1 The reasoning behind the design of network architectures

1.1 CNN

Conv1D

Conv1D is primarily employed for processing sequential data, such as time series or text sequences. In such cases, the convolutional kernel slides along the time axis (or sequence axis) to capture local patterns.

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 28, 64)	832
dropout (Dropout)	(None, 28, 64)	0
conv1d_1 (Conv1D)	(None, 26, 64)	12352
dropout_1 (Dropout)	(None, 26, 64)	0
max_pooling1d (MaxPooling1D)	(None, 13, 64)	0
flatten (Flatten)	(None, 832)	0
dense (Dense)	(None, 1)	833

Total params: 14017 (54.75 KB)
Trainable params: 14017 (54.75 KB)
Non-trainable params: 0 (0.00 Byte)

1D

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 4, 64)	256
dropout_2 (Dropout)	(None, 28, 4, 64)	0
max_pooling2d (MaxPooling2D)	(None, 14, 4, 64)	0
conv2d_1 (Conv2D)	(None, 12, 4, 64)	12352
dropout_3 (Dropout)	(None, 12, 4, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 6, 4, 64)	0
flatten_1 (Flatten)	(None, 1536)	0
dense_1 (Dense)	(None, 1)	1537

Total params: 14145 (55.25 KB)
Trainable params: 14145 (55.25 KB)
Non-trainable params: 0 (0.00 Byte)

2D

For the first convolutional layer, it has 832 parameters that can be calculated by filters * (kernel_size * input_channels + 1), where 1 represents the bias. Thus num_para = 64 * (3 * 4 + 1) = 832. The same as the second convolutional layer.

Conv2D

By the same equation mentioned above, we can calculate the number of parameters: 64 * (3 * 1 * 1 + 1) = 256. The same as the second convolutional layer.

Ablation study on number of convolutional layers

Intuitively, we assume that increasing the number of convolutional layers improves the model's performance, and in practice, it often does, but this improvement is limited to the training data. When applied to other data, performance tends to decline, which is known as the phenomenon of overfitting.

Test data	Conv2D	
	1 layer	2 layers
Precision	[0.9988, 0.8857]	[0.9994, 0.8888]
Recall	[0.9976, 0.9394]	[0.9976, 0.9697]
F-score	[0.9982, 0.9118]	[0.9985, 0.9275]

Test data	Conv1D	
	2 layer	3 layers
Precision	[1.0, 0.9705]	[1.0, 0.9705]
Recall	[0.9994, 1.0]	[0.9994, 1.0]
F-score	[0.9997, 0.9851]	[0.9997, 0.9851]

From the table above, we can find that with the number of layers increasing, Conv1D meets the problem of overfitting, which misclassified 10 normal data into anomalies. However, Con2D is more stable, when it comes to 4, 5 and more layers it still has a good performance.

1.2 LSTM

Here's the reasoning behind the design choices:

1. Multiple LSTM Layers: The model uses three LSTM layers stacked on top of each other, which enables the model to learn complex patterns. Each layer can potentially capture different levels of abstraction in the data. The return_sequences=True parameter in the first two LSTM layers ensures that the full sequence of hidden states is passed to the next layer, which is necessary for stacking.
2. Number of Units: Each LSTM layer has 100 units. A higher number of units can allow

the model to learn more complex patterns.

3. Dropout: Dropout layers are used after each LSTM layer with a rate of 0.2. Dropout is a regularization technique that helps prevent overfitting by randomly setting a fraction of the input units to 0 at each update during training time.
4. Final Dense Layer: A Dense layer with a single unit and a sigmoid activation function is used as the output layer. This is used for binary classification, where the output is the probability of the input sequence belonging to the positive class (anomaly detected). The sigmoid function squashes the output to a range between 0 and 1, suitable for representing probabilities.
5. Loss Function: The binary_crossentropy loss function is appropriate for binary classification problems. It measures the performance of a classification model whose output is a probability value between 0 and 1.

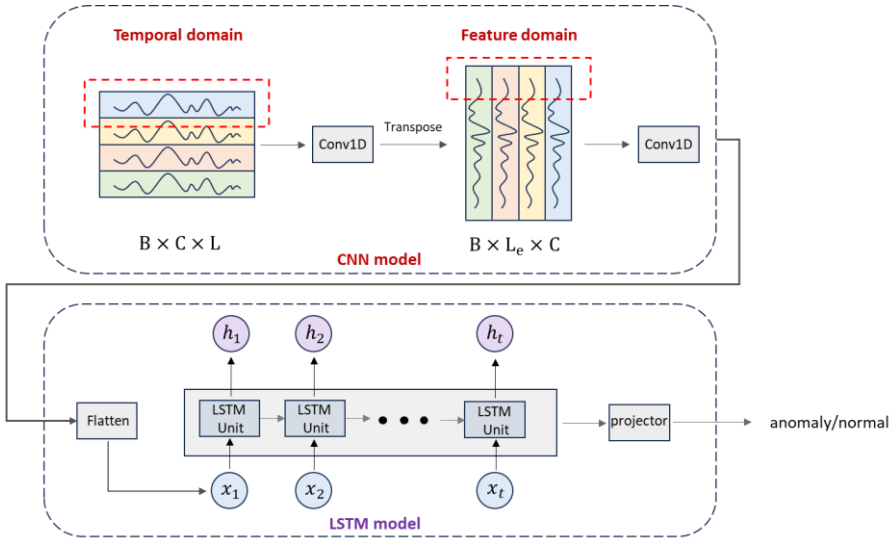
The architecture is designed to be flexible and powerful enough to capture complex relationships in the data,while also including measures to prevent overfitting.

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 40, 100)	42000
dropout (Dropout)	(None, 40, 100)	0
lstm_1 (LSTM)	(None, 40, 100)	80400
dropout_1 (Dropout)	(None, 40, 100)	0
lstm_2 (LSTM)	(None, 100)	80400
dropout_2 (Dropout)	(None, 100)	0
dense (Dense)	(None, 1)	101

Total params: 202,901
Trainable params: 202,901
Non-trainable params: 0

1.3 Hybrid CNN-LSTM

1.3.1 general structure of hybrid CNN-LSTM



1.3.2 design of structure

CNN model

We leverage two-stage CNN models to learn knowledge from both **temporal domain and feature domain**. Specifically, the input lookback window is firstly reshaped to B x 1 x C x L , where C denotes the variable number and Ldenotes the length of timesteps. As Conv1d is better

suited to capture the dependency of a sequence, we choose Conv1D layer. Then the Conv1D layer with 32 filters is used to capture the dependencies from the temporal domain.

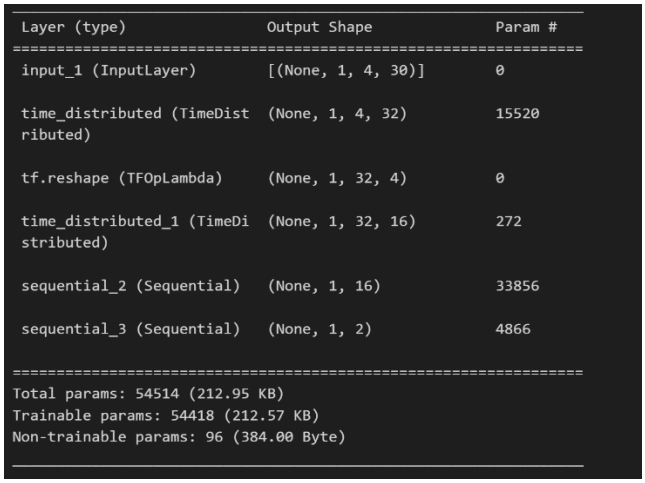
To equip our model with the ability to obtain cross-variable relationships, we employ another Conv1D layer to learn knowledge from the feature domain. We first transpose the embedding vectors from the 1st Conv1D layer to B x 1 x Le x C, where Le stands for the output dimension of the 1st Conv1D layer, i.e, 32. After that, the second Conv1D layer with 16 filters is applied in the feature domain, making our model capture the relationships between different variables. In addition, batch normalization and dropout layers are used after each CNN layers to prevent overfitting and improve model's generalization. Finally, we choose the ReLU activation function to make our model learn non-linear dynamics.

LSTM

After two-stage CNN layers, our model yield B × 1 × Le × Ce embedding vectors, where Ce represents the output dimension of the 2nd Conv1D layer, i.e, 16. To fit the requirement of input shape of LSTM, we first flatten() the embedding vectors to construct a sequence, then the sequence passes through the LSTM model with 16 units and yields 1 x 16 output. Following the setting of CNN models, we also add a dropout layer after LSTM layer to improve model's performance.

Finally, we design a 2-layer MLP projector with Softmax() activation function to predict the label (anomaly/ normal).

The number of trainable parameters calculation:



Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 1, 4, 30)]	0
time_distributed (TimeDistributed)	(None, 1, 4, 32)	15520
tf.reshape (TFOpLambda)	(None, 1, 32, 4)	0
time_distributed_1 (TimeDistributed)	(None, 1, 32, 16)	272
sequential_2 (Sequential)	(None, 1, 16)	33856
sequential_3 (Sequential)	(None, 1, 2)	4866

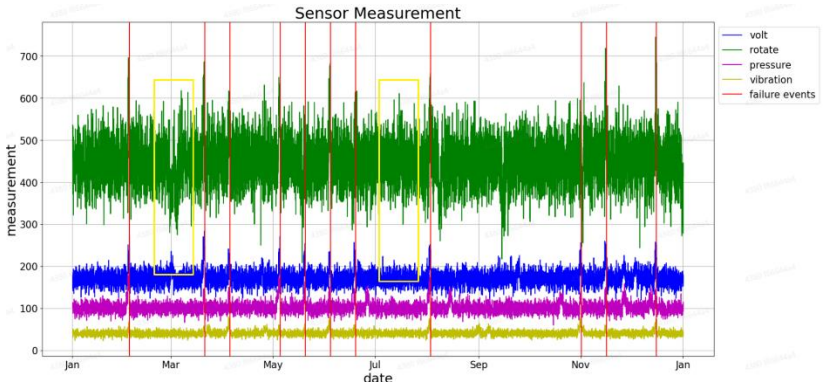
Total params: 54514 (212.95 KB)
Trainable params: 54418 (212.57 KB)
Non-trainable params: 96 (384.00 Byte)

- The number of weights for the Conv1D layer is given by the formula: filters * kernel_size * input_channels. Therefore, for the first Conv1D layer: 32 × 16 × 30 = 15360. Each filter has a bias term: 32, Batch normalization:4 × 32 = 128, adding them up: 15520
- for the second Conv1D layer, the formula is the same: 16 × 3 × 4 = 192. Each filter has a bias term：16，Batch normalization: 4 × 16 = 64, adding them up: 272
- The number of parameters for the LSTM layer is given by the formula: 4 * units * (input_size + units + 1). Therefore, for the LSTM layer: 4 × 16 × (512 + 16 + 1) = 33856
- The number of parameters for the Dense layer is given by the formula: input_size * units + units. Therefore, for the first Dense layer: 16 × 256 + 256 = 4352 for the second Dense layer: 256 * 2 + 2 = 514, adding them up: 4866

2 Answers to the questions in Section 7.1

2.1 (the answer to 7.1.1)

The comparison between the original sensor data plot and the sliding window averaged data plot would typically reveal the following observations:



- Original Sensor Measurements: In the original plot, it is difficult to distinguish between noise and actual anomalies or failures. Short-lived spikes might be mistaken for sensor errors rather than actual issues. For example, we are hard to tell whether the spikes in yellow rectangles represent potential failures.
- Sliding Window Average Data: The sliding window makes it easier to observe the underlying trends in sensor measurements. Sharp spikes or dips that persist over several time points are more likely to be real phenomena rather than noise, and these will still be visible after averaging.

2.2 (the answer to 7.1.2)

Because in the task of anomaly detection, anomalies are a minority (33 out of 1721). Thus, even if we fail to detect a single anomaly, we still have accuracy of ~0.98. Therefore, in this context, high accuracy does not indicate a good model.

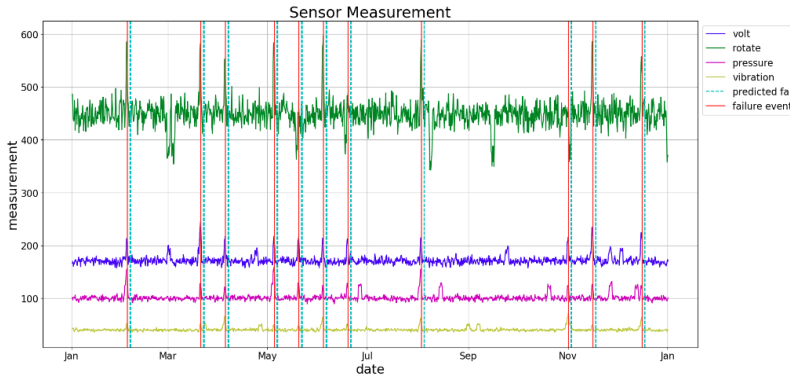
The following metrics are more important than accuracy:

1. **Recall**: This is the ability of the model to correctly identify all actual anomalies. In a factory setting, missing an anomaly could be costly or dangerous, so a high recall is important.
2. **Confusion Matrix**: The shape of confusion matrix matters and it provides a more detailed understanding of where the model is making errors. A good model reflects high values on the diagonal (TP and TN) and low values in the off-diagonal elements (FP and FN).

2.3 (the answer to 7.1.3)

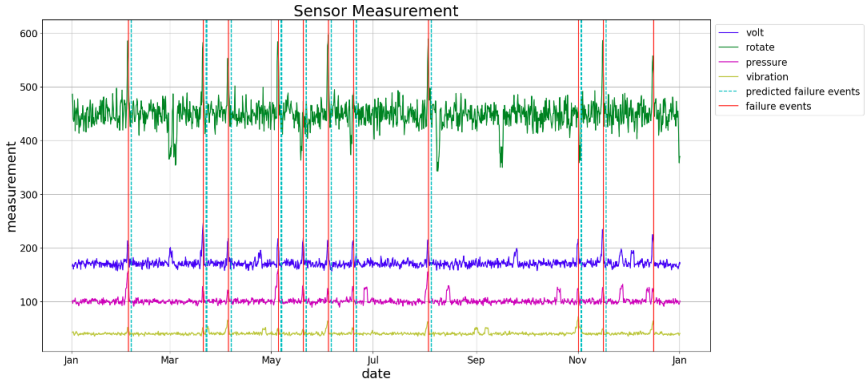
seq_len = 30：

Conv1D



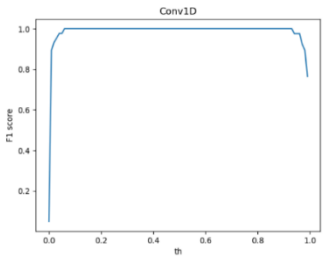

```
Accuracy:  0.999418942475305
Confusion matrix:
 [[1687    1]
 [   0   33]]
Precision: 0.9705882352941176
Recall:    1.0
F-score:   0.9850746268656716
```

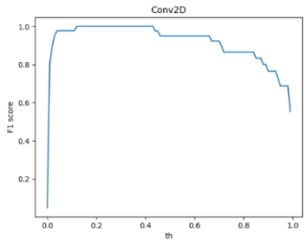
Conv2D

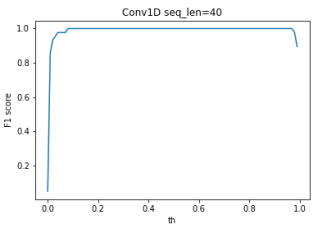


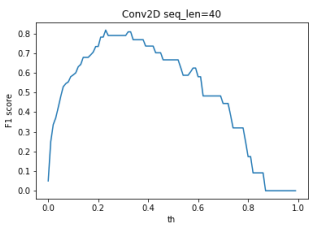

```
Accuracy:  1.0
Confusion matrix:
 [[1688    0]
 [   0   33]]
Precision: 1.0
Recall:    1.0
F-score:   1.0
```

We can find that in seq_len = 30, the model runs well in predefined th = 0.3. However, when observing a longer sequence, the performance gets worse. So I decide to find the relationship between them. From the figures below, we can see th = 0.3 is suitable in this case.



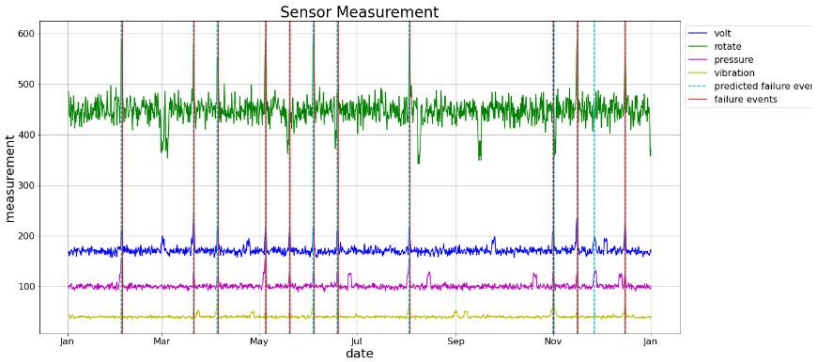






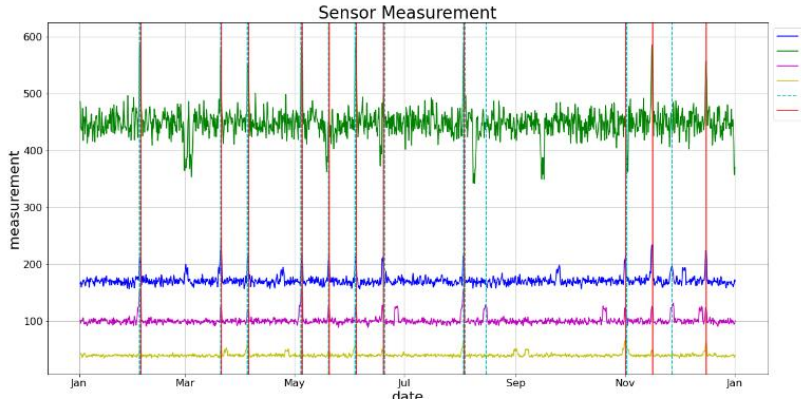
In this case, let th = 0.23, which may achieve a better performance.

Conv1D




```
Accuracy:  0.9976621858562245
Confusion matrix:
 [[1674    4]
 [   0   33]]
Precision: 0.8918918918918919
Recall:    1.0
F-score:   0.9428571428571428
```

Conv2D




```
Accuracy:   0.98305084745762
Confusion matrix:
 [[1662   16]
 [  13   20]]
Precision:  0.55555555555555
Recall:     0.60606060606060
F-score:    0.57971014492753
```

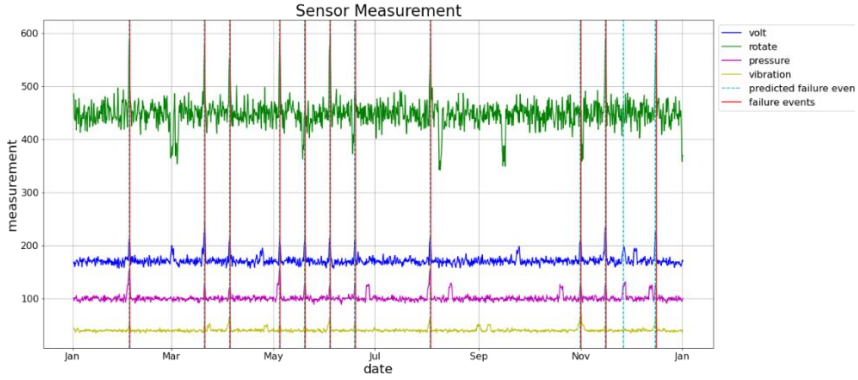
However, Conv2D still performs badly and from the relationship figure, we can find changing th won't get any benefits. From this experiment, we have two conclusions:

1. In practice, we should adjust the threshold based on the data in reality. As we can see from the line graph, different models and different datasets all lead to different thresholds
2. For sequential data, Conv1D enjoys more merits. In other words, using the Conv1D model is sufficient for the current situation. In addition, we can see from the code model that the kernel size of Conv2D is set to (3, 1), which is actually equivalent to Conv1D, and using Conv2D is redundant.

2.4 (the answer to 7.1.4)

seq_len = 30：

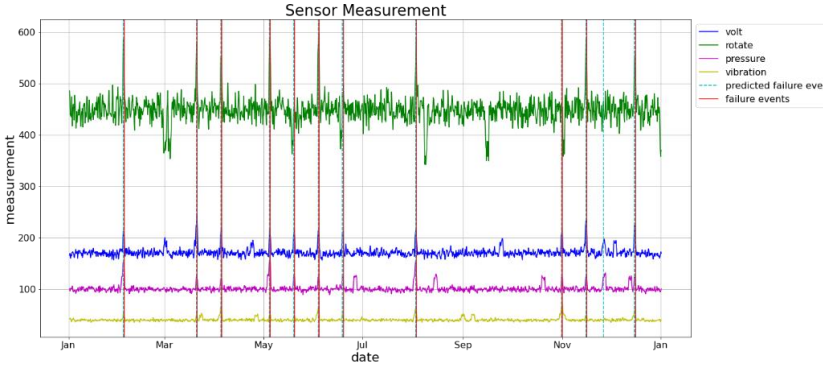
Predicted result plotting and metrics:




```
Accuracy on test data: 0.999418942475305
Confusion Matrix on test data:
[[1687    1]
 [   0   33]]
Precision on test data: 0.9705882352941176
Recall on test data: 1.0
F1 Score on test data: 0.9850746268656716
```

seq_len = 40：

Predicted result plotting and metrics:



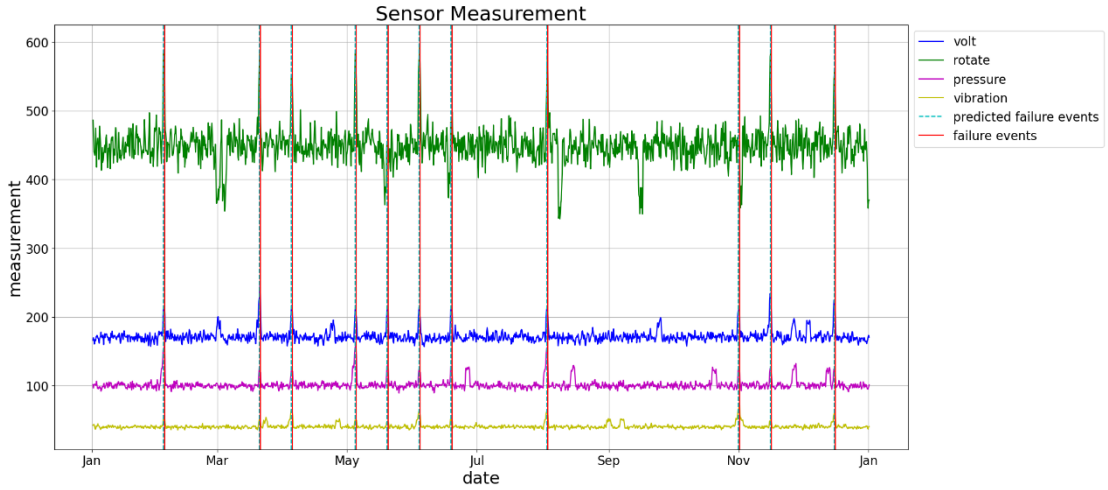

```
Accuracy on test data: 0.9994155464640561
Confusion Matrix on test data:
[[1677    1]
 [   0   33]]
Precision on test data: 0.9705882352941176
Recall on test data: 1.0
F1 Score on test data: 0.9850746268656716
```

From these metrics, we can infer that increasing the sequence length from 30 to 40 time points

does not significantly affect the model's performance on this particular test dataset. The model's ability to predict anomalies after observing a longer past sequence pattern (40 time points) is essentially equivalent to its performance with a shorter past sequence pattern (30 time points). This could mean that the additional 10 time points in the sequence do not provide significant additional information for the model to improve its predictions,or that the model has reached a performance plateau with the given dataset. It's also possible that the nature of the anomalies is such that a 30-point time window is sufficient to capture the necessary pattern for detection.
If there is no significant computational constraint, using a seq_len of 40 might be preferable as it could potentially capture longer-term dependencies that might become relevant as more data is collected. However, if computational resources are a concern, sticking with a seq_len of 30 could be more efficient without sacrificing performance.

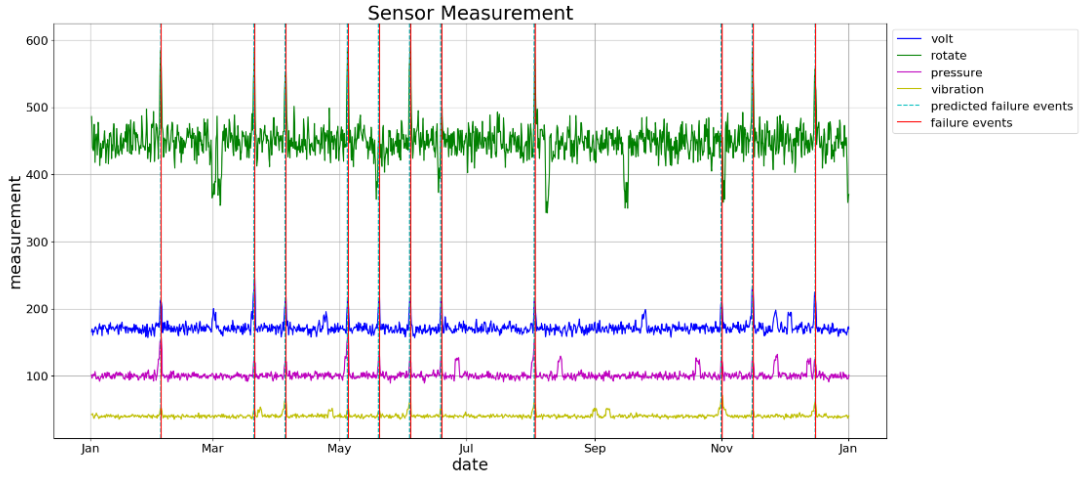
2.5 (the answer to 7.1.5)

seq_len = 30：




```
confusion matrix:
 [[1688    0]
 [   0   33]]
precision:1.0, recall:1.0, f_score:1.0
```

seq_len = 40：




```
confusion matrix:
 [[1678    0]
 [   1   32]]
precision:1.0, recall:0.9696969696969697, f_score:0.9846153846153847
```

We selected sequence lengths of 30 and 40 for experiments, and found that the performance was slightly lower when the length was 40, which may be due to the following reasons:

- A longer sequence imposes a strain on Conv1D to learn the explicit temporal pattern characteristics.
- A longer sequence may introduce more useless information, like noise, which may influence model's generalization.
- Conv1D layers with limited and fixed receptive fields are not applicable to handle longer sequence and learn global dependencies.

2.6 (the answer to 7.1.6)

model	accuracy	precision	recall	F-score
CNN-1D	0.999418942475305	1.0	0.9696969696969697	0.9846153846153847
CNN-2D	0.999418942475305	1.0	0.9696969696969697	0.9846153846153847
LSTM	0.9988378849506101	0.9696969696969697	0.9696969696969697	0.9696969696969697
Hybrid	1.0	1.0	1.0	1.0

From experimental results, we have the following findings:

1. In this anomaly detection context, the recall is considered to be the most important metric. Hybrid CNN-LSTM achieves the best result, while CNN and LSTM are slightly worse than the hybrid model.
2. In hybrid CNN-LSTM, better results are achieved while using fewer parameters than LSTM. CNN-LSTM and LSTM use 212.95 KB and 792.85 KB parameters respectively.
3. Training time comparison: CNN: 30.5 s, LSTM: 39.2 s, Hybrid CNN-LSTM: 1min 7s (Longest)
4. Comparing CNN and LSTM, LSTM performs slightly worse in this case (after careful parameter tuning may achieve the same results).

2.7 (the answer to 7.1.7)

CNN

Conv1D

The tables below show changing the sliding step size while fixing the sliding window width and vice versa.

sliding_window_width = 10	2	5	10
Accuracy	0.9986	0.9994	0.9835
Precision	0.9429	1.0	0.4286
Recall	1.0	0.9697	0.8182
F1 score	0.9706	0.9846	0.5625
Confusion matrix	[4241 6 0 99]	[1688 0 1 32]	[823 12 2 9]

sliding_step_size = 5	5	10	20
Accuracy	0.9954	0.9994	0.9983
Precision	0.86	1.0	0.9167
Recall	0.9773	0.9697	1.0
F1 score	0.9149	0.9846	0.9565
Confusion matrix	[1671 7 1 43]	[1688 0 1 32]	[1683 3 0 33]

Conv2D

sliding_window_width = 10	2	5	10
Accuracy	0.9965	0.9994	0.9917
Precision	0.8889	1.0	0.7
Recall	0.9697	0.9697	0.6364
F1 score	0.9275	0.9846	0.6667
Confusion matrix	[4235 12 3 96]	[1688 0 1 32]	[832 3 4 7]

sliding_step_size = 5	5	10	20
Accuracy	0.9954	0.9994	0.9988
Precision	0.9091	1.0	1.0
Recall	0.9091	0.9697	0.9394
F1 score	0.9091	0.9846	0.9688
Confusion matrix	[1674 4 4 40]	[1688 0 1 32]	[1686 0 2 31]

From the above table, we can conclude that the appropriate sliding window width and sliding step size are crucial, and the two need to achieve a trade-off.

However, in terms of controlling variables, a smaller sliding step size and a larger sliding window width are more likely to yield favorable results. This is because a smaller sliding step size can acquire more observational data, increasing the quantity of training data. On the other hand, a larger sliding window width can detect more anomalies but may lead to misjudgments of normal data.

It's worth noting that when the sliding step size and sliding window width are equal, the model performs the worst. When the window width and step size are equal, adjacent windows have complete overlap. This results in significant redundancy between neighboring windows, as they share most of the same data points. Overlapping data may cause the model to overly focus on similar data during training and testing, lacking a comprehensive learning of the overall data distribution.

LSTM

When we make such change: sliding_window_width = 8, sliding_step_size = 4, we have:

```
Accuracy on test data: 0.9930523390458545
Confusion Matrix on test data:
[[2098    6]
 [   9   46]]
Precision on test data: 0.8846153846153846
Recall on test data: 0.8363636363636363
F1 Score on test data: 0.8598130841121494
```

When we make such change: sliding_window_width = 15, sliding_step_size = 7, we have:

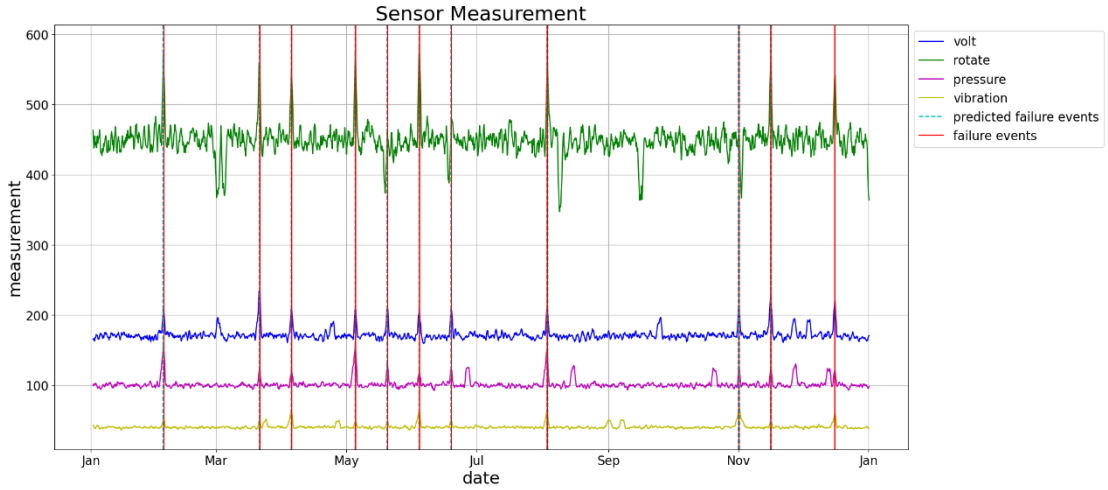
```
Accuracy on test data: 0.9860655737704918
Confusion Matrix on test data:
[[1192    1]
 [  16   11]]
Precision on test data: 0.9166666666666666
Recall on test data: 0.4074074074074074
F1 Score on test data: 0.5641025641025641
```

The adjustment of the sliding window width and step size has a direct impact on the ability of the LSTM model to recognize anomalies. Smaller window widths and step sizes and larger window widths and step sizes can lead to decreased recall. This is because smaller window widths and step sizes may cause the model to retain too much noise, while larger window widths and step sizes may cause the model to lose key features and smooth out short-term anomalous signals.

Hybrid CNN-LSTM

1. When we change the sliding window width as:

sliding_window_width = 20 sliding_step_size = 5



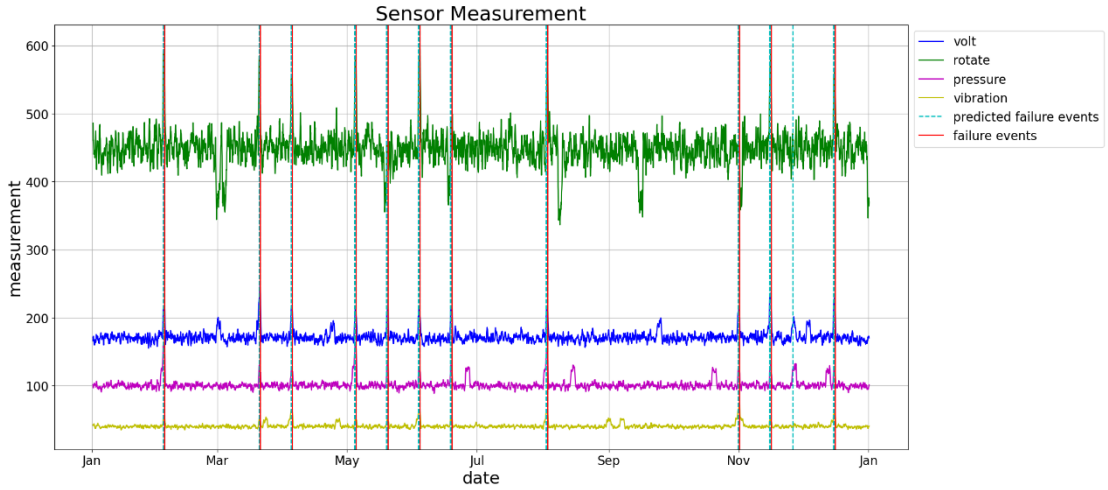

```
confusion matrix:
 [[1686    0]
 [   2   31]]
precision:1.0, recall:0.9393939393939394, f_score:0.96875
```

In this context, we could observe from the confusion matrix that the model predicts 2 anomaly as normal instances.

With a larger window size, the model may distract attention and overlook abnormal mutation, therefore failing to detect 2 abnormal events. Besides, longer sequences introduce more noise and useless information, which also impacts model's performance adversely.

2. When we change the sliding window width as:

sliding_window_width = 10 sliding_step_size = 2




```
confusion matrix:
 [[4244    3]
 [   2   97]]
precision:0.97, recall:0.9797979797979798, f_score:0.9748743718592964
```

From the figures shown above, we observe that the model predicts 3 normal instances as anomaly and 2 abnormal instances as normal events. One more blue line is shown in predicted failure events.

Smaller step size may potentially capture local patterns and fluctuations that may not be indicative of true anomalies. This high sensitivity to small changes might lead to more false alarms. Meanwhile, the model may overfit random fluctuations or noise, resulting in more false positives.

2.8 (the answer to 7.1.8)

From the figures in TensorBoard, we can draw the following conclusions:

1. Stability of weight distribution:

- ✓ The conv1d/kernel and conv1d 1/kernel of CNN-1D model show that the weight distribution gradually becomes more concentrated with the training process, which indicates that the model tends to be stable when learning features.
- ✓ The relatively wide weight distribution of the LSTM model indicates that it may be more active in capturing the diversity of time series data. Hybrid CNN-LSTM model combines the characteristics of both, and its weight distribution reflects the characteristics of the CNN and LSTM layers.

2. Range of weight values:

- ✓ The weight range of CNN-1D is small, which may be more suitable to capture local features.
- ✓ The LSTM model has a wide range of weight values that may help capture long-term dependencies.
- ✓ The weight range of the hybrid model may perform well in capturing both local features and long-term dependencies.

3 Explanations and hypotheses on the relative performance of the different models

1. Good Performance achieved in hybrid model: hybrid model combines the strengths of CNN and LSTM, making it particularly effective for tasks that involve both spatial and temporal dependencies.
2. Longer training time: hybrid model increased complexity compared to individual models, which may require more computational resources and time.
3. Comparing CNN and LSTM: there is no doubt that CNN has a concise structure but achieves a similar result. We suspect that LSTM performs slightly worse in this case because there is no obvious correlation between the data changes and time series. And CNN will be able to capture the changes in the data characteristics.
4. Parameter space: our LSTM model is more complex than traditional RNNs, which may result in longer training time. 3 LSTM layers also introduce larger parameter space compared the other models.

CNN has strength in capturing spatial patterns in data, while LSTM is proficient in learning temporal dependencies in sequential data. CNN-LSTM benefits from the spatial hierarchies and abstract features learned by CNNs, which can then be fed into LSTM layers for capturing temporal dependencies.

Contribution Division

Report:

- ✓ Hybrid: Zhou Danni
- ✓ CNN: Zhang Zhi
- ✓ LSTM: Yu Chengjie

Code:

- ✓ Hybrid: Zhou Danni, Tang Yixiang
- ✓ CNN: Zhang Zhi
- ✓ LSTM: Yu Chengjie