هوش مصنوعی

تمرین پنجم

مائده اسماعیل‌زاده

۸۱۰۶۰۲۱۶۱

استاد: دکتر شریعت‌پناهی

دانشکده مهندسی مکانیک

پردیس دانشکده‌های فنی دانشگاه تهران

فهرست مطالب

پیاده‌سازی مدل‌ها

آماده‌سازی داده‌ها

بارگذاری و انتخاب دادگان

در ابتدا کدهای زیر نوشته شده است تا زیرمجموعه FD001 بارگذاری گردد. همانطور که مشاهده می‌شود در این کد ابتدا فایل آپلود شده در مسیر cmaps_data/. استخراج می‌شود. این فایل حاوی فایل فشرده دوم (CMAPSSData.zip) است که داده‌های اصلی را شامل می‌شود. سپس فایل دوم از مسیر داخل فایل اول یافته و در پوشه‌ی جدیدی استخراج می‌شود. این فایل شامل چهار مجموعه داده اصلی با نام‌های train_FD001.txt تا train_FD004.txt است. اطمینان از موفقیت در استخراج فایل‌ها، مسیر تمام فایل‌هایی که از فایل دوم استخراج شده‌اند، نمایش داده شده است. در نام‌گذاری، unit شناسه موتور،cycle چرخه زمانی، op-setting پارامترهای عملیاتی و در نهایت اندازه‌گیری سنسورها می‌باشند. در نهایت نیز با df.head() پنج سطر ابتدایی داده نمایش داده شده‌اند.

```
[52] from google.colab import files
     uploaded = files.upload()

     import zipfile
     import os
```

```
Choose Files  6.+Turbofa...Data+Set.zip
• 6.+Turbofan+Engine+Degradation+Simulation+Data+Set.zip(application/x-zip-compressed) - 12429152 bytes, last modified: 6/30/2025 - 100% done
Saving 6.+Turbofan+Engine+Degradation+Simulation+Data+Set.zip to 6.+Turbofan+Engine+Degradation+Simulation+Data+Set (4).zip
```

```
import zipfile
import os

zip_path = "6.+Turbofan+Engine+Degradation+Simulation+Data+Set.zip"
extract_path = "./cmaps_data"

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

#Barresie fileha
# for root, dirs, files in os.walk(extract_path):
#     for file in files:
#         print(os.path.join(root, file))


zip2_path = "./cmaps_data/6. Turbofan Engine Degradation Simulation Data Set/CMAPSSData.zip"
extract_path2 = "./cmaps_data/CMAPSSData"

with zipfile.ZipFile(zip2_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path2)

# Filehaye estekhraj shode
for root, dirs, files in os.walk(extract_path2):
    for file in files:
        print(os.path.join(root, file))
```

```
import pandas as pd

# Tarife name sutunha
column_names = ['unit', 'cycle'] + [f'op_setting_{i}' for i in range(1, 4)] + \
               [f'sensor_measurement_{i}' for i in range(1, 22)]

# Masire file FD001
data_path = "./cmaps_data/CMAPSSData/train_FD001.txt"

df = pd.read_csv(data_path, sep='\s+', header=None, names=column_names)

# Chand satre aval
df.head()
```

- برای حذف ویژگی‌های با واریانس کم کد زیر نوشته شده است. در این مرحله، برای کاهش ابعاد و حذف ویژگی‌های غیرمؤثر، از تکنیک Variance Threshold استفاده شده است. همانطور که مشاهده می‌شود آستانه ۰.۰۱ تنظیم شده است و ویژگی‌های با واریانس کمتر از این مقدار حذف شده‌اند. ویژگی‌های دارای واریانس کم به شرح زیر می‌باشند:

```
[60] from sklearn.feature_selection import VarianceThreshold

    # Hazfe sutunhaye unit va cycle
    sensor_data = df.drop(columns=['unit', 'cycle'])

    # Filter variance
    Var_selector = VarianceThreshold(threshold=0.01)
    Var_selector.fit(sensor_data)

    # Entekhabe sutunhaye daraye variance kam
    low_var = sensor_data.columns[~Var_selector.get_support()]
    print("Vizhegihaye daraye variance kam:\n", list(low_var))

    # Hazf
    sensor_data2 = df.drop(columns=low_var)
    baghimande = sensor_data2.columns.tolist()
    print("Vizhegihaye baghimande:\n", baghimande)
```

Vizhegihaye daraye variance kam:
['op_setting_1', 'op_setting_2', 'op_setting_3', 'sensor_measurement_1', 'sensor_measurement_5', 'sensor_measurement_6', 'sensor_measurement_8', 'sensor_measurement_10', 'sensor_measurement_13', 'sensor_measurement_15', 'sensor_measurement_16', 'sensor_measurement_18', 'sensor_measurement_19']

Vizhegihaye baghimande:
['unit', 'cycle', 'sensor_measurement_2', 'sensor_measurement_3', 'sensor_measurement_4', 'sensor_measurement_7', 'sensor_measurement_9', 'sensor_measurement_11', 'sensor_measurement_12', 'sensor_measurement_14', 'sensor_measurement_17', 'sensor_measurement_20', 'sensor_measurement_21']

- ویژگی‌های مشابه یا تکراری بصورت زیر شناسایی و یکی از آنها باقی ماند. با استفاده از ماتریس همبستگی ویژگی‌های با همبستگی بیشتر از ۰.۹۵ بصورت ویژگی‌های مشابه تعریف شده و در نهایت یکی از آنها باقی ماندند.

```
import numpy as np

sensor_data_only = sensor_data2.drop(columns=['unit', 'cycle'])

# Matrix hambastegi
corr_matrix = sensor_data_only.corr().abs()
np.fill_diagonal(corr_matrix.values, 0)

to_drop = set()
vizhegi_moshabeh = []

for i in range(len(corr_matrix.columns)):
    for j in range(i + 1, len(corr_matrix.columns)):
        col1 = corr_matrix.columns[i]
        col2 = corr_matrix.columns[j]
        if corr_matrix.iloc[i, j] > 0.95:
            vizhegi_moshabeh.append((col1, col2))
            if col2 not in to_drop:
                to_drop.add(col2)

# Report
print("Joft vizhegihaye moshabeh (hambastegi > 0.95):")
for pair in vizhegi_moshabeh:
    print(pair)

print("\nVizhegihaye hazf shode (yek az har joft):\n", list(to_drop))

# Hazf
sensor_data2 = sensor_data2.drop(columns=to_drop)

final_sensor_columns = sensor_data2.drop(columns=['unit', 'cycle']).columns.tolist()
print("\nTedad vizhegihaye baghimande:", len(final_sensor_columns))
print("Vizhegihaye nahayi:\n", final_sensor_columns)
```

Joft vizhegihaye moshabeh (hambastegi > 0.95):
('sensor_measurement_9', 'sensor_measurement_14')

Vizhegihaye hazf shode (yek az har joft):
['sensor_measurement_14']

Tedad vizhegihaye baghimande: 10
Vizhegihaye nahayi:
['sensor_measurement_2', 'sensor_measurement_3', 'sensor_measurement_4', 'sensor_measurement_7', 'sensor_measurement_9', 'sensor_measurement_11', 'sensor_measurement_12', 'sensor_measurement_17', 'sensor_measurement_20', 'sensor_measurement_21']

- در این بخش عمر مفید باقی‌مانده(RUL) برای هر سطر از داده‌ها بصورت زیر محاسبه شد:

ابتدا حداکثر چرخه عملیاتی (max_cycle) برای هر موتور (unit) استخراج شد. سپس با تفریق مقدار چرخه فعلی از مقدار max_cycle ، مقدار RUL برای هر ردیف محاسبه گردید. در نهایت، ستون max_cycle حذف شد و فقط RUL باقی ماند که به عنوان هدف مدل یادگیری تعریف شده است. سپس با استفاده از تابع همبستگی، ضریب همبستگی بین هر ویژگی حسگر و مقدار RULمحاسبه شد. ویژگی‌هایی که مقدار ضریب همبستگی مطلق آن‌ها کمتر از ۰.۱ بود شناسایی و حذف می‌شوند که در اینجا هیچ ویژگی‌ای این شرایط را نداشت.

```python
# RUL
rul = sensor_data2.groupby('unit')['cycle'].max().reset_index()
rul.columns = ['unit', 'max_cycle']
df_with_rul = sensor_data2.merge(rul, on='unit', how='left')
df_with_rul['RUL'] = df_with_rul['max_cycle'] - df_with_rul['cycle']
df_with_rul.drop(columns=['max_cycle'], inplace=True)

# Chand satr
df_with_rul[['unit', 'cycle', 'RUL']].head()

# Entekhabe vizhegihaye sensor baraye hambastegi(az bakhshe ghabl)
Vizhegiha = ['sensor_measurement_2', 'sensor_measurement_3', 'sensor_measurement_4',
             'sensor_measurement_7','sensor_measurement_9', 'sensor_measurement_11', 'sensor_measurement_12',
             'sensor_measurement_17', 'sensor_measurement_20', 'sensor_measurement_21']

# Mohasebeye hambastegie har vizhegi ba RUL
correlations = df_with_rul[Vizhegiha + ['RUL']].corr()['RUL'].drop('RUL')
print("Hambastegie vizhegiha ba RUL:\n", correlations)

threshold = 0.1 # Astane hambastegi
Hambastegi_low = correlations[correlations.abs() < threshold].index.tolist()
print("\nVizhegihaye ba hambastegie kamtar az (abs) 0.1:\n", Hambastegi_low)

# Hazfe vizhegihaye kam ertebat
final_features_after_rf = [f for f in Vizhegiha if f not in Hambastegi_low]
print("Vizhegihaye nahayi pas az hazfe vizhegihaye kam ertebat: \n", final_features_after_rf)
```

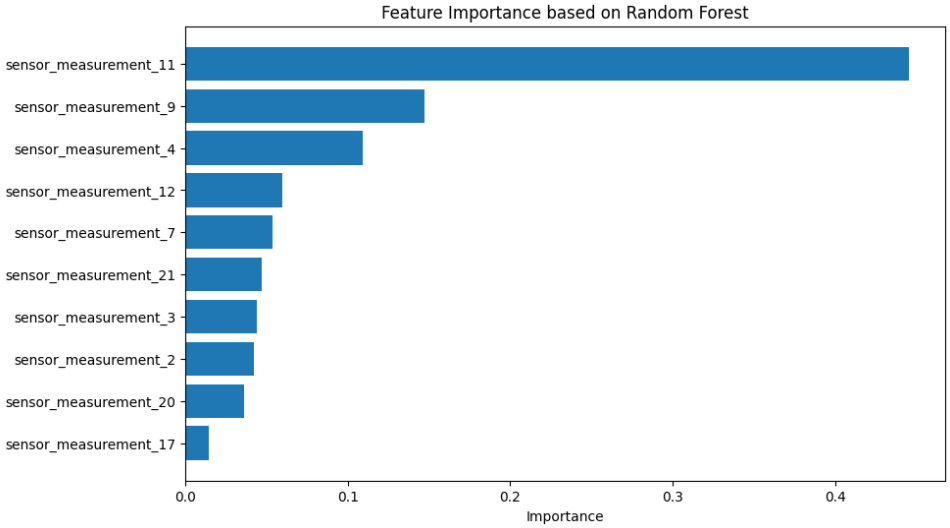
```
Hambastegie vizhegiha ba RUL:
 sensor_measurement_2   -0.606484
sensor_measurement_3    -0.584520
sensor_measurement_4    -0.678948
sensor_measurement_7     0.657223
sensor_measurement_9    -0.390102
sensor_measurement_11   -0.696228
sensor_measurement_12    0.671983
sensor_measurement_17   -0.606154
sensor_measurement_20    0.629428
sensor_measurement_21    0.635662
Name: RUL, dtype: float64

Vizhegihaye ba hambastegie kamtar az (abs) 0.1:
 []
Vizhegihaye nahayi pas az hazfe vizhegihaye kam ertebat:
 ['sensor_measurement_2', 'sensor_measurement_3', 'sensor_measurement_4', 'sensor_measurement_7', 'sensor_measurement_9', 'sensor_measurement_11',
'sensor_measurement_12', 'sensor_measurement_17', 'sensor_measurement_20', 'sensor_measurement_21']
```

- برای حذف ویژگی‌های مهم با مدل Random Forest ازکد زیر استفاده شده است. آستانه در این کد ۰.۰۵ در نظر گرفته شده است و ویژگی‌های با اهمیت کمتر از ۰.۰۵ حذف شدند. نتیجه در پایین قابل مشاهده می‌باشد.

```python
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt
import pandas as pd

features = ['sensor_measurement_2', 'sensor_measurement_3', 'sensor_measurement_4', 'sensor_measurement_7', 'sensor_measurement_9',
            'sensor_measurement_11', 'sensor_measurement_12', 'sensor_measurement_17', 'sensor_measurement_20', 'sensor_measurement_21']
X = df_with_rul[features]
y = df_with_rul['RUL']

rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X, y)

importances = rf.feature_importances_

feature_importance_df = pd.DataFrame({'feature': features, 'importance': importances})
feature_importance_df = feature_importance_df.sort_values(by='importance', ascending=False)

print("Ahamiate vizhegiha bar asase Random Forest:\n", feature_importance_df)

plt.figure(figsize=(10,6))
plt.barh(feature_importance_df['feature'], feature_importance_df['importance'])
plt.xlabel('Importance')
plt.title('Feature Importance based on Random Forest')
plt.gca().invert_yaxis()
plt.show()

threshold = 0.05
low_importance_features = feature_importance_df[feature_importance_df['importance'] < threshold]['feature'].tolist()
print("Vizhegihaye ba ahamiat kamtar az 0.05: {}".format(threshold, low_importance_features))

final_features_after_rf = [f for f in features if f not in low_importance_features]
print("Vizhegihaye nahayi pas az hazfe vizhegiha bar asase ahamiate model:\n", final_features_after_rf)
```

Ahamiate vizhegiha bar asase Random Forest:

	feature	importance
5	sensor_measurement_11	0.445368
4	sensor_measurement_9	0.147277
2	sensor_measurement_4	0.109418
6	sensor_measurement_12	0.059682
3	sensor_measurement_7	0.053638
9	sensor_measurement_21	0.046899
1	sensor_measurement_3	0.044255
0	sensor_measurement_2	0.042419
8	sensor_measurement_20	0.036501
7	sensor_measurement_17	0.014545



Vizhegihaye ba ahamiat kamtar az 0.05: 0.05

Vizhegihaye nahayi pas az hazfe vizhegiha bar asase ahamiate model:

['sensor_measurement_4', 'sensor_measurement_7', 'sensor_measurement_9', 'sensor_measurement_11', 'sensor_measurement_12']

*لازم به ذکر می‌باشد قسمت‌های بعدی با همان ۱۰ ویژگی باقیمانده قبل از random forests انجام شدند. چون این قسمت امتیازی بود صرفا نتیجه آن در این بخش آمده است.

تقسیم داده‌ها به آموزش وآزمون

در این بخش، داده‌ها به مجموعه‌های آموزشی، اعتبارسنجی و آزمون تقسیم شده‌اند. لازم به ذکر است چون داده‌ها وابسته به زمان هستند، تقسیم‌بندی به‌گونه‌ای انجام شده که ترتیب زمانی حفظ شود (یعنی بدون shuffle). ابتدا یک DataFrame به نام df_final ایجاد شده که شامل unit ، cycle ، ویژگی‌های نهایی باقیمانده از بخش قبل و RUL می‌باشد. سپس از داده‌های آموزشی، حدود ۲۰ درصد به عنوان مجموعه اعتبارسنجی (Validation) جدا شدند.

```
from sklearn.model_selection import train_test_split

df_final = df_with_rul[['unit', 'cycle'] + final_features_after_rf + ['RUL']].copy()

# Taghsime asli
train_df, test_df = train_test_split(df_final, test_size=0.3, random_state=42, shuffle=False)

# Taghsime dade amuzesh be train va validation
train_df, val_df = train_test_split(train_df, test_size=0.2, random_state=42, shuffle=False)

print("Tedad nemuneha dar kole DataFrame:", len(df_final))
print("Tedad nemuneha dar dadeye amuzesh:", len(train_df))
print("Tedad nemuneha dar dadeye test:", len(test_df))
print("Tedad nemuneha dar dadeye test:", len(val_df))
print("Units dar amuzesh:", train_df['unit'].nunique())
print("Units dar test:", test_df['unit'].nunique())
```

```
Tedad nemuneha dar kole DataFrame: 20631
Tedad nemuneha dar dadeye amuzesh: 11552
Tedad nemuneha dar dadeye test: 6190
Tedad nemuneha dar dadeye test: 2889
Units dar amuzesh: 59
Units dar test: 29
```

نرمال‌سازی داده‌ها

در این پروژه از روش Min-Max Scaling استفاده شده که داده‌ها را به بازه‌ی [0,1] تبدیل می‌کند. در نهایت نیز برای اطمینان از صحت نرمال‌سازی، میانگین و انحراف معیار تعدادی از ویژگی‌های نرمال‌شده نمایش داده شده است همان‌طور که دیده می‌شود، میانگین بیشتر ویژگی‌ها بین بازه [0,1] قرار دارد و انحراف معیار نیز در حد قابل قبولی است، که نشان می‌دهد نرمال‌سازی به‌درستی انجام شده است.

```
[9] from sklearn.preprocessing import MinMaxScaler

    # Normalsazi faghat bar asase train
    scaler = MinMaxScaler()
    scaler.fit(train_df[final_features_after_rf])

    train_df[final_features_after_rf] = train_df[final_features_after_rf].astype(float)
    val_df[final_features_after_rf] = val_df[final_features_after_rf].astype(float)
    test_df[final_features_after_rf] = test_df[final_features_after_rf].astype(float)

    # Normalsazi ruye tamame dadeha
    train_df.loc[:, final_features_after_rf] = scaler.transform(train_df[final_features_after_rf])
    val_df.loc[:, final_features_after_rf] = scaler.transform(val_df[final_features_after_rf])
    test_df.loc[:, final_features_after_rf] = scaler.transform(test_df[final_features_after_rf])

    print("\nMiangin va enheraf meyar chand vizhegie aval az dade train bad az normalsazi:")
    print(train_df[final_features_after_rf].describe().T[['mean', 'std']].head())
```

```
Miangin va enheraf meyar chand vizhegie aval az dade train bad az normalsazi:
                      mean       std
sensor_measurement_2  0.433661  0.154237
sensor_measurement_3  0.441354  0.140455
sensor_measurement_4  0.403815  0.164607
sensor_measurement_7  0.571761  0.143542
sensor_measurement_9  0.203473  0.109024
```

تقسیم‌بندی داده‌ها با روش پنجره لغزان و برچسب‌گذاری پنجره‌ها با مقدار RUL

در این بخش برای تقسیم‌بندی داده‌های سری زمانی به بخش‌های هم‌اندازه و پیوسته، از روش Sliding Window استفاده شد. برای هر پنجره، مقدار RUL آخرین چرخه به‌عنوان برچسب پنجره انتخاب شد. همچنین با در نظر گرفتن حداکثر مقدار مجاز برای

RUL(حداکثر ۱۳۰ چرخه)، از بروز برچسب‌های غیرواقع‌بینانه جلوگیری شد.

لازم به ذکر است این فرایند به‌صورت مستقل برای داده‌های آموزش، اعتبارسنجی و آزمون انجام شد. همچنین در این بخش فقط ستون‌های ویژگی‌های مهم انتخاب شده در مراحل قبل برای هر پنجره استخراج می‌شود. در نهایت ابعاد آرایه‌های داده و برچسب‌ها نمایش داده شده است که نشان‌دهنده تعداد نمونه‌های پنجره‌ای و اندازه هر پنجره است. در نهایت یک مثال نیز آمده که بررسی شود مراحل درست انجام شده‌اند.

نتایج نشان می‌دهد برای X_train_windows ۹۸۵۷ پنجره ساخته شده است، هر پنجره شامل ۳۰ چرخه متوالی (window_size=30) است و در هر چرخه ۱۰ ویژگی وجود دارد. بردار برچسب‌ها (y_train_windows) هم به تعداد پنجره‌ها یعنی ۹۸۵۷ مقدار دارد که به ازای هر پنجره یک مقدار RUL محدود شده است. مشابه همین ساختار برای داده‌های تست با ۵۳۴۹ نمونه پنجره‌ای و همان اندازه پنجره و ویژگی‌ها مشاهده می‌شود.

```
[ ] import numpy as np

    def sliding_windows(df, feature_cols, window_size=30, max_rul=130):
        X_windows = []
        y_labels = []

        for unit_number, unit_df in df.groupby('unit'):
            unit_df = unit_df.sort_values('cycle').reset_index(drop=True)
            if len(unit_df) < window_size:
                continue
            for i in range(len(unit_df) - window_size + 1):
                window = unit_df.iloc[i:i+window_size]
                X_window = window[feature_cols].values
                rul_value = window.iloc[-1]['RUL']
                rul_capped = min(rul_value, max_rul)
                X_windows.append(X_window)
                y_labels.append(rul_capped)

        return np.array(X_windows), np.array(y_labels)

    # Tolide dadehaye panjereyi baraye har bakhsh
    window_size = 30
    max_rul = 130

    X_train_windows, y_train_windows = sliding_windows(train_df, final_features_after_rf, window_size, max_rul)
    X_val_windows, y_val_windows = sliding_windows(val_df, final_features_after_rf, window_size, max_rul)
    X_test_windows, y_test_windows = sliding_windows(test_df, final_features_after_rf, window_size, max_rul)
```

```
[ ] print(f"X_train_windows shape: {X_train_windows.shape}")
    print(f"y_train_windows shape: {y_train_windows.shape}")
    print(f"X_test_windows shape: {X_test_windows.shape}")
    print(f"y_test_windows shape: {y_test_windows.shape}")

    # Mesal baraye tahlil
    print("\nX_train_windows:")
    print(X_train_windows[0])  # Matrix panjere aval

    print("\nBrachasb:")
    print(y_train_windows[0])
```

```
X_train_windows shape: (9857, 30, 10)
y_train_windows shape: (9857,)
X_test_windows shape: (5349, 30, 10)
y_test_windows shape: (5349,)

X_train_windows:
[[0.17378049 0.42515379 0.25735561 0.72624799 0.109755   0.36526946
  0.62197802 0.36363636 0.70866142 0.72548186]
 [0.27439024 0.47345637 0.3034871  0.62801932 0.1002423  0.37724551
  0.75824176 0.36363636 0.66141732 0.73200113]
 [0.33536585 0.38619275 0.32273883 0.71014493 0.14004308 0.24550898
  0.78901099 0.18181818 0.62204724 0.61947279]
 [0.33536585 0.26771474 0.28042136 0.74074074 0.12451763 0.16167665
  0.88571429 0.36363636 0.56692913 0.66156463]
 [0.34146341 0.2690818  0.35942608 0.66827697 0.14995962 0.25149701
  0.73846154 0.45454545 0.58267717 0.70479025]
 [0.25914634 0.30599225 0.21685434 0.77616747 0.12541506 0.17964072
  0.62637363 0.27272727 0.64566929 0.65164399]
 [0.375      0.48484848 0.20595714 0.72302738 0.16781836 0.2994012
  0.76703297 0.36363636 0.74015748 0.66652494]
 [0.39939024 0.27158806 0.26407555 0.64412238 0.08556942 0.22754491
```

0.8 0.27272727 0.63779528 0.57185374]
[0.2652439 0.45431761 0.15201598 0.61835749 0.11096653 0.25748503
0.65054945 0.36363636 0.7007874 0.70790816]
[0.1402439 0.46024151 0.25481293 0.60225443 0.13447905 0.10179641
0.65054945 0.45454545 0.62204724 0.79691043]
[0.31402439 0.24401914 0.25808209 0.75523349 0.12510096 0.17365269
0.56483516 0.36363636 0.61417323 0.8100907]
[0.24695122 0.28184097 0.24918271 0.75201288 0.12402405 0.19161677
0.65274725 0.27272727 0.70866142 0.65036848]
[0.55487805 0.2540442 0.26153287 0.57809984 0.11258189 0.31137725
0.66373626 0.45454545 0.60629921 0.52239229]
[0.33536585 0.49920255 0.23120232 0.74557166 0.11504981 0.34730539
0.62417582 0.45454545 0.80314961 0.67389456]
[0.3597561 0.2911825 0.28514348 0.61030596 0.13681235 0.26347305
0.80659341 0.27272727 0.65354331 0.62769274]
[0.26829268 0.38596491 0.32818743 0.65861514 0.12388944 0.22754491
0.58461538 0.36363636 0.63779528 0.77650227]
[0.4054878 0.31715653 0.24555031 0.63607085 0.14892758 0.15568862
0.67252747 0.36363636 0.51181102 0.60204082]
[0.41768293 0.45568467 0.17598983 0.70048309 0.12483173 0.20958084
0.64395604 0.36363636 0.57480315 0.69671202]
[0.16463415 0.37639553 0.25281511 0.69726248 0.14475455 0.32335329
0.67252747 0.27272727 0.50393701 0.6225907]
[0.54573171 0.22943723 0.3414457 0.79871176 0.10845374 0.21556886
0.71208791 0.36363636 0.68503937 0.72973356]
[0.34146341 0.34244703 0.21249546 0.68115942 0.11854976 0.17365269
0.78901099 0.36363636 0.73228346 0.57114512]
[0.46341463 0.49874687 0.25681075 0.60869565 0.17715157 0.22754491
0.6967033 0.36363636 0.5984252 0.66907596]
[0.27134146 0.3907496 0.15110788 0.66505636 0.11015884 0.23353293
0.59120879 0.36363636 0.61417323 0.77820295]
[0.3445122 0.45089998 0.22484562 0.58615137 0.0988513 0.34730539
0.72527473 0.36363636 0.66141732 0.65547052]
[0.46341463 0.52540442 0.23537959 0.66827697 0.14551737 0.2994012
0.81978022 0.45454545 0.62204724 0.73937075]
[0.27743902 0.41102757 0.17508173 0.68599034 0.11388316 0.23952096
0.75824176 0.54545455 0.5511811 0.7196712]
[0.36280488 0.44269765 0.2798765 0.67954911 0.11769721 0.30538922
0.6967033 0.45454545 0.65354331 0.76544785]
[0.33536585 0.26885395 0.23065746 0.77777778 0.126133 0.32934132
0.82197802 0.18181818 0.66929134 0.53429705]
[0.20121951 0.31419458 0.26443879 0.70853462 0.12895989 0.22754491
0.78681319 0.45454545 0.60629921 0.64143991]
[0.28963415 0.51218956 0.17526335 0.72785829 0.10719734 0.32335329
0.73846154 0.18181818 0.7007874 0.71414399]]

Brachasb:
130.0

پیاده‌سازی شبکه عصبی پیچشی

ورودی مدل به شکل (اندازه پنجره، تعداد ویژگی‌ها) است که از داده‌های پنجره‌ای آماده شده استخراج می‌شود. مطابق صورت سوال مدل به صورت متوالی (Sequential) تعریف شده و شامل لایه‌های زیر است:

۱. Conv1D با ۶۴ فیلتر: کرنل اندازه ۳، تابع فعال‌سازی ReLU ، پدینگ به صورت 'same' تا ابعاد خروجی ثابت بماند.

۲. MaxPooling1D : کاهش اندازه داده‌ها با پنجره‌ی سایز ۲.

۳. Conv1D با ۱۲۸ فیلتر: کرنل ۳، ReLU و پدینگ 'same'

۴. GlobalAveragePooling1D : میانگین‌گیری روی تمام طول توالی خروجی لایه کانولوشن، کاهش ابعاد به برداری یک‌بعدی.

۵. Dense : لایه کاملاً متصل با ۶۴ نورون و فعال‌سازی ReLU

۶. Dropout : رهاسازی تصادفی ۲۰٪ نورون‌ها برای جلوگیری از بیش‌برازش.

۷. Dense : لایه خروجی با یک نورون

سپس مطابق سوال، مدل با تابع خطای میانگین مربعات خطا (MSE) کامپایل می‌شود، بهینه‌ساز Adam با نرخ یادگیری ۰.۰۰۱ استفاده شده و معیار ارزیابی میانگین قدر مطلق خطا (MAE) تعریف شده است.

در نهایت مدل روی داده‌های آموزش (X_train_windows, y_train_windows) با اندازه دسته ۳۲ و تعداد دوره ۵۰ آموزش داده شده است. داده‌های اعتبارسنجی (X_val_windows, y_val_windows) در هر دوره برای کنترل عملکرد استفاده می‌شوند.

در نهایت نمودارها و مشخصه های خواسته شده در تمرین محاسبه و رسم شده‌اند.

```
[ ] import tensorflow as tf
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Conv1D, MaxPooling1D, GlobalAveragePooling1D, Dense, Dropout
    from tensorflow.keras.optimizers import Adam
    from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
    import matplotlib.pyplot as plt
    from sklearn.metrics import mean_squared_error
    import random
    import time
    from sklearn.metrics import r2_score

    # reproducibility
    seed = 42
    np.random.seed(seed)
    tf.random.set_seed(seed)
    random.seed(seed)

    # CNN
    input_shape = X_train_windows.shape[1:]
    model = Sequential([
        Conv1D(64, 3, padding='same', activation='relu', input_shape=input_shape),
        MaxPooling1D(pool_size=2),
        Conv1D(128, 3, padding='same', activation='relu'),
        GlobalAveragePooling1D(),
        Dense(64, activation='relu'),
        Dropout(0.2),
        Dense(1)
    ])
```

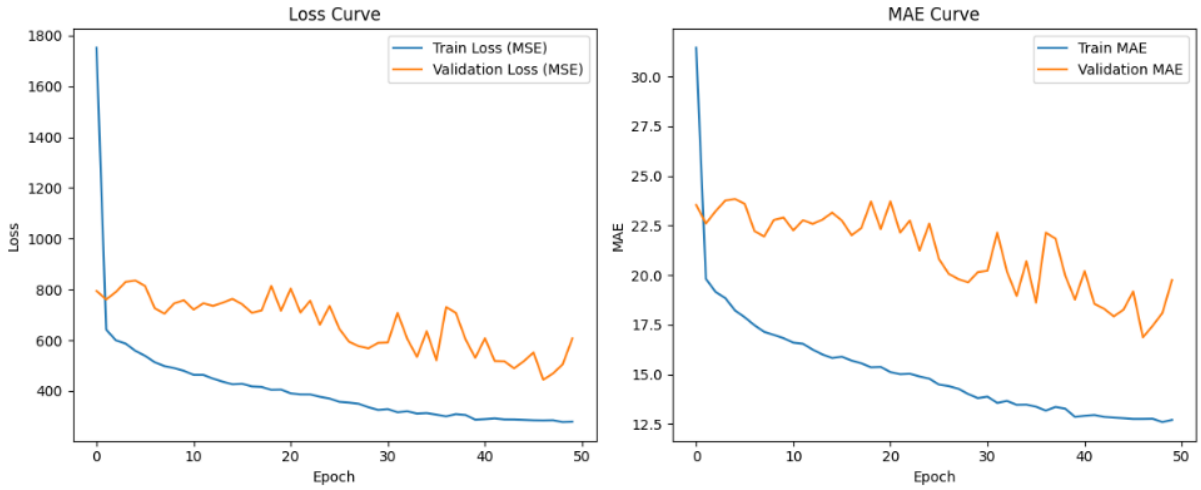
```
[ ] model.compile(
        optimizer=Adam(learning_rate=0.001),
        loss='mse',
        metrics=[tf.keras.metrics.MeanAbsoluteError(name='mae')]
    )

    model.summary()

    # # Callbacks
    # early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
    # reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6, verbose=1)

    # Zamane shurue amuzesh
    start_time = time.time()

    # Amuzeshe model
    history = model.fit(
        X_train_windows, y_train_windows,
        validation_data=(X_val_windows, y_val_windows),
        batch_size=32,
        epochs=50,
        callbacks=[],
        verbose=2
    )

    # Zamane payane amuzesh
    end_time = time.time()
    training_time = end_time - start_time
```

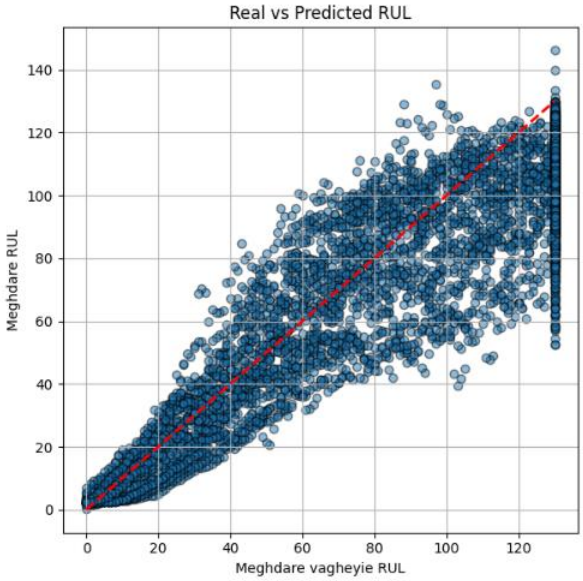
```
Epoch 1/50
309/309 - 7s - 22ms/step - loss: 1752.1079 - mae: 31.4640 - val_loss: 794.2220 - val_mae: 23.5446
Epoch 2/50
309/309 - 1s - 3ms/step - loss: 641.6176 - mae: 19.8216 - val_loss: 760.4941 - val_mae: 22.6051
Epoch 3/50
309/309 - 1s - 4ms/step - loss: 599.5368 - mae: 19.1586 - val_loss: 790.1995 - val_mae: 23.2282
Epoch 4/50
309/309 - 2s - 5ms/step - loss: 586.6824 - mae: 18.8487 - val_loss: 829.6028 - val_mae: 23.7645
```

Epoch 5/50
309/309 - 2s - 6ms/step - loss: 558.2981 - mae: 18.2266 - val_loss: 835.1032 - val_mae: 23.8402
Epoch 6/50
309/309 - 1s - 3ms/step - loss: 538.6989 - mae: 17.8828 - val_loss: 813.6921 - val_mae: 23.5966
Epoch 7/50
309/309 - 1s - 5ms/step - loss: 513.1932 - mae: 17.4828 - val_loss: 725.7574 - val_mae: 22.2257
Epoch 8/50
309/309 - 2s - 7ms/step - loss: 498.1468 - mae: 17.1495 - val_loss: 704.6996 - val_mae: 21.9516
Epoch 9/50
309/309 - 1s - 4ms/step - loss: 490.1612 - mae: 16.9948 - val_loss: 745.0223 - val_mae: 22.7875
Epoch 10/50
309/309 - 1s - 4ms/step - loss: 479.4615 - mae: 16.8303 - val_loss: 757.6230 - val_mae: 22.9047
Epoch 11/50
309/309 - 1s - 5ms/step - loss: 463.6826 - mae: 16.6061 - val_loss: 720.4159 - val_mae: 22.2598
Epoch 12/50
309/309 - 1s - 4ms/step - loss: 463.5093 - mae: 16.5493 - val_loss: 745.5660 - val_mae: 22.7742
Epoch 13/50
309/309 - 1s - 3ms/step - loss: 448.7548 - mae: 16.2535 - val_loss: 735.1830 - val_mae: 22.5870
Epoch 14/50
309/309 - 1s - 3ms/step - loss: 436.6946 - mae: 16.0109 - val_loss: 747.7476 - val_mae: 22.8023
Epoch 15/50
309/309 - 2s - 5ms/step - loss: 426.5928 - mae: 15.8322 - val_loss: 762.9299 - val_mae: 23.1530
Epoch 16/50
309/309 - 1s - 4ms/step - loss: 428.4519 - mae: 15.8952 - val_loss: 741.6001 - val_mae: 22.7554
Epoch 17/50
309/309 - 1s - 3ms/step - loss: 418.1249 - mae: 15.7030 - val_loss: 707.9769 - val_mae: 22.0128
Epoch 18/50
309/309 - 1s - 3ms/step - loss: 416.1113 - mae: 15.5659 - val_loss: 717.4688 - val_mae: 22.3755
Epoch 19/50
309/309 - 1s - 3ms/step - loss: 404.6671 - mae: 15.3648 - val_loss: 813.6177 - val_mae: 23.7092
Epoch 20/50
309/309 - 1s - 3ms/step - loss: 405.4845 - mae: 15.3824 - val_loss: 716.1199 - val_mae: 22.3282
Epoch 21/50
309/309 - 1s - 3ms/step - loss: 390.8886 - mae: 15.1230 - val_loss: 803.5231 - val_mae: 23.7217
Epoch 22/50
309/309 - 1s - 4ms/step - loss: 386.7499 - mae: 15.0162 - val_loss: 708.9901 - val_mae: 22.1445
Epoch 23/50
309/309 - 1s - 3ms/step - loss: 386.5551 - mae: 15.0384 - val_loss: 755.4164 - val_mae: 22.7579
Epoch 24/50
309/309 - 1s - 5ms/step - loss: 377.4248 - mae: 14.9021 - val_loss: 661.0602 - val_mae: 21.2446
Epoch 25/50
309/309 - 1s - 4ms/step - loss: 370.4015 - mae: 14.7917 - val_loss: 735.1842 - val_mae: 22.5992
Epoch 26/50
309/309 - 2s - 5ms/step - loss: 357.4793 - mae: 14.4930 - val_loss: 645.3134 - val_mae: 20.8177
Epoch 27/50
309/309 - 1s - 3ms/step - loss: 354.3963 - mae: 14.4175 - val_loss: 594.2355 - val_mae: 20.0745
Epoch 28/50
309/309 - 1s - 3ms/step - loss: 349.7443 - mae: 14.2829 - val_loss: 576.5229 - val_mae: 19.8057
Epoch 29/50
309/309 - 1s - 3ms/step - loss: 336.1053 - mae: 14.0202 - val_loss: 568.1516 - val_mae: 19.6419
Epoch 30/50
309/309 - 1s - 3ms/step - loss: 325.6908 - mae: 13.8122 - val_loss: 589.6657 - val_mae: 20.1546
Epoch 31/50
309/309 - 1s - 3ms/step - loss: 328.8303 - mae: 13.8867 - val_loss: 591.1899 - val_mae: 20.2272
Epoch 32/50
309/309 - 1s - 3ms/step - loss: 316.2696 - mae: 13.5695 - val_loss: 707.7805 - val_mae: 22.1520
Epoch 33/50
309/309 - 1s - 4ms/step - loss: 320.2303 - mae: 13.6691 - val_loss: 605.3565 - val_mae: 20.1842
Epoch 34/50
309/309 - 1s - 3ms/step - loss: 311.1967 - mae: 13.4758 - val_loss: 534.3707 - val_mae: 18.9620
Epoch 35/50

309/309 - 1s - 4ms/step - loss: 313.1350 - mae: 13.4853 - val_loss: 635.5728 - val_mae: 20.7032
Epoch 36/50
309/309 - 2s - 5ms/step - loss: 306.9713 - mae: 13.3780 - val_loss: 520.7448 - val_mae: 18.6131
Epoch 37/50
309/309 - 1s - 4ms/step - loss: 300.2816 - mae: 13.1827 - val_loss: 730.7276 - val_mae: 22.1462
Epoch 38/50
309/309 - 2s - 7ms/step - loss: 308.9963 - mae: 13.3672 - val_loss: 707.8040 - val_mae: 21.8442
Epoch 39/50
309/309 - 1s - 3ms/step - loss: 305.6000 - mae: 13.2794 - val_loss: 603.4782 - val_mae: 19.9825
Epoch 40/50
309/309 - 1s - 4ms/step - loss: 287.1256 - mae: 12.8678 - val_loss: 530.4536 - val_mae: 18.7772
Epoch 41/50
309/309 - 1s - 4ms/step - loss: 289.2954 - mae: 12.9224 - val_loss: 607.8920 - val_mae: 20.2109
Epoch 42/50
309/309 - 1s - 3ms/step - loss: 292.2492 - mae: 12.9603 - val_loss: 518.1780 - val_mae: 18.5616
Epoch 43/50
309/309 - 1s - 3ms/step - loss: 288.0579 - mae: 12.8667 - val_loss: 516.9916 - val_mae: 18.3114
Epoch 44/50
309/309 - 1s - 3ms/step - loss: 287.8311 - mae: 12.8291 - val_loss: 489.3580 - val_mae: 17.9281
Epoch 45/50
309/309 - 1s - 4ms/step - loss: 286.3511 - mae: 12.7996 - val_loss: 517.3658 - val_mae: 18.2659
Epoch 46/50
309/309 - 1s - 4ms/step - loss: 284.7812 - mae: 12.7633 - val_loss: 551.8780 - val_mae: 19.1881
Epoch 47/50
309/309 - 1s - 4ms/step - loss: 284.2140 - mae: 12.7641 - val_loss: 444.2758 - val_mae: 16.8730
Epoch 48/50
309/309 - 1s - 4ms/step - loss: 284.9115 - mae: 12.7727 - val_loss: 468.9955 - val_mae: 17.4532
Epoch 49/50
309/309 - 1s - 3ms/step - loss: 277.9487 - mae: 12.6080 - val_loss: 504.9226 - val_mae: 18.0963
Epoch 50/50
309/309 - 1s - 3ms/step - loss: 279.3629 - mae: 12.7143 - val_loss: 607.0616 - val_mae: 19.7633

```
Zamane amuzeshe model 66.61 sec
MSE: 481.7840
MAE: 17.1138
168/168 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
RMSE: 21.9496
log RMSE: 0.3331
R² Score: 0.7469
```





باتوجه به ایپاک‌های این مدل که در بالا قابل مشاهده می‌باشد، در ایپاک ۱ تا ۱۰، مقدار اولیه loss و MAE بسیار بالا هستند، همچنین کاهش سریعی در خطاها دیده می‌شود، اما اعتبارسنجی نوسان دارد. بنابراین بطور کلی مدل در حال یادگیری ویژگی‌هاست، ولی در ایپاک‌های اولیه نشانه‌هایی از نوسان و عدم پایداری در اعتبارسنجی دیده می‌شود.

در ایپاک ۱۱ تا ۳۰، کاهش training loss و MAE ادامه دارد، ولی validation loss همچنان نوسانی است(گاهی بهبود و گاهی افزایش). بنابراین تا ایپاک ۳۰ به نظر می‌رسد مدل overfit نشده ولی به طور کامل نیز generalize نمی‌شود. البته همانطور که گفته شد بهبود با کارهایی مانند earlystopping، dropoutیا regularization ممکن است. اما از آنجاییکه قرار است شرایط تمام مدلها یکسان باشد تغییری در مدل ایجاد نمی‌کنیم.

در ایپاک‌های پایانی ۳۱ تا ۵۰، Training loss به کمترین حد می‌رسد و در اواخر آموزش، نقاط بهینه‌ای برای اعتبارسنجی داریم(ایپاک ۴۴ و ۴۷) . در نتیجه می‌توان گفت بطور کلی مدل در برخی نقاط توانایی درک بهتر الگو را دارد، اما به علت ضعف در درک وابستگی زمانی عملکردش پایدار نیست.

باتوجه به نمودار Train loss می‌توان گفت این نمودار به‌صورت یکنواخت کاهش یافته و در انتها به مقدار نسبتاً پایین رسیده است. اما نمودار Validation loss نوسانات زیادی دارد و روند کاهش واضحی مشاهده نمی‌شود. در نتیجه می‌توان گفت احتمالاً overfitting اتفاق افتاده است، زیرا مدل روی داده‌های آموزش عملکرد خوبی دارد اما در داده‌های اعتبارسنجی ضعیف‌تر عمل می‌کند.

با توجه به نمودار RUL واقعی و پیش‌بینی شده، مشاهده می‌شود خط قرمز (خط ایده‌آل) با نقاط فاصله زیادی دارد و پیش‌بینی‌ها در محدوده RULهای بالا پراکندگی بیشتری دارند. بطور کلی می‌توان گفت عملکرد قابل‌قبول است ولی دقت کلی متوسط است. (کاهش یکنواخت خطای آموزش و عدم بروز overfitting شدید). بطور کلی زمان آموزش بسیار کم، MSE نسبتا زیاد، MAE متوسط رو به بالا، RMSE نسبتا بالاو دقت نسبتا خوبی دارد.

علت این اتفاق آن است که CNN فقط ویژگی‌های محلی را می‌بیند، نه وابستگی بین زمان‌ها. یعنی مدل CNN به تنهایی برای مسأله تخمین RUL مناسب نیست، چون ساختار آن برای تحلیل وابستگی‌های زمانی مناسب طراحی نشده است.

پیاده‌سازی شبکه LSTM

در این بخش کد زیر نوشته شده است. تمامی شرایط خواسته شده در صورت سوال رعایت شده‌اند و معماری دقیقا همان معماری خواسته شده می‌باشد. با افزودن شرایطی مثل Earlystopping و .. که در بخش CallBack در کد کامنت شده است، می‌توان نتایج را بهبود بخشید اما از آنجاییکه گفته شده شرایط تمامی مدل‌ها (Batch size و Epoch) یکسان باشد از آن بخش استفاده نشده است.

```
[ ] from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import LSTM, Dense, Dropout
    from tensorflow.keras.optimizers import Adam
    from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
    from sklearn.metrics import mean_squared_error
    import random
    import tensorflow as tf
    from sklearn.metrics import r2_score
    import numpy as np
    import matplotlib.pyplot as plt
    import time

    # reproducibility
    seed = 42
    np.random.seed(seed)
    tf.random.set_seed(seed)
    random.seed(seed)

    # LSTM
    input_shape = X_train_windows.shape[1:]

    model_lstm = Sequential([
        LSTM(100, return_sequences=True, input_shape=input_shape),
        Dropout(0.2),
        LSTM(50, return_sequences=False),
        Dense(64, activation='relu'),
        Dropout(0.2),
        Dense(1)
    ])
```

```
[ ] model_lstm.compile(
        optimizer=Adam(learning_rate=0.001),
        loss='mse',
        metrics=[tf.keras.metrics.MeanAbsoluteError(name='mae')]
    )

    model_lstm.summary()

    # # callbacks
    # early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
    # reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6, verbose=1)

    start_time = time.time()

    # Amuzeshe model
    history_lstm = model_lstm.fit(
        X_train_windows, y_train_windows,
        validation_data=(X_val_windows, y_val_windows),
        batch_size=32,
        epochs=50,
        callbacks=[],
        verbose=2
    )


    end_time = time.time()
    print(f"\nTraining time: {(end_time - start_time)/60:.2f} minutes")
```

```python
# Arzyabi ruye dade test
eval_results = model_lstm.evaluate(X_test_windows, y_test_windows, verbose=0)
print(f"MSE: {eval_results[0]:.4f}")
print(f"MAE: {eval_results[1]:.4f}")

y_pred_lstm = model_lstm.predict(X_test_windows)
rmse = np.sqrt(mean_squared_error(y_test_windows, y_pred_lstm))
y_pred_safe1 = np.maximum(y_pred_lstm, 0)
y_test_safe1 = np.maximum(y_test_windows, 0)
log_rmse = np.sqrt(np.mean(np.square(np.log1p(y_test_safe1) - np.log1p(y_pred_safe1))))


#log_rmse = np.log(rmse + 1e-10)
r2 = r2_score(y_test_windows, y_pred_lstm)

print(f"RMSE: {rmse:.4f}")
print(f"log RMSE: {log_rmse:.4f}")
print(f"R² Score: {r2:.4f}")

# Nemudarha
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history_lstm.history['loss'], label='Train Loss (MSE)')
plt.plot(history_lstm.history['val_loss'], label='Validation Loss (MSE)')
plt.xlabel('Epoch'); plt.ylabel('Loss'); plt.title('Loss Curve - LSTM'); plt.legend()

plt.subplot(1,2,2)
plt.plot(history_lstm.history['mae'], label='Train MAE')
```

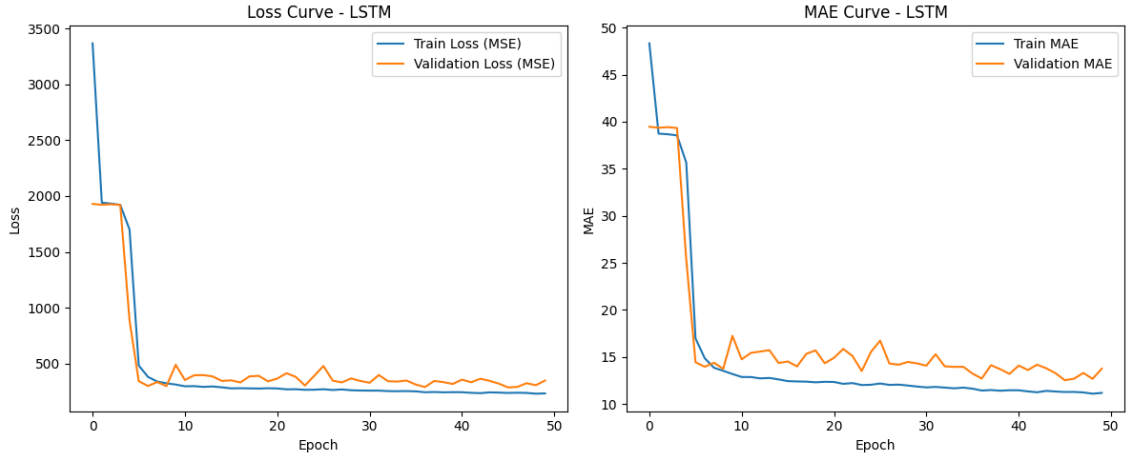
```python
plt.plot(history_lstm.history['val_mae'], label='Validation MAE')
plt.xlabel('Epoch'); plt.ylabel('MAE'); plt.title('MAE Curve - LSTM'); plt.legend()

plt.tight_layout()
plt.show()

# Nemudare vagheyi va pishbini shode
plt.figure(figsize=(6,6))
plt.scatter(y_test_windows, y_pred_lstm, alpha=0.5, edgecolors='k')
plt.plot([y_test_windows.min(), y_test_windows.max()],
         [y_test_windows.min(), y_test_windows.max()],
         'r--', lw=2)
plt.xlabel("Meghdare vagheyie RUL")
plt.ylabel("Meghdare RUL")
plt.title("Real vs Predicted RUL")
plt.grid(True)
plt.tight_layout()
plt.show()
```

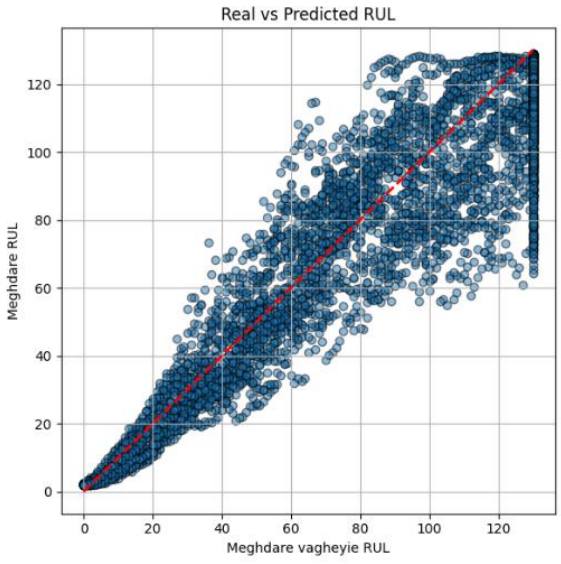
```
Epoch 1/50
309/309 - 7s - 24ms/step - loss: 3366.1685 - mae: 48.3240 - val_loss: 1928.8341 - val_mae: 39.4596
Epoch 2/50
309/309 - 3s - 9ms/step - loss: 1940.5917 - mae: 38.7286 - val_loss: 1921.3629 - val_mae: 39.3537
Epoch 3/50
309/309 - 3s - 9ms/step - loss: 1931.9180 - mae: 38.6648 - val_loss: 1925.8969 - val_mae: 39.4190
Epoch 4/50
309/309 - 5s - 16ms/step - loss: 1920.5151 - mae: 38.5464 - val_loss: 1919.6014 - val_mae: 39.3284
Epoch 5/50
309/309 - 5s - 16ms/step - loss: 1700.4163 - mae: 35.6480 - val_loss: 887.2142 - val_mae: 25.3586
Epoch 6/50
309/309 - 3s - 10ms/step - loss: 483.2847 - mae: 16.9785 - val_loss: 343.0648 - val_mae: 14.4400
Epoch 7/50
309/309 - 3s - 8ms/step - loss: 381.2501 - mae: 14.8644 - val_loss: 299.9770 - val_mae: 13.9605
Epoch 8/50
309/309 - 2s - 8ms/step - loss: 339.5921 - mae: 13.8561 - val_loss: 334.8543 - val_mae: 14.3854
Epoch 9/50
309/309 - 2s - 8ms/step - loss: 323.6913 - mae: 13.5317 - val_loss: 298.5952 - val_mae: 13.7435
Epoch 10/50
309/309 - 3s - 9ms/step - loss: 312.7867 - mae: 13.1885 - val_loss: 489.6844 - val_mae: 17.2444
Epoch 11/50
309/309 - 5s - 16ms/step - loss: 297.1247 - mae: 12.8761 - val_loss: 353.3381 - val_mae: 14.7519
Epoch 12/50
309/309 - 2s - 8ms/step - loss: 298.6225 - mae: 12.8673 - val_loss: 395.9778 - val_mae: 15.4290
Epoch 13/50
309/309 - 3s - 8ms/step - loss: 291.9412 - mae: 12.7246 - val_loss: 397.1234 - val_mae: 15.5793
Epoch 14/50
309/309 - 3s - 10ms/step - loss: 295.4506 - mae: 12.7697 - val_loss: 385.5269 - val_mae: 15.7200
Epoch 15/50
309/309 - 3s - 8ms/step - loss: 287.9074 - mae: 12.6149 - val_loss: 345.4636 - val_mae: 14.3560
Epoch 16/50
309/309 - 2s - 8ms/step - loss: 278.5830 - mae: 12.4338 - val_loss: 350.4322 - val_mae: 14.5238
Epoch 17/50
309/309 - 3s - 8ms/step - loss: 279.6964 - mae: 12.3992 - val_loss: 331.6288 - val_mae: 14.0005
Epoch 18/50
309/309 - 3s - 9ms/step - loss: 278.4665 - mae: 12.3798 - val_loss: 386.0461 - val_mae: 15.3304
```

Epoch 19/50
309/309 - 5s - 15ms/step - loss: 277.0053 - mae: 12.3095 - val_loss: 390.8988 - val_mae: 15.6987
Epoch 20/50
309/309 - 2s - 8ms/step - loss: 279.6289 - mae: 12.3613 - val_loss: 341.1932 - val_mae: 14.3318
Epoch 21/50
309/309 - 3s - 8ms/step - loss: 277.7705 - mae: 12.3462 - val_loss: 366.4558 - val_mae: 14.9113
Epoch 22/50
309/309 - 3s - 10ms/step - loss: 269.8778 - mae: 12.1458 - val_loss: 414.0476 - val_mae: 15.8536
Epoch 23/50
309/309 - 5s - 15ms/step - loss: 270.9143 - mae: 12.2257 - val_loss: 381.6169 - val_mae: 15.1057
Epoch 24/50
309/309 - 3s - 8ms/step - loss: 265.8003 - mae: 12.0179 - val_loss: 303.7809 - val_mae: 13.5017
Epoch 25/50
309/309 - 3s - 8ms/step - loss: 266.5339 - mae: 12.0489 - val_loss: 390.2833 - val_mae: 15.5346
Epoch 26/50
309/309 - 3s - 10ms/step - loss: 270.8489 - mae: 12.1802 - val_loss: 478.9626 - val_mae: 16.7357
Epoch 27/50
309/309 - 5s - 15ms/step - loss: 263.8727 - mae: 12.0323 - val_loss: 347.8285 - val_mae: 14.3037
Epoch 28/50
309/309 - 5s - 17ms/step - loss: 268.9483 - mae: 12.0622 - val_loss: 332.0685 - val_mae: 14.1807
Epoch 29/50
309/309 - 3s - 9ms/step - loss: 261.8083 - mae: 11.9660 - val_loss: 367.6436 - val_mae: 14.4725
Epoch 30/50
309/309 - 5s - 16ms/step - loss: 259.3731 - mae: 11.8574 - val_loss: 344.8060 - val_mae: 14.3176
Epoch 31/50
309/309 - 2s - 8ms/step - loss: 258.4615 - mae: 11.7675 - val_loss: 328.6169 - val_mae: 14.0737
Epoch 32/50
309/309 - 3s - 11ms/step - loss: 258.2429 - mae: 11.8192 - val_loss: 398.0578 - val_mae: 15.2909
Epoch 33/50
309/309 - 5s - 15ms/step - loss: 254.1084 - mae: 11.7462 - val_loss: 342.7823 - val_mae: 14.0005
Epoch 34/50
309/309 - 5s - 16ms/step - loss: 253.1110 - mae: 11.6709 - val_loss: 339.6804 - val_mae: 13.9480
Epoch 35/50
309/309 - 5s - 16ms/step - loss: 254.1585 - mae: 11.7445 - val_loss: 348.6435 - val_mae: 13.9557
Epoch 36/50
309/309 - 3s - 8ms/step - loss: 251.8003 - mae: 11.6331 - val_loss: 312.5370 - val_mae: 13.2357
Epoch 37/50
309/309 - 3s - 8ms/step - loss: 243.7687 - mae: 11.4327 - val_loss: 290.4431 - val_mae: 12.7106
Epoch 38/50
309/309 - 3s - 11ms/step - loss: 246.0332 - mae: 11.4871 - val_loss: 345.1331 - val_mae: 14.1198
Epoch 39/50
309/309 - 4s - 14ms/step - loss: 243.1211 - mae: 11.4124 - val_loss: 334.4877 - val_mae: 13.7050
Epoch 40/50
309/309 - 5s - 16ms/step - loss: 244.6160 - mae: 11.4660 - val_loss: 317.8849 - val_mae: 13.1994
Epoch 41/50
309/309 - 5s - 16ms/step - loss: 244.1124 - mae: 11.4592 - val_loss: 356.7004 - val_mae: 14.0812
Epoch 42/50
309/309 - 3s - 8ms/step - loss: 238.6346 - mae: 11.3420 - val_loss: 333.4558 - val_mae: 13.6134
Epoch 43/50
309/309 - 3s - 8ms/step - loss: 235.6059 - mae: 11.2496 - val_loss: 364.9118 - val_mae: 14.1815
Epoch 44/50
309/309 - 3s - 11ms/step - loss: 242.7090 - mae: 11.4008 - val_loss: 345.8574 - val_mae: 13.7915
Epoch 45/50
309/309 - 2s - 8ms/step - loss: 240.6930 - mae: 11.3286 - val_loss: 320.7698 - val_mae: 13.2660
Epoch 46/50
309/309 - 3s - 8ms/step - loss: 237.5101 - mae: 11.2784 - val_loss: 286.9190 - val_mae: 12.5257
Epoch 47/50
309/309 - 3s - 8ms/step - loss: 239.5587 - mae: 11.2874 - val_loss: 291.6729 - val_mae: 12.6974
Epoch 48/50
309/309 - 3s - 9ms/step - loss: 237.7989 - mae: 11.2335 - val_loss: 324.9650 - val_mae: 13.3028
Epoch 49/50

309/309 - 5s - 16ms/step - loss: 230.7903 - mae: 11.0972 - val_loss: 306.8300 - val_mae: 12.6764
Epoch 50/50
309/309 - 2s - 8ms/step - loss: 233.4579 - mae: 11.1828 - val_loss: 348.7077 - val_mae: 13.7577

```
Training time: 2.88 minutes
MSE: 274.1943
MAE: 12.1496
168/168 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
RMSE: 16.5588
log RMSE: 1.2694
R² Score: 0.8560
```





با توجه به نمودارهای نمودارهایMAE مشاهده می‌شود افت شدید در ۵ ایپاک اول، سپس تثبیت در مقادیر پایین دیده می‌شود. هم در داده‌های آموزش و هم اعتبارسنجی، مدل به خوبی همگرا شده است. نمودارها کاملاً پایدار و بدون نوسان شدید هستند که نشانه‌ای از generalization خوب مدل است.

باتوجه به نمودار RUL واقعی و پیش‌بینی شده مشاهده می‌شود نقاط پیش‌بینی‌شده بسیار نزدیک به خط ایده‌آل قرار گرفته‌اند و پیش‌بینی در تمام بازه‌های RUL نسبتاً دقیق است. در کل نسبت به CNN ، دقت پیش‌بینی بالاتر و پراکندگی کمتر است.

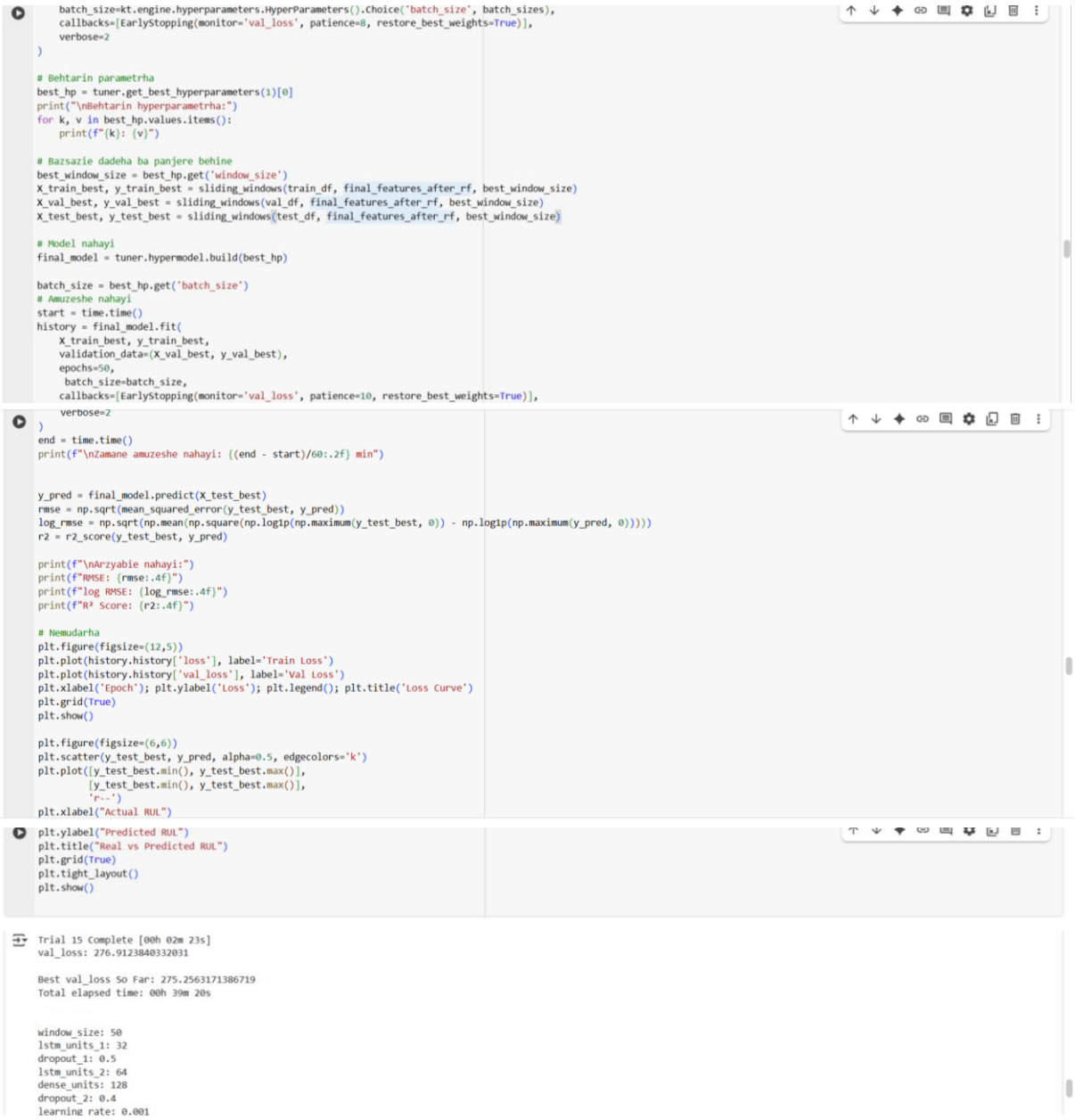
تنظیم ابرپارامترها

در این بخش از کد زیر استفاده شده است.

```python
import keras_tuner as kt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import numpy as np
import time

def sliding_windows(df, feature_cols, window_size=30, max_rul=130):
    X_windows, y_labels = [], []
    for unit_number, unit_df in df.groupby('unit'):
        unit_df = unit_df.sort_values('cycle').reset_index(drop=True)
        if len(unit_df) < window_size:
            continue
        for i in range(len(unit_df) - window_size + 1):
            window = unit_df.iloc[i:i+window_size]
            X_window = window[feature_cols].values
            rul_value = window.iloc[-1]['RUL']
            rul_capped = min(rul_value, max_rul)
            X_windows.append(X_window)
            y_labels.append(rul_capped)
    return np.array(X_windows), np.array(y_labels)


def build_model(hp):
```

```python
    window_size = hp.Int('window_size', min_value=20, max_value=50, step=5)
    batch_size = hp.Choice('batch_size', [32, 64, 128])


    X_train, y_train = sliding_windows(train_df, final_features_after_rf, window_size)
    X_val, y_val = sliding_windows(val_df, final_features_after_rf, window_size)

    model = Sequential()
    # Tedad vahedhaye LSTM
    model.add(LSTM(units=hp.Int('lstm_units_1', 32, 128, step=32), return_sequences=True,
                   input_shape=(window_size, len(final_features_after_rf))))
    # Nerkhe Dropout
    model.add(Dropout(hp.Float('dropout_1', 0.1, 0.5, step=0.1)))

    model.add(LSTM(units=hp.Int('lstm_units_2', 16, 64, step=16), return_sequences=False))

    model.add(Dense(units=hp.Int('dense_units', 32, 128, step=32), activation='relu'))
    model.add(Dropout(hp.Float('dropout_2', 0.1, 0.5, step=0.1)))
    model.add(Dense(1))

    # Nerkhe yadgiri
    lr = hp.Choice('learning_rate', [1e-2, 1e-3, 1e-4])

    # Behinesaz
    optimizer_name = hp.Choice('optimizer', ['adam', 'rmsprop'])
    optimizer = Adam(learning_rate=lr) if optimizer_name == 'adam' else RMSprop(learning_rate=lr)

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

```python
    # Zakhire dadeha baraye amuzesh kharej az tuner
    model.X_train = X_train
    model.y_train = y_train
    model.X_val = X_val
    model.y_val = y_val

    return model

tuner = kt.RandomSearch(
    build_model,
    objective='val_loss',
    max_trials=15,
    directory='kerastuner_logs',
    project_name='lstm_full_tuning'
)


X_start, y_start = sliding_windows(train_df, final_features_after_rf, window_size=30)
X_val_start, y_val_start = sliding_windows(val_df, final_features_after_rf, window_size=30)


batch_sizes = [16, 32, 64]

# Ejraye tuner
tuner.search(
    x=X_start,
    y=y_start,
    validation_data=(X_val_start, y_val_start),
    epochs=40,
```



```
    batch_size=kt.engine.hyperparameters.HyperParameters().Choice('batch_size', batch_sizes),
    callbacks=[EarlyStopping(monitor='val_loss', patience=8, restore_best_weights=True)],
    verbose=2
)

# Behtarin parametrha
best_hp = tuner.get_best_hyperparameters(1)[0]
print("\nBehtarin hyperparametrha:")
for k, v in best_hp.values.items():
    print(f"{k}: {v}")

# Bazsazie dadeha ba panjere behine
best_window_size = best_hp.get('window_size')
X_train_best, y_train_best = sliding_windows(train_df, final_features_after_rf, best_window_size)
X_val_best, y_val_best = sliding_windows(val_df, final_features_after_rf, best_window_size)
X_test_best, y_test_best = sliding_windows(test_df, final_features_after_rf, best_window_size)

# Model nahayi
final_model = tuner.hypermodel.build(best_hp)

batch_size = best_hp.get('batch_size')
# Amuzeshe nahayi
start = time.time()
history = final_model.fit(
    X_train_best, y_train_best,
    validation_data=(X_val_best, y_val_best),
    epochs=50,
     batch_size=batch_size,
    callbacks=[EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)],
    verbose=2
)
end = time.time()
print(f"\nZamane amuzeshe nahayi: {(end - start)/60:.2f} min")


y_pred = final_model.predict(X_test_best)
rmse = np.sqrt(mean_squared_error(y_test_best, y_pred))
log_rmse = np.sqrt(np.mean(np.square(np.log1p(np.maximum(y_test_best, 0)) - np.log1p(np.maximum(y_pred, 0)))))
r2 = r2_score(y_test_best, y_pred)

print(f"\nArzyabie nahayi:")
print(f"RMSE: {rmse:.4f}")
print(f"log RMSE: {log_rmse:.4f}")
print(f"R² Score: {r2:.4f}")

# Nemudarha
plt.figure(figsize=(12,5))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.xlabel('Epoch'); plt.ylabel('Loss'); plt.legend(); plt.title('Loss Curve')
plt.grid(True)
plt.show()

plt.figure(figsize=(6,6))
plt.scatter(y_test_best, y_pred, alpha=0.5, edgecolors='k')
plt.plot([y_test_best.min(), y_test_best.max()],
         [y_test_best.min(), y_test_best.max()],
         'r--')
plt.xlabel("Actual RUL")
plt.ylabel("Predicted RUL")
plt.title("Real vs Predicted RUL")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Trial 15 Complete [00h 02m 23s]
val_loss: 276.9123840332031

Best val_loss So Far: 275.2563171386719
Total elapsed time: 00h 39m 20s


window_size: 50
lstm_units_1: 32
dropout_1: 0.5
lstm_units_2: 64
dense_units: 128
dropout_2: 0.4
learning rate: 0.001
```

پیاده‌سازی مدل ترکیبی CNN+LSTM

در این بخش نیز ورودی مدل دارای شکل دو مدل قبل می‌باشد، که در این تمرین برابر (30, 10) می‌باشد (۳۰ سیکل متوالی از ۱۰ ویژگی منتخب). معماری مدل نیز دقیقا به‌صورت گفته شده در صورت سوال پیاده‌سازی شده است. تمام شرایط دیگر نیز مانند مدل‌های قبلی می‌باشد. کد مورد استفاده در این بخش بصورت زیر می‌باشد:

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, LSTM, Dropout, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from sklearn.metrics import mean_squared_error
import numpy as np
import matplotlib.pyplot as plt

# CNN + LSTM
input_shape = X_train_windows.shape[1:]

model_cnn_lstm = Sequential([
    Conv1D(filters=64, kernel_size=3, activation='relu', padding='same', input_shape=input_shape),
    MaxPooling1D(pool_size=2),
    LSTM(100, return_sequences=False),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dense(1)
])

model_cnn_lstm.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='mse',
    metrics=['mae']
)

model_cnn_lstm.summary()
```

```python
# # Callbacks
# early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
# reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6, verbose=1)

start_time = time.time()

history_cnn_lstm = model_cnn_lstm.fit(
    X_train_windows, y_train_windows,
    validation_data=(X_val_windows, y_val_windows),
    batch_size=32,
    epochs=50,
    callbacks=[],
    verbose=2
)
end_time = time.time()
print(f"Training time: {(end_time - start_time)/60:.2f} minutes")

eval_results = model_cnn_lstm.evaluate(X_test_windows, y_test_windows, verbose=0)
print(f"MSE: {eval_results[0]:.4f}")
print(f"MAE: {eval_results[1]:.4f}")

y_pred = model_cnn_lstm.predict(X_test_windows)
rmse = np.sqrt(mean_squared_error(y_test_windows, y_pred))
y_pred_safe2 = np.maximum(y_pred, 0)
y_test_safe1 = np.maximum(y_test_windows, 0)

log_rmse = np.sqrt(np.mean(np.square(np.log1p(y_test_safe1) - np.log1p(y_pred_safe2))))
r2 = r2_score(y_test_windows, y_pred)
```

```python
print(f"RMSE: {rmse:.4f}")
print(f"log RMSE: {log_rmse:.4f}")
print(f"R² Score: {r2:.4f}")

# Nemudarha
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history_cnn_lstm.history['loss'], label='Train Loss (MSE)')
plt.plot(history_cnn_lstm.history['val_loss'], label='Validation Loss (MSE)')
plt.xlabel('Epoch'); plt.ylabel('Loss'); plt.title('Loss Curve - CNN+LSTM'); plt.legend()

plt.subplot(1,2,2)
plt.plot(history_cnn_lstm.history['mae'], label='Train MAE')
plt.plot(history_cnn_lstm.history['val_mae'], label='Validation MAE')
plt.xlabel('Epoch'); plt.ylabel('MAE'); plt.title('MAE Curve - CNN+LSTM'); plt.legend()

plt.tight_layout()
plt.show()

# Nemudare vagheyi va pishbini shode
plt.figure(figsize=(6,6))
plt.scatter(y_test_windows, y_pred, alpha=0.5, edgecolors='k')
plt.plot([y_test_windows.min(), y_test_windows.max()],
         [y_test_windows.min(), y_test_windows.max()],
         'r--', lw=2)
plt.xlabel("Meghdare vagheyie RUL")
plt.ylabel("Meghdare RUL")
plt.title("Real vs Predicted RUL")
plt.grid(True)
```

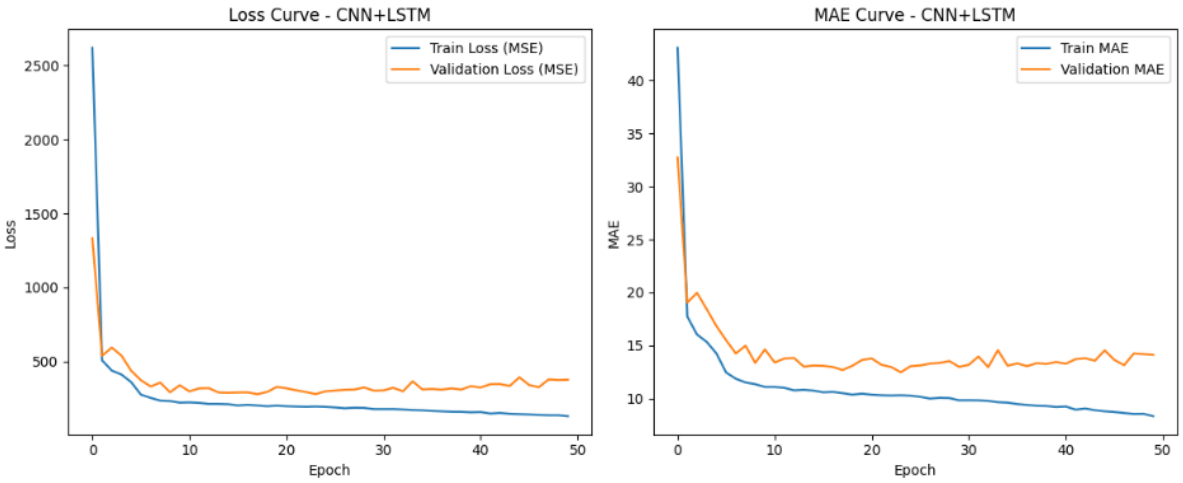
```python
plt.tight_layout()
plt.show()
```

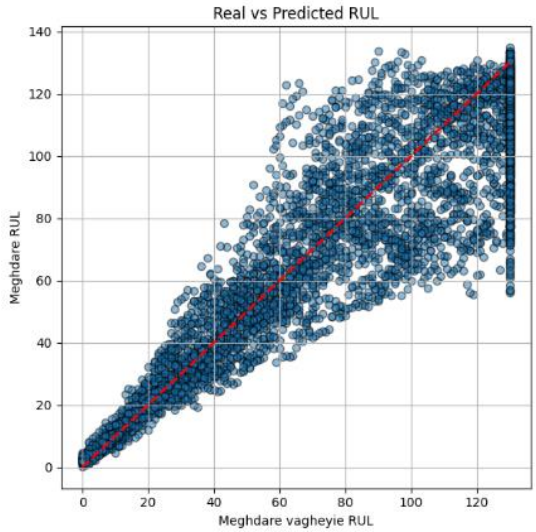
Epoch 1/50
309/309 - 4s - 13ms/step - loss: 2618.3308 - mae: 43.0691 - val_loss: 1332.2490 - val_mae: 32.7159
Epoch 2/50
309/309 - 2s - 7ms/step - loss: 509.0211 - mae: 17.7358 - val_loss: 539.5680 - val_mae: 19.0562
Epoch 3/50
309/309 - 3s - 8ms/step - loss: 439.6791 - mae: 16.0428 - val_loss: 595.0062 - val_mae: 19.9731
Epoch 4/50
309/309 - 2s - 7ms/step - loss: 412.7113 - mae: 15.3466 - val_loss: 540.5478 - val_mae: 18.4320
Epoch 5/50
309/309 - 2s - 6ms/step - loss: 361.6722 - mae: 14.2602 - val_loss: 438.5044 - val_mae: 16.8423
Epoch 6/50
309/309 - 2s - 8ms/step - loss: 277.3334 - mae: 12.4621 - val_loss: 374.4758 - val_mae: 15.5056
Epoch 7/50
309/309 - 2s - 8ms/step - loss: 255.3806 - mae: 11.8736 - val_loss: 332.5313 - val_mae: 14.2547

Epoch 8/50
309/309 - 2s - 7ms/step - loss: 236.6844 - mae: 11.5241 - val_loss: 358.3727 - val_mae: 14.9899
Epoch 9/50
309/309 - 2s - 6ms/step - loss: 233.9507 - mae: 11.3596 - val_loss: 293.2041 - val_mae: 13.3694
Epoch 10/50
309/309 - 3s - 8ms/step - loss: 223.2796 - mae: 11.1000 - val_loss: 340.6988 - val_mae: 14.6376
Epoch 11/50
309/309 - 3s - 8ms/step - loss: 224.9339 - mae: 11.1002 - val_loss: 299.0411 - val_mae: 13.4105
Epoch 12/50
309/309 - 2s - 7ms/step - loss: 221.7163 - mae: 11.0199 - val_loss: 319.5276 - val_mae: 13.7773
Epoch 13/50
309/309 - 3s - 9ms/step - loss: 214.5772 - mae: 10.7807 - val_loss: 320.9280 - val_mae: 13.8293
Epoch 14/50
309/309 - 2s - 6ms/step - loss: 214.6770 - mae: 10.8321 - val_loss: 291.7893 - val_mae: 13.0157
Epoch 15/50
309/309 - 2s - 8ms/step - loss: 212.6980 - mae: 10.7557 - val_loss: 289.7533 - val_mae: 13.1180
Epoch 16/50
309/309 - 2s - 6ms/step - loss: 204.1898 - mae: 10.5946 - val_loss: 292.0098 - val_mae: 13.0941
Epoch 17/50
309/309 - 3s - 9ms/step - loss: 208.5473 - mae: 10.6332 - val_loss: 292.4539 - val_mae: 12.9773
Epoch 18/50
309/309 - 2s - 8ms/step - loss: 204.0630 - mae: 10.5136 - val_loss: 280.4218 - val_mae: 12.6980
Epoch 19/50
309/309 - 2s - 7ms/step - loss: 199.0636 - mae: 10.3754 - val_loss: 295.2343 - val_mae: 13.0929
Epoch 20/50
309/309 - 2s - 6ms/step - loss: 203.1472 - mae: 10.4672 - val_loss: 329.4050 - val_mae: 13.6415
Epoch 21/50
309/309 - 3s - 8ms/step - loss: 199.2770 - mae: 10.3719 - val_loss: 320.1825 - val_mae: 13.7886
Epoch 22/50
309/309 - 3s - 9ms/step - loss: 196.8971 - mae: 10.3171 - val_loss: 306.5001 - val_mae: 13.1902
Epoch 23/50
309/309 - 2s - 7ms/step - loss: 195.3164 - mae: 10.2939 - val_loss: 294.6824 - val_mae: 12.9814
Epoch 24/50
309/309 - 2s - 8ms/step - loss: 196.9901 - mae: 10.3168 - val_loss: 281.0188 - val_mae: 12.4839
Epoch 25/50
309/309 - 2s - 8ms/step - loss: 194.9863 - mae: 10.2774 - val_loss: 299.4315 - val_mae: 13.0598
Epoch 26/50
309/309 - 2s - 6ms/step - loss: 190.7790 - mae: 10.1698 - val_loss: 304.6839 - val_mae: 13.1269
Epoch 27/50
309/309 - 3s - 10ms/step - loss: 185.5423 - mae: 10.0034 - val_loss: 310.0092 - val_mae: 13.2946
Epoch 28/50
309/309 - 2s - 7ms/step - loss: 188.8601 - mae: 10.0754 - val_loss: 311.8035 - val_mae: 13.3658
Epoch 29/50
309/309 - 2s - 7ms/step - loss: 187.7646 - mae: 10.0521 - val_loss: 325.7608 - val_mae: 13.5388
Epoch 30/50
309/309 - 2s - 6ms/step - loss: 179.7982 - mae: 9.8425 - val_loss: 304.2903 - val_mae: 12.9927
Epoch 31/50
309/309 - 2s - 8ms/step - loss: 179.5487 - mae: 9.8488 - val_loss: 305.9499 - val_mae: 13.1888
Epoch 32/50
309/309 - 3s - 9ms/step - loss: 179.7593 - mae: 9.8365 - val_loss: 323.5071 - val_mae: 13.9683
Epoch 33/50
309/309 - 2s - 8ms/step - loss: 176.9674 - mae: 9.7853 - val_loss: 299.7350 - val_mae: 12.9661
Epoch 34/50
309/309 - 2s - 6ms/step - loss: 173.4926 - mae: 9.6653 - val_loss: 367.2130 - val_mae: 14.5709
Epoch 35/50
309/309 - 2s - 6ms/step - loss: 171.3853 - mae: 9.6214 - val_loss: 312.4105 - val_mae: 13.1040
Epoch 36/50
309/309 - 3s - 8ms/step - loss: 167.3157 - mae: 9.4893 - val_loss: 315.5608 - val_mae: 13.3191
Epoch 37/50
309/309 - 3s - 9ms/step - loss: 164.6430 - mae: 9.4032 - val_loss: 311.1996 - val_mae: 13.0589
Epoch 38/50

309/309 - 2s - 8ms/step - loss: 161.8327 - mae: 9.3307 - val_loss: 319.5019 - val_mae: 13.3417
Epoch 39/50
309/309 - 2s - 7ms/step - loss: 161.2951 - mae: 9.3070 - val_loss: 311.5931 - val_mae: 13.2678
Epoch 40/50
309/309 - 3s - 9ms/step - loss: 157.9538 - mae: 9.2118 - val_loss: 334.4437 - val_mae: 13.4539
Epoch 41/50
309/309 - 2s - 8ms/step - loss: 159.7740 - mae: 9.2564 - val_loss: 325.1090 - val_mae: 13.2885
Epoch 42/50
309/309 - 3s - 9ms/step - loss: 149.6931 - mae: 8.9627 - val_loss: 347.9361 - val_mae: 13.7217
Epoch 43/50
309/309 - 2s - 7ms/step - loss: 153.6581 - mae: 9.0686 - val_loss: 348.4548 - val_mae: 13.8016
Epoch 44/50
309/309 - 2s - 6ms/step - loss: 147.9862 - mae: 8.9074 - val_loss: 335.8564 - val_mae: 13.5772
Epoch 45/50
309/309 - 2s - 6ms/step - loss: 145.5624 - mae: 8.8059 - val_loss: 394.3321 - val_mae: 14.5495
Epoch 46/50
309/309 - 3s - 8ms/step - loss: 143.3685 - mae: 8.7402 - val_loss: 342.2191 - val_mae: 13.6533
Epoch 47/50
309/309 - 2s - 6ms/step - loss: 140.3242 - mae: 8.6440 - val_loss: 327.1191 - val_mae: 13.1486
Epoch 48/50
309/309 - 2s - 8ms/step - loss: 138.1750 - mae: 8.5561 - val_loss: 379.7982 - val_mae: 14.2522
Epoch 49/50
309/309 - 2s - 6ms/step - loss: 138.1105 - mae: 8.5683 - val_loss: 376.0762 - val_mae: 14.1977
Epoch 50/50
309/309 - 2s - 6ms/step - loss: 131.7626 - mae: 8.3441 - val_loss: 377.9645 - val_mae: 14.1292

```
Training time: 1.98 minutes
MSE: 316.1653
MAE: 12.6833
168/168 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
RMSE: 17.7810
log RMSE: 1.2154
R² Score: 0.8339
```





در نمودار LOSS و MAE افت سریع در ابتدا، سپس کاهش یکنواخت مشاهده می‌شود. عملکرد مدل در validation تقریباً پایدار است اما مقدار آن از train بیشتر است (فاصله بین دو منحنی زیاد نیست). بطور کلی این مدل تعادل خوبی بین CNN و LSTM ایجاد کرده است.

در نمودار RUL پیش‌بینی شده و واقعی، شبیه به مدلLSTM نقاط پیش‌بینی شده نزدیک به خط ایده‌آل هستند. در بخش‌هایی از نمودار مربوط به مدل ترکیبی CNN+LSTM نسبت به LSTM پراکندگی کمتری دیده می‌شود. بطور کلی دقت پیش‌بینی عالی و توزیع متقارن اطراف خط قرمز مشاهده می‌شود.

برای بخش LSTM+CNN از کد زیر استفاده شده است. لازم به ذکر است در این حالت با تغییر ترتیب اولیه با ارور مواجه می‌شویم زیرا آرایه y_predسه‌بعدی (3D) است، ولی تابع mean_squared_errorاز sklearnفقط آرایه‌های دوبعدی (2D) یا کمتر را قبول می‌کند. زیرا زمانی که در مدل LSTM + CNN، لایه‌ی LSTM مقدار return_sequences=True دارد و بعد از آن لایه کانولوشن استفاده می‌شود، خروجی نهایی مدل ممکن است شکل سه بعدی باقی بماند. در نتیجه، model.predict() نیز خروجی سه‌بعدی می‌دهد. بنابراین برای تبدیل خروجی به شکل مناسب، باید y_predرا به یک بردار ۱بعدی فشرده کنیم. نتایج در زیر قابل مشاهده می‌باشند.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, LSTM, Dropout, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from sklearn.metrics import mean_squared_error
import numpy as np
import matplotlib.pyplot as plt

# LSTM + CNN
input_shape = X_train_windows.shape[1:]

model_cnn_lstm = Sequential([
    Conv1D(filters=64, kernel_size=3, activation='relu', padding='same', input_shape=input_shape),
    MaxPooling1D(pool_size=2),
    LSTM(100, return_sequences=False),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dense(1)  # خروجی RUL
])

model_cnn_lstm.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='mse',
    metrics=['mae']
)

model_cnn_lstm.summary()
```

```python
start_time = time.time()

history_cnn_lstm = model_cnn_lstm.fit(
    X_train_windows, y_train_windows,
    validation_data=(X_val_windows, y_val_windows),
    batch_size=32,
    epochs=50,
    callbacks=[],
    verbose=2
)
end_time = time.time()
print(f"Training time: {(end_time - start_time)/60:.2f} minutes")

eval_results = model_cnn_lstm.evaluate(X_test_windows, y_test_windows, verbose=0)
print(f"\nMSE: {eval_results[0]:.4f}")
print(f"MAE: {eval_results[1]:.4f}")

y_pred = model_cnn_lstm.predict(X_test_windows)
rmse = np.sqrt(mean_squared_error(y_test_windows, y_pred))
r2 = r2_score(y_test_windows, y_pred)
y_pred_safe2 = np.maximum(y_pred, 0)
y_test_safe1 = np.maximum(y_test_windows, 0)

log_rmse = np.sqrt(np.mean(np.square(np.log1p(y_test_safe1) - np.log1p(y_pred_safe2))))

print(f"log RMSE: {log_rmse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"R² (Coefficient of Determination): {r2:.4f}")
```

```python
# Nemudarha
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history_cnn_lstm.history['loss'], label='Train Loss (MSE)')
plt.plot(history_cnn_lstm.history['val_loss'], label='Validation Loss (MSE)')
plt.xlabel('Epoch'); plt.ylabel('Loss'); plt.title('Loss Curve - CNN+LSTM'); plt.legend()

plt.subplot(1,2,2)
plt.plot(history_cnn_lstm.history['mae'], label='Train MAE')
plt.plot(history_cnn_lstm.history['val_mae'], label='Validation MAE')
plt.xlabel('Epoch'); plt.ylabel('MAE'); plt.title('MAE Curve - CNN+LSTM'); plt.legend()

plt.tight_layout()
plt.show()


# Nemudare vagheyi dar barabare pishbini shode
plt.figure(figsize=(6,6))
plt.scatter(y_test_windows, y_pred, alpha=0.5, edgecolors='k')
plt.plot([y_test_windows.min(), y_test_windows.max()],
         [y_test_windows.min(), y_test_windows.max()],
         'r--', lw=2)
plt.xlabel("Meghdare vagheyie RUL")
plt.ylabel("Meghdare RUL")
plt.title("Real vs Predicted RUL")
plt.grid(True)
plt.tight_layout()
plt.show()
```

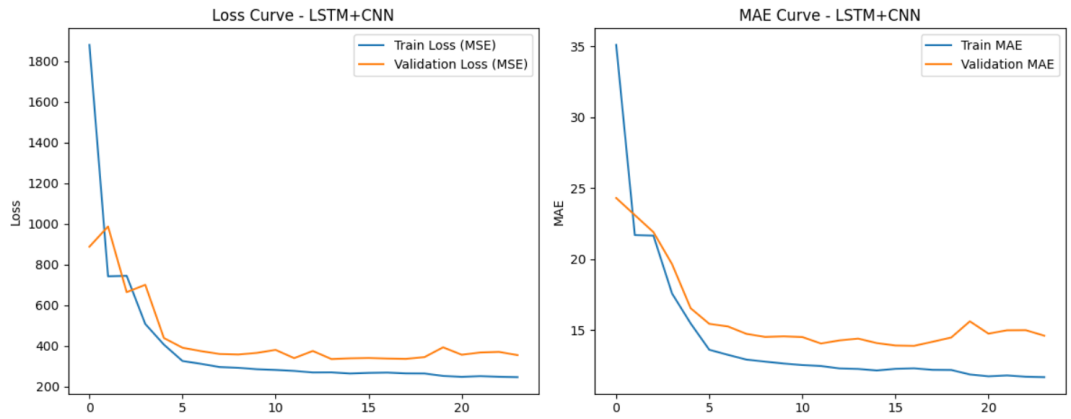
```
Epoch 1/50
309/309 - 4s - 13ms/step - loss: 1880.7828 - mae: 35.1136 - val_loss: 888.2714 - val_mae: 24.3161 - learning_rate: 1.0000e-03
Epoch 2/50
309/309 - 2s - 8ms/step - loss: 742.7701 - mae: 21.7009 - val_loss: 987.7303 - val_mae: 23.1104 - learning_rate: 1.0000e-03
Epoch 3/50
309/309 - 2s - 8ms/step - loss: 745.0851 - mae: 21.6627 - val_loss: 664.8163 - val_mae: 21.9099 - learning_rate: 1.0000e-03
Epoch 4/50
309/309 - 3s - 9ms/step - loss: 508.2572 - mae: 17.5806 - val_loss: 700.8996 - val_mae: 19.6527 - learning_rate: 1.0000e-03
Epoch 5/50
309/309 - 2s - 7ms/step - loss: 406.8119 - mae: 15.4694 - val_loss: 439.2726 - val_mae: 16.5421 - learning_rate: 1.0000e-03
Epoch 6/50
309/309 - 2s - 7ms/step - loss: 326.1360 - mae: 13.6131 - val_loss: 391.4900 - val_mae: 15.4416 - learning_rate: 1.0000e-03
Epoch 7/50
309/309 - 2s - 7ms/step - loss: 312.1475 - mae: 13.2562 - val_loss: 374.6626 - val_mae: 15.2578 - learning_rate: 1.0000e-03
Epoch 8/50
309/309 - 3s - 8ms/step - loss: 296.3859 - mae: 12.9185 - val_loss: 360.5879 - val_mae: 14.7337 - learning_rate: 1.0000e-03
Epoch 9/50
309/309 - 3s - 9ms/step - loss: 292.6331 - mae: 12.7769 - val_loss: 358.1679 - val_mae: 14.5185 - learning_rate: 1.0000e-03
Epoch 10/50
309/309 - 4s - 14ms/step - loss: 285.7373 - mae: 12.6421 - val_loss: 366.0318 - val_mae: 14.5584 - learning_rate: 1.0000e-03
Epoch 11/50
309/309 - 2s - 8ms/step - loss: 282.1852 - mae: 12.5342 - val_loss: 380.8348 - val_mae: 14.5106 - learning_rate: 1.0000e-03
Epoch 12/50
309/309 - 3s - 8ms/step - loss: 277.3513 - mae: 12.4682 - val_loss: 340.2207 - val_mae: 14.0541 - learning_rate: 1.0000e-03
Epoch 13/50
```

309/309 - 3s - 11ms/step - loss: 269.8197 - mae: 12.2956 - val_loss: 375.5051 - val_mae: 14.2746 - learning_rate: 1.0000e-03
Epoch 14/50
309/309 - 2s - 7ms/step - loss: 270.2611 - mae: 12.2614 - val_loss: 335.8754 - val_mae: 14.4005 - learning_rate: 1.0000e-03
Epoch 15/50
309/309 - 2s - 8ms/step - loss: 264.6187 - mae: 12.1527 - val_loss: 339.3054 - val_mae: 14.0844 - learning_rate: 1.0000e-03
Epoch 16/50
309/309 - 2s - 7ms/step - loss: 267.6993 - mae: 12.2693 - val_loss: 340.9465 - val_mae: 13.9130 - learning_rate: 1.0000e-03
Epoch 17/50
309/309 - 3s - 8ms/step - loss: 269.2409 - mae: 12.3026 - val_loss: 338.2405 - val_mae: 13.8889 - learning_rate: 1.0000e-03
Epoch 18/50
309/309 - 3s - 9ms/step - loss: 265.1464 - mae: 12.1991 - val_loss: 336.7134 - val_mae: 14.1780 - learning_rate: 1.0000e-03
Epoch 19/50

Epoch 19: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
309/309 - 2s - 7ms/step - loss: 264.6819 - mae: 12.1877 - val_loss: 345.1955 - val_mae: 14.4785 - learning_rate: 1.0000e-03
Epoch 20/50
309/309 - 2s - 8ms/step - loss: 252.7438 - mae: 11.8716 - val_loss: 393.4560 - val_mae: 15.6189 - learning_rate: 5.0000e-04
Epoch 21/50
309/309 - 3s - 9ms/step - loss: 248.1903 - mae: 11.7401 - val_loss: 357.0158 - val_mae: 14.7505 - learning_rate: 5.0000e-04
Epoch 22/50
309/309 - 2s - 8ms/step - loss: 251.6751 - mae: 11.8019 - val_loss: 367.9651 - val_mae: 14.9881 - learning_rate: 5.0000e-04
Epoch 23/50
309/309 - 3s - 9ms/step - loss: 248.5506 - mae: 11.7124 - val_loss: 370.8950 - val_mae: 14.9963 - learning_rate: 5.0000e-04
Epoch 24/50

Epoch 24: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
309/309 - 2s - 7ms/step - loss: 246.8369 - mae: 11.6835 - val_loss: 355.1582 - val_mae: 14.6049 - learning_rate: 5.0000e-04

```
Test Loss (MSE): 272.2599
Test MAE: 12.8561
168/168 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
Test RMSE: 16.5003
Test log RMSE: 2.8034
```



پیاده‌سازی معماری LSTM+Attention

در این بخش از کد زیر استفاده شده است. در این کد یک لایه Attention طراحی شده است که روی خروجی‌های زمانی LSTM کار می‌کند.(return_sequences=True) و وزن توجه (Attention weights) را با استفاده از softmax محاسبه می‌کند. سپس ضرب وزندار بین attention و خروجی LSTM گرفته می‌شود. در نهایت با K.sum ترکیب (aggregate) می‌شود.

بطور کلی و خلاصه ابتدا معماری مدل با استفاده از لایه Input و یک لایه LSTM با ۱۰۰ واحد و خروجی در تمامی گام‌های زمانی طراحی شد. سپس لایه Attention سفارشی که به‌صورت کلاس جداگانه تعریف شده بود، بر خروجی‌های زمانی LSTM اعمال گردید. این لایه وزن‌هایی قابل یادگیری را برای هر گام زمانی محاسبه کرده و با استفاده از این وزن‌ها، ترکیب وزن‌دار شده‌ای از خروجی‌های LSTM تولید می‌کند. این ترکیب نهایی در واقع تجمیعی از اطلاعات مهم‌تر در دنباله زمانی ورودی است.

پس از لایه Attention، از لایه Dropout با نرخ ۳۰ درصد برای جلوگیری از overfitting استفاده شد. سپس، لایه‌ای کاملاً متصل (Dense) با ۶۴ نورون و تابع فعال‌سازی ReLU اضافه شد و نهایتاً، خروجی مدل از طریق یک لایه Dense با یک نورون به دست آمد.

مدل طراحی‌شده با استفاده از بهینه‌ساز Adam و نرخ یادگیری ۰٫۰۰۱ کامپایل شد. تابع خطای مورد استفاده Mean Squared Error (MSE) بوده و از میانگین خطای مطلق (MAE) نیز به‌عنوان معیار ارزیابی استفاده شد. آموزش مدل روی داده‌های آموزشی پنجره‌بندی‌شده انجام گرفت و از داده‌های اعتبارسنجی برای پایش عملکرد مدل استفاده شد. فرآیند آموزش به مدت ۵۰ دوره و با اندازه دسته (batch size) برابر ۳۲ انجام شد.

```python
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, Layer
from tensorflow.keras.layers import Permute, Multiply, Lambda, RepeatVector
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from sklearn.metrics import mean_squared_error
import tensorflow.keras.backend as K
import matplotlib.pyplot as plt
import numpy as np

# Laye Attention
class Attention(Layer):
    def __init__(self, **kwargs):
        super(Attention, self).__init__(**kwargs)

    def build(self, input_shape):
        self.W = self.add_weight(name='att_weight', shape=(input_shape[-1], 1),
                                 initializer='random_normal', trainable=True)
        self.b = self.add_weight(name='att_bias', shape=(input_shape[1], 1),
                                 initializer='zeros', trainable=True)
        super(Attention, self).build(input_shape)

    def call(self, x):
        e = K.tanh(K.dot(x, self.W) + self.b)  # Energy attention
        a = K.softmax(e, axis=1)               # normalsazie vaznha
        output = x * a                         # Vazndehi be khurujihaye LSTM
        return K.sum(output, axis=1)           # Tajmi'e vazndar
```

```python
    def compute_output_shape(self, input_shape):
        return (input_shape[0], input_shape[-1])

# LSTM + Attention
input_shape = X_train_windows.shape[1:]
inputs = Input(shape=input_shape)

x = LSTM(100, return_sequences=True)(inputs)  # Khurujie gamhaye zamani
x = Attention()(x)                            # Laye attention ruye khurujihaye zamani
x = Dropout(0.3)(x)
x = Dense(64, activation='relu')(x)
outputs = Dense(1)(x)

model_lstm_attention = Model(inputs, outputs)

model_lstm_attention.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='mse',
    metrics=['mae']
)

model_lstm_attention.summary()

# # callbacks
# early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
# reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6, verbose=1)

start_time = time.time()
```

```
history_lstm_attention = model_lstm_attention.fit(
    X_train_windows, y_train_windows,
    validation_data=(X_val_windows, y_val_windows),
    batch_size=32,
    epochs=50,
    callbacks=[],
    verbose=2
)
end_time = time.time()
print(f"\nTraining time: {(end_time - start_time)/60:.2f} minutes")

eval_results = model_lstm_attention.evaluate(X_test_windows, y_test_windows, verbose=0)
print(f"MSE: {eval_results[0]:.4f}")
print(f"MAE: {eval_results[1]:.4f}")

y_pred = model_lstm_attention.predict(X_test_windows)
y_pred = y_pred.squeeze()

r2 = r2_score(y_test_windows, y_pred)


y_pred_safe2 = np.maximum(y_pred, 0)
y_test_safe1 = np.maximum(y_test_windows, 0)

log_rmse = np.sqrt(np.mean(np.square(np.log1p(y_test_safe1) - np.log1p(y_pred_safe2))))

rmse = np.sqrt(mean_squared_error(y_test_windows, y_pred))

print(f"RMSE: {rmse:.4f}")
```

```
print(f"log RMSE: {log_rmse:.4f}")
print(f"R² Score: {r2:.4f}")

# Nemudarha
plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
plt.plot(history_lstm_attention.history['loss'], label='Train Loss (MSE)')
plt.plot(history_lstm_attention.history['val_loss'], label='Validation Loss (MSE)')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss Curve - LSTM + Attention')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history_lstm_attention.history['mae'], label='Train MAE')
plt.plot(history_lstm_attention.history['val_mae'], label='Validation MAE')
plt.xlabel('Epoch')
plt.ylabel('MAE')
plt.title('MAE Curve - LSTM + Attention')
plt.legend()

plt.tight_layout()
plt.show()

# Nemudare vagheyi va pishbini shode
plt.figure(figsize=(6,6))
plt.scatter(y_test_windows, y_pred, alpha=0.5, edgecolors='k')
plt.plot([y_test_windows.min(), y_test_windows.max()],
```

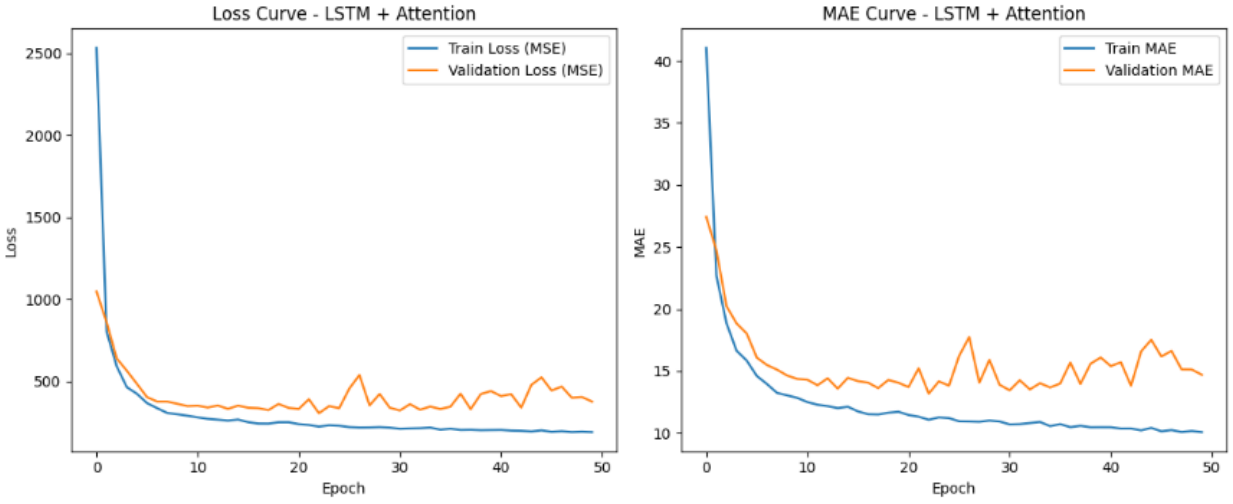
```
         [y_test_windows.min(), y_test_windows.max()],
         'r--', lw=2)
plt.xlabel("Meghdare vagheyie RUL")
plt.ylabel("Meghdare RUL")
plt.title("Real vs Predicted RUL")
plt.grid(True)
plt.tight_layout()
plt.show()
```

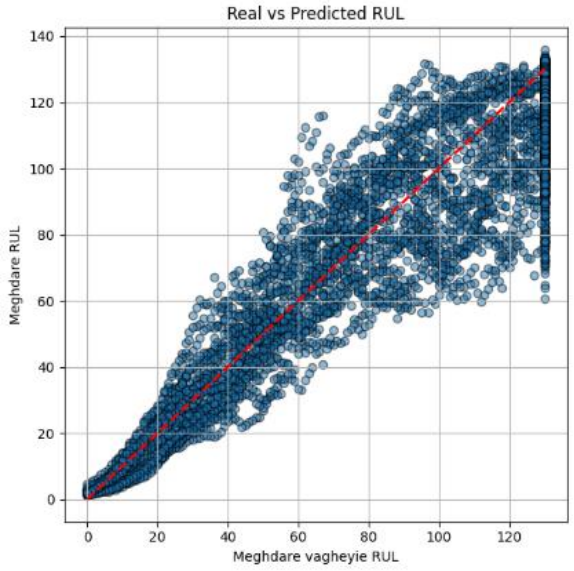
```
Epoch 1/50
309/309 - 4s - 13ms/step - loss: 2532.1130 - mae: 41.0396 - val_loss: 1047.2261 - val_mae: 27.4218
Epoch 2/50
309/309 - 2s - 8ms/step - loss: 802.1819 - mae: 22.7094 - val_loss: 859.6444 - val_mae: 24.7506
Epoch 3/50
309/309 - 2s - 8ms/step - loss: 590.4273 - mae: 18.8552 - val_loss: 638.7042 - val_mae: 20.2296
Epoch 4/50
309/309 - 2s - 7ms/step - loss: 463.3655 - mae: 16.6353 - val_loss: 563.6229 - val_mae: 18.8387
Epoch 5/50
309/309 - 2s - 6ms/step - loss: 422.9155 - mae: 15.8450 - val_loss: 483.4717 - val_mae: 18.0264
Epoch 6/50
309/309 - 3s - 8ms/step - loss: 366.1019 - mae: 14.6049 - val_loss: 402.4351 - val_mae: 16.0818
Epoch 7/50
309/309 - 3s - 9ms/step - loss: 335.2005 - mae: 13.9567 - val_loss: 375.6992 - val_mae: 15.4882
Epoch 8/50
309/309 - 3s - 8ms/step - loss: 305.8845 - mae: 13.2400 - val_loss: 375.5912 - val_mae: 15.1125
Epoch 9/50
309/309 - 2s - 6ms/step - loss: 298.1722 - mae: 13.0422 - val_loss: 361.8155 - val_mae: 14.6395
Epoch 10/50
309/309 - 2s - 7ms/step - loss: 289.0920 - mae: 12.8384 - val_loss: 347.9315 - val_mae: 14.3671
Epoch 11/50
309/309 - 3s - 8ms/step - loss: 279.1949 - mae: 12.5005 - val_loss: 350.4487 - val_mae: 14.3026
Epoch 12/50
309/309 - 2s - 7ms/step - loss: 270.2689 - mae: 12.2871 - val_loss: 339.4127 - val_mae: 13.8574
Epoch 13/50
309/309 - 3s - 9ms/step - loss: 264.9520 - mae: 12.1740 - val_loss: 351.5461 - val_mae: 14.4202
```

Epoch 14/50
309/309 - 2s - 7ms/step - loss: 259.4541 - mae: 12.0141 - val_loss: 331.1108 - val_mae: 13.5824
Epoch 15/50
309/309 - 2s - 7ms/step - loss: 265.6764 - mae: 12.1258 - val_loss: 350.7772 - val_mae: 14.4543
Epoch 16/50
309/309 - 2s - 6ms/step - loss: 249.9807 - mae: 11.7478 - val_loss: 337.9338 - val_mae: 14.1891
Epoch 17/50
309/309 - 2s - 7ms/step - loss: 241.2945 - mae: 11.5242 - val_loss: 335.2358 - val_mae: 14.0675
Epoch 18/50
309/309 - 3s - 8ms/step - loss: 240.7964 - mae: 11.5022 - val_loss: 323.4703 - val_mae: 13.6267
Epoch 19/50
309/309 - 5s - 15ms/step - loss: 249.4768 - mae: 11.6414 - val_loss: 361.0088 - val_mae: 14.2835
Epoch 20/50
309/309 - 2s - 8ms/step - loss: 249.6444 - mae: 11.7212 - val_loss: 337.0062 - val_mae: 14.0602
Epoch 21/50
309/309 - 3s - 8ms/step - loss: 237.9267 - mae: 11.4607 - val_loss: 330.3635 - val_mae: 13.7091
Epoch 22/50
309/309 - 3s - 8ms/step - loss: 232.8600 - mae: 11.3230 - val_loss: 389.8926 - val_mae: 15.2272
Epoch 23/50
309/309 - 5s - 15ms/step - loss: 222.8217 - mae: 11.0756 - val_loss: 304.3416 - val_mae: 13.1781
Epoch 24/50
309/309 - 3s - 8ms/step - loss: 231.1581 - mae: 11.2665 - val_loss: 347.8812 - val_mae: 14.1675
Epoch 25/50
309/309 - 2s - 6ms/step - loss: 228.1826 - mae: 11.2084 - val_loss: 335.1828 - val_mae: 13.8246
Epoch 26/50
309/309 - 3s - 10ms/step - loss: 219.6675 - mae: 10.9530 - val_loss: 454.7964 - val_mae: 16.2152
Epoch 27/50
309/309 - 2s - 7ms/step - loss: 217.0030 - mae: 10.9395 - val_loss: 537.7012 - val_mae: 17.7403
Epoch 28/50
309/309 - 2s - 7ms/step - loss: 217.5040 - mae: 10.9133 - val_loss: 352.0812 - val_mae: 14.0616
Epoch 29/50
309/309 - 2s - 6ms/step - loss: 219.5073 - mae: 11.0133 - val_loss: 422.0312 - val_mae: 15.9036
Epoch 30/50
309/309 - 2s - 6ms/step - loss: 216.3885 - mae: 10.9381 - val_loss: 338.1909 - val_mae: 13.9021
Epoch 31/50
309/309 - 2s - 8ms/step - loss: 209.5217 - mae: 10.7004 - val_loss: 320.9338 - val_mae: 13.4452
Epoch 32/50
309/309 - 2s - 8ms/step - loss: 211.7155 - mae: 10.7246 - val_loss: 359.9117 - val_mae: 14.2604
Epoch 33/50
309/309 - 2s - 7ms/step - loss: 213.2395 - mae: 10.8116 - val_loss: 325.7417 - val_mae: 13.5193
Epoch 34/50
309/309 - 2s - 6ms/step - loss: 217.0218 - mae: 10.8998 - val_loss: 345.3510 - val_mae: 14.0235
Epoch 35/50
309/309 - 3s - 8ms/step - loss: 204.2509 - mae: 10.5629 - val_loss: 330.0336 - val_mae: 13.6770
Epoch 36/50
309/309 - 3s - 10ms/step - loss: 209.9718 - mae: 10.7242 - val_loss: 344.5232 - val_mae: 13.9972
Epoch 37/50
309/309 - 2s - 7ms/step - loss: 202.5690 - mae: 10.4786 - val_loss: 423.6386 - val_mae: 15.6906
Epoch 38/50
309/309 - 2s - 6ms/step - loss: 203.6226 - mae: 10.5899 - val_loss: 329.0994 - val_mae: 13.9610
Epoch 39/50
309/309 - 3s - 9ms/step - loss: 201.0897 - mae: 10.4695 - val_loss: 421.2580 - val_mae: 15.5915
Epoch 40/50
309/309 - 2s - 8ms/step - loss: 201.9473 - mae: 10.4744 - val_loss: 439.7568 - val_mae: 16.0929
Epoch 41/50
309/309 - 2s - 8ms/step - loss: 202.6759 - mae: 10.4714 - val_loss: 409.0469 - val_mae: 15.4019
Epoch 42/50
309/309 - 2s - 7ms/step - loss: 198.5703 - mae: 10.3654 - val_loss: 421.0331 - val_mae: 15.7096
Epoch 43/50
309/309 - 2s - 7ms/step - loss: 197.1288 - mae: 10.3696 - val_loss: 338.5336 - val_mae: 13.8132
Epoch 44/50

309/309 - 2s - 6ms/step - loss: 193.3599 - mae: 10.2227 - val_loss: 477.1730 - val_mae: 16.5847
Epoch 45/50
309/309 - 3s - 8ms/step - loss: 199.9813 - mae: 10.4207 - val_loss: 523.4717 - val_mae: 17.5213
Epoch 46/50
309/309 - 2s - 8ms/step - loss: 190.9487 - mae: 10.1464 - val_loss: 442.7521 - val_mae: 16.1729
Epoch 47/50
309/309 - 2s - 7ms/step - loss: 194.7238 - mae: 10.2449 - val_loss: 467.0367 - val_mae: 16.6254
Epoch 48/50
309/309 - 2s - 6ms/step - loss: 189.5080 - mae: 10.0881 - val_loss: 398.9907 - val_mae: 15.1378
Epoch 49/50
309/309 - 3s - 8ms/step - loss: 191.9101 - mae: 10.1698 - val_loss: 403.2309 - val_mae: 15.1262
Epoch 50/50
309/309 - 3s - 8ms/step - loss: 189.2086 - mae: 10.0792 - val_loss: 375.6285 - val_mae: 14.7027

```
Training time: 2.06 minutes
MSE: 281.1054
MAE: 12.4965
168/168 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
RMSE: 16.7662
log RMSE: 0.2471
R² Score: 0.8523
```





در این مدل log RMSE بسیار پایین است یعنی مدل در مقادیر کوچک RUL بسیار دقیق است. دقت این مدل نیز فقط کمی

کمتر از LSTM است. بطورخلاصه در این مدل، زمان آموزش کمتر از LSTM است، درحالی که دقت آن تقریباً یکسان یا حتی کمی بهتر درRUL های پایین است. MAE نیز کمتر از CNN و CNN+LSTM و تقریباً برابر با LSTM است. بعنوان نتیجه‌گیری می‌توان گفت این مدل بین دقت و سرعت، توازن خوبی دارد، در حقیقت Attention به LSTM کمک کرده است تا روی بازه‌های بحرانی (مانند شروع یا پایان سیکل موتور) تمرکز بیشتری داشته باشد.

ارزیابی عملکرد مدل‌ها

نمودارها و مقایسه بصری

تمامی کدها و نمودارهای خواسته شده در این بخش در بخش‌های قبل و مربوط به هر مدل آمده و تحلیل شده‌اند.

معیارهای ارزیابی مدل

تمامی معیارهای خواسته شده در بخش‌های قبل محاسبه و تحلیل شدند که در جدول زیر آمده‌اند:

مدل	RMSE	MAE	R^2	Time
CNN	۲۱.۹۴۹۶	۱۷.۱۱۳۸	۰.۷۴۶۹	۶۶.۶۱ ثانیه
LSTM	۱۶.۵۵۸۸	۱۲.۱۴۹۶	۰.۸۵۶۰	۲.۸۸ دقیقه
CNN+LSTM	۱۷.۷۸۱۰	۱۲.۶۸۳۳	۰.۸۳۳۹	۱.۹۸ دقیقه
LSTM+Attention	۱۶.۷۶۶۲	۱۲.۴۹۶۵	۰.۸۵۲۳	۲.۰۶ دقیقه

تحلیل عملکرد

قبل از پاسخ به سوالات، بطور خلاصه مدل‌ها مقایسه می‌شوند.

- مدل CNN: سرعت بالا، دقت پایین، overfitting، نامناسب
- مدل LSTM: بالاترین دقت کلی(R^2)، زمان زیاد آموزش، دقیق‌ترین
- مدل CNN+LSTM: تعادل بین دقت و سرعت، دقت کمی کمتر از LSTM
- مدل LSTM+Attention: دقت بالا، پیچیدگی کمی بیشتر از LSTM

بطور کلی اگر بخواهیم یک مدل واحد را به‌عنوان بهترین انتخاب برای تخمین RUL معرفی کنیم، LSTM + Attentionبهترین گزینه است، زیرا دقت هم در R^2 و هم در log RMSE بسیار بالاست، آموزش سریع‌تر از LSTM است و تعمیم‌پذیری خوب همراه

با تمرکز روی نواحی مهم‌تر داده‌ها دارد. بطور کلی برای استفاده نهایی LSTM یا LSTM+Attention مناسب بوده و اگر سرعت اجرا مهم است و مدل سبک‌تر مد نظر باشد، CNN+LSTM مناسب است.

- مدل LSTM دقیق‌ترین پیش‌بینی را ارائه داده است. زیرا نسبت به بقیه مدل‌ها MSE ، MAE و RMSE کمتر و بالاترین دقت پیش‌بینی را دارد. زیرا LSTM توانایی در درک وابستگی‌های زمانی بلندمدت دارد و داده‌های C-MAPSS به‌شدت وابسته به زمان هستند (سری‌زمانی). همچنین این مدل می‌تواند اطلاعات وضعیت موتور را در طول زمان به‌خوبی دنبال کند و الگوهای تغییر را یاد گرفته و اطلاعات گذشته را بدون فراموشی ناگهانی حفظ کند؛ برخلاف CNN که فقط اطلاعات محلی کوتاه‌مدت را بررسی می‌کند. البته لازم به ذکر است مدل LSTM+Attention نیز عملکرد بسیار مشابه با LSTM داشته و عملکرد خوب بوده است. مدل‌های ترکیبی مانند CNN+LSTM یا LSTM+Attention اگرچه مفیدند، اما ممکن است اگر به‌خوبی تنظیم نشده باشند، باعث افزایش نویز یا overfitting شوند.
- Overfitting زمانی رخ می‌دهد که مدل عملکرد خوبی روی داده‌های آموزش دارد ولی نتایج ضعیفی روی داده‌های اعتبارسنجی (validation) نشان می‌دهد. در اینجا برخی مدل‌ها تا حد کمی دارایoverfitting هستند اما مدل CNN دارای بیشترین overfittingمی‌باشد. روش تشخیص این اتفاق تفاوت زیاد بین دقت آموزش و تست، نمودارهای Loss و دقت در طول epoch ها، دقت بالا ولی خطای بزرگ روی داده‌های دیده‌نشده و افت R^2 یا افزایش ناگهانی log RMSE در تست می‌باشد.
- مدل CNN با اینکه سریع‌ترین زمان آموزش را (حدود ۶۶ ثانیه) دارد، اما پایین‌ترین دقت را نیز ارائه داده است .این مدل به دلیل ساختار ساده و تعداد پارامترهای کمتر، زمان آموزش کمی دارد، اما توانایی کافی برای یادگیری روابط پیچیده در داده‌های زمانی را ندارد.در مقابل، مدل LSTM دقیق‌ترین پیش‌بینی را ارائه داده ، اما بیشترین زمان آموزش را نیز دارد (حدود ۲.۸۸ دقیقه). این مدل با بهره‌گیری از حافظه بلندمدت، قادر به درک بهتر الگوهای زمانی در داده‌هاست، اما ساختار پیچیده‌تری دارد که آموزش آن را زمان‌بر می‌کند .مدل ترکیبی CNN + LSTM از لحاظ زمان آموزش در سطح متوسطی قرار دارد (حدود ۲ دقیقه) اما دقت آن کمتر از LSTM بوده است. ترکیب این دو مدل در این مسئله خاص، نتوانسته به بهینه‌ترین عملکرد برسد، احتمالاً به دلیل اینکه ویژگی‌هایی که CNN استخراج کرده برای یادگیری LSTM به‌خوبی قابل استفاده نبوده‌اند.

.در نهایت، مدل LSTM + Attention توانسته است تعادل بسیار خوبی بین دقت و زمان ایجاد کند. این مدل با زمان آموزش حدود ۲ دقیقه و دقتی نزدیک به($R^2 = 0.8523$) LSTM ، نشان می‌دهد که استفاده از مکانیزم توجه (Attention)باعث بهبود تمرکز مدل روی ویژگی‌های مهم‌تر داده شده و عملکرد آن را بدون افزایش زمان آموزش، بهبود داده است.

به طور خلاصه، اگر محدودیت زمانی یا پردازشی وجود داشته باشد، مدل CNN انتخاب مناسبی است. اما اگر دقت بالا در اولویت باشد، مدل LSTM یا LSTM همراه با Attention بهترین گزینه خواهند بود. در بسیاری از کاربردها، مدل LSTM + Attentionبا توجه به تعادل خوب بین دقت و زمان، بهترین انتخاب محسوب می‌شود.

****لازم به ذکر است گزارش و کد در گیت هاب نیز ارائه شده است. لینک آن در زیر آمده است****

https://github.com/maedehesmz8010/HW5_Esmaeilzade_810602161