake_change.py` outline the implementation of these transformations.

```
import geopandas as gpd
import pandas as pd
from shapely.affinity import translate, scale

```

```python
def process_lake_layer(map_path, output_path, mask_path):
    """Delete, translate, expand, and shrink lake data, and save the mask including only changed areas"""
    # Read original data
    gdf_original = gpd.read_file(map_path, layer="T56_DKM25_GEWAESSER_PLY")
    gdf_modified = gdf_original.copy()

    # Select 10% of lakes with an area smaller than 300 for deletion
    small_lakes = gdf_modified[gdf_modified['Shape_Area'] < 300]
    to_delete = small_lakes.sample(frac=0.1, random_state=1).index
    gdf_modified = gdf_modified.drop(index=to_delete)

    # Select 10% of lakes with an area smaller than 300 for translation
    to_translate = small_lakes.sample(frac=0.1, random_state=2).index
    gdf_modified.loc[to_translate, 'geometry'] = gdf_modified.loc[to_translate, 'geometry'].apply(
        lambda geom: translate(geom, xoff=50, yoff=50)
    )

    # Select 20% of lakes with an area smaller than 500 for expansion
    medium_lakes = gdf_modified[gdf_modified['Shape_Area'] < 500]
    to_expand = medium_lakes.sample(frac=0.2, random_state=3).index
    gdf_modified.loc[to_expand, 'geometry'] = gdf_modified.loc[to_expand, 'geometry'].apply(
        lambda geom: scale(geom, xfact=1.2, yfact=1.2)
    )

    # Select 20% of lakes with an area smaller than 500 for shrinking
    to_shrink = medium_lakes.sample(frac=0.2, random_state=4).index
    gdf_modified.loc[to_shrink, 'geometry'] = gdf_modified.loc[to_shrink, 'geometry'].apply(
        lambda geom: scale(geom, xfact=0.8, yfact=0.8)
    )

    # Use symmetric_difference to compute all changed areas
    mask = gpd.overlay(gdf_modified, gdf_original, how='symmetric_difference')

    # Save modified lake data and mask data
    gdf_modified.to_file(output_path, layer="T56_DKM25_GEWAESSER_PLY", driver="GPKG")
    mask.to_file(mask_path, layer="T56_DKM25_GEWAESSER_PLY_mask", driver="GPKG")

    print(f"Modified lake data saved to {output_path}")
    print(f"Mask of all changes saved to {mask_path}")
```

The function `process_lake_layer()` reads the lake dataset from a

GeoPackage file using `geopandas.read_file()`, specifically selecting the layer `"T56_DKM25_GEWAESSER_PLY"`, which contains lake geometries. A copy of the original dataset is created for modification.

To simulate the disappearance of smaller lakes over time due to natural or anthropogenic influences, 10% of lakes with an area smaller than 300 square units are randomly selected and removed from the dataset. Another type of lake modification involves the shifting of lakes, which can represent addition of new lakes. To simulate this, another 10% of lakes smaller than 300 square units are selected and translated by shifting them 50 units in both the x and y directions using `shapely.affinity.translate()`. To reflect lake growth due to factors such as increased water levels or land subsidence, 20% of lakes with an area smaller than 500 square units are randomly selected and enlarged. This is achieved by scaling their geometries by a factor of 1.2 along both the x and y axes using `shapely.affinity.scale()`. Conversely, some lakes may shrink over time due to climate change, droughts, or human intervention. To simulate this process, 20% of lakes smaller than 500 square units are selected and reduced in size by scaling their geometries by a factor of 0.8 along both the x and y axes.

A change mask is created to capture all modifications, including deleted, translated, expanded, and shrunken lakes. This is achieved using `geopandas.overlay()` with the `symmetric_difference` method, which extracts the differences between the original and modified datasets. The resulting mask highlights all areas that have undergone change.

The modified dataset, incorporating the translated, expanded, and shrunken lakes while excluding the deleted ones, is saved as a new GeoPackage file (`output_path`). Additionally, the change mask, which records all lake modifications, is stored separately (`mask_path`).

3.2.4 Stream manipulation

Stream layer manipulation consists of three key operations: random deletion of short stream segments, translation of selected streams to create new ones, and modification of existing stream geometries through extension or shortening. The following code snippets belonging to `stream_change.py` outline the implementation of these transformations.

```
1 import geopandas as gpd
2 import random
3 from shapely.geometry import LineString, MultiLineString
4 import pandas as pd
5
6 def process_stream_layer(map_path, output_path, mask_path):
7     """Delete, add, and adjust stream data, saving the modified
    dataset and a mask of all changes."""
8     original_gdf = gpd.read_file(map_path, layer="
    T57_DKM25_GEWAESSER_LIN")
```

```
    gdf = original_gdf.copy()
    mask_gdf = gpd.GeoDataFrame(columns=gdf.columns, geometry=gdf.geometry.name, crs=gdf.crs)

    # Randomly delete 10% of stream segments with Shape_Length < 50
    small_length_gdf = gdf[gdf['Shape_Length'] < 50]
    to_delete = small_length_gdf.sample(frac=0.1, random_state=1).index
    deleted_features = gdf.loc[to_delete]

    # Add deleted features to the mask
    mask_gdf = pd.concat([mask_gdf, deleted_features], ignore_index=True)

    # Remove these features and update gdf
    gdf = gdf.drop(index=to_delete)

    # Select 50% of streams with a length < 200 and translate them to create new streams
    short_streams = gdf[gdf['Shape_Length'] < 200].sample(frac=0.5, random_state=2)
    new_streams = short_streams.copy()
    new_streams['geometry'] = new_streams['geometry'].translate(xoff=50, yoff=50)

    # Add newly generated streams to the mask
    mask_gdf = pd.concat([mask_gdf, new_streams], ignore_index=True)

    gdf = pd.concat([gdf, new_streams], ignore_index=True)
```

The function `process_stream_layer()` reads the stream dataset from a GeoPackage file using `geopandas.read_file()`, specifically selecting the layer `"T57_DKM25_GEWAESSER_LIN"`, which contains stream geometries. A copy of the original dataset is created to apply modifications, ensuring that the original data remains unchanged while introducing synthetic stream changes.

To represent the disappearance of smaller streams, 10% of the stream segments with a length smaller than 50 units are randomly selected and removed from the dataset. These deleted features are stored separately in a mask layer to track their modifications. To simulate the formation of new water, 50% of the streams with a length below 200 units are selected and translated by shifting them 50 units in both the x and y directions. The newly created streams are added back to the dataset, and their corresponding changes are recorded in the mask layer.

```
    # Randomly extend or shorten some streams
    modified_streams = []
    modified_attributes = []
    modified_masks = []
    mask_attributes = []

```

```
    for idx, row in gdf.sample(frac=0.1, random_state=3).iterrows():
        multiline = row['geometry']
        new_geoms = []
        mask_geoms = []

        if isinstance(multiline, MultiLineString):
            for line in multiline.geoms:
                if random.random() > 0.5:
                    # Extend the stream
                    new_coord = (line.coords[-1][0] + random.uniform(10, 50), line.coords[-1][1] + random.uniform(10, 50))
                    extended_line = LineString(list(line.coords) + [new_coord])
                    new_geoms.append(extended_line)

                    # Record the newly added segment in the mask
                    mask_geoms.append(LineString([line.coords[-1], new_coord]))
                    mask_attributes.append(row.drop(labels='geometry'))
                else:
                    # Shorten the stream
                    if len(line.coords) > 2:  # Ensure at least two points remain
                        shortened_line = LineString(line.coords[:-1])
                        new_geoms.append(shortened_line)

                        # Record the removed segment in the mask
                        mask_geoms.append(LineString([line.coords[-2], line.coords[-1]]))
                        mask_attributes.append(row.drop(labels='geometry'))
                    else:
                        new_geoms.append(line)  # Keep unchanged if too few points remain

            modified_streams.append(MultiLineString(new_geoms))
            modified_attributes.append(row.drop(labels='geometry'))
            modified_masks.extend(mask_geoms)

    # Create a GeoDataFrame for modified streams while retaining original attributes
    modified_stream_gdf = gpd.GeoDataFrame(modified_attributes, geometry=modified_streams, crs=gdf.crs)

    # Add modified segments to the mask while keeping corresponding attributes
    mask_increment_gdf = gpd.GeoDataFrame(mask_attributes, geometry=modified_masks, crs=gdf.crs)
    mask_gdf = pd.concat([mask_gdf, mask_increment_gdf], ignore_index=True)

```

```
    # Add modified streams to gdf
    gdf = pd.concat([gdf, modified_stream_gdf], ignore_index=True)

    # Assign appropriate change type labels to mask data
    mask_gdf['Change_Type'] = ['deleted' if idx in to_delete else '
    translated' if idx in new_streams.index else 'modified' for idx
     in mask_gdf.index]

    # Save the modified stream data and mask data
    gdf.to_file(output_path, layer="T57_DKM25_GEWAESSER_LIN",
    driver="GPKG")
    mask_gdf.to_file(mask_path, layer="T57_DKM25_GEWAESSER_LIN_mask
    ", driver="GPKG")

    print(f"Modified stream data saved to {output_path}")
    print(f"Mask of all changes saved to {mask_path}")
```

To account for natural changes in stream length, 10% of the stream segments are randomly modified. If selected for extension, a new coordinate is added to the endpoint of the stream, increasing its length. If selected for shortening, the last coordinate is removed, reducing the stream's length. These modifications simulate real-world processes such as erosion, sediment deposition, or stream channeling. The segments that were extended or removed are recorded in the change mask.

The modified dataset, incorporating newly generated, extended, shortened, and remaining streams while excluding the deleted ones, is saved as a new GeoPackage file (`output_path`). Additionally, the change mask, which highlights all altered stream segments, is stored separately (`mask_path`).

3.2.5 Forest manipulation

Forest layer manipulation consists of two key operations: random deletion of small forest areas and buffer-based expansion or shrinkage. The following code snippets belonging to `forest_change.py` outline the implementation of these transformations.

```
import geopandas as gpd
import pandas as pd
import numpy as np

def apply_buffer_and_delete(gdf, area_threshold=1000,
    delete_fraction=0.1):
    """Apply buffer to small forest areas and randomly delete a
    fraction, returning the changed areas (buffered and deleted).
    """
    # Filter small area features and randomly delete a fraction
    small_area_gdf = gdf[gdf['Shape_Area'] < area_threshold]
    to_delete = small_area_gdf.sample(frac=delete_fraction,
    random_state=1).index
```

```
    deleted_features = gdf.loc[to_delete]
    filtered_gdf = gdf.drop(index=to_delete)

    # Compute buffer changes
    buffer_values = np.random.uniform(-20, 25, size=len(
    filtered_gdf))
    buffered_geometries = filtered_gdf['geometry'].buffer(
    buffer_values)
    buffer_changes = buffered_geometries.symmetric_difference(
    filtered_gdf['geometry'])

    # Merge deleted geometries and buffer-induced changes
    all_changes = pd.concat([buffer_changes, deleted_features['
    geometry']])

    # Update filtered_gdf geometries to buffered geometries
    filtered_gdf['geometry'] = buffered_geometries

    return filtered_gdf, all_changes

def process_forest_layer(map_path, output_path, mask_path):
    """Process the forest layer and save the results."""
    gdf = gpd.read_file(map_path, layer="T65_DKM25_BODENBEDECKUNG")
    modified_gdf, all_changes = apply_buffer_and_delete(gdf)

    # Save all changed areas as a mask
    mask = gpd.GeoDataFrame(geometry=all_changes, crs=gdf.crs)

    # Save modified data and mask data
    modified_gdf.to_file(output_path, layer="
    T65_DKM25_BODENBEDECKUNG", driver="GPKG")
    mask.to_file(mask_path, layer="T65_DKM25_BODENBEDECKUNG_mask",
    driver="GPKG")

    print(f"Modified forest data saved to {output_path}")
    print(f"Mask of all changes saved to {mask_path}")
```

The function `process_forest_layer()` reads the forest dataset from a GeoPackage file using `geopandas.read_file()`, specifically selecting the layer `"T65_DKM25_BODENBEDECKUNG"`, which contains forest geometries.

To simulate forest loss due to deforestation, land-use change, or natural disturbances, 10% of forest patches with an area smaller than 1000 square units are randomly selected and removed from the dataset. This deletion step reflects real-world scenarios where smaller forested regions are more susceptible to urban expansion, logging, or environmental changes. To account for dynamic changes in forest boundaries over time, a buffer operation is applied to the remaining forest geometries. The buffer values, randomly selected from a range of -20 to 25 units, simulate both forest expansion and shrinkage: Positive buffer values lead to expansion, representing forest regrowth or afforestation. Negative buffer values lead to shrinkage, reflecting gradual deforestation or land degradation. The

changes introduced by the buffer operation are computed using the symmetric difference between the original and modified geometries, ensuring that only the altered areas are captured.

A change mask is created to capture both the deleted forest patches and the areas modified by the buffer operation. This mask is generated by merging the deleted geometries with the buffer-induced changes.

The modified dataset, incorporating the expanded, shrunken, and remaining forests while excluding the deleted ones, is saved as a new GeoPackage file (`output_path`). Additionally, the change mask, which highlights all affected areas, is stored separately (`mask_path`).

3.2.6 Overall change simulation

For the contour and depth contour layers, no modifications are applied; instead, they are directly copied from the original dataset into the manipulated dataset to maintain consistency.

In the `oldNational` manipulation class, calling the functions `process_forest_layer`, `process_stream_layer`, `process_lake_layer`, `process_road_layer`, `process_railway_layer`, and `process_building_layer` applies the respective transformations to generate the manipulated layers.

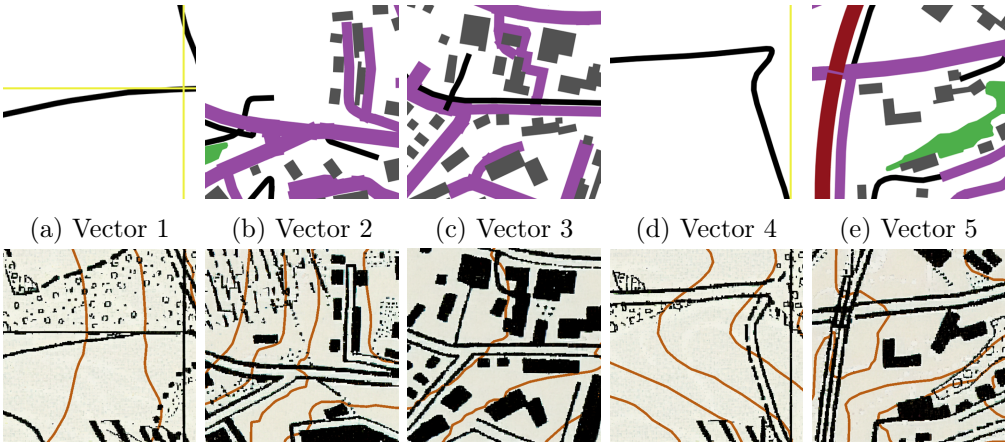
The corresponding change masks are saved in `mask_path`, serving as **ground truth** for change detection tasks.

3.3 Map tile generation

3.3.1 Evaluation of ControlNet

Affolter (2024) trains three specialized ControlNet models for Swisstopo style, Siegfried style, and Old National style maps, among which Swisstopo style map is the modern design and the other two styles are historical designs. According to Figure 2.3, 2.4 and2.5, after post-processing, the Swisstopo-style map tiles closely resemble the real maps. In contrast, the raw outputs for the Old National and Siegfried style map tiles are less accurate due to inconsistencies between the vector data used for training and the real maps. Additionally, the Siegfried style maps are significantly affected by map labels, further degrading the generation quality. As a result, these outputs heavily rely on post-processing and augmentation methods. The following sections analyze the generated Old National and Siegfried style map tiles and explain why the Old National map is chosen for further analysis in this project.

Siegfried style



(a) Vector 1 (b) Vector 2 (c) Vector 3 (d) Vector 4 (e) Vector 5

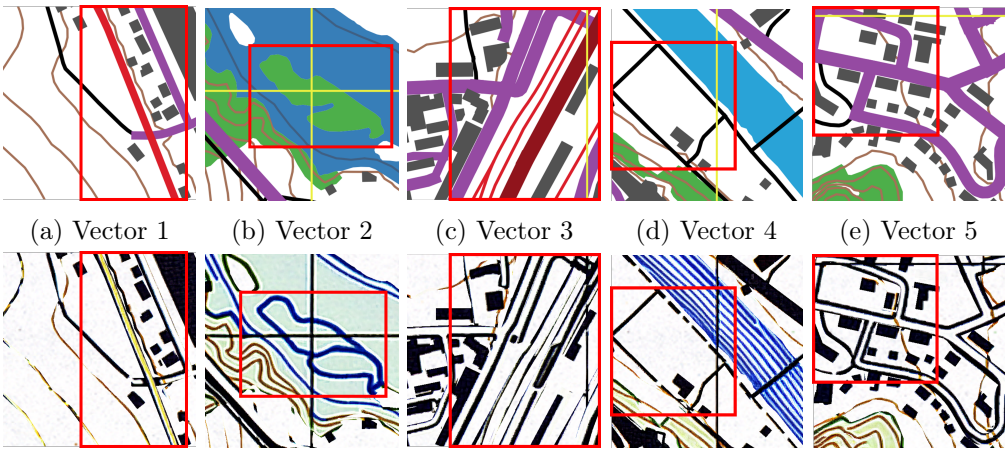
(f) Map 1 (g) Map 2 (h) Map 3 (i) Map 4 (j) Map 5

Figure 3.1: Example of generated Siegfried map tiles

Figure 3.1 presents the results of generating Siegfried-style map tiles from the corresponding vector maps. The first row shows the vector maps, while the second row illustrates the generated outputs. The generated results indicate that ControlNet has largely succeeded in producing textures and features that closely resemble those of the real maps. But upon closer inspection, several artificial labels, along with random textures and structures, can be observed in the background regions. Figure 3.1f shows that a forest texture has been generated in the background area, while an unintended label appears in the lower right corner. In Figure 3.1g, random textures are present, and some roads and buildings are missing. Similarly, in Figure 3.1h, certain buildings and roads are also missing. Moreover, in both Figure 3.1g and 3.1h, the differentiation between different types of roads in the generated outputs is not very distinct. Figure 3.1i and Figure 3.1j illustrate the issues in the generation of roads and railways. The distinction between different road types is not very clear in the generated map tiles, and the road symbols do not always match the vector data. Additionally, railways and roads are not easily distinguishable in the generated outputs.

Old National style

Figure 3.2 presents the results of generating Old National-style map tiles from the corresponding vector maps. The first row shows the vector maps, while the second row illustrates the generated outputs. From Figure 3.2f, it can be observed that the quality of features has decreased after post-processing. This is due to discrepancies in the feature sizes between the generated images and the vector maps. For instance, in the generated map, railways represented in red become less distinct after background cleanup based on the vector map. In Figure 3.2g, forest patches within the river are depicted in blue instead of green. From 3.2h, it is evident that the distinction between railways (represented in red) and roads



(a) Vector 1 (b) Vector 2 (c) Vector 3 (d) Vector 4 (e) Vector 5

(f) Map 1 (g) Map 2 (h) Map 3 (i) Map 4 (j) Map 5

Figure 3.2: Example of generated Old National map tiles

(represented in purple) in the vector map is not very clear. In Figure 3.2i, the representation of roads appears somewhat random. Although roads of the same type are consistent in the vector map, their depiction in the generated map varies. However, in Figure 3.2j, different types of roads remain clearly distinguishable in the generated map tiles.

Overall, the Old National-style map is less affected by fake labels and random textures, allowing for a more accurate representation of features. Additionally, it provides finer details, such as the distinction between different road types, making it more suitable for manipulation and change detection.

3.3.2 OldNational style map tile and change mask generation

Based on Claudio's work, with all relevant vector data symbolized and adjusted for the best possible alignment with the raster data, the vector map, i.e., conditioning images with which Stable Diffusion will be controlled could be created. First, the vector dataset needs to be obtained for map tile generation. For this purpose, large vector map image pairs (i.e., symbolized vector data) with a size of 2500 × 2000 mm (29527 × 23622 pixels) were downloaded using QGIS' Print Layout. At the same time, the manipulated vector map is symbolized in QGIS using the same symbolization as the original vector map, resulting in the symbolized manipulated vector map. The changed areas are stored in the change mask. They are symbolized in the same way as the other layers, and then all colors are set to white. By setting the background color to white when printing layout, the ground truth for change detection can be obtained. Subsequently, the large mapsheets are divided into 512 × 512 pixel tiles. A total of 2622 tiles are obtained. The corresponding before- and after-change OlaNational style map tiles

are generated using Claudio's trained specialized `OlaNational.ckpt` model. After generating the Old National-style map tiles, the background areas are cleaned up during post-processing. For this, these regions are replaced with the original Old National Map background texture.

3.4 OldNational style map change detection

Change detection is performed based on the generated pre- and post-change map tiles, including overall change detection and building change detection. Since buildings are the most clearly represented features in the generated map tiles and have reference images for evaluating the model's performance, building change detection is specifically conducted. Below are the specific steps of the process.

3.4.1 Train building change detection model

In building change detection, only a single map image pair is used, and the change mask includes only building changes. The specific procedures for each model are as follows:

ChangeFormer: The generated 512×512 pixel map tiles are first divided into 256×256 pixel tiles to match the model's training input size. The pre-change map tile, post-change map tile, and change mask are then organized into three separate folders: `A`, `B`, and `label`. Additionally, in the `list` folder, text files are created to store the image filenames for training, validation, and testing. Once this setup is complete, the training process can be executed.

MambaBCD: The image splitting method is the same as in ChangeFormer. However, the file organization differs. In MambaBCD, the training, validation, and test images are placed in separate directories, each containing three subfolders: `T1`, `T2`, and `GT`, which store the pre-change images, post-change images, and ground truth masks, respectively. The corresponding filenames must be saved in designated text files for each dataset split.

Each large mapsheet initially yields #2,622 512×512 pixel map tiles, which are further divided into #10,488 256×256 pixel map tiles. Table 3.2 presents the number of map tiles used for training.

Dataset split	# Tiles
Training	7 120
Validation	1 024
Test	2 048

Table 3.2: Number of tiles for training, validation, and testing for building change detection.

3.4.2 Train overall change detection model

When performing change detection on all features, the process follows the same workflow as building change detection. The key difference is that the mask used represents all changes rather than just building changes. Additionally, since the dataset for training the overall change detection model is larger, only ChangeFormer is used. Table 3.3 presents the number of map tiles used for training.

Dataset split	# Tiles
Training	14 240
Validation	2 048
Test	4 096

Table 3.3: Number of tiles for training, validation, and testing.

3.4.3 Evaluation on real datasets

To evaluate the performance of our trained model on real-world datasets, we conducted testing using maps from the same region but from different years—specifically, 1:25,000 scale maps from 1954 and 1987. This analysis aimed to assess the performance of both the building change detection and overall change detection models.

Since ControlNet generates map tiles based on 1:5,000 vector maps, the TIFF files of the 1954 and 1987 1:25,000 Old National-style maps were imported into QGIS. The scale was set to 1:5,000, and the maps were exported as 2000×1500 mm ($23,622 \times 17,716$ pixels) images. These images were then divided into smaller tiles, resulting in a total of #6,348 256×256 pixel map tiles. The inference was performed on these tiles to generate the change maps.

Although ground truth maps for buildings are available, the representations of the same buildings in different years are not perfectly identical, leading to boundary inconsistency. As a result, the difference maps contain many boundary artifacts. Post-processing further degrades the ground truth maps, making them unsuitable for quantitative evaluation. Therefore, they are used only as a

reference, and the performance of change detection is assessed primarily through visual interpretation. To facilitate visual interpretation, the ground truth maps of buildings from different years are differenced to obtain the true change areas. The `cv2.absdiff` function is used to compute the absolute difference, highlighting the differing regions between the two images. The specific code is as follows:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os

def load_image(png_path):
    """
    Load a PNG image and convert it to a grayscale image.
    """
    img = cv2.imread(png_path, cv2.IMREAD_GRAYSCALE)  # Load as a grayscale image
    return img

def compute_difference(img1, img2):
    """
    Compute the absolute difference between two PNG images.
    """
    diff = cv2.absdiff(img1, img2)
    return diff

# Input PNG image paths
png_path1 = "./samples_oldnational2/1954_building.png"  # Replace with the path to the first PNG image
png_path2 = "./samples_oldnational2/1987_building.png"  # Replace with the path to the second PNG image

# Load PNG images
image1 = load_image(png_path1)
image2 = load_image(png_path2)

# Compute the difference image
difference_image = compute_difference(image1, image2)

# Save the difference image
output_path = "./samples_oldnational2/difference_image.png"
os.makedirs(os.path.dirname(output_path), exist_ok=True)
cv2.imwrite(output_path, difference_image)
print(f"Difference image saved: {output_path}")
```

Chapter 4

Results

This chapter presents the results of the study. First, the vector manipulation result is evaluated. Then, the performance of the change detection models is analyzed.

4.1 Vector manipulation results

4.1.1 Building manipulation results

Figure 4.1 illustrates the results of building vector manipulation. Figure 4.1a shows the vector data before manipulation, while Figure 4.1c presents the vector data after manipulation. Figure 4.1b displays the map tiles generated based on Figure 4.1c, and Figure 4.1d shows the map tiles generated based on Figure 23. Figure 4.1e represents the mask that captures the changes preserved during vector manipulation. From these results, we can see that the buildings are successfully deleted, and the map tiles generated by the model effectively simulate this change.

(a) Before vector (b) After vector (c) Before map (d) After map (e) Change mask

Figure 4.1: Building deletion in OldNational map

Similarly, Figure 4.2 demonstrates the addition of buildings, the generation of corresponding map tiles, and the associated change mask. The newly added building in the lower-left corner partially overlaps with the original building, implicitly simulating changes in building morphology. This indicates that morphological changes can be achieved through translation transformations.

(a) Before vector (b) Before map (c) After vector (d) After map (e) Change mask

Figure 4.2: Building addition in OldNational map

Figure 4.3 illustrates the addition of buildings. Similar to Figure 4.2, when the newly added buildings overlap with the original ones, the resulting structures undergo morphological changes, and our change mask captures the altered regions. However, the figure also reveals that when generating map tiles using the manipulated vector map as control, the trained ControlNet may sometimes fail to produce the corresponding feature types due to changes in the spatial layout of different features. For instance, in Figure 4.3c, a forest patch is present, but it does not appear in Figure 4.3d.

(a) Before vector (b) Before map (c) After vector (d) After map (e) Change mask

Figure 4.3: Building morphology change in OldNational map

4.1.2 Road, railway manipulation results

Figure 4.4 illustrates the addition of new roads, which are clearly visible in the generated map tiles. Figure 4.5 presents the deletion of roads, which can also be observed in the figure. The manipulation and results for railways are similar to those for roads.

(a) Before vector (b) Before map (c) After vector (d) After map (e) Change mask

Figure 4.4: Road expansion in OldNational map

(a) Before vector (b) Before map (c) After vector (d) After map (e) Change mask

Figure 4.5: Road shrinkage in OldNational map

4.1.3 River, lake manipulation results

Figure 4.6 illustrates the addition of a lake. While the manipulation of the vector map was successful, the manipulated vector did not produce the corresponding river, indicating that when water bodies and forests are in close proximity, the model struggles to distinguish between them effectively. Figure 4.7 demonstrates the shrinkage of a lake, which, in this case, was caused by the overlap of a building onto the lake. This suggests that when manipulating vectors without performing conflict detection, certain types of changes can still be simulated. Additionally, since only a limited number of rivers or lakes were modified in the `lake_change.py` script, the amount of change observed in the final generated map tiles used for change detection is relatively small.

(a) Before vector (b) Before map (c) After vector (d) After map (e) Change mask

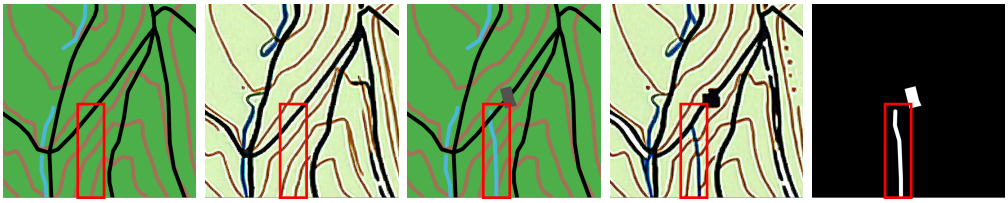
Figure 4.6: Lake expansion in OldNational map

(a) Before vector (b) Before map (c) After vector (d) After map (e) Change mask

Figure 4.7: Lake shrinkage in OldNational map

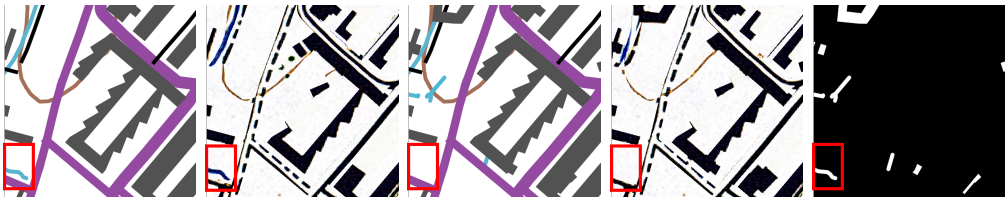
4.1.4 Stream manipulation results

Figure 4.8 represents the addition of a stream, while Figure 4.9 represents the reduction of a stream. Both changes are clearly visible in the generated map tiles and the change mask.



(a) Before vector (b) Before map (c) After vector (d) After map (e) Change mask

Figure 4.8: Stream expansion in OldNational map

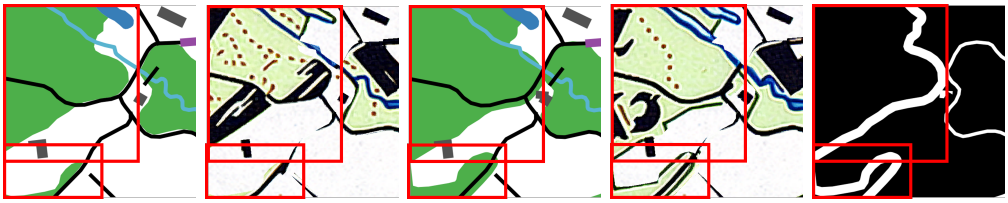


(a) Before vector (b) Before map (c) After vector (d) After map (e) Change mask

Figure 4.9: Stream shrinkage in OldNational map

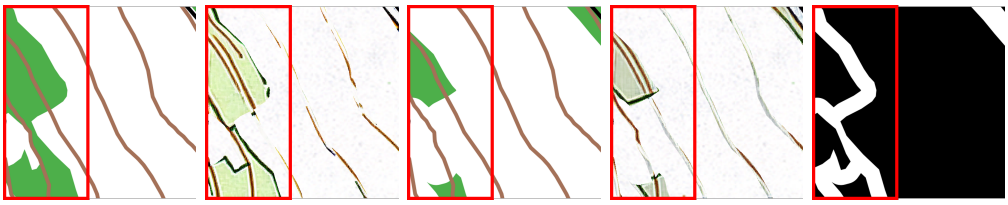
4.1.5 Forest manipulation results

Figure 4.10 illustrates the expansion of a forest. While the generated map tiles display the forest texture, some random textures also appear in the generated map tiles. These artifacts may affect the results of change detection. Figure 4.11 illustrates the shrinkage of the forest, with both the generated map tiles and the change mask successfully capturing this change.



(a) Before vector (b) Before map (c) After vector (d) After map (e) Change mask

Figure 4.10: Forest expansion in OldNational map



(a) Before vector (b) Before map (c) After vector (d) After map (e) Change mask

Figure 4.11: Forest shrinkage in OldNational map

4.2 Change detection results

Table 4.1 presents the evaluation results of different models on the test set. It shows that the models achieve excellent performance on the generated maps; however, their performance on real-world datasets still requires further assessment. The following analysis is based on the real-world datasets.

In these datasets, the ground truth difference maps are obtained by computing the absolute difference between the building maps of two different years. However, due to discrepancies in how the same buildings are represented across different years, the resulting difference maps exhibit boundary artifacts around nearly all unchanged buildings. These artifacts cover a significant portion of the maps, as shown in Appendix A, making quantitative comparison infeasible. Consequently, the following analysis relies solely on visual interpretation.

Model	Recall	Precision	OA	F1	IoU
ChangeFormer	90.64	94.55	98.61	92.49	86.77
ChangeFormer (building)	84.82	90.87	99.28	87.59	79.99
MambaBCD Tiny (building)	85.57	90.23	99.62	87.84	78.31

Table 4.1: Performance comparison of different models on test set.

4.2.1 Building change detection results

Figure 4.12, 4.13, and 4.14 present the results of building change detection for building addition, building deletion, and building morphology changes, respectively. It can be observed that both ChangeFormer and MambaBCD perform well on the real-world dataset. Additionally, in Figure 4.13, the model successfully detects real changes that are not annotated in the ground truth map. Since our models perform pixel-level detection, they can also capture morphological changes, such as building fusion and building shape alterations.

(a) 1954 map (b) 1987 map (c) ChangeForm (d) MambaBCD (e) GT

Figure 4.12: Building addition detection

(a) 1954 map (b) 1987 map (c) ChangeForm (d) MambaBCD (e) GT

Figure 4.13: Building deletion detection

(a) 1954 map (b) 1987 map (c) ChangeForm (d) MambaBCD (e) GT

Figure 4.14: Building morphology detection

4.2.2 Overall change detection results

Figure 4.15 and 4.16 show the model's detection results for forest shrinkage. It can be observed that the model, trained on the synthetic maps, generalizes well to detecting forest changes in the real-world dataset.

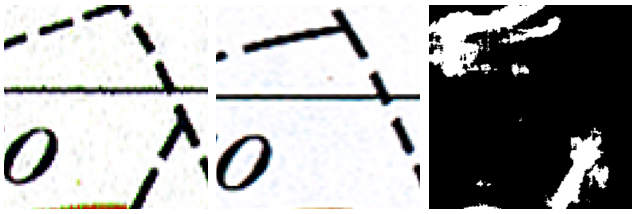
(a) 1954 map (b) 1987 map (c) Change map

Figure 4.15: Forest change detection 1

(a) 1954 map (b) 1987 map (c) Change map

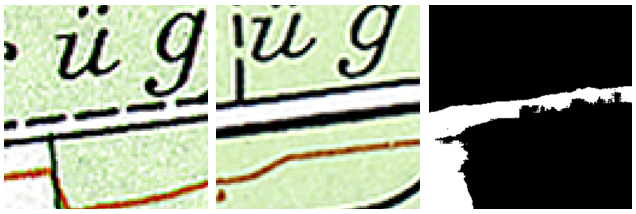
Figure 4.16: Forest change detection 2

Figure 4.17, 4.18, and 4.19 present the model's detection results for road changes. Figure 4.17 shows that the model successfully detects the addition of a road in the lower-right corner. At the same time, it can be observed that the same road in the upper-left corner appears in different positions on the two maps, causing the model to classify this area as a change. Figure 4.18 demonstrates the model detecting changes in road type. However, since this type of change was not explicitly simulated in the synthetic maps, it is inferred that some wide roads are represented by two solid lines, while narrower roads are depicted by a single solid line. The model may interpret this as a road addition, leading to the detected change. Figure 4.19 indicates that the model fails to detect a road widening change. This could be attributed to the lack of clear road type differentiation in the generated maps and the degradation of road representations during post-processing.



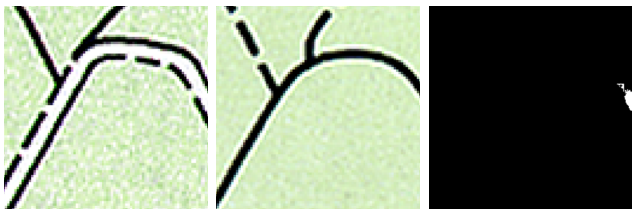
(a) 1954 map (b) 1987 map (c) Change map

Figure 4.17: Road change detection 1



(a) 1954 map (b) 1987 map (c) Change map

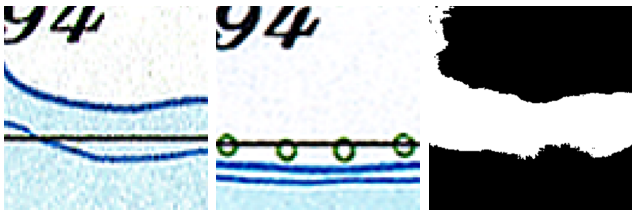
Figure 4.18: Road change detection 2



(a) 1954 map (b) 1987 map (c) Change map

Figure 4.19: Road change detection 3

Figure 4.20 shows that the model successfully detects the reduction of river areas, indicating that the model generalizes well to river change detection in the real-world datasets.



(a) 1954 map (b) 1987 map (c) Change map

Figure 4.20: River change detection 1

Figure 4.21 shows that the model detects an increase in streams. However, in reality, streams are represented as solid lines on the map rather than dashed lines. The additional lines in Figure 4.21b may not actually represent streams but rather some other features with a similar appearance. This suggests that when performing change detection on real-world datasets, the results can be influenced by other feature types. Additionally, this type of feature was not considered when generating the synthetic maps.

(a) 1954 map (b) 1987 map (c) Change map

Figure 4.21: Stream change detection 1

Figure 4.22 presents the model's detection results for other types of changes. It can be observed that the model's performance is not ideal. Although the figure contains changes related to roads, buildings, and trees, it is difficult to determine whether the detected changes truly correspond to these features. The detection is influenced by other elements, such as trees, which were not included in the synthetic maps. Moreover, the representation of trees closely resembles that of forests, further contributing to the confusion in the detection results.

(a) 1954 map (b) 1987 map (c) Change map

Figure 4.22: Other change detection 1

4.2.3 Urban change pattern recognition

Figure 4.23 presents the 1954 Old National-style map used for testing, while Figure 4.24 shows the 1987 map of the same region. Figure 4.25 illustrates the change detection results obtained using the trained ChangeFormer on the real-world dataset.

Observing the real maps, it is evident that urban changes over these decades were primarily driven by building additions, with relatively fewer building deletions. Consequently, most detected building changes are concentrated on building additions. From this perspective, we can infer the urban evolution of the Bern metropolitan area over these decades: During this period, Bern underwent significant urbanization, characterized by a rapid increase in buildings. The urban expansion occurred on top of the existing infrastructure, with new buildings spreading outward from already developed areas. Additionally, scattered developments appeared in regions further away from the city center, aligning with the

well-known patterns of urbanization.

For comparison, Appendix A includes change maps obtained by computing differences from the ground truth, as well as change detection results using MambaBCD.

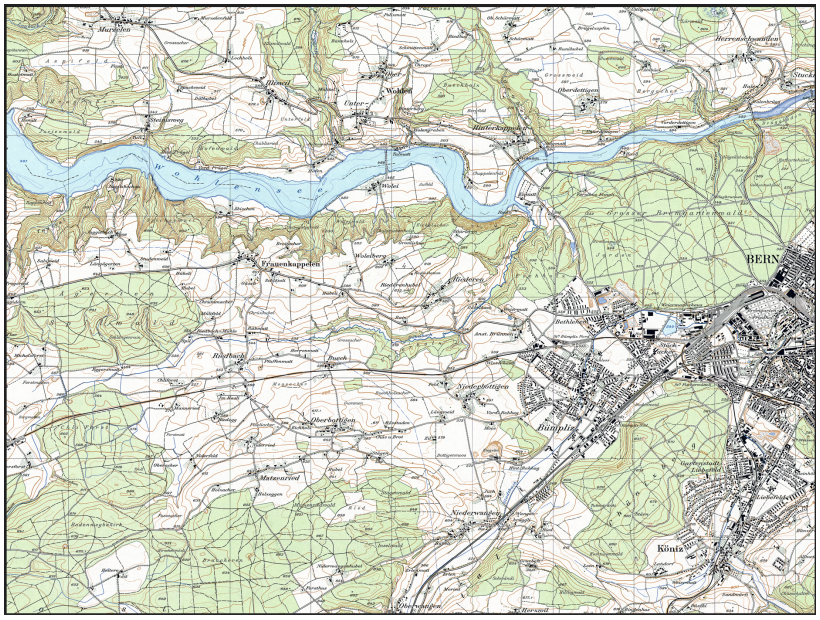


Figure 4.23: 1954 map

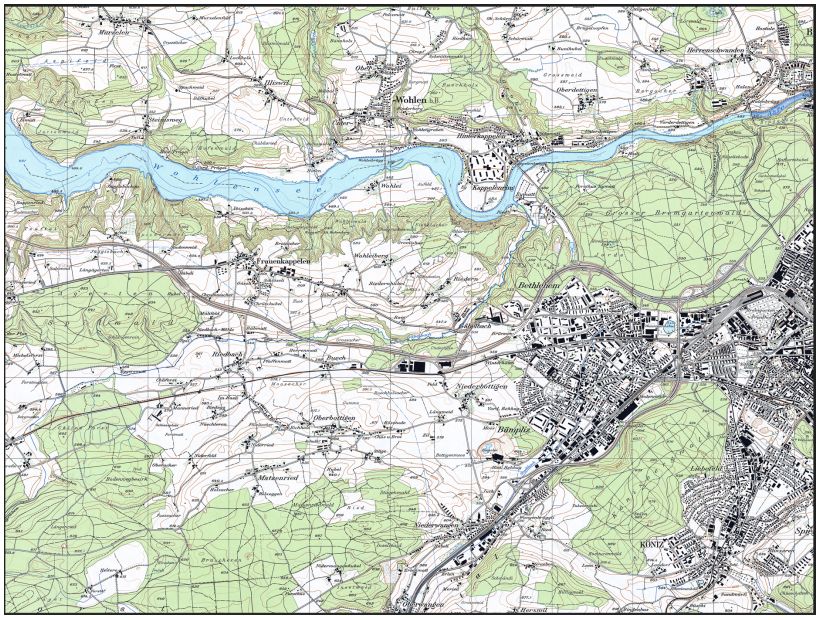


Figure 4.24: 1987 map

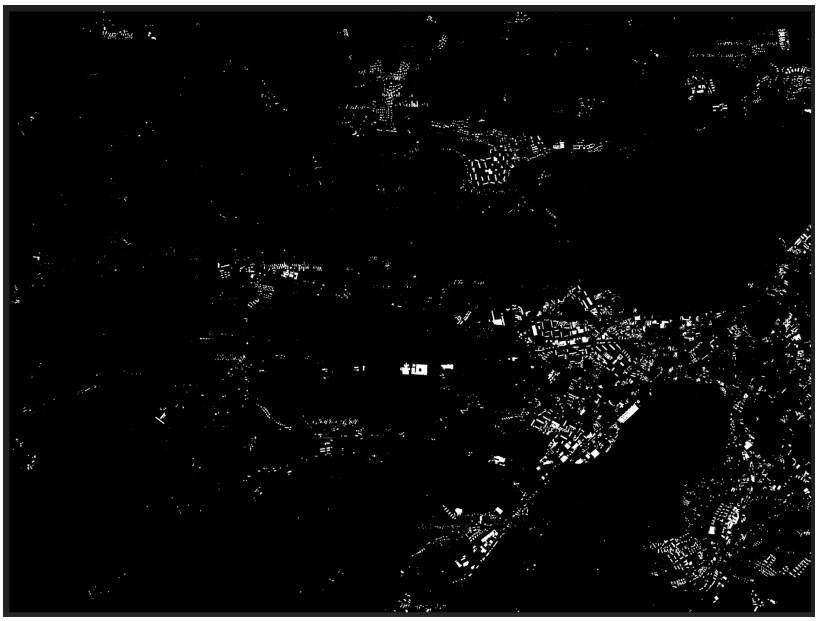


Figure 4.25: Building change detection result

Chapter 5

Discussion

In this chapter, the results of this thesis are first interpreted and discussed. Additionally, potential consequences and limitations are highlighted.

5.1 Interpretation and discussion of results

Vector manipulation

Due to these reasons:

- Historical maps discretely represent geospatial objects, making it difficult to distinguish urban and non-urban areas.
- Urban morphology classification requires additional information, such as land parcel delineation.
- Urban changes are primarily reflected in individual geospatial objects rather than large-scale layout shifts.

Urban change is not classified based on infilling, edge-expansion, and outlying, as defined in urban planning, but instead by changes in different geospatial objects, which were then classified and simulated accordingly.

Based on the results of vector manipulation, the different types of feature changes summarized in Table 3.1 are largely covered.

- Building changes such as addition, deletion, and morphology modifications can all be manipulated through copying, deleting, and overlapping features.
- For roads and railways, addition and elongation can be achieved by extending from existing nodes, while deletion and shortening can be accomplished by removing segments. However, widening and narrowing changes were not manipulated. Modifying the attribute table could be a potential way to simulate these types of changes.

- River and lake expansion and shrinkage can be simulated through copying, deleting, constructing inward/outward buffers, and removing features.
- Stream expansion and shrinkage are achieved by copying, deleting, and adding or removing nodes.
- Forest expansion and shrinkage can be simulated by constructing inward/outward buffers and deleting features.

Map generation

The specialized ControlNet for generating Old National-style map tiles can produce maps with textures and structures similar to those in real-world datasets. Training change detection models on these generated maps allows them to be applied to real datasets, validating the model's performance.

However, there are some discrepancies between the synthetic maps and real maps. Certain detailed features, such as trees, are not represented. Additionally, the road generation process does not accurately capture road types, and post-processing further degrades the quality of the generated features. In some cases, the model also struggles to generate forests and water bodies accurately. This is partly due to different degrees of misalignment between vector and raster data during ControlNet training, and partly because the vector map does not cover all the features present in the real Old National-style maps. To make the generated maps more similar to real datasets, the labels need to be overlaid onto the generated maps.

Change detection

Both ChangeFormer and MambaBCD perform well in building change detection on the real-world dataset, successfully detecting almost all building changes. This indicates that the simulation of building changes was highly effective. Additionally, MambaBCD demonstrates slightly better results than ChangeFormer. The detected building change maps can also reveal urbanization trends and spatial distribution, providing valuable insights for urban change research. Although extracting buildings or other features from different years and computing their differences can generate change maps, change detection offers a more streamlined approach that eliminates the need for additional processing steps.

ChangeFormer's performance in overall change detection is not as sensitive as in building change detection. While it can successfully detect changes in buildings, forests, some roads, large water bodies, and streams, several issues remain:

- The model is influenced by other features that have similar symbolization to those in the synthetic maps, leading to false positives, where unchanged features are misclassified as changes.

- The model can only detect some road additions, while certain newly added roads remain undetected.
- Due to misalignment between the same features in maps from different years, the model mistakenly classifies these discrepancies as actual changes.

5.2 Consequences

Based on the results presented in Chapter 4, the following can be said:

- The vector manipulation successfully simulates various urban changes, and testing on real-world datasets further validates its effectiveness.
- The specialized ControlNet for generating Old National-style map tiles can produce maps with textures and structures similar to those in real-world datasets. However, discrepancies exist (e.g., road classes are not represented correctly and trees are not generated.)
- Although change detection models successfully identify major changes, artifacts in synthetic maps and similarities in feature representations in historical maps impact detection accuracy, potentially leading to false positives.
- Simple features like buildings, forests, streams, rivers, and lakes are clearly represented in the synthetic maps, leading to better change detection results. However, other features, such as roads, have multiple categories in the Old National-style map. Since the trained ControlNet fails to generate these different road types correctly, it also affects the accuracy of change detection results.

5.3 Limitations

Despite the promising results, this study has several limitations that need to be addressed:

- Change simulation is conducted at a microscopic scale, focusing on modifications of geospatial objects, without considering larger-scale transformations. A more systematic classification could be applied to capture broader patterns of change.
- Only binary change detection is conducted to identify change regions, without performing semantic change detection to distinguish from-to changes or to classify whether a feature is added or removed.

- Since vector manipulation is performed to achieve the change detection objective, certain constraints are applied to control the scale of changes, such as limiting alterations in rivers and lakes. However, other types of conflict checks, such as conflicts between buildings and other features, are not considered. If the goal is to generate realistic maps, conflict detection rules need to be established. On the other hand, if the focus is solely on optimizing model performance for change detection, maximizing the number of simulated changes without considering conflicts is more beneficial.

Chapter 6

Conclusion and Outlook

This thesis introduces a novel approach for urban change detection by simulating vector-based urban modifications and generating synthetic historical map tiles using ControlNet-guided Stable Diffusion. A structured classification of urban change types is established based on geospatial features, and corresponding vector manipulations are implemented to create a dataset of pre- and post-change maps with ground truth change masks. Deep learning models, including ChangeFormer and MambaBCD, are trained on the generated dataset and evaluated on real historical maps in 1954 and 1987 to assess their generalization capabilities. The results demonstrate that the proposed approach successfully captures key urban transformation patterns, particularly in detecting building and forest changes. When applied to real historical maps, the models effectively identified large-scale urban expansion, revealing that Bern undergoes significant growth during this period, with increased building density and the development of new urban clusters. However, challenges remain, including inconsistencies in road classification, feature misalignment between different years, and the influence of other map symbols on detection accuracy. Additionally, the study primarily focuses on binary change detection without distinguishing specific change types such as building addition versus new building deletion.

Future research could enhance the realism of generated maps by improving ControlNet's ability to produce fine-grained details, refining road classification, and implementing conflict detection rules to avoid unrealistic feature overlaps. Addressing spatial misalignment between historical maps and incorporating semantic change detection could further improve model accuracy and interpretability. By overcoming these challenges, diffusion-based synthetic data generation could become a valuable tool for historical map analysis and urban change monitoring.

Bibliography

Affolter, C. (2024). *From vector features to stylized maps – exploration of stable diffusion applied to maps* [Master's thesis]. ETH Zurich.

Armenakis, C., Cyr, I., & Papanikolaou, E. (2002). Change detection methods for the revision of topographic databases. *INTERNATIONAL ARCHIVES OF PHOTOGRAMMETRY REMOTE SENSING AND SPATIAL INFORMATION SCIENCES*, *34*(4), 792–797.

Bandara, W. G. C., & Patel, V. M. (2022). A transformer-based siamese network for change detection. *IGARSS 2022-2022 IEEE International Geoscience and Remote Sensing Symposium*, 207–210.

Camagni, R., Gibelli, M. C., & Rigamonti, P. (2002). Urban mobility and urban form: The social and environmental costs of different patterns of urban expansion. *Ecological economics*, *40*(2), 199–216.

Ceresola, S., Fusiello, A., Bicego, M., Belussi, A., & Murino, V. (2005). Automatic updating of urban vector maps. *Image Analysis and Processing–ICIAP 2005: 13th International Conference, Cagliari, Italy, September 6-8, 2005. Proceedings 13*, 1133–1139.

Chen, H., Song, J., Han, C., Xia, J., & Yokoya, N. (2024). Changemamba: Remote sensing change detection with spatio-temporal state space model. *arXiv preprint arXiv:2404.03425*.

Chen, J., Gao, J., & Yuan, F. (2016). Growth type and functional trajectories: An empirical study of urban expansion in nanjing, china. *Plos One*, *11*(2), e0148389.

Deng, K., Hu, X., Zhang, Z., Su, B., Feng, C., Zhan, Y., Wang, X., & Duan, Y. (2024). Cross-modal change detection using historical land use maps and current remote sensing images. *ISPRS Journal of Photogrammetry and Remote Sensing*, *218*, 114–132.

Doucette, P., Kovalerchuk, B., Kovalerchuk, M., & Brigantic, R. (2009). An evaluation methodology for vector data updating. *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XV*, *7334*, 512–520.

Fang, Z., Jin, Y., Zheng, S., Zhao, L., & Yang, T. (2024). Urbanclassifier: A deep learning-based model for automated typology and temporal analysis of urban fabric across multiple spatial scales and viewpoints. *Computers, Environment and Urban Systems*, *111*, 102132.

Federal Office of Topography swisstopo. (2024a). Digital old national maps [Retrieved July 23, 2024]. https://www.swisstopo.admin.ch/en/digital-old-national-maps

Federal Office of Topography swisstopo. (2024b). National maps [Retrieved July 23, 2024]. https://www.swisstopo.admin.ch/en/national-maps

Federal Office of Topography swisstopo. (2024c). Siegfried map [Retrieved July 17, 2024]. https://www.swisstopo.admin.ch/en/siegfried-map

Johnson, C. W., & Peirce, N. R. (2008). Century of the city: No time to lose. *The Rockefeller Foundation*.

Li, M., Zhang, Z., Lo Seen, D., Sun, J., & Zhao, X. (2016). Spatiotemporal characteristics of urban sprawl in chinese port cities from 1979 to 2013. *Sustainability*, *8*(11), 1138.

Li, X., & Gong, P. (2016). Urban growth models: Progress and perspective. *Science bulletin*, *61*, 1637–1650.

Li, Z., Guan, R., Yu, Q., Chiang, Y.-Y., & Knoblock, C. A. (2021). Synthetic map generation to provide unlimited training data for historical map text detection. *Proceedings of the 4th ACM SIGSPATIAL International Workshop on AI for Geographic Knowledge Discovery*, 17–26.

Liu, Y., Shi, S., Zheng, Z., Wang, J., Tian, S., & Zhong, Y. (2024). Mapchange: Enhancing semantic change detection with temporal-invariant historical maps based on deep triplet network. *arXiv preprint arXiv:2401.11489*.

Moosavi, V. (2022). Urban morphology meets deep learning: Exploring urban forms in one million cities, towns, and villages across the planet. *Machine learning and the city: Applications in architecture and urban design*, 379–392.

Reba, M., & Seto, K. C. (2020). A systematic review and assessment of algorithms to detect, characterize, and monitor urban land change. *Remote sensing of environment*, *242*, 111739.

Rui, Y., & Ban, Y. (2010). Multi-agent simulation for modeling urban sprawl in the greater toronto area. *13th AGILE International Conference on Geographic Information Science 2010, Guimarães, Portugal, 10-14 May 2010*.

Rui, Y., & Ban, Y. (2011). Urban growth modeling with road network expansion and land use development. In *Advances in cartography and giscience. volume 2: Selection from icc 2011, paris* (pp. 399–412). Springer.

Rui, Y., Ban, Y., Wang, J., & Haas, J. (2013). Exploring the patterns and evolution of self-organized urban street networks through modeling. *The European Physical Journal B*, *86*, 1–8.

Shen, J., Liu, C., Ren, Y., & Zheng, H. (2020). Machine learning assisted urban filling. *Proceedings of the 25th CAADRIA Conference, Bangkok, Thailand*, 5–6.

Sun, C., Wu, Z.-f., Lv, Z.-q., Yao, N., & Wei, J.-b. (2013). Quantifying different types of urban growth and the change dynamic in guangzhou using multitemporal remote sensing data. *International Journal of Applied Earth Observation and Geoinformation*, *21*, 409–417.

Sun, Y., Wu, Y., Yu, H., & Li, Y. (2024). Spatiotemporal evolution of urban landscapes in chinese historic water towns (1918–2021). *Landscape Research*, 1–16.

Tian, S., Zhong, Y., Zheng, Z., Ma, A., Tan, X., & Zhang, L. (2022). Large-scale deep learning based binary and semantic change detection in ultra high resolution remote sensing imagery: From benchmark datasets to urban application. *ISPRS Journal of Photogrammetry and Remote Sensing*, *193*, 164–186.

Vanderhaegen, S., & Canters, F. (2017). Mapping urban form and function at city block level using spatial metrics. *Landscape and Urban Planning*, *167*, 399–409.

Wang, D., Kong, L., Chen, Z., Yang, X., & Luo, M. (2022). Physical urban area identification based on geographical data and quantitative attribution of identification threshold: A case study in chongqing municipality, southwestern china. *Land*, *12*(1), 30.

Wang, J., Hadjikakou, M., Hewitt, R. J., & Bryan, B. A. (2022). Simulating large-scale urban land-use patterns and dynamics using the u-net deep learning architecture. *Computers, Environment and Urban Systems*, *97*, 101855.

Wilson, E. H., Hurd, J. D., Civco, D. L., Prisloe, M. P., & Arnold, C. (2003). Development of a geospatial model to quantify, describe and map urban growth. *Remote sensing of environment*, *86*(3), 275–285.

Xu, C., Liu, M., Zhang, C., An, S., Yu, W., & Chen, J. M. (2007). The spatiotemporal dynamics of rapid urban growth in the nanjing metropolitan region of china. *Landscape ecology*, *22*, 925–937.

Yang, Y., Piao, X., Xu, L., Duan, Y., & Yu, X. (n.d.). Multiple parameters controlled generation of historical urban fabric based on conditional diffusion model. *Available at SSRN 4992972*.

Zhang, L., Rao, A., & Agrawala, M. (2023). Adding conditional control to text-to-image diffusion models. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 3836–3847.

Appendix A

Building change map

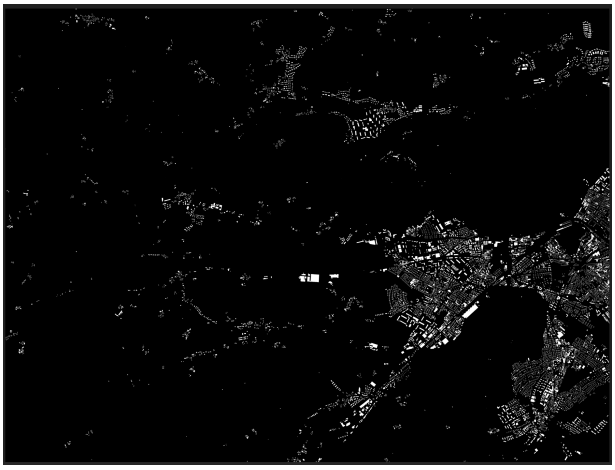


Figure A.1: Building chagne map after differencing

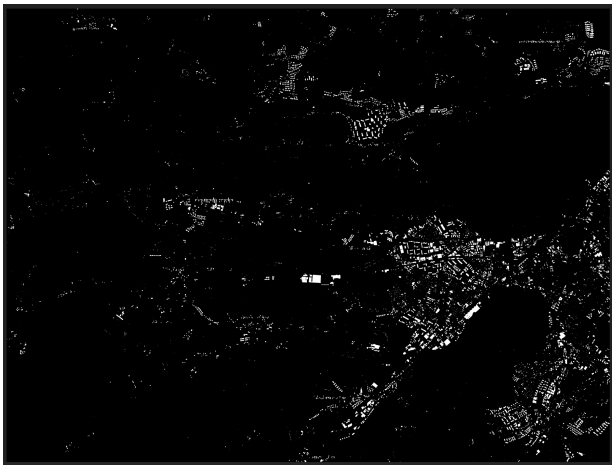


Figure A.2: Building change detection result (MambaBCD)

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every written paper or thesis authored during the course of studies. In consultation with the supervisor, one of the following three options must be selected:

- (●) I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used no generative artificial intelligence technologies[1].
- ( ) I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used and cited generative artificial intelligence technologies[2].
- ( ) I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used generative artificial intelligence technologies[3]. In consultation with the supervisor, I did not cite them.

Title of paper or thesis:

Leveraging Diffusion Models for Urban Change Detection and Classification from Historical Maps

Authored by:
If the work was compiled in a group, the names of all authors are required.

Last name(s):	**First name(s):**
Chunyang	Gao

With my signature I confirm the following:

- I have adhered to the rules set out in the Citation Guide.
- I have documented all methods, data and processes truthfully and fully.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for originality.

Place, date	**Signature(s)**
Zurich, 09.03.2025	Chunyang Gao

If the work was compiled in a group, the names of all authors are required. Through their signatures they vouch jointly for the entire content of the written work.

[1] E.g. ChatGPT, DALL E 2, Google Bard
[2] E.g. ChatGPT, DALL E 2, Google Bard
[3] E.g. ChatGPT, DALL E 2, Google Bard