Deep Learning-based Leak Detection for Smart Water Metering with Consumption Forecasting

Problem Identified

DENR airs concern over water leaks in houses, buildings

Bella Cariaso - The Philippine Star

May 9, 2024 | 12:00am

0

Almost 1.7 cubic meters per second of water lost due to pipe leaks —Manila Water

By GISELLE OMBAY, GMA Integrated News

Published April 3, 2023 10:15am
Updated April 3, 2023 11:36am

Problem Statement

Water pipeline leaks cause significant losses in urban systems, yet traditional detection methods are often manual, inaccurate, and inefficient. This project proposes a deep learning-based approach that combines automated leak detection and water consumption forecasting using images and numerical data. Our goal is to enhance detection accuracy, and support smarter, real-time water management.

Dataset Chosed

OBJECT DETECTION

Pipeline Leak Prediction (Roboflow)
https://universe.roboflow.com/gas-leak/pipeline-leak-prediction

Contents:
images of pipelines that has leaks and no leaks

FOR LSTM

A synthetic dataset made of water consumption consisting 10 thousands of sets of it.

Methodology

Dual-Model Approach: Object Detection and Time Series Analysis

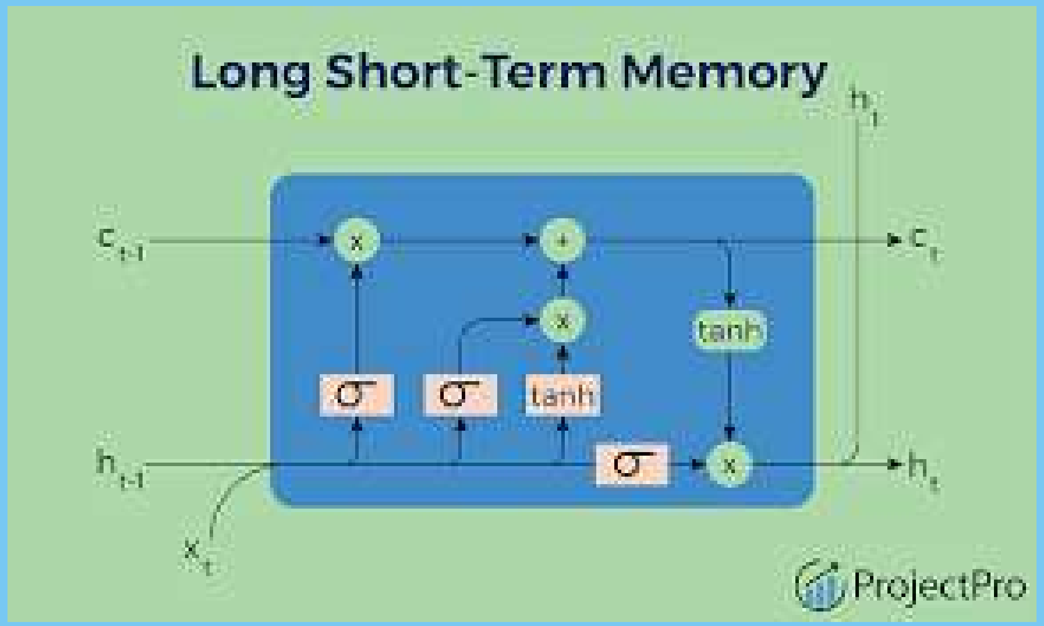
Leak Detection Model

ultralytics YOLO11

Detectron2

RetinaNet

Consumption Forecasting Model



TSN Time Sensitive™ Networking Powered by IEEE 802

Dataset prep for Yolov11 Model

Pipeline Leak Prediction Dataset (from Roboflow)

SET	Total Images	Leak	No leak	Water	Crack
Training	2739	1099	623	847	170
Validation	488	193	91	168	36
Total	**3227**	**1292**	**714**	**1015**	**206**

Data Augmentation already applied

85% training
15% validation

Preprocessed via Roboflow

Training of Objection Detection Models

ultralytics YOLO11

- enhance feature extraction + object localization
- high inference speed and improved detection precision

Detectron2

- fast training on single or multiple GPU servers and includes high-quality implementations of numerous advanced models like Mask R-CNN, Cascade R-CNN, and Panoptic FPN.

RetinaNet

- Addresses Class Imbalance in One-Stage Detectors
- Balances Speed and Accuracy

Evaluating the performance of Leak Detection Models

ultralytics YOLO11

Detectron2

RetinaNet

mAP50

Precision

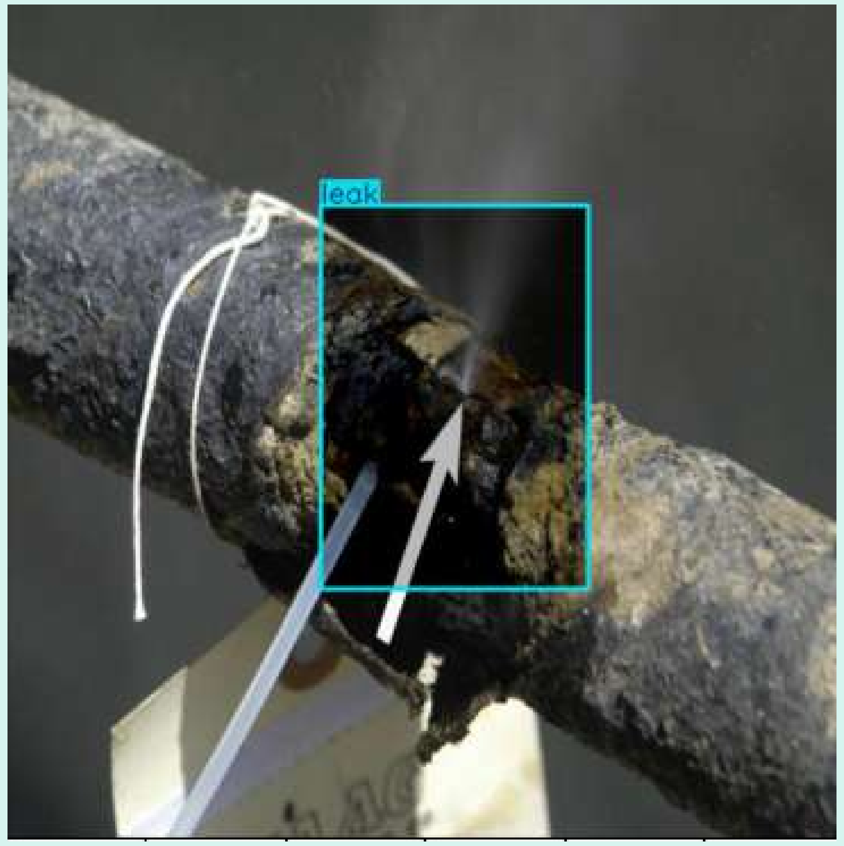
Recall

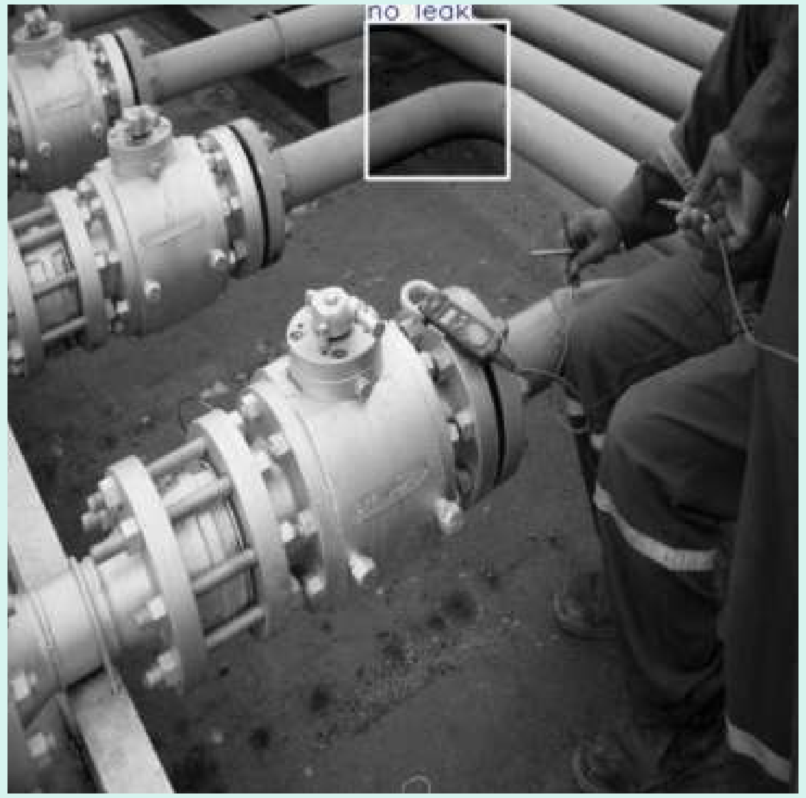
F1 Score

Deployment of Leak Detection Model in Streamlit

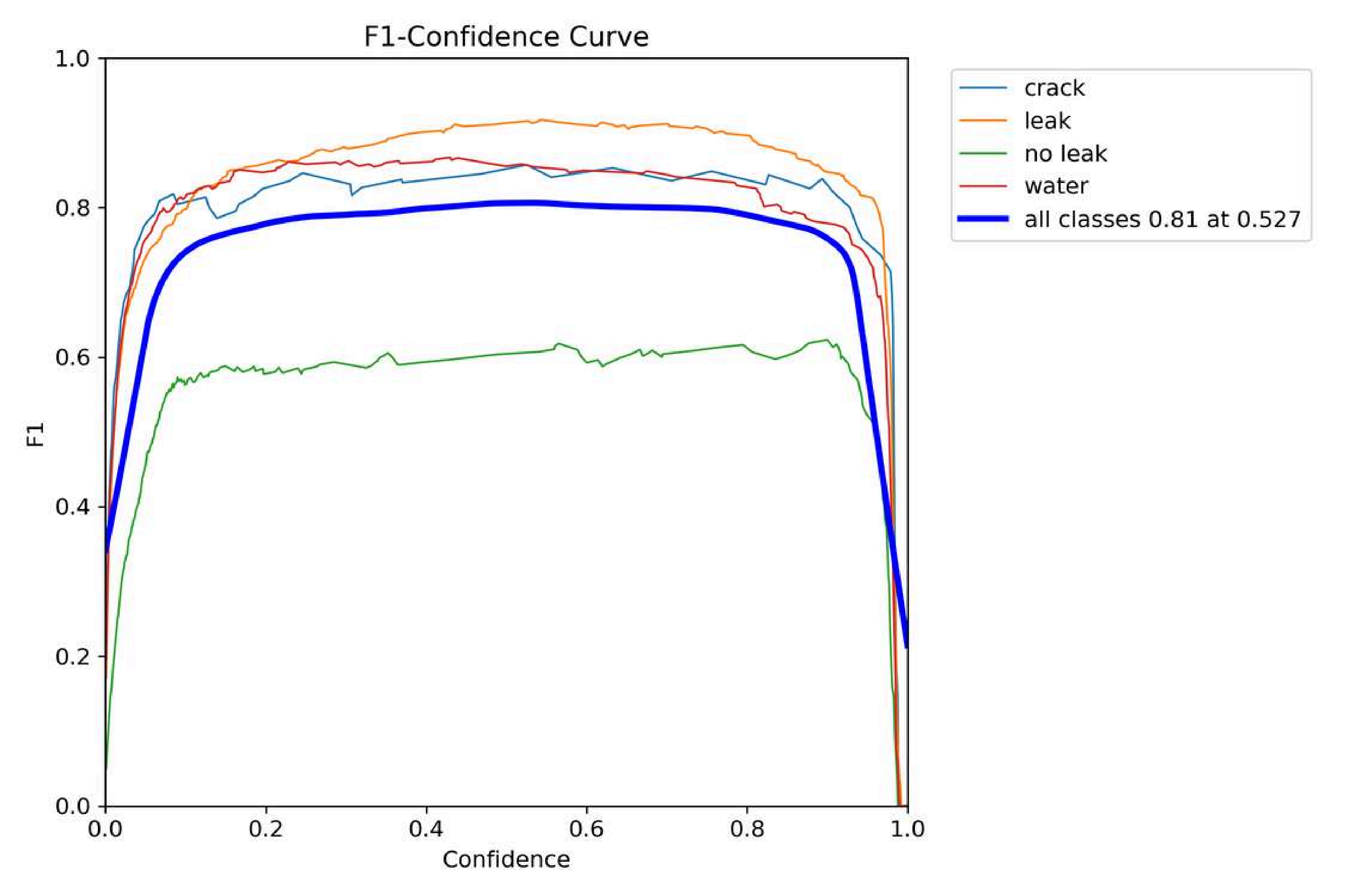
Results and Discussion

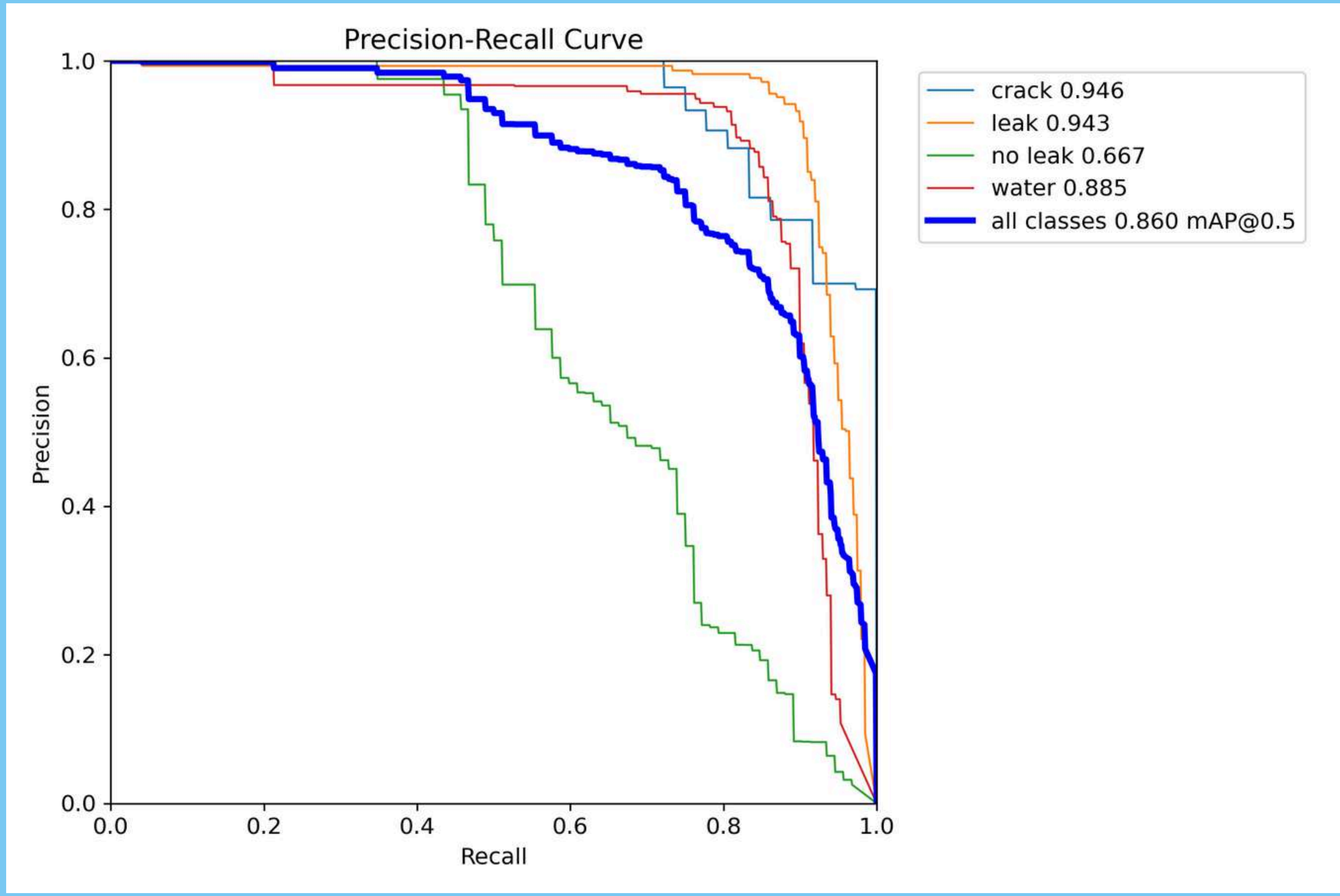
Leak Detection Model Performance Evaluation





Performance Metrics of YOLOv11 model





- peaking at 0.81 around a confidence threshold of ~0.527. This shows good indication that the yolo model performs reliably when it is reasonably confident

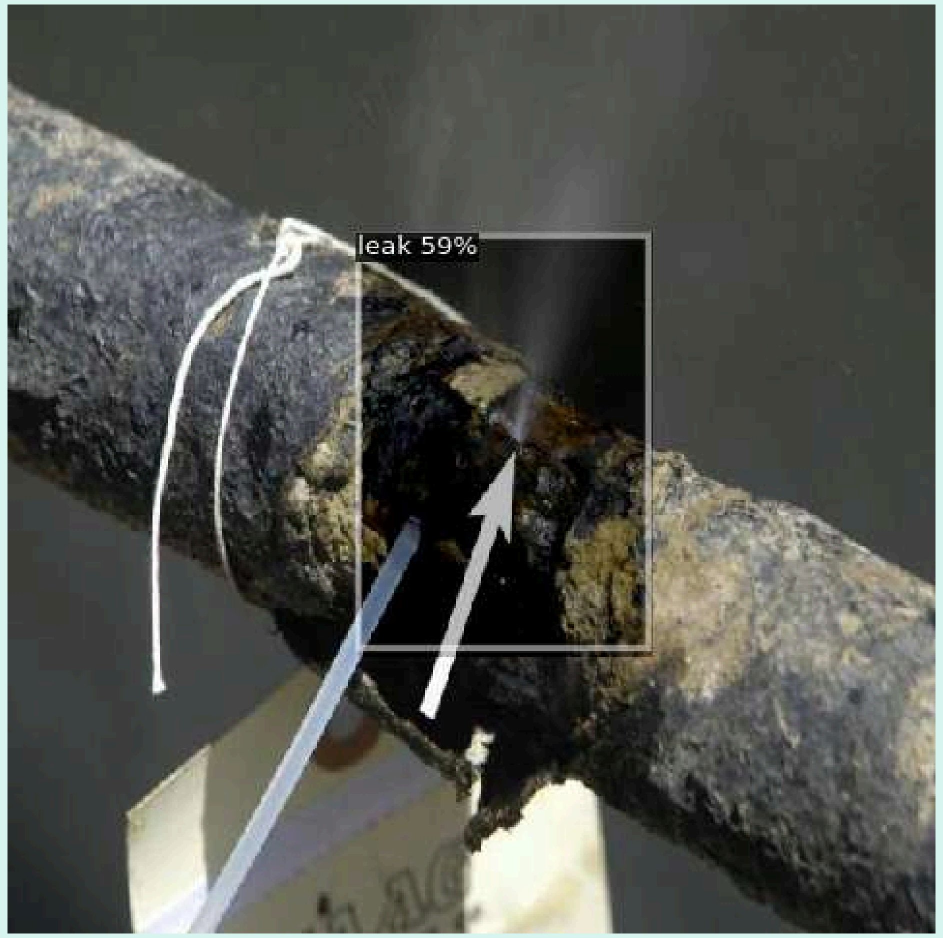
- Precision and recall trade-offs are well balanced for most classes except “no leak.”
- An mAP@0.5 of .860 indicates good object localization and classification confidence, particularly for leak-related targets.

Performance Metrics of YOLOv11 model

YOLOv11 Model Performance Evaluation				
Class	Images	Instances	R	mAP50
all	502	492	0.767	0.859
crack	36	36	0.832	0.946
leak	193	199	0.899	0.942
no leak	91	92	0.554	0.663
water	168	169	0.783	0.885

1st test of Detectron2 model

```
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("leak_dataset_train",)
cfg.DATASETS.TEST = ("leak_dataset_val",)
cfg.DATALOADER.NUM_WORKERS = 2
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml")
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.00125
cfg.SOLVER.MAX_ITER = 2000
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128
```



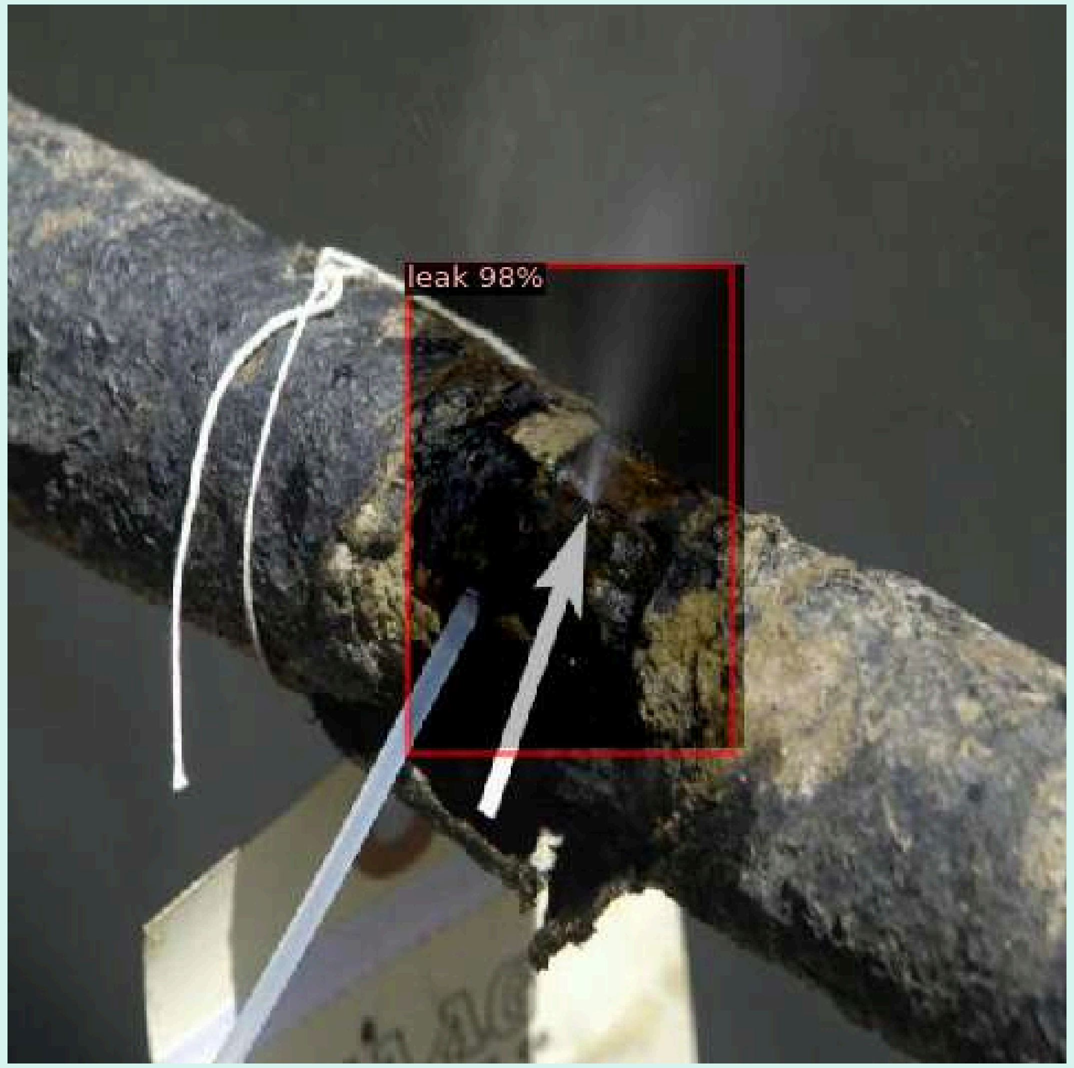
1st test of Detectron2 model

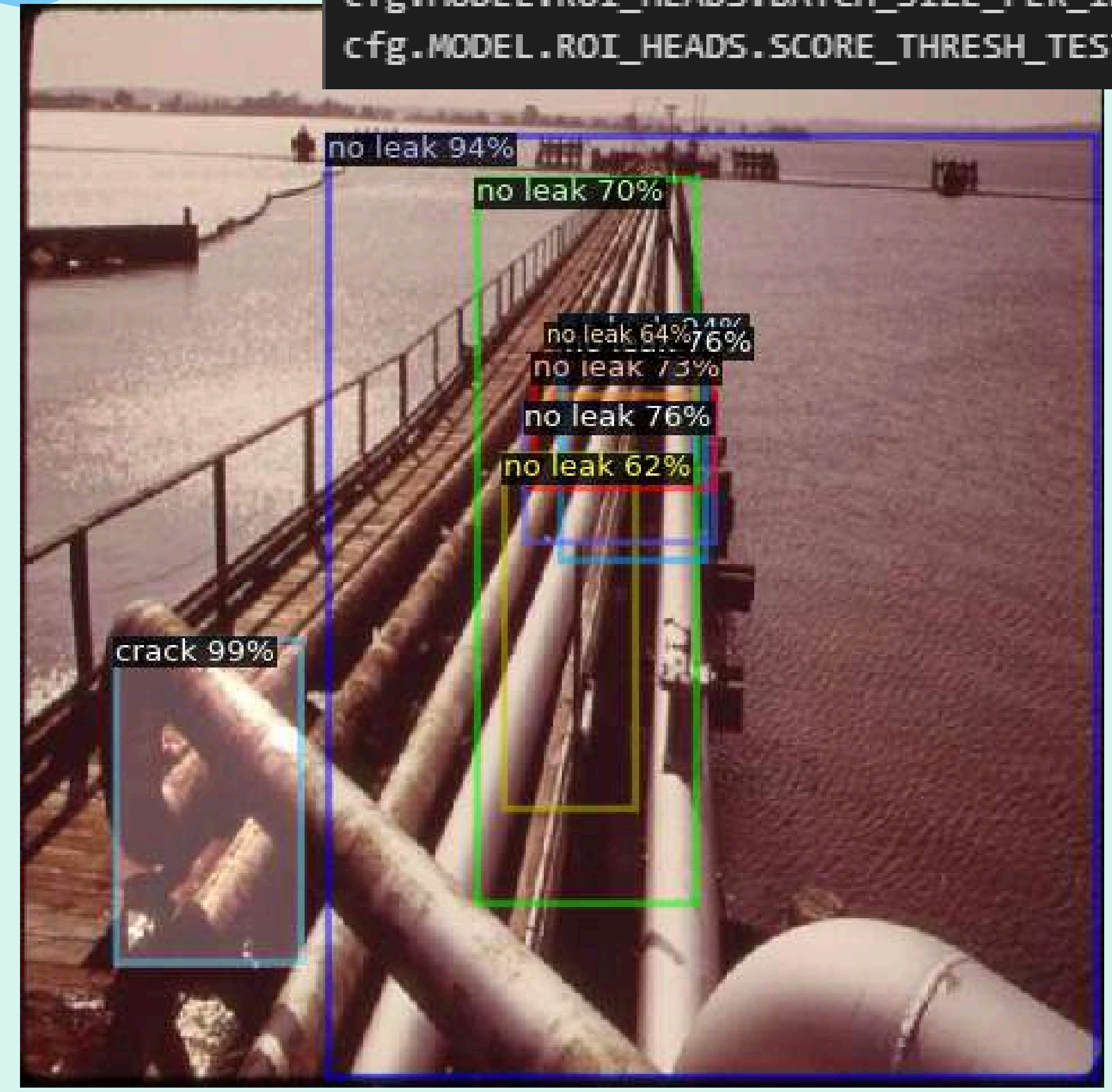
1st test of Detectron2 model

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.344
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.497
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.370
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.212
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.348
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.534
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.622
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.631
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.268
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.654
[05/22 16:31:55 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs  |  APm   |  APl   |
|:------:|:------:|:------:|:-----:|:------:|:------:|
| 34.432 | 49.703 | 36.961 |  nan  | 21.236 | 34.785 |
[05/22 16:31:55 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
[05/22 16:31:55 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP     | category   | AP     | category   | AP     |
|:-----------|:-------|:-----------|:-------|:-----------|:-------|
| leakage    | nan    | crack      | 42.730 | leak       | 48.876 |
| no leak    | 10.821 | water      | 35.301 |            |        |
OrderedDict([('bbox',
              {'AP': 34.43207423228552,
               'AP50': 49.70307085589206,
               'AP75': 36.961087166617936,
               'APs': nan,
               'APm': 21.23628686307657,
               'APl': 34.78472621458232,
               'AP-leakage': nan,
               'AP-crack': 42.73049901827959,
               'AP-leak': 48.876384873901344,
               'AP-no leak': 10.820891749531853,
               'AP-water': 35.3005212874293})])
```

2nd test of Detectron2 model

```python
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml"))

cfg.DATASETS.TRAIN = ("leak_dataset_train",)
cfg.DATASETS.TEST = ("leak_dataset_val",)
cfg.DATALOADER.NUM_WORKERS = 2

cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml")
cfg.SOLVER.IMS_PER_BATCH = 4
cfg.SOLVER.BASE_LR = 0.00125
cfg.SOLVER.MAX_ITER = 5000
cfg.SOLVER.STEPS = []
cfg.TEST.EVAL_PERIOD = 500
cfg.SOLVER.OPTIMIZER = "ADAMW"
cfg.SOLVER.WEIGHT_DECAY = 0.0005
cfg.SOLVER.AMSGRAD = True
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 32
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5
```





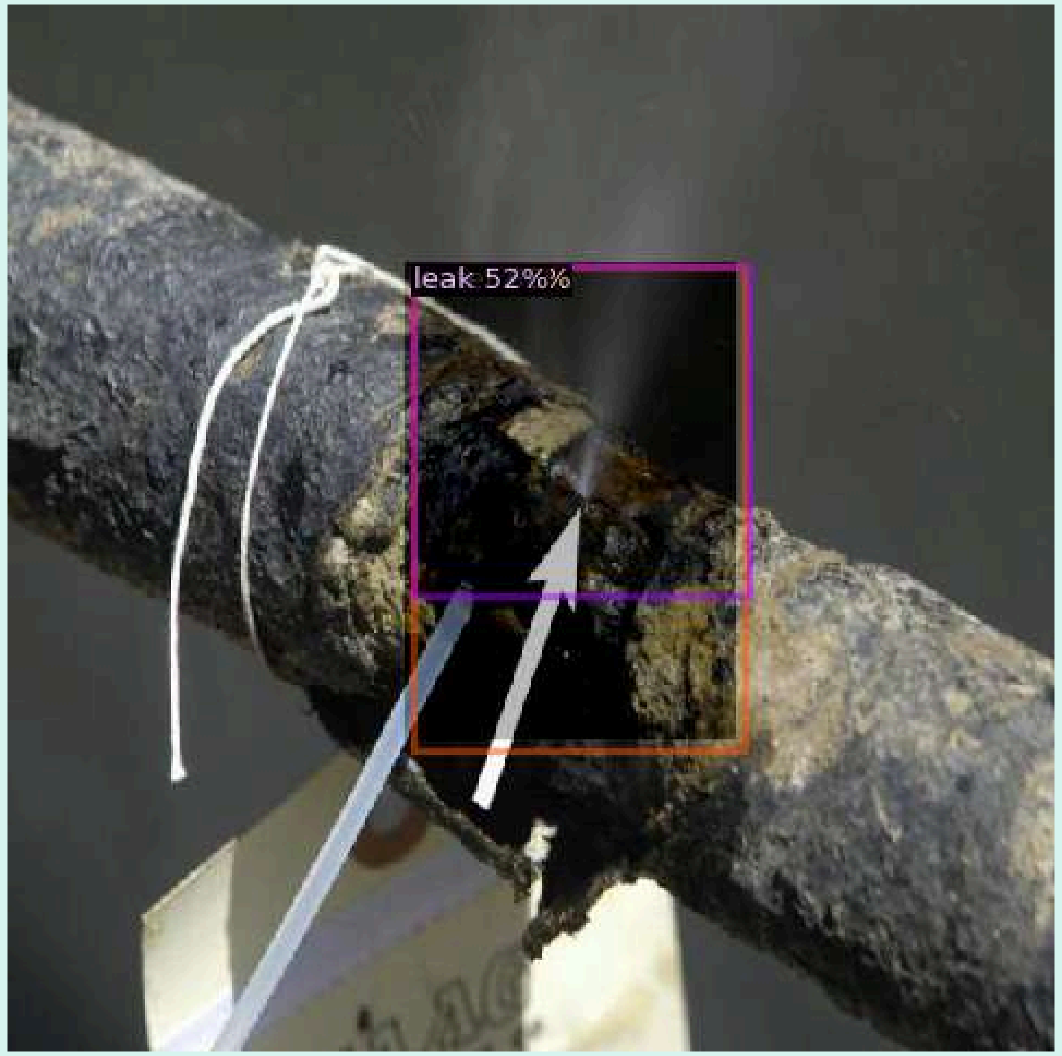
2nd test of Dectectron 2 model

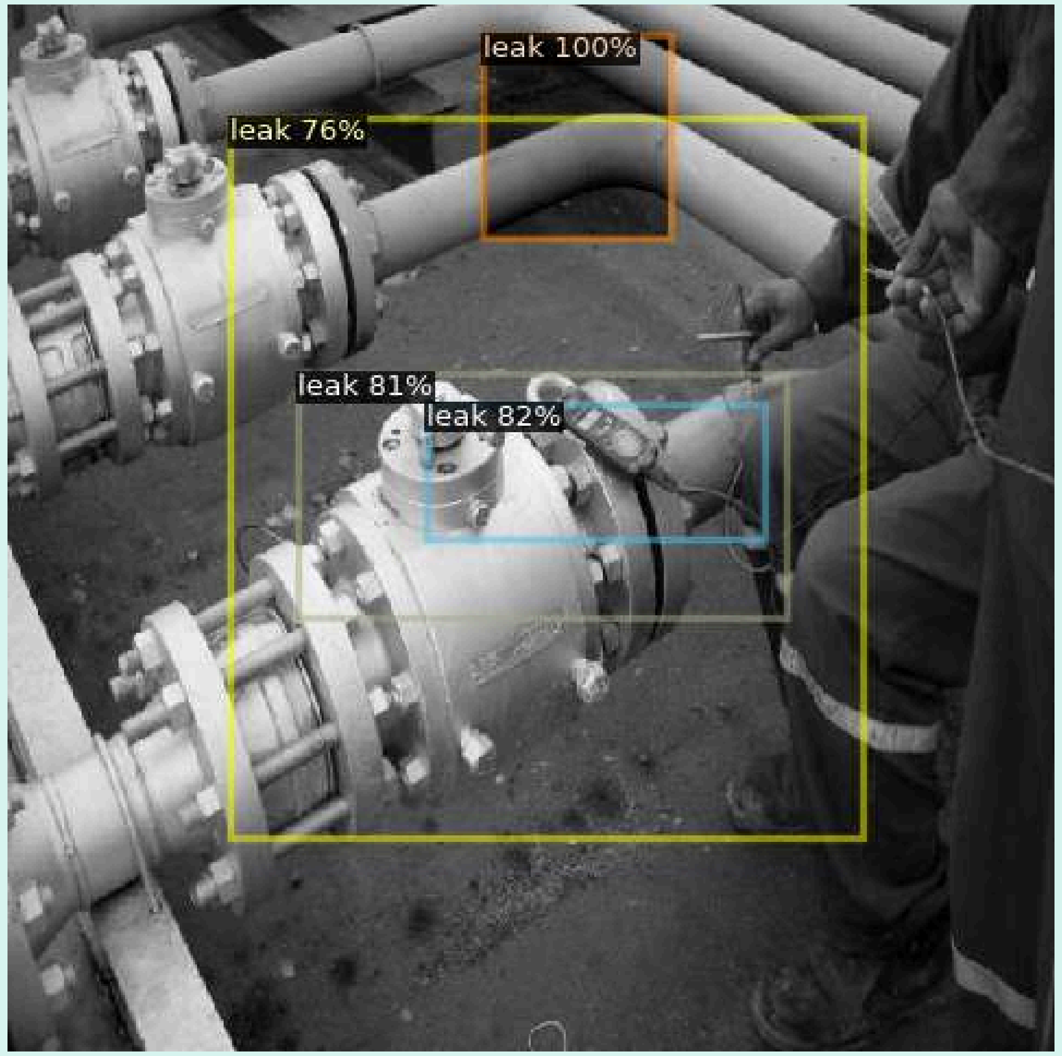
2nd test of Detectron2 model

```
[05/22 17:53:23 d2.evaluation.fast_eval_api]: COCOeval_opt.accumulate() finished in 0.02 seconds.
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.571
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.742
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.608
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.375
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.592
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.651
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.700
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.700
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.535
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.716
[05/22 17:53:23 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs  |  APm   |  APl   |
|:------:|:------:|:------:|:-----:|:------:|:------:|
| 57.051 | 74.164 | 60.793 |  nan  | 37.521 | 59.171 |
[05/22 17:53:23 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
[05/22 17:53:23 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP     | category   | AP     | category   | AP     |
|:-----------|:-------|:-----------|:-------|:-----------|:-------|
| leakage    | nan    | crack      | 63.591 | leak       | 67.730 |
| no leak    | 38.043 | water      | 58.841 |            |        |
```

1st test on Retinanet Model

```
# Load configuration from a RetinaNet config file
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/retinanet_R_50_FPN_3x.yaml"))

cfg.DATASETS.TRAIN = ("leak_dataset_train",)
cfg.DATASETS.TEST = ("leak_dataset_val",)
cfg.DATALOADER.NUM_WORKERS = 2

# Load weights from a RetinaNet checkpoint
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/retinanet_R_50_FPN_3x.yaml")

cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.00125
cfg.SOLVER.MAX_ITER = 2000
num_classes_from_dataset = len(leak_metadata.thing_classes)
cfg.MODEL.RETINANET.NUM_CLASSES = num_classes_from_dataset
```



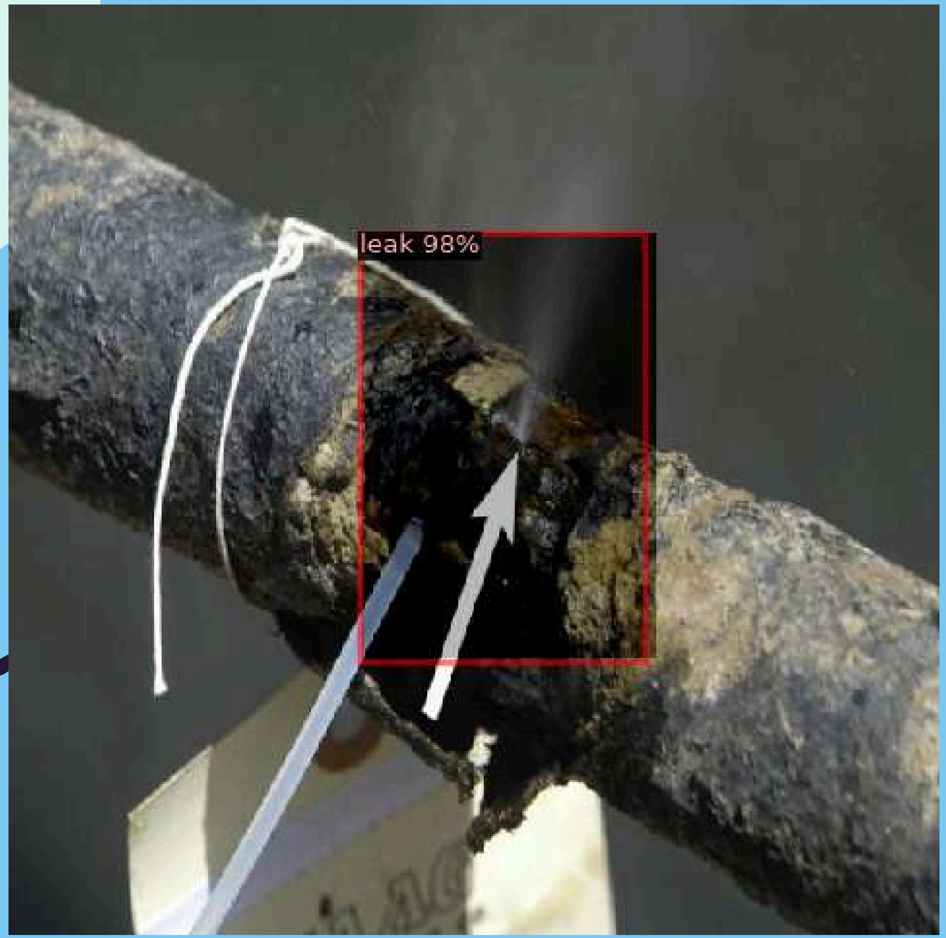


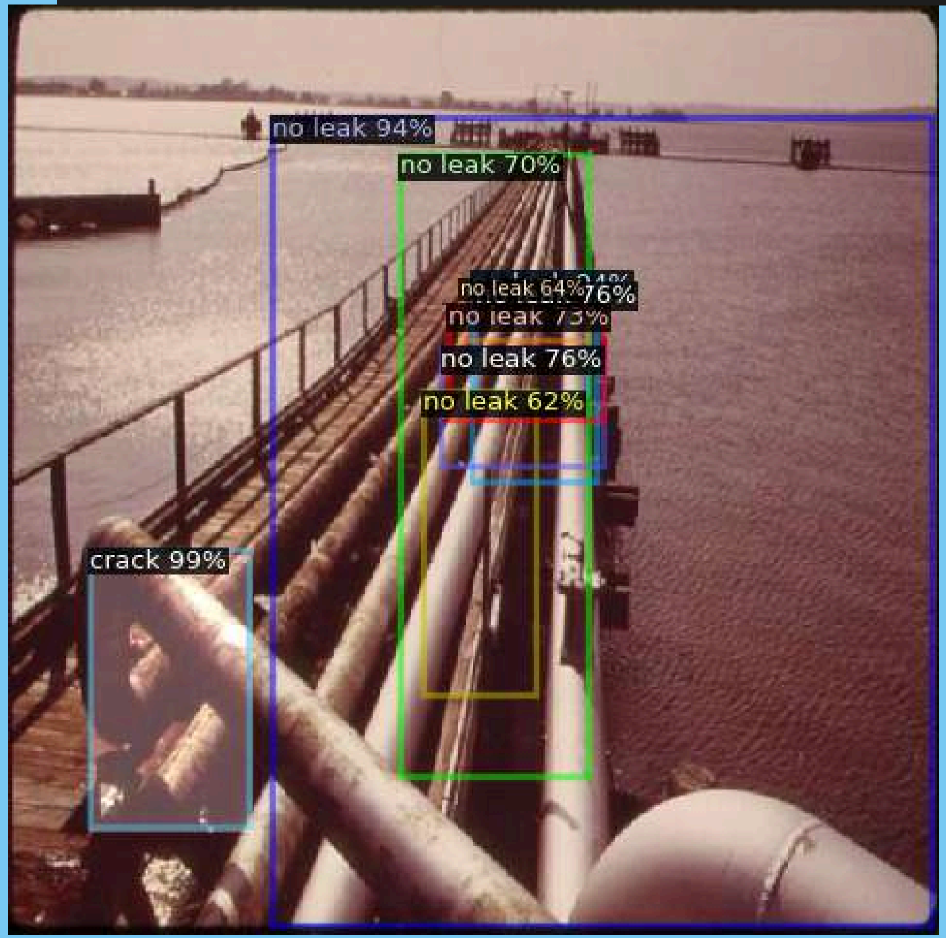
1st test on Retinanet Model

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.411
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.580
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.437
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.253
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.416
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.520
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.622
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.633
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.324
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.650
[05/18 18:53:10 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs  |  APm   |  APl   |
|:------:|:------:|:------:|:-----:|:------:|:------:|
| 41.078 | 58.032 | 43.691 |  nan  | 25.302 | 41.558 |
[05/18 18:53:10 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
[05/18 18:53:10 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP     | category   | AP     | category   | AP     |
|:-----------|:-------|:-----------|:-------|:-----------|:-------|
| leakage    | nan    | crack      | 44.277 | leak       | 57.518 |
| no leak    | 14.504 | water      | 48.013 |            |        |
OrderedDict([('bbox',
```

2nd test of Retinanet model

```
cfg_tuned = get_cfg()
cfg_tuned.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml"))
cfg_tuned.DATASETS.TRAIN = ("leak_dataset_train",)
cfg_tuned.DATASETS.TEST = ("leak_dataset_val",)
cfg_tuned.DATALOADER.NUM_WORKERS = 2 # Increased number of workers
cfg_tuned.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml")
cfg_tuned.SOLVER.IMS_PER_BATCH = 4 # Increased batch size
cfg_tuned.SOLVER.BASE_LR = 0.00125  # Increased learning rate
cfg_tuned.SOLVER.MAX_ITER = 5000  # Increased iterations
cfg_tuned.SOLVER.STEPS = (4000, 4500) # Add learning rate decay steps
cfg.SOLVER.OPTIMIZER = "ADAMW"
cfg.TEST.EVAL_PERIOD = 500
cfg.SOLVER.WEIGHT_DECAY = 0.0005
cfg_tuned.SOLVER.GAMMA = 0.1 # Learning rate decay factor
cfg_tuned.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128 # Increased RoIHead batch size
cfg_tuned.MODEL.ROI_HEADS.NUM_CLASSES = num_classes_from_dataset

os.makedirs(cfg_tuned.OUTPUT_DIR, exist_ok=True)
trainer_tuned = DefaultTrainer(cfg_tuned)
trainer_tuned.resume_or_load(resume=False)
trainer_tuned.train()
```

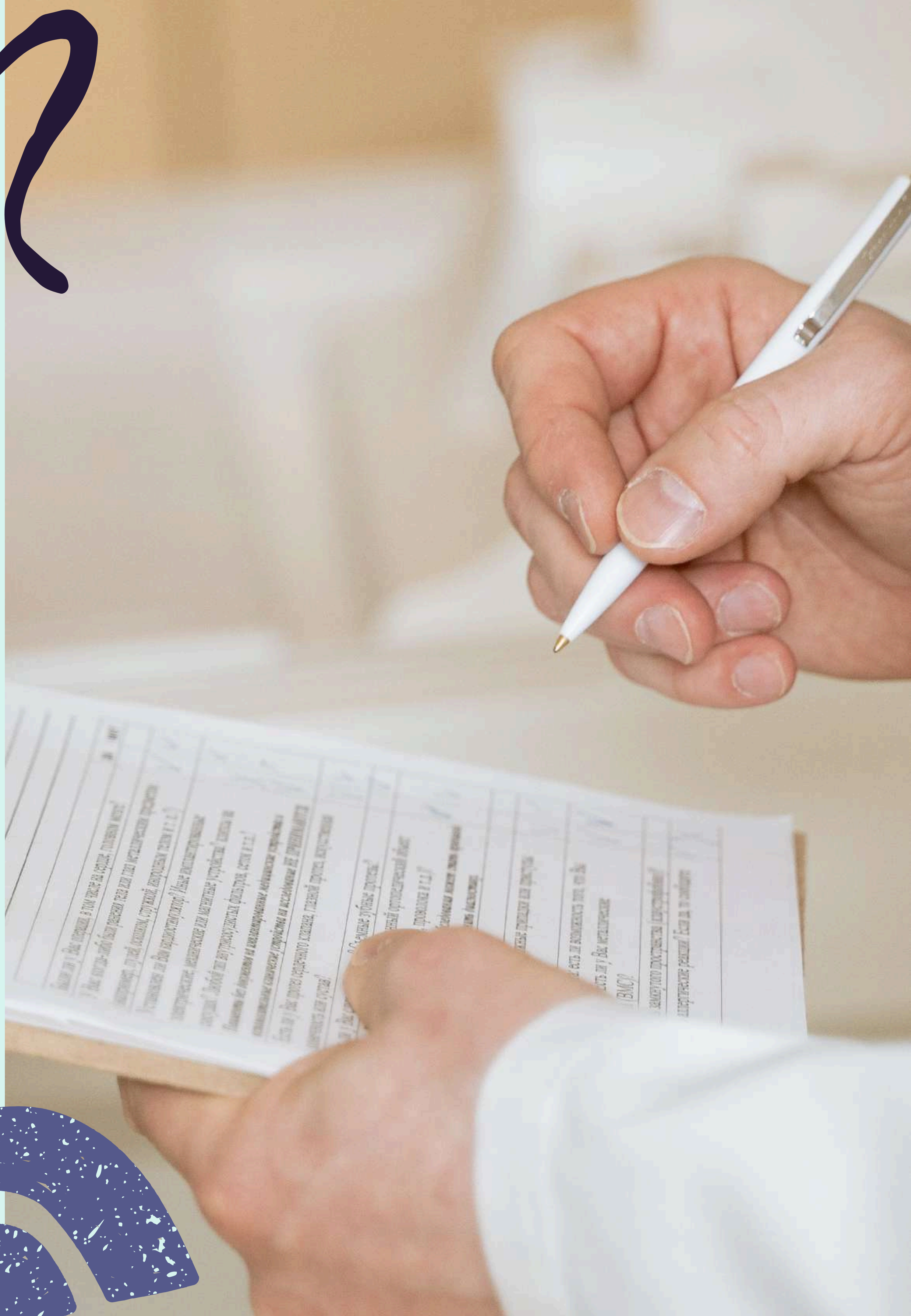




2nd test of Retinanet model

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.499
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.695
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.547
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.210
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.522
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.608
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.664
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.680
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.347
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.698
[05/18 20:36:21 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs  |  APm   |  APl   |
|:------:|:------:|:------:|:-----:|:------:|:------:|
| 49.855 | 69.490 | 54.737 |  nan  | 20.981 | 52.226 |
[05/18 20:36:21 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
[05/18 20:36:21 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP     | category   | AP     | category   | AP     |
|:-----------|:-------|:-----------|:-------|:-----------|:-------|
| leakage    | nan    | crack      | 51.123 | leak       | 65.168 |
| no leak    | 30.524 | water      | 52.604 |            |        |
```

Conclusion and Recommendation

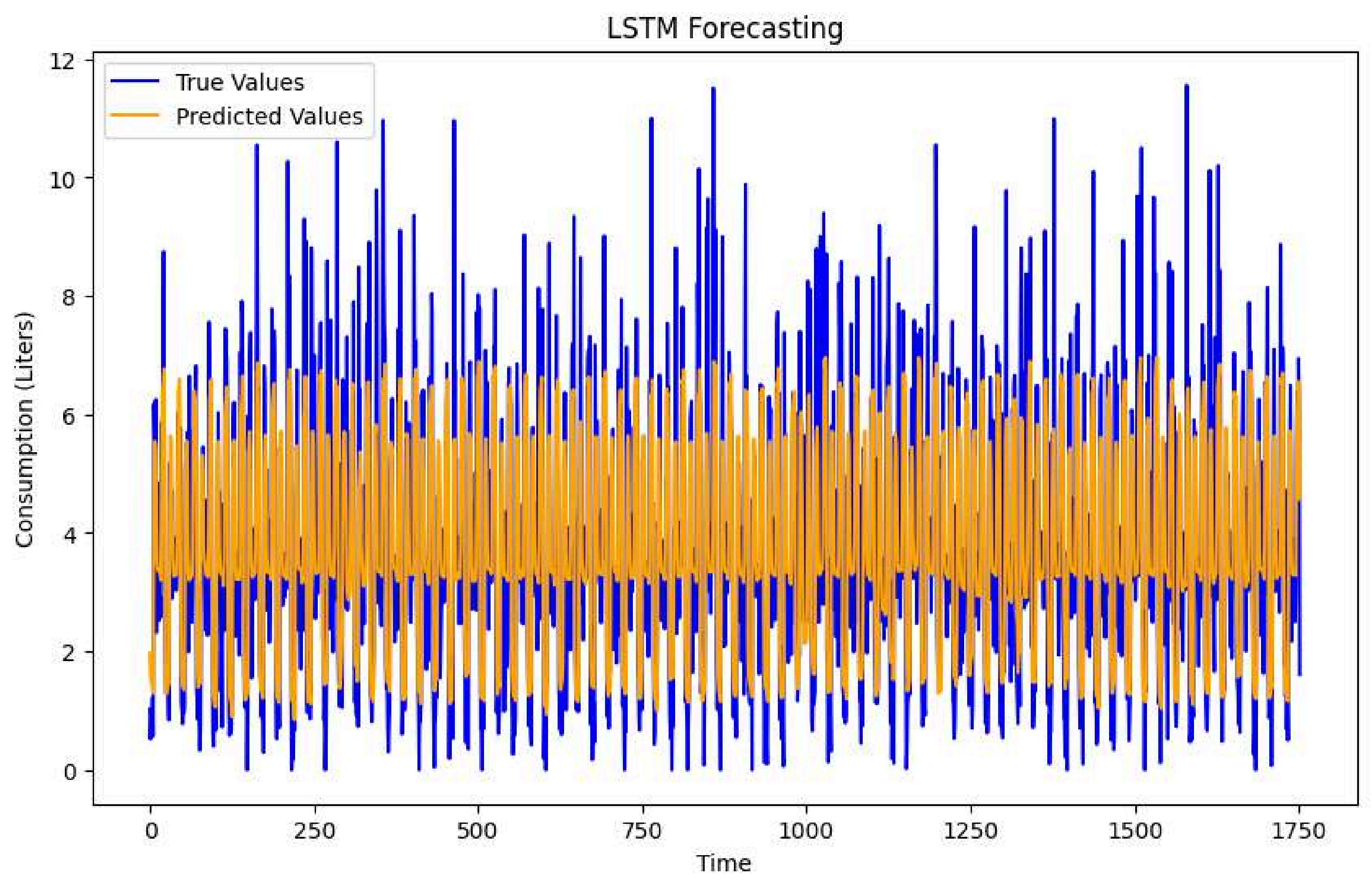


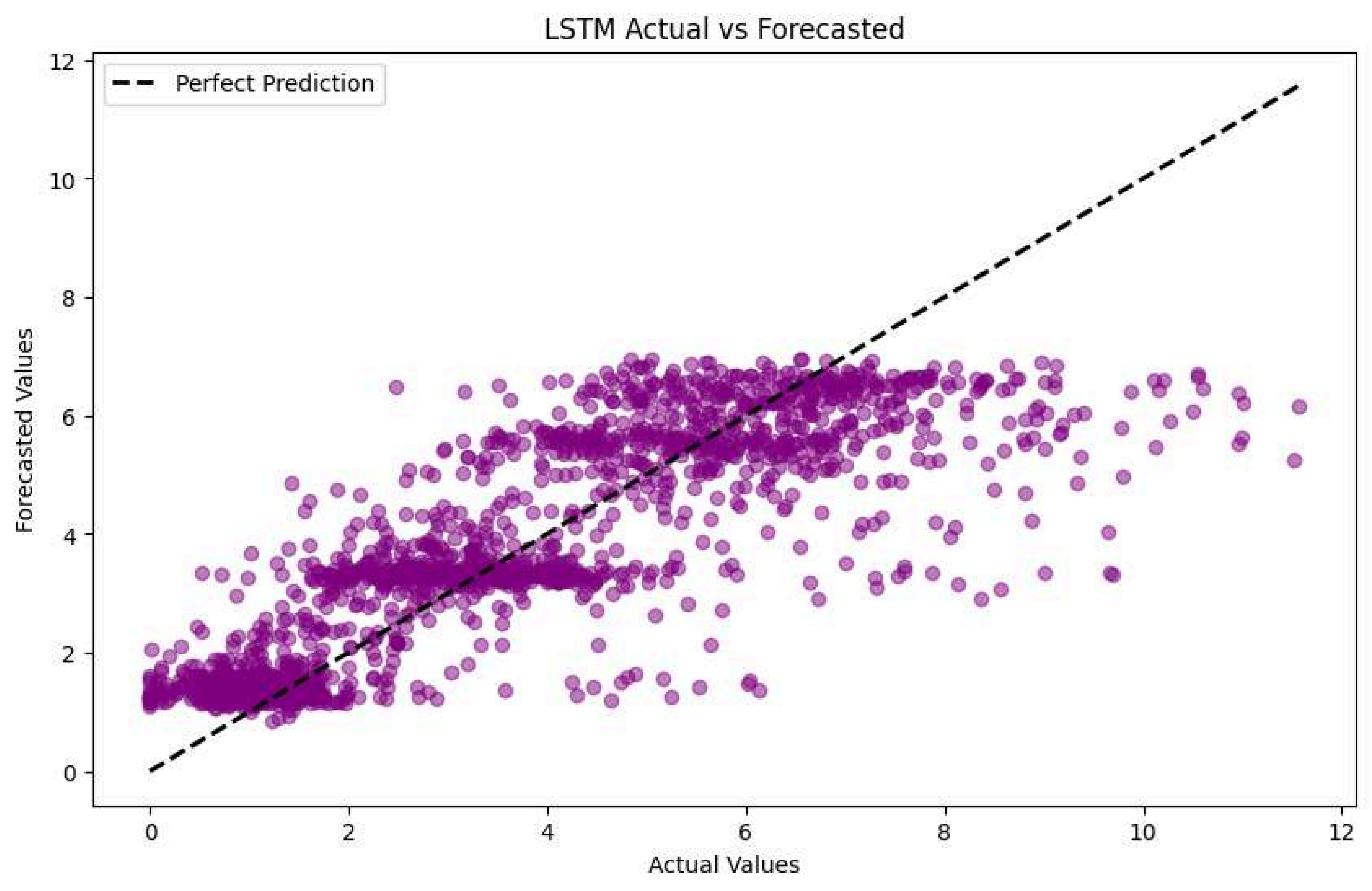
Which is the best model?

- Highest Detection Accuracy – Achieved an mAP50 of 85.9%, outperforming Detectron2 and RetinaNet.
- Fast and Real-Time – Optimized for low-latency inference, ideal for smart water metering systems.
- Strong Recall on Leaks – Detected nearly 90% of actual leaks, ensuring critical issues aren't missed.
- Robust Generalization – Handled diverse image conditions thanks to built-in data augmentations.
- Stable and Efficient Training – Smooth convergence without overfitting, simplifying deployment.

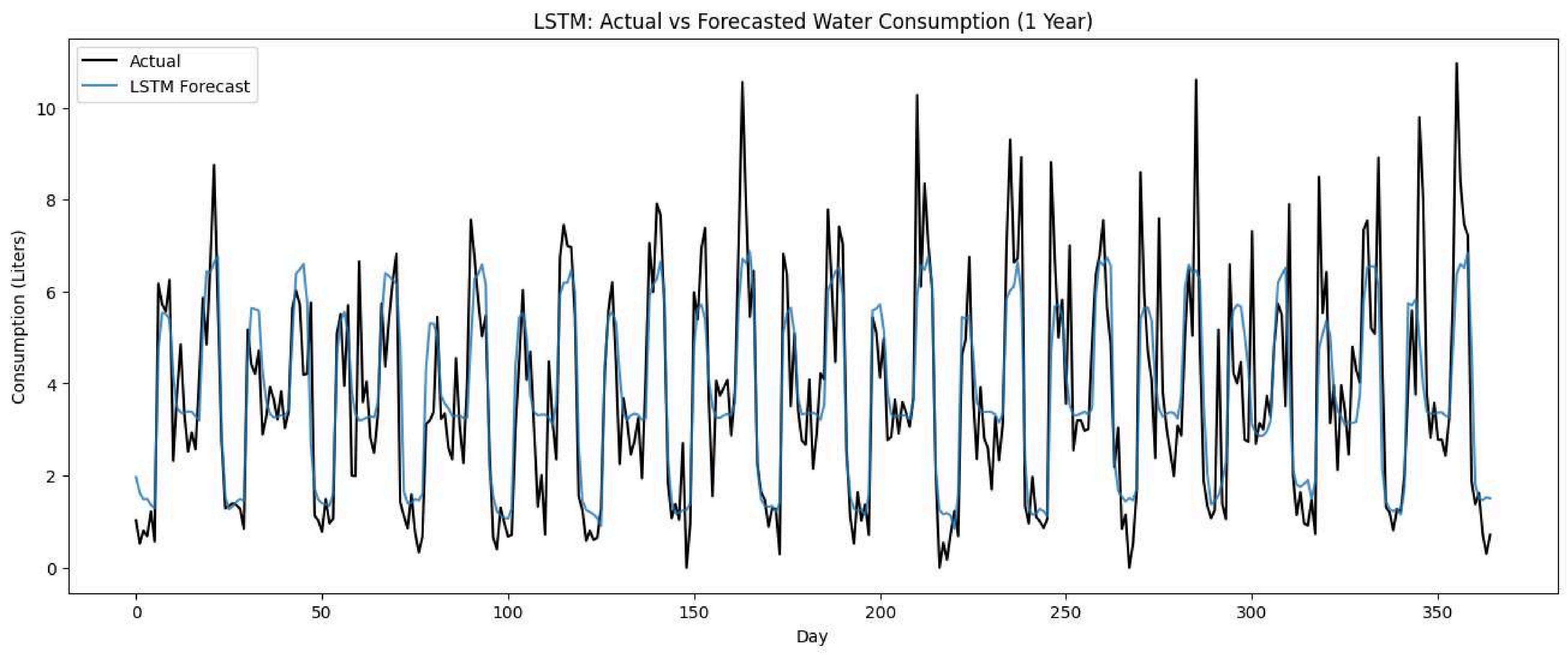
LSTM

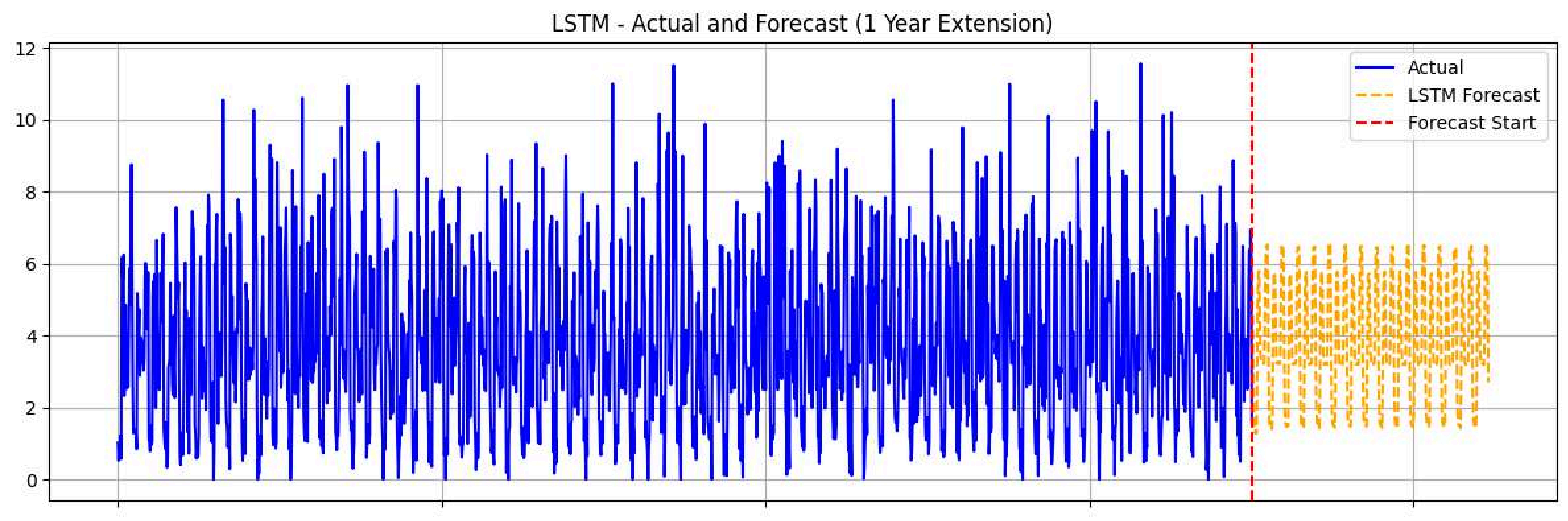
```python
# Optimized LSTM Model with Dropout and Early Stopping
model_lstm = Sequential([
    Input(shape=(window_size, 1)),
    LSTM(64, return_sequences=True),
    Dropout(0.2),
    LSTM(32),
    Dropout(0.2),
    Dense(1)
])
model_lstm.compile(optimizer='adam', loss='mse')
early_stopping = EarlyStopping(patience=7, restore_best_weights=True)
model_lstm.fit(X_train, y_train, validation_split=0.1, epochs=50, batch_size=32,
               callbacks=[early_stopping], verbose=1)
```





LSTM

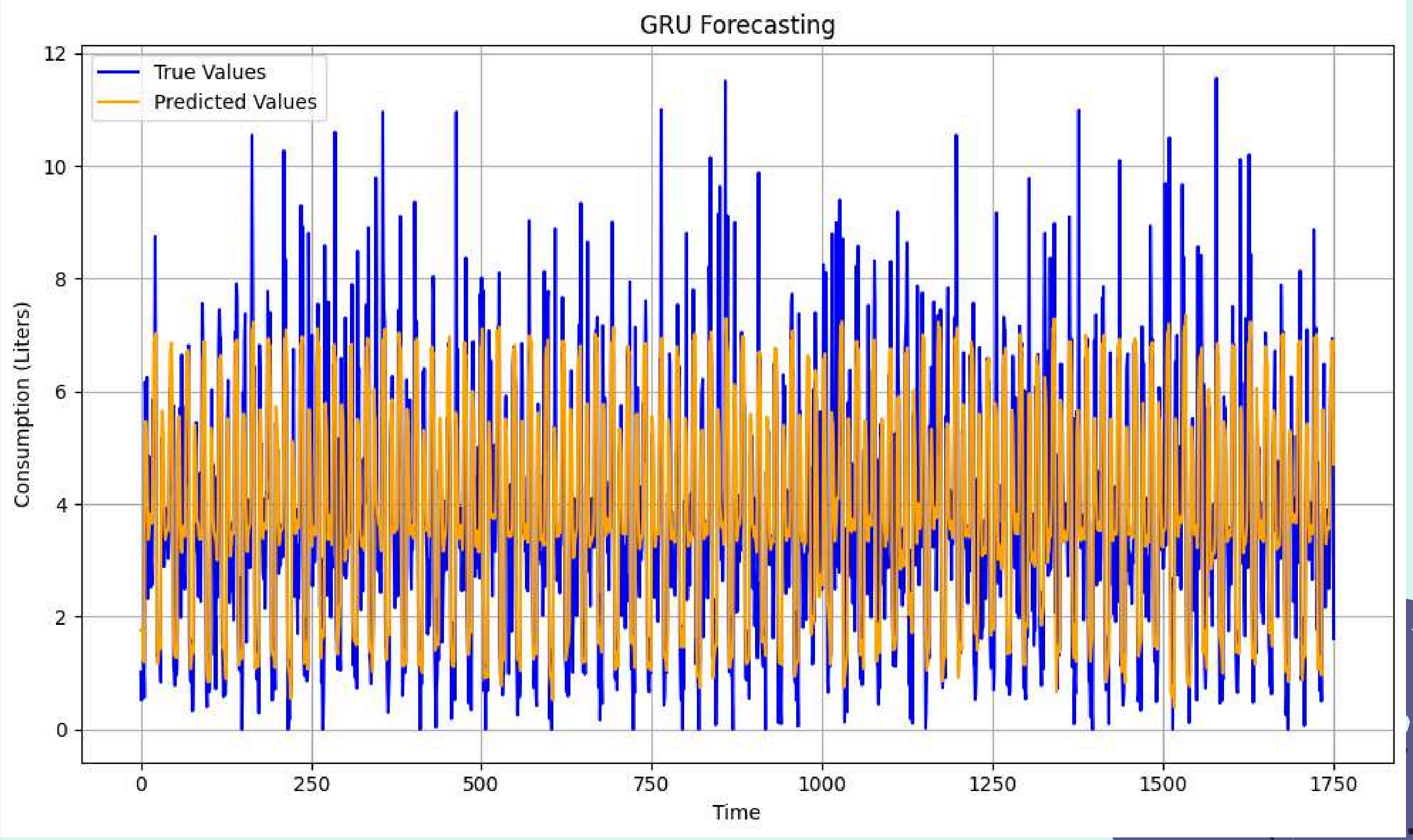


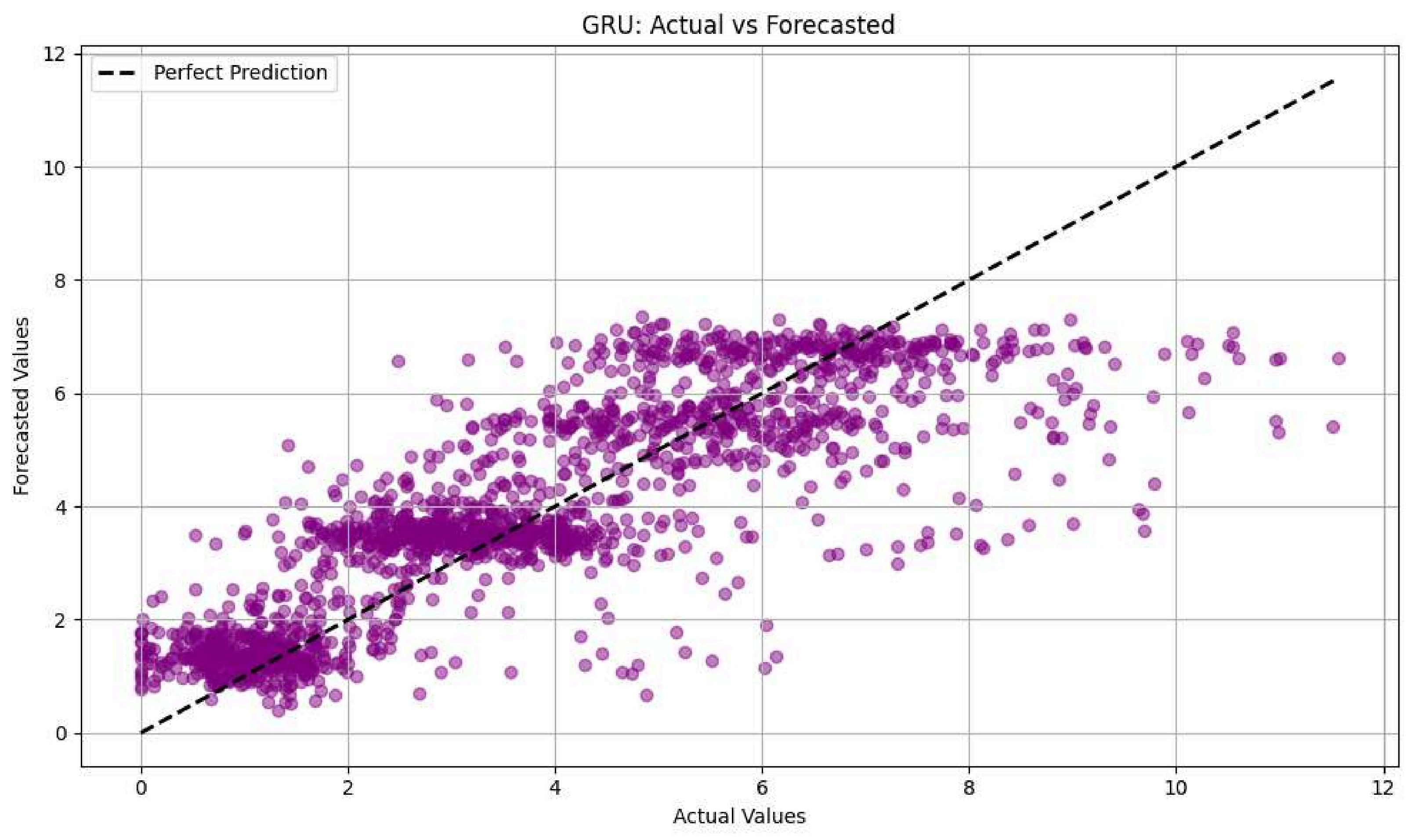


Forecast Behavior: Produces smooth and stable forecasts with clear periodic patterns.
Strengths: Good at capturing long-term dependencies and seasonality.
Limitations: May underrepresent short-term fluctuations or sudden changes in consumption.

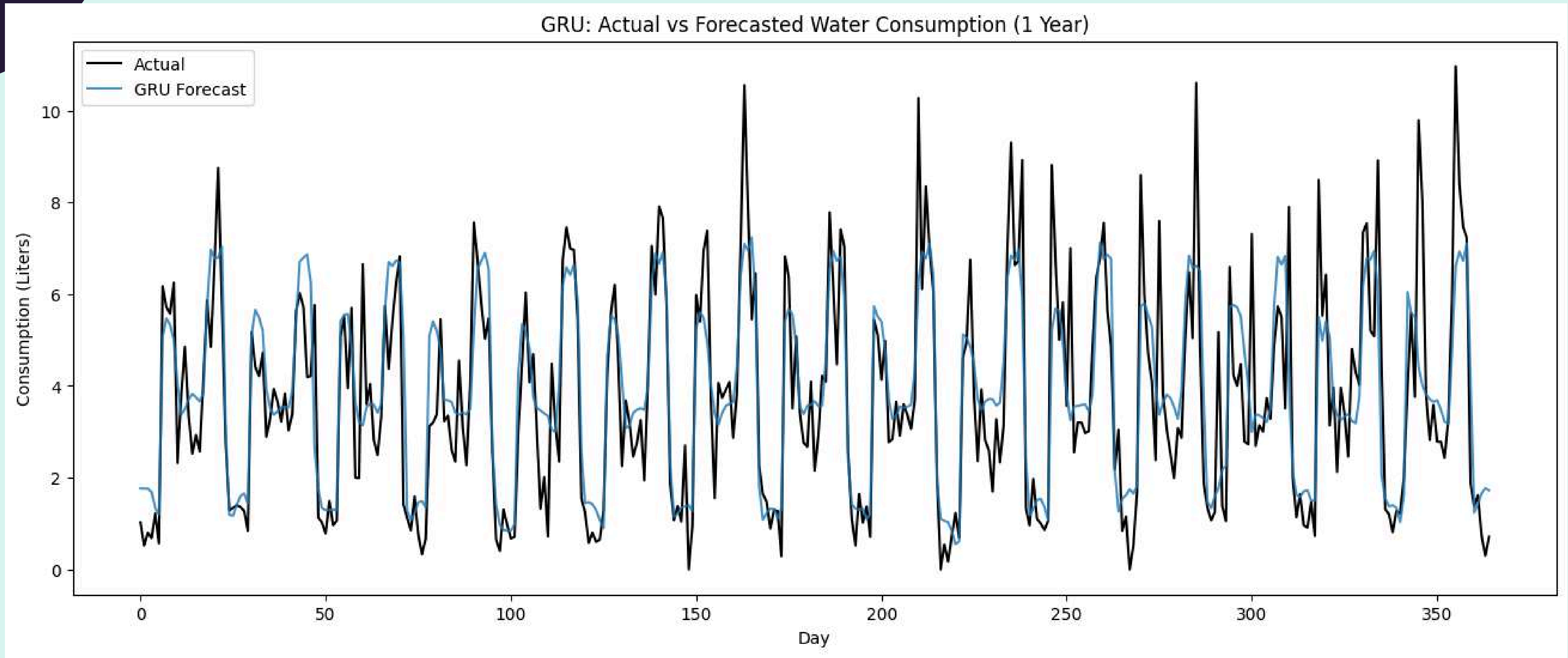
GRU

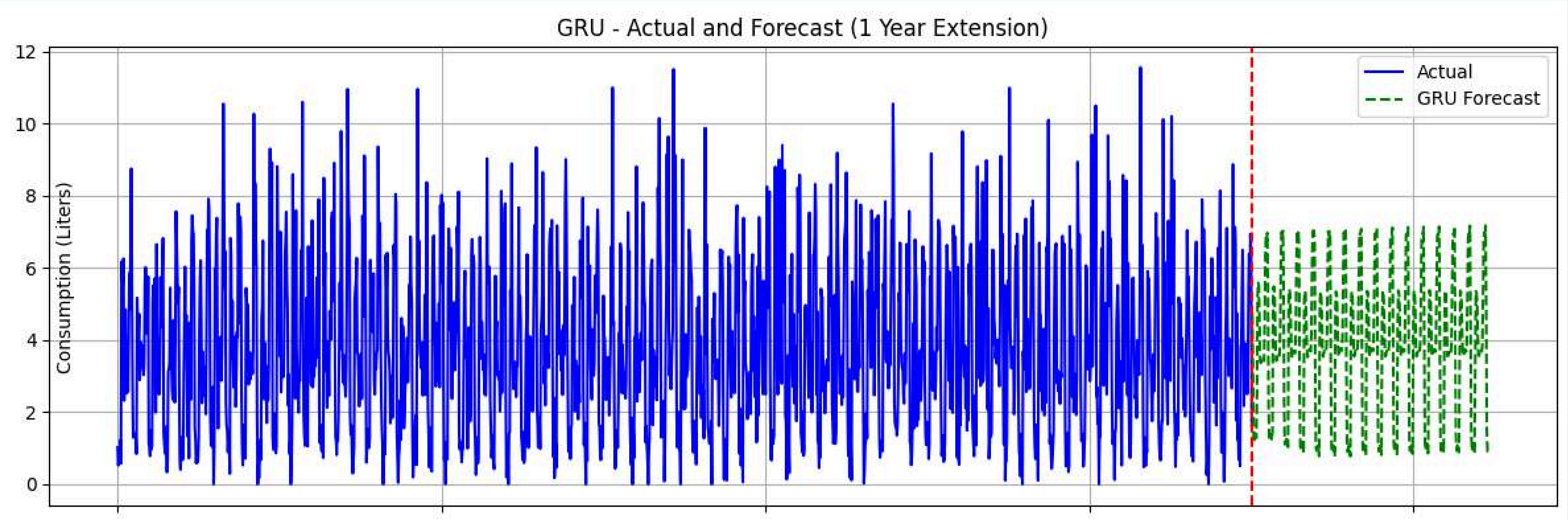
```python
# Optimized GRU Model with Dropout and Early Stopping
model_gru = Sequential([
    Input(shape=(window_size, 1)),
    GRU(64, return_sequences=True),
    Dropout(0.2),
    GRU(32),
    Dropout(0.2),
    Dense(1)
])
model_gru.compile(optimizer='adam', loss='mse')
model_gru.fit(X_train, y_train, validation_split=0.1, epochs=50, batch_size=32,
              callbacks=[early_stopping], verbose=1)
```





GRU

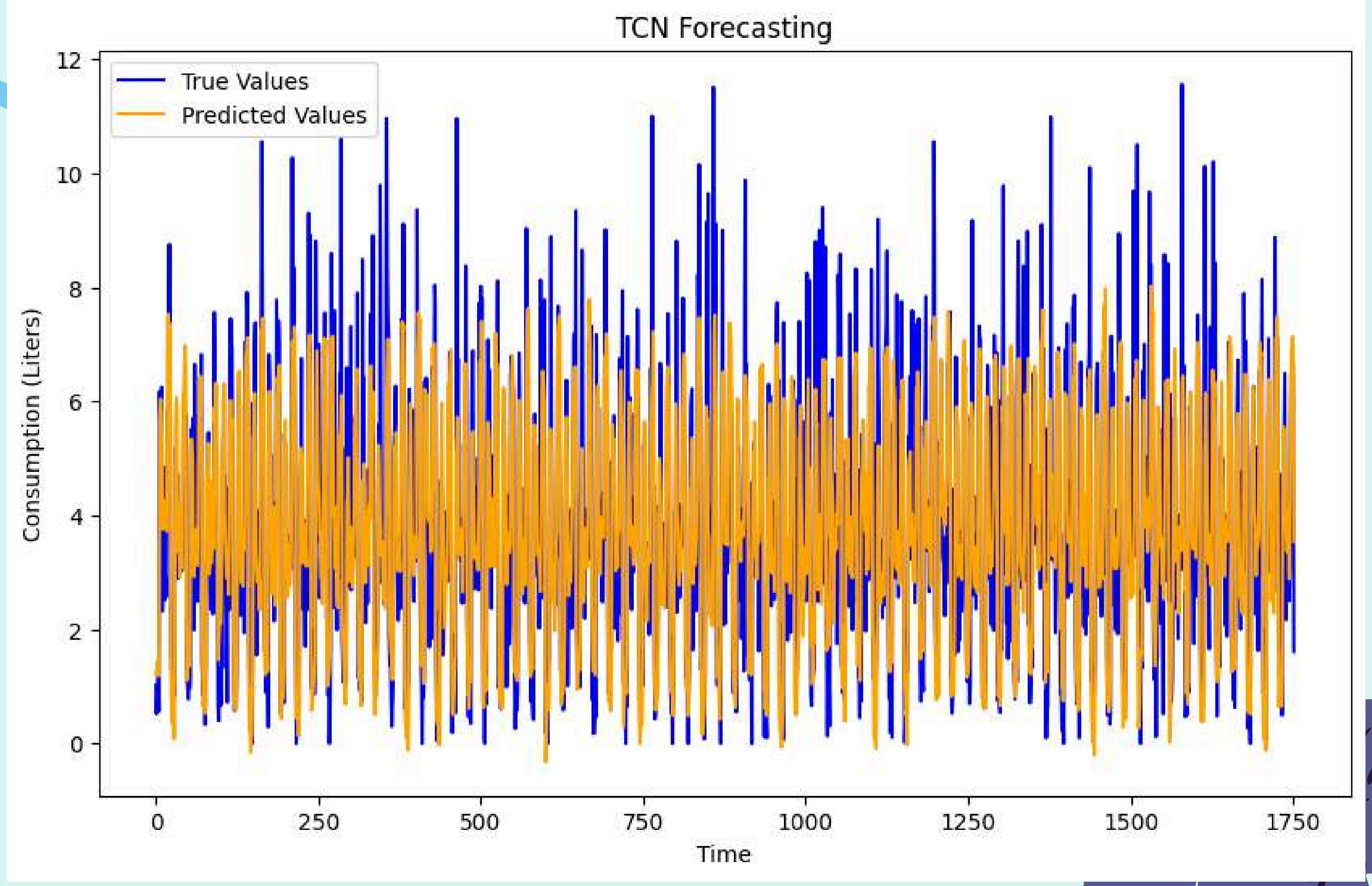


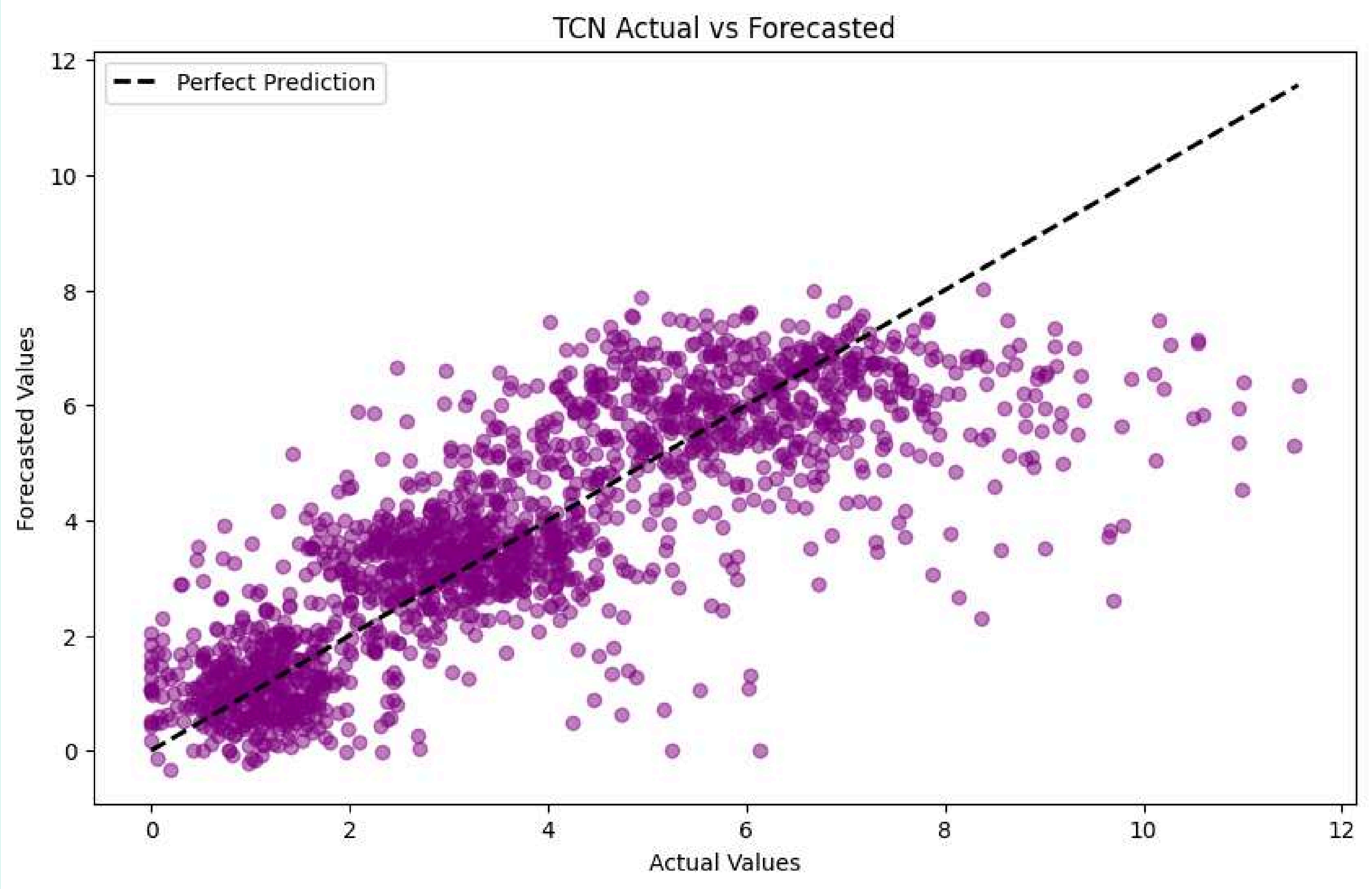


GRU (Gated Recurrent Unit)
Forecast Behavior: Similar to LSTM with stable and consistent trends, slightly more responsive to short-term variations.
Strengths: Efficient in training and performs well with fewer parameters.
Limitations: Like LSTM, may smooth out high-frequency noise or irregularities.

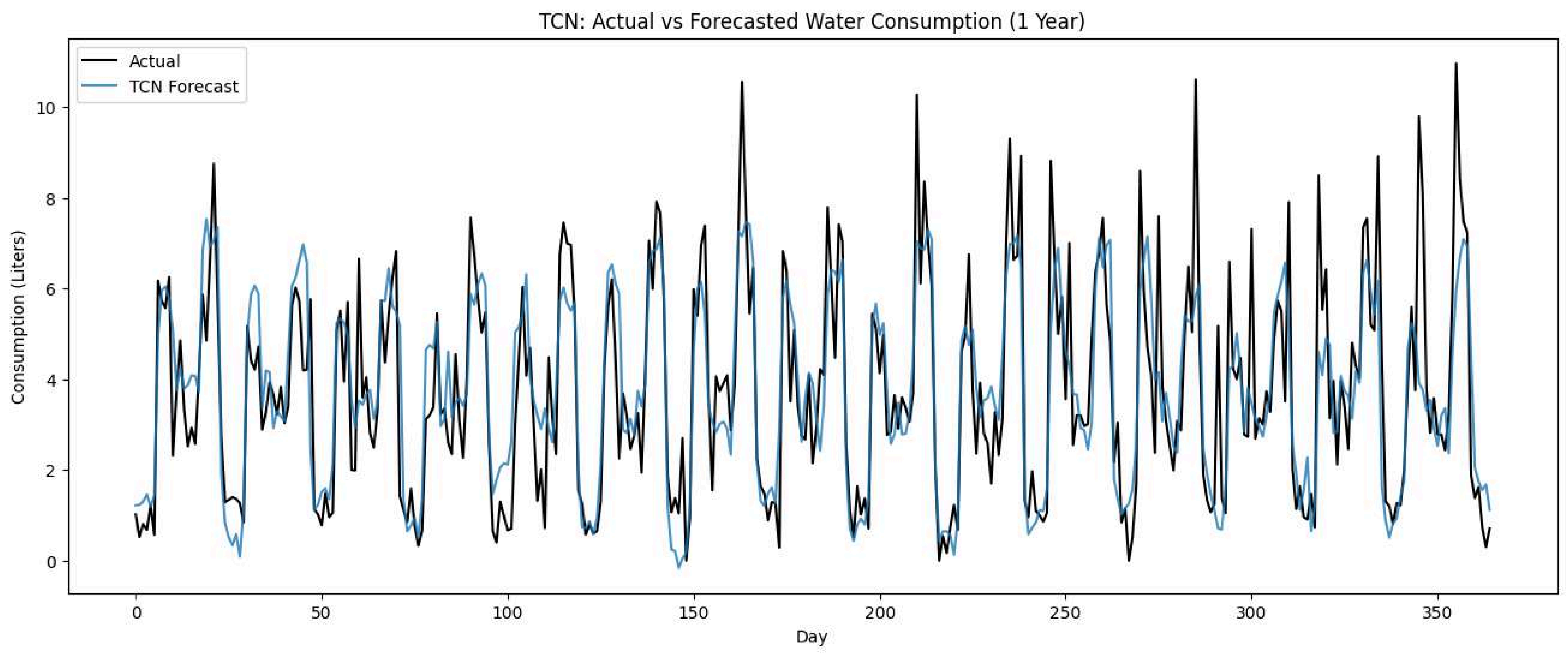
TCN

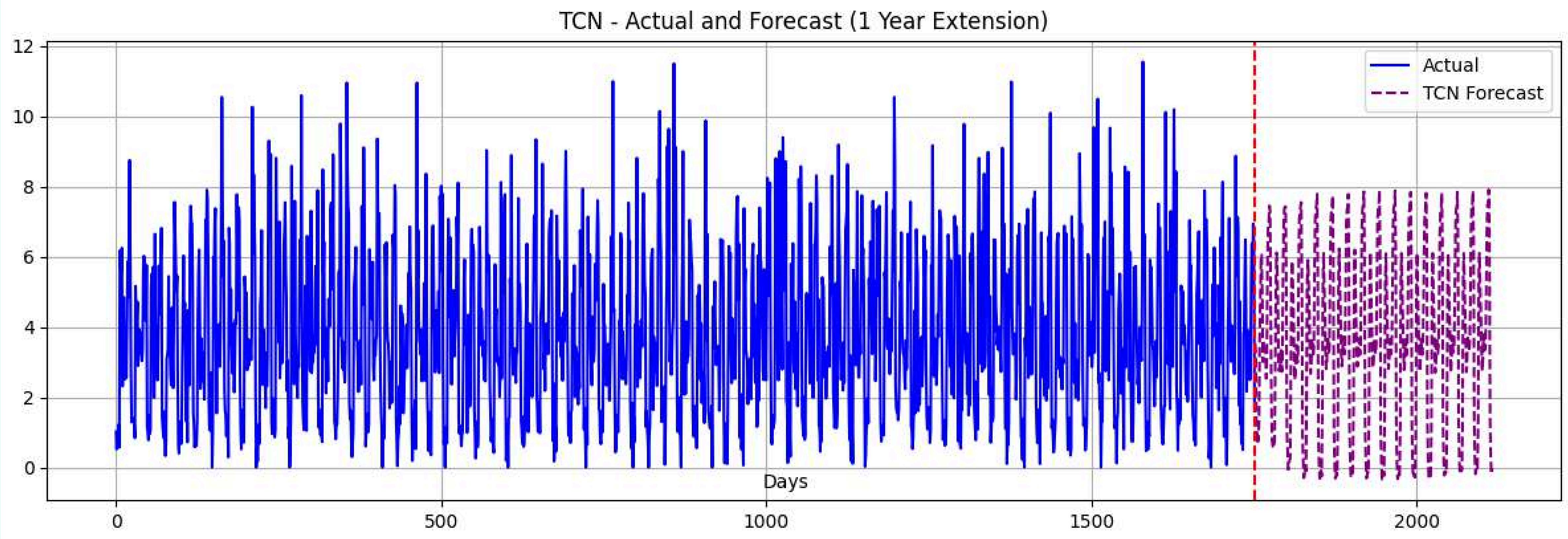
```python
# Optimized TCN Model with Dropout and Early Stopping
def residual_block(x, filters, kernel_size, dilation_rate):
    shortcut = x
    x = Conv1D(filters, kernel_size, dilation_rate=dilation_rate, padding='causal')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Dropout(0.2)(x)
    x = Conv1D(filters, kernel_size, dilation_rate=dilation_rate, padding='causal')(x)
    x = BatchNormalization()(x)
    if shortcut.shape[-1] != x.shape[-1]:
        shortcut = Conv1D(filters, 1, padding='same')(shortcut)
    x = Add()([shortcut, x])
    return Activation('relu')(x)

inputs = Input(shape=(window_size, 1))
x = residual_block(inputs, 32, 3, 1)
x = residual_block(x, 32, 3, 2)
x = residual_block(x, 32, 3, 4)
x = GlobalAveragePooling1D()(x)
outputs = Dense(1)(x)

model_tcn = Model(inputs, outputs)
model_tcn.compile(optimizer='adam', loss='mse')
model_tcn.fit(X_train, y_train, validation_split=0.1, epochs=50, batch_size=32,
              callbacks=[early_stopping], verbose=1)
```





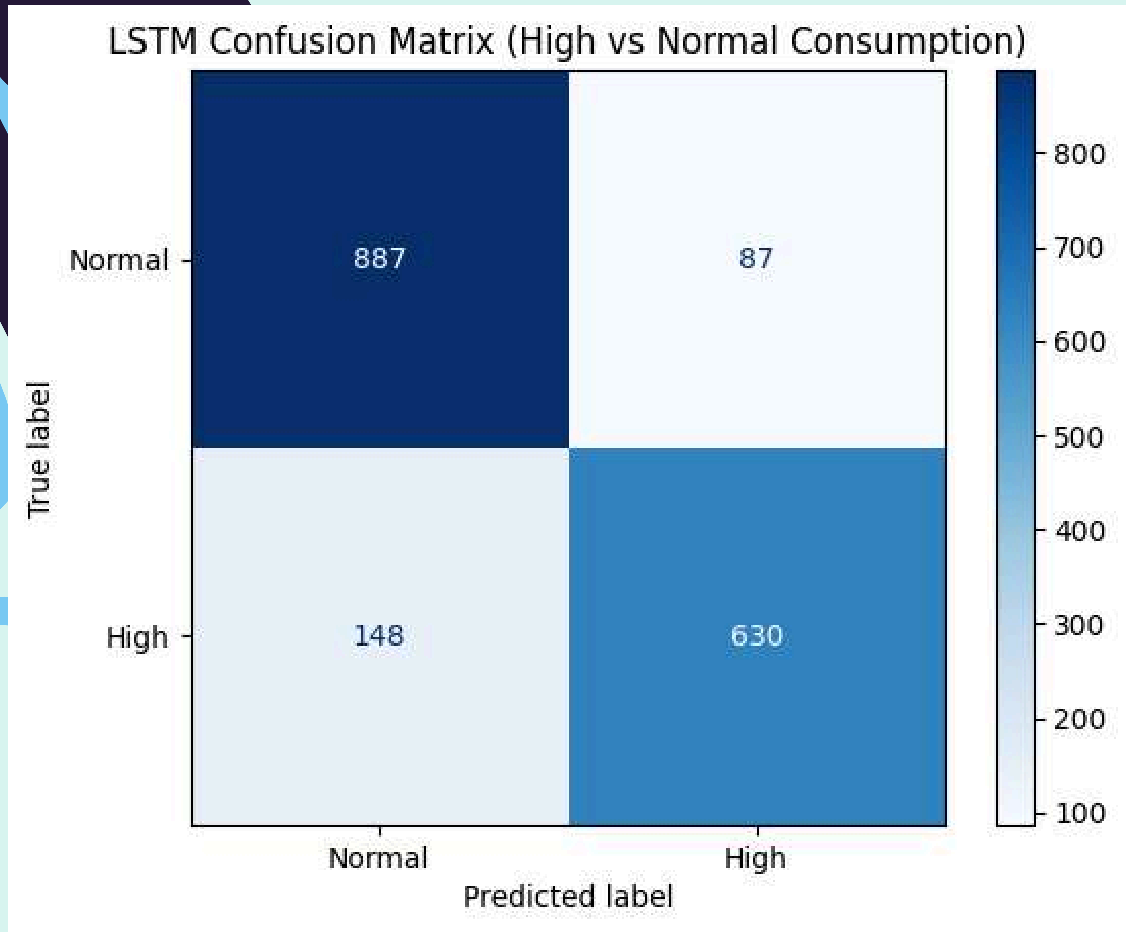
TCN



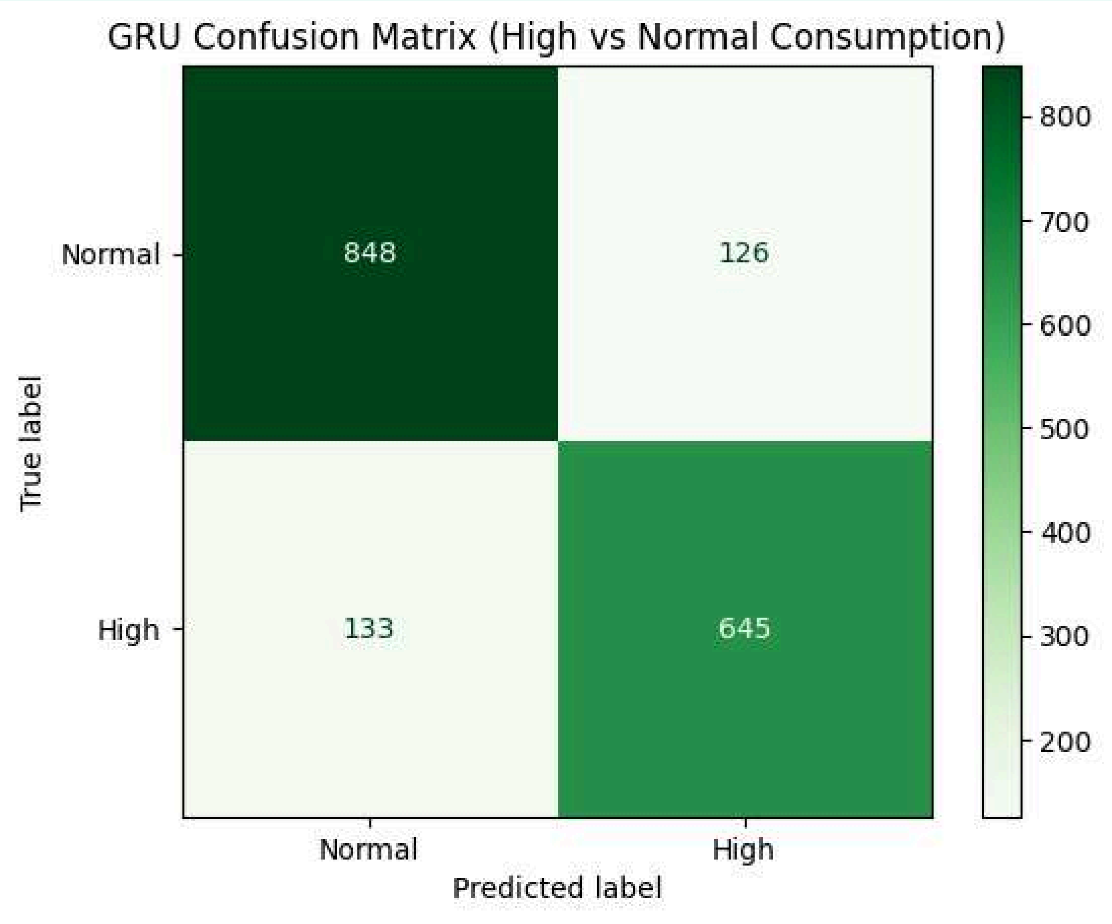


TCN (Temporal Convolutional Network)
Forecast Behavior: Forecast is more detailed with visible high-frequency fluctuations.
Strengths: Better at capturing short-term patterns and local variations in consumption.
Limitations: Slightly more complex and potentially more sensitive to noise.

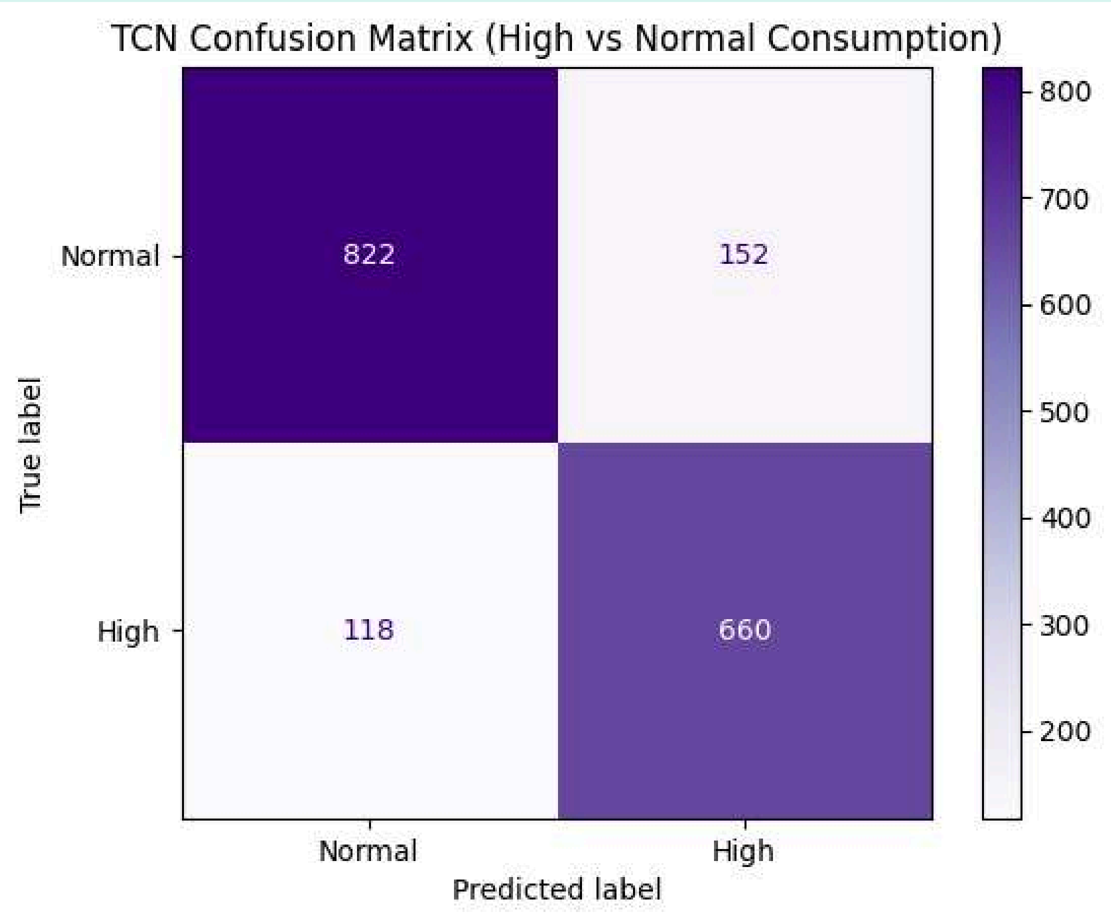
MODELS MATRIX



The LSTM model is better at identifying Normal consumption (887 correct predictions) compared to High consumption (630 correct predictions).
It misclassifies 87 instances of Normal as High and 148 instances of High as Normal.

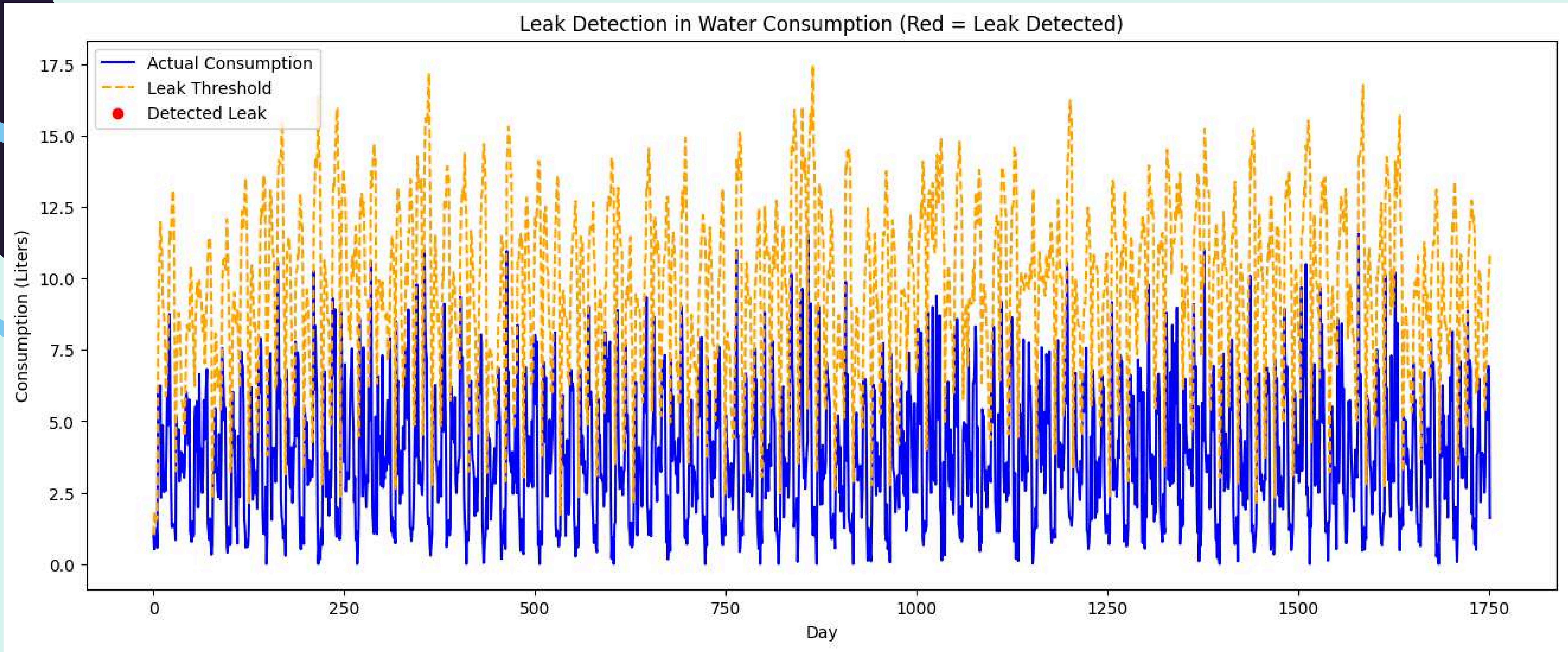


The GRU model shows balanced performance, with slightly fewer correct predictions for Normal (848) than LSTM but better accuracy for High (645).
It has fewer misclassifications for High (133) compared to LSTM, suggesting improved sensitivity for detecting High consumption.



The TCN model excels in identifying High consumption (660 correct predictions, the highest among the three models).
However, it has the lowest accuracy for Normal (822) and the highest False High errors (152), indicating it may be more conservative in labeling consumption as High.

Model	MAE ↓	MSE ↓	RMSE ↓	Performance
LSTM	0.92	1.66	1.29	⭐ Best
GRU	0.94	1.69	1.3	👍 Good
TCN	0.99	1.88	1.37	👎 Worst



Leak Detection in Water Consumption (Red = Leak Detected)
Actual Consumption
Leak Threshold
Detected Leak
Consumption (Liters)
Day
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
250
500
750
1000
1250
1500
1750

Thank you for listening!

EMAIL
qjladornado01@tip.edu.ph
qvabapuyan01@tip.edu.ph