# REFERENCES


[1] Bhinder, D., & Verma, D. (2023). Literature Review on Change Detection Using Remote Sensing Imagery. Reviewed traditional & ML-based CD, proposed TL & Siamese nets, limited by annotation needs.

[2] Hafner, S., et al. (2023). Continuous Urban Change Detection... Used transformer with TFR & MTI for real-time CD; complex design.

[3] Srivastava, S., & Ahmed, T. (2023). Urban Growth Monitoring via Deep Learning. Used Sentinel-2 (2017–22), NDVI/NDBI, ANN/SVM; high storage demand.

[4] Feng, X., et al. (2022). Urban Area Change via Temporal Texture... Used Sentinel-2 textures + Iocrf; not great for global scale.

[5] Mandicou, B. A., et al. (2023). LULC Change Detection in Richard Toll... Segmentation masks for 8 classes; lacked real-time scaling.

[6] Sawant, S., & Ghosh, J. (2021). LULC Classification Using Sentinel-2... 7-class QGIS- labeled images; accurate but cloud-sensitive.

[7] Neupane, B., et al. (2024). Meta-Analysis of Urban Segmentation... Reviewed 71 papers; flagged imbalance & annotation gaps.

[8] Miller, L., et al. (2022). Satellite Image Time Series Analysis... SITS with DL, SVM, regression; multitask boosted output.

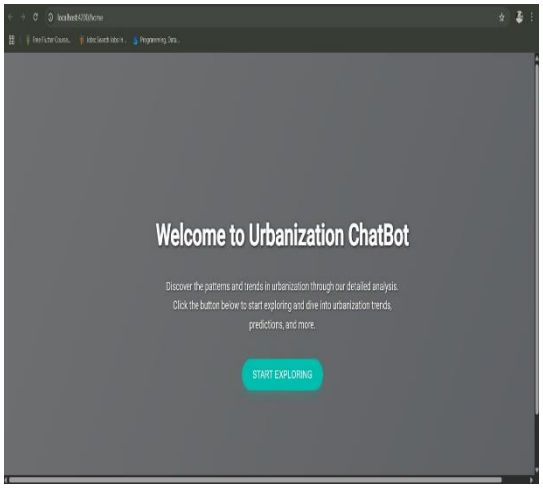
[9] Albert, A., et al. (2020). Urban Pattern Detection at Scale... Used histograms + TL on 6 EU cities; risked overfitting

.

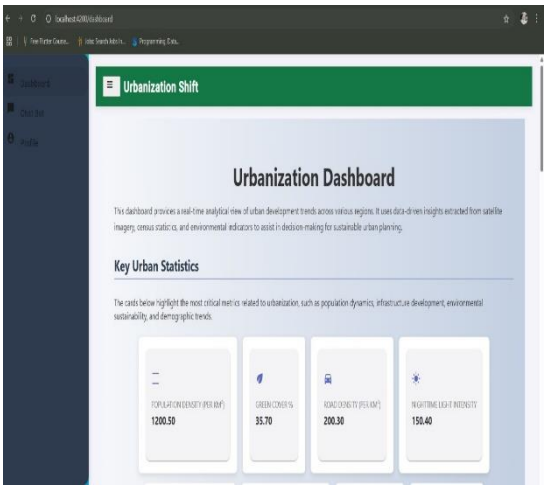
[10] Mehta, A., & Sharma, D. (2023). Change Detection via Remote Sensing... Surveyed CD methods; automation helped, calibration hard.

# APPENDIX A

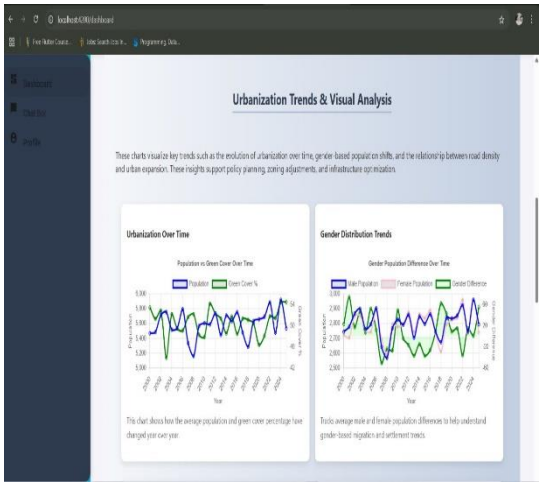
## OUTPUT



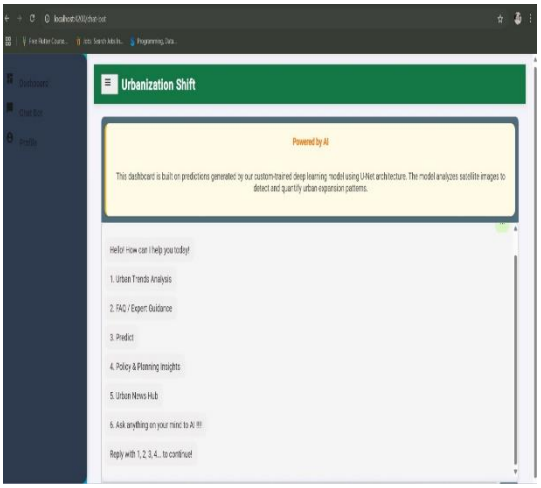

**A1: Home Page of project**



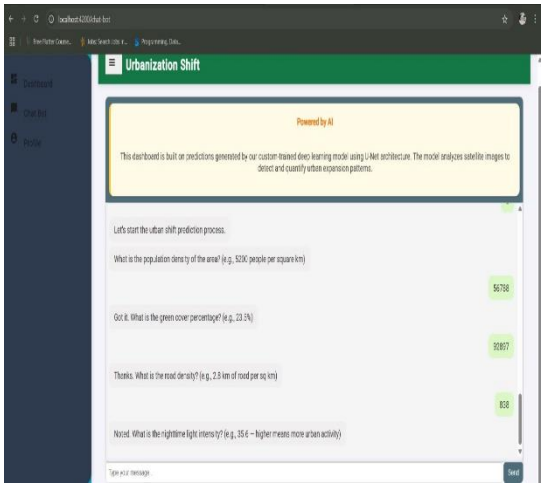

**A2: Urbanshift Dashboard**



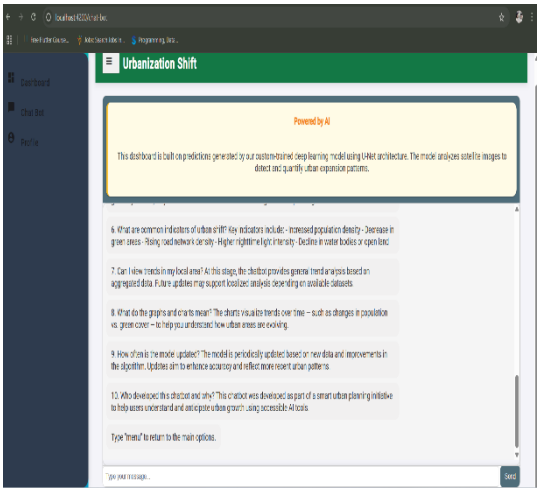

**A3: Plots of trends and Features**




**A4: Urban-shift Chatbot**




**A5: Urban Trend Analysis trough Chatbot**




**A4: FAQ / Expert Guidance**

# APPENDIX B

## CODE IMPLEMENTATION

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from glob import glob
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D,
concatenate, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.metrics import MeanIoU
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import img_to_array, load_img

data_path = "/content/drive/MyDrive/segm"
image_files = glob(os.path.join(data_path, "*.jpg")) + glob(os.path.join(data_path,
"*.png")) + glob(os.path.join(data_path, "*.jpeg"))

IMG_SIZE = (256, 256)
X_train = []
for img_path in image_files:
    img = cv2.imread(img_path)
    if img is not None:
        img = cv2.resize(img, IMG_SIZE)
        X_train.append(img)

X_train = np.array(X_train, dtype=np.uint8)

mask_path = "/content/drive/MyDrive/masks"
mask_files = glob(os.path.join(mask_path, "*.jpg")) + glob(os.path.join(mask_path,
```

```
"*.png")) + glob(os.path.join(mask_path, "*.jpeg"))

Y_train = []
for mask_path in mask_files:
    mask = cv2.imread(mask_path, 0)
    if mask is not None:
        mask = cv2.resize(mask, IMG_SIZE)
        Y_train.append(mask)

Y_train = np.array(Y_train, dtype=np.uint8)
Y_train = np.expand_dims(Y_train, axis=-1)

X_train = X_train.astype("float32") / 255.0
Y_train = Y_train.astype("float32") / 255.0

def unet_model(input_size=(256, 256, 3)):
    inputs = Input(input_size)
    c1 = Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
    c1 = Dropout(0.1)(c1)
    c1 = Conv2D(64, (3, 3), activation='relu', padding='same')(c1)
    p1 = MaxPooling2D((2, 2))(c1)
    c2 = Conv2D(128, (3, 3), activation='relu', padding='same')(p1)
    c2 = Dropout(0.1)(c2)
    c2 = Conv2D(128, (3, 3), activation='relu', padding='same')(c2)
    p2 = MaxPooling2D((2, 2))(c2)
    c3 = Conv2D(256, (3, 3), activation='relu', padding='same')(p2)
    c3 = Dropout(0.2)(c3)
    c3 = Conv2D(256, (3, 3), activation='relu', padding='same')(c3)
     p3 = MaxPooling2D((2, 2))(c3)    p3 = MaxPooling2D((2, 2))(c3)

     c4 = Conv2D(512, (3, 3), activation='relu', padding='same')(p3)
     c4 = Dropout(0.2)(c4)
     c4 = Conv2D(512, (3, 3), activation='relu', padding='same')(c4)
     p4 = MaxPooling2D(pool_size=(2, 2))(c4)
     c5 = Conv2D(1024, (3, 3), activation='relu', padding='same')(p4)
```

```
    c5 = Dropout(0.3)(c5)
    c5 = Conv2D(1024, (3, 3), activation='relu', padding='same')(c5)
    u6 = UpSampling2D((2, 2))(c5)
    u6 = concatenate([u6, c4])
    c6 = Conv2D(512, (3, 3), activation='relu', padding='same')(u6)
    c6 = Dropout(0.2)(c6)
    c6 = Conv2D(512, (3, 3), activation='relu', padding='same')(c6)
    u7 = UpSampling2D((2, 2))(c6)
    u7 = concatenate([u7, c3])
    c7 = Conv2D(256, (3, 3), activation='relu', padding='same')(u7)
    c7 = Dropout(0.2)(c7)
    c7 = Conv2D(256, (3, 3), activation='relu', padding='same')(c7)
    u8 = UpSampling2D((2, 2))(c7)
    u8 = concatenate([u8, c2])
    c8 = Conv2D(128, (3, 3), activation='relu', padding='same')(u8)
    c8 = Dropout(0.1)(c8)
    c8 = Conv2D(128, (3, 3), activation='relu', padding='same')(c8)
    u9 = UpSampling2D((2, 2))(c8)
    u9 = concatenate([u9, c1])
    c9 = Conv2D(64, (3, 3), activation='relu', padding='same')(u9)
    c9 = Dropout(0.1)(c9)
    c9 = Conv2D(64, (3, 3), activation='relu', padding='same')(c9)
    outputs = Conv2D(1, (1, 1), activation='sigmoid')(c9)
    model = Model(inputs=[inputs], outputs=[outputs])
    return model

model = unet_model()
model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])

X_train_split, X_val_split, Y_train_split, Y_val_split = train_test_split(X_train, Y_train,
test_size=0.1, random_state=42)

checkpoint = ModelCheckpoint("best_model.h5", verbose=1, save_best_only=True)

results = model.fit(
```

```
    X_train_split, Y_train_split,
    validation_data=(X_val_split, Y_val_split),
    batch_size=8,
    epochs=25,
    callbacks=[checkpoint]
)

model.load_weights("best_model.h5")

Y_pred = model.predict(X_val_split)

Y_pred_thresholded = Y_pred > 0.5

iou = MeanIoU(num_classes=2)
iou.update_state(Y_val_split, Y_pred_thresholded)
iou_result = iou.result().numpy()

def display_sample(X, Y, prediction, index):
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 3, 1)
    plt.imshow(X[index])
    plt.title('Original Image')
    plt.axis('off')
    plt.subplot(1, 3, 2)
    plt.imshow(Y[index].squeeze(), cmap='gray')
    plt.title('Ground Truth')
    plt.axis('off')
    plt.subplot(1, 3, 3)
    plt.imshow(prediction[index].squeeze(), cmap='gray')
    plt.title('Prediction')
    plt.axis('off')
    plt.show()

for i in range(5):
    display_sample(X_val_split, Y_val_split, Y_pred_thresholded, i)
```

```
def calculate_iou(y_true, y_pred):
    y_true_f = y_true.flatten()
    y_pred_f = y_pred.flatten()
    intersection = np.sum(y_true_f * y_pred_f)
    union = np.sum(y_true_f) + np.sum(y_pred_f) - intersection
    if union == 0:
        return 0.0
    return intersection / union

ious = []
for i in range(len(Y_val_split)):
    iou_val = calculate_iou(Y_val_split[i], Y_pred_thresholded[i])
    ious.append(iou_val)

average_iou = np.mean(ious)

print("Mean IoU:", average_iou)

def predict_image(image_path, model):
    img = load_img(image_path, target_size=IMG_SIZE)
    img_array = img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)
    pred_mask = model.predict(img_array)
    pred_mask = (pred_mask > 0.5).astype(np.uint8)
    return img, pred_mask

test_image_path = image_files[0]
original_img, predicted_mask = predict_image(test_image_path, model)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(original_img)
plt.title('Test Image')
plt.axis('off')
```

```
plt.subplot(1, 2, 2)
plt.imshow(predicted_mask.squeeze(), cmap='gray')
plt.title('Predicted Mask')
plt.axis('off')
plt.show()

def save_predictions(X_data, Y_true, model, folder="predictions", num=5):
    if not os.path.exists(folder):
        os.makedirs(folder)
    preds = model.predict(X_data)
    preds = (preds > 0.5).astype(np.uint8)
    for i in range(num):
        img = (X_data[i] * 255).astype(np.uint8)
        true_mask = (Y_true[i] * 255).astype(np.uint8).squeeze()
        pred_mask = (preds[i] * 255).astype(np.uint8).squeeze()
        cv2.imwrite(os.path.join(folder, f"img_{i}.png"), img)
        cv2.imwrite(os.path.join(folder, f"true_mask_{i}.png"), true_mask)
        cv2.imwrite(os.path.join(folder, f"pred_mask_{i}.png"), pred_mask)

save_predictions(X_val_split, Y_val_split, model, num=10)

test_images = image_files[:5]
for path in test_images:
    img, pred = predict_image(path, model)
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(img)
    plt.axis('off')
    plt.title("Input")
    plt.subplot(1, 2, 2)
    plt.imshow(pred.squeeze(), cmap='gray')
    plt.axis('off')
    plt.title("Prediction")
    plt.show()
```

FastAPI Prediction Route

Handles image upload and prediction from the
trained model. python
import pandas as pd import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,
OneHotEncoder from sklearn.compose import
ColumnTransformer
from sklearn.pipeline import Pipeline import
tensorflow as tf
import joblib

# Load your CSV
df = pd.read_csv("urbanization_data.csv")

# Define the label and features
label_col = "urban_shift" # You must have this in
your CSV numeric_features = [
"population_density", "green_cover_percentage",
"road_density", "nighttime_light_intensity",
"water_bodies_nearby", "Male_Count",
"Female_Count", "Year",
"Slum Area Proportion (%)"
categorical_features = ["Place_Name", "Land Use
Type", "Zoning Code or Urban Tier"]
X = df[numeric_features + categorical_features] y =
df[label_col]

# Create the preprocessor
preprocessor = ColumnTransformer(transformers=[
("num", StandardScaler(), numeric_features),

```
("cat", OneHotEncoder(handle_unknown="ignore",
        sparse_output=False), categorical_features)
])
# Preprocess and split data
X_processed = preprocessor.fit_transform(X)
X_train,        X_test, y_train,        y_test  =
        train_test_split(X_processed, y,
        test_size=0.2, random_state=42)

# Define a simple model model =
tf.keras.Sequential([
tf.keras.layers.Dense(64, activation='relu',
input_shape=(X_processed.shape[1],)),
tf.keras.layers.Dense(32, activation='relu'),
tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam',
loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10,
batch_size=16, validation_data=(X_test, y_test))

# Save the components
model.save("urbanization_model.h5")
joblib.dump(preprocessor.named_transformers_["nu
m"], "scaler.pkl")
joblib.dump(preprocessor.named_transformers_["cat
"], "encoder.pkl")
print("Model, scaler, and encoder saved.")

chatbot.component.html – UI for Upload + Display
<div class="chat-container">
<div class="chat-header">
<mat-card class="ml-model-note">
<h4>Powered by AI</h4>
<p>
```

This dashboard is built on predictions generated by our custom-trained deep learning model using U-Net architecture.
The model analyzes satellite images to detect and quantify urban expansion patterns.
</p>
</mat-card>

<div class="chat-body" #chatBody>
<div *ngFor="let message of messages">

<div [ngClass]="{'user-message': message.sender === 'user', 'bot-message': message.sender === 'bot'}">
<ng-container [ngSwitch]="message.type || 'text'">
<ng-container *ngSwitchCase="'text'">
{{ message.text }}
</ng-container>
<ng-container *ngSwitchCase="'chart'">
<canvas #trendChart width="400" height="200"></canvas>
</ng-container>
</ng-container>

<label for="userMessageInput" class="visually-hidden">Type your message</label>
<input id="userMessageInput" type="text" [(ngModel)]="userInput"

(keyup.enter)="sendMessage()"
[attr.placeholder]="!conversationStarted ? 'Type hi to
begin...' : 'Type your message...'" title="Chat input"
/>
<button (click)="sendMessage()" aria-label="Send
Message">Send</button>

Angular Frontend – Image Upload & Chatbot display

This snippet shows how the Angular component
handles image upload and fetches results. typescript
import { ComponentFixture, TestBed } from
'@angular/core/testing';
import { ChatBotComponent } from './chat-
bot.component';
describe('ChatBotComponent', () => { let component:
ChatBotComponent;
let fixture: ComponentFixture<ChatBotComponent>;
beforeEach(async () => {
await TestBed.configureTestingModule({
imports: [ChatBotComponent]
})
.compileComponents();
fixture =
TestBed.createComponent(ChatBotComponent);
component = fixture.componentInstance;
fixture.detectChanges();
});
it('should create', () => {
expect(component).toBeTruthy();
});
});