1.2.2 Model Baselines Report

1. Introduction to water-anomaly detection

This project builds an end-to-end anomaly analytics pipeline for daily water consumption data. The raw input is a large, continuous time series (millions of daily records) spanning multiple customers and meters, enriched with geospatial, hardware and weather information. There is no ground-truth label for "anomaly", so the design uses a two-layer approach. First, an unsupervised k-means module learns "normal" consumption patterns per meter brand and flags anomalous days based on distance to cluster centroids. Second, a supervised LSTM model uses those anomaly labels, together with the full temporal and contextual feature set, to estimate the probability of future anomalies at several horizons (next day, next week, next month). The overall goal is to generate robust, interpretable anomaly labels and then forecast short-term risk in a way that is scalable, brand-aware and suitable for operational decision-making.

The workflow is structured as a single feature-engineering backbone feeding both the k-means and LSTM layers. Raw daily observations contain meter identifiers (POLIZA_SUMINISTRO), timestamps (FECHA), measured consumption (CONSUMO_REAL), census and municipality codes (SECCIO_CENSAL, NUM_DTE_MUNI, NUM_COMPLET), installation date (DATA_INST_COMP), weather variables (tavg, tmin, tmax, prcp), usage category flags (US_AIGUA_GEST_*), meter model flags (CODI_MODEL_*), brand flags (MARCA_COMP_*), service area codes (NUM_MUN_SGAB_*) and diameter flags (DIAM_COMP_*). Feature engineering converts this heterogeneous mix into a rich set of numeric time-series features per meter. From that full feature space, we define a subset of "baseline pattern" variables used by k-means (level, seasonality, volatility, smooth trend), while the LSTM sees the entire engineered feature set, including more aggressive anomaly-intensity signals. The k-means block groups days by brand-specific consumption patterns, measures distances to centroids, and marks the top 2.5 % of farthest points in each cluster as anomalies. Those anomaly indicators and distances then become additional input features and supervised targets for the LSTM, which operates on sliding sequences of fixed length and predicts the probability that an anomaly will occur within each future horizon. Per-brand models are trained to capture hardware-dependent behaviour, and training/validation splitting is done at meter level to avoid leakage across overlapping windows.

2. Future Engineering

Feature engineering starts by standardising the temporal and meter structure of the data. The date column (FECHA) is cast to a proper datetime type and each meter (POLIZA_SUMINISTRO) is sorted chronologically; this ensures that all rolling and lagged computations are well-defined and that later windowing for the LSTM is consistent. The core consumption signal CONSUMO_REAL is transformed into a log-scaled variable CONSUMO_LOG = log(1 + CONSUMO_REAL), which compresses the heavy right tail of consumption, stabilises variance, and makes differences and ratios more interpretable for both clustering and sequence modelling. On top of that, two per-meter normalisations are computed: a classical z-score CONSUMO_ZSCORE, which subtracts the per-meter mean and divides by the per-meter standard deviation, and a robust z-score CONSUMO_ROBUST_ZSCORE, which uses the per-meter median and median absolute deviation (MAD, rescaled to be comparable to a standard deviation). These z-scores do not enter the k-means feature subset, but they are retained as high-level anomaly-intensity indicators for the LSTM to understand how extreme a given day is relative to that meter's history.

Temporal structure is encoded in several complementary ways. From the calendar date we derive day of week, day of month, month, quarter and ISO week number; we then add a binary IS_WEEKEND flag, and a coarse season code SEASON $\in \{1,2,3,4\}$ representing winter, spring, summer and autumn in the northern hemisphere. To preserve the cyclical nature of the calendar, day of week and month are also encoded with sine and cosine transforms (DAY_OF_WEEK_SIN/COS, MONTH_SIN/COS), so that, for example, December (12) and January (1) are close in the embedding space. These features allow k-means to separate structurally different consumption regimes (weekday vs weekend, summer vs winter), and they provide the LSTM with explicit context about the position of each day in recurrent seasonal cycles.

A key block in the feature pipeline consists of rolling statistics computed per meter over multiple time scales. For each meter time series we define windows of 7, 14, 30 and 90 days, always using a minimum history length (e.g. 60 days) before trusting these statistics to avoid spurious behaviour at the very start of a meter's life. For each window we compute a rolling mean, standard deviation and median of CONSUMO_REAL, along with a coefficient of variation (rolling standard deviation divided by rolling mean) and a standardised deviation of the current value from the rolling mean. The short windows (7 and 14 days) capture weekly and bi-weekly patterns and volatility, while the longer windows (30 and 90 days) summarise monthly and seasonal baselines. The coefficients of variation ROLLING_CV_* indicate how stable consumption is relative to its level, which is useful to distinguish "steady" household use from more erratic profiles, and the DEVIATION_FROM_MEAN_* features measure how unusual today's reading is compared to its recent local context. For k-means we keep a reduced subset of these rolling features (for example, 7-day and 30-day CV and deviations) to capture essential dynamics without exploding dimensionality; the full set remains available to the LSTM.

Lagged features capture short- and medium-term temporal dependencies. For each meter we shift the raw consumption series to create lag-1, lag-7 and lag-30 variables (CONSUMO_LAG_1D, _7D, _30D), representing yesterday's consumption, consumption on the same weekday a week ago, and consumption roughly a month ago. Instead of raw percentage changes, which can be very noisy and explode near zero, we compute smoothed change signals such as average log differences over the last 7 and 30 days, optionally clipped to a reasonable range. This provides a robust notion of trend: is consumption increasing or decreasing, and how quickly, without being dominated by single-day spikes. These lag/trend features are not used directly by k-means if they are too noisy, but they are central for the LSTM, which is designed to learn temporal patterns over windows of 90 days and beyond.

Meter characteristics and categorical encodings are incorporated to reflect hardware and configuration effects. The diameter of the meter is reconstructed from one-hot DIAM_COMP_* flags into a numeric DIAMETER feature, and an associated DIAMETER_RISK score is constructed to reflect the empirically observed risk that smaller diameters (e.g. 15 mm) may be more prone to anomalies. The binary CODI_MODEL_* flags for different meter models are counted into NUM_MODELS, a proxy for hardware complexity per installation, and are also retained as individual indicator features for the LSTM. Brand information is extracted from MARCA_COMP_* one-hot columns into a single BRAND tag; this tag is used as the grouping key for k-means clustering and for assigning LSTM models, while the underlying brand flags remain available as numeric input for the LSTM. In other words, k-means and the LSTM each have a separate model per brand, but see the full suite of model and hardware features as context inside each brand.

Weather features connect consumption behaviour to environmental drivers. The raw daily average, minimum and maximum temperatures (tavg, tmin, tmax) and precipitation (prcp) are retained, and additional variables are derived, such as diurnal temperature range (tmax - tmin), indicators of hot or cold days relative to seasonal norms, and rolling aggregates of precipitation over 7 and 30 days. These rolling precipitation sums and dry-spell indicators encode phenomena like sustained rainfall (which may reduce outdoor use) or prolonged dry periods (which may increase irrigation or leak detection sensitivity). All weather-related columns are included in the LSTM feature space so that the network can learn, for example, that high consumption peaks during heat waves are less worrying than similar peaks in winter. For k-means we include a subset of weather features that shift the baseline pattern in a smooth way (e.g. tavg, precipitation aggregates), but we avoid using weather-derived anomaly scores directly in clustering.

Beyond basic level and seasonality, we create explicit consumption pattern and anomaly-proxy features. A binary IS_ZERO_CONSUMPTION flag identifies days with no registered usage, which in some contexts may indicate a closed property or a measurement issue. A per-meter consumption percentile is computed on a rolling 365-day window (CONSUMPTION_PERCENTILE_365D), so that each day is ranked relative to the last year of that meter's own history; from this we define IS_HIGH_CONSUMPTION and IS_LOW_CONSUMPTION flags for values above the 95th percentile and below the 5th percentile (excluding true zeros). A MACD-like indicator compares the 7-day and 30-day rolling means (MACD = ROLLING_MEAN_7D – ROLLING_MEAN_30D), capturing short-term acceleration versus longer-term baseline, and a SUDDEN_CHANGE_SCORE divides the absolute deviation from the 7-day rolling median by the 7-day rolling standard deviation, highlighting abrupt jumps that are large compared to recent noise. These fields are all kept as high-level anomaly signals for the LSTM, but they are intentionally excluded from the subset of features used to train k-means, to avoid circularity where cluster structure is driven primarily by "outlier-ness" rather than true baseline behaviour.

Finally, the feature engineering step includes careful cleaning and consolidation. All rolling statistics are computed per meter on time-ordered data; initial NaN values near the start of the series are imputed by forward and backward filling within each meter, and any remaining missing or infinite values are replaced with sensible defaults (zeros for centered quantities, capped values for clipped differences). The resulting feature-engineered dataframe includes identifier columns (POLIZA_SUMINISTRO, FECHA, CONSUMO_REAL, BRAND, and a derived brand_model tag mirroring the brand) plus the full set of engineered numeric features described above. This single dataframe is the canonical input to the rest of the pipeline: the k-means module receives it together with an explicit list of "baseline" feature columns to use for clustering; after k-means adds cluster, distance and is_anomaly labels, the LSTM module uses the entire numeric feature set, including those new fields, to build sliding windows and estimate future anomaly probabilities.

3. K-Means-based anomaly detection

The second core block of the pipeline is the k-means anomaly detection module, whose purpose is to transform unlabeled daily consumption records into a consistent, reproducible notion of "anomalous day" per brand. Conceptually, the approach is to let the algorithm learn what typical behaviour looks like for each brand of meter, in the space of baseline features that describe level, seasonality and volatility but not explicit outlier scores, and then define anomalies as the small fraction of points that lie far away from the cluster centroids. The output of this block is the same feature-engineered dataframe augmented with three key columns: a discrete cluster label, a continuous distance to the assigned centroid, and a binary is_anomaly flag, plus a derived anomaly_score that normalises distance within each cluster. These fields are then used both directly for descriptive analytics and as supervisory signals and features in the subsequent LSTM stage.

The algorithm operates separately for each meter brand. During feature engineering, the one-hot MARCA_COMP_* flags are collapsed into a single categorical BRAND tag. For the k-means module, the data is partitioned by this tag: each brand receives its own model, provided that it has at least a minimum number of records (for example, at least 30 × min_clusters rows). This per-brand partitioning allows clustering to capture patterns specific to each hardware family and avoids the situation where very large brands dominate the learned centroids for smaller ones. Inside each brand group, we build a design matrix by selecting a curated subset of the engineered features: log consumption, per-meter robust normalisations, cyclical calendar encodings, a reduced set of rolling coefficients of variation and deviations from rolling means (e.g. 7-day and 30-day windows), basic weather aggregates and simple meter characteristics such as DIAMETER_RISK. Features that directly encode outlier intensity (z-scores, sudden change scores, annual percentiles) are deliberately excluded from this subset to ensure that clusters are defined by normal behaviour rather than by pre-computed anomaly flags. All selected columns are numeric and are scaled with a RobustScaler, which centres each feature by its median and scales it by its interquartile range. This choice is more robust than standard scaling in the presence of outliers, and helps ensure that no single feature with a long tail dominates the distance computation.

Because the dataset can contain millions of daily records, the k-means implementation is designed to be both scalable and reproducible. Within each brand, we optionally subsample rows for clustering based on a configurable fraction and/or an absolute cap; this keeps the computational cost of fitting within bounds while retaining coverage of the distribution. All random sampling is driven by a fixed random_state, so repeated runs with the same configuration yield the same subsample indices and, consequently, the same fitted models. Clustering is performed with MiniBatchKMeans rather than full KMeans, using relatively large batch sizes and a small number of n_init restarts. Mini-batch updates allow the model to converge quickly on representative centroids for each brand, even when the total number of points is high. Once a model is fitted on the subsample, all available rows for that brand are transformed with the same scaler and assigned to clusters using the model's predict method.

Selecting the number of clusters per brand is handled through an explicit optimisation loop over a user-specified range [min_clusters, max_clusters]. For each candidate k, the algorithm draws a subsample of points (again with a cap to avoid quadratic computation) and fits a fresh MiniBatchKMeans model on that subset. It then computes several clustering quality metrics: the silhouette score, which measures how well separated clusters are in Euclidean space; the Davies–Bouldin index, which captures compactness and separation (lower is better); the Calinski–Harabasz score, which measures between-cluster dispersion relative to within-cluster dispersion (higher is better); and the model's inertia, a within-cluster sum of squared distances (lower is better). Silhouette computation, which is typically the most expensive component, is itself performed on a subsample of the candidate subset to keep cost under control. Each metric is normalised across the tested k values and combined into a scalar score, with silhouette and Calinski–Harabasz given higher weights, and Davies–Bouldin and inertia acting as regularisers for compactness. The best k is initially defined as the one with the highest combined score, but a small "tie band" is applied so that if several k values achieve near-identical scores, the smallest of those is chosen, favouring simpler models when quality differences are marginal. If min_clusters equals max_clusters, or if there are too few points in a brand to meaningfully explore the range, the optimization is skipped and a fixed k is used.

Once a k is chosen and a final MiniBatchKMeans model is fitted on the brand's training subset, each scaled point is assigned to a cluster and its distance to the corresponding centroid is computed. Two distance metrics are supported: Euclidean and Mahalanobis. For Euclidean distance, the model simply subtracts the centroid coordinates from each point and computes the L2 norm of the difference. For Mahalanobis distance, which accounts for the covariance structure of each cluster, the algorithm first estimates a covariance matrix for the points in each cluster, regularises it by adding a small multiple of the identity to ensure invertibility, and then inverts it to obtain the precision matrix. The Mahalanobis distance is then computed in vectorised form as the square root of $\delta^T \Sigma^{-1} \delta$ for each point-centroid pair, where δ is the difference vector and Σ^{-1} is the cluster's precision matrix. This computation is fully vectorised over the cluster's points using efficient linear algebra operations, which is crucial for performance on large datasets. If the covariance matrix for a cluster is ill-conditioned or has too few points, the algorithm falls back to Euclidean distance for that cluster to ensure robustness.

Anomaly detection is performed within each cluster based on the empirical distribution of distances. For cluster c, the algorithm gathers all distances of points assigned to c and computes summary statistics such as the mean and standard deviation. Rather than using a fixed multiple of the standard deviation, which would translate into varying anomaly rates depending on the cluster's shape, the method uses a tail quantile: all points whose distance lies in the top α fraction of the cluster's distance distribution are flagged as anomalies, where α is a configurable "anomaly tail fraction" (for example, 0.025 to mark the farthest 2.5 % of points as anomalies). Formally, if q is the $(1 - \alpha)$ quantile of the distances in cluster c, an observation i in that cluster is assigned is_anomaly $= 1$ iff distance_i $\geq$ q. This results in an approximately uniform anomaly rate per cluster (modulo rounding in small clusters), and ensures that the global anomaly rate is controlled by α rather than by ad hoc thresholds. Alongside is_anomaly, an anomaly_score is defined as the ratio between the actual distance and the cluster's threshold distance, capped at a maximum value; this creates a dimensionless measure of "how far into the tail" each point sits, which is later used by the LSTM as a graded signal.

The k-means module logs and persists detailed information for each brand. Fitted models and scalers are saved to disk per brand, along with the per-cluster distance thresholds and summary statistics (mean, standard deviation, cluster size and anomaly count). A consolidated metadata file records the global configuration (cluster range, distance metric, anomaly tail fraction), the list of brands with trained models, and their clustering quality metrics. Diagnostic plots are generated for each brand: if k optimisation was performed, a multi-panel figure shows the silhouette, Davies–Bouldin, Calinski–Harabasz and inertia values as functions of k, with the chosen k marked; a second figure displays bar charts of cluster sizes overlaid with line plots of anomaly counts; a third figure shows histograms and kernel density estimates of distance distributions per cluster with the anomaly threshold indicated; and a fourth figure visualises time series for a small sample of meters, highlighting days flagged as anomalies. Together, these artefacts allow practitioners to inspect how well clusters are separated, how anomalies are distributed across clusters, and how anomaly labels manifest in the original temporal context.

Finally, the k-means code exposes a scoring interface for new data in the form of a score_df method. Given a new feature-engineered dataframe with the same schema as used in training, it reconstructs brand groups, applies the corresponding scaler and k-means model for each group, recomputes distances to centroids, and applies the stored per-cluster thresholds to produce new cluster, distance, anomaly_score and is_anomaly values. If a brand appears at prediction time for which no model has been trained, the pipeline can either skip it or fall back to a default model if configured. This symmetry between training and scoring, together with the explicit per-brand metadata and the use of fixed random seeds and sampling caps, ensures that the anomaly labelling stage is both scalable and reproducible, and that the LSTM layer downstream always receives anomaly signals that are consistent with the k-means model used to generate them.

4. LSTM-based anomaly risk forecasting

The final block of the pipeline is a supervised sequence model based on Long Short-Term Memory (LSTM) networks. Its role is different from the k-means module: instead of defining what an anomaly is, the LSTM estimates the probability that an anomaly will occur in the near future, at several horizons (next day, next week, next month). The motivation for using an LSTM is that anomaly risk in water consumption is inherently temporal: it depends not only on the current consumption level but also on how that level has evolved over recent weeks and months, how it compares to the meter’s own history, and how k-means has been flagging past days as normal or anomalous. Recurrent architectures such as LSTMs are designed precisely to capture such temporal dependencies, including long-range effects, while handling variable-length and noisy sequences.

The LSTM operates not on single days but on sliding sequences of fixed length, typically 90 days, built per meter. Starting from the k-means-labeled dataframe (which now includes is_anomaly, cluster, distance, anomaly_score and all engineered features), the pipeline first sorts each meter’s history by date and then slides a window of length sequence_length over it with a configurable stride. For each window, the model’s input is a matrix of shape (sequence length × number of features), where each row is one day and each column is one engineered feature; the output is a vector of three binary targets indicating whether any anomaly occurs in the next day, in the next seven days, or in the next thirty days. These targets are computed by looking ahead from the end of the window and checking the future is_anomaly column, so the model never sees information from the future in its inputs. By sliding the window, the pipeline generates many overlapping sequences per meter, but in training and validation we ensure that no meter contributes sequences to both sets, which avoids leakage through overlapping windows and gives a more honest estimate of generalisation across customers and time.

Unlike k-means, which uses a restricted subset of features, the LSTM sees the full engineered feature space. For each day in a sequence, it receives level and normalisation features (CONSUMO_LOG, z-scores, robust z-scores), temporal encodings (cyclical month and weekday, season, weekend), rolling statistics at different scales (short- and long-term means, variances, coefficients of variation and deviations from rolling baselines), lagged values and smoothed change indicators, meter and hardware characteristics (diameter, DIAMETER_RISK, model flags, brand flags, number of active models), weather variables and derived quantities (temperature, precipitation, rolling aggregates, dry/wet spell indicators), and all outputs from the k-means stage. The k-means outputs include the discrete cluster label and continuous measures of distance and anomaly intensity. These values are treated as regular numeric features, so the recurrent model can learn patterns such as “a series of days with high anomaly scores, even if not all flagged as anomalies, is a strong precursor to a future event” or “certain cluster regimes combined with specific weather conditions are more prone to anomalies". In other words, k-means acts both as a label generator and as a feature extractor: it supplies the is_anomaly labels used to build the future targets and it produces additional context variables that the LSTM can exploit to understand the state of the system.

To keep models specialised and to mirror the k-means design, a separate LSTM is trained for each brand. During dataset preparation, the BRAND tag produced in feature engineering is propagated as brand_model, and sequences are grouped by this field. For each brand group, the pipeline first splits meters into training and validation sets based on the meter_id rather than splitting individual sequences randomly. All sequences from "train meters" go into the training set, and all sequences from "validation meters" go into the validation set; this prevents nearly identical windows for the same meter from appearing in both sets. The raw sequence arrays are then reshaped and normalised with a StandardScaler fitted only on the training data (flattening time and sample dimensions); the scaling parameters are cleaned to remove NaNs and zeros, and the resulting scaler is stored per brand for later reuse in prediction.

The LSTM architecture itself is a compact but expressive two-layer recurrent network followed by dense layers. The model takes sequences of shape (sequence length, number of features) and passes them through two stacked LSTM layers: the first returns the full sequence to allow the second to process the temporal structure further, and the second outputs only a final hidden state summarising the 90-day window. Batch normalisation and dropout are applied between layers to stabilise training and reduce overfitting. The recurrent backbone is followed by two fully connected layers with ReLU activation and dropout, which learn non-linear combinations of the temporal features. The final output layer has three units, one per horizon (day, week, month), with sigmoid activations, so each unit returns a probability between 0 and 1 for "an anomaly will occur in this horizon". The model is trained with a binary cross-entropy loss and standard metrics such as accuracy, AUC, precision and recall.

Class imbalance is a central issue for this task: even after aggregating anomalies at the sequence level ("is there at least one anomaly in the next week"), positive sequences are rare compared to negatives. To address this, the pipeline computes per-sample, per-horizon weights and feeds them as sample_weight to the optimiser. For each horizon and brand, it measures the proportion of positive and negative labels in the training set and assigns a higher weight to positives, roughly proportional to the ratio of negatives to positives, with a lower bound to avoid extreme values. These weights are then multiplied by horizon-specific importance coefficients (for example, day > week > month), reflecting the operational preference for catching very near-term events more aggressively. This weighting scheme means that errors on positive sequences, especially at the day horizon, contribute more to the loss than errors on negative sequences, which helps the model learn to recognise rare patterns that precede anomalies instead of collapsing to “no anomaly” everywhere. At the same time, the per-horizon nature of the weights allows the network to calibrate differently for short- and long-term risk.

Training is guided by several mechanisms to control overfitting and to produce well-documented results. For each brand, once the training and validation sets are prepared and the LSTM is built, the pipeline trains with early stopping on validation loss, ReduceLROnPlateau to lower the learning rate when progress slows, and model checkpointing to persist the best weights. During and after training, it tracks and saves the full history of loss and metric curves for both training and validation, and computes per-horizon metrics on the validation set by thresholding the predicted probabilities at 0.5. For each horizon it evaluates precision, recall, F1 score and ROC AUC. These metrics and the training curves are plotted in multi-panel figures that show loss, accuracy, AUC, precision and recall over epochs, as well as summary bar charts of precision, recall, F1 and AUC for the day, week and month horizons. All models, scalers and metadata (feature names, sequence length, horizons, metrics) are saved to disk in a structured directory per brand and per training run, and a consolidated training report summarises performance across brands.

The prediction pipeline mirrors the training configuration in a consistent way. Given a new feature-engineered and k-means-labeled dataframe, it first builds sequences with the same prepare_lstm_dataset function, using the same sequence length and horizons as during training. For each brand group, it loads the corresponding LSTM model, scaler and metadata. Sequences are scaled using the stored scaler, reshaped into the appropriate input shape, and passed through the model to obtain probability predictions for each horizon. For each sequence, the pipeline compares these probabilities to configurable classification thresholds. Thresholds can be specified as a single scalar applied to all horizons or as a horizon-specific dictionary; internally, they are used both to produce binary predictions (pred_anomaly_day, pred_anomaly_week, pred_anomaly_month) and to derive an overall qualitative risk level such as LOW, MEDIUM, HIGH or CRITICAL. The risk level is computed from the maximum predicted probability across horizons and a set of derived cut-offs based on the classification thresholds, so that a consistently high probability in any horizon drives the risk upwards. The output is a dataframe with one row per sequence, including meter identifier, brand, sequence time span, probabilities and binary predictions per horizon, and the risk level. This allows downstream systems to aggregate risk at the meter level (for example, by selecting the maximum risk across all sequences for a given meter) and to prioritise inspections or alerts accordingly.

5. Conclusions

The resulting system is a coherent, two-layer anomaly analytics pipeline that starts from raw daily water consumption data and produces both retrospective anomaly labels and prospective risk scores. The first layer, based on per-brand k-means clustering, provides an unsupervised, model-free definition of anomalous days: it learns typical consumption regimes from a carefully chosen subset of features, measures distances to cluster centroids with scalable vectorised computations, and marks the top tail of each cluster's distance distribution as anomalous. Its design explicitly separates baseline pattern features from anomaly-intensity features, uses robust scaling and per-cluster quantile thresholds, and provides rich diagnostics and a reproducible scoring interface. The second layer, based on per-brand LSTM models, takes the full engineered feature set, including k-means outputs, and learns to predict the probability of future anomalies at several horizons. It is built to respect temporal structure and avoid leakage through meter-level splitting; it accounts for severe class imbalance with per-horizon sample weights; and it exposes interpretable, horizon-specific metrics and risk scores suitable for operational decisions.

By combining these two layers, the pipeline leverages the strengths of both unsupervised and supervised learning. K-means supplies consistent and explainable labels where none existed, allowing the system to bootstrap supervision from structure in the data rather than from manually curated events. The LSTM then builds on top of these labels, integrating them with temporal, hardware and environmental context to produce forward-looking forecasts that can anticipate anomalies before they happen. The shared feature engineering backbone ensures that both layers work on a rich, well-aligned representation of the time series, and the per-brand modelling strategy captures hardware-specific behaviour while remaining scalable through sampling and efficient implementations. The design is modular, so improvements in any component—for example, a refined anomaly tail fraction, a more detailed weather model, or a more advanced hyperparameter search for LSTM architectures—can be incorporated without breaking the overall workflow. As a result, the system forms a solid technical foundation for data-driven water consumption monitoring, anomaly detection and proactive maintenance, and it can be extended or adapted to other utility or sensor contexts with similar characteristics.