Assignment 5: Rainfall runoff modelling using ANN

Excerpt – LSTM model by Python code

PAN XINXIN STU_ID: A0283981N

Remarks:

1. Section.1~2 involve the data pre-process methods,
2. Section.3 and Section.7 include the training methods about LSTM model in Python.

Contents

1. Input features of discharge

The first thing is to define the number of input discharge locations to forecast Q_MD04, aiming to the better model fitting and less compute cost, and it is necessary to consider comprehensively the rainfall-runoff paths at the whole catchment (layout seen in Fig 1) that may affect the flow rates Q_MD04.

Fig 1. Discharge locations at the Kent Ridge Catchment

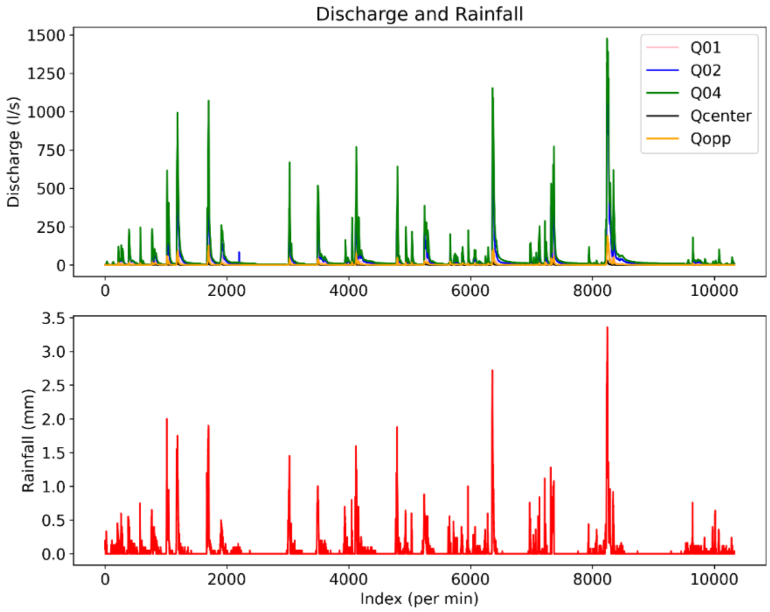

Fig 2. Rainfall-Runoff condition in one continue event

Thus, we compare their correlation and relative locations, and then use the model to simply verify the chose inputs.

1) Compare the higher correlation with Q_MD04

```
> cor_versue_Q04
Q_MD01      Q_MD02     Q_CNTRLIB    Q_OPPRLINK
 0.8702252  0.9789598  0.8496635    0.9534476
```

From the above correlations versus Q_MD04, the smallest one occurs in Q_ CNTRLIB with 0.85, although still shows relatively high correlation. Considering that Q_ CNTRLIB is far away from Q_MD04 and located on the boundary of this small catchment, we can ignore Q_CNTRLIB, also use the Q_opp located in the middle allows input discharge locations to cover a larger area for the rainfall.

2) Use MLP (RNNs) model to simply verify the choice of inputs

The MLP model set-up is built in R, and the default split percentage is 80%(training)+20%(validation).

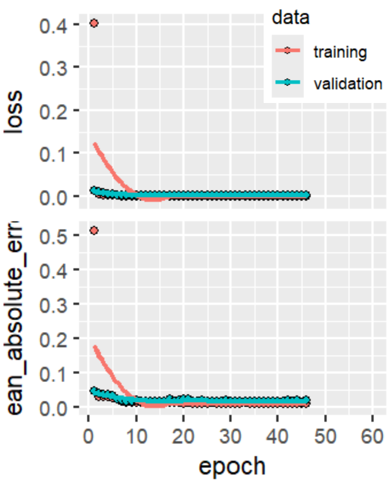

Fig 3. loss and error of T1

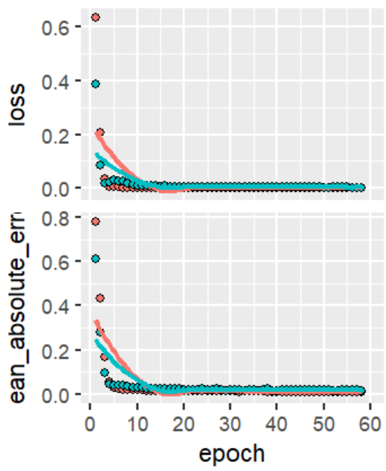

Fig 4. loss and error of T2

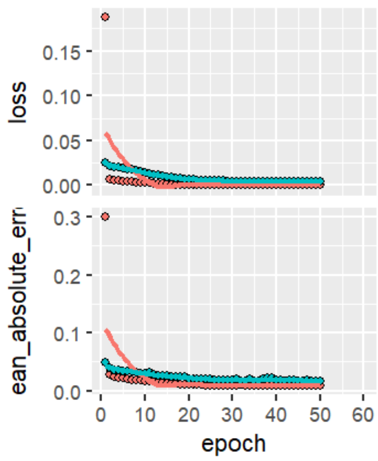

Fig 5. loss and error of T3

Table 1. 3 schemes of input features and statistical indicators

No.	Input features	MLP layer	accuracy	correlation	RMSE
T1	R+4Q	5+10	0.857	0.994	10.966
T2	R+3Q (without Q_CNTRLIB)	4+8	0.955	0.994	11.695
T3	R+2Q (Q_01+Q_02)	3+6	0.931	0.992	12.466

where the formula of accuracy: $accuracy = 1 - abs\_deviation = 1 - \left|\frac{actual - predicted}{actual}\right|$, and RMSE $= \sqrt{\frac{\sum(actual - predicted)^2}{n}}$.

From the training and validation loss and error (Fig 3~Fig 5), the difference is small, indicating that the MLP structure is suitable to avoid the overfitting or underfitting.

From the statistical indicators of the model, it shows that the accuracy and correlation in T2 is largest, while the RMSE is relatively large, indicating that the overall spread or dispersion of the predictions around the actual values is smaller, but some individual predictions might be further from the actual values (as indicated by the higher RMSE). Consider the discharge condition we focus on includes its overall trend and extreme discharge, T2 is a suitable choice.

Therefore, the input variables are **rainfall+Q01+Q02+Q_opp**.

2. Discuss choice of training and testing data sets

In deep learning model training, it's common to split the dataset into training and validation sets. The training set is used for updating model parameters, while the validation set is used to evaluate the modeling performance.

In this comparison, we use MLP model to compare and choose suitable splitting friction. The comparison is shown in Fig 6, and S2 and S3 avoid the test dataset ranging from the peak value of Q_MD04.

From the prediction (in Fig 7) and corresponding statistical indicators (in Table 2), the better prediction is based on S1. Although the RMSE is relatively high, showing an insufficient predictive performance in some cases, its prediction in the overall trend is better with a higher accuracy. Besides, the test sets in the model commence with peak values, but it does not hinder the more correct prediction, demonstrating the model's strong robust adaptability.

In S2, for test sets with initially lower discharge, the prediction exhibits a slight underestimation compared to the actual values, resulting in an overall downward shift in predictions.

In S3, to avoid the overfitting, the smaller units are used. And its prediction in overall trend is worse than S1 and that in extreme peak is worse than S2.

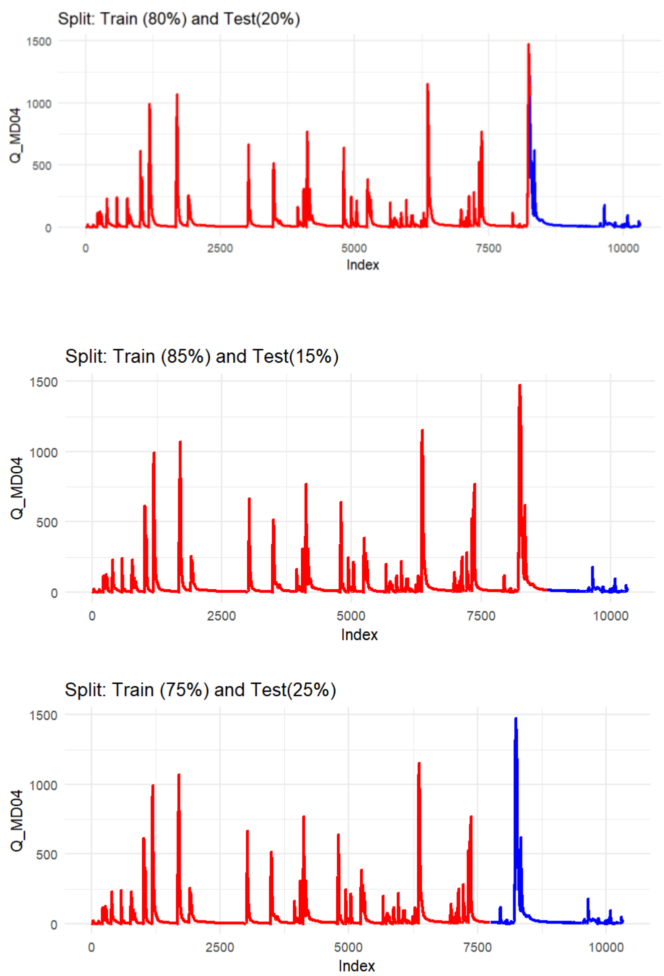

Fig 6. 3 split choices in dataset

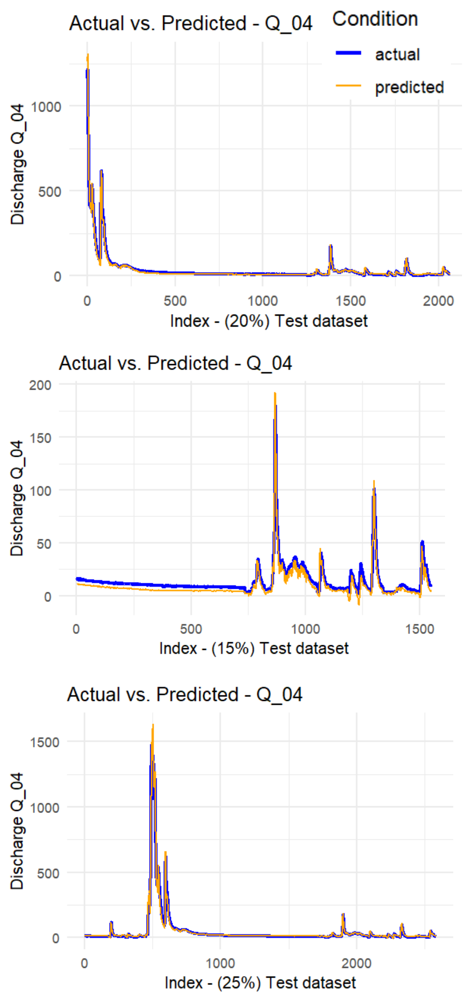

Fig 7. Comparison in test sets

Table 2. Comparison of 3 split choices and statistical indicators

No.	Split Choice	MLP layer	accuracy	correlation	RMSE
S1	80% + 20%	4+8	0.955	0.994	11.695
S2	85% + 15%	5+10	0.612	0.959	5.946
S3	75% + 25%	3+6	0.799	0.990	24.013

Thus, as we want to capture the overall trend, the split percentage as 80% training + 20% validation(test) is more suitable, because of the highest accuracy and its RMSE falling into the center ranges of 3 schemes.

Additionally, we can infer that for discharge datasets characterized by low overall trends and localized occurrences of peak, the corresponding RNNs model does not expand prediction errors even that the training sets start from lower value and the test sets commence with larger values, referring to suitable splitting choice. However, if commencing the test sets with lower values, that may cause a decrease in overall prediction accuracy.

Also, it seems the friction (80%+20%) is a better balance of the sufficient data of training and validation.

3. Train a LSTM model by .py

LSTM is commonly employed in modeling and predicting time-series data. And it needs a wrapped function to reshape the dimension of original datasets, to ensure the input data be 3-dimensional, where dimension 1 again is the batch dimension, dimension 2 again corresponds to the number of timesteps (10-, 20-, 60- min), and dimension 3 is the size of the wrapped layer (4 features). That wrapper's task is to apply the same calculation (i.e., the same weight matrix) to every state input it receives.

Then we need to take trials and tests to train a model with suitable units in the lstm layer.

```
# Build up a LSTM model
inputs = layers.Input(x_train.shape[1:], name='input')
lstm     = layers.LSTM(units=5, name='lstm')(inputs)
output = layers.Dense(units=1, name='dense', activation='linear')(lstm)

model   = models.Model(inputs, output)
model.summary()
es        = callbacks.EarlyStopping(monitor='val_loss', mode='min', verbose=1,
patience=40, min_delta=0.01, restore_best_weights=True)
model.compile(loss='mse', metrics=['mean_absolute_error'], optimizer='adam')
history = model.fit(x_train_scaled, y_train_scaled, epochs=80,
validation_split=0.2, callbacks=[es, PlotLossesKeras()])
```

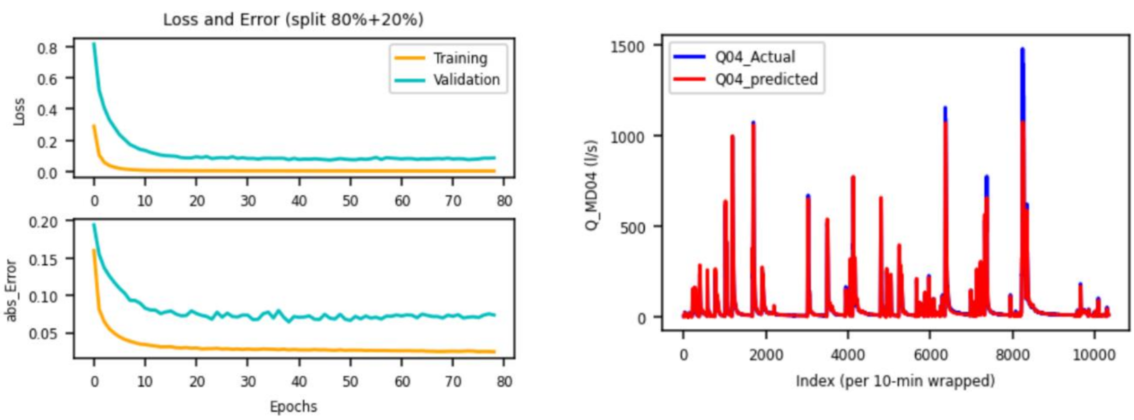

Fig 8. loss and error plot and prediction of LSTM.1

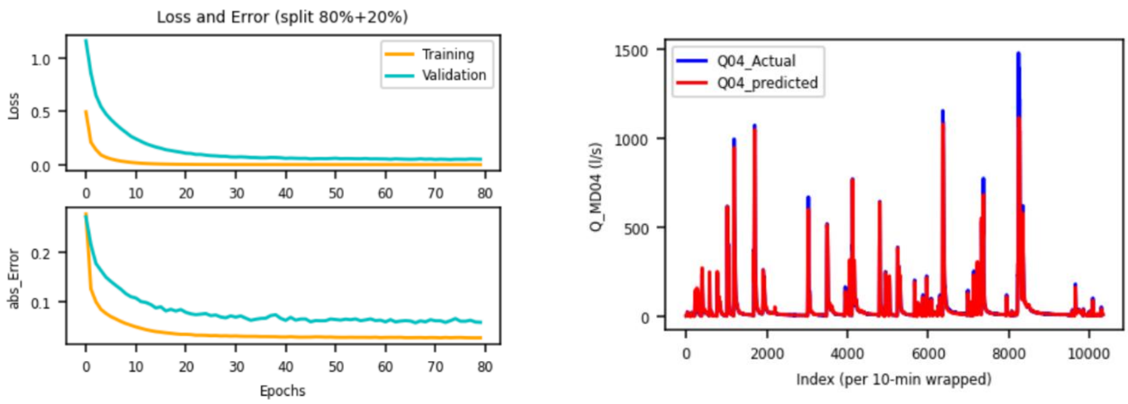

Fig 9. loss and error plot and prediction of LSTM.2

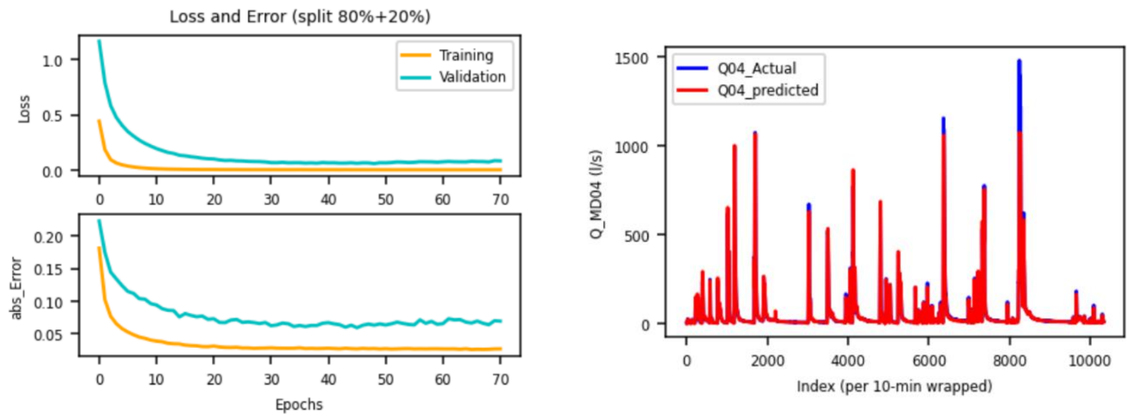

Fig 10. loss and error plot and prediction of LSTM.3

Table 3. Trials and tests indicators for LSTM (split-friction 80%+20%)

No.	Wrapped	Input/ Lstm	epochs	accuracy	RMSE
LSTM.1	10	(10, 4)/ 8	80	0.9960	9.9919
LSTM.2	10	(10, 4)/ 4	80	0.9562	6.9175
LSTM.3	**10**	**(10, 4)/ 5**	**70**	**0.9934**	**5.3363**

From the Table 3, it shows the units=5 is suitable for this model, as the smaller epochs needed and higher accuracy with lower RMSE.

4. Produce 10-, 20- and 60-minute forecast

We use the following two model to test the performance of different minutes forecast by MLP model and LSTM model. The corresponding data is extracted from the last several minutes.

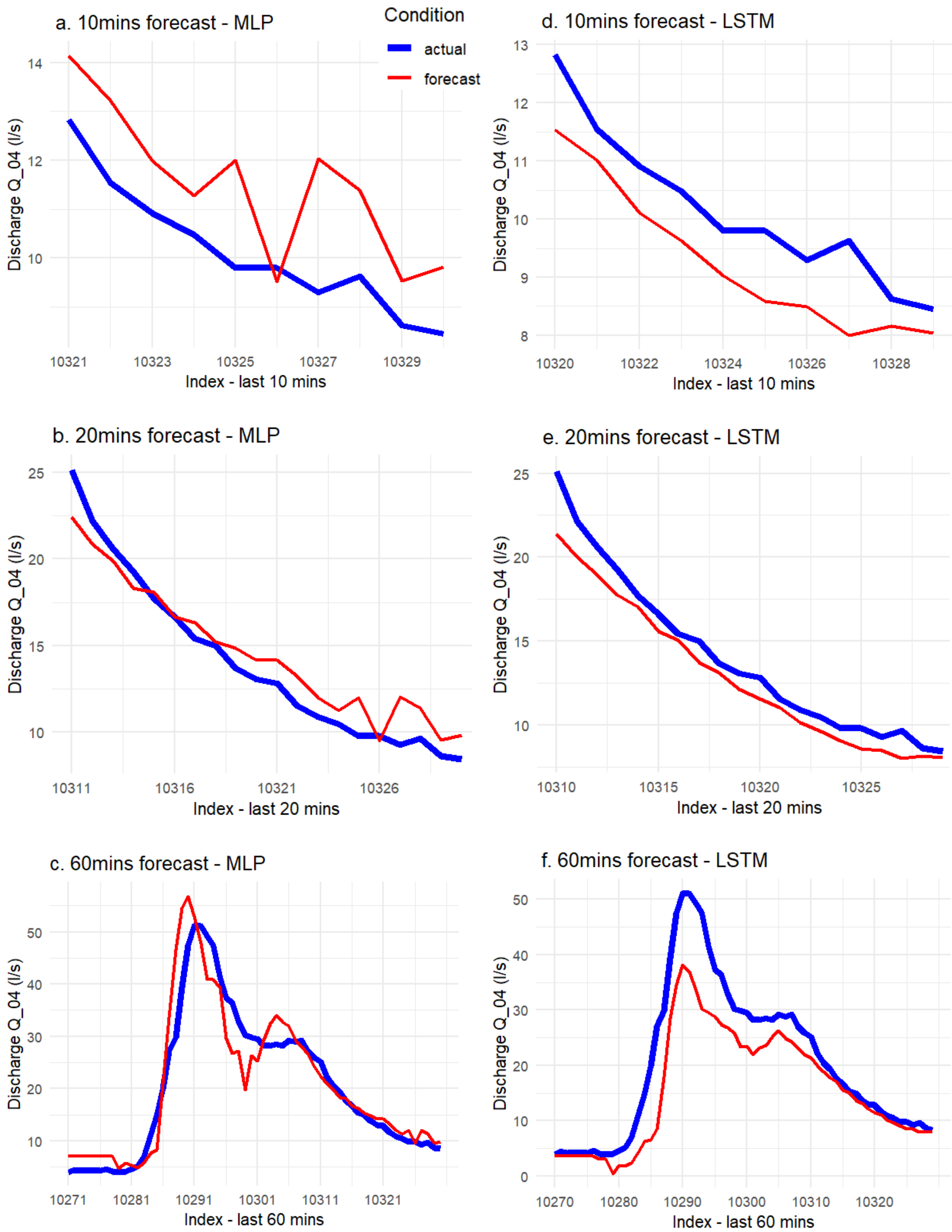

Fig 11. Comparison of Different Minutes Forecast by MLP model and LSTM model

From the Fig 11, the forecast by MLP obviously overestimates, showing a significant upwards deviation from the actual value in the overall series. And that by LSTM significantly underestimates, especially for the larger discharge scenario, the underestimation is more obvious.

Besides, the comparison of statistical indicators is in Table 4. For the last 10- and 20-min forecast, the LSTM model shows a better prediction, as higher accuracy and lower RMSE. But for the last 60-min forecast, its performance is significantly worse than MLP model.

Table 4. Indicators of Different Minutes Forecast by MLP model (DL.4)

No.	Forecast	MLP - DL.4			LSTM - LSTM.3		
		accuracy	correlation	RMSE	accuracy	correlation	RMSE
1	10min	0.8639	0.8455	1.5641	0.9135	0.9566	0.9559
2	20min	0.931	0.9842	1.3909	0.9204	0.9921	1.3692
3	60min	0.887	0.9409	4.7447	0.7751	0.9572	6.6944

In generally, the LSTM represents better performance for time series forecast compared to MLP. From the model work frame, MLPs do not have inherent memory capabilities. They process each input independently without considering any temporal dependencies between them. Therefore, MLPs can respond to the peak actual values as a physiological effect of changing inputs.
As a result, if we want to obtain a more aggressive estimation showing a higher prediction, MLP model can exhibit optimistic projections instead of the conservatism. However, if want to consider the rainfall-runoff time series effect, LSTM shows a more meaningful prediction.

5. Discuss forecast accuracy as function of lead time

In LSTM model, we want to discuss the impact of different wrappers with 10-, 20-, 60-mins. Firstly, we consider its impact on the overall trend in Table 5.

Table 5. Indicators in the Overall trend of LSTM (in the whole dataset)

Wrapped	No.	Trial 1		Trial 2		Trial 3		average	
		accuracy	RMSE	accuracy	RMSE	accuracy	RMSE	accuracy	RMSE
10	LSTM.4	0.9641	7.0274	0.9658	9.4024	0.9931	6.2844	0.9743	7.5714
20	LSTM.5	0.9770	8.7198	0.9696	8.8683	0.9823	6.8119	0.9763	8.1333
60	LSTM.6	0.9662	7.1721	0.9605	5.6516	0.9503	5.4183	0.9590	6.0807

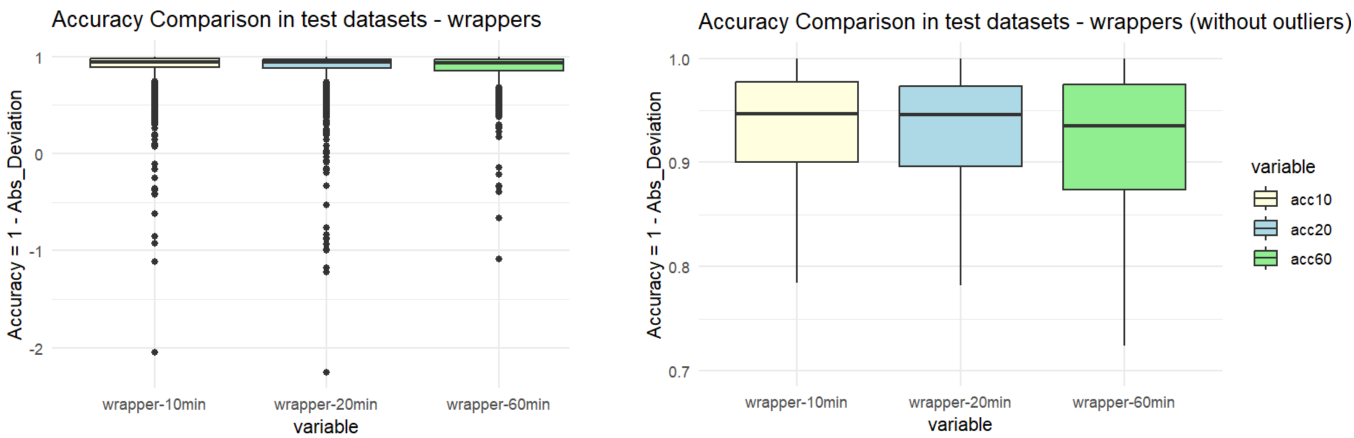

Fig 12. Boxplot for Accuracy Comparison in test datasets

After training, the model with 10-min and 20-min can generate a more accurate prediction but with some significant underestimation in some points (as higher RMSE in the whole datasets). If we only evaluate the model performance in test sets in Fig 12, it shows around 10% outliers in test sets, considering the significant underestimation in

peak discharge, and the model with 60min leading time shows a better adjustability in peak value.
The wrapper's task is to apply the same calculation (i.e., the same weight matrix) to every state input it receives, and it works as adding or removing features iterative. In the catchment evaluation, the leading time shows the rainfall-runoff concentration condition in this catchment.
To get a comprehensive prediction, we can use the shorter leading time to evaluate the lower discharge scenario, and use a longer leading time to obtain the peak value with a higher accuracy.

6. Choose a neural network architecture for the catchment evaluation

In the hydrological theory, we know that the rainfall-runoff events have significant time series properties. And generally, the LSTM represents better performance for time series forecast compared to MLP.
From the above analysis, we can find that LSTM can forecast with a stable and higher accuracy. Besides, if generate a shorter time prediction, LSTM can train a model in a high-efficiency and provide a better performance both in the overall trend and capturing peak values.

7. Appendix: Script of LSTM in .py

```
# define a function to reshape the dataset to meet LSTM model
def get_wrapped_data(dataset, wrap_length=10):
    data_x, data_y = [], []

    for i in range(len(dataset) - wrap_length):
        # dataset (rainfall, Q01, Q02, Q04, Qopp)
        data_x.append(dataset.iloc[i:i+wrap_length, [0, 1, 2, 4]].to_numpy(dtype='float64'))
        data_y.append(dataset.iloc[i+wrap_length, [3]].astype('float64'))

    return np.array(data_x), np.array(data_y)

# Wrap the data
data_x, data_y = get_wrapped_data(dataset, wrap_length=10)
data_x.shape
split_index = int(len(data_x) * 0.8) # splitting = 80% test + 20% train

# Split the data into training and testing sets
x_train = data_x[:split_index]
y_train = data_y[:split_index]
x_test = data_x[split_index:]
y_test = data_y[split_index:]

# Initialize scale parameters
scale_params = {
    "train_x_mean": 0,
    "train_x_std": 1,
    "train_y_mean": 0,
    "train_y_std": 1
}

# Calculate mean and standard deviation for scaling
scale_params["train_x_mean"] = np.mean(x_train, axis=(0, 1))
scale_params["train_x_std"] = np.std(x_train, axis=(0, 1))
scale_params["train_y_mean"] = np.mean(y_train, axis=0)
scale_params["train_y_std"] = np.std(y_train, axis=0)

# Normalize training and testing data
x_train_scaled = (x_train - scale_params["train_x_mean"][None, None, :]) / scale_params["train_x_std"][None, None, :]
y_train_scaled = (y_train - scale_params["train_y_mean"][None, :]) / scale_params["train_y_std"][None, :]
x_test_scaled = (x_test - scale_params["train_x_mean"][None, None, :]) /
```

```
scale_params["train_x_std"][None, None, :]
y_test_scaled = (y_test - scale_params["train_y_mean"][None, :]) /
scale_params["train_y_std"][None, :]

# Build up a LSTM model
inputs = layers.Input(x_train.shape[1:], name='input')
lstm   = layers.LSTM(units=5, name='lstm')(inputs)
output = layers.Dense(units=1, name='dense', activation='linear')(lstm)

model  = models.Model(inputs, output)
model.summary()
es     = callbacks.EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=40,
                                  min_delta=0.01, restore_best_weights=True)
model.compile(loss='mse', metrics=['mean_absolute_error'], optimizer='adam')
history = model.fit(x_train_scaled, y_train_scaled, epochs=80, validation_split=0.2,
                    callbacks=[es, PlotLossesKeras()])

# Prediction
pred_train = model.predict(x_train_scaled)
pred_test  = model.predict(x_test_scaled)
Training_result = pred_train * scale_params["train_y_std"] + scale_params["train_y_mean"]
testing_result = pred_test* scale_params["train_y_std"] + scale_params["train_y_mean"]
dataset['flow_pred'] = None
dataset.iloc[10:len(Training_result)+10, dataset.columns.get_loc('flow_pred')] = Training_result
dataset.iloc[len(Training_result)+10:, dataset.columns.get_loc('flow_pred')] = testing_result

# assessment and accuracy
actual_data = dataset['Q04'].values[len(Training_result)+10:]
denormalized_predictions = dataset['flow_pred'].values[len(Training_result)+10:]
# accuracy
deviation = (actual_data - denormalized_predictions) / actual_data
accuracy_rate = 1 - np.abs(np.mean(deviation))

# RMSE
RMSE = np.sqrt(np.mean((actual_data - denormalized_predictions) ** 2))
```