Cellular Automata for Urban Growth - Team 4

Zhangding Liu
Jayda G Ritchie
DaMarcus D Patterson
Tingyu Liu

Github repo:
https://github.gatech.edu/tliu479/CSE6730-urban-growth

Google Colab:
https://colab.research.google.com/drive/1y52Vt0aoEq4o9S9bIAJDIaFhklNrcwJZ

1.Abstract

The study focuses on analyzing land use patterns as cell states and developing a Cellular Automata (CA) model with Moore neighbors. One of the major innovations of this project is the implementation of distinct transition rules for different urban development intensities, integrating environmental parameters and forest conservation. The model's performance is evaluated using two validation indicators, Overall Accuracy (OA) and Figure of Merit (FOM).

By utilizing real-world land use data from the city of Atlanta, the simulator produces relatively accurate results. Through the analysis and fine-tuning of hyperparameters, we have identified that the transition rule from low to medium intensity urban development plays a crucial role in shaping the observed urban growth patterns in Atlanta.

The resulting CA model serves as a powerful and detailed tool for predicting urban expansion, assisting urban planners, and informing policy-making strategies. This simulator and the information it provides can contribute to the development of sustainable urban planning practices and the efficient allocation of resources.

2. Project Statement

2.1 Background and Problem Statement

Extensive urban growth models have been developed in recent years to study and predict population growth and sprawl over time. Urban growth modeling is a field of study that aims to predict and understand the growth patterns of urban areas. Studying these models can provide insight into the impact of urban expansion on the ambient environment and assist in urban policy-making strategies. The goal of this project is to simulate the urban growth of the City of Atlanta area, a dynamic and complex system that has seen significant urban area changes over recent years. To effectively model and understand how urban areas grow over the years, the Cellular Automata (CA) model will be used.

Atlanta has been experiencing significant urban growth. The metro Atlanta region's total population and employment are forecasted to increase by 51% and 34% respectively between 2015 and 2050. From this growth, it is expected that the Metro Atlanta area will add 2.9 million people by 2050. This rapid urban growth presents many future challenges such as traffic congestion, air pollution, and loss of forest and agricultural lands.

Research also shows that increased urban growth can cause communities to become more segregated by race and class. Therefore, studying urban growth in Atlanta can provide valuable insights into these dynamics and help inform urban planning policies.

2.2 Phenomenon and Terms

2.2.1 Urban Growth

The real-world phenomenon investigated in this study is urban growth. To effectively replicate urban expansion, several elements must be considered within the simulation. These elements are highlighted below, but fall into the categories of land use patterns, spatial constraints and dynamic parameters.

2.2.2 Land Use Patterns and Development Intensity

Analyzing and incorporating existing land use patterns is essential for accurately simulating urban growth. Factors such as residential, commercial, industrial, and recreational areas significantly influence the expansion of urban regions, examples of these are shown in Figure 1.

Figure 1: Examples of the various levels of urban development intensity

High-intensity urban areas are characterized by extensive development, high population density, and a concentration of buildings and infrastructure. Examples include apartment complexes, row houses and commercial/industrial. Impervious surfaces account for 80% to 100% of the total cover.

Medium-intensity urban areas are zones that exhibit a moderate level of development and population density, characterized by a mix of residential, commercial, and industrial zones with a moderate concentration of buildings and infrastructure. Impervious surfaces account for 50% to 79% of the total cover. These areas most commonly include single-family housing units.

A low-intensity urban area is a zone characterized by relatively sparse development, with a lower concentration of buildings and infrastructure. Impervious surfaces account for 20% to 49% percent of total cover. These areas most commonly include single-family housing units.

Non-urban areas include regions that are not extensively developed or densely populated.

2.2.3 Spatial Constraints

Physical constraints, such as topography, natural reserves, water bodies, and legal regulations, impose limitations on urban expansion. Hence, it is necessary to incorporate these constraints into the model to simulate more realistic growth patterns.

2.2.4 Dynamic Parameter

Parameters governing the rules of CA, such as neighborhood interactions, growth rates, and spatial interactions need careful calibration and validation to accurately represent urban growth dynamics.

3 Literature Reviews

3.1 Existing Models and Simulators in Urban Growth

Urban growth modeling has garnered significant interest among researchers and scientists. Various models and simulation techniques have been developed to predict and analyze urban growth patterns. These models serve as valuable tools for understanding the complex dynamics of urban areas. Several models and simulators have been used in urban growth modeling. One notable model is the Cellular Automata (CA) model, which has been widely employed in spatial modeling. The CA model allows for the simulation of urban growth and land use changes. Another prominent model is the SLEUTH model, which offers a comprehensive approach to urban growth simulation, land use change simulation, and planning policy evaluation.

3.2 Existing Work in Urban Growth

The paper written by Aburus et al.[1] provides an overview of various methods as well as strengths and weaknesses of the various models in simulating urban growth. Overall, the objective of the paper is to provide a review of urban Cellular Automata (CA) to determine the most suitable approach for the simulation of land use changes. Furthermore, the paper suggests a novel approach to improve CA model predictions by integrating CA with other models. The models compared in this paper are built with RS and GIS data, and they include a CA model integrated with an Analytic hierarchy process and a CA model integrated with a land transformation model. The result of this paper confirms the validity of CA model for urban modeling. But it also highlights the overall weakness of CA which is its lack of ability to include some of the driving forces for urban growth. A possible solution for this is to integrate CA with other models like Markov Chain and frequency ratio models.

Liu et al. [2] uses a Cellular Automata model integrated with Landscape Expansion Index (LEI) in the paper. This paper proposes a novel method that incorporates the Lanscape Expansion Index (LEI) into the Cellular Automata model to simulate urban growth. A case study using this method was performed in Dongguan, China and overall yielded a better performance over traditional methods. LEI-CA can assist with better understanding urban population evolution (specifically classifying urban growth types) and the motivations behind this growth. The models use GIS and remote sensing data (for LEI). Based on the results from this paper it is evident that the CA model with LEI can address the problem of CA not being able to simulate the outlying growth type. The model has been successively applied to simulate urban growth in Dongguan and from this study it was found that this approach improves simulation accuracy by 13.8%

Rienow's [3] work combined cellular automaton SLEUTH model with support vector machines as a way of combining information for urban growth prediction. In addition to the variables in classic SLEUTH rules, they combine ecological and socioeconomic information to guide predictions. This can be extended to other machine learning models as well, but this paper provides a good overview

of metrics and tradeoffs when doing comparative analysis on the performance of different urban growth simulations. As for the result, they compared SLEUTH with and without SVM and BLR (binomial logistic regression)

Li and Yeh's [4] work concerns making transition rules for CA models using data mining. They explore using techniques like decision trees to make comprehensible rules for cell transitions in urban growth models. The main desire for this is interpretability–it's often easier for urban planners to interpret explicit transition rules (eg if road < 3 and slope < 6 degrees, convert to urban development) than matrices or equations that are used in other methods. Additionally, the use of data mining reduces the bottleneck on domain expertise

Tripathy et al. [5] used Artificial neural network (ANN) urban growth model to provide local policymakers and urban planners with an instrument that could help them analyze various future development scenarios and take the right decisions going forward. To offer a user-friendly assessment of commonly used urban growth models, Zhang et al. [6] selected six urban growth models in five software packages to compare, their findings can help users decide which of the six urban growth models suits them.

In a study conducted by Liu [7], the expansion of urban built-up land in Hefei from 2015 to 2040 was predicted under three different urban growth scenarios: historical growth, urban planning, and land suitability growth. The SLEUTH (Slope, Land use, Exclusion, Urban, Transportation, and Hill-shade) model was used in conjunction with simulation techniques to analyze historical data from 2000, 2005, 2010, and 2015. The study aimed to understand the historical expansion patterns and constraints of the urban area and determine how future urban growth would expand. It was found that land suitability growth would help form a compact urban space and avoid excessive protection of farmland and ecological land.

Wei (2023) [8] constructed a patch-based cellular automaton (CA) model to simulate the dynamic and complex growth process of Urban Underground Space (UUS), subject to the ecological constraints generated by the agent-based land allocation optimization model. The model utilized an Urban Underground Space vector dataset that included parameters such as distance, slope, elevation, average rainfall, average temperature, population, and GDP. The study concluded that the growth rate of UUS development is sustainable, but the scale is significantly lower when ecological constraints are present compared to when they are not.

3.3 Distinction from Previous Studies

The innovation for this project lies in the novel approach to the transition rules used within Cellular Automata for urban development. This innovative strategy treats varying intensities of urban development distinctly, acknowledging the diverse dynamics and requirements of low-intensity, medium-intensity, and high-intensity urban areas. The different intensities of urban sprawl are especially important in this study because it is modeling a section of urban Atlanta (which can be defined largely by medium and high-intensity urban areas). By incorporating specific transition rules for each intensity level, we aim to capture the complexity of urban growth and simulate the unique characteristics of different urban environments.

Another innovation includes hyperparameter tuning which we use to improve our CA model.

4 System Under Investigation (SUI) and Data Descriptions

4.1 System Under Investigation

In this project, we conducted a comprehensive study on the urban growth in the city of Atlanta, as depicted by the orange line in Figure 2.

Adhering to Tobler's second law of geography, which states that the the external phenomenon of interest impacts the dynamics within a geographic area [9], we chose to encompass not only the city of Atlanta itself but also its adjacent areas, as depicted by the red line in Figure 2.

By considering the minimum bounding box of the Atlanta boundary, we accounted for the influence of the neighboring areas on the urban growth patterns within Atlanta. The area within our scope is 67578.7 hectares.

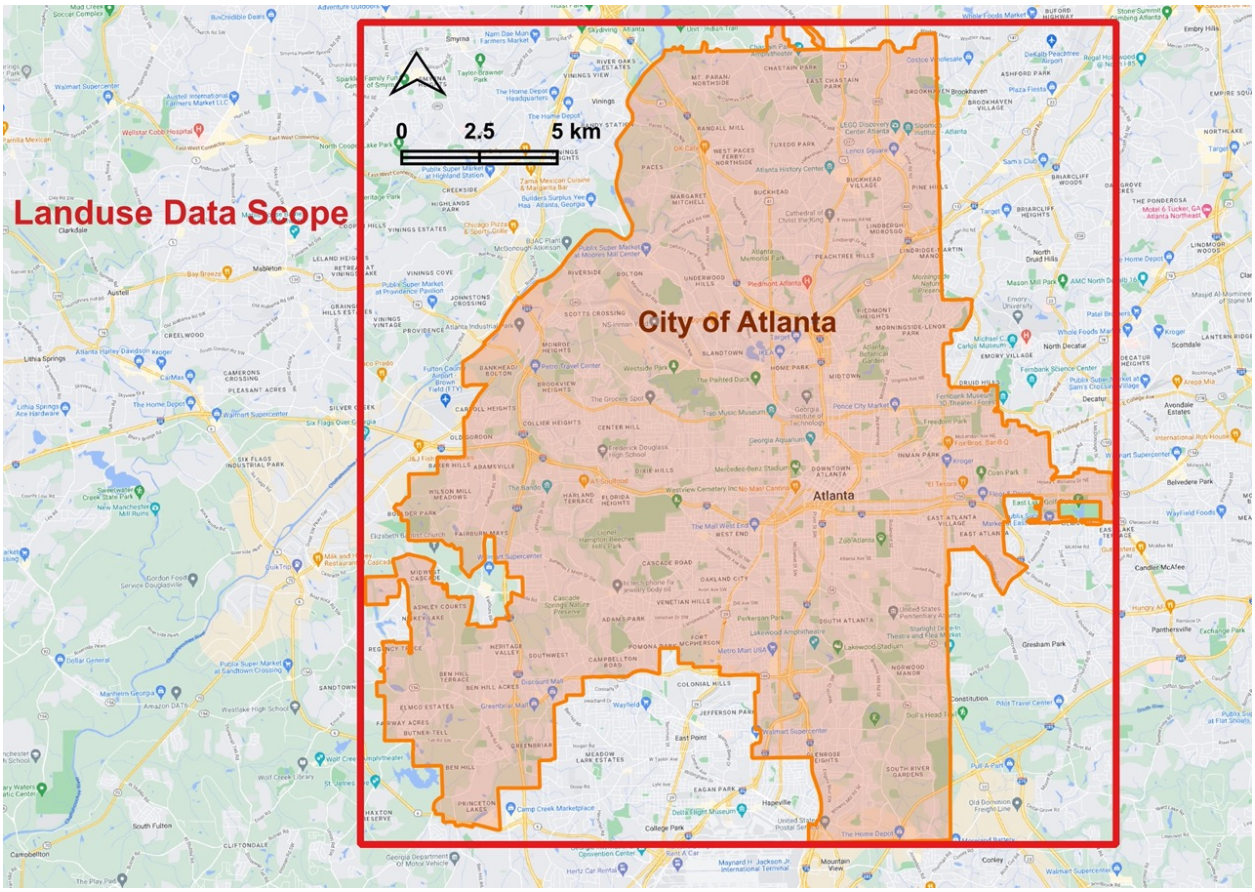

Figure 2: Area under investigation

4.2 Data

For our analysis, we used land cover data from the years 2011, 2013, 2016, 2019, and 2021 obtained from the NLCD (National Land Cover Database) provided by the U.S. Geological Survey (USGS). The land cover map product offers a 30-meter spatial resolution and follows a 16-class legend based on a modified Anderson Level II classification system.

To prepare the data for model building, testing, and validation, we followed the following data processing steps:

1. Unification of Coordinate Reference System (CRS): We converted all the land cover data to a standardized CRS, specifically the WGS84 - Universal Transverse Mercator (UTM) system. This conversion ensured that the unit of measurement was in meters, facilitating accurate analysis by assigning coordinates to locations on the Earth's surface.

2. Extraction of Atlanta Area: We extracted the land cover data corresponding to the city of Atlanta by using the minimum bounding box of the Atlanta boundary. This step allowed us to focus specifically on the urban growth patterns within Atlanta.

3. Land cover re-classification: The original NLCD dataset contains a 16-class legend. We have reclassified them into 6 class legends based on cell states.

4. Format Conversion: The land cover data was exported from the geo-spatial database to GEOTIFF file format, which is widely used for storing raster data.

5. Conversion to Array: Each pixel in the raster data represents a specific land cover type or cell state, which aligns with the urban growth types we aimed to model and simulate. We read the raster data as a two-dimensional array, enabling efficient manipulation and simulation. Each pixel represents an area of 32 square meters in the real world.

6. Splitting the data into modeling, testing, and validation sets, we utilized the land cover data from 2016 as the initial data for simulation. The data from 2019 was used for validation, where we compared the results with the simulated outcomes. Additionally, the land cover data from 2011, 2013, and 2021 were employed for testing purposes.

Figure 3 is the processed data for the simulation input. It is the processed land use raster for the year 2019 in Atlanta.

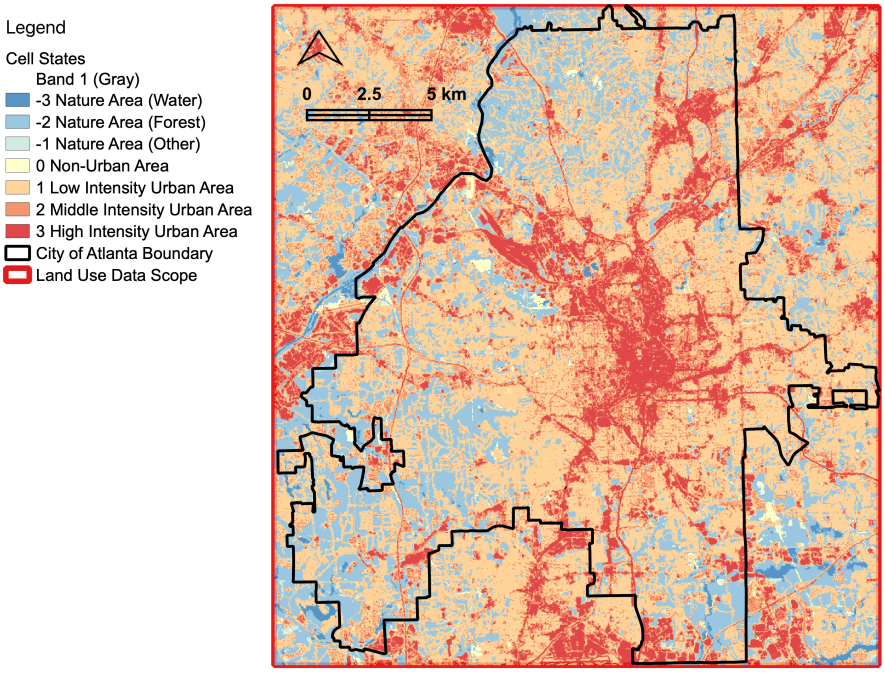

Figure 3: Processed land use data for 2019

5 Conceptual Model

5.1 Cell States

Cell states are defined as follows: non-urban (0), low-intensity urban (1), medium-intensity urban (2), high-intensity urban (3), nature-other (-1), nature-forest (-2), nature-water (-3).

Figure 4 illustrates the spatial distribution of cell states in urban growth from year 2019.

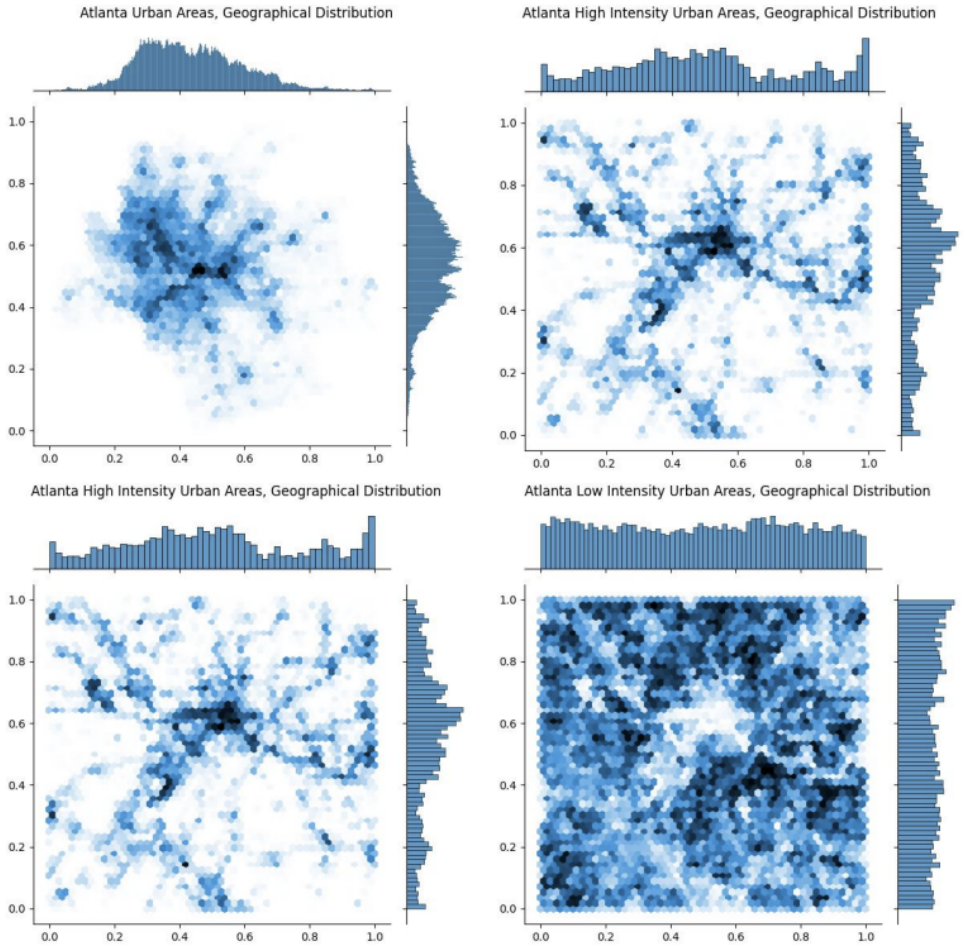

Figure 4: Spatial distribution of cell states for 2019

5.2 Rules

Our CA model's transition rules are informed by urban planning policies and theories related to urban growth. These rules dictate how each cell changes state over time, thereby simulating the process of urban development. We choose to use Moore Neighbors based on literature review, because the majority of CA models for urban growth are carried out using Moore neighbors. Transition rules are as follow:

1. Non-Urban to Low-Intensity Urban Transition Rule:

A cell in the 'non-urban (0)' state transitions to the 'low-intensity urban (1)' state if at least `rule1_trans1(5)`) of its 8 neighboring cells are in the 'low-intensity urban (1)' state. Reflecting

the tendency for urbanization to occur when a significant portion of nearby cells is already in a state of low-intensity urban development

2. Low Intensity Urban to Medium-Intensity Urban Transition Rule:

A cell in the 'low-intensity urban (1)' state transitions to the 'medium-intensity urban' state if at least `rule2_trans1(5)`) of its 8 neighboring cells are in the 'medium-intensity urban (2)' state, or `rule2_trans2(4)`) of its 8 neighboring cells are in the 'high-intensity urban (3)' state.

3. Medium-Intensity Urban to High-Intensity Urban Transition Rule:

A cell in the 'medium-intensity urban (2)' state transitions to the 'high-intensity urban (3)' state if at least `rule3_trans(5)`) of its 8 neighboring cells are in the 'high-intensity urban (3)' state. This transition rule reflects the propensity for areas with moderate development to evolve into high-intensity urban zones when surrounded by a predominantly high-intensity urban environment.

4. Environmental Rule:

A cell in the 'non-urban' state will not transition to the 'urban' state if at least `rule4_trans(3)`) of its 8 neighboring cells are classified as 'water' or 'vegetation'. A cell in the 'water' state cannot be transitioned to another state, because the city of Atlanta's water bodies are protected by the EPA.

5. Forest Conservation Rule:

A cell in the 'non-urban (0)' state will not transition to any urban state if it is classified as a 'nature-forest (-2)' cell. This rule safeguards areas designated as nature-forests, preventing their conversion into urban zones.

5.3 Parameters and Parameter Adjustment Methodology

The parameters are p_1, p_2, p_3, p_4.

p_1 is the possibility for a cell in the 'non-urban (0)' state transitions to the 'low-intensity urban (1)' state, in rule 1.

p_2 is the possibility for a cell in the 'low-intensity urban (1)' state transitions to the 'medium-intensity urban (2)' state, in rule 2.

p_3 is the possibility for a cell in the 'medium-intensity urban (2)' state transitions to the 'high-intensity urban (3)' state, in rule 3.

p_4 is the possibility for a cell in the 'non-urban (0)' state transitions to the 'low-intensity urban (1)' state, in rule 4 and rule 5. The default parameters (inside of each rule, i.e., `rule2_trans1(5)`) above were set based on domain knowledge and historical data, but without hyperparameter tuning this can yield low simulation accuracy. Optuna is an automated hyperparameter optimization framework that uses Bayesian optimization, tree Parzen Estimator (TPE), and other algorithms to search for the optimal combination of hyperparameters. Optuna has many advantages, it's algorithms can efficiently navigate the hyperparameter space, focusing on more promising areas, which leads to faster convergence to optimal solutions. Hence, Optuna was used to adjust different parameter combinations to optimize our result.

The steps to use Optuna were as follows:

1. Define the Objective Function

Create an objective function that Optuna will optimize. In our model, this function returns the validation value indicating the performance with the given hyperparameters.

2. Create a Study

A study in Optuna represents the entire optimization process. We created a study to maximize the objective function.

3. Run the Optimization

Use the "optimize" method of the study object to find the best hyperparameters, the number of trials carried out was 150.

4. Analyze the Results

After running, the Optuna-dashboard tool was used to visualize the optimization process and results.

5. Use the Best Parameters

The best parameters identified by Optuna were used to train our model on the remaining years.

5.4 Timestep

Based on previous CA simulator for urban growth and reviews by Aburus et al.[1], a basic timestep was set for one year. At each time step, the rules are applied to update the state of each cell. The final part of the model is to validate and calibrate the CA model and adjust the parameters.

6 Simulation/ Simulator/ Simulation Model

[ ]:
```
import os, math
import numpy as np
from osgeo import gdal # updated
from copy import deepcopy
from PIL import Image
import optuna

# Defining function to read raster file and return array and datasource
def readraster(file):
    dataSource = gdal.Open(file)
    band = dataSource.GetRasterBand(1)
    band = band.ReadAsArray()
    return(dataSource, band)
```

6.1 Simulation Functions

[ ]:
```
'''
A: input array
B: output array
k: size of neighborhood (k=1 gets a moore neighborhood)
rule: function (numpy array, float) -> float

For each element a_ij in A, sets B[i,j] to the output
of rule evaluated on k-neighborhood around a_ij and the value of a_ij
```

```
'''
def conv(A, B, k=1, rule=None):
    M, N = A.shape[0], A.shape[1]
    for i in range(k, M-k+1):
        if i % (M//10) == 0:
            print(i)
        for j in range(k, N-k+1):
            sub = A[i-k:i+k+1, j-k:j+k+1]
            B[i,j] = rule(sub, A[i,j])

'''
A: input array
B: output array
k: size of neighborhood (k=1 gets a moore neighborhood)
rule: function (numpy array, float) -> float

For each element a_ij in A, sets B[i,j] to the output
of rule evaluated on k-neighborhood around a_ij and the value of a_ij
'''
def conv_full(A,B, k=1, rule=None):
    M, N = A.shape[0], A.shape[1]
    for i in range(M):
        for j in range(N):
            xmin = max(0, i-k)
            xmax = min(M, i+k+1)
            ymin = max(0,j-k)
            ymax = min(N, j+k+1)
            sub = A[xmin:xmax, ymin:ymax]
            z = (ymax-ymin)*(xmax-xmin)
            point = (i,j)
            B[i,j] = rule(sub, A[i,j])

'''
helper function to use conv or conv_full
'''
def apply_rule(A, B, type='reg', k=1, rule=None):
    if type == 'reg':
        conv(A, B, k=k, rule=rule)
    elif type == 'full':
        conv_full(A, B, k=k, rule=rule)
```

6.2 Implement on CA Rules

[ ]:
```
'''

*************** RULE 1 ***************
```

```
Non-Urban -> Urban transition rule:
if a cell is non-urban and 5 or more of its neighbors are low intensity urban,
transition that cell to low-intensity urban

sub: (moore) neighborhood of pixel; 3X3 numpy array
val: value of pixel at center of the moore neighborhood (eg 3 for high intensity␣
 ↪urban)
'''
def rule1(sub, val):
    if val == 0:
        sub1 = np.where(sub==1, 1, 0)
        if sub1.sum() >= rule_parameter["rule1_trans"]:    #default originally␣
 ↪set 5
            return 1

'''

************** RULE 2 ***************

Low Intensity Urban -> Medium Intensity Urban transition rule
if a cell is low intensity urban and 5 or more its neighbors are medium␣
 ↪intensity urban
OR 4 or more of its neighbors are high intensity urban, transition that cell to
medium-intensity urban

sub: (moore) neighborhood of pixel; 3X3 numpy array
val: value of pixel at center of the moore neighborhood (eg 3 for high intensity␣
 ↪urban)
'''
def rule2(sub, val):
    if val == 1:
        sub2 = np.where(sub==2, 1, 0)
        if sub2.sum() - 1 >= rule_parameter["rule2_trans1"]:  #default␣
 ↪originally set 5
            return 2
        else:
            sub3 = np.where(sub==3, 1, 0)
            if sub3.sum() - 1 >= rule_parameter["rule2_trans2"]:  #default␣
 ↪originally set 4
                return 2
    else:
        return rule1(sub, val)

'''

************** RULE 3 ***************
```

```
Medium Intensity Urban -> High Intensity Urban transition rule
if a cell is medium intensity urban and 5 or more its neighbors are high␣
 ↪intensity urban,
transition that cell to high-intensity urban

sub: (moore) neighborhood of pixel; 3X3 numpy array
val: value of pixel at center of the moore neighborhood (eg 3 for high intensity␣
 ↪urban)
'''
def rule3(sub, val):
    if val == 2:
        sub3 = np.where(sub==3, 1, 0)
        if sub3.sum() >= rule_parameter["rule3_trans"]:  #default originally set␣
 ↪5
            return 3
    else:
        return rule2(sub, val)

'''

*************** RULE 4 AND 5 ***************

sub: (moore) neighborhood of pixel; 3X3 numpy array
val: value of pixel at center of the moore neighborhood (eg 3 for high intensity␣
 ↪urban)
'''
def rule45(sub, val):
    sub46 = np.where(sub<=-2, 1, 0)
    if val==-3 or val==-2:
        return val
    elif val==0 & sub46.sum()>=rule_parameter["rule4_trans"]:  #default␣
 ↪originally set 3
      return val
    else:
        return rule3(sub, val)

'''
Applies growth rules (in order).
Change 'v = rule3(sub, val)' to whichever rule is highest
'''
def growth_rules(sub, val):
    v = rule45(sub, val)
    return val if v is None else v
```

6.3 Initial Simulation

[ ]:
```
# Load Data
ds11, arr11 = readraster('2011.tif')
ds13, arr13 = readraster('2013.tif')
ds16, arr16 = readraster('2016.tif')
ds19, arr19 = readraster('2019.tif')
ds21, arr21 = readraster('2021.tif')
```

[ ]:
```
(arr19==arr21).all()
```

[ ]:
```
False
```

[ ]:
```
'''
Takes base data, actual (ground truth) data, and predicted data, then iterates␣
 ↪pixel
by pixel and computes accuracy metrics. Here:

True Positive: a transition that was predicted and actually occurred (at the␣
 ↪same pixel location)
False Positive: a transition that was predicted, but did not actually occur␣
 ↪(pixel value stayed the same between base and actual data)
True Negative: where no transition was predicted, and no transition happened␣
 ↪(pixel value at [i,j] the same in base, actual, and predicted files)
False Negative: no transition was predicted, but one occurred (pixel value at␣
 ↪[i,j] is same in base and predicted data, but value is higher in actual)

Actual Transition: a transition from a lower intensity urban state to a higher␣
 ↪intensity urban state
    between base and actual data
Predicted Transition: a transition from a lower intensity urban state to a␣
 ↪higher intensity urban state
    between base and predicted data

base: original data before running simulation (eg 2016 data)
actual: true data for a later date (eg 2021)
predicted: predicted data (eg after running simulation on 2016 data)
'''
def compare(base, actual, predicted):
    M, N = base.shape[0], base.shape[1]
    compdict = {'Actual Transitions - High Intensity Urban':0,
                'Actual Transitions - Medium Intensity Urban':0,
                'Actual Transitions - Low Intensity Urban':0,
                'Predicted Transitions - High Intensity Urban':0,
                'Predicted Transitions - Medium Intensity Urban':0,
                'Predicted Transitions - Low Intensity Urban':0,
                'True Positive - High Intensity':0,
                'True Negative - High Intensity':0,
                'False Positive - High Intensity':0,
```

```
                'False Negative - High Intensity':0,
                'True Positive - Medium Intensity':0,
                'True Negative - Medium Intensity':0,
                'False Positive - Medium Intensity':0,
                'False Negative - Medium Intensity':0,
                'True Positive - Low Intensity':0,
                'True Negative - Low Intensity':0,
                'False Positive - Low Intensity':0,
                'False Negative - Low Intensity':0,
                'Actual Equals Predicted':0}
    # iterate pixel by pixel
    for i in range(M):
        for j in range(N):
            if actual[i,j] == 3 and base[i,j] < 3: # if actual is high intensity␣
↪and base was less than high intensity
                compdict['Actual Transitions - High Intensity Urban'] += 1 #␣
↪count as an actual transition to high intensity
                if predicted[i,j] == 3 and base[i,j] < 3: # if a high intensity␣
↪transition was also predicted in the same location [i,j]
                    compdict['True Positive - High Intensity'] += 1 # count as a␣
↪true positive high intensiy prediction
                else:
                    compdict['False Negative - High Intensity'] += 1 #␣
↪otherwise, no transition was predicted, so count as a false negative

            elif actual[i,j] == 2 and base[i,j] < 2: # logic the same as above,␣
↪but for medium intensity
                compdict['Actual Transitions - Medium Intensity Urban'] += 1
                if predicted[i,j] == 3 and base[i,j] < 3:
                    compdict['True Positive - Medium Intensity'] += 1
                else:
                    compdict['False Negative - Medium Intensity'] += 1

            elif actual[i,j] == 1 and base [i,j] < 1: # logic the same as above,␣
↪but for low intensity
                compdict['Actual Transitions - Low Intensity Urban'] += 1
                if predicted[i,j] == 3 and base[i,j] < 3:
                    compdict['True Positive - Low Intensity'] += 1
                else:
                    compdict['False Negative - Low Intensity'] += 1

            if predicted[i,j] == 3 and base[i,j] < 3: # if a high intensity␣
↪transition was predicted
                compdict['Predicted Transitions - High Intensity Urban'] += 1
                if not (actual[i,j] == 3 and base[i,j] < 3): # and a high␣
↪intensity transition didn't actually occur
```

```
                    compdict['False Positive - High Intensity'] += 1 # count as␣
↪a false positive for high intensity predictions

            elif predicted[i,j] == 2 and base[i,j] < 2: # same logic, but for␣
↪medium intensity
                compdict['Predicted Transitions - Medium Intensity Urban'] += 1
                if not (actual[i,j] == 3 and base[i,j] < 3):
                    compdict['False Positive - Medium Intensity'] += 1

            elif predicted[i,j] == 1 and base [i,j] < 1: # same logic but for␣
↪low intensity
                compdict['Predicted Transitions - Low Intensity Urban'] += 1
                if not (actual[i,j] == 3 and base[i,j] < 3):
                    compdict['False Positive - Low Intensity'] += 1

            if actual[i,j] == base[i,j]: # if no transition took place
                if actual[i,j] == 3: # if no transition was predicted either
                    pass#compdict['No Change - High Intensity'] += 1
                    if predicted[i,j] == base [i,j]:
                        compdict['True Negative - High Intensity'] += 1 # count␣
↪as a true negative
                if actual[i,j] == 2:
                    pass#compdict['No Change - Medium Intensity'] += 1
                    if predicted[i,j] == base [i,j]:
                        compdict['True Negative - Medium Intensity'] += 1
                if actual[i,j] == 1:
                    pass#compdict['No Change - Low Intensity'] += 1
                    if predicted[i,j] == base [i,j]:
                        compdict['True Negative - Low Intensity'] += 1

            if actual[i,j] == predicted[i,j]:
                compdict['Actual Equals Predicted'] += 1

    compdict['Percent Same'] = compdict['Actual Equals Predicted']/(M*N)
    compdict['True Positive - Total'] = compdict['True Positive - High␣
↪Intensity'] + compdict['True Positive - Medium Intensity'] + compdict['True␣
↪Positive - Low Intensity']
    compdict ['True Negative - Total'] = compdict['True Negative - High␣
↪Intensity'] + compdict['True Negative - Medium Intensity'] + compdict['True␣
↪Negative - Low Intensity']
    compdict['False Positive - Total'] = compdict['False Positive - High␣
↪Intensity'] + compdict['False Positive - Medium Intensity'] + compdict['False␣
↪Positive - Low Intensity']
    compdict['False Negative - Total'] = compdict['False Negative - High␣
↪Intensity'] + compdict['False Negative - Medium Intensity'] + compdict['False␣
↪Negative - Low Intensity']
```

```
    return compdict
```

[ ]:
```
set(arr19.flatten()), set(arr19.flatten())
```

[ ]:
```
({-3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0},
 {-3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0})
```

[ ]:
```
landtype_mapping = {'Non-Urban':0,
                    'Low-Intensity Urban':1,
                    'Medium-Intensity Urban':2}
```

6.3.2 Visualize Simulation Result

[ ]:
```
arr = arr19+3
```

[ ]:
```
best=study.best_params
rule1_trans = best["rule1_trans"]
rule2_trans1 = best["rule2_trans1"]
rule2_trans2 = best["rule2_trans2"]
rule3_trans = best["rule3_trans"]
rule4_trans = best["rule4_trans"]
global rule_parameter
rule_parameter={"rule1_trans": rule1_trans, "rule2_trans1": rule2_trans1,␣
↪"rule2_trans2": rule2_trans2, "rule3_trans": rule3_trans, "rule4_trans":␣
↪rule4_trans}
```

[ ]:
```
arrpred = arr16.copy()
time_steps = 3  #16 19
for _ in range(time_steps):
  apply_rule(arrpred, arrpred, 'reg', 1, growth_rules)
```

[ ]:
```
arr16.shape, arr19.shape, arr21.shape
```

[ ]:
```
((835, 767), (842, 772), (842, 772))
```

[ ]:
```
arrpred
```

[ ]:
```
array([[ 1.,  1.,  1., ...,  2.,  2.,  2.],
       [ 1.,  1.,  1., ...,  1.,  2.,  2.],
       [-2.,  1.,  1., ...,  2.,  2.,  1.],
       ...,
       [-2., -2.,  2., ..., -3., -3., -3.],
       [-2., -2., -2., ..., -3., -3., -3.],
       [-2., -2., -2., ..., -2., -3., -3.]], dtype=float32)
```

[ ]:
```
ap = arrpred+3
```

[ ]:
```
img = Image.fromarray(np.uint8(ap/ap.max() * 255) , 'L')
img
```

Initial area before applying changes from CA model.

[ ]:

Visualize Changes: Red is a change to high intensity; green is a change to medium intensity; blue is a change to low intensity

[ ]:
```
g1 = (arr19==0)*(arrpred==1)
g2 = (arr19==1)*(arrpred==2)
g3 = (arr19==2)*(arrpred==3)
```

[ ]:
```
r = 255*(arrpred+3)/6
g = 255*(arrpred+3)/6
b = 255*(arrpred+3)/6

r = r*(1-g1)*(1-g2)*(1-g3)
g = g*(1-g1)*(1-g2)*(1-g3)
b = b*(1-g1)*(1-g2)*(1-g3)

r += g3*255
#g += g3*255

#r += 55*g2
g += 255*g2

#r += 75*g1
#g += 160*g1
b += 255*g1
```

[ ]:
```
Image.fromarray(np.array([r,g,b]).astype(np.uint8).transpose(1,2,0))

# Red is a change to high intensity; green is a change to medium intensity; blue␣
 ↪is a change to low intensity
```

[ ]:

7 Experimental Results and Validation

7.1 Verification

To ensure the reliability of our model and simulator, we conducted thorough unit tests to examine each rule function. Additionally, we performed a comprehensive review of open-source code from previous CA models of urban growth[1]. This comparison allowed us to ensure that the classes and functions involved in the simulation process were correctly implemented.

[ ]:
```
import unittest
import numpy as np

class TestRule1(unittest.TestCase):
        def test_case_1(self):
                sub = np.array([0, 1, 1, 1, 1, 1, 0, 0])
                val = 0
                expected_output = 1
                actual_output = rule1(sub, val)
                self.assertEqual(actual_output, expected_output)

        def test_case_2(self):
                sub = np.array([0, 0, 0, 1, 1, 1, 0, 0])
                val = 0
                expected_output = None
                actual_output = rule2(sub, val)
                self.assertEqual(actual_output, expected_output)

        def test_case_3(self):
                sub = np.array([3, 3, 3, 3, 3, 0, -1, -2])
                val = 2
                expected_output = 3
                actual_output = rule3(sub, val)
                self.assertEqual(actual_output, expected_output)

        def test_case_45(self):
                sub = np.array([3, 2, 1, 0, -3, -2, 0, 0])
                val = -3
                expected_output = -3
                actual_output = rule45(sub, val)
                self.assertEqual(actual_output, expected_output)
```

[ ]:
```
unittest.main(argv=[''], verbosity=2, exit=False)
```

7.2 Validation

We utilized two performance indicators to evaluate the simulation: Overall Accuracy (OA) and Figure of Merit (FOM).

OA measures the percentage of correct simulations and is a common indicator used in simulations. FOM assesses the consistency between the simulated results and the observed map, providing a specific evaluation of the model's urban growth performance. FOM is widely used in urban Cellular Automata (CA) models [10]. In a recent publication[10], the OA was approximately 95%, while the FOM reached around 40%. It is important to note that a higher FOM indicates better performance, but even the best result obtained in this study is relatively low.

In our simulator, the best OA after hyper parameter tuning is 62.4%, and the best FOM is 13.4%. We also implemented face validation in our simulator to ensure the face validity of the simulation progress. At each timestep, which represents one year of simulation, we included a callback function that visually displays the changes in pixels. This face validation step adds an additional layer of confidence and credibility to our research findings.

7.2.1 Overall Accuracy (OA) and Figure of Merit (FOM)

[ ]:
```
# Validation: Figure of Merit(FOM)
def FOM_high_intensity(compdict):
        return compdict['True Positive - High Intensity']/(compdict['True␣
 ↪Positive - High Intensity'] + compdict['False Positive - High Intensity'] +␣
 ↪compdict['False Negative - High Intensity'])

def FOM_mid(compdict):
        return compdict['True Positive - Medium Intensity']/(compdict['True␣
 ↪Positive - Medium Intensity'] + compdict['False Positive - Medium Intensity']␣
 ↪+ compdict['False Negative - Medium Intensity'])

def FOM_low(compdict):
        return compdict['True Positive - Low Intensity']/(compdict['True␣
 ↪Positive - Low Intensity'] + compdict['False Positive - Low Intensity'] +␣
 ↪compdict['False Negative - Low Intensity'])

# Overall Accuracy Validation:(TP+TN)/(TP+TN+FP+FN)
def OA(compdict):
        return (compdict['True Positive - Total'] + compdict['True Negative -␣
 ↪Total'])/(compdict['True Positive - Total'] + compdict['True Negative -␣
 ↪Total'] + compdict['False Positive - Total'] + compdict['False Negative -␣
 ↪Total'])

# Overall Accuracy Validation:(TP+TN)/(TP+TN+FP+FN)
# def OA(compdict):
#         return (compdict['Predicted Transitions - High Intensity Urban']+␣
 ↪compdict['Predicted Transitions - Medium Intensity Urban'] +␣
 ↪compdict['Predicted Transitions - Low Intensity Urban'])/(compdict['True␣
 ↪Positive - Total'])/ (compdict['Actual Transitions - High Intensity Urban'] +␣
 ↪compdict['Actual Transitions - Medium Intensity Urban'] + compdict['Actual␣
 ↪Transitions - Low Intensity Urban'])
```

```
# True Positive: a transition that was predicted and actually occurred (at the␣
 ↪same pixel location)
# False Positive: a transition that was predicted, but did not actually occur␣
 ↪(pixel value stayed the same between base and actual data)
# True Negative: where no transition was predicted, and no transition happened␣
 ↪(pixel value at [i,j] the same in base, actual, and predicted files)
# False Negative: no transition was predicted, but one occurred (pixel value at␣
 ↪[i,j] is same in base and predicted data, but value is higher in actual)
# b: correctly simulated urban(1,2,3 states)   tp
# a: observed urban that we simulated wrongly..   fn
# c: observed non-urban that we simulated as urban  fp
```

[ ]:
```
OA(compdict=cd)
```

[ ]:
```
0.6239027587439671
```

[ ]:
```
FOM_high_intensity(compdict=cd)
```

[ ]:
```
0.1337791962271055
```

7.3 Hyper parameter Adjustment

[ ]:
```
def objective(trail):
  rule1_trans = trail.suggest_int('rule1_trans', 1, 8,step=1)
  rule2_trans1 = trail.suggest_int('rule2_trans1', 1, 8,step=1)
  rule2_trans2 = trail.suggest_int('rule2_trans2', 1, 8,step=1)
  rule3_trans = trail.suggest_int('rule3_trans', 1, 8,step=1)
  rule4_trans = trail.suggest_int('rule4_trans', 1, 8,step=1)
  global rule_parameter
  rule_parameter={"rule1_trans": rule1_trans, "rule2_trans1": rule2_trans1,␣
 ↪"rule2_trans2": rule2_trans2, "rule3_trans": rule3_trans, "rule4_trans":␣
 ↪rule4_trans}
  arrpred = arr16.copy()
  year_num_simulations = 3   # 2019-2016=3
  for _ in range(year_num_simulations):   # (can change the input; notes)
    apply_rule(arrpred, arrpred, 'reg', 1, growth_rules)
    num_cells_changed = np.sum(arrpred != arr16)
    print(f"round {_+1} simulation done, {num_cells_changed} cells changed")

  cd=compare(arr16, arr19, arrpred)
  optimize=OA(compdict=cd)
  return optimize

study = optuna.
 ↪create_study(study_name="trans_rule_parameter_tunning1204_1642",direction='maximize',␣
 ↪storage="sqlite:///db.sqlite3",load_if_exists=True)      # add␣
 ↪"load_if_exists=True" or change the study name when you run this code again
```

```
study.optimize(objective, n_trials=20)     #  n_trials can change eg:80  150
print(study.best_params)
print(study.best_trial)
print(study.best_trial.value)
print("use this command to show result online: optuna-dashboard sqlite:///db.
 ↪sqlite3")
```

Visualize the parameter optimization result by optuna locally. (steps are as follows)

[ ]:
```
# optuna-dashboard is a web-based tool that needs to run on a local server and␣
 ↪be accessed through a browser.
# The Colab environment does not support running a local server and accessing it␣
 ↪within the same environment.
# However, you can use Optuna for optimization in Colab, download the results␣
 ↪database file to your local machine,
# and then run optuna-dashboard on your local computer by following these steps:

# 1. Optimize with Optuna in Colab  (has finished)

# 2. Download the Database File to Your Local Machine
from google.colab import files
files.download('db.sqlite3')

# 3. Install optuna-dashboard on Your Local Machine
# On your local machine, ensure that optuna-dashboard is installed. It can be␣
 ↪installed via pip:
# pip install optuna-dashboard

# 4. Run optuna-dashboard Locally
# Run optuna-dashboard locally using the command line, pointing to the database␣
 ↪file you just downloaded:
# optuna-dashboard sqlite:///path/to/db.sqlite3


# 5. Access optuna-dashboard Through a Browser
# optuna-dashboard will start a web server locally. Access it in your browser␣
 ↪using the URL provided in the command line (usually http://localhost:8080).
# This way, you can view and analyze your Optuna optimization results in your␣
 ↪local browser.
```

Figure 5 illustrates the process of hyperparameter adjustment, showcasing the overall accuracy among 150 trials. This visualization was generated from the local dashboard, utilizing the code cell above.

We find that the most crucial parameter in model is p_2, the possibility for a cell in the 'low-intensity urban (1)' state transitions to the 'medium-intensity urban (2)' state. Figure 6 shows that the importance of parameter in each rules.

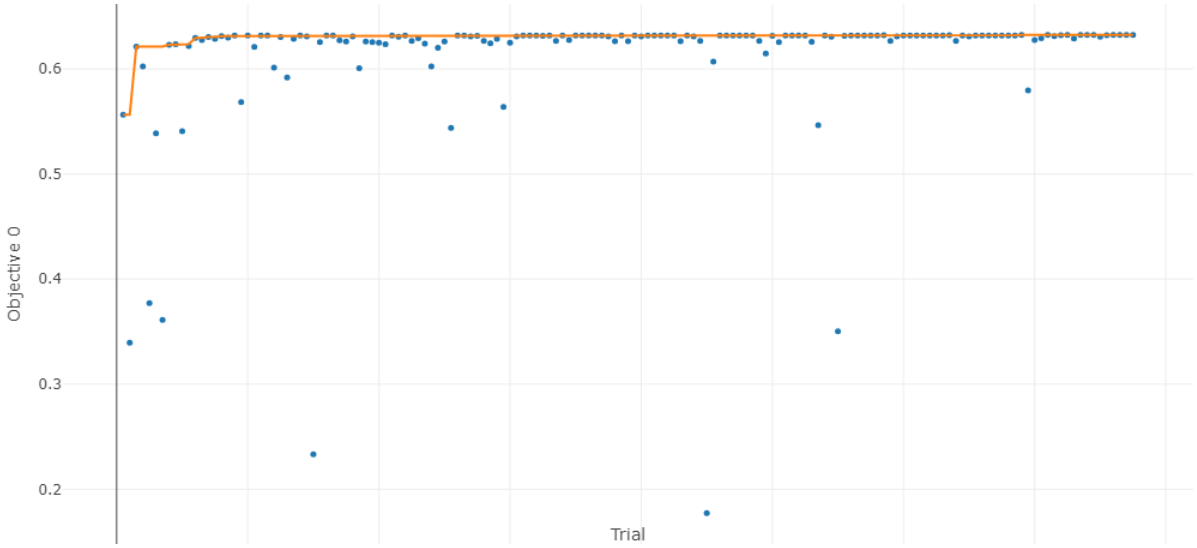

Figure 5: Hyperparameter Adjustment Trials

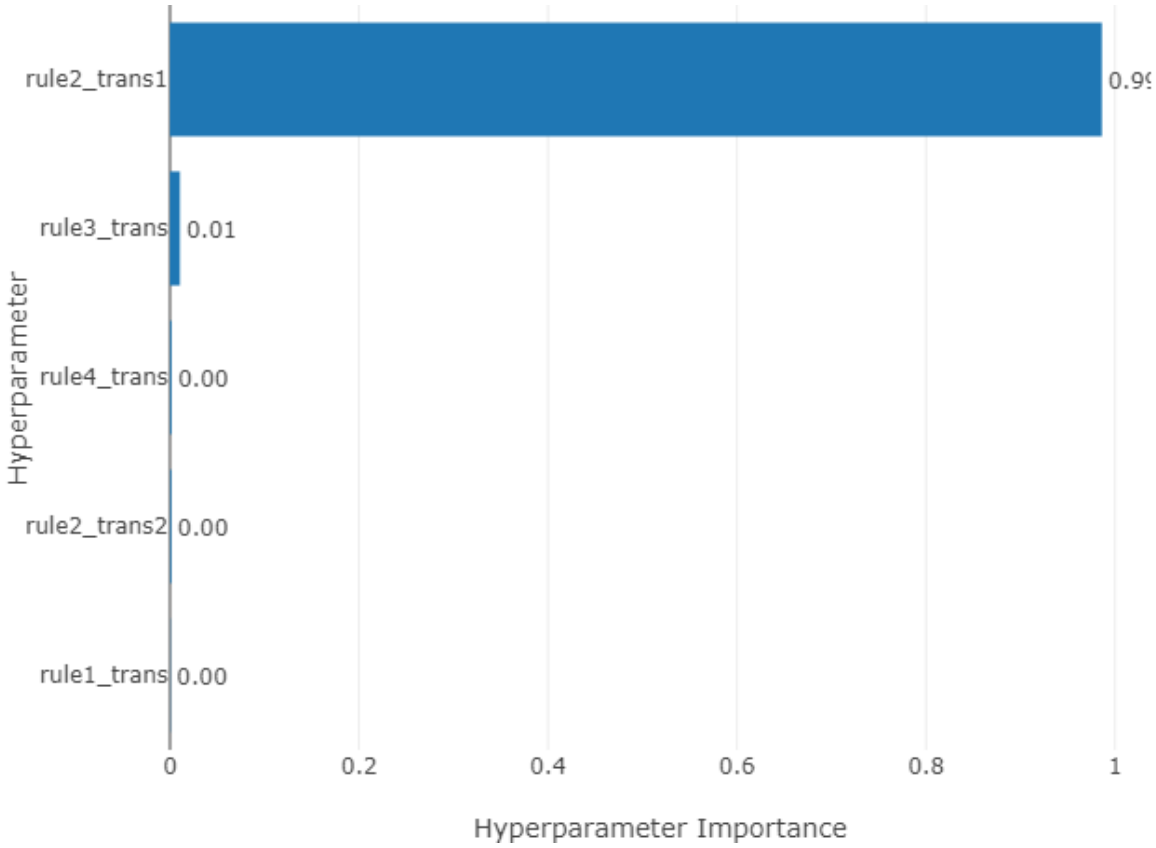

Figure 6: Visualization of Hyperparameter Importance

8 Conclusion

8.1 Findings and Project Summary

This project presents the modeling and simulation process of using CA for urban growth. Utilizing real-world land use data from the city of Atlanta, the simulator demonstrates relatively accurate

results. Through the analysis and tuning of hyper parameters, we have identified that Rule 2 (Low Intensity Urban to Medium Intensity Urban transition rule) plays a crucial role in the urban growth patterns observed in Atlanta.

This finding aligns perfectly with the urban sprawl situation in Atlanta, where low-intensity areas outside urban cores are expanding rapidly. Our project provides a quantified view of the extent of urban sprawl: if no policy changes are made, the CA model predicts the emergence of approximately 13.25 hectares of new medium intensity urban areas per year.

The potential consequences of urban sprawl include impacts on species and ecosystems, as well as increased costs for urban infrastructure [11]. Therefore, this simulator and the information it provides can be shared with urban planners and various stakeholders to facilitate the use of digital twins in considering new policy and planning strategies.

8.2 Limitations

There is an inconsistency in the NLCD (National Land Cover Database) data. Despite our diligent efforts to process the data, we have encountered variations in the Coordinate Reference System (CRS) of the source land use data across different years. Specifically, the data from 2011 and 2013 utilize the Albert CRS, while the data from 2016-2020 adopt the WGS84 CRS. This discrepancy poses a challenge in achieving pixel-level uniformity in the raster files (.tif), leading to disparities in the margins. Consequently, the accuracy of the validation matrix (Overall accuracy, figure of metric) is compromised. Nonetheless, given that urban growth is considered at the city level and extends beyond the boundaries of the Atlanta area, this approach remains the most suitable choice.

It is important to acknowledge that, due to the limitations, the optimization of parameters does not yield the best precision. However, it is worth noting that the scale at which our analysis operates is not representative of a real city. With each pixel or cell representing an area of approximately 32 square meters, it is understandable that there may be some degree of inaccuracy at the pixel level. In the field of urban planning, where the unit of analysis typically encompasses blocks or buildings spanning at least 720 square meters, such pixel-level imprecision is deemed acceptable. The primary objective of this project is to generate predictions for novel urban policies and facilitate the development of digital twins, enabling decision-makers to visualize land use changes at the city level.

8.3 Future Improvement

In the initial stages of this project, the consideration of integrating a deep learning model into the traditional approach was contemplated. However, due to limitations in terms of time and after extensive discussions with domain experts, the decision was made to develop a traditional CA model. Consequently, potential future enhancements can be achieved through the integration of a deep learning model. Noteworthy advancements in CA models have been observed in recent research, particularly through the utilization of logistic regression or Artificial Neural Networks (ANN) for accurate urban growth prediction[5]. By amalgamating the lucidity of traditional modeling techniques with the capacity of deep learning to handle “wide data,” the prediction of urban growth can be substantially refined.

9 Division of Labor

Member	Division of Labor
DaMarcus D Patterson	Implement simulator of rule 1-3 , unit testing, literature reviews
Zhangding Liu	Implement simulator of rule 4-5 , hyperparameter tuning, literature reviews
Jayda G Ritchie	Model refinement, write document, file adjustment, innovation research, literature reviews
Tingyu Liu	Conceptual model design, data processing, file adjustment, unit testing, literature reviews

10 References

[1]Aburas, M. M., Ho, Y. M., Ramli, M. F., & Ash'aari, Z. H. (2016). The simulation and prediction of spatio-temporal urban growth trends using cellular automata models: A review. International Journal of Applied Earth Observation and Geoinformation, 52, 380-389.

https://doi.org/10.1016/j.jag.2016.07.007

[2]Liu, X., Ma, L., Li, X., Ai, B., Li, S., & He, Z. (2014). Simulating urban growth by integrating landscape expansion index (LEI) and cellular automata. International Journal of Geographical Information Science, 28(1), 148-163.

https://doi.org/10.1080/13658816.2013.831097

[3] Andreas Rienow, Roland Goetzke, Supporting SLEUTH – Enhancing a cellular automaton with support vector machines for urban growth modeling (2015). Computers, Environment and Urban Systems, 49, 66-81. https://doi.org/10.1016/j.compenvurbsys.2014.05.001.

[4] Xia Li & Anthony Gar-On Yeh (2004) Data mining of cellular automata's transition rules, International Journal of Geographical Information Science, 18:8, 723-744, https://doi.org/10.1080/13658810410001705325

[5]Tripathy, P., & Kumar, A. (2019). Monitoring and modelling spatio-temporal urban growth of Delhi using Cellular Automata and geoinformatics. Cities, 90, 52-63. https://doi.org/10.1016/j.cities.2019.01.021

[6]Zhang, Y., Kwan, M., & Yang, J. (2023). A user-friendly assessment of six commonly used urban growth models. Computers, Environment and Urban Systems, 104, 102004. https://doi.org/10.1016/j.compenvurbsys.2023.102004()

[7] Liu, Y., Id, L. L., & Chen, L. (2019). Urban growth simulation in different scenarios using the SLEUTH model: A case study of Hefei. East China.

[8]Wei, L., Guo, D., Chen, Z., Hu, Y., Wu, Y., & Ji, J. (2023). Growth Simulations of Urban Underground Space with Ecological Constraints Using a Patch-Based Cellular Automaton. ISPRS International Journal of Geo-Information, 12(10), 387. MDPI AG. Retrieved from http://dx.doi.org/10.3390/ijgi12100387

[9]Waldo, T. (1999) Linear pycnophylactic reallocation comment on a paper by D. Martin, International Journal of Geographical Information Science, 13:1, 85-90, DOI: 10.1080/136588199241472

[10] Li, X., Zhou, Y. & Chen, W. An improved urban cellular automata model by using the trend-adjusted neighborhood. Ecol Process 9, 28 (2020). https://doi.org/10.1186/s13717-020-00234-9

[11] Terando, A. J., Costanza, J., Belyea, C., Dunn, R. R., McKerrow, A., & Collazo, J. A. (2014). The southern megalopolis: using the past to predict the future of urban sprawl in the Southeast U.S. PloS one, 9(7), e102261. https://doi.org/10.1371/journal.pone.0102261