```
!nvidia-smi
```

```
Sat Dec  7 04:44:25 2024
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 556.13                 Driver Version: 556.13         CUDA Version: 12.5     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                  Driver-Model | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce RTX 4060 ...  WDDM  |   00000000:01:00.0 Off |                  N/A |
| N/A   41C    P8              1W /   80W |       0MiB /   8188MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|    0   N/A  N/A      5436    C+G   ...8a181b75f1f43801\x64\SysInfoCap.exe      N/A      |
+-----------------------------------------------------------------------------------------+
```

```
#  !export  PATH=/usr/local/cuda/bin:$PATH
#  !export  LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH

```

```
!pip  install  tensorflow

```

```
Requirement already satisfied: tensorflow in c:\python312\lib\site-packages (2.18.0)
Requirement already satisfied: tensorflow-intel==2.18.0 in c:\python312\lib\site-packages (from tensorflow) (2.18.0)
Requirement already satisfied: absl-py>=1.0.0 in c:\python312\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (2.1.0)
Requirement already satisfied: astunparse>=1.6.0 in c:\python312\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in c:\python312\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in c:\python312\lib\site-packages (from tensorflow-intel==2.18.0->tensorflo
Requirement already satisfied: google-pasta>=0.1.1 in c:\python312\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in c:\python312\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in c:\python312\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (3.4.0)
Requirement already satisfied: packaging in c:\users\25122\appdata\roaming\python\python312\site-packages (from tensorflow-intel==2.18.0->tens
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in c:\python312\lib\site-packa
Requirement already satisfied: requests<3,>=2.21.0 in c:\users\25122\appdata\roaming\python\python312\site-packages (from tensorflow-intel==2.
Requirement already satisfied: setuptools in c:\python312\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (75.3.0)
Requirement already satisfied: six>=1.12.0 in c:\users\25122\appdata\roaming\python\python312\site-packages (from tensorflow-intel==2.18.0->te
Requirement already satisfied: termcolor>=1.1.0 in c:\python312\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in c:\python312\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (4.10.0)
Requirement already satisfied: wrapt>=1.11.0 in c:\python312\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (1.16.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in c:\python312\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (1.68.1)
Requirement already satisfied: tensorboard<2.19,>=2.18 in c:\python312\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in c:\python312\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (3.7.0)
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in c:\python312\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (1.26.4)
Requirement already satisfied: h5py>=3.11.0 in c:\python312\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (3.12.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in c:\python312\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (0.4.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in c:\python312\lib\site-packages (from astunparse>=1.6.0->tensorflow-intel==2.18.0->tensorf
Requirement already satisfied: rich in c:\python312\lib\site-packages (from keras>=3.5.0->tensorflow-intel==2.18.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in c:\python312\lib\site-packages (from keras>=3.5.0->tensorflow-intel==2.18.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in c:\python312\lib\site-packages (from keras>=3.5.0->tensorflow-intel==2.18.0->tensorflow) (0.13.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\25122\appdata\roaming\python\python312\site-packages (from requests<3,>=2.
Requirement already satisfied: idna<4,>=2.5 in c:\users\25122\appdata\roaming\python\python312\site-packages (from requests<3,>=2.21.0->tenso
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\25122\appdata\roaming\python\python312\site-packages (from requests<3,>=2.21.0->
Requirement already satisfied: certifi>=2017.4.17 in c:\users\25122\appdata\roaming\python\python312\site-packages (from requests<3,>=2.21.0->
Requirement already satisfied: markdown>=2.6.8 in c:\python312\lib\site-packages (from tensorboard<2.19,>=2.18->tensorflow-intel==2.18.0->tens
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in c:\python312\lib\site-packages (from tensorboard<2.19,>=2.18->tensorf
Requirement already satisfied: werkzeug>=1.0.1 in c:\python312\lib\site-packages (from tensorboard<2.19,>=2.18->tensorflow-intel==2.18.0->tens
Requirement already satisfied: MarkupSafe>=2.1.1 in c:\users\25122\appdata\roaming\python\python312\site-packages (from werkzeug>=1.0.1->tenso
Requirement already satisfied: markdown-it-py>=2.2.0 in c:\python312\lib\site-packages (from rich->keras>=3.5.0->tensorflow-intel==2.18.0->ten
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in c:\users\25122\appdata\roaming\python\python312\site-packages (from rich->keras>=3.5
Requirement already satisfied: mdurl~=0.1 in c:\python312\lib\site-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->tensorflow-intel=

[notice] A new release of pip is available: 24.0 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
import  tensorflow  as  tf
print("Num  GPUs  Available:  ",  len(tf.config.experimental.list_physical_devices('GPU')))

```

```
Num GPUs Available:  0
```

```
#  from  google.colab  import  drive
#  drive.mount('/content/drive')
```

```
import numpy as np
import tensorflow as tf
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import IPython
from sklearn import metrics
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense



import pandas as pd

!pip install h3
import h3
import folium
import branca.colormap as cm

import torch.utils.data
from torch import optim, nn


import pytz

import scipy.optimize
# %matplotlib widget
```

```
Requirement already satisfied: h3 in c:\python312\lib\site-packages (4.1.1)

[notice] A new release of pip is available: 24.0 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
import numpy as np
import pandas as pd
import h3
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras import layers, models, Input
import matplotlib.pyplot as plt


# df = pd.read_csv('/content/all_waybill_info_meituan_0322.csv')
# df.dropna(inplace=True)

```

```
# === 数据加载与初步处理 ===
# 模拟加载数据（替换为您的数据文件路径）
df = pd.read_csv('/mingxuan/Courses/24Fall/CS Capstone/Meituan-INFORMS-TSL-Research-Challenge-main/all_waybill_info_meituan_0322.csv')
# df = pd.read_csv('/content/drive/MyDrive/NYU/24Fall/CS Cap/all_waybill_info_meituan_0322.csv')
# df = pd.read_csv('/content/all_waybill_info_meituan_0322.csv')
df.dropna(inplace=True)
# df.drop(['Zodiac'], axis=1, inplace=True)

df.reset_index(drop=True, inplace=True)
```

```
#用来可视化的df2，不是df
# df2 = df[['is_courier_grabbed','is_prebook','platform_order_time','order_push_time','recipient_lng','recipient_lat']]
df2 = df[['is_courier_grabbed','is_prebook','platform_order_time','order_push_time','estimate_meal_prepare_time','recipient_lng','recipient_l

#去重,只看接受了的订单,只看非预约订单
df2 = df2[df2['is_prebook'] == 0]
df2 = df2[df2['is_courier_grabbed'] == 1]
df2= df2.sort_values(by='platform_order_time')
df2= df2.sort_values(by='order_push_time')

df2.reset_index(drop=True, inplace=True)
df2 = df2.drop(columns=['is_courier_grabbed'])
df2 = df2.drop(columns=['is_prebook'])

# 经纬度转换为浮点数
df2['recipient_lng'] = pd.to_numeric(df2['recipient_lng'], errors='coerce')
df2['recipient_lat'] = pd.to_numeric(df2['recipient_lat'], errors='coerce')
```

```

#转换time系列下单时间为date time
#转换完后数据格式是pandas._libs.tslibs.timestamps.Timestamp
df2['platform_order_time_date'] = pd.to_datetime(df2['platform_order_time'], unit='s')
df2['order_push_time_date'] = pd.to_datetime(df2['order_push_time'], unit='s')
df2['estimate_meal_prepare_time_date'] = pd.to_datetime(df2['estimate_meal_prepare_time'], unit='s')

#时区换成UTC+8hour,不要多次按！每次按都会在原基础上+8!
df2['platform_order_time_date'] = df2['platform_order_time_date'].dt.tz_localize('UTC').dt.tz_convert('Asia/Singapore')
df2['platform_order_time_date'] = df2['platform_order_time_date'].dt.tz_localize(None)
df2['order_push_time_date'] = df2['order_push_time_date'].dt.tz_localize('UTC').dt.tz_convert('Asia/Singapore')
df2['order_push_time_date'] = df2['order_push_time_date'].dt.tz_localize(None)
df2['estimate_meal_prepare_time_date'] = df2['estimate_meal_prepare_time_date'].dt.tz_localize('UTC').dt.tz_convert('Asia/Singapore')
df2['estimate_meal_prepare_time_date'] = df2['estimate_meal_prepare_time_date'].dt.tz_localize(None)

#帮助df2的时间戳数据添加一天之内的特征辅助
day = 24*60*60

df2['Day sin'] = np.sin(df2['platform_order_time'] * (2 * np.pi / day))
df2['Day cos'] = np.cos(df2['platform_order_time'] * (2 * np.pi / day))


```

```
# 转换为 H3 指数
def compute_h3_and_boundaries(row, resolution=9):
    lng = row['recipient_lng'] / 1e6
    lat = row['recipient_lat'] / 1e6
    h3_index = h3.latlng_to_cell(lat, lng, resolution)
    return pd.Series([h3_index])
# 添加 H3 索引列
df2[['H3_Index']] = df2.apply(compute_h3_and_boundaries, axis=1)

```

```
df2[['H3_Index']]
```

	H3_Index
0	89329b5888fffff
1	89329b58d13ffff
2	89329b58c7bffff
3	89329b5aabbffff
4	89329b585a3ffff
...	...
546360	8916cb6f587ffff
546361	89329b58cc7ffff
546362	89329b58823ffff
546363	89329b58943ffff
546364	89329b58893ffff

546365 rows × 1 columns

```
df3=df2.copy()

# 将 H3_Index 从十六进制转为二进制，并提取前19位
df3['H3_Binary'] = df3['H3_Index'].apply(lambda x: bin(int(x, 16))[2:].zfill(64))  # 转为64位二进制
df3['H3_Binary_19'] = df3['H3_Binary'].str[:19]  # 提取前19位

# 将 H3_Index 转为十进制
df3['H3_Decimal'] = df3['H3_Index'].apply(lambda x: int(x, 16))

# 计算每个前19位的出现次数
group_counts = df3['H3_Binary_19'].value_counts()
print(group_counts)
#处理后45bit并转化成十六进制坐标
df3['H3_Binary_Last45'] = df3['H3_Binary'].str[-45:]  # 后45位
df3['H3_Binary_Groups'] = df3['H3_Binary_Last45'].apply(lambda x: [x[i:i+3] for i in range(0, 45, 3)])

# 转换每个3位的二进制为十六进制
df3['H3_Hex_Groups'] = df3['H3_Binary_Groups'].apply(lambda groups: [hex(int(g, 2))[2:] for g in groups])
# 获取出现次数最高的前19位分区作为第一组
top_group = group_counts.index[0]
```

```
df_top_group = df3[df3['H3_Binary_19'] == top_group]

# 将其余的分为第二组
df_other_groups = df3[df3['H3_Binary_19'] != top_group]


```

```
H3_Binary_19
0000100010010011001    506142
0000100010010001011     40223
Name: count, dtype: int64
```

```
df3
```

	platform_order_time	order_push_time	estimate_meal_prepare_time	recipient_lng	recipient_lat	sender_lng	sender_l
0	1665935379	1665935381	1665936161	174547139	45897170	174529930	459058
1	1665935707	1665935711	1665935711	174529156	45880736	174535543	458895
2	1665935814	1665935820	1665936419	174565608	45890492	174553324	458983
3	1665935881	1665935883	1665935883	174571770	45855290	174582434	458618
4	1665935996	1665936000	1665935999	174600135	45852786	174579111	458626
...	...	...	...	...	...	...	
546360	1666627181	1666627186	1666627845	174941500	46045530	174941512	460501
546361	1666627173	1666627189	0	174569259	45879156	174595486	458709
546362	1666627181	1666627194	0	174536704	45905336	174554670	458956
546363	1666627168	1666627198	0	174522175	45909554	174528007	459122
546364	1666627188	1666627199	0	174549457	45891735	174555813	458960

546365 rows × 19 columns

```
import gc

# 删除不需要的变量
del df,df2

# 强制垃圾回收
gc.collect()

```

```
0
```

```

# df3 = df3.drop(columns=['order_push_time','estimate_meal_prepare_time','estimate_meal_prepare_time'])
# df3 = df3.drop(columns=['recipient_lng','recipient_lat','sender_lng','sender_lat'])
# df3 = df3.drop(columns=['order_push_time_date','estimate_meal_prepare_time_date'])
# df3 = df3.drop(columns=['H3_Binary','H3_Binary_19','H3_Decimal','H3_Binary_Last45'])
# df3 = df3.drop(columns=['H3_Binary_Groups','H3_Hex_Groups'])


```

```
df_larger=df_top_group.copy()
df_smaller=df_other_groups.copy()

df_larger.reset_index(drop=True, inplace=True)
df_smaller.reset_index(drop=True, inplace=True)
```

```
6
7
8 df_larger  =  df_larger.drop(columns=['order_push_time','estimate_meal_prepare_time','estimate_meal_prepare_time'])
9 df_larger  =  df_larger.drop(columns=['recipient_lng','recipient_lat','sender_lng','sender_lat'])
10 #  df_larger  =  df_larger.drop(columns=['is_weekend','recipient_lng','recipient_lat','sender_lng','sender_lat'])
11 df_larger  =  df_larger.drop(columns=['order_push_time_date','estimate_meal_prepare_time_date'])
12 df_larger  =  df_larger.drop(columns=['H3_Binary','H3_Binary_19','H3_Decimal','H3_Binary_Last45'])
13 df_larger  =  df_larger.drop(columns=['H3_Binary_Groups','H3_Hex_Groups'])
14
15 df_smaller  =  df_smaller.drop(columns=['order_push_time','estimate_meal_prepare_time','estimate_meal_prepare_time'])
16 df_smaller  =  df_smaller.drop(columns=['recipient_lng','recipient_lat','sender_lng','sender_lat'])
17 #  df_smaller  =  df_smaller.drop(columns=['is_weekend','recipient_lng','recipient_lat','sender_lng','sender_lat'])
18 df_smaller  =  df_smaller.drop(columns=['order_push_time_date','estimate_meal_prepare_time_date'])
19 df_smaller  =  df_smaller.drop(columns=['H3_Binary','H3_Binary_19','H3_Decimal','H3_Binary_Last45'])
20 df_smaller  =  df_smaller.drop(columns=['H3_Binary_Groups','H3_Hex_Groups'])
```

```
1 df_larger  =  df_larger.drop(columns=['Day  sin','Day  cos'])
2
3 df_smaller  =  df_smaller.drop(columns=['Day  sin','Day  cos'])
```

```
1 df_larger
2 #  df_top_group
3
```

	platform_order_time	platform_order_time_date	H3_Index
0	1665935379	2022-10-16 23:49:39	89329b5888fffff
1	1665935707	2022-10-16 23:55:07	89329b58d13ffff
2	1665935814	2022-10-16 23:56:54	89329b58c7bffff
3	1665935881	2022-10-16 23:58:01	89329b5aabbffff
4	1665935996	2022-10-16 23:59:56	89329b585a3ffff
...	...	...	...
506137	1666627167	2022-10-24 23:59:27	89329b5aaabffff
506138	1666627173	2022-10-24 23:59:33	89329b58cc7ffff
506139	1666627181	2022-10-24 23:59:41	89329b58823ffff
506140	1666627168	2022-10-24 23:59:28	89329b58943ffff
506141	1666627188	2022-10-24 23:59:48	89329b58893ffff

506142 rows × 3 columns

```
1
```

```
1 df_smaller
```

	platform_order_time	platform_order_time_date	H3_Index
0	1665936090	2022-10-17 00:01:30	8916cb61ba7ffff
1	1665936177	2022-10-17 00:02:57	8916cb6e677ffff
2	1665936340	2022-10-17 00:05:40	8916cb6e6d3ffff
3	1665936368	2022-10-17 00:06:08	8916cb6e673ffff
4	1665936142	2022-10-17 00:02:22	8916cb6e657ffff
...	...	...	...
40218	1666626921	2022-10-24 23:55:21	8916cb6e673ffff
40219	1666626941	2022-10-24 23:55:41	8916cb6e6cfffff
40220	1666626959	2022-10-24 23:55:59	8916cb61ba7ffff
40221	1666627083	2022-10-24 23:58:03	8916cb6e677ffff
40222	1666627181	2022-10-24 23:59:41	8916cb6f587ffff

40223 rows × 3 columns

```
1
```

```
1 #  提取  H3  索引（假设原始数据已计算出  H3_Index  列）
2 all_h3_indices  =  df_larger['H3_Index'].unique()    #  提取大区LARGER  所有出现的  H3  cells
```

```
3
```

```
1
```

```
all_h3_indices.shape
```

```
(1560,)
```

```
1
```

```
import h3
import numpy as np

def axial_offset_mapping(h3_indices, resolution=9):
    """
    使用轴向偏移将 H3 网格映射到二维矩阵。
    :param h3_indices: H3 网格索引列表
    :param resolution: H3 分辨率
    :return: 二维矩阵、H3 索引到 (i, j) 的映射
    """
    # 选择参考点
    reference_h3 = h3_indices[0]  # 以第一个 H3 网格为参考点
    h3_to_ij_map = {}
    max_i, max_j= h3.cell_to_local_ij(reference_h3, reference_h3)
    min_i, min_j=h3.cell_to_local_ij(reference_h3, reference_h3)
    for h3_idx in h3_indices:
        # 将 H3 转为局部 (i, j) 坐标
        i, j = h3.cell_to_local_ij(reference_h3, h3_idx)
        if i is not None and j is not None:
            h3_to_ij_map[h3_idx] = (i, j)
            max_i, max_j = max(max_i, i), max(max_j, j)
            min_i, min_j = min(min_i, i), min(min_j, j)
        print(min_i,min_j)



    # 初始化矩阵
    rows, cols = max_i - min_i + 1, max_j - min_j + 1
    grid_matrix = np.zeros((rows, cols),dtype=int)
    # grid_matrix = np.zeros((max(rows,cols), max(rows,cols)),dtype=int)
    # a = geek.zeros([2, 2], dtype = int)


    # 调整坐标使最小值为 (0, 0)
    adjusted_map = {h3_idx: (i - min_i, j - min_j) for h3_idx, (i, j) in h3_to_ij_map.items()}
    # adjusted_map = {h3_idx: (i - min_i, j - min_j) for h3_idx, (i, j) in h3_to_ij_map.items()}

    return grid_matrix, adjusted_map


# 调用轴向偏移方法
grid_matrix, h3_to_ij_map = axial_offset_mapping(all_h3_indices)

# 输出结果
# print("H3 to 2D mapping:", h3_to_ij_map)
# print("Grid matrix shape:", grid_matrix.shape)

```

```
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
-993 2893
```

```
grid_matrix
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
```

```
# h3_to_ij_map
```

```
# adjusted_h3_to_ij_map
```

```
# # 验证所有坐标是否在第一象限
# all_coords = list(adjusted_h3_to_ij_map.values())
# print("Minimum i:", min(coord[0] for coord in all_coords))
# print("Minimum j:", min(coord[1] for coord in all_coords))
# print("Maximum i:", max(coord[0] for coord in all_coords))
# print("Maximum j:", max(coord[1] for coord in all_coords))

```

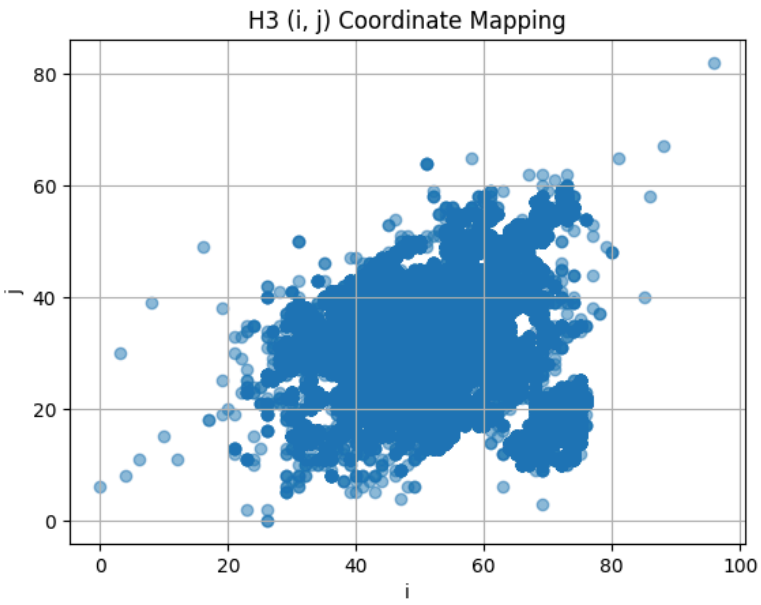
```
def map_h3_to_ij(df, h3_column, ij_map):
    """
    将 DataFrame 中的 H3_Index 转换为 (i, j) 坐标。
    :param df: 包含 H3_Index 的 DataFrame
    :param h3_column: 表示 H3_Index 的列名
    :param ij_map: H3_Index 到 (i, j) 坐标的映射
    :return: 添加了 (i, j) 坐标的 DataFrame
    """
    # 定义映射函数
    def get_ij(h3_idx):
        return ij_map.get(h3_idx, (None, None))  # 默认返回 (None, None) 表示无效 H3_Index

    # 应用映射
    df[['i', 'j']] = df[h3_column].apply(lambda x: pd.Series(get_ij(x)))
```

```
        return df

# 应用到 df_larger
df_larger_with_ij = map_h3_to_ij(df_larger, h3_column='H3_Index', ij_map=h3_to_ij_map)

# 检查结果
print(df_larger_with_ij.head())

```

```
   platform_order_time platform_order_time_date         H3_Index   i   j
0           1665935379      2022-10-16 23:49:39  89329b5888fffff  42  32
1           1665935707      2022-10-16 23:55:07  89329b58d13ffff  42  26
2           1665935814      2022-10-16 23:56:54  89329b58c7bffff  47  35
3           1665935881      2022-10-16 23:58:01  89329b5aabbffff  55  31
4           1665935996      2022-10-16 23:59:56  89329b585a3ffff  61  37
```

```

```

```
# 可视化 i, j 坐标分布
import matplotlib.pyplot as plt

plt.scatter(df_larger_with_ij['i'], df_larger_with_ij['j'], alpha=0.5)
plt.title('H3 (i, j) Coordinate Mapping')
plt.xlabel('i')
plt.ylabel('j')
plt.grid()
plt.show()

```




```

```

```
!free -h    # 查看内存
!nvidia-smi    # 查看GPU显存

```

```
'free' 不是内部或外部命令，也不是可运行的程序
或批处理文件。
ERROR: Option # is not recognized. Please run 'nvidia-smi -h'.
```

```

```

```

```

```
# def preprocess_order_time_h3_with_time_features(df, time_col, interval='10min'):
#         import numpy as np
#         import pandas as pd
#         from itertools import product

#         # # 确保时间列为 datetime 格式 #platform_order_time_date
#         # df[time_col] = pd.to_datetime(df[time_col], unit='s', errors='coerce')

```

```
#         # # 检查转换后的时间列是否有效
#         # if df[time_col].isna().any():
#         #         raise ValueError(f"{time_col} 中存在无效值，无法转换为 datetime！请检查数据。")
#
#         # 添加时间特征
#         day_minutes = 24 * 60    # 一天的分钟数
#
#         df['time_bucket'] = df[time_col].dt.floor(interval)
#         df['Day_sin'] = np.sin((df['time_bucket'].dt.hour * 60 + df['time_bucket'].dt.minute) * (2 * np.pi / day_minutes))
#         df['Day_cos'] = np.cos((df['time_bucket'].dt.hour * 60 + df['time_bucket'].dt.minute) * (2 * np.pi / day_minutes))
#
#         # 聚合订单量
#         grouped = df.groupby(['time_bucket', 'i', 'j']).size().reset_index(name='order_volume')
#
#         # 获取完整的时间段
#         # time_buckets = sorted(grouped['time_bucket'].unique())
#
#         # 获取完整的时间段（从最早到最晚时间桶）
#         full_time_range = pd.date_range(
#                 start=grouped['time_bucket'].min(),
#                 end=grouped['time_bucket'].max(),
#                 freq=interval
#         )
#
#         # 获取所有 i, j 的最大值，构建完整的 (i, j) 组合
#         i_max = grouped['i'].max()
#         j_max = grouped['j'].max()
#
#         # 生成所有可能的 (i, j) 组合
#         size_max = np.max([i_max, j_max])
#         ij_pairs = list(product(range(size_max + 1), range(size_max + 1)))
#
#         # 构造完整索引
#         full_index = pd.MultiIndex.from_product([time_buckets, ij_pairs], names=['time_bucket', 'ij'])
#
#         # 将 (i, j) 映射为元组，便于索引
#         grouped['ij'] = list(zip(grouped['i'], grouped['j']))
#         grouped = grouped.set_index(['time_bucket', 'ij'])['order_volume']
#
#
#
#
#         # 将补全数据转为二维矩阵 (时间步数, 网格数)
#         filled['i'] = filled['ij'].apply(lambda x: x[0])
#         filled['j'] = filled['ij'].apply(lambda x: x[1])
#         grid_data = (
#                 filled.pivot(index='time_bucket', columns='ij', values='order_volume')
#                 .reindex(index=full_time_range, fill_value=0)
#                 .reset_index(drop=True)
#         )
#         # 补全数据，填充缺失的订单量为0
#         filled = grouped.reindex(full_index, fill_value=0).reset_index()
#         # 分离时间特征
#         time_features = (
#                 df[['time_bucket', 'Day_sin', 'Day_cos']]
#                 .drop_duplicates()
#                 .set_index('time_bucket')
#                 .reindex(full_time_range)
#                 .reset_index()
#         )
#
#
#         # 将补全数据转为二维矩阵 (时间步数, 网格数)
#         filled['i'] = filled['ij'].apply(lambda x: x[0])
#         filled['j'] = filled['ij'].apply(lambda x: x[1])
#         grid_data = (
#                 filled.pivot(index='time_bucket', columns='ij', values='order_volume')
#                 .reindex(index=time_buckets, fill_value=0)
#                 .reset_index(drop=True)
#         )
#
#         # 转换时间特征为数组并扩展维度
#         time_features_array = time_features[['Day_sin', 'Day_cos']].to_numpy()
#         time_features_expanded = np.repeat(time_features_array[:, None, :], grid_data.shape[1], axis=1)
#
#         # 转换网格数据为数组
#         grid_data_array = grid_data.to_numpy()
#
#         # 拼接时间特征与网格数据
#         lstm_array = np.concatenate([grid_data_array[:, :, None], time_features_expanded], axis=2)
#
#         return lstm_array
```

```
#          ##############################################
```

```
def preprocess_order_time_h3_with_time_features(df, time_col, interval='10min'):
        import numpy as np
        import pandas as pd
        from itertools import product

        # # 确保时间列为 datetime 格式
        # df[time_col] = pd.to_datetime(df[time_col], unit='s', errors='coerce')

        # # 检查转换后的时间列是否有效
        # if df[time_col].isna().any():
        #         raise ValueError(f"{time_col} 中存在无效值，无法转换为 datetime！请检查数据。")

        # 添加时间桶
        df['time_bucket'] = df[time_col].dt.floor(interval)

        # 聚合订单量
        grouped = df.groupby(['time_bucket', 'i', 'j']).size().reset_index(name='order_volume')

        # 获取完整的时间段（从最早到最晚时间桶）
        full_time_range = pd.date_range(
                start=grouped['time_bucket'].min(),
                end=grouped['time_bucket'].max(),
                freq=interval
        )

        # 获取所有 i, j 的最大值，构建完整的 (i, j) 组合
        i_max = grouped['i'].max()
        j_max = grouped['j'].max()

        # 生成所有可能的 (i, j) 组合
        size_max = np.max([i_max, j_max])
        ij_pairs = list(product(range(size_max + 1), range(size_max + 1)))

        # 构造完整索引
        full_index = pd.MultiIndex.from_product([full_time_range, ij_pairs], names=['time_bucket', 'ij'])

        # 将 (i, j) 映射为元组，便于索引
        grouped['ij'] = list(zip(grouped['i'], grouped['j']))
        grouped = grouped.set_index(['time_bucket', 'ij'])['order_volume']

        # 补全所有时间桶的完整数据，填充缺失的订单量为 0
        filled = grouped.reindex(full_index, fill_value=0).reset_index()
        filled.columns = ['time_bucket', 'ij', 'order_volume']   # 重命名列方便后续操作

        # 分离 i 和 j
        filled['i'] = filled['ij'].apply(lambda x: x[0])
        filled['j'] = filled['ij'].apply(lambda x: x[1])

        # 将补全数据转为二维矩阵 (时间步数, 网格数)
        grid_data = (
                filled.pivot(index='time_bucket', columns='ij', values='order_volume')
                .reindex(index=full_time_range, fill_value=0)
                .reset_index(drop=True)
        )

        # 转换网格数据为数组
        grid_data_array = grid_data.to_numpy()

        # 构造仅包含订单量的 LSTM 输入数组
        lstm_array = np.expand_dims(grid_data_array, axis=-1)

        return lstm_array
```

```
# 应用函数
lstm_array = preprocess_order_time_h3_with_time_features(
        df_larger_with_ij,
        time_col='platform_order_time_date'
)

# 查看结果形状
print("LSTM Array Shape:", lstm_array.shape)
```

```
LSTM Array Shape: (1155, 9409, 1)
```

有slide_window的数据集拆分 且部分分步处理数据

格式转换

```
# 转换为 ConvLSTM 格式，调整数据类型

def reshape_to_convlstm_input(lstm_array, rows, cols):
        """
        将 LSTM array 转换为 ConvLSTM 的输入格式
        :param lstm_array: 原始 LSTM array，形状为 (time_steps, grid, features)
        :param rows: 网格行数
        :param cols: 网格列数
        :return: 重塑后的数据，形状为 (time_steps, rows, cols, features)
        """
        import numpy as np

        grid_size = rows * cols
        if lstm_array.shape[1] != grid_size:
                raise ValueError("LSTM array 的网格数量与指定的 rows x cols 不匹配")

        # 重塑为 ConvLSTM 输入格式
        return lstm_array.reshape(-1, rows, cols, lstm_array.shape[2])

rows, cols = 97, 97
convlstm_input = reshape_to_convlstm_input(lstm_array, rows, cols).astype(np.float32)

```

分批次处理滑动窗口，然后合并

```
def apply_sliding_window_with_targets_in_batches(data, window_size, step=1, batch_size=100):
        """
        分批对时间序列数据应用滑动窗口，同时生成目标值
        :param data: 输入数据，形状为 (time_steps, rows, cols, channels)
        :param window_size: 滑动窗口的时间步数
        :param step: 滑动的步长
        :param batch_size: 每批次生成的窗口数
        :return: 分批生成的滑动窗口数据和目标值
        """
        import numpy as np

        time_steps, rows, cols, channels = data.shape
        num_windows = (time_steps - window_size) // step

        for start in range(0, num_windows, batch_size):
                end = min(start + batch_size, num_windows)
                windows = np.array([
                        data[i: i + window_size]
                        for i in range(start * step, end * step, step)
                ], dtype=np.float32)
                targets = np.array([
                        data[i + window_size, :, :, 0]
                        for i in range(start * step, end * step, step)
                ], dtype=np.float32)
                yield windows, targets

```

```
# 设置窗口参数
window_size = 10
step = 1
batch_size = 128

# 初始化列表存储结果
sliding_windows_list = []
targets_list = []

# 分批生成滑动窗口和目标值
for batch_windows, batch_targets in apply_sliding_window_with_targets_in_batches(convlstm_input, window_size, step, batch_size):
        sliding_windows_list.append(batch_windows)
        targets_list.append(batch_targets)

# 合并结果
```

```
sliding_windows = np.concatenate(sliding_windows_list)
targets = np.concatenate(targets_list)

# 释放中间变量
del sliding_windows_list, targets_list, convlstm_input
import gc
gc.collect()

```

```
2949
```

数据集拆分

```
def split_dataset(windows, targets, train_ratio=0.8):
    """
    划分数据集为训练集和验证集
    :param windows: 滑动窗口数据，形状为 (num_windows, window_size, rows, cols, channels)
    :param targets: 目标值，形状为 (num_windows, rows, cols)
    :param train_ratio: 训练集的比例
    :return: 训练集滑动窗口, 验证集滑动窗口, 训练集目标值, 验证集目标值
    """
    import numpy as np

    # 打乱索引
    indexes = np.arange(windows.shape[0])
    np.random.shuffle(indexes)

    # 划分数据集
    split_index = int(train_ratio * len(indexes))
    train_indexes = indexes[:split_index]
    val_indexes = indexes[split_index:]

    train_windows = windows[train_indexes]
    val_windows = windows[val_indexes]
    train_targets = targets[train_indexes]
    val_targets = targets[val_indexes]

    return train_windows, val_windows, train_targets, val_targets

```

```
# 数据集拆分
train_windows, val_windows, train_targets, val_targets = split_dataset(sliding_windows, targets)

# 释放中间变量
del sliding_windows, targets
gc.collect()

```

```
0
```

分批次归一化，然后合并

```
def normalize_in_batches(data, scaler=None, batch_size=100):
        """
        分批归一化数据
        :param data: 输入数据，形状为 (num_windows, window_size, rows, cols, channels)
        :param scaler: 可选的 sklearn MinMaxScaler 实例
        :param batch_size: 批次大小
        :return: 归一化后的数据
        """
        import numpy as np
        from sklearn.preprocessing import MinMaxScaler

        if scaler is None:
                scaler = MinMaxScaler()

        data_reshaped = data.reshape(-1, data.shape[-1])
        num_samples = data_reshaped.shape[0]
        normalized_data = np.zeros_like(data_reshaped, dtype=np.float32)

        for start in range(0, num_samples, batch_size):
                end = min(start + batch_size, num_samples)
                normalized_data[start:end] = scaler.fit_transform(data_reshaped[start:end])

        return normalized_data.reshape(data.shape), scaler
```

```
# 对训练集归一化
train_normalized, scaler = normalize_in_batches(train_windows)

# 使用相同 scaler 归一化验证集
val_normalized, _ = normalize_in_batches(val_windows, scaler=scaler)

# 释放中间变量
del train_windows, val_windows
gc.collect()
```

```
0
```

```
def normalize_in_batches_optimized(data, scaler=None, batch_size=100):
        import numpy as np
        from sklearn.preprocessing import MinMaxScaler

        if scaler is None:
                scaler = MinMaxScaler()

        data_reshaped = data.reshape(-1, data.shape[-1])
        scaler.fit(data_reshaped)    # 全局拟合 scaler

        num_samples = data_reshaped.shape[0]
        normalized_data = np.zeros_like(data_reshaped, dtype=np.float32)

        for start in range(0, num_samples, batch_size):
                end = min(start + batch_size, num_samples)
                normalized_data[start:end] = scaler.transform(data_reshaped[start:end])

        return normalized_data.reshape(data.shape), scaler
```

构建Tensorflow数据集

```
import tensorflow as tf

batch_size = 16

# 创建训练集和验证集
train_dataset = tf.data.Dataset.from_tensor_slices((train_normalized, train_targets))
train_dataset = train_dataset.batch(batch_size).shuffle(buffer_size=100)

val_dataset = tf.data.Dataset.from_tensor_slices((val_normalized, val_targets))
val_dataset = val_dataset.batch(batch_size)
```

验证

```
print("训练数据形状:", train_normalized.shape)
print("训练目标形状:", train_targets.shape)
print("验证数据形状:", val_normalized.shape)
print("验证目标形状:", val_targets.shape)

```

```
训练数据形状: (916, 10, 97, 97, 1)
训练目标形状: (916, 97, 97)
验证数据形状: (229, 10, 97, 97, 1)
验证目标形状: (229, 97, 97)
```

```
1
```

模型构建

```
# import tensorflow as tf
# import numpy as np
# from tensorflow.keras import Sequential
# from tensorflow.keras.layers import ConvLSTM2D, BatchNormalization, Conv3D

# # 模拟归一化后的训练和验证数据
# # 假设 train_normalized 和 val_normalized 形状为 (num_samples, time_steps, height, width, channels)
# num_samples = 100
# time_steps = 10
# height, width, channels = 97, 97, 3


# # 定义前 20 帧预测后 20 帧的逻辑
# def create_shifted_frames(data, input_length, predict_length):
#     x, y = [], []
#     for i in range(data.shape[1] - input_length - predict_length + 1):
#         x.append(data[:, i:i + input_length, :, :, :])
#         y.append(data[:, i + input_length:i + input_length + predict_length, :, :, :])
#     return np.concatenate(x, axis=0), np.concatenate(y, axis=0)

# input_length = 20
# predict_length = 20

# x_train, y_train = create_shifted_frames(train_normalized, input_length, predict_length)
# x_val, y_val = create_shifted_frames(val_normalized, input_length, predict_length)

# print("x_train shape:", x_train.shape)
# print("y_train shape:", y_train.shape)

# # 构建 tf.data.Dataset
# batch_size = 16
# train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
# train_dataset = train_dataset.shuffle(buffer_size=100).batch(batch_size).prefetch(buffer_size=tf.data.AUTOTUNE)

# val_dataset = tf.data.Dataset.from_tensor_slices((x_val, y_val))
# val_dataset = val_dataset.batch(batch_size).prefetch(buffer_size=tf.data.AUTOTUNE)

# # 构建模型
# model = Sequential([
#     ConvLSTM2D(filters=64, kernel_size=(5, 5), input_shape=(None, 97, 97, 3),
#                        padding='same', return_sequences=True),
#     BatchNormalization(),
#     ConvLSTM2D(filters=64, kernel_size=(3, 3), padding='same', return_sequences=True),
#     BatchNormalization(),
#     ConvLSTM2D(filters=64, kernel_size=(1, 1), padding='same', return_sequences=True),
#     Conv3D(filters=1, kernel_size=(3, 3, 3), activation='sigmoid', padding='same')
# ])

# model.compile(loss='binary_crossentropy', optimizer='adadelta')

# # 打印模型结构
# model.summary()

# # 训练模型
# model.fit(train_dataset, validation_data=val_dataset, epochs=20)

```

加载数据到TensorFlow

```
# batch_size = 16

# train_dataset = tf.data.Dataset.from_tensor_slices((train_normalized, train_targets))
# train_dataset = train_dataset.batch(batch_size).shuffle(buffer_size=100)

# val_dataset = tf.data.Dataset.from_tensor_slices((val_normalized, val_targets))
# val_dataset = val_dataset.batch(batch_size)
```

```
# for sample in train_dataset.take(1):  # 取一个批次的数据
#     print(sample)  # 打印样本
#     print(sample[0].shape)  # 如果是 (features, labels)，打印特征的形状
#     print(sample[1].shape)  # 打印标签的形状
# total_samples = train_dataset.cardinality().numpy()
# print(f"Total samples in train_dataset: {total_samples}")
```

```
# for sample in val_dataset.take(1):  # 取一个批次的数据
#     print(sample)  # 打印样本
#     print(sample[0].shape)  # 如果是 (features, labels)，打印特征的形状
#     print(sample[1].shape)  # 打印标签的形状
# total_samples = val_dataset.cardinality().numpy()
# print(f"Total samples in train_dataset: {total_samples}")
```

```
# def create_shifted_frames(x):
#     """
#     接受一个序列数据的张量，返回前 20 帧作为 x，后 20 帧作为 y。
#     假设输入形状为 (time_steps, height, width, channels)。
#     """
#     x_frames = x[:-1, :, :, :]  # 前 n-1 帧
#     y_frames = x[1:, :, :, :]  # 后 n-1 帧
#     return x_frames, y_frames
```

```
# # 创建训练集和验证集
# batch_size = 16
#
# # 训练数据集
# train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
# train_dataset = train_dataset.shuffle(buffer_size=100).batch(batch_size).prefetch(buffer_size=tf.data.AUTOTUNE)
#
# # 验证数据集
# val_dataset = tf.data.Dataset.from_tensor_slices((x_val, y_val))
# val_dataset = val_dataset.batch(batch_size).prefetch(buffer_size=tf.data.AUTOTUNE)
```

```
# # 构造训练集
# train_dataset = tf.data.Dataset.from_tensor_slices(train_normalized)
# train_dataset = train_dataset.map(
#     lambda x: create_shifted_frames(x)
# )  # 应用 frame 分离逻辑
# train_dataset = train_dataset.batch(batch_size).shuffle(buffer_size=100)
#
# # 构造验证集
# val_dataset = tf.data.Dataset.from_tensor_slices(val_normalized)
# val_dataset = val_dataset.map(
#     lambda x: create_shifted_frames(x)
# )  # 应用 frame 分离逻辑
# val_dataset = val_dataset.batch(batch_size)
```

```
# def create_shifted_frames_batch(x):
#     """
#     处理整个 batch 的帧切分逻辑。
#     假设输入形状为 (batch_size, time_steps, height, width, channels)。
#     """
#     x_frames = x[:, :-1, :, :, :]  # 前 n-1 帧
#     y_frames = x[:, 1:, :, :, :]  # 后 n-1 帧
#     return x_frames, y_frames
#
# # 构造训练集
# train_dataset = tf.data.Dataset.from_tensor_slices(train_normalized)
# train_dataset = train_dataset.batch(batch_size)
# train_dataset = train_dataset.map(
#     lambda x: create_shifted_frames_batch(x)
```

```
# ).shuffle(buffer_size=100)

# # 构造验证集
# val_dataset = tf.data.Dataset.from_tensor_slices(val_normalized)
# val_dataset = val_dataset.batch(batch_size)
# val_dataset = val_dataset.map(
#         lambda x: create_shifted_frames_batch(x)
# )


# # 分离x和y,注意，此时的y是下一帧图像，既最后一个片子，我们用前20帧预测后20帧，既序号0-19
# def create_shifted_frames(data):
#         x = data[:, 0: data.shape[1] - 1, :, :]
#         y = data[:, 1: data.shape[1], :, :]
#         return x, y
# x_train, y_train = create_shifted_frames(train_dataset)
# x_val, y_val = create_shifted_frames(val_dataset)


1
```

convLSTM模型构建

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import ConvLSTM2D, BatchNormalization, Conv2D, Flatten, Dense,Reshape

# def create_convlstm_model(input_shape, output_size):
#         model = Sequential([
#                 ConvLSTM2D(filters=32, kernel_size=(3, 3), padding="same", return_sequences=False, input_shape=input_shape),
#                 BatchNormalization(),
#                 Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding="same"),
#                 Flatten(),
#                 Dense(128, activation='relu'),
#                 Dense(output_size, activation='linear')
#         ])
#         return model

def create_convlstm_model(input_shape,output_size):

        model = Sequential([
                ConvLSTM2D(filters=32, kernel_size=(5, 5), padding="same", return_sequences=False, input_shape=input_shape),
                BatchNormalization(),
                Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding="same"),
                Flatten(),
                Dense(128, activation='relu'),
                Dense(output_size, activation='linear'),    # 全连接层，输出展平的网格
                Reshape((97, 97))    # 调整形状为 (97, 97)
        ])
        return model


# # 展平目标值
# train_targets = train_targets.reshape(train_targets.shape[0], -1)    # (batch_size, 9409)
# val_targets = val_targets.reshape(val_targets.shape[0], -1)               # (batch_size, 9409)


# # 模型构建核心代码，这里我们修改超参数与keras官方超参数一致
# model = Sequential([
#         keras.layers.ConvLSTM2D(filters=64, kernel_size=(5, 5),
#                                 input_shape=(None, 97, 97, 3),
#                                 padding='same', return_sequences=True),
#         keras.layers.BatchNormalization(),
#         keras.layers.ConvLSTM2D(filters=64, kernel_size=(3, 3),
#                                 padding='same', return_sequences=True),
#         keras.layers.BatchNormalization(),
#         keras.layers.ConvLSTM2D(filters=64, kernel_size=(1, 1),
#                                 padding='same', return_sequences=True),
#         keras.layers.Conv3D(filters=1, kernel_size=(3, 3, 3),
#                             activation='sigmoid',
#                             padding='same', data_format='channels_last')
# ])
# model.compile(loss='binary_crossentropy', optimizer='adadelta')
# model.summary()
```

```
# # 保存模型
# model.save('conv_lstm_model.h5')

# # 加载模型
# from tensorflow.keras.models import load_model
# model = load_model('conv_lstm_model.h5')


```

```
1
```

```
input_shape = (10, 97, 97, 1)  # 与你的训练数据形状匹配
# output_size = (97 , 97)  # 每个网格点的目标数量
output_size = 97 * 97  # 每个网格点的目标数量
# model = create_convlstm_model(input_shape)
model = create_convlstm_model(input_shape, output_size)

model.compile(optimizer='adam', loss='mse', metrics=['mae'])
model.summary()

```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv_lstm2d_2 (ConvLSTM2D)	(None, 97, 97, 32)	105,728
batch_normalization_2 (BatchNormalization)	(None, 97, 97, 32)	128
conv2d_2 (Conv2D)	(None, 97, 97, 64)	18,496
flatten_2 (Flatten)	(None, 602176)	0
dense_4 (Dense)	(None, 128)	77,078,656
dense_5 (Dense)	(None, 9409)	1,213,761
reshape_2 (Reshape)	(None, 97, 97)	0

Total params: 78,416,769 (299.14 MB)
Trainable params: 78,416,705 (299.14 MB)
Non-trainable params: 64 (256.00 B)

训练

```
# # 填充时间序列到统一长度（假设最大时间步为 20）
# max_time_steps = 20
# x_train = tf.keras.preprocessing.sequence.pad_sequences(x_train, maxlen=max_time_steps, dtype='float32', padding='post')
# y_train = tf.keras.preprocessing.sequence.pad_sequences(y_train, maxlen=max_time_steps, dtype='float32', padding='post')

```

双击（或按回车键）即可修改

```
epochs = 10
history = model.fit(
        train_dataset,
        # validation_data=val_dataset,
        epochs=epochs,
        verbose=1
)

```

```
Epoch 1/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 103s 2s/step - loss: 0.2338 - mae: 0.1063
Epoch 2/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 94s 2s/step - loss: 0.1583 - mae: 0.0478
Epoch 3/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 92s 2s/step - loss: 0.1769 - mae: 0.0506
Epoch 4/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 97s 2s/step - loss: 0.1465 - mae: 0.0493
Epoch 5/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 94s 2s/step - loss: 0.1592 - mae: 0.0521
Epoch 6/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 95s 2s/step - loss: 0.1462 - mae: 0.0517
Epoch 7/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 92s 2s/step - loss: 0.1457 - mae: 0.0511
Epoch 8/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 90s 2s/step - loss: 0.1479 - mae: 0.0527
Epoch 9/10
```

```
58/58 ━━━━━━━━━━━━━━━━━━━━ 90s 2s/step - loss: 0.1454 - mae: 0.0522
Epoch 10/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 93s 2s/step - loss: 0.1430 - mae: 0.0533
```

```
1
```

验证/评估

```
val_loss, val_mae = model.evaluate(val_dataset)
print(f"Validation Loss: {val_loss}, Validation MAE: {val_mae}")

```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 8s 500ms/step - loss: 0.1645 - mae: 0.0610
Validation Loss: 0.15237420797348022, Validation MAE: 0.05850024148821831
```

```
import numpy as np

# 获取验证集中的数据和真实目标值
val_data = []
val_true = []

for batch in val_dataset:
    inputs, targets = batch
    val_data.append(inputs.numpy())
    val_true.append(targets.numpy())

# 合并所有批次
val_data = np.concatenate(val_data, axis=0)  # 形状 (num_val_windows, time_steps, rows, cols, channels)
val_true = np.concatenate(val_true, axis=0)  # 形状 (num_val_windows, rows, cols)

# 使用模型进行预测
val_predictions = model.predict(val_data)  # 输出形状 (num_val_windows, rows * cols)

# 将预测值重塑为空间形状
val_predictions = val_predictions.reshape(val_true.shape)

```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 8s 970ms/step
```

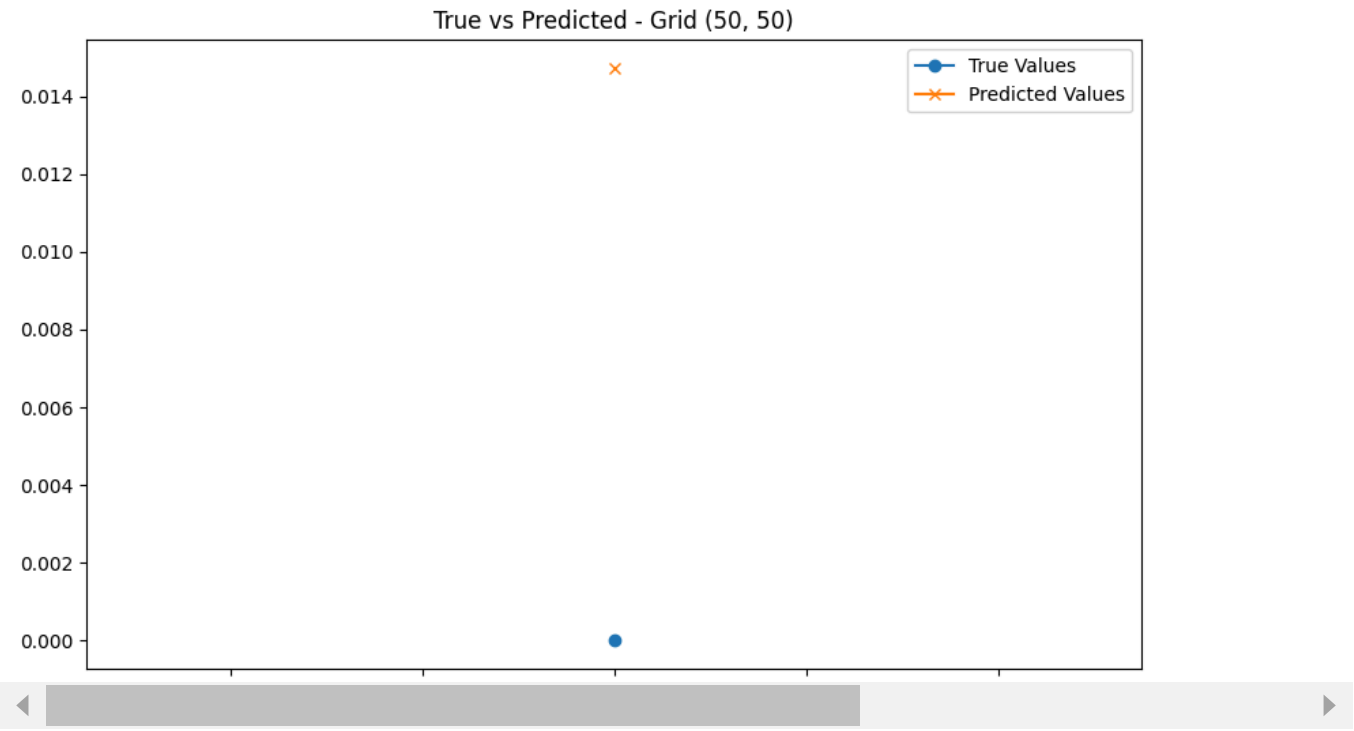
还原：返归一

```
# 反归一化
val_predictions_restored = scaler.inverse_transform(val_predictions_broadcasted)[:, 0]  # 恢复第一个通道的值
val_true_restored = scaler.inverse_transform(val_true_broadcasted)[:, 0]  # 恢复第一个通道的值

# 恢复原始形状
val_predictions_restored = val_predictions_restored.reshape(val_true.shape)
val_true_restored = val_true_restored.reshape(val_true.shape)

```

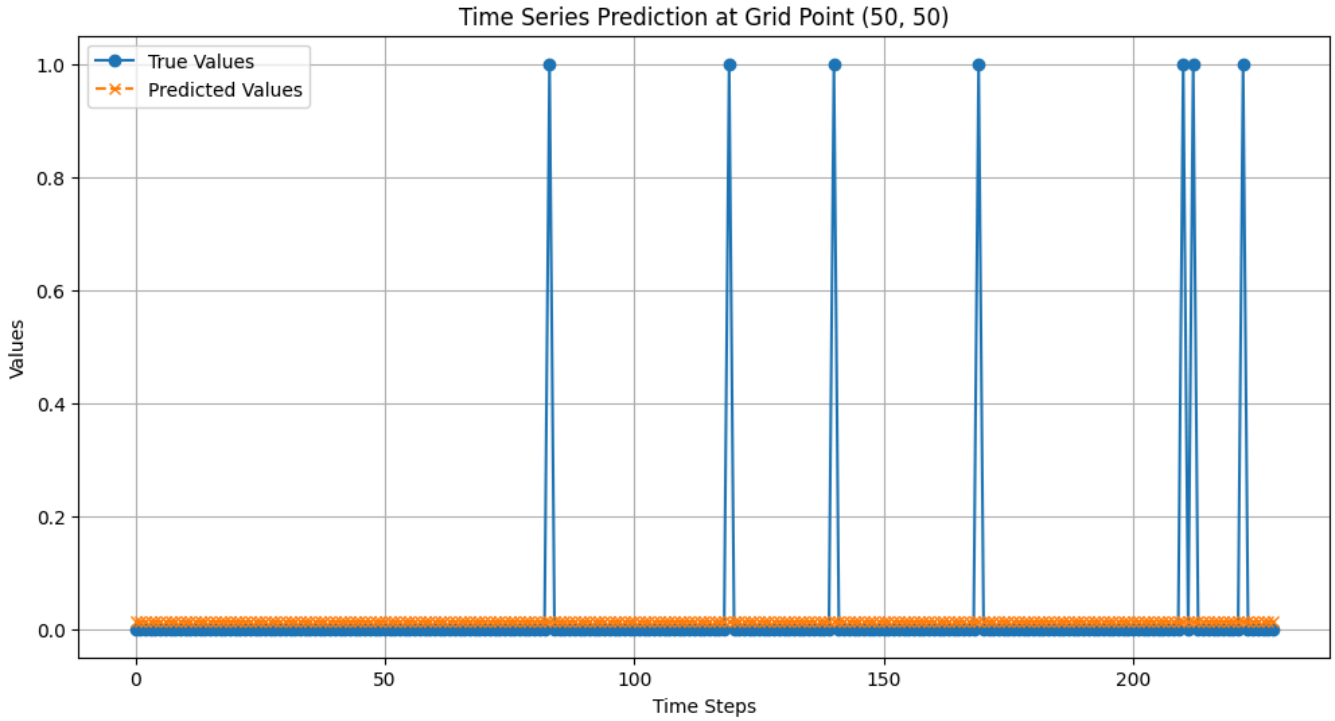
```
print("Restored predictions shape:", val_predictions_restored.shape)
print("Restored true values shape:", val_true_restored.shape)

```

```
Restored predictions shape: (229, 97, 97)
Restored true values shape: (229, 97, 97)
```

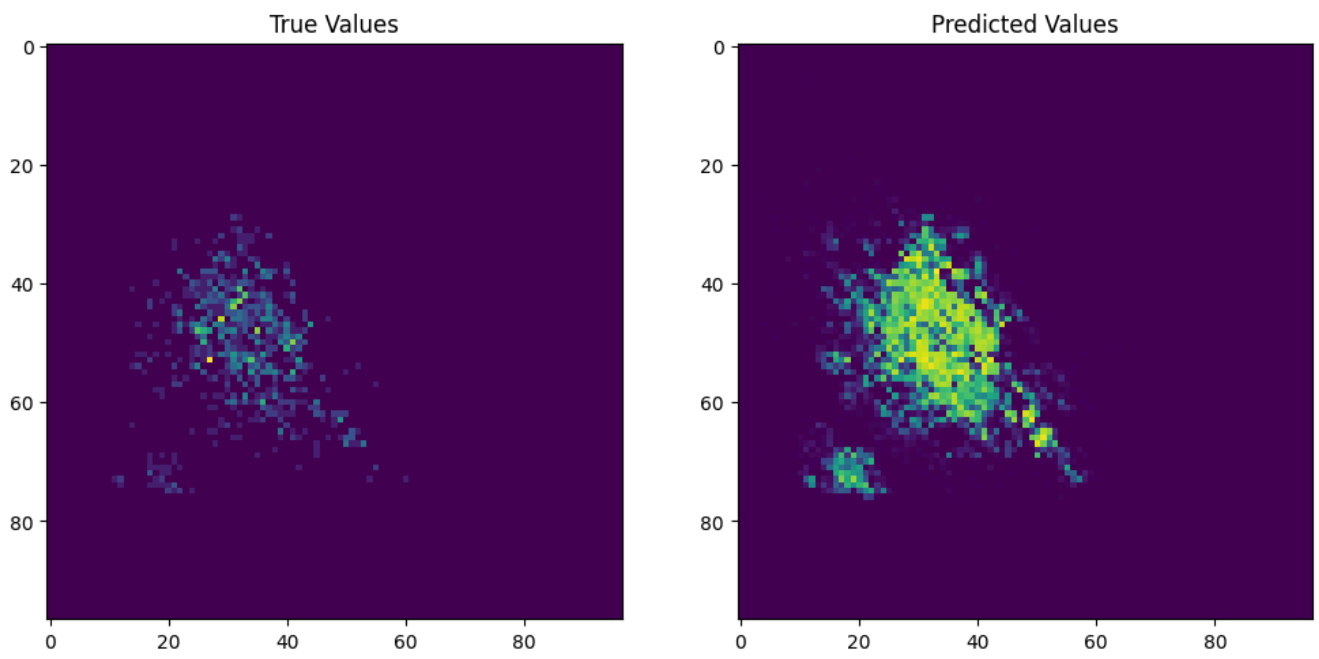
```
# 绘制第一个网格点的预测和真实值对比
grid_point = (50, 50)  # 选择一个网格点
window_idx = 0  # 选择第一个窗口

plt.figure(figsize=(10, 6))
plt.plot(val_true_restored[window_idx, grid_point[0], grid_point[1]], label='True Values', marker='o')
plt.plot(val_predictions_restored[window_idx, grid_point[0], grid_point[1]], label='Predicted Values', marker='x')
plt.legend()
plt.title(f"True vs Predicted - Grid {grid_point}")
plt.show()

```




```
# 选择一个验证窗口和网格点
window_idx = 200  # 第一个滑动窗口
grid_point = (50, 50)  # 网格点 (50, 50)

# 获取真实值和预测值的时间序列
true_series = val_true_restored[:, grid_point[0], grid_point[1]]
pred_series = val_predictions_restored[:, grid_point[0], grid_point[1]]

# 创建时间轴
time_steps = np.arange(len(true_series))

# 绘图
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
plt.plot(time_steps, true_series, label='True Values', marker='o', linestyle='-')
plt.plot(time_steps, pred_series, label='Predicted Values', marker='x', linestyle='--')
plt.xlabel('Time Steps')
plt.ylabel('Values')
plt.title(f'Time Series Prediction at Grid Point {grid_point}')
plt.legend()
plt.grid()
plt.show()

```




```
# 选择一个验证窗口 index
window_idx = 0 # 第一个滑动窗口

# 获取真实值和预测值的空间分布
true_grid = val_true[window_idx]
pred_grid = val_predictions[window_idx]

# 绘制空间分布对比
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# 真实值
axes[0].imshow(true_grid, cmap='viridis')
axes[0].set_title('True Values')

# 预测值
axes[1].imshow(pred_grid, cmap='viridis')
axes[1].set_title('Predicted Values')

plt.show()

```



有slide_window的数据集拆分

转换成ConvLSTM需要的数据形式（batch size暂时不考虑，在处理完滑动窗口期分数据集之后，训练时引入动态batchsize）

```
# def reshape_to_convlstm_input(lstm_array, rows, cols):
#     """
#     将 LSTM array 转换为 ConvLSTM 的输入格式
#     :param lstm_array: 原始 LSTM array，形状为 (time_steps, grid, features)
#     :param rows: 网格行数
#     :param cols: 网格列数
#     :return: 重塑后的数据，形状为 (time_steps, rows, cols, features)
#     """
#     import numpy as np

#     grid_size = rows * cols
#     if lstm_array.shape[1] != grid_size:
#         raise ValueError("LSTM array 的网格数量与指定的 rows x cols 不匹配")

#     # 重塑为 ConvLSTM 输入格式
#     return lstm_array.reshape(-1, rows, cols, lstm_array.shape[2])

```

应用滑动窗口

```
# def apply_sliding_window_with_targets(data, window_size, step=1):
#     """
#     对时间序列数据应用滑动窗口，同时生成目标值
#     :param data: 输入数据，形状为 (time_steps, rows, cols, channels)
#     :param window_size: 滑动窗口的时间步数
#     :param step: 滑动的步长
#     :return: 滑动窗口数据, 目标值
#     """
#     import numpy as np

#     time_steps, rows, cols, channels = data.shape

#     # 修正 num_windows 的计算，确保不会越界
#     num_windows = (time_steps - window_size) // step

#     # 构造滑动窗口和目标值
#     windows = np.array([
#         data[i: i + window_size]
#         for i in range(0, num_windows * step, step)
#     ])
#     targets = np.array([
#         data[i + window_size, :, :, 0]   # 使用第 0 通道的最后一个时间步作为目标值
#         for i in range(0, num_windows * step, step)
#     ])

#     return windows, targets

```

拆分数据集

```
# def split_dataset(windows, targets, train_ratio=0.9):
#     """
#     划分数据集为训练集和验证集
#     :param windows: 滑动窗口数据，形状为 (num_windows, window_size, rows, cols, channels)
#     :param targets: 目标值，形状为 (num_windows, rows, cols)
#     :param train_ratio: 训练集的比例
#     :return: 训练集滑动窗口, 验证集滑动窗口, 训练集目标值, 验证集目标值
#     """
#     import numpy as np

#     # 打乱索引
#     indexes = np.arange(windows.shape[0])
#     np.random.shuffle(indexes)

#     # 划分数据集
#     split_index = int(train_ratio * len(indexes))
#     train_indexes = indexes[:split_index]

```

```
# val_indexes = indexes[split_index:]

# train_windows = windows[train_indexes]
# val_windows = windows[val_indexes]
# train_targets = targets[train_indexes]
# val_targets = targets[val_indexes]

# return train_windows, val_windows, train_targets, val_targets

```

去除周末时间

```
# def remove_weekends(data, time_buckets):
#     """
#     根据时间桶去除周末数据
#     :param data: 原始时间序列数据，形状为 (time_steps, rows, cols, channels)
#     :param time_buckets: 对应的时间步列表，与 data 的第一个维度对齐
#     :return: 非周末数据
#     """
#     import pandas as pd
#     import numpy as np

#     time_buckets = pd.to_datetime(time_buckets)
#     weekdays_mask = ~time_buckets.weekday.isin([5, 6])  # 过滤出工作日
#     return data[weekdays_mask]

```

```
# # 1. 将 LSTM array 转换为 ConvLSTM 格式
# rows, cols = 97, 97
# convlstm_input = reshape_to_convlstm_input(lstm_array, rows, cols)

# # # 2. 去除周末数据（可选）
# # time_buckets = pd.date_range(start="2023-01-01 00:00:00", periods=convlstm_input.shape[0], freq="10min")
# # filtered_data = remove_weekends(convlstm_input, time_buckets)
# # 3. 应用滑动窗口
# # window_size = 10
# # sliding_windows = apply_sliding_window(filtered_data, window_size)

# # 3. 应用滑动窗口
# window_size = 10
# sliding_windows = apply_sliding_window(convlstm_input, window_size)

# # 4. 数据集拆分
# train_dataset, val_dataset = split_dataset(sliding_windows)

# print("训练集形状:", train_dataset.shape)
# print("验证集形状:", val_dataset.shape)

```

归一化

```
# from sklearn.preprocessing import MinMaxScaler

# def normalize_data(train_windows, val_windows):
#     """
#     对滑动窗口数据进行归一化
#     :param train_windows: 训练集滑动窗口
#     :param val_windows: 验证集滑动窗口
#     :return: 归一化后的训练集, 验证集, scaler 对象
#     """
#     train_reshaped = train_windows.reshape(-1, train_windows.shape[-1])
#     val_reshaped = val_windows.reshape(-1, val_windows.shape[-1])

#     scaler = MinMaxScaler()
#     train_scaled = scaler.fit_transform(train_reshaped)
#     val_scaled = scaler.transform(val_reshaped)

#     train_normalized = train_scaled.reshape(train_windows.shape)
#     val_normalized = val_scaled.reshape(val_windows.shape)

#     return train_normalized, val_normalized, scaler

```

```
# # 1. 转换为 ConvLSTM 格式
# rows, cols = 97, 97
# convlstm_input = reshape_to_convlstm_input(lstm_array, rows, cols)

# # 2. 生成滑动窗口和目标值
# window_size = 10
# step = 1
# sliding_windows, targets = apply_sliding_window_with_targets(convlstm_input, window_size, step)

# # 3. 数据集拆分
# train_windows, val_windows, train_targets, val_targets = split_dataset(sliding_windows, targets)

# # 4. 数据归一化
# train_normalized, val_normalized, scaler = normalize_data(train_windows, val_windows)

# # 输出形状验证
# print("训练数据形状:", train_normalized.shape)
# print("训练目标形状:", train_targets.shape)
# print("验证数据形状:", val_normalized.shape)
# print("验证目标形状:", val_targets.shape)

```

```
# import tensorflow as tf

```