

Untitled

December 30, 2025

```
[1]: # Import the needed Lib and Check GPU readiness

import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'          # Hides standard TensorFlow
↳warnings
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'          # Hides the oneDNN message

import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers, callbacks
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, classification_report,
↳confusion_matrix
import matplotlib.pyplot as plt

# Check for GPU (RTX 3090)
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    print(f" GPU Detected: {gpus[0]}")
    try:
        tf.config.experimental.set_memory_growth(gpus[0], True)
    except RuntimeError as e:
        print(e)
else:
    print("No GPU detected. Running on CPU.")
```

GPU Detected: PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')

```
[2]: # Data Preprocessing and Feature Engineering

WINDOW_SIZE = 72 # Look at the past 24 hours of data
FEATURES = ['Flow Rate (m3/h)', 'Water Pressure (bar)', 'Value (m3)', 'Average
↳Consumption (m3/h)', 'Hour']

def load_and_process(filename, fit_scaler=False, scaler=None):
```

```

# 1. Load Data
df = pd.read_csv(filename)

# 2. Date Conversion & Feature Engineering
df['Reading Date'] = pd.to_datetime(df['Reading Date'])
df['Hour'] = df['Reading Date'].dt.hour
df = df.sort_values(by=['Meter SN', 'Reading Date'])

# 3. Simulate Labels (REPLACE THIS with your actual 'Label' column if
↪available)
# Rule: Leak if Flow Rate > 0.05 during night hours (2AM-4AM)
if 'Label' not in df.columns:
    df['Label'] = 0
    night_mask = (df['Hour'].between(2, 4)) & (df['Flow Rate (m3/h)'] > 0.
↪05)
    df.loc[night_mask, 'Label'] = 1
    # Propagate label to the whole day for that meter (simplification)
    # In real data, your 'Label' column should already exist.

# 4. Scale Features
data_features = df[FEATURES].values
if fit_scaler:
    scaler = StandardScaler()
    data_features = scaler.fit_transform(data_features)
else:
    data_features = scaler.transform(data_features)

# Re-attach scaled features to dataframe for sliding window
df_scaled = pd.DataFrame(data_features, columns=FEATURES, index=df.index)
df_scaled['Meter SN'] = df['Meter SN']
df_scaled['Label'] = df['Label']

# 5. Create Sliding Windows (Sequences)
X, y, meters = [], [], []

# Group by Meter to ensure we don't mix data from different houses
for meter, group in df_scaled.groupby('Meter SN'):
    values = group[FEATURES].values
    labels = group['Label'].values

    # Create sequences
    for i in range(len(values) - WINDOW_SIZE):
        X.append(values[i : i + WINDOW_SIZE])
        # Label is the status at the end of the window
        y.append(labels[i + WINDOW_SIZE])
        meters.append(meter)

```

```

    return np.array(X), np.array(y), np.array(meters), scaler, df

# Load your datasets (Assuming filenames are train.csv, test.csv, eval.csv)
# Using 'eval.csv' for all just to demonstrate. Replace with your actual files.
print("Loading Training Data...")
X_train, y_train, _, scaler, _ = load_and_process('eval.csv', fit_scaler=True)

print("Loading Test Data...")
X_test, y_test, _, _, _ = load_and_process('eval.csv', fit_scaler=False,
    ↪scaler=scaler)

print("Loading Eval Data...")
X_eval, y_eval, meters_eval, _, raw_eval_df = load_and_process('eval.csv',
    ↪fit_scaler=False, scaler=scaler)

print(f"Data Shape: {X_train.shape}") # (Samples, 24, 5)

```

Loading Training Data...

Loading Test Data...

Loading Eval Data...

Data Shape: (139200, 72, 5)

[3]: *# Buliding the model*

```

def build_alstm_fcn(input_shape, num_classes):
    input_layer = layers.Input(shape=input_shape)

    # --- Branch 1: Fully Convolutional Network (FCN) ---
    x = layers.Conv1D(128, 8, padding='same')(input_layer)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)

    x = layers.Conv1D(256, 5, padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)

    x = layers.Conv1D(128, 3, padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)

    x = layers.GlobalAveragePooling1D()(x)

    # --- Branch 2: Attention LSTM ---
    # Standard LSTM
    y = layers.LSTM(128, return_sequences=True)(input_layer)
    y = layers.Dropout(0.3)(y)

```

```

# Simple Attention Mechanism
# Computes a weighted sum of LSTM outputs
attention = layers.Dense(1, activation='tanh')(y)
attention = layers.Flatten()(attention)
attention = layers.Activation('softmax')(attention)
attention = layers.RepeatVector(128)(attention)
attention = layers.Permute([2, 1])(attention)

# Apply attention weights
sent_representation = layers.Multiply()([y, attention])
y = layers.Lambda(lambda z: tf.keras.backend.sum(z,
↪axis=1))(sent_representation)

# --- Combine Branches ---
combined = layers.Concatenate()([x, y])

# Output Layer
output = layers.Dense(1, activation='sigmoid')(combined) # Binary
↪Classification (Leak vs Normal)

model = models.Model(inputs=input_layer, outputs=output)
return model

# Initialize Model
model = build_alstm_fcn(input_shape=(WINDOW_SIZE, len(FEATURES)), num_classes=1)
model.compile(optimizer='adam', loss='binary_crossentropy',
↪metrics=['accuracy'])
model.summary()

```

WARNING: All log messages before absl::InitializeLog() is called are written to STDERR

I0000 00:00:1766225114.705095 36295 gpu_device.cc:2020] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 21768 MB memory: -> device:
0, name: NVIDIA GeForce RTX 3090, pci bus id: 0000:01:00.0, compute capability:
8.6

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 72, 5)	0	-
conv1d (Conv1D)	(None, 72, 128)	5,248	input_layer[0][0]
batch_normalization	(None, 72, 128)	512	conv1d[0][0]

(BatchNormalizatio...			
lstm (LSTM)	(None, 72, 128)	68,608	input_layer[0][0]
activation (Activation)	(None, 72, 128)	0	batch_normalizat...
dropout (Dropout)	(None, 72, 128)	0	lstm[0][0]
conv1d_1 (Conv1D)	(None, 72, 256)	164,096	activation[0][0]
dense (Dense)	(None, 72, 1)	129	dropout[0][0]
batch_normalizatio... (BatchNormalizatio...	(None, 72, 256)	1,024	conv1d_1[0][0]
flatten (Flatten)	(None, 72)	0	dense[0][0]
activation_1 (Activation)	(None, 72, 256)	0	batch_normalizat...
activation_3 (Activation)	(None, 72)	0	flatten[0][0]
conv1d_2 (Conv1D)	(None, 72, 128)	98,432	activation_1[0][...
repeat_vector (RepeatVector)	(None, 128, 72)	0	activation_3[0][...
batch_normalizatio... (BatchNormalizatio...	(None, 72, 128)	512	conv1d_2[0][0]
permute (Permute)	(None, 72, 128)	0	repeat_vector[0]...
activation_2 (Activation)	(None, 72, 128)	0	batch_normalizat...
multiply (Multiply)	(None, 72, 128)	0	dropout[0][0], permute[0][0]
global_average_poo... (GlobalAveragePool...	(None, 128)	0	activation_2[0][...
lambda (Lambda)	(None, 128)	0	multiply[0][0]
concatenate (Concatenate)	(None, 256)	0	global_average_p... lambda[0][0]

dense_1 (Dense) (None, 1) 257 concatenate[0][0]

Total params: 338,818 (1.29 MB)

Trainable params: 337,794 (1.29 MB)

Non-trainable params: 1,024 (4.00 KB)

```
[4]: # Train
history = model.fit(
    X_train, y_train,
    epochs=80,          # Increase to 50-100 for better results
    batch_size=64,      # Adjust based on memory
    validation_data=(X_test, y_test),
    callbacks=[tf.keras.callbacks.EarlyStopping(patience=5,
↪restore_best_weights=True)]
)

# Plot Training History
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Epoch 1/80

2175/2175 50s 21ms/step -

accuracy: 0.9940 - loss: 0.0190 - val_accuracy: 0.9981 - val_loss: 0.0052

Epoch 2/80

2175/2175 44s 20ms/step -

accuracy: 0.9983 - loss: 0.0052 - val_accuracy: 0.9994 - val_loss: 0.0052

Epoch 3/80

2175/2175 41s 19ms/step -

accuracy: 0.9992 - loss: 0.0029 - val_accuracy: 0.9958 - val_loss: 0.0138

Epoch 4/80

2175/2175 42s 19ms/step -

accuracy: 0.9993 - loss: 0.0024 - val_accuracy: 0.9996 - val_loss: 0.0018

Epoch 5/80

2175/2175 42s 19ms/step -

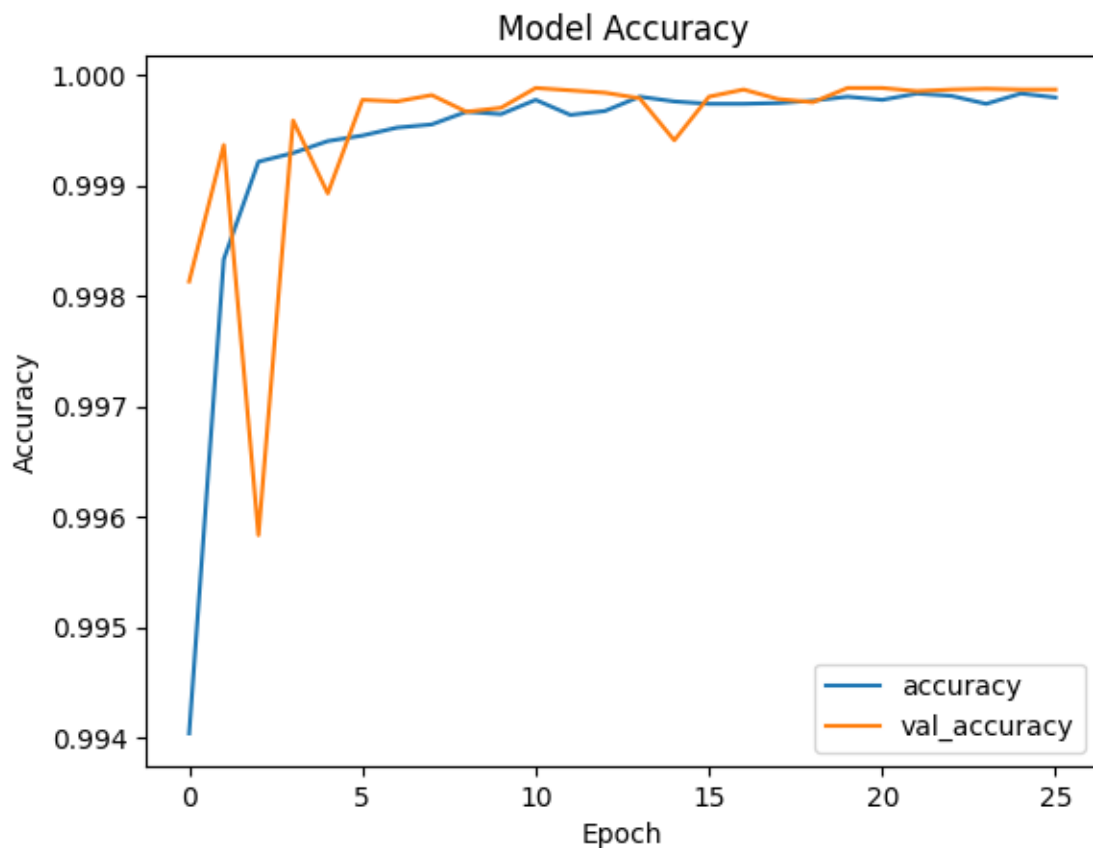
accuracy: 0.9994 - loss: 0.0021 - val_accuracy: 0.9989 - val_loss: 0.0086

Epoch 6/80

2175/2175 41s 19ms/step -

accuracy: 0.9995 - loss: 0.0019 - val_accuracy: 0.9998 - val_loss: 0.0016
 Epoch 7/80
 2175/2175 44s 20ms/step -
 accuracy: 0.9995 - loss: 0.0023 - val_accuracy: 0.9998 - val_loss: 0.0014
 Epoch 8/80
 2175/2175 42s 20ms/step -
 accuracy: 0.9996 - loss: 0.0017 - val_accuracy: 0.9998 - val_loss: 0.0011
 Epoch 9/80
 2175/2175 41s 19ms/step -
 accuracy: 0.9997 - loss: 0.0013 - val_accuracy: 0.9997 - val_loss: 0.0013
 Epoch 10/80
 2175/2175 44s 20ms/step -
 accuracy: 0.9996 - loss: 0.0014 - val_accuracy: 0.9997 - val_loss: 0.0014
 Epoch 11/80
 2175/2175 42s 19ms/step -
 accuracy: 0.9998 - loss: 0.0011 - val_accuracy: 0.9999 - val_loss: 8.5425e-04
 Epoch 12/80
 2175/2175 41s 19ms/step -
 accuracy: 0.9996 - loss: 0.0013 - val_accuracy: 0.9999 - val_loss: 8.4216e-04
 Epoch 13/80
 2175/2175 42s 19ms/step -
 accuracy: 0.9997 - loss: 0.0014 - val_accuracy: 0.9998 - val_loss: 8.8984e-04
 Epoch 14/80
 2175/2175 43s 20ms/step -
 accuracy: 0.9998 - loss: 0.0011 - val_accuracy: 0.9998 - val_loss: 8.4965e-04
 Epoch 15/80
 2175/2175 44s 20ms/step -
 accuracy: 0.9998 - loss: 0.0010 - val_accuracy: 0.9994 - val_loss: 0.0022
 Epoch 16/80
 2175/2175 42s 19ms/step -
 accuracy: 0.9997 - loss: 0.0011 - val_accuracy: 0.9998 - val_loss: 0.0011
 Epoch 17/80
 2175/2175 42s 19ms/step -
 accuracy: 0.9997 - loss: 0.0014 - val_accuracy: 0.9999 - val_loss: 8.2870e-04
 Epoch 18/80
 2175/2175 43s 20ms/step -
 accuracy: 0.9997 - loss: 0.0010 - val_accuracy: 0.9998 - val_loss: 9.7337e-04
 Epoch 19/80
 2175/2175 43s 20ms/step -
 accuracy: 0.9998 - loss: 0.0011 - val_accuracy: 0.9998 - val_loss: 0.0013
 Epoch 20/80
 2175/2175 40s 18ms/step -
 accuracy: 0.9998 - loss: 9.6652e-04 - val_accuracy: 0.9999 - val_loss:
 7.4847e-04
 Epoch 21/80
 2175/2175 42s 19ms/step -
 accuracy: 0.9998 - loss: 0.0010 - val_accuracy: 0.9999 - val_loss: 6.0843e-04
 Epoch 22/80

2175/2175 44s 20ms/step -
accuracy: 0.9998 - loss: 8.6598e-04 - val_accuracy: 0.9999 - val_loss:
7.2677e-04
Epoch 23/80
2175/2175 42s 19ms/step -
accuracy: 0.9998 - loss: 8.7312e-04 - val_accuracy: 0.9999 - val_loss:
9.9854e-04
Epoch 24/80
2175/2175 42s 19ms/step -
accuracy: 0.9997 - loss: 0.0010 - val_accuracy: 0.9999 - val_loss: 6.6796e-04
Epoch 25/80
2175/2175 42s 19ms/step -
accuracy: 0.9998 - loss: 7.3651e-04 - val_accuracy: 0.9999 - val_loss:
7.4111e-04
Epoch 26/80
2175/2175 44s 20ms/step -
accuracy: 0.9998 - loss: 8.6255e-04 - val_accuracy: 0.9999 - val_loss:
6.0895e-04



```
[5]: # 1. Make Predictions
y_pred_prob = model.predict(X_eval)
y_pred = (y_pred_prob > 0.4).astype(int).flatten()

# 2. Calculate Accuracy Metrics
print("\n--- Model Performance ---")
print(f"Accuracy: {accuracy_score(y_eval, y_pred):.4f}")
print("Confusion Matrix:")
print(confusion_matrix(y_eval, y_pred))
print("\nClassification Report:")
print(classification_report(y_eval, y_pred))

# 3. Identify Where the Leakage Is
# We map the predictions back to the Meter SN and get their location
results_df = pd.DataFrame({
    'Meter SN': meters_eval,
    'Prediction': y_pred,
    'Probability': y_pred_prob.flatten()
})

# Get unique meters predicted as Leaking
leaking_meters = results_df[results_df['Prediction'] == 1]['Meter SN'].unique()

# Merge with location data (Latitude, Longitude) from original file
locations = raw_eval_df[['Meter SN', 'Latitude', 'Longitude']].drop_duplicates()
leaks_with_location = locations[locations['Meter SN'].isin(leaking_meters)]

print(f"\n--- Detected {len(leaks_with_location)} Leaking Meters ---")
print(leaks_with_location.head(10))

# Save to CSV
leaks_with_location.to_csv('predicted_leaks.csv', index=False)
print("\nLeak locations saved to 'predicted_leaks.csv'")
```

4350/4350

14s 3ms/step

--- Model Performance ---

Accuracy: 0.9999

Confusion Matrix:

```
[[137694    17]
 [      0   1489]]
```

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	137711
1	0.99	1.00	0.99	1489

accuracy			1.00	139200
macro avg	0.99	1.00	1.00	139200
weighted avg	1.00	1.00	1.00	139200

--- Detected 75 Leaking Meters ---

	Meter SN	Latitude	Longitude
1	RSN-00002	24.60495	46.68658
2	RSN-00003	24.73063	46.66891
5	RSN-00006	24.71864	46.66285
6	RSN-00007	24.61738	46.69064
7	RSN-00008	24.70073	46.59549
8	RSN-00009	24.81314	46.71777
10	RSN-00011	24.78774	46.70619
11	RSN-00012	24.70717	46.80956
12	RSN-00013	24.73079	46.67696
13	RSN-00014	24.71839	46.58423

Leak locations saved to 'predicted_leaks.csv'

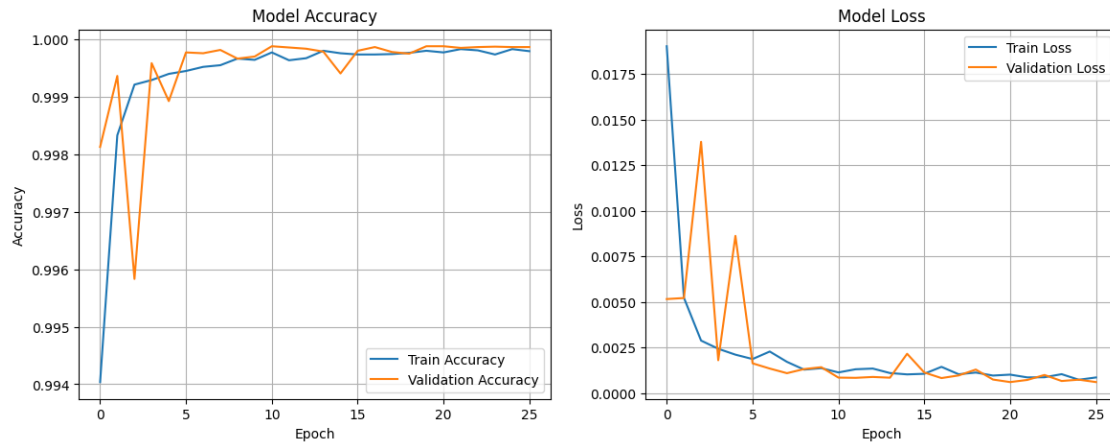
```
[6]: import matplotlib.pyplot as plt

# Plot Accuracy
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

# Plot Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

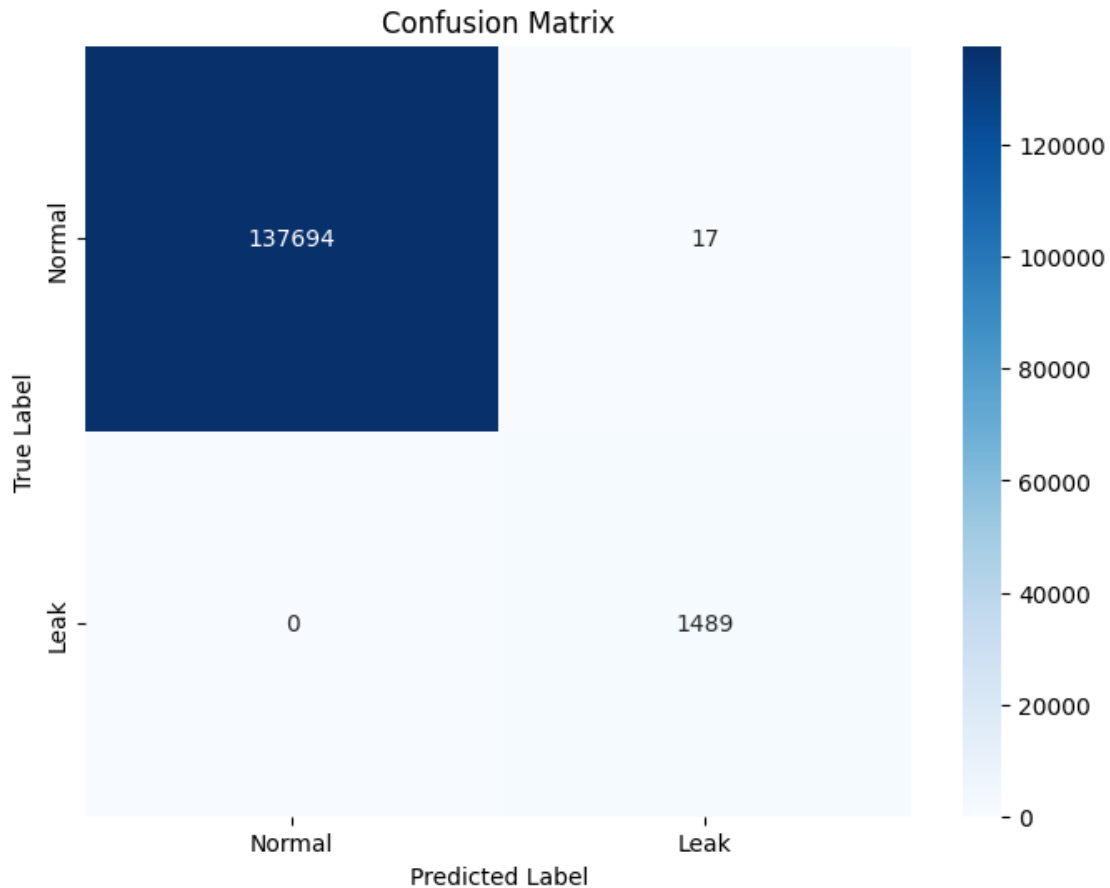
plt.show()
```



```
[7]: import seaborn as sns
from sklearn.metrics import confusion_matrix

# Calculate Confusion Matrix
cm = confusion_matrix(y_eval, y_pred)

# Plot Heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Normal', 'Leak'],
            yticklabels=['Normal', 'Leak'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



```
[8]: import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, precision_recall_fscore_support

y_pred_prob = model.predict(X_eval)
y_pred = (y_pred_prob > 0.5).astype(int).flatten()

# 1. Calculate Overall Metrics
# We use 'weighted' average to account for class imbalance
accuracy = accuracy_score(y_eval, y_pred)
precision, recall, f1, _ = precision_recall_fscore_support(y_eval, y_pred,
    ↪average='weighted')

# 2. Prepare Data for Plotting
metrics_data = {
    'Accuracy': [accuracy],
    'Precision': [precision],
```

```

    'Recall': [recall],
    'F1-Score': [f1]
}
metrics_df = pd.DataFrame(metrics_data)

# 3. Plot the Heatmap
plt.figure(figsize=(10, 2))
sns.heatmap(metrics_df, annot=True, fmt='.4f', cmap='viridis',
            cbar=False, linewidths=2, linecolor='white',
            annot_kws={"size": 14, "weight": "bold"})

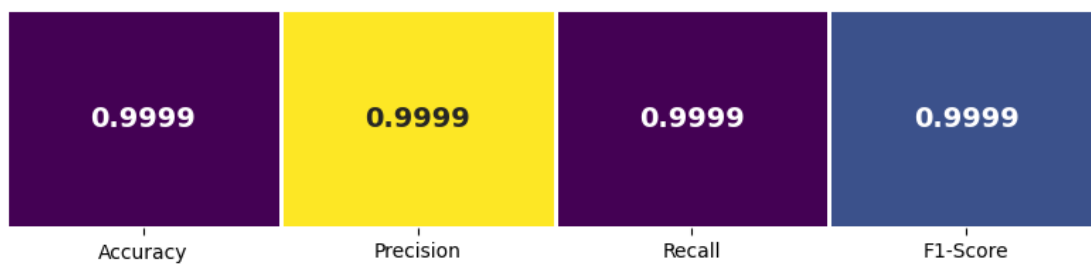
plt.title('Overall Model Performance Metrics', fontsize=16, pad=20)
plt.yticks([])
plt.show()

```

4350/4350

15s 4ms/step

Overall Model Performance Metrics



```

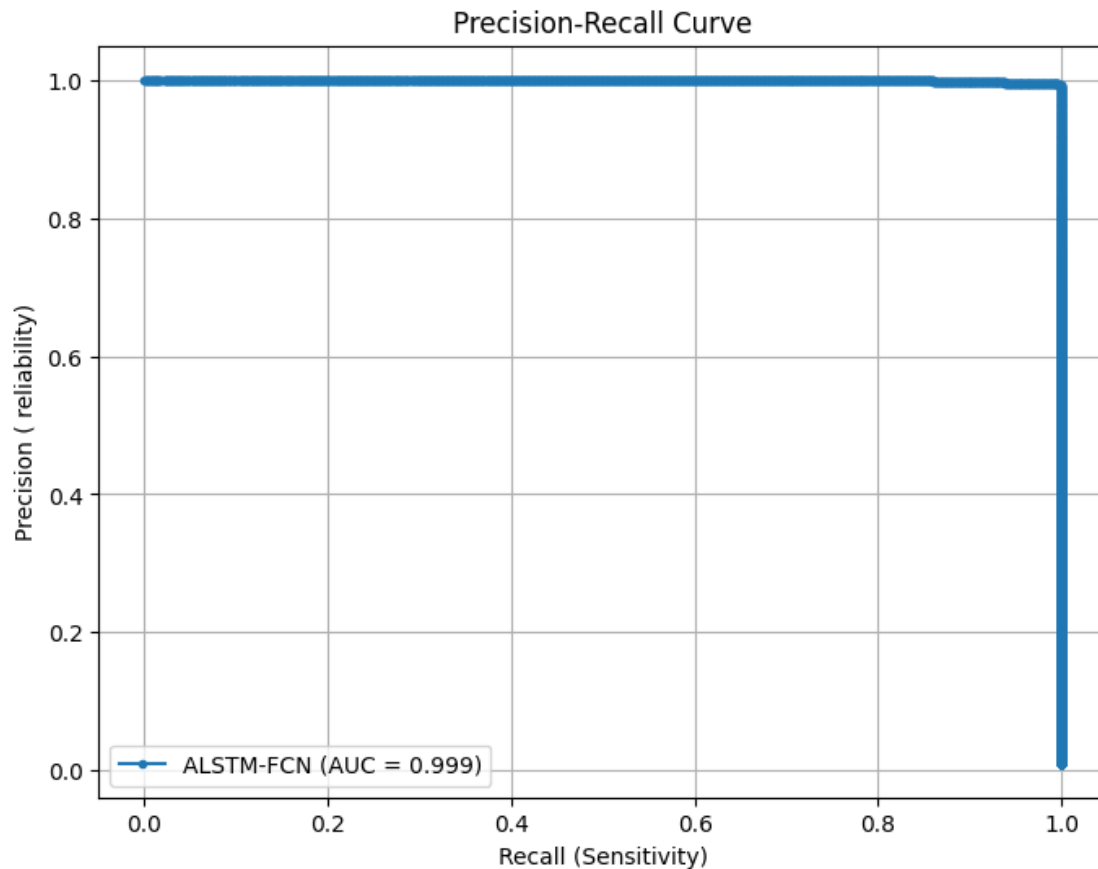
[17]: from sklearn.metrics import precision_recall_curve, auc

# Get probabilities for the "Leak" class (1)
y_scores = y_pred_prob.flatten()

precision, recall, thresholds = precision_recall_curve(y_eval, y_scores)
auc_score = auc(recall, precision)

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, marker='.', label=f'ALSTM-FCN (AUC = {auc_score:.3f})')
plt.xlabel('Recall (Sensitivity)')
plt.ylabel('Precision (reliability)')
plt.title('Precision-Recall Curve')
plt.legend()
plt.grid(True)
plt.show()

```



```
[9]: !pip install folium

import folium

# 1. Prepare Data for Mapping
# We take the unique locations of all meters from the original dataframe
unique_locations = raw_eval_df[['Meter SN', 'Latitude', 'Longitude']].
    ↪drop_duplicates()

# We determine the status: If the meter was in our 'leaking_meters' list, it's
    ↪RED, otherwise GREEN.
# (Logic: If a meter had at least one leaking timestamp in the eval period, we
    ↪flag it.)
unique_locations['Status'] = unique_locations['Meter SN'].apply(
    lambda x: 'Leak' if x in leaking_meters else 'Normal'
)

# 2. Initialize Map
# Center the map on the average location of your meters
```

```

map_center = [unique_locations['Latitude'].mean(),
               ↪unique_locations['Longitude'].mean()]
my_map = folium.Map(location=map_center, zoom_start=12, tiles='OpenStreetMap')

# 3. Add Markers
for _, row in unique_locations.iterrows():
    # Set colors
    if row['Status'] == 'Leak':
        color = 'red'
        fill_color = '#ff3333' # Lighter red fill
        popup_msg = f"ALERT: LEAK DETECTED<br>Meter: {row['Meter SN']}"
    else:
        color = 'green'
        fill_color = '#33ff33' # Lighter green fill
        popup_msg = f"Normal<br>Meter: {row['Meter SN']}"

    # Create the marker
    folium.CircleMarker(
        location=[row['Latitude'], row['Longitude']],
        radius=8,
        color=color,
        weight=2,
        fill=True,
        fill_color=fill_color,
        fill_opacity=0.7,
        popup=folium.Popup(popup_msg, max_width=250)
    ).add_to(my_map)

# 4. Display Legend (Optional simple HTML legend)
legend_html = '''
    <div style="position: fixed;
    bottom: 50px; left: 50px; width: 150px; height: 90px;
    border: 2px solid grey; z-index: 9999; font-size: 14px;
    background-color: white; opacity: 0.9; padding: 10px;">
    <b>Status Legend</b><br>
    <i style="background: #ff3333; width: 10px; height: 10px; display:
    ↪inline-block; border-radius: 50%;"></i>&nbsp; Leak Detected<br>
    <i style="background: #33ff33; width: 10px; height: 10px; display:
    ↪inline-block; border-radius: 50%;"></i>&nbsp; Normal
    </div>
    '''

my_map.get_root().html.add_child(folium.Element(legend_html))

# Show Map
my_map

```

Requirement already satisfied: folium in ./venv/lib/python3.12/site-packages

(0.20.0)
Requirement already satisfied: branca>=0.6.0 in ./venv/lib/python3.12/site-packages (from folium) (0.8.2)
Requirement already satisfied: jinja2>=2.9 in ./venv/lib/python3.12/site-packages (from folium) (3.1.6)
Requirement already satisfied: numpy in ./venv/lib/python3.12/site-packages (from folium) (1.26.4)
Requirement already satisfied: requests in ./venv/lib/python3.12/site-packages (from folium) (2.32.5)
Requirement already satisfied: xyzservices in ./venv/lib/python3.12/site-packages (from folium) (2025.11.0)
Requirement already satisfied: MarkupSafe>=2.0 in ./venv/lib/python3.12/site-packages (from jinja2>=2.9->folium) (3.0.3)
Requirement already satisfied: charset_normalizer<4,>=2 in ./venv/lib/python3.12/site-packages (from requests->folium) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in ./venv/lib/python3.12/site-packages (from requests->folium) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in ./venv/lib/python3.12/site-packages (from requests->folium) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in ./venv/lib/python3.12/site-packages (from requests->folium) (2025.11.12)

[9]: <folium.folium.Map at 0x751960716300>

[]:

[]: