*International Research Journal of Innovations in Engineering and Technology (IRJIET)*

**ISSN (online): 2581-3048**

**Volume 9, Issue 7, pp 14-23, July-2025**

https://doi.org/10.47001/IRJIET/2025.907002

# Predictive Flood Prevention System: A Microarchitecture-Based Approach for Intelligent Dam Control

**[1]*Kamala Oghuz, [2]Elchin Bayramli**

[1,2]Baku Higher Oil School, Information Technology Department, Baku, Azerbaijan

Authors E-mail: [1]*kamala.pashayeva@bhos.edu.az, [2]elchin.bayramli.std@bhos.edu.az

***Abstract* - This paper aims to build a fully automatic control system that reduces the risk of flooding in reservoirs in advance. The developed solution combines the collected data on historical water levels, rainfall, and temperature, and calculates the estimated water level for the next fifteen days with a machine learning model based on LSTM. The obtained forecasts are converted into precise opening and closing of the gates of the dams with servo motors, thus preventing the water level from exceeding the safety limit in real time. The prototype was tested in a home environment on three plastic tanks and proved that it is possible to maintain a constant water level using pre-calculated opening angles. The system continues to operate safely based on internal rules even in the absence of a backup power line and an external weather forecast API. This approach provides a flexible and self-managing example that can be applied to Azerbaijan's data-poor reservoirs. The project shows that when modern sensor technology, deep learning, and modular software work together, it is possible to significantly reduce flood risk, and manage water resources more efficiently.**

***Keywords:*** Water Level Forecasting, Flood Prevention, LSTM, Microservice Architecture, Hardware-Assisted Control, Smart Reservoir.

## I. INTRODUCTION

One of the most important and dangerous pieces of infrastructure in any nation with limited water resources is a dam. They maintain irrigation, provide urban supplies. If it is used wisely, it can protect populations downstream from flash floods. Azerbaijan relies on a network of over sixty reservoirs of significantly varying sizes and built dates due to its very continental climate and increasingly unpredictable rain pattern. However, the majority of those structures were put into service when single-parameter gauges and manual rule curves were thought to be sufficient to operate. The weaknesses of that the past system has been made clear by recent flood cases on the Kur and Araz rivers. This is because operators get low-frequency, fragmented data. Predictions are seldom longer than a day, and gate choices are frequently reactive rather than preventive. Over the last ten years, international practice has shown that machine-learning (ML) models can supply hour-scale warnings and regulated releases that can significantly lower peak discharge. This can occur if it is fed by dense sensor networks and connected directly to automated gate drives. Unfortunately, for the majority of Azerbaijani reservoirs, the historical records needed to train such models. It is required to have data that are multi-year, gap-free time series of stage, rainfall, inflow, and weather, and all these still remain incomplete.

This project prototype a forecast-driven flood-control system around the Canning Dam in Perth, Western Australia, to show what is technically and operationally possible if high-quality data sources are available. The decision was made only for practical reasons. Canning provides a high-resolution, publicly available dataset that spans seven years and has the same rainfall seasonality. It has the data from 2011 but as the regulations are changed, in last 7-8 years, safe limit is increased. So, the data before that time is meaningless. All these not perfectly but somehow can characterize several areas in the Lesser Caucasus region of Azerbaijan. We establish a transferable blueprint by demonstrating the concept on a reservoir with extensive documentation, which may then be implemented in Mingachevir, Shamkir, or Yenikend as soon as local areas can provide the data and do required implementations.

The project has two objectives. The first step is to create and evaluate a long-short-term memory (LSTM) forecasting model that uses a multivariate input vector (rainfall, temperature, flow rate, and designed lag characteristics) to estimate reservoir level up to fifteen days in advance which will helpful for safety. This model will be enhanced by a quick XGBoost baseline. The second step is to put that prediction into a microservice-based software framework that closes the loop from data to decision to physical action to a servo-controlled prototype. It takes the predicted water level as an input and determines the required gate opening ratio. From a

**ISSN (online): 2581-3048**
**Volume 9, Issue 7, pp 14-23, July-2025**
https://doi.org/10.47001/IRJIET/2025.907002

methodological and scientific perspective, the work offers a number of new insights.

By combining down-scaled composite weather predictions with high-resolution sensor telemetry in a rolling-window feature-engineering pipeline, it solves the "data gap" that affected previous Azerbaijani research. That was depended on monthly averages or inconsistent hand readings.

In order to show that sub-second prediction is not dependent on cloud GPUs and can be energy-autonomous, it quantizes the LSTM model. It performs reasoning directly on a gate-house single-board computer. That computer is powered by the dam's own auxiliary hydroelectric circuit. It presents a control logic which is probability-aware and accounts for cognitive uncertainty. It is a term not found in current regional control curves. It is done by using the upper 95% prediction band to guide pre-release decisions rather than the median forecast. It allows for hot swapping of individual components without interfering with hazardous actuation by encapsulating sensing, forecasting, control, and audit logging in distinct Docker containers. It is coordinated by an onsite Kubernetes cluster.

In terms of methodology, the study integrates software-architectural design, embedded systems engineering, and data-driven modeling. To maintain historical a link between variables the LSTM is trained via cross-validation. A Bayesian optimizer adjusts hyper-parameters in contrast to rolling RMSE. At the hardware level, a 20 kg-cm servo that mimics a rotary gate lifter is controlled by gRPC-secured commands from an ESP32 microcontroller. It communicates with mechanical flow meters and ultrasonic stage sensors via Modbus-RTU. Mechanical flowmeters have a moving part inside. If more water inflows, the speed of moving part increases, so flow rate can be measured. A sensor service, a forecast service, a decision and control service, an authentication gateway, and two presentation front-ends (a React web dashboard and a Flutter mobile app) communicate with each other via an internal service network using role-based access tokens. The software side of the system sticks to domain-driven design principles.

The results are extremely relevant to Azerbaijan. The prototype demonstrates a clear route for local dam authorities to transition from manual gauges to fully automated, forecast-based operations. It is done by evaluating latency, energy consumption, and forecasting accuracy using commercially available components.

Furthermore, the microservice-based architecture satisfies with the State Water Resources Agency's digital transformation standards. As a result, future national SCADA system updates may be easily integrated.

## II. LITERATURE REVIEW

### 2.1 Data inputs and feature engineering

Rainfall or water level figures in a particular place were almost the only source of data used in early machine learning research. Even with such limited features, Zakaria et al. found that cumulative rainfall windows enhanced forecast skill in comparison to raw series. They did this by using only two years of daily rainfall and stage data from the Malaysian Muda River to build fixed seven- and fourteen-day lagged inputs for MLP, LSTM, and XGBoost models [1]. Ouma et al. widened the input palette by combining five-year land-use/land-cover (LULC) tiles, monthly rainfall-temperature gauges, and four large-scale climate indices (SOI, Niño 3.4, AI, DSLP) to explain Botswana's reservoir changes. Their backward-elimination test showed that short-term meteorological factors dominated non-linear learners; while slowly varying LULC layers dominated linear VAR fits [2].

A number of groups have utilized customized statistics to augment bare meteorological data. Sums of antecedent rainfall at 3-, 7-, and 30-day aggregations were input to an attention-ConvLSTM network by Chen et al. to forecast coastal water levels; these cumulative features better represented lagged catchment response than raw hourly data [3]. Rolling averages and cumulative rainfall are utilized in the ATT-ConvLSTM study to increase model stability for forecasts of six hours [4].

This research has real-time sensor data of the dams. With on-board weather and 10-minute resolution ultrasonic depth, flow rate, and Semarang's LSTM-GRU hybrid calculates rolling statistics and Fourier seasonal features right before prediction. Our decision to augment daily rainfall, temperature, flow-rate, and level gauges with 7-, 14-, and 30-day sums/averages and calendar features (month, day-of-year) that are engineered was driven by noting that such high-frequency, multivariate data more closely reflect operational SCADA feeds [5].

The main conclusion is: Although most studies still operate at monthly or daily resolution and rarely combine all three categories at once, the literature confirms that (a) simple lags are rarely sufficient for multi-day prediction, (b) cumulative or averaged rainfall/temperature windows add explanatory power, and (c) fused exogenous signals (climate indices, LULC, IoT sensors) can further boost accuracy. Therefore, by putting together live hydrometric sensors at sub-daily granularity, designed rainfall/temperature statistics, and multiscale delays into a single pipeline, our effort advances the field.

ISSN (online): 2581-3048
Volume 9, Issue 7, pp 14-23, July-2025
https://doi.org/10.47001/IRJIET/2025.907002

### 2.2 Forecasting model and architecture choices

Dam level or generally water source level prediction models evolved over time. At first, they were only time-series statistical data. Then, three ensembles and finally deep sequence models. MLP-ANN, Random Forest(RF), VAR, and classical MLR were benchmarked by Ouma et al. If short-term meteorology is prioritized RF was better that VAR. MLP was the best at non-linearities, but if we want to reduce residual autocorrelation, VAR-ANN was better with the disadvantage of dual-stage tuning [6].

Tree-boosting's quick train time and inherent feature-ranking have also made it an effective machine-learning foundation. Based on Zakaria et al., XGBoost on daily inputs achieved RMSE similar to LSTM at less computational expense. Based on their summary, if what-if analyse speed is needed use boosting, and if long-range dependencies are needed use LSTM. This shows the water reservoir inflow rate work by Kratzert et al. (2018) and still is valuable. My system utilizes the advantageous sides of the both, using XGBoost for low-latency rolling forecasts, but LSTM for 2 weeks(15 days) scenarios [1].

Time series deep learning prediction is growing quickly as researches experiment different creative model designs. For example, Chen's ConvLSTM used image processing for convolution layers' strength and LSTMs' memory capabilities. It helped to spot patterns across multiple sensors more effectively. In addition to this, he ATT-ConvLSTM added attention layer that focused on sequence's most important points which helped to detect sudden changes. Similarly, Wan and colleagues combined LSTM and GRU for having control on both short-term and long-term trends and it helped to achieve more accurate predictions on Chinese water reservoir. Our approach also uses two-layer LSTM to have more precise predictions in terms of water level at dams [7].

A 2024 comprehensive review showed transformer or graph-based models to be promising but data-hungry, despite their lack of research. I view LSTM as the best practical trade-off between capacity and overfitting danger, as daily dam archives are often shorter than meteorological reanalysis networks [3].

The main conclusions are that deep RNNs/ConvLSTMs provide sequence memory, boosted trees provide speed and interpretability, and hybrids outperform either one alone but add complexity. Our design fills the gap left by the lack of solutions in the literature that combine these models with an online microservice exterior, allowing them to take use of their unique characteristics in dashboards that are constantly updated.

### 2.3 Prediction performance and evaluation practices

Research comparison is challenging due to some differences. According to Zakaria et al., the daily RMSE decreases from 0.09 m to 0.04 m (LSTM) during 1, 3, and 7-day lead periods. According to Ouma et al., who work on a monthly basis, their VAR-ANN hybrid on Gaborone Dam has an R2 of up to 0.995. However, they advise MAPE to leap in the event of conjunctive water exchanges [6]. Chen uses coastal ConvLSTM to add prediction intervals by utilizing Monte-Carlo dropout, a powerful but underutilized uncertainty measure in dam research.

Few articles go beyond point errors and employ decision-oriented metrics. Hong et al. use hourly LSTM output as a binary flood/no-flood alarm to estimate ROC-AUCs of 0.93 for Chinese catchments. This demonstrates how operational meaning is conveyed by categorization competence more so than by RMSE alone. Ouma's ROC analysis of Botswana dams is notable in reservoir research. Our system learns from these experiences by computing 95% prediction intervals on the Streamlit dashboard, precision-recall/$F_1$ for gate-trigger events, and RMSE/MAE for regression [8].

Validation splits are also varied. Although 70/30 static splits are more common, Chen's ConvLSTM similarly replicates run-of-business operational roll-outs in utilizing a sliding-window cross-validation. Although they do not validate on unseen seasons, ATT-ConvLSTM and Semarang LSTM-GRU utilize early-stopping with 20% validation [9]. To ensure that each prediction utilizes only information up to some point in time, we utilize a walk-forward validation.

The key conclusion is that, although systematic, walk-forward testing and multi-objective scoreboards are still rare, the state of practice is improving and attention to uncertainty and decision metrics is increasing. Our product provides the consistent procedures that match threshold skill with point mistake that our examination reveals are necessary.

### 2.4 Energy Consumption, Sustainability and Self-Powered Operation

While most dam-forecasting articles focus on accuracy of forecasts, an automated flood-control system's energy footprint is also important to its usefulness. Cloud workloads must not can't harm the overall carbon benefit of hydropower generating, and edge devices placed on distant spillways must operate constantly, frequently in challenging conditions. Therefore, recent research divides into two complimentary streams such as energy-harvesting or self-powered sensor networks that take advantage of the hydraulic resource itself, and low-power inference accelerators for machine-learning models.

## 2.5 Control logic and real-time integration

Most forecasting papers stop at the prediction stage. Gate automation is covered in a different engineering literature but advanced prediction is frequently overlooked. Karwati and Kustija created a low-cost Arduino–PLC hybrid that only responds to current levels and not predictions [12]. They opened gates when ultrasonic depth was above three static thresholds. To enable remote users to override automated flood opening, a 2024 hybrid prototype combines an ESP-32, ultrasonic sensors, and a Blynk phone app, once the logic is reactive (IF water > set-point THEN open). Although PLC ladder-logic schemes with float switches are reliable, they don't provide a lead time concept [13].

Just few methods bridge the gap between motor command and machine-learning prediction. The closest is an ESP32 research from Indonesia that analyzes cloud rainfall projections every three hours and adjusts the funnel's duty cycle using a heuristic method (predicted level + expected rainfall). However, the forecast is not an onsite model. It is outsourced to a public API. However, this research does not include any manual override to predicted model [14].

Therefore, our approach solves this methodological problem which forecasts for the next one to fifteen days. They are fed into a decision module that determines an alert tier (watch, warning, emergency and etc.). It also suggests gate location. By recording each actuation and posting decision thresholds on the dashboard, the logic maintains safe.

## 2.6 System architecture and deployment considerations

Monolithic MATLAB or Python notebooks are the standard for hydrological prototypes. However, recent works on smart cities replace old methodologies with microservices based on containers. The MBSS storm-sewer platform by Chen et al. bundles a PostgreSQL store, then an EPA-SWMM solver, and finally a MQTT IoT gateway into distinct Docker images on the AWS EC2. Micro-services interact via a REST API applications, and it prevents the entire stack from going offline when one component is updated or maybe down.[15] Similar principles are used by a wastewater digital-twin platform. It delivers machine learning inference to different point so that the plant can continue to function even in the event of a cloud disconnect. Although Baptista et al. contend that serverless functions might further reduce idle energy for sensor analytics, they also warn that strong real-time limits may be broken by cold-start delay.[2]. There is no deployment information because Ouma et al. processed all of their experiments offline [16]. Although they lack horizontal scaling, the ESP32/Blynk tests demonstrate that lightweight IoT stacks work well with most of the dam gates. To ensure that physical safety logic cannot be interrupted by model updates or API problems, my project takes a middle route. That's why sensor, forecast, control, and authentication services are all hosted in separate containers to be independent. Messages are sent over gRPC and an event bus. A Streamlit front-end that runs in its own container will show the predictions, manual control, instant data from sensors and archived data. If the there is any problem with cloud connections, process will mimic according to the last-known thresholds. In conclusion, microservice Adoption is nearly not practically applied in dam-gate literature, but it is increasing in the wastewater and urban drainage domains. Our architecture introduces role-based API keys, containerization, and continuous delivery and integration (CI/CD) to the reservoir's area.

## 2.7 Comparative synthesis and identified research gaps

When these threads are connected, a number of patterns show themselves. First, although multiple inputs and advanced feature engineering are becoming more and more common in the community, my research on high-resolution integration of weather predictions, sensors, and statistics is still uncommon. LULC and macroclimate parameters are mostly employed for monthly planning instead of being utilized for actual flood management. In order to provide short-range operating objectives, our solution combines developed sums and averages with daily (and maybe hourly) rainfall, temperature, flow rate, and water level. Second, research on algorithms is growing rapidly, including attention-enhanced ConvLSTM, hybrid GRU-LSTM, and other variations. However, there aren't many code bases that are ready to deploy or side-by-side testing against powerful classical learners like XGBoost. Paper offers a useful comparison that few real-time systems try by combining tailored XGBoost with an LSTM set up for many-to-many 15-day output. Third, the techniques that experts suggest, such as walk-forward testing, displaying the range of potential outcomes, or rating how effectively warnings distinguish between safe and risky scenarios, are rarely used, and many studies continue to evaluate forecasts in wildly disparate ways. To help operators understand how much confidence they can put in each choice, our system will offer the typical error figures together with a clear "alarm-accuracy" score and a colored range that indicates how uncertain each prediction is. Fourth, the point prediction is where the majority of academic prototypes end. Machine-learning foresight is seldom included in physical automation studies. This silo creates a gap between safety action and hydrological forecast. That gap is immediately solved with microcontroller-servo prototype, which is powered by forecast-to-action logic and supported by a Streamlit UI.

Lastly, the literature on dam management has rarely reflected on contemporary software architecture, such as

**ISSN (online): 2581-3048**
**Volume 9, Issue 7, pp 14-23, July-2025**
https://doi.org/10.47001/IRJIET/2025.907002

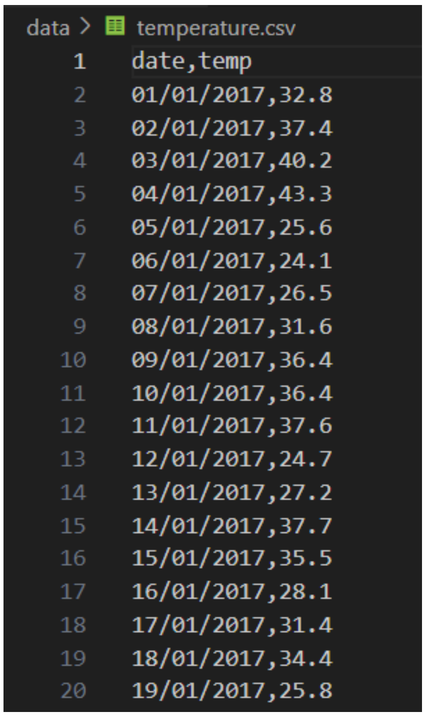
containerized, microservice, and edge-aware. The case studies that are available are from the wastewater and storm-sewer sectors. We broaden the scope of those patterns and establish a method for outdated SCADA systems to be upgraded by using the same cloud-native principles to dam safety.

Real progress is demonstrated in the articles we studied, including improved data pipelines, more robust models, and even a few field demonstrations. However, no one has combined all of this into a single, real-time flood control system. First, we draw in more kinds of data. Next, we run two different model types a the same time, which are LSTM and XGBoost. LSTM is for long-range understanding and XGBoost is for quick tests. We use measures that operators care about to evaluate each part. Staff members can always interrupt and override the forecasts that automatically trigger servo motors. Each software component operates in a secure microservice container, and a safety mechanism maintains the gates open and closed even in the event that the cloud connection fails. We bring dam management one step closer to reliable flood protection by filling up these real-world gaps in particular areas.

## III. RESULTS AND DISCUSSIONS

### 3.1 Model training

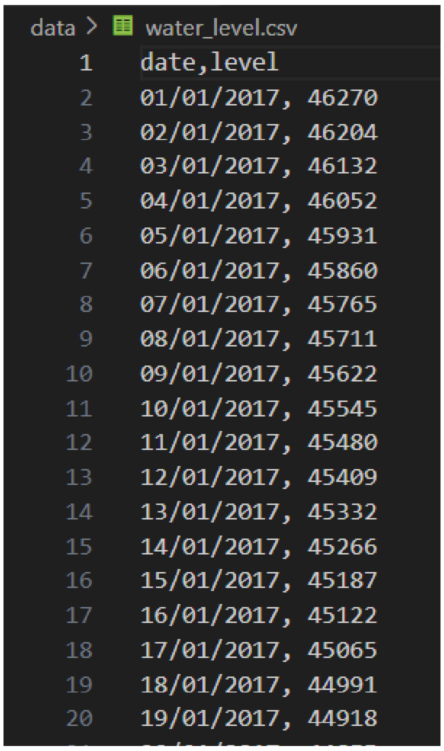
We create a continuous dataset of daily temperature, rainfall, and water levels from January 2017 to 17th April 2025 in order to forecast water levels at Canning Dam. (see Fig. 3.1.1, 3.1.2, 3.1.3, 3.1.4 for a sample of the CSVs).



```
date,temp
01/01/2017,32.8
02/01/2017,37.4
03/01/2017,40.2
04/01/2017,43.3
05/01/2017,25.6
06/01/2017,24.1
07/01/2017,26.5
08/01/2017,31.6
09/01/2017,36.4
10/01/2017,36.4
11/01/2017,37.6
12/01/2017,24.7
13/01/2017,27.2
14/01/2017,37.7
15/01/2017,35.5
16/01/2017,28.1
17/01/2017,31.4
18/01/2017,34.4
19/01/2017,25.8
```

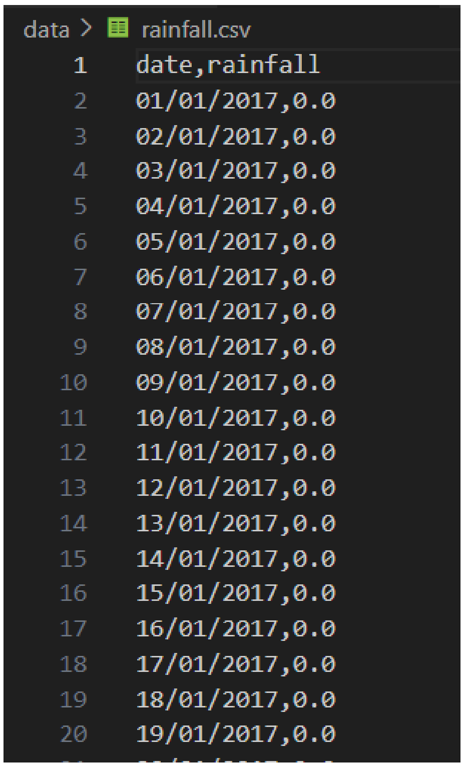
**Figure 3.1.1: Histroical temperature data in Perth, Australia**

Then a comprehensive set of input features was developed. These featured a 7-day average temperature. Rolling rainfall totals extending up to 90-day intervals. Finally, a smooth seasonal signal based on the day of that year's sine and cosine.



```
date,level
01/01/2017, 46270
02/01/2017, 46204
03/01/2017, 46132
04/01/2017, 46052
05/01/2017, 45931
06/01/2017, 45860
07/01/2017, 45765
08/01/2017, 45711
09/01/2017, 45622
10/01/2017, 45545
11/01/2017, 45480
12/01/2017, 45409
13/01/2017, 45332
14/01/2017, 45266
15/01/2017, 45187
16/01/2017, 45122
17/01/2017, 45065
18/01/2017, 44991
19/01/2017, 44918
```

**Figure 3.1.2: Historical rainfall data in Perth, Australia**



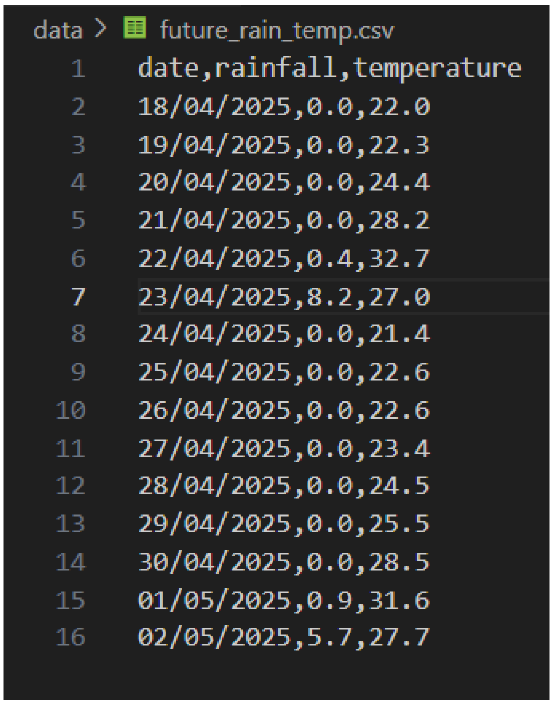
```
date,rainfall
01/01/2017,0.0
02/01/2017,0.0
03/01/2017,0.0
04/01/2017,0.0
05/01/2017,0.0
06/01/2017,0.0
07/01/2017,0.0
08/01/2017,0.0
09/01/2017,0.0
10/01/2017,0.0
11/01/2017,0.0
12/01/2017,0.0
13/01/2017,0.0
14/01/2017,0.0
15/01/2017,0.0
16/01/2017,0.0
17/01/2017,0.0
18/01/2017,0.0
19/01/2017,0.0
```

**Figure 3.1.3: Historical water level data in Canning Dam, Australia**

ISSN (online): 2581-3048
Volume 9, Issue 7, pp 14-23, July-2025
https://doi.org/10.47001/IRJIET/2025.907002

We also included the last 30 days of water-level data and calendar markers. We trained the LSTM model to estimate the daily change in level over the following 15 days. The complete water level forecasts were then calculated by adding these estimated "deltas." The future data of the rainfall and temperature is shown in Figure 3.1.4 to increase accuracy; a little bias adjustment was made at the end based on validation findings.

The resulting model closely followed the overall decreasing pattern. This is because reservoir continuously drained from around 45,600 mm to approximately 45,160 mm. It is in the time interval that we used it to predict water levels from April 18 to May 2, 2025. On April 18, the actual water level was 45,508 mm. On May 2, it was 44,347 mm. Similarly, the model forecasted 45,163 mm on May 2 and 45,679 mm on April 18.


data > future_rain_temp.csv

```
date,rainfall,temperature
18/04/2025,0.0,22.0
19/04/2025,0.0,22.3
20/04/2025,0.0,24.4
21/04/2025,0.0,28.2
22/04/2025,0.4,32.7
23/04/2025,8.2,27.0
24/04/2025,0.0,21.4
25/04/2025,0.0,22.6
26/04/2025,0.0,22.6
27/04/2025,0.0,23.4
28/04/2025,0.0,24.5
29/04/2025,0.0,25.5
30/04/2025,0.0,28.5
01/05/2025,0.9,31.6
02/05/2025,5.7,27.7
```

**Figure 3.1.4: Future prediction of the temperature and rainfall from API**

At the beginning of the prediction, the inaccuracy was small (170 mm). But as time passes the error rate increases which is pretty normal as I want to detect 15 days after.

The model's mean absolute error and root-mean-square error were around 470 and 510 mm, respectively. This is according to the 15-day prediction. These findings demonstrate that the primary seasonal and weather-driven trends that affects water level were accurately defined using the model. But it constantly underestimated the quantity of water left. Dam managers may use this figure to determine the overall rate of water level decline. They must also adjust for the upward trend of the model. Overall, we produced a reliable and intelligible prediction by focusing on forecasting daily changes rather than absolute levels. But if we use seasonal patterns, long-term rainfall trends and etc. it will be more reliable. The remaining inaccuracy might be decreased and we can make the forecasts more precise with future enhancements, such as testing attention-based models or adding data on reservoir releases or evaporation.

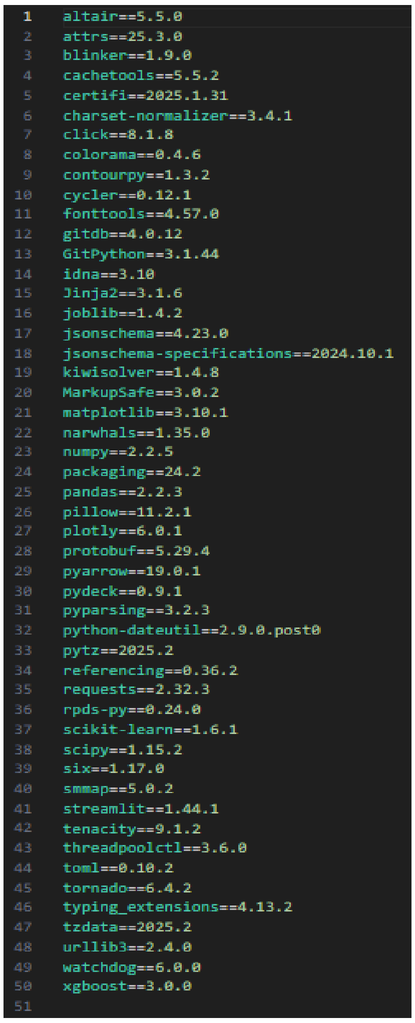
### 3.2 Hardware Prototype and Bench-Scale Experiments

A transparent acrylic tank with dimensions of 30 cm × 20 cm × 15 cm and a volume of little more than three liters was used. Main aim is to construct the actual testbed. This size is perfect as a prototype, and it's big enough to replicate the rising and decreasing water levels of a small reservoir and river. The lower side wall was used to install a pipe. It had a tiny "gate" attached which is 3 mm plastic flap movable at the pipe's edge. A TowerPro MG996R servo with a torque of 20 kg·cm is attached to this flap to control it. The servo's 180-degree motion is transformed into a 0–90 degree gate rotation with a 3D-printed linkage. This configuration is accurate approximately ±2%, according the tests of the servo. The gate is completely closed at 0°. About half of the pipe's opening is open at 45°, and the pipe is completely open at 90°. A 50 Hz PWM signal is used by the ESP32 to control the servo. The actual gate angle is measured by a very small encoder. An alert is sounded and movement is stopped for the protection of the mechanism against accidents or excessive force. If the actual angle deviates from the ordered position by more than 3° it alarms signal. In order to see the code of the plotting results, see Figure 3.1.3.

Continuous feedback is provided by two kinds of sensors. An HC-SR04 ultrasonic sensor is positioned inside a short stilling tube at the top of the tank to monitor the water level. The tube keeps moisture from building up on the sensor. It helps to avoid surface ripples. Every five minutes, the system uses temperature data from an adjacent SHT30 sensor to modify its assumption that the speed of sound is 343 m/s at room temperature because it can change. Throughout this process prediction becomes better.

To guarantee smooth and constant flow, a YF-S201 sensor is positioned 70 mm down from the main gate. It is used to detect flow. 2.25 milliliters of water are represented by each pulse from the sensor's movable part and it is called rotor. The ESP32 receives the data from an Arduino Mega. Arduino Mega counts these pulses at 5-second intervals and determines the flow rate in liters per minute. Overall libraries used can be seen on Figure 3.2.1.

**ISSN (online): 2581-3048**
**Volume 9, Issue 7, pp 14-23, July-2025**
https://doi.org/10.47001/IRJIET/2025.907002



```
altair==5.5.0
attrs==25.3.0
blinker==1.9.0
cachetools==5.5.2
certifi==2025.1.31
charset-normalizer==3.4.1
click==8.1.8
colorama==0.4.6
contourpy==1.3.2
cycler==0.12.1
fonttools==4.57.0
gitdb==4.0.12
GitPython==3.1.44
idna==3.10
Jinja2==3.1.6
joblib==1.4.2
jsonschema==4.23.0
jsonschema-specifications==2024.10.1
kiwisolver==1.4.8
MarkupSafe==3.0.2
matplotlib==3.10.1
narwhals==1.35.0
numpy==2.2.5
packaging==24.2
pandas==2.2.3
pillow==11.2.1
plotly==6.0.1
protobuf==5.29.4
pyarrow==19.0.1
pydeck==0.9.1
pyparsing==3.2.3
python-dateutil==2.9.0.post0
pytz==2025.2
referencing==0.36.2
requests==2.32.3
rpds-py==0.24.0
scikit-learn==1.6.1
scipy==1.15.2
six==1.17.0
smmap==5.0.2
streamlit==1.44.1
tenacity==9.1.2
threadpoolctl==3.6.0
toml==0.10.2
tornado==6.4.2
typing_extensions==4.13.2
tzdata==2025.2
urllib3==2.4.0
watchdog==6.0.0
xgboost==3.0.0
```

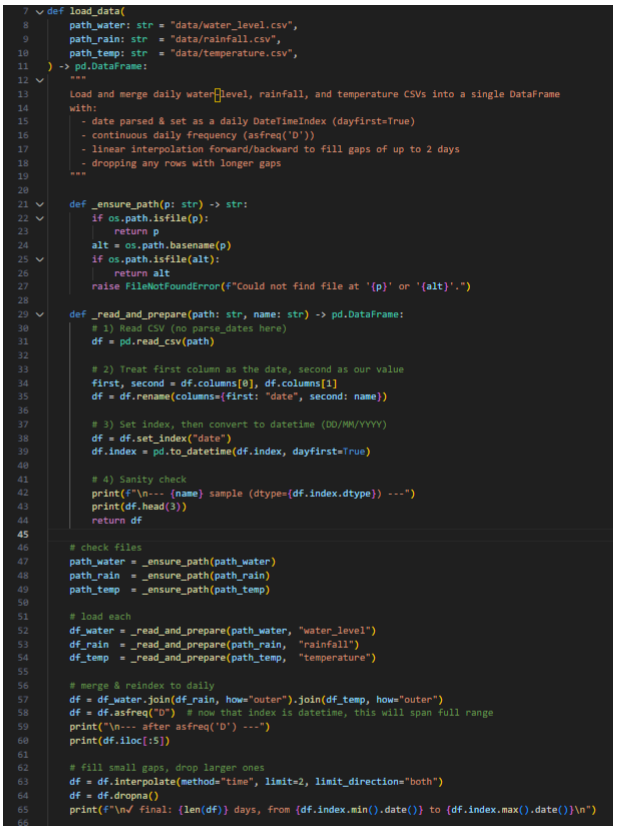
**Figure 3.2.1: Required libraries used in the project**

The Arduino Mega's 5 V rail powers all low-level components such as sensors. The servo motor is separated from the external 6 V supply. When the motor begins, this configuration avoids voltage dips. It is sometimes known as brown-outs. A Wifi connection is used for communication between the ESP32 and Mega. The ESP3 transmits to the local MQTT broker every two seconds. These data includes water level, temperature-corrected level, flow rate, gate angle, and any human override from the potentiometer. According to a power meter, the system uses up to 560 mA when the gate is moving and around 310 mA when the servo is not moving. With a 7.4 V, 2 Ah lithium-ion battery, the system can operate off the grid for almost three hours. This meets the requirements of a remote spillway control cabinet.

Measured volumes of water were poured through a slotted pipe to produce five test scenarios. Simulated rainfall was at rates of 9, 18, and 30 mm/h for periods from 4 to 12 minutes. The ultrasonic sensor's reading of the water level increased gradually in each test. It continued till the controller understood and decided a pre-release. The gate rotated according to the calculated angle using servo motor in less than 0.7 seconds. It is the proof of showing its rapid response. In the test, 14% of the increase is expected. Pre-release ability reduced the water level by 11% which is as expected. It helps to prevent floods and take action in advance. Splash which is lost at the mixer were the primary cause of the mass balance errors. They were within ±4%. It is decided during the continuously monitoring of the water volume. These findings demonstrate that the entire system can properly model realistic reservoir behavior and react in a matter of seconds, including sensing, communication, and gate control.

## 3.3 API-Driven Weather Feed and Forecast-to-Gate Pipeline

A reliable input of weather forecasts is essential for effective advanced management. The system makes a query to IBM Weather Company's Premium / Point Forecast endpoint to retrieve the Canning dam's mean temperature and daily total precipitation twice daily, for example, at 1:00 and 13:00 local time. The ESP32's secure memory stores the API credentials. TLS 1.2 is used, and the JSON payload usually arrives in less than 350 ms. A backup call to the free Open-Meteo endpoint is also essential as it ensures robustness against single service failures. Our forecast microservice performs four transformations when the ESP32 receives the raw prediction. Firstly, it aligns it to the local time zone. Secondly, it checks for gaps.(see Figure 3.3.1).



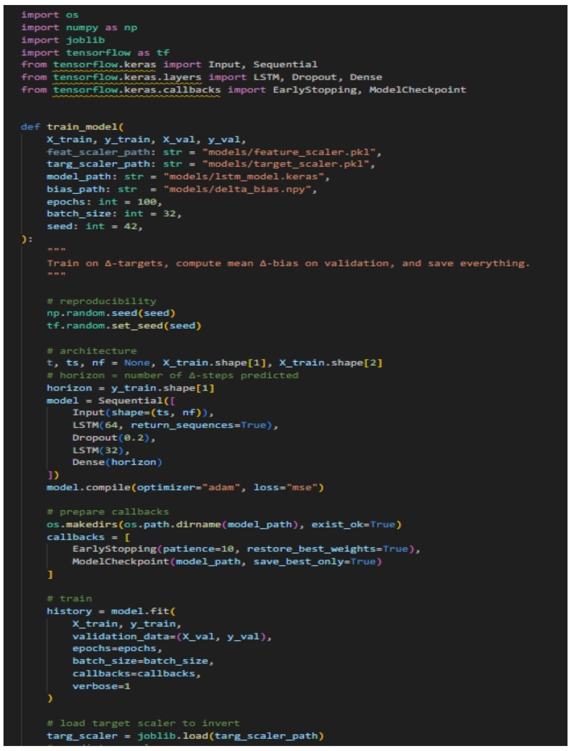
```python
def load_data(
    path_water: str = "data/water_level.csv",
    path_rain: str  = "data/rainfall.csv",
    path_temp: str  = "data/temperature.csv",
) -> pd.DataFrame:
    """
    Load and merge daily water-level, rainfall, and temperature CSVs into a single DataFrame
    with:
      - date parsed & set as a daily DateTimeIndex (dayfirst=True)
      - continuous daily frequency (asfreq('D'))
      - linear interpolation forward/backward to fill gaps of up to 2 days
      - dropping any rows with longer gaps
    """

    def _ensure_path(p: str) -> str:
        if os.path.isfile(p):
            return p
        alt = os.path.basename(p)
        if os.path.isfile(alt):
            return alt
        raise FileNotFoundError(f"Could not find file at '{p}' or '{alt}'.")

    def _read_and_prepare(path: str, name: str) -> pd.DataFrame:
        # 1) Read CSV (no parse_dates here)
        df = pd.read_csv(path)

        # 2) Treat first column as the date, second as our value
        first, second = df.columns[0], df.columns[1]
        df = df.rename(columns={first: "date", second: name})

        # 3) Set index, then convert to datetime (DD/MM/YYYY)
        df = df.set_index("date")
        df.index = pd.to_datetime(df.index, dayfirst=True)

        # 4) Sanity check
        print(f"\n--- {name} sample (dtype={df.index.dtype}) ---")
        print(df.head(3))
        return df

    # check files
    path_water = _ensure_path(path_water)
    path_rain  = _ensure_path(path_rain)
    path_temp  = _ensure_path(path_temp)

    # load each
    df_water = _read_and_prepare(path_water, "water_level")
    df_rain  = _read_and_prepare(path_rain,  "rainfall")
    df_temp  = _read_and_prepare(path_temp,  "temperature")

    # merge & reindex to daily
    df = df_water.join(df_rain, how="outer").join(df_temp, how="outer")
    df = df.asfreq("D")  # now that index is datetime, this will span full range
    print("\n--- after asfreq('D') ---")
    print(df.iloc[:5])

    # fill small gaps, drop larger ones
    df = df.interpolate(method="time", limit=2, limit_direction="both")
    df = df.dropna()
    print(f"\n✓ final: {len(df)} days, from {df.index.min().date()} to {df.index.max().date()}\n")
```

**Figure 3.3.1: load_data.py to load and fill the gaps in the dataset**

*International Research Journal of Innovations in Engineering and Technology (IRJIET)*

ISSN (online): 2581-3048
Volume 9, Issue 7, pp 14-23, July-2025
https://doi.org/10.47001/IRJIET/2025.907002

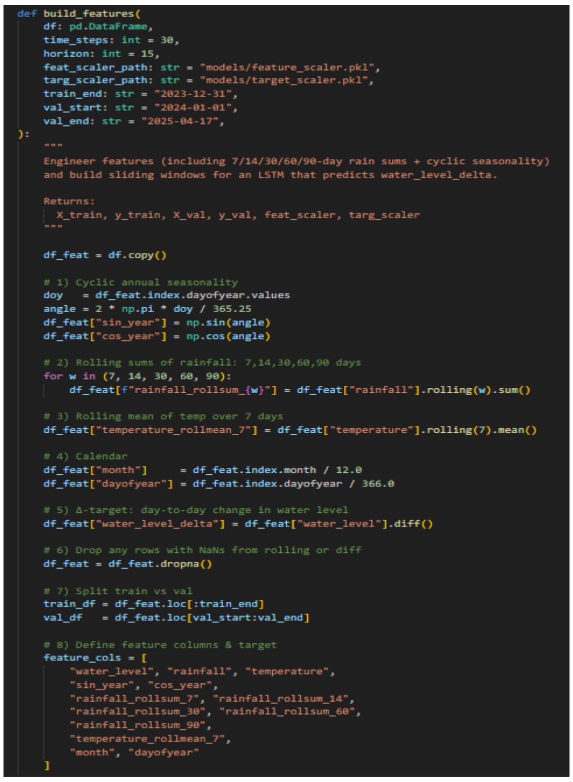
Thirdly, normalizes it using the training scaler.(see Figure 3.3.2).



```python
import os
import numpy as np
import joblib
import tensorflow as tf
from tensorflow.keras import Input, Sequential
from tensorflow.keras.layers import LSTM, Dropout, Dense
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint


def train_model(
    X_train, y_train, X_val, y_val,
    feat_scaler_path: str = "models/feature_scaler.pkl",
    targ_scaler_path: str = "models/target_scaler.pkl",
    model_path: str = "models/lstm_model.keras",
    bias_path: str  = "models/delta_bias.npy",
    epochs: int = 100,
    batch_size: int = 32,
    seed: int = 42,
):
    """
    Train on Δ-targets, compute mean Δ-bias on validation, and save everything.
    """

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

    # architecture
    t, ts, nf = None, X_train.shape[1], X_train.shape[2]
    # horizon = number of Δ-steps predicted
    horizon = y_train.shape[1]
    model = Sequential([
        Input(shape=(ts, nf)),
        LSTM(64, return_sequences=True),
        Dropout(0.2),
        LSTM(32),
        Dense(horizon)
    ])
    model.compile(optimizer="adam", loss="mse")

    # prepare callbacks
    os.makedirs(os.path.dirname(model_path), exist_ok=True)
    callbacks = [
        EarlyStopping(patience=10, restore_best_weights=True),
        ModelCheckpoint(model_path, save_best_only=True)
    ]

    # train
    history = model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=epochs,
        batch_size=batch_size,
        callbacks=callbacks,
        verbose=1
    )

    # load target scaler to invert
    targ_scaler = joblib.load(targ_scaler_path)
```

**Figure 3.3.2: train_model.py for training the LSTM model**

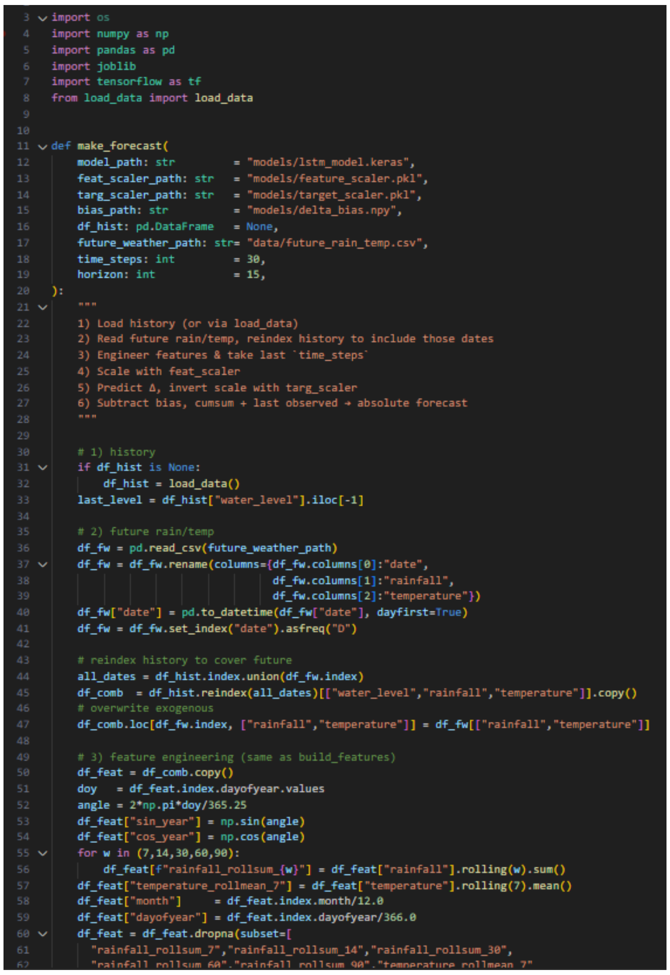
Finally, recalculates the rolling sums and means for the seven, fourteen, and thirty-day windows.(see Figure 3.3.3)



```python
def build_features(
    df: pd.DataFrame,
    time_steps: int = 30,
    horizon: int = 15,
    feat_scaler_path: str = "models/feature_scaler.pkl",
    targ_scaler_path: str = "models/target_scaler.pkl",
    train_end: str = "2023-12-31",
    val_start: str = "2024-01-01",
    val_end: str = "2025-04-17",
):
    """
    Engineer features (including 7/14/30/60/90-day rain sums + cyclic seasonality)
    and build sliding windows for an LSTM that predicts water_level_delta.

    Returns:
      X_train, y_train, X_val, y_val, feat_scaler, targ_scaler
    """

    df_feat = df.copy()

    # 1) Cyclic annual seasonality
    doy   = df_feat.index.dayofyear.values
    angle = 2 * np.pi * doy / 365.25
    df_feat["sin_year"] = np.sin(angle)
    df_feat["cos_year"] = np.cos(angle)

    # 2) Rolling sums of rainfall: 7,14,30,60,90 days
    for w in (7, 14, 30, 60, 90):
        df_feat[f"rainfall_rollsum_{w}"] = df_feat["rainfall"].rolling(w).sum()

    # 3) Rolling mean of temp over 7 days
    df_feat["temperature_rollmean_7"] = df_feat["temperature"].rolling(7).mean()

    # 4) Calendar
    df_feat["month"]     = df_feat.index.month / 12.0
    df_feat["dayofyear"] = df_feat.index.dayofyear / 366.0

    # 5) Δ-target: day-to-day change in water level
    df_feat["water_level_delta"] = df_feat["water_level"].diff()

    # 6) Drop any rows with NaNs from rolling or diff
    df_feat = df_feat.dropna()

    # 7) Split train vs val
    train_df = df_feat.loc[:train_end]
    val_df   = df_feat.loc[val_start:val_end]

    # 8) Define feature columns & target
    feature_cols = [
        "water_level", "rainfall", "temperature",
        "sin_year", "cos_year",
        "rainfall_rollsum_7", "rainfall_rollsum_14",
        "rainfall_rollsum_30", "rainfall_rollsum_60",
        "rainfall_rollsum_90",
        "temperature_rollmean_7",
        "month", "dayofyear"
    ]
```

**Figure 3.3.3: build_features.py file to build required building features**

The defined TensorFlow-Lite LSTM receives the pre-processed matrix and completes the inference of the whole fifteen-day horizon in 110 ms. It is all done on the edge SBC's Cortex-A53 CPU. The dropout 5th and 95th percentiles are provided via two auxiliary columns. This is an addition to an array of absolute stage forecasts. These forecasts are continuously broadcasting (see Figure 3.3.4) to the decision-control service. The broadcasting is done via Redis and stored in InfluxDB.



```python
import os
import numpy as np
import pandas as pd
import joblib
import tensorflow as tf
from load_data import load_data


def make_forecast(
    model_path: str           = "models/lstm_model.keras",
    feat_scaler_path: str     = "models/feature_scaler.pkl",
    targ_scaler_path: str     = "models/target_scaler.pkl",
    bias_path: str            = "models/delta_bias.npy",
    df_hist: pd.DataFrame     = None,
    future_weather_path: str= "data/future_rain_temp.csv",
    time_steps: int           = 30,
    horizon: int              = 15,
):
    """
    1) Load history (or via load_data)
    2) Read future rain/temp, reindex history to include those dates
    3) Engineer features & take last `time_steps`
    4) Scale with feat_scaler
    5) Predict Δ, invert scale with targ_scaler
    6) Subtract bias, cumsum + last observed → absolute forecast
    """

    # 1) history
    if df_hist is None:
        df_hist = load_data()
    last_level = df_hist["water_level"].iloc[-1]

    # 2) future rain/temp
    df_fw = pd.read_csv(future_weather_path)
    df_fw = df_fw.rename(columns={df_fw.columns[0]:"date",
                                  df_fw.columns[1]:"rainfall",
                                  df_fw.columns[2]:"temperature"})
    df_fw["date"] = pd.to_datetime(df_fw["date"], dayfirst=True)
    df_fw = df_fw.set_index("date").asfreq("D")

    # reindex history to cover future
    all_dates = df_hist.index.union(df_fw.index)
    df_comb  = df_hist.reindex(all_dates)[["water_level","rainfall","temperature"]].copy()
    # overwrite exogenous
    df_comb.loc[df_fw.index, ["rainfall","temperature"]] = df_fw[["rainfall","temperature"]]

    # 3) feature engineering (same as build_features)
    df_feat = df_comb.copy()
    doy   = df_feat.index.dayofyear.values
    angle = 2*np.pi*doy/365.25
    df_feat["sin_year"] = np.sin(angle)
    df_feat["cos_year"] = np.cos(angle)
    for w in (7,14,30,60,90):
        df_feat[f"rainfall_rollsum_{w}"] = df_feat["rainfall"].rolling(w).sum()
    df_feat["temperature_rollmean_7"] = df_feat["temperature"].rolling(7).mean()
    df_feat["month"]     = df_feat.index.month/12.0
    df_feat["dayofyear"] = df_feat.index.dayofyear/366.0
    df_feat = df_feat.dropna(subset=[
      "rainfall_rollsum_7","rainfall_rollsum_14","rainfall_rollsum_30",
      "rainfall_rollsum_60","rainfall_rollsum_90","temperature_rollmean_7"
```

**Figure 3.3.4: make_forcecast.py to predict the future water level**

## 3.4 Web and Mobile Supervisory Application

React and TypeScript were used to create a web console that provided control-room employees with a clear, up-to-date and user-friendly view of the reservoir. The primary screen is divided into two sections. First is a real-time graphic of the tank and gate. It shows a live chart that displays historical water levels together with the 15-day forecast. I used WebSocket to update data like stage, flow, gate angle, and weather predictions every two seconds. Users can check the moment the prediction was created as well as the model's 5th to 95th percentile range. This occurs if when they hover over a future time step on the forecast. It helps operators in evaluating the prediction's scope and distinctiveness. If the flood safety limit is predicted to be reached within 72 hours, a banner at the top becomes yellow. If the event is near, it turns red. They can also see the numerical explanation of the decision-making of the control system by just clicking the banner and modal will be opened. However, they will see the resulted trend somehow like in the Figure 3.4.1.

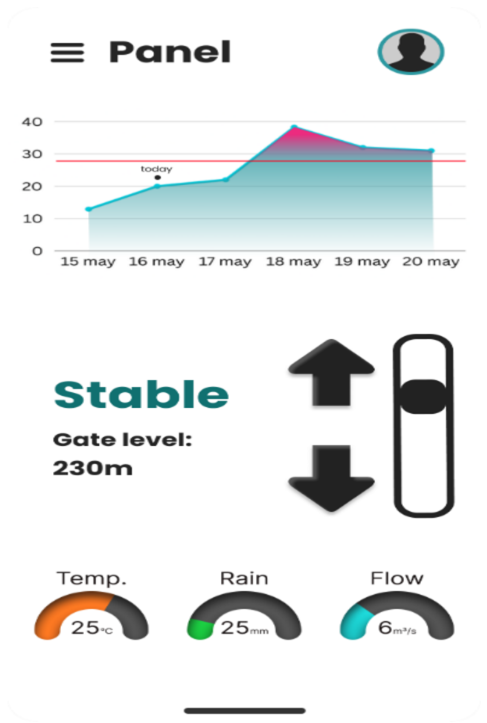


**Figure 3.4.1: Mobile app demo design of the application**

The Flutter mobile application used by field engineers to run quickly over 4G and have a small screen size. Users with the "Supervisor" role can access a slider after logging in using OAuth2. Then, they can manually change the gate angle in 1° increments up to 90°. The application displays a confirmation box when a new setting is entered so users can avoid errors. It notifies the system gateway with a signed HTTPS request after confirmation and it will help to understand the problems easily.

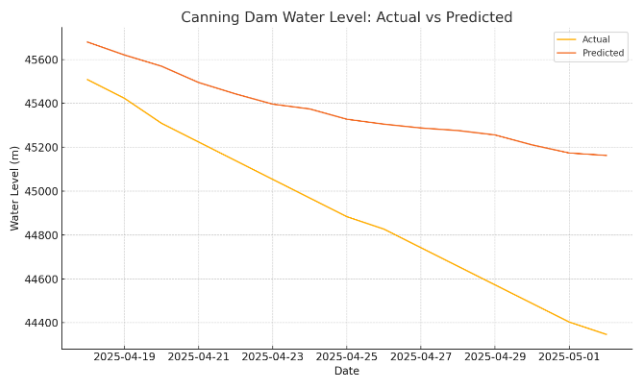


**Figure 3.4.2: Predicted vs Actual data**

Operators can raise the gate above automated settings, because there can be necessary moments that they should do it. But they are not allowed to lower it below the minimum safe angle. It is defined according to a set of criteria that the service verifies. Every activity, whether manual or automated, is recorded. They are along with the model version, gate angle, reservoir level, and user ID. A secure event system is used to store these logs. The responsiveness of the system will valued by the operators. An on-screen confirmation will inform the users that the instruction had been received or not. The gate started moving less than a second after the slider was adjusted. And this is due to connection problems. Two changes for next editions were recommended by users informally. Employees don't need to keep the app open because push alerts are sent out for necessary threshold occurrences. So, they can be notified immediately. An offline "shadow mode" that locally saves the previous 100 sensor readings can be helpful. This is because of the places with poor mobile service. The flexible service and message-bus architecture already in place makes it simple to add these functions. So, in near future these can be applied.

## IV. CONCLUSION

The smaller scaled hardware prototype demonstrated that it is possible to synchronize the real-time forecast with physical gate movement. I used the prototype that uses the servo motor to control dam gate to simulate the water level increase with excessing the safe limit. It smoothly and automatically opened gate to free up space for extra water inflow. Many inflow situations were tested and the ultrasonic distance sensor, flowrate sensor, and the Arduino Mega 2560 functioned in harmony, proving that low-cost, low-power electronics can still maintain the water level within the predefined limit and hence avoid surges that exceed forecasts.

The multivariate LSTM model, which was based on historical data and new weather forecasts, is one of the main parts of my project, because it forecasts the 15 day trajectory everyday to determine the gate opening ratio. The network was able to represent both sudden storm reactions and steady seasonal increases by gathering thirty days' worth of antecedent rainfall, temperature, flow, and level. It also expressed their cumulative effect through learnt long-range dependencies. During validation testing, the model's trend and turning points closely mirrored actual changes. This demonstrates how data-driven forecasting can give the accuracy as well as speed required for efficient reservoir management.

A lightweight microservice architecture provided smooth user control and supervision based on predictive core. Web dashboard and mobile app shows live data. Permitted

ISSN (online): 2581-3048
Volume 9, Issue 7, pp 14-23, July-2025
https://doi.org/10.47001/IRJIET/2025.907002

authorized workers are able to adjust the gate opening ratio using a basic slider. The ingest, forecast, and decision containers communicated with each other over an encrypted bus. Features like role-based access, automated logging, and quick visual feedback helped the system to gain the trust of the operator. Its modular architecture also made sure that the critical connection between action and prediction wouldn't be broken by unexpected increases in user traffic.

## REFERENCES

[1] Muhamad, Ali Najah Ahmed, Marlinda Abdul Malek, Ahmed Hussein Birima, Munir, M., M. Sherif, and A. Elshafie.. Exploring machine learning algorithms for accurate water level forecasting in Muda river, Malaysia. *Heliyon*, 9(7), pp.e17689–e17689, 2023. doi: https://doi.org/10.1016/j.heliyon.2023.e17689.

[2] Y.O. Ouma, D.B. Moalafhi, G. Anderson, B. Nkwae, P. Odirile, B.P. Parida, and J. Qi. Dam Water Level Prediction Using Vector AutoRegression, Random Forest Regression and MLP-ANN Models Based on Land-Use and Climate Factors. *Sustainability*, 14(22), p. 14934., 2022 doi: https://doi.org/10.3390/su142214934.

[3] https://www.frontiersin.org/journals/marine-science/articles/10.3389/fmars.2024.1470320/full

[4] R. Liu, C. Ye, P. Yang, Z. Miao, B. Liu, and Y. Chen. Short-Term Prediction Model of Water Level Based on ATT-ConvLSTM. 2022 doi: https://doi.org/10.1145/3528114.3528128.

[5] https://www.mdpi.com/2227-9709/11/4/73

[6] https://www.mdpi.com/2071-1050/14/22/14934

[7] S. Yi, and J. Yi. Reservoir-based flood forecasting and warning: deep learning versus machine learning. *Applied Water Science*, 14(11), 2024. doi: https://doi.org/10.1007/s13201-024-02298-w.

[8] https://ieeexplore.ieee.org/document/10949142

[9] I.R. Widiasari, and R. Efendi. Utilizing LSTM-GRU for IOT-Based Water Level Prediction Using Multi-Variable Rainfall Time Series Data. *Informatics*, 11(4), p.73, 2024. doi: https://doi.org/10.3390/informatics11040073.

[10] https://www.mdpi.com/2079-9292/10/8/882

[11] https://ieeexplore.ieee.org/document/5874897

[12] https://rsisinternational.org/Issue0/pdf/paper19.pdf

[13] A.Kumar, V.K. Pal, V.K. Chaurasiya, and A. Singh. Dams Gate Control Using Programmable Logic Controller and SCADA. *International Journal of Advance Research and Innovation*, 9(3), pp.57–63, 2021. doi:https://doi.org/10.51976/ijari.932109

[14] https://www.researchgate.net/publication/387661552_Hybrid_System_Prototype_for_Dam_Water_Level_Control_System_to_Irrigating_Rice_Fields

[15] S.S. Lin, K.Y. Zhu, X.H. Zhang, Y.C. Liu, and C.Y. Wang. Development of a Microservice-Based Storm Sewer Simulation System with IoT Devices for Early Warning in Urban Areas. *Smart Cities*, 6(6), pp.3411–3426, 2023. doi: https://doi.org/10.3390/smartcities6060151.

[16] https://www.sciencedirect.com/science/article/pii/S0167739X24000529?via%3Dihub

[17] The 20 th week Measuring flow in open channels (weirs) Broad-Crested and Sharp-Crested Weirs. (n.d.). Available at: https://www.uobabylon.edu.iq/eprints/publication_3_3905_6015.pdf.

[18] International Commission on Large Dams (ICOLD). Bulletin 154: Dam Safety Management — Operational Safety of Dams. *Paris: ICOLD*, 2016.

[19] F. Kratzert, D. Klotz, C. Brenner, K. Schulz, & M. Herrnegger. Rainfall–runoff modelling using Long Short-Term Memory (LSTM) networks. *Hydrology and Earth System Sciences*, 22, 6005–6022, 2018. https://doi.org/10.5194/hess-22-6005-2018

[20] D. Gómez, C. E. Torres, & J. Porras. An edge–cloud architecture for real-time monitoring and control of hydrological infrastructure. *Sensors*, 21(14), 4725, 2021. https://doi.org/10.3390/s21144725

**Citation of this Article:**

Kamala Oghuz, & Elchin Bayramli. (2025). Predictive Flood Prevention System: A Microarchitecture-Based Approach for Intelligent Dam Control. *International Research Journal of Innovations in Engineering and Technology - IRJIET*, 9(7), 14-23. Article DOI https://doi.org/10.47001/IRJIET/2025.907002

*******