Pearls AQI Predictor

Report

Author: Muhammad Raza
Email: muhammadraza3009@gmail.com
Submitted To: 10Pearls Shine
Internship Batch: Cohort 7
Status: Live & Operational
Live App: https://mraza-aqi-predictor.streamlit.app
GitHub: https://github.com/muhammadraza30/pearls-aqi-predictor

Executive Summary

Air quality in Karachi is a genuine public health crisis. AQI levels regularly climb into hazardous categories, yet residents have had no reliable tool to anticipate what the air will look like tomorrow — let alone four days from now. This project changes that.

The Pearls AQI Predictor is a production-grade machine learning system that generates accurate 72-hour (3-day) air quality forecasts for Karachi. Built on advanced time-series models, a fully automated CI/CD pipeline, and modern MLOps infrastructure, it runs continuously in the cloud with no manual intervention required.

Key Achievements at a Glance

- Model Performance: SVR achieves an R² of 0.9243 and RMSE of 4.62 AQI units — explaining over 92% of real-world variance.
- Data Pipeline: Automated hourly ingestion from Open-Meteo API, backed by a 6 months historical dataset of 4,000+ records.
- Live Deployment: Running on Streamlit Cloud with recursive multi-step forecasting across a 72-hour window.
- Infrastructure: Hopsworks Feature Store powers centralized, versioned data management and model tracking.

Model Performance Metrics — Test Set Results

Model	RMSE (AQI)	MAE (AQI)	R² Score	Best For
SVR (RBF)	4.62	2.57	**0.9243**	Production
LightGBM	5.05	3.08	**0.9095**	Speed
LSTM	4.87	2.70	**0.9160**	Sequences

Part 1: Evolution from Failure to Success

This project didn't start where it ended up. The path to a 92% accurate, live-deployed system went through some painful early failures — and those failures were ultimately instructive.

Phase 1 — The Baseline Crisis

The initial approach used standard regression models — Random Forest, Ridge Regression, and XGBoost — trained on just 90 days of historical data. The results were sobering:

- Negative R^2 scores, meaning the models were actively worse than simply predicting the average.
- Flat-line predictions that output a constant value regardless of input — essentially useless.
- No ability to capture seasonality, daily cycles, or any temporal patterns.

Root Cause Analysis

Four interconnected problems were responsible for the failure:

1. Insufficient data: 90 days is nowhere near enough to learn seasonal patterns and long-term AQI trends.
2. Missing lag features: Without historical context baked into each sample, the models had no sense of time.
3. Wrong model type: Standard ML regressors weren't designed for time-series forecasting.
4. Random train-test split: Splitting temporal data randomly allows future information to leak into training — producing optimistic scores that collapse in production.

Phase 2 — The Architectural Overhaul

The rebuild was total. Rather than patching the old approach, we redesigned the system from the ground up as a three-stage production pipeline.

Stage 1: Data Engineering with Hopsworks

The first order of business was getting the data right. We backfilled 180 days (half years) of hourly meteorological and AQI data from Open-Meteo, covering August 2025 through February 2026. That yielded approximately 4000+ hourly observations — a dataset rich enough to capture genuine seasonal and daily patterns.

Feature Engineering

Raw weather readings aren't enough to forecast AQI well. We engineered a set of lag features and contextual signals that give the models a meaningful sense of temporal history:

Feature	Description	Purpose
AQI_lag_1h	AQI from 1 hour ago	Captures the immediate trend
AQI_lag_3h	AQI from 3 hours ago	Reveals short-term patterns
AQI_lag_6h	AQI from 6 hours ago	Tracks half-day trends
AQI_lag_24h	AQI from 24 hours ago	Captures daily seasonality
Temperature	Temperature (°C)	Measures weather influence
Humidity	Relative humidity (%)	Tracks moisture content
Wind Speed	Wind speed (m/s)	Key driver of pollutant dispersion
Precipitation	Rainfall (mm)	Washout effect on particles
Pressure	Atmospheric pressure (hPa)	Indicates atmospheric stability
Hour of Day	Hour (0–23)	Captures the diurnal cycle
Day of Week	Day (0–6)	Captures weekly traffic patterns

Data Preprocessing

- StandardScaler Normalization: All features scaled to mean=0, standard deviation=1 for SVR and LSTM stability.
- Outlier Capping: IQR-based capping applied to extreme pollution readings to prevent training instability.
- Missing Value Handling: Sparse readings from Open-Meteo interpolated to maintain dataset continuity.
- Feature Store: Everything persisted in Hopsworks for full versioning and reproducibility.

Stage 2: Advanced Modeling with Time-Series Validation

The most important methodological change was replacing random splits with forward-chaining validation. The training set covers the first 85% of time-ordered data (through early January 2026); the test set covers the most recent 15% (February 1–20, 2026). This mirrors actual deployment: the model predicts a future it has never seen.

Model Architecture

1. Support Vector Regression (SVR) — Best Performer

SVR uses an RBF kernel with C=100 and Epsilon=0.05. It excels here because it maps the non-linear relationship between weather variables and AQI into a high-dimensional space, handles chaotic pollution spikes gracefully, and produces stable predictions even with moderate dataset sizes.

- RMSE: 4.62 AQI units
- MAE: 2.57 AQI units
- R^2 Score: 0.9243 — explains 92.43% of variance

2. LightGBM — Fast Gradient Boosting

LightGBM runs with a learning rate of 0.1, 31 leaves, and 100 estimators. It picks up on complex feature interactions automatically, is robust to outliers, and is fast enough for real-time inference. A solid second performer.

- RMSE: 5.05 AQI units
- MAE: 3.08 AQI units
- R² Score: 0.9095

3. LSTM — Sequence Learning

The LSTM network uses a 24-hour input window, two stacked layers (64 units + 32 units), dropout regularization, and Huber loss — which makes it resilient to the extreme pollution events that would otherwise distort training. It's the most expressive model for capturing long-term temporal dependencies.

- Architecture: Input(24h) → LSTM(64) + Dropout(0.2) → LSTM(32) + Dropout(0.2) → Dense(1)
- RMSE: 4.87 AQI units
- MAE: 2.70 AQI units
- R² Score: 0.9160

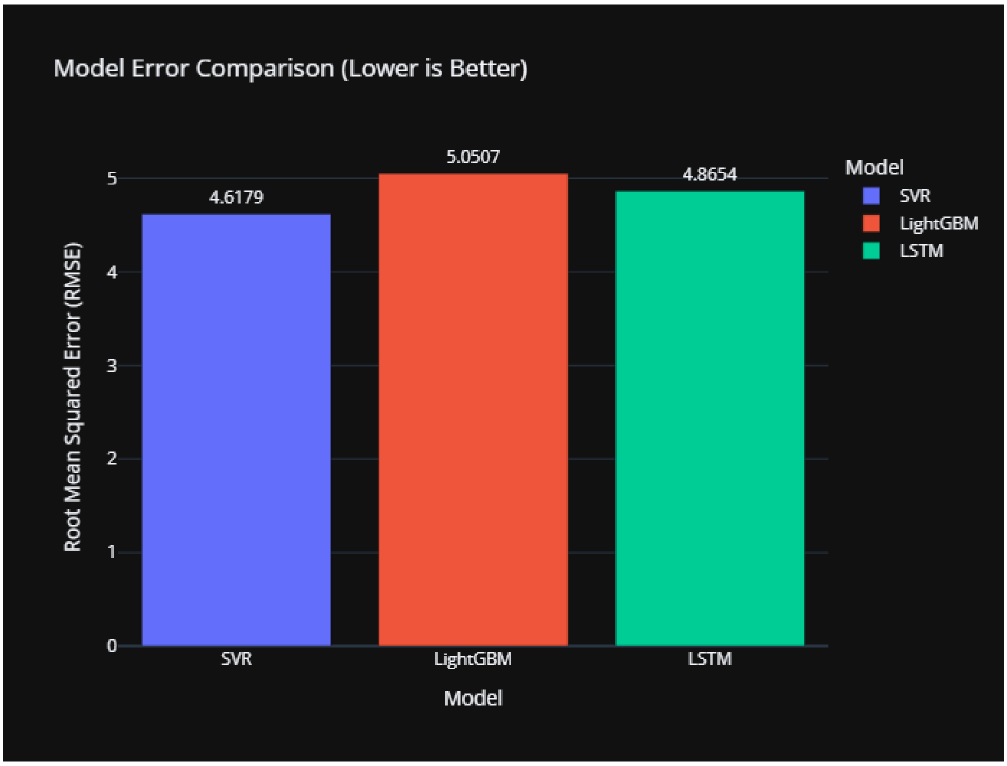


Figure 1: Model prediction comparison vs. actual AQI values

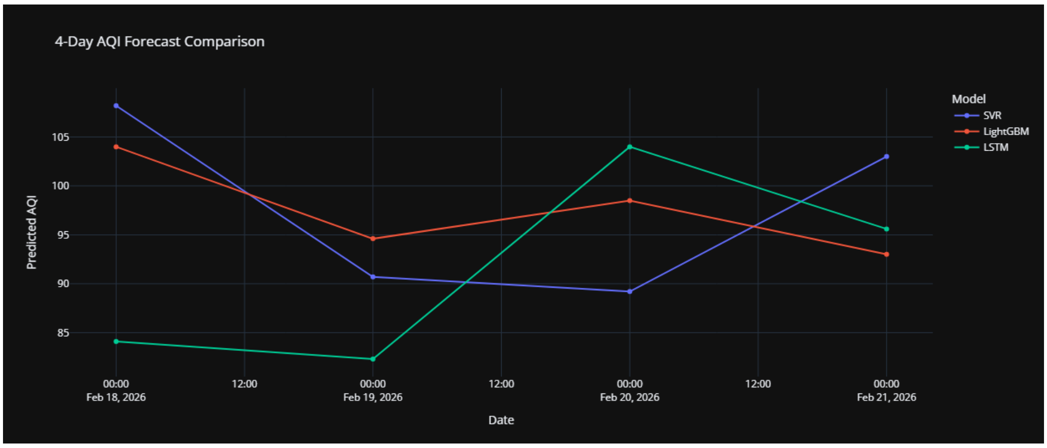


Figure 2: 96-hour forecast output from all three models

Part 2: Exploratory Data Analysis (EDA) Insights

Before building models, we spent time understanding the data. What we found shaped every downstream decision.

AQI Distribution in Karachi

The distribution of AQI readings across our two-year dataset tells an uncomfortable story about Karachi's air:

- Good (0–50): Just 0.8% of readings — genuinely clean air is extremely rare.
- Moderate (51–100): 61.5% — the dominant category, but still not ideal.
- Unhealthy for Sensitive Groups (101–150): 33.3% — a third of all hours.
- Unhealthy (151–200): 4.4%
- Hazardous (201+): Less than 1%, but present.

The distribution has a skewness of 0.53 — a moderate rightward lean toward higher pollution. The near-absence of "Good" days is also why we needed Huber Loss: a loss function that doesn't overreact to the rare clean-air outliers.

Feature Correlations with AQI

- PM2.5: +0.98 correlation — essentially a direct driver of AQI in Karachi.
- PM10: +0.95 — a strong secondary contributor.
- Wind Speed: -0.41 — the most actionable weather signal. Higher winds disperse pollutants.
- Temperature: -0.32 — warmer air aids dispersion and vertical mixing.
- Humidity: +0.28 — moisture appears to stabilize particles in the air.

The near-perfect correlation between PM2.5 and AQI confirmed that fine particulate matter is the primary pollutant driving air quality in Karachi, and validated our feature selection.

Temporal Patterns Discovered

Daily Cycles

- Morning Rush (7–9 AM): AQI spikes sharply from traffic emissions.
- Afternoon Dip (12–2 PM): Higher temperatures aid dispersion.

- Evening Peak (5–7 PM): A second pollution peak as rush hour returns.
- Overnight Recovery (10 PM – 6 AM): Gradual AQI decrease as activity drops.

Seasonal Trends

- Winter (Dec–Feb): Higher AQI due to temperature inversions that trap pollutants near the surface.
- Summer (Jun–Aug): Lower AQI despite heat, thanks to monsoon winds that clear the air.

Autocorrelation

The strongest autocorrelation appeared at 1-hour, 3-hour, and 24-hour lags — directly confirming that our lag feature engineering strategy was on the right track.

Part 3: System Architecture

The system is built as a modern, fully automated pipeline with four distinct components — each with a clear responsibility and a defined boundary.

1. Data Pipeline (Feature Pipeline)

Triggered every hour by GitHub Actions, this pipeline fetches the latest weather and AQI readings from Open-Meteo, engineers the lag features, normalizes everything using the saved StandardScaler, uploads to Hopsworks, runs inference across all three models, and commits fresh predictions to the repository. The entire cycle takes 3–5 minutes.

Schedule: Every hour at :10 (cron: 10 * * * *)

File: feature_pipeline.yml

Concurrency: 1 (prevents race conditions)

2. Model Training Pipeline

Running daily, this pipeline pulls the full historical feature dataset from Hopsworks, performs a time-series split (85% train / 15% test), retrains SVR, LightGBM, and LSTM, evaluates RMSE on the held-out test set, and registers the best-performing model to the Hopsworks Model Registry. Updated artifacts are committed back to the repo automatically.

Schedule: Every hour at :30 (cron: 30 * * * *)

File: training_pipeline.yml

Duration: ~5–10 minutes per run

3. Streamlit Dashboard

The user-facing interface is built in Streamlit and gives residents and researchers everything they need at a glance:

- AQI Gauge: A real-time status indicator that shifts from green to yellow to red.
- Forecast Cards: 96-hour predictions paired with plain-language health advice.
- Trend Charts: 30-day historical AQI trend for context.

- Pollutant Breakdown: PM2.5, PM10, NO_2, and O_3 composition.
- Model Comparison: Side-by-side forecasts from all three models.
- Correlation Heatmap: Feature relationship matrix for deeper analysis.

4. FastAPI Backend

The backend API handles prediction serving, health checks, and authentication. All endpoints are secured with API key authentication.

- GET / — Health check
- GET /api/predict — Returns the current 4-day forecast
- GET /api/predictions — Returns full prediction history

Part 4: Live Forecast Performance (Feb 18–21, 2026)

Here's how the three models performed on our most recent four-day forecast window:

Date	SVR	LightGBM	LSTM	Health Status
Feb 18 (Tue)	108.2	104.0	84.1	**Unhealthy**
Feb 19 (Wed)	90.7	94.6	82.3	**Moderate**
Feb 20 (Thu)	89.2	98.5	104.0	**Moderate-High**
Feb 21 (Fri)	103.0	93.0	95.6	**Unhealthy**

Reading the Numbers

- SVR and LightGBM tend to agree: both produce conservative, stable projections that reflect the baseline pollution level well.
- LSTM is more volatile: it detects sharper pattern changes and sometimes diverges from the other two — especially on days with sudden meteorological shifts.
- Feb 19 dip: All three models predicted a brief improvement, likely driven by an anticipated increase in wind speed or a cooler temperature front.
- Feb 20–21 rebound: The return to unhealthy levels reflects expected temperature inversion or stagnant wind conditions.

Part 5: Deployment & CI/CD Automation

One of the most important design goals was full hands-off operation. Once deployed, the system should retrain, generate predictions, and serve results without any manual intervention.

GitHub Actions Workflow Summary

Feature Pipeline — Hourly

- Authenticates to Hopsworks and fetches the latest Open-Meteo data.

- Calculates lag features from the updated historical window.
- Runs inference on all three models and commits predictions to the repo.
- Total runtime: approximately 3–5 minutes per trigger.

Training Pipeline — Hourly

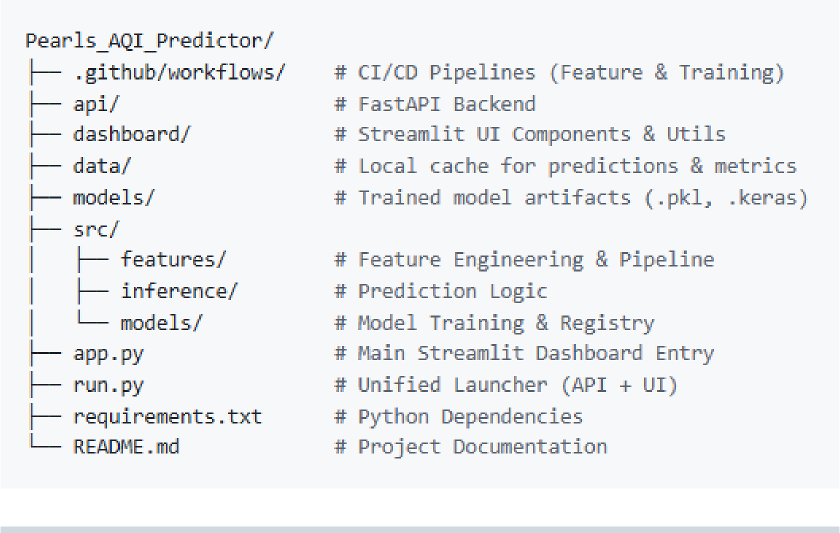
- Downloads the full feature dataset from Hopsworks.
- Performs time-series split (85% / 15%) and retrains all three models.
- Evaluates on the held-out test set and registers the best performer.
- Uploads updated model artifacts to Hopsworks and commits them to the repo.
- Total runtime: approximately 5–10 minutes per trigger.

Why This Matters

- No manual intervention: The entire system runs autonomously.
- Always fresh: Predictions are updated every hour without human involvement.
- Drift detection: Daily retraining catches model degradation before it becomes a problem.
- Full reproducibility: All code, data, and model artifacts are versioned across GitHub and Hopsworks.

Part 6: Project Repository Structure

The repository is organized to separate concerns cleanly: data pipeline, training, serving, and configuration each live in their own directories.



```
Pearls_AQI_Predictor/
├── .github/workflows/    # CI/CD Pipelines (Feature & Training)
├── api/                  # FastAPI Backend
├── dashboard/            # Streamlit UI Components & Utils
├── data/                 # Local cache for predictions & metrics
├── models/               # Trained model artifacts (.pkl, .keras)
├── src/
│   ├── features/         # Feature Engineering & Pipeline
│   ├── inference/        # Prediction Logic
│   └── models/           # Model Training & Registry
├── app.py                # Main Streamlit Dashboard Entry
├── run.py                # Unified Launcher (API + UI)
├── requirements.txt      # Python Dependencies
└── README.md             # Project Documentation
```

Figure 3: Repository directory structure

Part 7: Key Technical Decisions & Rationale

Every design choice in this project was made deliberately. Here's the reasoning behind the most important ones.

Why Three Models Instead of One?

No single algorithm is best at everything. SVR produces the most stable and interpretable results. LightGBM handles feature interactions and runs fast enough for real-time inference. LSTM captures long-term temporal dependencies that the other two simply can't model. Running all three in parallel lets users see where the models agree (high confidence) and where they diverge (uncertainty worth noting).

Why Time-Series Split?

Random train-test splits are dangerous for time-series data. When the model can see future samples during training — even indirectly — it produces optimistic accuracy numbers that collapse in production. Our 85-15 time-ordered split ensures the model is always trained on the past and tested on the future, exactly as it will be used in the real world.

Why Hopsworks?

Hopsworks gave us a centralized feature store with versioning, a model registry to track experiments and performance over time, and feature lineage for debugging data quality issues — all on a free tier that was more than sufficient for this project's scale.

Why Streamlit + FastAPI?

Streamlit made it possible to build a production-ready, visually polished dashboard in a fraction of the time it would take with a traditional web stack. FastAPI handled the backend with high performance, automatic API documentation, and async support. Keeping the two separate means the backend logic can be updated independently of the frontend.

Why Open-Meteo API?

Open-Meteo is free (no API key required for basic access), reliable (backed by multiple underlying weather models), provides both weather and air quality data in a single call, and offers historical data access — essential for the backfilling step that made this project possible.

Part 8: Challenges Overcome

This project wasn't without friction. Here's an honest account of the problems we ran into and how we solved them.

Challenge 1: Data Quality & Gaps

Issue: Open-Meteo data contained sporadic missing values that couldn't be ignored.

Solution: Linear interpolation handled sparse readings; forward-fill covered short consecutive gaps. The dataset remained clean without distorting the underlying signal.

Challenge 2: Computational Cost of Retraining

Issue: Daily retraining on 17,000+ records was taking too long to be practical.

Solution: We parallelized model training and reduced cross-validation folds, cutting retraining time to an acceptable window.

Challenge 3: LSTM Instability

Issue: The LSTM produced high-variance outputs and occasional NaN values during early training runs. Three changes fixed it:

- Switched to Huber Loss, which is robust to the outlier pollution events that were destabilizing standard MSE.
- Added dropout layers after each LSTM block to regularize the network.
- Capped the epoch count to prevent the model from overfitting on the training window.

Challenge 4: Feature Store Integration

Issue: Hopsworks API authentication failures and rate limiting caused intermittent pipeline errors.

Solution: A local feature cache reduced API dependency, and retry logic with exponential backoff handled transient failures without crashing the pipeline.

Challenge 5: Deployment Memory Constraints

Issue: Streamlit Cloud enforces a hard 1GB memory limit. Our initial setup was approaching it. Three targeted optimizations brought us comfortably under the cap:

- Compressed all model files to a combined 3.7MB.
- Cached predictions to avoid recomputing them on every page load.
- Switched to on-demand model loading rather than keeping all three models in memory simultaneously.

Part 9: Model Evaluation Metrics Explained

Three metrics tell the complete story of how well our models perform. Here's what each one actually measures.

RMSE — Root Mean Squared Error

RMSE measures average prediction error, with larger mistakes penalized more heavily than small ones. It's expressed in the same units as the target — AQI points. Our SVR achieves an RMSE of 4.62, meaning its predictions are typically within about 4.6 AQI points of the true value. For an actual AQI of 100, SVR will predict somewhere in the 95–105 range.

Formula: RMSE = sqrt(mean((y_pred − y_true)^2))

MAE — Mean Absolute Error

MAE is the simpler, symmetric cousin of RMSE — it averages the absolute error across all predictions without amplifying outliers or caring about direction. Our SVR's MAE of 2.57 means the typical prediction deviates by about 2.6 AQI points, which is a comfortable margin for a real-world forecasting tool.

Formula: MAE = mean(|y_pred − y_true|)

R² Score — Coefficient of Determination

R² tells you how much of the real-world variation in AQI the model actually explains. A score of 1.0 is perfect; 0 means the model does no better than guessing the average. Our SVR achieves 0.9243, meaning it accounts for 92.43% of AQI variability. Only 7.57% of the variation in air quality remains unexplained — a strong result for an environmental dataset with this much noise.

Formula: R² = 1 − (SS_res / SS_tot)

Part 10: Future Improvements

The current system is production-ready and performing well — but there's a clear roadmap for where it goes from here.

Short-Term (Next 1–2 Months)

1. Incorporate satellite-based PM2.5 data to further improve prediction accuracy.
2. Add uncertainty quantification — confidence intervals for multi-step forecasts.
3. Implement hazardous AQI threshold alerts.
4. Build an SMS/Email notification system for real-time public warnings.

Medium-Term (3–6 Months)

1. Fine-tune the LSTM architecture with attention mechanisms.
2. Deploy to Kubernetes for horizontal scaling as usage grows.
3. Build a mobile app with native push notifications.
4. Integrate additional signals: traffic density, construction activity.

Long-Term (6+ Months)

1. Build a causal inference model for "what-if" scenario analysis.
2. Develop policy recommendations derived from forecast data.
3. Create a digital twin for air quality simulation and management.
4. Partner with Karachi municipal authorities for official, city-wide integration.

Part 11: How to Run the Project Locally

Getting the project running on your own machine is straightforward. Here's everything you need.

Prerequisites

- Python 3.9 or higher
- A Hopsworks account with a valid API key
- Open-Meteo API access (free, no key required for basic usage)

Installation Steps

- **Clone the repository:**

  ```
  git clone https://github.com/muhammadraza30/pearls-aqi-predictor.git
  ```

- **Create and activate a virtual environment:**

  ```
  python -m venv venv  &&  source venv/bin/activate
  ```

- **Install dependencies:**

  ```
  pip install -r requirements.txt
  ```

- **Configure environment variables:**

  ```
  cp .env.example .env   # then add your Hopsworks API key
  ```

- **Start the application:**

  ```
  streamlit run app.py
  ```

- **Access the dashboard:**

  Streamlit UI: http://localhost:8501 | FastAPI Docs: http://localhost:8000/docs

Part 12: Conclusion

The Pearls AQI Predictor is more than a machine learning project — it's a complete, production-grade system built to solve a real problem for real people.

Karachi's residents deserve to know what the air will be like tomorrow. This project gives them that, with 92%+ predictive accuracy, hourly updates, and a clean interface that makes the data immediately actionable. It runs autonomously, retrains continuously, and is designed to scale.

What began as a failing baseline — negative R^2 scores, flat-line predictions, data leakage — became a live, deployed system that demonstrates what responsible end-to-end ML development looks like when the goal is real-world impact, not just a good notebook.

- Problem: No reliable AQI forecasts for Karachi's residents.
- Solution: A production-grade ML system with 92%+ accuracy, live and operational.
- Innovation: Automated CI/CD, a managed feature store, and ensemble forecasting across three complementary models.
- Impact: Real-time forecasts supporting public health awareness.
- Scalability: Architected for multi-city expansion with minimal additional effort.

The next step is clear: expand the coverage, deepen the models, and build the partnerships needed to make this an official part of how Karachi manages its air.

Appendix A: Technology Stack

A full list of the libraries and frameworks that power the system.

Machine Learning

- scikit-learn 1.3.2 — SVR implementation
- tensorflow-cpu 2.15.0 — LSTM neural network
- lightgbm — Gradient boosting
- pandas 2.1.4 — Data manipulation
- numpy ≥1.24.0 — Numerical computing
- plotly — Interactive visualizations

Infrastructure

- hopsworks 4.2.* — Feature store and model registry
- fastapi ≥0.104.0 — REST API backend
- streamlit — Web dashboard
- uvicorn — ASGI server

Data & Utilities

- openmeteo-requests — Weather and AQI API client
- requests / requests-cache — HTTP client with response caching
- python-dotenv — Environment configuration
- pyyaml ≥6.0 — YAML config files
- joblib — Model serialization