

CAPSTONE PROJECT

Integrated Air Quality Monitoring: From Spatial Estimation to Edge Deployment

Samir Shabani

Department of Computer Science
South East European University (SEEU)
Tetovo, North Macedonia
s.shabani@seeu.edu.mk

Mentor: Prof. Dr. Mennan Selimi

Abstract

Urban air quality monitoring presents a compound engineering challenge: the sensors that produce data need communication infrastructure to transmit it, the communication infrastructure needs to be planned using knowledge of the environment, and the data itself needs to be processed and stored reliably without depending on cloud connectivity. These three problems are usually treated in isolation. This thesis treats them together.

The first contribution is a hybrid PM2.5 estimation model. An XGBoost model learns the relationship between meteorological inputs and measured pollution levels; a second layer of physics-based corrections adds spatial structure through distance-decay terms for road proximity, industrial zone influence, vegetation attenuation, and terrain elevation. Neither component alone produces spatially structured pollution maps-the combination does.

The second contribution is a GPU-accelerated LoRaWAN network planning system built on a 3D digital twin derived from public DSM and DTM raster data. Building heights are computed per-pixel and voxelized for line-of-sight analysis. A Monte Carlo propagation model estimates link reliability across the city grid. Gateway placement is then solved as a multi-objective optimization using NSGA-II, producing Pareto-optimal configurations that trade coverage against average spreading factor-giving network operators a concrete set of deployment options rather than a single opaque answer.

The third contribution is a benchmarked IoT pipeline on a Raspberry Pi 4 cluster, tested across 1-, 2-, 3-, and 5-node configurations. The stack uses K3s, MetalLB, EMQX, NATS JetStream, Benthos, VictoriaMetrics, and Longhorn, provisioned via Ansible and kept in sync through ArgoCD GitOps. A 5-node cluster achieves 92-100% of target throughput with zero message drops across 184 runs over 62 days. The 1-node baseline reached 4-48% efficiency with 77-95% drop rates. Four engineering fixes-removing an unbounded metric label, correcting publisher timing drift, fixing a DNS network policy, and aligning K3s versions-plus a switch from MQTT QoS 2 to QoS 0 account for most of the improvement. An accidental 2-node run with QoS 0 and the corrected pipeline already hit near-perfect efficiency, which confirms that configuration was the binding constraint, not node count.

The fourth contribution is a CNN-BiGRU model for multi-day PM2.5 forecasting, combining a causal 1D convolutional block with two stacked bidirectional BiGRU layers. Trained on MOEPP reference data with 234 engineered features, it achieves $R^2 = 0.87$ on the test set. When the same architecture is trained on low-cost sensor data, predictions are systematically too high-traced to humidity-induced overestimation in the optical particle counters, not the model. The fix is humidity correction before training, not a different architecture.

The same pattern appears across all four contributions: the binding constraint is almost never the hardware. It is a protocol choice, a label cardinality bug, an uncalibrated sensor, or a missing spatial correction. Getting those right matters more than adding nodes or depth.

Contents

1 Introduction **12**
- 1.1 The Urban Air Quality Problem . . . 12
- 1.2 The Infrastructure Gap . . . 12
- 1.3 This Work . . . 13
- 1.4 Objectives . . . 13
- 1.5 Contributions . . . 14
- 1.6 Thesis Structure . . . 14

2 Related Work **15**
- 2.1 Urban Air Quality Estimation . . . 15
- 2.2 LoRaWAN Network Planning . . . 16
- 2.3 Edge Computing and Cloud-Native IoT . . . 16
- 2.4 Deep Learning for Air Quality Forecasting . . . 17

3 System Architecture **18**
- 3.1 Overview . . . 18
- 3.2 Design Principles . . . 20

4 Air Quality Modeling **21**
- 4.1 Problem Formulation . . . 21
- 4.2 Machine Learning Baseline: XGBoost . . . 21
- 4.3 Spatial Correction Layer . . . 22
- 4.4 Calibration and Validation . . . 25

5 Digital Twin and Urban Modeling **25**
- 5.1 Motivation and Approach . . . 25
- 5.2 DSM/DTM Processing . . . 26
- 5.3 Line-of-Sight Computation . . . 27

6 Wireless Network Modeling **28**
- 6.1 Path Loss Model . . . 28
- 6.2 Spreading Factor Assignment . . . 28
- 6.3 Monte Carlo Reliability Estimation . . . 29

7 Network Optimization **30**
- 7.1 Problem Formulation . . . 30
- 7.2 NSGA-II Optimizer . . . 31
- 7.3 Optimized Deployment Configurations . . . 32
- 7.4 Field Validation . . . 33
  - 7.4.1 Hardware and Setup . . . 34
  - 7.4.2 Results and Interpretation . . . 34

8 Edge Computing Pipeline **35**
8.1 Hardware Configuration . . . 35
8.2 Orchestration: K3s and MetalLB . . . 36
8.3 MQTT Ingestion: EMQX . . . 36
8.4 Event Streaming: NATS JetStream . . . 37
8.5 Transformation: Benthos . . . 38
8.6 Time-Series Storage: VictoriaMetrics . . . 38
8.7 Distributed Storage: Longhorn . . . 39
8.8 Infrastructure Automation: Ansible . . . 39
8.9 GitOps with ArgoCD . . . 40
8.10 Observability and Resource Budget . . . 40

9 Experimental Evaluation **41**
9.1 Benchmark Design . . . 41
9.1.1 Incremental Engineering Fixes . . . 43
9.2 Benchmark Campaign and Timeline . . . 43
9.3 Throughput Results . . . 44
9.4 Latency Results . . . 45
9.5 Bottleneck Analysis . . . 47
9.6 Run Variability, Resource Usage, and Storage Endurance . . . 49

10 PM2.5 Forecasting **50**
10.1 Motivation and Problem Formulation . . . 50
10.2 Dataset . . . 50
10.2.1 MOEPP Air Quality Data . . . 50
10.2.2 Meteorological Data . . . 52
10.3 Data Pre-processing . . . 54
10.3.1 Missing Value Imputation . . . 54
10.3.2 Merging and Alignment . . . 56
10.4 Feature Engineering . . . 57
10.4.1 Temporal Encoding . . . 58
10.4.2 Lag Features . . . 58
10.4.3 Rolling Statistics . . . 58
10.4.4 Sequence Construction . . . 60
10.5 Data Splitting and Normalization . . . 61
10.6 Model Architecture: CNN-BiGRU . . . 63
10.6.1 Design Rationale . . . 63
10.6.2 Convolutional Block . . . 63
10.6.3 Bidirectional BiGRU Layers . . . 64
10.6.4 Dense Output Layers . . . 64
10.6.5 Federated Variant Architecture . . . 66
10.7 Training . . . 67
10.8 Federated Learning Variant . . . 67

10.9 Evaluation Metrics . . . 68
10.10 Results . . . 68
10.10.1 Feature Correlation Analysis . . . 68
10.10.2 Model Performance on MOEPP Test Set . . . 70
10.10.3 Prediction Analysis: MOEPP Test Set . . . 71
10.10.4 Comparison: Own Sensor Network . . . 72
10.11 Discussion of Forecasting Results . . . 73

11 Discussion **74**
11.1 The QoS Level Is the Architectural Decision . . . 74
11.2 What Dominates Latency . . . 75
11.3 Cloud-Native Overhead Is Acceptable . . . 75
11.4 Longhorn Replication and Fault Tolerance . . . 76
11.5 Scaling the System . . . 76
11.6 GitOps Made the Experiment Reproducible . . . 77
11.7 Practical Design Guidance . . . 77

12 Fault Tolerance **78**
12.1 Architecture-Level Fault Containment . . . 78
12.2 Known Limitations . . . 79

13 Limitations and Future Work **79**
13.1 Limitations of the Air Quality Model . . . 79
13.2 Limitations of the LoRaWAN Planner . . . 79
13.3 Limitations of the Edge Pipeline . . . 80
13.4 Future Work . . . 80

14 Conclusion **82**

List of Figures

1 End-to-end system architecture. The air quality model (left) estimates the pollution field and guides sensor placement. The LoRaWAN planner (center) determines gateway positions using the digital twin for propagation modeling. The edge pipeline (right) ingests and stores sensor data continuously. Arrows indicate data flow; the three layers are loosely coupled through documented interfaces. 19

2 Cloud-native IoT pipeline deployed on the Raspberry Pi cluster. IoT sensors connect to EMQX via MQTT over the LoRaWAN gateway. Accepted messages are forwarded to NATS JetStream for durable queuing. Benthos subscribes to NATS, validates and transforms messages, and writes to VictoriaMetrics. Longhorn replicates stateful volumes (EMQX, Loki, Tempo) across nodes, while VictoriaMetrics and NATS use local-path storage for maximum write throughput. The entire stack is orchestrated by K3s, with MetalLB providing load-balanced external access and ArgoCD reconciling cluster state from Git. 20

3 Feature importances from the XGBoost model, expressed as mean split gain normalized to the maximum. Meteorological variables and the heating season indicator dominate. Temporal features (hour, month) capture systematic diurnal and seasonal patterns in emission activity. 22

4 Effect of the road correction term. The pollution field acquires spatial structure corresponding to the road network geometry. The decay length scale λ_r determines the width of elevated-PM corridors. 24

5 Effect of industrial and vegetation correction terms. Both corrections are spatially localized and physically interpretable: industrial zones increase PM; vegetation decreases it. 25

6 Distribution of building heights derived from DSM/DTM subtraction across the study area. The dominant mode at 8-12 m corresponds to three- to four-story residential construction, which is the prevailing typology in Tetovo's urban core. The secondary mode at 18-22 m corresponds to larger apartment blocks and institutional buildings. Values above 30 m are rare and correspond to industrial structures. 27

7 Per-link communication reliability across the study grid, estimated via Monte Carlo simulation over the digital twin geometry. Reliability is non-uniform: elevated terrain near the gateway achieves high reliability due to clear LOS, while dense building clusters in the center produce shadow zones with substantially reduced reliability. These patterns are invisible in a flat-earth coverage model. 30

8 Pareto fronts for different gateway counts after 2000 generations with a population of 5000. Each point represents a non-dominated gateway placement configuration. The front shifts toward higher coverage and lower average SF as the gateway count increases. The shape of the front indicates the marginal cost in average SF of extending coverage to the final few percent of devices. 32
9 Final optimized gateway positions overlaid on the urban grid. Gateway positions cluster at elevated terrain and building rooftops that offer wide-area line-of-sight. With 10 gateways, the optimizer achieves near-complete coverage of the device grid while maintaining low average SF. 32
10 Optimized network topology for the 10-gateway configuration achieving 99.9% device coverage. Devices are colored by spreading factor assignment: SF7 (closest devices, highest data rate) through SF12 (devices at maximum range). The spatial structure reflects the underlying propagation geometry: devices in building-dense areas require higher SFs even at moderate distances. 33
11 PM2.5 and PM_{10} daily concentrations from the MOEPP reference station in Tetovo. Winter peaks are driven by the combination of residential wood burning and valley-confined thermal inversions. The PM2.5/PM_{10} ratio is consistently high, indicating fine-mode combustion aerosol rather than coarse crustal dust. 51
12 Time series of gaseous co-pollutants from the MOEPP dataset. O_3 peaks in summer; NO_2, SO_2, and CO peak in winter alongside PM2.5. These species provide complementary information about atmospheric conditions and emission sources that the model can exploit. 52
13 Raw hourly weather observations from the Meteostat station nearest to Tetovo: temperature, relative humidity, precipitation, wind speed, and atmospheric pressure. Hourly data is aggregated to daily summaries before merging with the air quality dataset. 53
14 Distributions of key hourly weather features. Temperature spans roughly -10 to $35°$C; humidity is bimodal with modes near 50% and 85%; wind speed is right-skewed with most hours below 10 km/h, consistent with the sheltered valley location. 54
15 Zoomed view of the imputation effect on all six pollutant species. Red dashed lines show the raw data with gaps; blue lines show the imputed series; green crosses mark individually interpolated points. The interpolation preserves local trends without introducing step discontinuities. 55
16 Full-range imputation effect across all pollutant species. The imputed series (blue) matches the raw data (red) where observations are available and interpolates smoothly across gaps. No spurious trends are introduced. 56

17 Impact of the inner join merge on date coverage. Orange lines mark dates present only in the air quality dataset; green lines mark dates present only in the weather dataset; blue lines mark dates retained in the merged dataset. The inner join is conservative: it discards any date without complete information from both sources. 57
18 PM2.5 and mean temperature after merging. The anti-phase relationship is clear: PM2.5 peaks in winter when temperature is at its minimum. This is the primary signal the model must learn, and its presence in the merged dataset confirms the merge is correctly aligned. 57
19 Examples of engineered features. Top: original PM2.5 and its 1-day lag (the lag shifts the series rightward by one day, providing the model with yesterday's reading as an explicit input). Second: mean temperature with its 3-day rolling mean, which smooths local noise while preserving the trend. Third and fourth: cyclical day-of-year and day-of-week sine-cosine pairs, which encode periodicity without imposing ordinal distances. 59
20 Effect of the initial row drop after feature engineering. The grey shaded region is removed from the dataset. This loss is small relative to the dataset length and unavoidable given the lag structure. 60
21 Illustration of the sliding window sequence construction. The purple shaded region is the 14-day input window for one training sample; the red dashed vertical line marks the day being predicted. The window slides forward by one day to produce the next sample. 61
22 Chronological train-validation-test split on the PM2.5 series. The training set covers the majority of the heating seasons in the dataset; the test set covers August 2024 to April 2025, spanning one full seasonal cycle from summer baseline through winter peak and back. 62
23 PM2.5 distribution before and after `StandardScaler` on the training set. The raw distribution is right-skewed due to winter peaks; the scaled distribution has mean 0 and standard deviation 1 but retains the same shape. 62
24 PM_{10} distribution before and after scaling. The strong positive correlation between PM_{10} and PM2.5 means both distributions share a similar skewed shape. 63
25 Architecture of the CNN-BiGRU model for PM2.5 forecasting. The input sequence passes through a causal convolutional block (batch norm, max pool, dropout), two stacked bidirectional BiGRU blocks, and finally two dense layers before the scalar output. 65
26 Architecture of the CNN-BiLSTM-AM model for PM2.5 forecasting. The input sequence passes through a causal convolutional block (batch norm, max pool, dropout), two stacked bidirectional LSTM blocks with additive attention, and finally two dense layers before the scalar output. 67

27 Pearson correlation matrix for PM2.5, co-pollutants, and meteorological variables. Values range from $+1$ (dark red, perfect positive correlation) to -1 (dark blue, perfect negative correlation). Strong inter-feature correlations confirm that the selected variables carry substantial joint predictive information about PM2.5. 69
28 CNN-BiGRU model predictions (red dashed) versus actual PM2.5 (blue) on the MOEPP test set, covering August 2024 to April 2025. The model correctly captures the summer-to-winter seasonal transition, tracks the winter peak timing, and follows the spring decline. Predicted values are slightly smoother than actuals, which is characteristic of MSE-trained regressors. 71
29 CNN-BiGRU predictions on the centralized own-sensor dataset (14-day horizon). Predicted values are systematically elevated relative to actuals. The model has learned to predict high because that is what the training data predominantly shows, due to humidity-induced overestimation in the optical particle counters. 72
30 Federated learning variant predictions on the own-sensor dataset (7-day horizon). The systematic upward bias is similar to the centralized model, confirming that FedAvg aggregation does not correct for measurement errors that are common across all clients. 73

List of Tables

1 LoRaWAN spreading factor parameters at 125 kHz bandwidth for a 20-byte payload. Higher spreading factors extend range at the cost of lower data rate and longer time on air. 29
2 Simulation versus field measurements on the SEEU campus (Building 816 $\leftrightarrow$ Building 101 / Rectorate, $d \approx 350\,\mathrm{m}$, near-LOS geometry). 34
3 Hardware and software configuration for each cluster node. All nodes are identical; the K3s version was aligned to v1.35.4+k3s1 across all nodes in the five-node configuration (Section 9.1.1). 36
4 Pod placement in the 5-node configuration. The master carries all stateful pipeline components; workers host consumer replicas and monitoring agents. 41
5 Memory usage by major pod during 5-node benchmarks. The observability stack (Grafana, Prometheus, OpenTelemetry Collector) consumes more RAM than the data pipeline. 41
6 Benchmark scenarios. The 1000 and 2000 msg/s scenarios were only achievable on the 5-node cluster; lower node counts saturate before reaching these targets. 42
7 Cluster configurations tested, in order. Changes at each step are cumulative; everything not listed carried over from the previous configuration. 42
8 Benchmark timeline by phase over the 62-day campaign (184 runs total). . . 44
9 Achieved throughput (msg/s) across cluster sizes and scenarios. The 5-node cluster sustains 92-100% of target across all eight scenarios; the 1-node cluster reaches at most 4.7-22.9% efficiency. 44
10 Pipeline efficiency as percentage of target rate. The 5-node cluster maintains above 92% efficiency across all scenarios, including 2000 msg/s under 100 concurrent clients. 45
11 End-to-end latency for 1- and 2-node configurations. The 1-node cluster under high load accumulates latency in the tens of minutes due to message queuing; the 2-node cluster reduces this to sub-second for most scenarios. . 46
12 Inter-arrival latency at the 5-node consumer (time between consecutive messages received). Mean latency decreases as target rate increases because higher throughput reduces inter-message gaps. P99 latency at 100c_100r (843 ms) reflects occasional bursts when 100 clients deliver at low per-client rates. 46
13 Per-stage latency on the 5-node gold-standard run (10c_500r, May 14). Each individual stage stays under 10 ms at P99; the end-to-end total is dominated by the 2-second batch-accumulation window in the Python consumer. 47
14 May 13 accidental run on 2 active nodes with QoS 0 and the fixed pipeline. Compared with the April 2-node results (Table 11), the protocol and pipeline changes-not the node count-account for the improvement. 48

15 10c_500r results across all May 5-node runs, showing the impact of the DNS break and subsequent SD-card degradation. 49
16 Master-node resource usage: idle vs. peak benchmark (5-node, 2000 msg/s). Load-average figures are for the BCM2711 quad-core processor. 49
17 CNN-BiGRU model hyperparameters. 66
18 CNN-BiGRU model performance on the MOEPP test set (August 2024 - April 2025). An R^2 of 0.87 indicates strong predictive capability across the full seasonal range. 70

1 Introduction

1.1 The Urban Air Quality Problem

Fine particulate matter below 2.5 micrometers in aerodynamic diameter (PM2.5) penetrates the alveolar region of the lungs and enters the bloodstream. Long-term exposure raises the risk of cardiovascular disease, respiratory illness, and premature mortality [1]; short-term exposure causes acute effects including reduced lung function and increased hospital admissions.

Tetovo sits in the Polog Valley of northwestern North Macedonia, and its air quality problem is severe by any European standard. Winter PM2.5 concentrations rank among the highest recorded on the continent [2], [3]. The valley acts as a natural bowl: under high-pressure winter conditions a thermal inversion forms at low altitude, trapping cold air and the pollutants it carries near the ground. Residential heating in Tetovo is predominantly wood-fired, and the combination of geography, heating practice, and traffic produces winter episodes where 24-hour average PM2.5 regularly exceeds the WHO guideline of 15 μg/m^3 by factors of five to ten [4]. The health burden is measurable: studies in Macedonian cities have linked chronic exposure to statistically significant reductions in life expectancy [5], [6], and preliminary estimates of health-related economic costs across the Western Balkans put the figure in the billions of euros annually [7], [8].

The monitoring infrastructure is not proportional to the problem. A certified reference station costs between €20 000 and €100 000 to install, requires controlled siting, periodic calibration, and trained staff. Most mid-sized cities in the Western Balkans have two or three. Two or three point measurements cannot capture the spatial structure of an urban pollution field, which can vary by a factor of five or more between a busy arterial road and a park three hundred meters away [9]. Without spatial coverage, it is impossible to identify the highest-exposure neighbourhoods, evaluate mitigation measures, or communicate health risk at the block level.

Low-cost IoT sensor nodes based on optical particle counters now cost between €50 and €300 per unit. They do not meet reference station accuracy standards, but they can be deployed densely enough to capture spatial variation. The practical question is not whether to use them, but how to deploy and operate them effectively-and in particular how to correct the well-documented measurement biases that optical particle counters develop under field conditions [10], [11].

1.2 The Infrastructure Gap

Dense IoT sensor deployment introduces two infrastructure requirements that do not arise with sparse reference stations. The first is communications. A sensor node mounted on a lamp post or building facade needs a way to transmit its measurements. Wi-Fi is unsuitable: outdoor deployments cannot rely on managed Wi-Fi infrastructure, and cellular connectivity (NB-IoT, LTE-M) carries per-device subscription costs that scale poorly to hundreds of nodes. LoRaWAN, the Long Range Wide Area Network protocol, is the standard choice for this class

of deployment. It operates in license-free sub-GHz bands, achieves ranges of 2-10 km in urban environments with a single gateway, consumes microwatts during idle periods, and requires no per-device subscription. The practical limitation is coverage: LoRaWAN signal propagation is strongly affected by buildings and terrain. A gateway placement that looks adequate on a simplified map can leave significant areas without connectivity when the actual urban geometry is accounted for. Planning a LoRaWAN deployment for an irregularly built city requires a realistic propagation model and a systematic approach to finding good gateway positions. The second infrastructure requirement is data processing. Hundreds of sensor nodes generating data at intervals of one to five minutes produce a continuous time-series stream. This stream needs to be ingested reliably, validated (malformed messages and sensor dropouts are common), transformed into a storage format, and persisted for querying. The natural approach is to send everything to a cloud platform. Cloud processing is scalable and operationally simple, but it introduces two dependencies: reliable WAN connectivity and an ongoing service cost. In cities where internet infrastructure is inconsistent, and where municipal budgets are constrained, a system that stops working when the uplink drops is not suitable. Edge computing-processing data locally, on hardware installed near the sensors-eliminates both dependencies.

1.3 This Work

This thesis designs and implements an integrated framework at all three layers: estimating the pollution field from sparse measurements, planning the LoRaWAN network that supports sensor connectivity, and deploying the edge pipeline that processes sensor data. The components feed into each other. The air quality model identifies areas of high spatial variability-where the uncertainty in the pollution estimate is largest-and those are the locations where additional sensors are most useful. The LoRaWAN planner takes candidate sensor positions and finds gateway placements that cover them, working from the digital twin rather than a simplified map. The edge pipeline receives data from deployed sensors and stores it in a queryable form, continuously. End to end: a city with sparse reference data and a constrained infrastructure budget gets a continuously updated, spatially detailed picture of its air quality.

1.4 Objectives

- Develop a hybrid air quality estimation model that combines gradient boosting with spatially-aware physics-based corrections for road proximity, industrial influence, vegetation, and elevation.
- Build a 3D digital twin of the urban environment from publicly available DSM and DTM raster data, suitable for line-of-sight radio propagation analysis.
- Design and implement a Monte Carlo link reliability model over the digital twin.

- Formulate LoRaWAN gateway placement as a multi-objective optimization problem and solve it using NSGA-II.
- Deploy a complete cloud-native IoT data pipeline on commodity Raspberry Pi edge hardware and benchmark its throughput and latency under controlled load.
- Identify and quantify the binding performance constraints of the full stack.
- Develop and evaluate a CNN-BiGRU deep learning model for multi-day PM2.5 forecasting, trained on reference instrument data with 234 engineered features, and analyse the systematic bias introduced by low-cost sensor data.

1.5 Contributions

1. A hybrid PM2.5 estimation model combining XGBoost with distance-decay spatial correction terms for roads, industry, vegetation, and elevation.
2. A GPU-accelerated radio propagation simulator over a 3D digital twin derived from public DSM/DTM rasters.
3. An NSGA-II-based gateway placement optimizer producing Pareto-optimal coverage-energy trade-off configurations.
4. A containerized, Kubernetes-managed IoT edge pipeline on a Raspberry Pi cluster (benchmarked across 1-, 2-, 3-, and 5-node configurations), provisioned with Ansible and managed continuously by ArgoCD GitOps.
5. A systematic throughput and latency benchmark identifying MQTT QoS 2 serialization as the binding performance constraint, with each subsequent gain attributed to a specific configuration fix rather than to added hardware.
6. A CNN-BiGRU sequence forecasting model with 234-dimensional input features, trained on MOEPP reference data, with a root-cause analysis of humidity-induced bias in low-cost sensor forecasts.

1.6 Thesis Structure

Section 2 reviews prior work on air quality estimation, LoRaWAN planning, edge computing, and deep learning for time-series forecasting. Section 3 describes the overall system architecture. Section 4 develops the spatial air quality model. Section 5 describes the digital twin construction. Section 6 details the wireless propagation model. Section 7 formulates and solves the gateway placement problem. Section 8 describes the edge pipeline components. Section 9 presents the edge pipeline benchmark. Section 10 develops and evaluates the CNN-BiGRU forecasting model. Section 11 discusses all results. Section 12 addresses fault tolerance. Section 13 discusses limitations and future work. Section 14 concludes.

2 Related Work

2.1 Urban Air Quality Estimation

Air quality modelling has a long history in atmospheric science, where physics-based dispersion models derived from the advection-diffusion equation have been the regulatory standard since the 1960s. These models require detailed emission inventories and meteorological inputs and remain accurate when properly calibrated. Their limitation is computational cost: running a Gaussian plume or CFD-based simulation for a whole city at fine resolution is not practical in an operational monitoring context.

Data-driven approaches became viable once IoT monitoring produced sufficient training data. Zheng et al. [12] introduced U-Air, which infers air quality at unmonitored locations using spatiotemporal correlations across heterogeneous urban data sources including taxi traces, road networks, and meteorological measurements. The key insight was that co-located data streams can act as proxies for air quality even when they do not measure it directly.

Gradient boosting methods, particularly XGBoost [13], have become the standard baseline for tabular environmental prediction tasks. They handle nonlinear interactions between features, are robust to missing data, and produce well-calibrated predictions without extensive feature engineering. The limitation of purely statistical models for spatial estimation is that they treat all locations identically. Land-use regression (LUR) models [14] address this by incorporating spatial predictors-distance to roads, land cover class, traffic counts-as model features. LUR models are widely used in epidemiological exposure assessment and have been validated across European cities. The hybrid approach in this thesis draws on the LUR tradition but uses gradient boosting as the regression method and applies spatial corrections as explicit additive terms rather than model features.

Low-cost optical particle counters introduce a separate body of literature. Field evaluations have consistently found that raw OPC readings overestimate PM2.5 under elevated relative humidity, because water uptake by hygroscopic aerosols increases apparent particle size and scattering cross-section [11], [15], [16]. This bias is not random: it is correlated with the meteorological conditions that also produce the highest true PM2.5 concentrations in valley environments, which means naive use of raw sensor data for model training systematically inflates predictions at exactly the most critical times [17]. Several correction approaches have been validated in the literature. Antonio et al. [18] developed a relative humidity correction framework applicable to ambient OPC data; Si et al. [19] showed that machine-learning calibration models trained on co-located reference data can substantially reduce bias; and Brattich et al. [10] provide practical guidelines for sensor deployment and correction that have become a standard reference for field practitioners. For the Western Balkans specifically, Evagelopoulos et al. [20] provide a 12-year characterisation of PM2.5 and PM10 in a comparable lignite-heating valley environment in northern Greece, offering useful context for the concentration ranges and seasonal patterns observed in Tetovo.

2.2 LoRaWAN Network Planning

LoRa (Long Range) modulation was developed by Cycleo and acquired by Semtech in 2012. LoRaWAN is the MAC-layer protocol standardized by the LoRa Alliance that defines how LoRa devices communicate with gateways and network servers. Its key parameters-spreading factor (SF7 to SF12), bandwidth (125, 250, or 500 kHz), and coding rate-control the trade-off between range, data rate, and energy consumption. Higher spreading factors extend range but reduce data rate and increase time-on-air, which raises energy consumption per message and limits the channel's aggregate throughput. Raza et al. [21] survey LPWAN technologies and provide a comprehensive comparison of LoRaWAN, Sigfox, NB-IoT, and LTE-M. For outdoor environmental sensing at city scale, LoRaWAN is the dominant choice due to its license-free operation, open ecosystem, and established gateway hardware market. Radio propagation modeling for LoRaWAN in urban environments typically uses empirical path loss models. The Okumura-Hata model and its COST-231 extension are calibrated to urban macro-cell environments at 150-2000 MHz and provide a reasonable baseline. More accurate models incorporate terrain and building obstruction explicitly. Measurement-based studies [22] have shown that LoRaWAN links in dense urban areas experience 10-30 dB more attenuation than free-space predictions suggest, attributable primarily to building penetration loss and diffraction. The network planning problem-deciding where to place gateways-is a coverage optimization problem that has been studied extensively in the cellular network literature. For LoRaWAN, the additional complexity is the spreading factor assignment: devices far from a gateway must use higher spreading factors, which consumes more airtime and limits the network's aggregate capacity. Minimizing the average spreading factor across all devices is therefore an energy and capacity objective. Sales et al. [23] propose a heuristic placement strategy but do not model building obstruction or derive the Pareto front. This thesis does both. Multi-objective evolutionary algorithms are well-suited to problems where objectives conflict and the decision-maker needs to choose from a set of non-dominated solutions. NSGA-II [24] is the most widely cited algorithm in this class. It maintains a population of solutions ranked by Pareto dominance, uses crowding distance to preserve diversity along the Pareto front, and converges reliably on well-defined fronts for combinatorial problems of the scale addressed here.

2.3 Edge Computing and Cloud-Native IoT

The term "edge computing" describes a processing paradigm where computation occurs close to the data source rather than in a centralized data center. Bonomi et al. [25] framed this as "fog computing," a continuum of processing resources between end devices and the cloud. In the IoT context, edge processing reduces latency for time-sensitive applications, reduces WAN bandwidth consumption, and improves resilience to connectivity outages. Kubernetes has emerged as the standard orchestration layer for containerized workloads. Its extension to edge hardware has been driven by lightweight distributions, most notably K3s (Rancher Labs, now a CNCF project), which packages the Kubernetes control plane into a single binary with a memory footprint below 512 MB. Marella **marella2024** compared K3s

against standard Kubernetes on ARM hardware and found K3s reduced memory overhead by approximately 40% with full API compatibility. MicroK8s (Canonical) and k0s (Mirantis) offer similar trade-offs. MQTT (Message Queuing Telemetry Transport) is the dominant IoT messaging protocol, designed for constrained devices and unreliable networks. Handosa et al. [26] analyzed QoS behavior empirically and found that QoS 2, while providing exactly-once semantics, imposes a throughput penalty proportional to round-trip latency. This thesis quantifies that penalty precisely on ARM64 hardware. Time-series databases optimized for IoT workloads include InfluxDB, Prometheus, and VictoriaMetrics. Benchmarks on ARM hardware consistently favor VictoriaMetrics for write throughput and compression efficiency [27]. Distributed storage for Kubernetes on edge hardware is addressed by Longhorn (CNCF), which replicates block storage volumes across cluster nodes and provides automatic recovery from node failures. Zanella et al. [28] surveyed smart city IoT architectures and identified message brokers and time-series databases as the two core infrastructure components. This thesis confirms that finding and characterizes the performance limits of a complete stack built from these components on commodity hardware.

2.4 Deep Learning for Air Quality Forecasting

Recurrent neural networks became the standard approach for air quality time-series forecasting after Hochreiter and Schmidhuber introduced the LSTM architecture [29]. LSTMs address the vanishing gradient problem of standard RNNs through gating mechanisms that selectively retain and forget information over long sequences. For PM2.5 forecasting, LSTMs consistently outperform simple regression baselines when input sequences are long enough to capture multi-day autocorrelation. The BiGRU [30] simplifies the LSTM by merging the forget and input gates into a single update gate, reducing parameter count while maintaining comparable performance on sequence modeling tasks. For daily environmental time series at horizon lengths of one to two weeks, the BiGRU's simplification is generally sufficient, and the reduced parameter count lowers overfitting risk on moderately sized datasets. Hybrid CNN-RNN architectures have shown consistent improvements over pure recurrent models for time series with local periodic patterns. The convolutional layer acts as a learnable local feature extractor across adjacent time steps, producing compact representations that the recurrent layers then model over the full sequence. Zhang et al. [31] demonstrated this architecture's effectiveness for PM2.5 forecasting in Chinese cities, with the CNN block capturing diurnal patterns and the LSTM block capturing multi-day persistence. Bidirectional recurrent layers, which process sequences in both forward and backward directions, improve representation quality in offline settings where the full input window is available at inference time. They are appropriate here because the model predicts day $t + 1$ from a window of days $t - L + 1$ through t; within this window, bidirectional processing is valid. Federated learning for environmental monitoring has been proposed as a privacy-preserving alternative to centralized data aggregation. McMahan et al. [32] introduced FedAvg, which aggregates locally trained model updates rather than raw data. For air quality monitoring networks where sensor stations may be owned by different organizations, federated train-

ing allows a shared global model to be trained without transferring raw measurements. The trade-off is that systematic data quality issues-such as calibration drift-that are common across all clients cannot be corrected by aggregation alone.

3 System Architecture

3.1 Overview

The framework integrates three functionally distinct layers: an air quality estimation layer, a network planning layer, and an edge computing layer. Each layer can operate independently, but they are designed to feed into one another. The air quality estimation layer takes historical measurements from reference stations and meteorological data as input. It produces a continuous spatial pollution estimate across the urban area. This estimate serves two purposes: it characterizes the current state of air quality for public reporting, and it identifies areas of high spatial variability that benefit most from additional sensor coverage. The network planning layer takes the sensor deployment geometry-including candidate sensor positions derived from the air quality estimates-and computes a set of Pareto-optimal gateway placements. The planner operates on a 3D digital twin of the city, which provides building geometry for realistic propagation modeling. The output is a set of deployment configurations from which an operator can select based on available budget and coverage requirements. The edge computing layer receives continuous data streams from deployed sensors via MQTT and processes them through a containerized pipeline. The pipeline validates, transforms, and stores each measurement in VictoriaMetrics, from which it can be queried for dashboards, alerting, and model retraining.

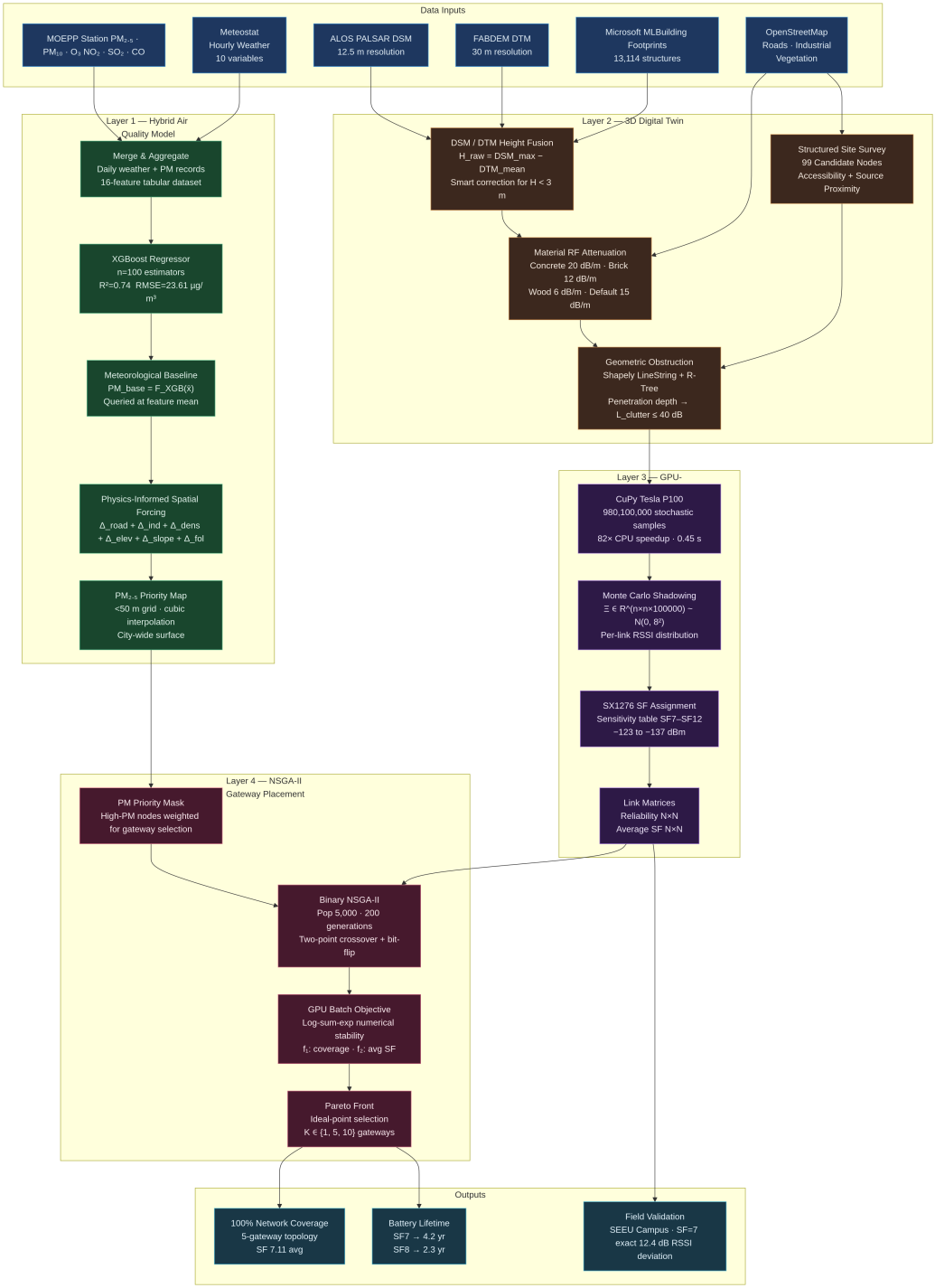


Figure 1: End-to-end system architecture. The air quality model (left) estimates the pollution field and guides sensor placement. The LoRaWAN planner (center) determines gateway positions using the digital twin for propagation modeling. The edge pipeline (right) ingests and stores sensor data continuously. Arrows indicate data flow; the three layers are loosely coupled through documented interfaces.

3.2 Design Principles

Several design decisions are shared across all three layers. First, all components use publicly available data and open-source software. The DSM and DTM rasters come from Copernicus Land Monitoring Service, available at no cost for the entire European continent. The edge pipeline components (K3s, EMQX, NATS, VictoriaMetrics, Longhorn) are all open source with permissive licenses. Second, each layer is reproducible: the air quality model is trained from documented data sources with pinned library versions; the edge pipeline is provisioned via version-controlled Ansible playbooks and its application state is reconciled continuously from Git by ArgoCD, so every benchmarked configuration corresponds to a tagged commit. Third, the layers are coupled by data interfaces rather than code dependencies, making it possible to replace any layer independently-for example, substituting a different optimization algorithm without changing the edge pipeline. The edge pipeline architecture is detailed in Figure 2.

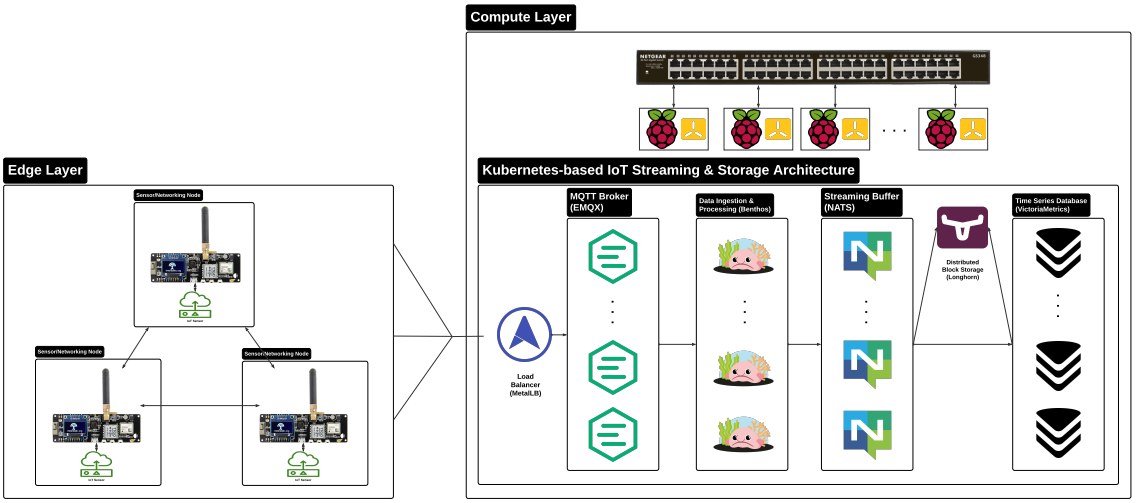


Figure 2: Cloud-native IoT pipeline deployed on the Raspberry Pi cluster. IoT sensors connect to EMQX via MQTT over the LoRaWAN gateway. Accepted messages are forwarded to NATS JetStream for durable queuing. Benthos subscribes to NATS, validates and transforms messages, and writes to VictoriaMetrics. Longhorn replicates stateful volumes (EMQX, Loki, Tempo) across nodes, while VictoriaMetrics and NATS use local-path storage for maximum write throughput. The entire stack is orchestrated by K3s, with MetalLB providing load-balanced external access and ArgoCD reconciling cluster state from Git.

4 Air Quality Modeling

4.1 Problem Formulation

The air quality estimation problem can be stated precisely. Let $\mathcal{S} = \{s_1, \ldots, s_k\}$ be the set of reference monitoring stations, each located at a known geographic coordinate (x_i, y_i) and producing measurements $c_i(t)$ of PM2.5 concentration at discrete times t. Let $\mathbf{m}(t)$ be a vector of meteorological variables observed at time t. The goal is to estimate the concentration field $\hat{C}(x, y, t)$ at every point (x, y) in the urban domain for any time t. This is an interpolation and extrapolation problem with two distinct challenges. The interpolation challenge is that station density is too low to support standard spatial interpolation methods: with two or three stations in a city, kriging or inverse distance weighting produces smooth fields that miss local features driven by traffic corridors, parks, or industrial zones. The extrapolation challenge is temporal: the model must produce estimates at times and conditions that may not be well-represented in the training data, such as pollution events under unusual meteorological conditions. The hybrid approach addresses both challenges. The machine learning component handles temporal extrapolation by learning the meteorology-pollution relationship from historical data. The spatial correction component handles local spatial features by encoding known emission sources and sinks as spatially varying functions.

4.2 Machine Learning Baseline: XGBoost

Gradient boosted trees have become the standard method for tabular regression tasks in environmental science. The underlying model is an additive ensemble of regression trees, each trained to fit the residuals of the previous ensemble:

$$F_T(\mathbf{x}) = \sum_{t=1}^{T} f_t(\mathbf{x}), \quad f_t \in \mathcal{F}$$

where $\mathcal{F}$ is the space of regression trees and F_T is the ensemble after T trees. Each tree f_t is fit by minimizing a regularized objective:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} \ell\big(y_i,\, F_{t-1}(\mathbf{x}_i) + f_t(\mathbf{x}_i)\big) + \Omega(f_t)$$

where ℓ is a loss function (mean squared error for regression), y_i is the measured PM2.5 at training sample i, and $\Omega(f_t) = \gamma T + \frac{1}{2}\lambda\|\mathbf{w}\|^2$ is a regularization term penalizing tree complexity via leaf count T and leaf weight magnitude $\|\mathbf{w}\|$. XGBoost [13] solves this objective efficiently using second-order Taylor approximations of the loss and a column subsampling strategy that reduces overfitting while enabling parallel computation. For this application, the training features are:

- Meteorological variables: temperature (°C), relative humidity (%), wind speed (m/s), wind direction (degrees), precipitation (mm/h), atmospheric pressure (hPa).

- Temporal features: hour of day (0-23), day of week (0-6), month (1-12), and a binary indicator for heating season (November-March).
- Lag features: PM2.5 measurements from the preceding 1, 3, 6, and 24 hours at the reference station.

The lag features are particularly important. PM2.5 concentration is autocorrelated: pollution events build up and dissipate over hours. Including recent history allows the model to capture persistence effects that are invisible to a model conditioned only on instantaneous meteorology. Figure 3 shows the feature importances assigned by the trained model, computed as the average gain per split across all trees. Wind speed, temperature, and the heating season indicator are the three most important features, consistent with the physical understanding that thermal inversions and residential heating are the dominant pollution drivers in the study area. The hour-of-day feature carries significant importance because traffic and heating follow strong diurnal patterns.

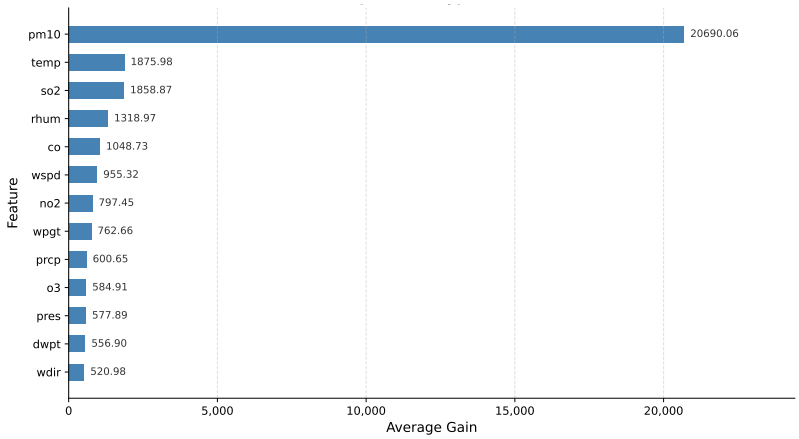


Figure 3: Feature importances from the XGBoost model, expressed as mean split gain normalized to the maximum. Meteorological variables and the heating season indicator dominate. Temporal features (hour, month) capture systematic diurnal and seasonal patterns in emission activity.

4.3 Spatial Correction Layer

The ML baseline produces a scalar prediction for a given time and set of meteorological conditions. It has no spatial coordinate as input, so it produces the same prediction at every location under the same conditions. The spatial correction layer adds location-dependent terms:

$$\text{PM}(x, y, t) = F_T\big(\mathbf{m}(t)\big) + \Delta_{\text{road}}(x, y) + \Delta_{\text{industrial}}(x, y) + \Delta_{\text{foliage}}(x, y) + \Delta_{\text{elevation}}(x, y) \quad (1)$$

Each correction term is time-invariant (it modifies the spatial distribution but not the temporal dynamics) and is defined over the same grid as the final output. The design choice to

keep correction terms additive and time-invariant reflects a practical constraint: the spatial data sources (road networks, land use, elevation) do not change on short timescales, while the meteorological inputs-and therefore the baseline-vary hourly.

Road proximity correction. Near-road PM2.5 is elevated due to direct traffic emissions (exhaust and brake/tire wear) and the resuspension of road dust. Empirical near-road gradient studies [14] find that PM2.5 decays roughly exponentially with distance from road centerlines, reaching background levels at 150-300 m. The correction term is:

$$\Delta_{\text{road}}(x, y) = \sum_{r \in R} A_r \cdot w_r \cdot \exp\left(-\frac{d_r(x, y)}{\lambda_r}\right)$$

where R is the set of road segments, $d_r(x, y)$ is the distance from point (x, y) to segment r, A_r is an amplitude coefficient proportional to road traffic volume, w_r is a road type weight (higher for major arterials than residential streets), and λ_r is the decay length scale. In this implementation, road network data is obtained from OpenStreetMap; traffic volumes are approximated by road functional class.

Industrial zone correction. Industrial facilities emit PM2.5 through combustion and mechanical processes. Unlike traffic emissions, industrial sources are localized and may have elevated effective stack heights, which changes the spatial distribution of ground-level impact. The correction term models each facility as a point source with a distance-decay plume:

$$\Delta_{\text{industrial}}(x, y) = \sum_{p \in P} B_p \cdot \exp\left(-\frac{d_p(x, y)^2}{2\sigma_p^2}\right)$$

where P is the set of industrial point sources, $d_p(x, y)$ is the distance to source p, B_p is an amplitude reflecting the facility's emission intensity, and σ_p controls the spatial spread of the influence zone. The Gaussian form is used here rather than the exponential decay of the road term because industrial plumes spread in two dimensions rather than along a line. Industrial facility locations are taken from the European Pollutant Release and Transfer Register (E-PRTR) and supplemented with locally identified sources.

Vegetation correction. Vegetation removes PM2.5 from the atmosphere through dry deposition, impaction on leaf surfaces, and turbulence modification. The correction is negative (a PM reduction) and proportional to vegetation density:

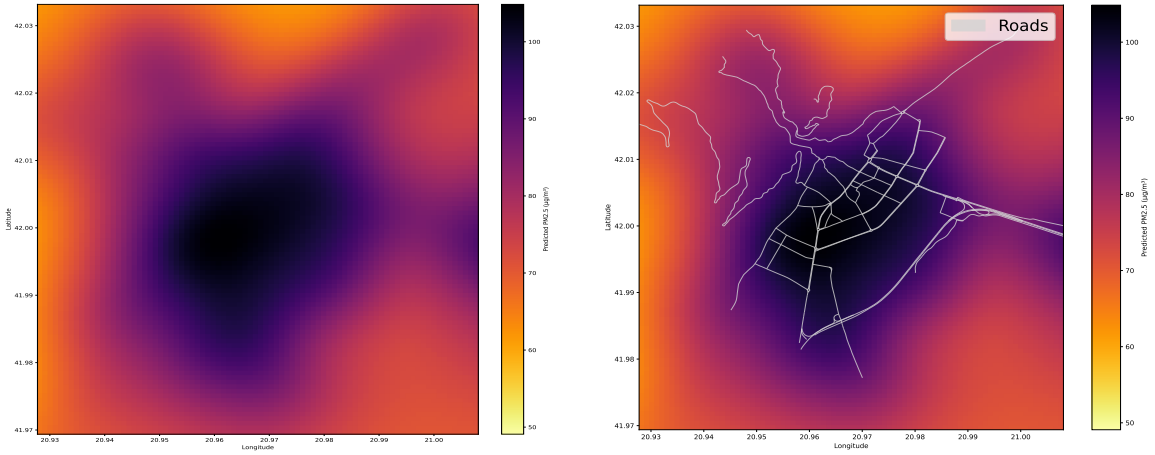
$$\Delta_{\text{foliage}}(x, y) = -C \cdot \text{NDVI}(x, y) \cdot f_{\text{canopy}}(x, y)$$

where NDVI (Normalized Difference Vegetation Index) is derived from Sentinel-2 satellite imagery, f_{canopy} is the estimated canopy fraction at the same pixel, and C is a calibration constant relating vegetation density to PM removal efficiency. The negative sign reflects the PM-reducing effect of vegetation. This term has the smallest magnitude of the four corrections in urban environments where vegetation is sparse, but it becomes significant in well-treed residential areas.

Elevation correction. In valley topography, thermal inversions create a sharp vertical boundary between the polluted cold air pool at valley-floor level and the cleaner air above. Locations above the inversion height experience substantially lower PM2.5 concentrations. The correction models this as a sigmoid transition:

$$\Delta_{\text{elevation}}(x, y) = -D \cdot \sigma\left(\frac{z(x, y) - z_{\text{inv}}}{s}\right)$$

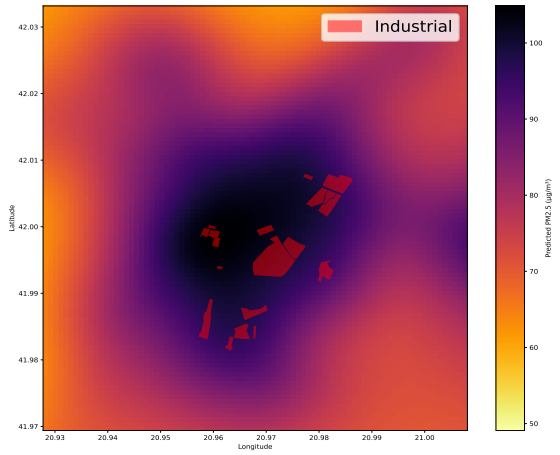
where $z(x, y)$ is terrain elevation from the DTM, z_{inv} is the estimated inversion height (derived from radiosonde data or reanalysis), s is a smoothing parameter controlling the sharpness of the transition, D is the maximum reduction amplitude, and $\sigma(\cdot)$ is the logistic function. For Tetovo, the valley floor is at approximately 450 m above sea level; the surrounding ridges are at 900-1400 m. During winter inversion events, the inversion base is typically at 600-700 m. Figures 4 and 5 illustrate the progressive effect of applying correction terms to a baseline ML prediction.



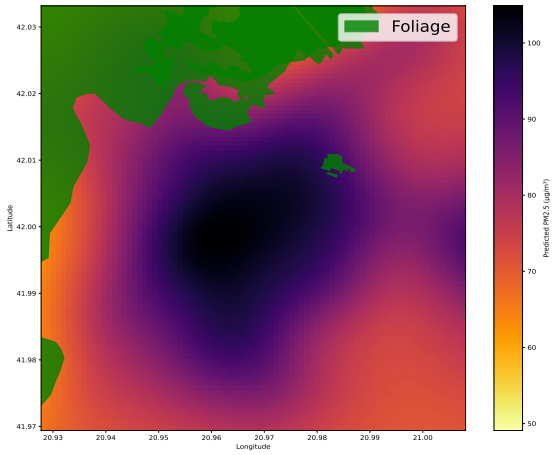
(a) Baseline ML prediction. Spatially uniform; reflects only the current meteorological conditions.

(b) After road proximity correction. PM concentrations increase along major road corridors and decay away from them.

Figure 4: Effect of the road correction term. The pollution field acquires spatial structure corresponding to the road network geometry. The decay length scale λ_r determines the width of elevated-PM corridors.



(a) After adding industrial correction. Localized PM spikes appear at facility locations with Gaussian spatial spread.



(b) After adding vegetation correction. PM concentrations are attenuated in vegetated areas, particularly in parks and tree-lined corridors.

Figure 5: Effect of industrial and vegetation correction terms. Both corrections are spatially localized and physically interpretable: industrial zones increase PM; vegetation decreases it.

4.4 Calibration and Validation

The correction term amplitudes (A_r, B_p, C, D) and decay parameters (λ_r, σ_p, s) are calibrated by minimizing the mean absolute error between model predictions and withheld reference station measurements. Because the correction terms are additive and the ML baseline is fixed after training, the calibration reduces to a bounded nonlinear optimization over the correction parameters. L-BFGS-B is used for this step. Cross-validation is performed by station holdout: for each fold, one reference station is withheld from both the ML training data and the calibration, and model performance is evaluated against its measurements. This tests the model's ability to generalize to new spatial locations, which is the primary operational use case.

5 Digital Twin and Urban Modeling

5.1 Motivation and Approach

Accurate radio propagation modeling in urban environments requires knowledge of the built environment. Buildings attenuate and diffract LoRaWAN signals; terrain elevation causes shadowing and in some cases multipath enhancement. A coverage model that ignores these effects will overestimate connectivity in dense urban areas and underestimate it in elevated suburban locations. The 3D digital twin approach used here derives building geometry directly from publicly available elevation raster data, without requiring a building footprint

database. This makes the approach applicable to any city with Copernicus-grade DSM/DTM coverage, which includes most of Europe. The trade-off is that raster-derived building models have limited accuracy at the edges of buildings and cannot represent internal structure, but this is acceptable for the purpose of radio propagation modeling, where the principal need is an estimate of obstruction between transmitter and receiver.

5.2 DSM/DTM Processing

The input data consists of two co-registered rasters at 1 m spatial resolution:

- The Digital Surface Model (DSM) records the elevation of the highest surface at each pixel: rooftops, tree canopy, and bridges.
- The Digital Terrain Model (DTM) records the bare-earth elevation beneath all surface features.

Building height at each pixel is computed as:

$$H(x, y) = \mathrm{DSM}(x, y) - \mathrm{DTM}(x, y) \tag{2}$$

Pixels where $H > 2$ m are classified as potential building or vegetation cover. To distinguish buildings from tree canopy, a normalized difference vegetation index (NDVI) mask is applied: pixels with high NDVI that also have $H > 2$ m are classified as trees rather than buildings. The remaining pixels with $H > 2$ m are classified as built structures. The building classification is rasterized into a 3D occupancy grid at 2 m horizontal resolution and 1 m vertical resolution. Each voxel is marked as occupied (building material) or free (air). The ground surface is represented by the DTM at the same resolution. Figure 6 shows the distribution of derived building heights across the study area.

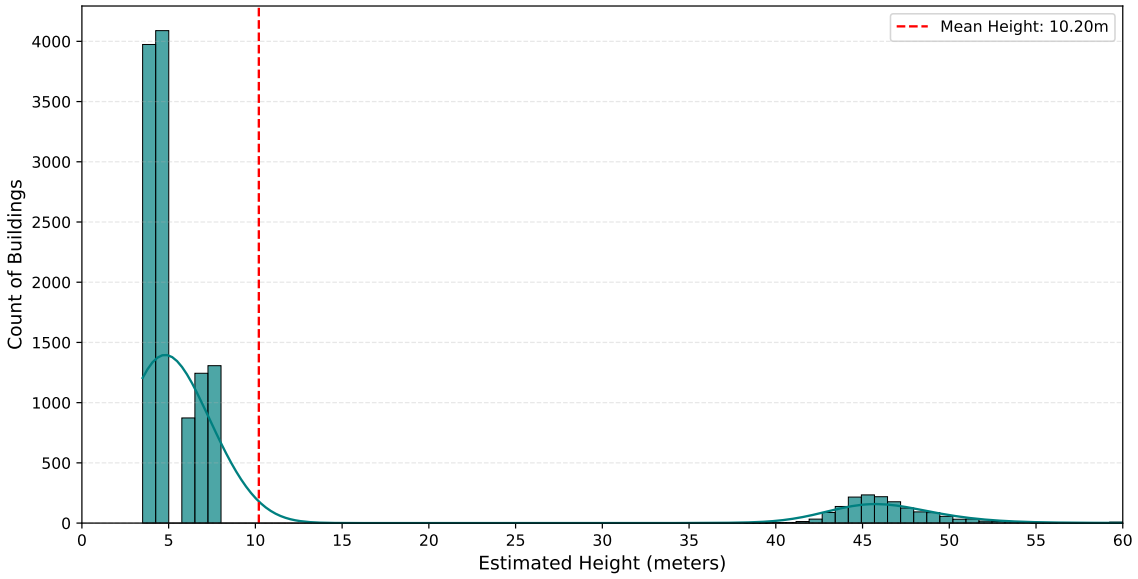


Figure 6: Distribution of building heights derived from DSM/DTM subtraction across the study area. The dominant mode at 8-12 m corresponds to three- to four-story residential construction, which is the prevailing typology in Tetovo's urban core. The secondary mode at 18-22 m corresponds to larger apartment blocks and institutional buildings. Values above 30 m are rare and correspond to industrial structures.

5.3 Line-of-Sight Computation

For a gateway at position $\mathbf{g} = (x_g, y_g, z_g)$ and a device at $\mathbf{d} = (x_d, y_d, z_d)$, line-of-sight (LOS) is defined as the absence of occupied voxels along the ray $\mathbf{r}(t) = \mathbf{g} + t(\mathbf{d} - \mathbf{g})$, $t \in [0, 1]$. The LOS check is implemented using a 3D DDA (Digital Differential Analyzer) traversal of the voxel grid, which visits each voxel intersected by the ray in order. The traversal terminates early on the first occupied voxel. On a single GPU thread, a single LOS check for a pair at maximum range (5 km) traverses approximately 2500 voxels. With a city grid of 10^4 device positions and 10^2 candidate gateway positions, the full coverage matrix requires 10^6 LOS checks, which are computed in parallel across GPU threads. In addition to strict LOS, the first Fresnel zone clearance is checked. The first Fresnel zone is an ellipsoid of revolution with foci at the transmitter and receiver; at any point along the path, its radius is:

$$r_F(d_1, d_2) = \sqrt{\frac{\lambda \cdot d_1 \cdot d_2}{d_1 + d_2}}$$

where d_1 and d_2 are the distances from the point to the transmitter and receiver, and λ is the wavelength ($\lambda \approx 0.345$ m at 868 MHz). A link is classified as "clear LOS" if the first Fresnel zone is unobstructed, "partial LOS" if the first Fresnel zone is partially obstructed but the

geometric ray is clear, and "NLOS" if the ray itself is blocked. This three-way classification is used to assign different path loss models to each link type.

6 Wireless Network Modeling

6.1 Path Loss Model

The received signal strength indicator at a device is:

$$\text{RSSI} = P_{\text{tx}} + G_{\text{tx}} + G_{\text{rx}} - L_{\text{total}} \tag{3}$$

where P_{tx} is the gateway transmit power (dBm), G_{tx} and G_{rx} are antenna gains (dBi), and L_{total} is total path loss (dB). For a link classified as clear LOS, total path loss is:

$$L_{\text{total}}^{\text{LOS}} = L_{\text{FSPL}}(d, f) + X_\sigma$$

For a partial LOS link, an additional clutter loss term is added:

$$L_{\text{total}}^{\text{pLOS}} = L_{\text{FSPL}}(d, f) + L_{\text{clutter}} + X_\sigma$$

For NLOS links, the path loss uses an empirical NLOS model calibrated to similar European urban environments:

$$L_{\text{total}}^{\text{NLOS}} = L_{\text{FSPL}}(d, f) + L_{\text{NLOS}}(d) + X_\sigma$$

Free-space path loss is:

$$L_{\text{FSPL}}(d, f) = 20\log_{10}(d) + 20\log_{10}(f) + 32.44 \tag{4}$$

with d in km and f in MHz. At 868 MHz and 1 km, $L_{\text{FSPL}} = 91.7$ dB. The NLOS excess loss $L_{\text{NLOS}}(d)$ is modeled as a linear function of distance calibrated to LoRaWAN measurement campaigns in comparable cities, adding approximately 20-35 dB at 1 km for typical dense urban morphology. The shadowing term $X_\sigma \sim \mathcal{N}(0, \sigma^2)$ with $\sigma = 7.8$ dB represents residual signal variation not captured by the geometry-based model-variations in building material, window placement, vegetation density, and other factors that are not represented in the voxel grid.

6.2 Spreading Factor Assignment

LoRaWAN devices select their spreading factor based on the received signal strength from the best-serving gateway. The minimum RSSI required for reliable demodulation at each SF is:

Spreading Factor	Min RSSI (dBm)	Data Rate (bps)	Time on Air (ms)
SF7	-123	5470	46
SF8	-126	3125	83
SF9	-129	1760	165
SF10	-132	980	329
SF11	-134.5	440	659
SF12	-137	250	1319

Table 1: LoRaWAN spreading factor parameters at 125 kHz bandwidth for a 20-byte payload. Higher spreading factors extend range at the cost of lower data rate and longer time on air.

A device is considered "covered" by a gateway if its RSSI from that gateway is sufficient for at least SF12 demodulation. The assigned spreading factor is the lowest (fastest) SF for which the RSSI threshold is met. Minimizing the assigned SF across all devices simultaneously minimizes energy consumption and maximizes network capacity, since lower SFs reduce time on air.

6.3 Monte Carlo Reliability Estimation

Because X_σ is a random variable, link reliability is a probability rather than a deterministic quantity. For a given device-gateway pair (i, j) with deterministic path loss $\bar{L}_{ij}$ and an SF threshold θ_{ij}, link reliability is:

$$P_{ij}(\text{link}) = P\big(\text{RSSI}_{ij} > \theta_{ij}\big) = P\big(X_\sigma < \bar{L}_{ij} - L_{\text{threshold}}(\theta_{ij})\big)$$

This has a closed-form expression using the normal CDF, but the composite reliability across all devices (requiring at least one covering gateway per device) requires numerical computation. We estimate it via Monte Carlo:

$$P_i(\text{covered}) = 1 - \prod_{j \in G} \Big(1 - \hat{P}_{ij}(\text{link})\Big) \tag{5}$$

where $\hat{P}_{ij}$ is estimated by drawing $N = 500$ samples of X_σ per link and computing the empirical success rate. Network-level coverage is then $\bar{P} = \frac{1}{|D|} \sum_{i \in D} \mathbf{1}[P_i(\text{covered}) > \delta]$ for a coverage reliability threshold δ (set to 0.95 in this work). Figure 7 shows the resulting per-device reliability matrix for a representative gateway placement over the study area.

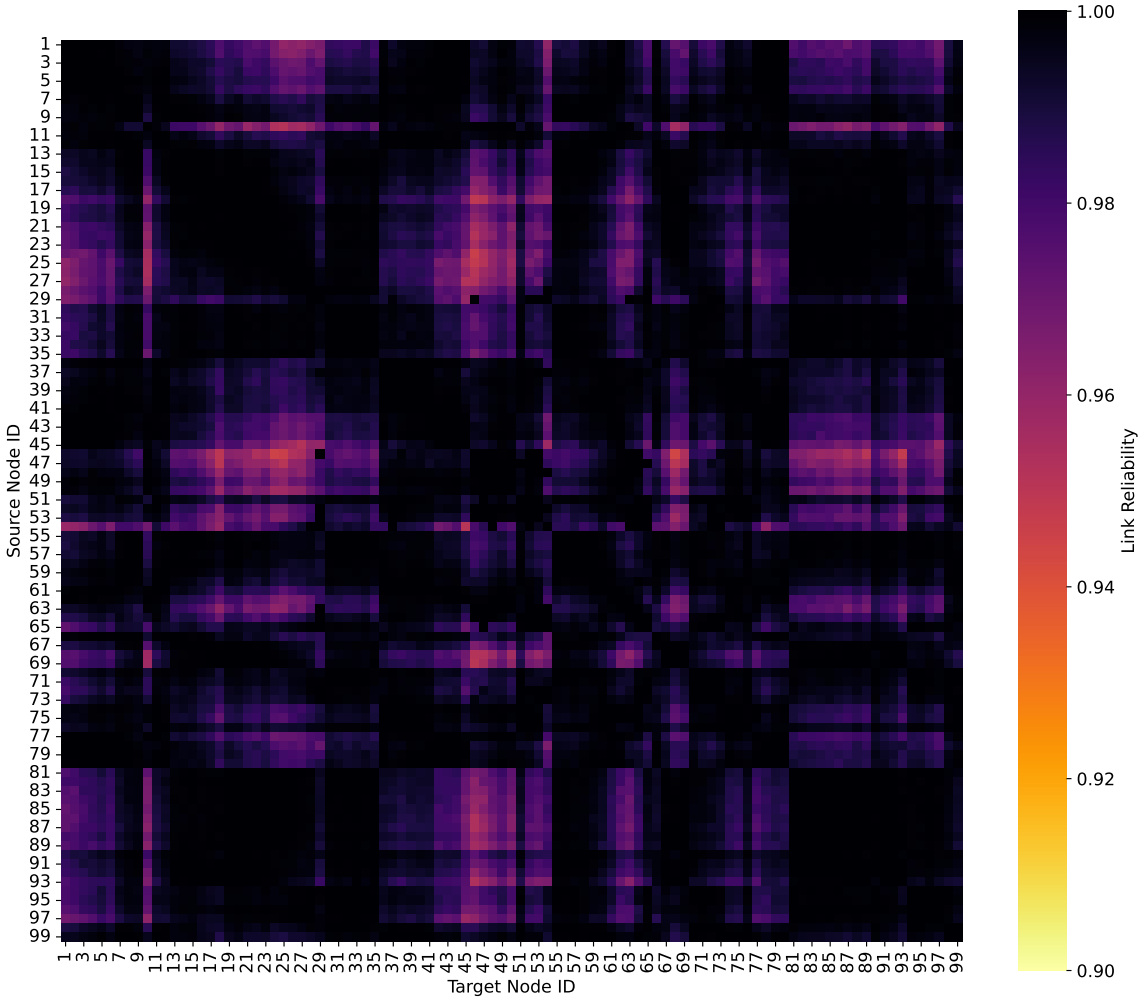


Figure 7: Per-link communication reliability across the study grid, estimated via Monte Carlo simulation over the digital twin geometry. Reliability is non-uniform: elevated terrain near the gateway achieves high reliability due to clear LOS, while dense building clusters in the center produce shadow zones with substantially reduced reliability. These patterns are invisible in a flat-earth coverage model.

7 Network Optimization

7.1 Problem Formulation

The gateway placement problem is formulated as follows. Let $\mathcal{C}$ be the set of candidate gateway positions (rooftops, lamp posts, or other elevated structures identified from the digital twin). Let $\mathcal{D}$ be the set of device positions (sensor deployment locations). For a subset

$G \subseteq \mathcal{C}$ of placed gateways, define:

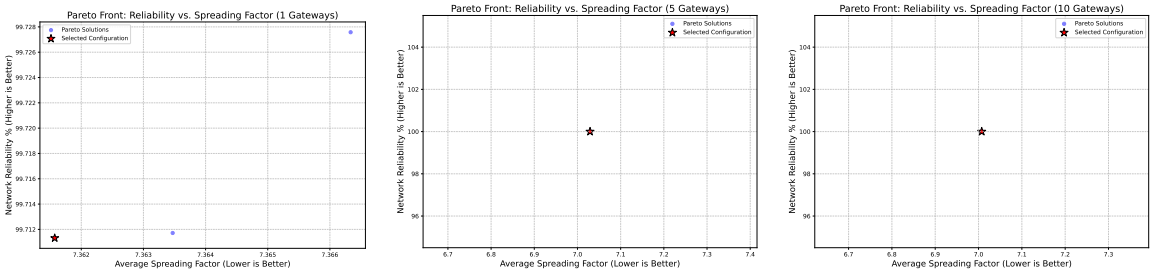
$$f_1(G) = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \mathbf{1}\big[P_i(\text{covered} \mid G) \geq \delta\big] \tag{6}$$

$$f_2(G) = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \mathsf{SF}_i^*(G) \tag{7}$$

where f_1 is network coverage fraction (to be maximized) and f_2 is average spreading factor of covered devices (to be minimized). The cardinality constraint $|G| = k$ fixes the number of gateways, and separate optimization runs are conducted for $k \in \{1, 5, 10\}$. The two objectives conflict: placing gateways close to devices reduces their required SF and improves f_2, but close placement may leave distant devices uncovered, harming f_1. The optimal trade-off depends on the specific topology of the device distribution and the urban geometry; it cannot be determined analytically.

7.2 NSGA-II Optimizer

NSGA-II [24] is a multi-objective genetic algorithm that maintains a population of candidate solutions $\{G_1, \ldots, G_P\}$, each a subset of k gateway positions from $\mathcal{C}$. At each generation, offspring are produced by tournament selection, crossover (swapping gateway positions between parent solutions), and mutation (replacing a randomly selected gateway with a different candidate position). The combined parent-offspring population is ranked by Pareto dominance: solution G dominates G' if $f_1(G) \geq f_1(G')$ and $f_2(G) \leq f_2(G')$ with at least one strict inequality. Solutions in rank 1 (the non-dominated front) are preferred over all others; within the same rank, crowding distance is used to preserve diversity along the front. The algorithm was run for 2000 generations with a population size of 5000, which was sufficient for convergence of the Pareto front across all tested gateway counts. GPU acceleration is used to evaluate the f_1 and f_2 objectives: for a population of 5000 solutions, each requiring $|\mathcal{D}| \approx 10^4$ device-level reliability computations, serial evaluation would require hours per generation. GPU parallelism reduces this to seconds. Figure 8 shows the Pareto fronts for 1, 5, and 10 gateways.

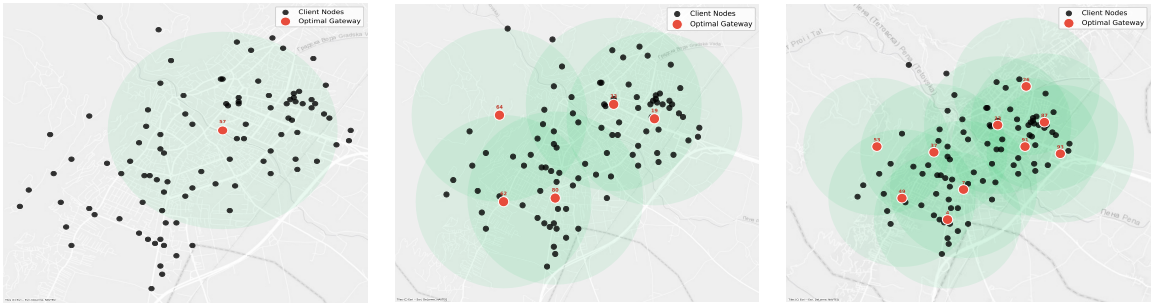


(a) 1 gateway. (b) 5 gateways. (c) 10 gateways.

Figure 8: Pareto fronts for different gateway counts after 2000 generations with a population of 5000. Each point represents a non-dominated gateway placement configuration. The front shifts toward higher coverage and lower average SF as the gateway count increases. The shape of the front indicates the marginal cost in average SF of extending coverage to the final few percent of devices.

7.3 Optimized Deployment Configurations

Figure 9 shows the selected deployment configurations from the Pareto front for each gateway count, at the operating point that maximizes coverage subject to average SF ≤ 9.



(a) 1 gateway. (b) 5 gateways. (c) 10 gateways.

Figure 9: Final optimized gateway positions overlaid on the urban grid. Gateway positions cluster at elevated terrain and building rooftops that offer wide-area line-of-sight. With 10 gateways, the optimizer achieves near-complete coverage of the device grid while maintaining low average SF.

Figure 10 shows the complete network topology for the 10-gateway configuration at 99.9% device coverage, with each device assigned to its best-RSSI gateway and colored by SF assignment.

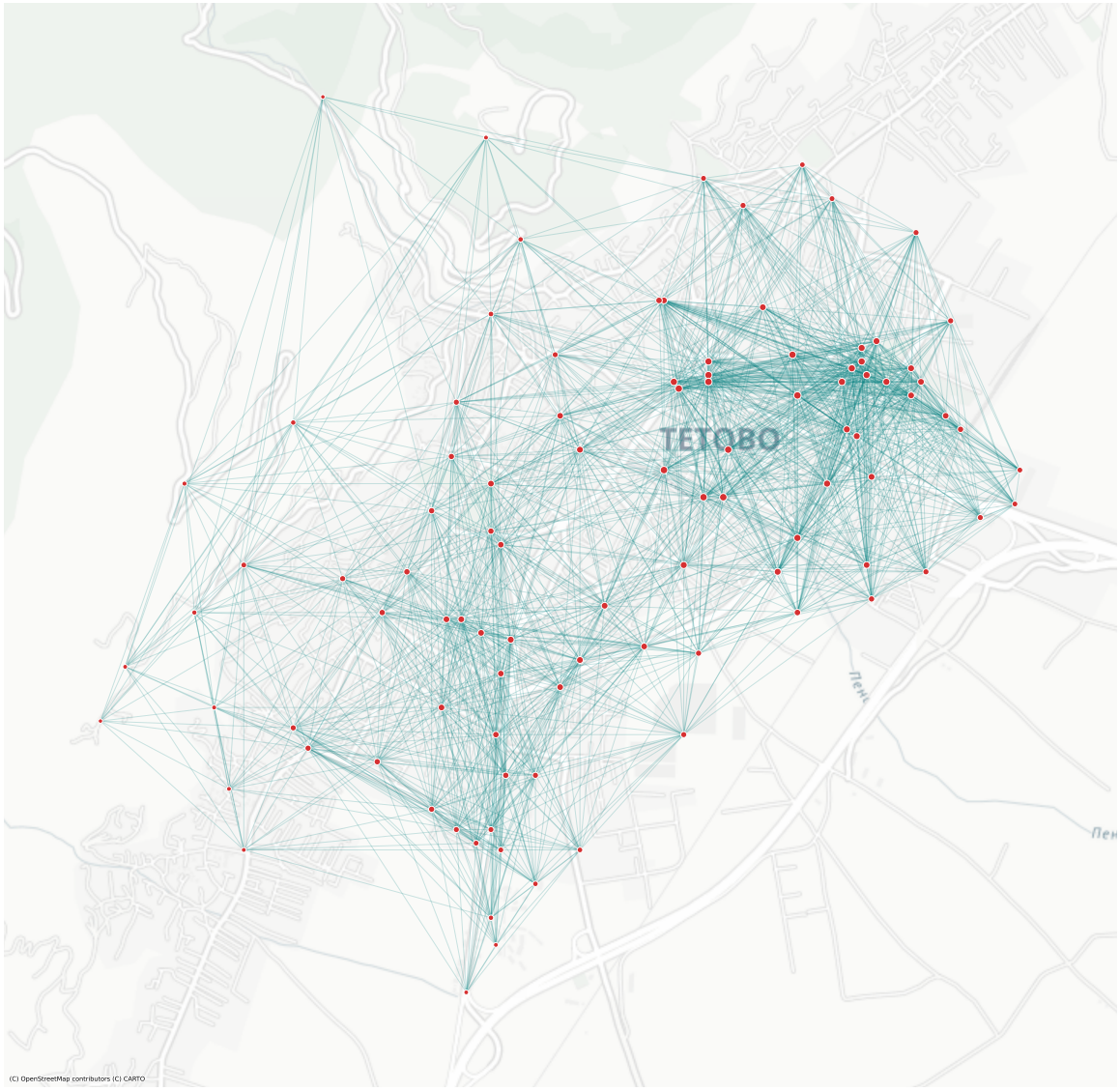


Figure 10: Optimized network topology for the 10-gateway configuration achieving 99.9% device coverage. Devices are colored by spreading factor assignment: SF7 (closest devices, highest data rate) through SF12 (devices at maximum range). The spatial structure reflects the underlying propagation geometry: devices in building-dense areas require higher SFs even at moderate distances.

7.4 Field Validation

Simulation results are only as useful as the model they are derived from. To test whether the Monte Carlo propagation model produces predictions that match real hardware behavior, a

field measurement campaign was conducted on the SEEU campus in Tetovo.

7.4.1 Hardware and Setup

Two LilyGO T-Beam v1.2 nodes were deployed as a point-to-point LoRa link. The T-Beam integrates a SX1276 LoRa transceiver, an ESP32 microcontroller, and a GPS module on a single board, making it a practical low-cost platform for RSSI measurement campaigns. One node was placed at POI 1 (Building 816) and one at POI 3 (Building 101), with a separation of approximately 350 m. The geometry between the two points is near line-of-sight: the buildings along the path are predominantly two to three stories, and the link geometry clears the first Fresnel zone under the modeled conditions. RSSI measurements were logged over multiple transmissions and averaged. The corresponding simulation output was extracted from the Monte Carlo model at the same two grid coordinates, using the digital twin geometry derived from the campus DSM/DTM data.

7.4.2 Results and Interpretation

Table 2 compares the simulation predictions against the field measurements.

Table 2: Simulation versus field measurements on the SEEU campus (Building 816 $\leftrightarrow$ Building 101 / Rectorate, $d \approx 350\,\mathrm{m}$, near-LOS geometry).

Metric	Simulation	Measured
RSSI (mean)	-102.6 dBm	-115.0 dBm
RSSI deviation	12.4 dB (optimistic bias in simulation)	
Network reliability	100%	100%
Spreading factor	7.003	7.000

Three observations follow from this comparison. The simulation overestimates RSSI by 12.4 dB. This is a known characteristic of geometry-based propagation models that use published clutter loss parameters calibrated to average European urban environments rather than to the specific building materials, vegetation density, and construction typology of the measurement site. A 12.4 dB optimistic bias is within the expected range for this class of model: measurement campaigns in similar environments typically report simulation-to-measurement deviations of 8-15 dB depending on how well the digital twin captures local obstructions. The bias is systematic rather than random, which means it can in principle be corrected by a single calibration factor once a broader set of reference measurements is available. Despite the RSSI offset, both the simulated and measured spreading factor assignments are SF7, and both predict 100% link reliability for this geometry. This is the practically important result. The propagation model is conservative enough in its coverage predictions to assign the correct SF even when it overestimates the absolute signal level: the 12.4 dB

optimistic bias shifts the predicted RSSI from -115.0 dBm to -102.6 dBm, but both values are well above the SF7 demodulation threshold of -123 dBm. For link budget planning purposes-deciding which spreading factor a device will use and whether a given gateway placement provides coverage-the model produces correct decisions even with uncalibrated clutter loss parameters. The implication for the gateway placement optimizer is straightforward. The NSGA-II solutions derived from the uncalibrated model will assign correct spreading factors to most devices, but predicted RSSI margins will be overstated by approximately 12 dB on average. Devices predicted to be covered near the sensitivity threshold (RSSI just above -137 dBm for SF12) may in practice fall below it. The effect is that actual coverage will be slightly lower than the optimizer predicts for marginal links at maximum range. For the deployment configurations selected from the Pareto front-which target average SF ≤ 9, meaning most devices are operating well above SF12 minimum margins-this bias is unlikely to change the practical outcome of any placement decision. Field calibration of the clutter loss parameter would eliminate the bias entirely and improve confidence in marginal-link predictions.

8 Edge Computing Pipeline

8.1 Hardware Configuration

The edge cluster comprises up to five Raspberry Pi 4B single-board computers. The pipeline was benchmarked in 1-, 2-, 3-, and 5-node configurations; the five-node configuration is the production reference, and the pod-placement and resource figures reported below refer to it. The Pi 4B uses the Broadcom BCM2711 SoC, which integrates a quad-core ARM Cortex-A72 processor at 1.5 GHz. The Cortex-A72 implements the ARMv8-A 64-bit architecture and includes a hardware floating-point unit and NEON SIMD extensions, which matter for the signal processing tasks performed by Benthos. Each node has 8 GB of LPDDR4-3200 SDRAM-the maximum available for the Pi 4B-and a 64 GB SanDisk Extreme V30 microSD card for storage. The nodes are connected via Gigabit Ethernet to a managed Layer 2 switch. Layer 2 switching (without routing) was chosen to minimize forwarding latency and ensure that Longhorn replication traffic and application traffic share the same flat network without routing overhead.

Component	Specification
SoC	Broadcom BCM2711
CPU	Quad-core ARM Cortex-A72 (ARMv8-A), 1.5 GHz
Memory	8 GB LPDDR4-3200
Storage	64 GB SanDisk Extreme V30 microSD (read: 100 MB/s, write: 60 MB/s)
Network	Gigabit Ethernet (IEEE 802.3ab)
Network topology	Layer 2 switch, no routing
Operating system	Raspberry Pi OS Lite 64-bit (Debian Bookworm base)
Kernel	Linux 6.1, cgroups v2 enabled
Container runtime	containerd 1.7 (bundled with K3s)
Orchestration	K3s v1.35.4+k3s1
Distributed storage	Longhorn v1.7.3

Table 3: Hardware and software configuration for each cluster node. All nodes are identical; the K3s version was aligned to v1.35.4+k3s1 across all nodes in the five-node configuration (Section 9.1.1).

8.2 Orchestration: K3s and MetalLB

K3s packages the Kubernetes API server, scheduler, controller manager, kubelet, and the containerd runtime into a single binary of approximately 70 MB. It replaces etcd with SQLite by default (this thesis uses a single-server cluster with up to four agent nodes, so the embedded datastore runs only on the server node). Compared to full Kubernetes, K3s eliminates several optional components-cloud provider integrations, in-tree storage drivers, and legacy admission controllers-that are unused in edge deployments. The result is a Kubernetes distribution that is API-compatible with upstream but consumes approximately 300 MB of RAM at idle rather than 1-2 GB. MetalLB implements the Kubernetes `LoadBalancer` service type for clusters that do not have a cloud provider to provision external load balancers. In layer 2 mode, MetalLB announces service IPs using ARP, allowing external clients (sensor nodes connecting via MQTT) to reach the EMQX broker at a stable IP address regardless of which node the broker pod is currently running on.

8.3 MQTT Ingestion: EMQX

EMQX is an open-source MQTT broker designed for high connection density and low-latency message delivery. It is implemented in Erlang/OTP, which gives it the actor-model concurrency properties that make it well-suited to handling thousands of persistent TCP connections. Sensor nodes connect to EMQX and publish JSON payloads containing:

- `timestamp`: ISO 8601 UTC timestamp of the measurement.
- `device_id`: Unique sensor node identifier (UUID).

- `pm1`: PM_1 concentration ($\mu g/m^3$).
- `pm2_5`: PM2.5 concentration ($\mu g/m^3$).
- `pm10`: PM_{10} concentration ($\mu g/m^3$).
- `temperature`: Ambient temperature (°C).
- `humidity`: Relative humidity (%).

The serialized payload is approximately 120 bytes. Messages are published to a topic hierarchy structured as `sensors/{city}/{device_id}/air`, which allows subscription filtering by city or individual device. EMQX is configured with QoS 2 in the early (1/2/3-node) deployments. QoS 2 (exactly-once delivery) is the strictest MQTT quality of service level and requires a four-message exchange per published message:

1. **PUBLISH**: Sender transmits message with a message ID and stores it locally.
2. **PUBREC**: Receiver acknowledges receipt and stores the message ID.
3. **PUBREL**: Sender releases the message, confirming it is safe to deliver downstream.
4. **PUBCOMP**: Receiver completes the transaction; both parties discard stored state.

This four-way handshake ensures exactly-once delivery even in the presence of retransmissions and network partitions. The cost is that message delivery is serialized within each client connection: a new PUBLISH cannot begin until the preceding PUBCOMP is received. This serialization is the binding throughput constraint, as the benchmark results confirm; the five-node runs switch to QoS 0 to remove it (Section 9.1.1).

8.4 Event Streaming: NATS JetStream

NATS is a lightweight publish-subscribe message system written in Go. JetStream, NATS's persistence layer, adds durable streams with configurable retention, replication, and replay. Messages published to a JetStream stream are written to disk and replicated before being acknowledged, providing at-least-once delivery guarantees to consumers. EMQX forwards accepted MQTT messages to NATS using the EMQX NATS bridge plugin. From the NATS perspective, each MQTT message becomes a JetStream message on a subject corresponding to the MQTT topic. Downstream consumers (Benthos) use pull-based JetStream consumers with explicit acknowledgment, meaning a message is only removed from the stream after Benthos confirms it has been written to VictoriaMetrics. If Benthos crashes or restarts, it resumes from the last acknowledged offset and processes any backlogged messages. NATS was chosen over Apache Kafka for two reasons. First, Kafka requires a JVM with a minimum heap of 512 MB, and its ZooKeeper (or KRaft) coordination overhead adds another 256-512 MB. On nodes with 8 GB of RAM shared across many workloads, this is acceptable but wasteful. NATS has an idle memory footprint below 50 MB. Second, Kafka's throughput advantages (millions of messages per second) are irrelevant at the message rates of this deployment.

8.5 Transformation: Benthos

Benthos (now maintained as Redpanda Connect) is a stream processor that reads from one or more input sources, applies a processing pipeline, and writes to one or more outputs. It is configured via YAML and requires no code. The Benthos pipeline in this deployment performs three operations on each message:

1. **Schema validation**: The JSON payload is parsed and checked for required fields with correct types. Messages that fail validation are logged and discarded rather than forwarded to storage-invalid data should not pollute the time series.

2. **Field transformation**: Timestamps are normalized to UTC Unix epoch format. Field names are mapped to VictoriaMetrics label conventions. Device metadata (city, location coordinates) is injected from a static configuration based on `device_id`.

3. **Output formatting**: Each PM measurement is converted to a Prometheus remote write request (the format VictoriaMetrics accepts), with appropriate metric names (`air_quality_pm25`, `air_quality_pm10`, etc.) and labels (`device_id`, `city`, `lat`, `lon`).

Benthos runs as a stateless Kubernetes deployment with a configurable replica count. Because it does not hold state-JetStream tracks message offsets, and VictoriaMetrics stores the data-horizontal scaling is straightforward: adding replicas increases the number of messages processed in parallel without any coordination overhead.

8.6 Time-Series Storage: VictoriaMetrics

VictoriaMetrics is a time-series database optimized for high write throughput and efficient storage on resource-constrained hardware. It accepts data via the Prometheus remote write protocol, OpenTSDB HTTP API, and InfluxDB line protocol, making it compatible with a wide range of data sources. For this deployment, VictoriaMetrics runs as a single-instance deployment (rather than a cluster) appropriate for the write volumes expected from the sensor fleet. It stores data on a local-path persistent volume rather than a Longhorn-replicated one: VictoriaMetrics maintains its own write-ahead log, so layering synchronous Longhorn replication on top would double SD-card write I/O for no reliability gain (NATS JetStream uses local-path storage for the same reason). VictoriaMetrics provides a HTTP query API compatible with PromQL (Prometheus Query Language), which is used by Grafana dashboards and by the air quality model retraining pipeline. VictoriaMetrics was preferred over InfluxDB and Prometheus for storage because benchmarks on ARM64 hardware consistently show lower write latency, better compression ratios (typically 0.4-0.8 bytes per sample versus 1-2 bytes for InfluxDB), and lower memory usage under sustained write load. Under benchmark conditions it consumed approximately 109 MB of RAM.

8.7 Distributed Storage: Longhorn

Longhorn is a CNCF-incubating distributed block storage system for Kubernetes. It manages persistent volumes by maintaining replicated volumes across cluster nodes: each volume has a configurable replica count, and Longhorn ensures that writes are committed to all replicas before acknowledging to the application. Each Longhorn volume has replication factor 2: two replicas placed on separate nodes. If a node fails, a surviving node already holds a current replica and can serve data immediately without a recovery phase. Longhorn's control plane detects the failure via Kubernetes node status and schedules a new replica rebuild once the node recovers or a replacement is added. Longhorn is used for the stateful components that lack their own durability mechanism-EMQX, Loki, and Tempo-while VictoriaMetrics and NATS use local-path storage as described above. The cost of this fault tolerance is I/O overhead. Every write to a Longhorn volume must be committed to both replicas and flushed to the underlying storage (fsync) before the write is acknowledged. On microSD storage, where individual fsyncs can take 10-50 ms during write amplification events, this synchronous replication is the dominant contributor to tail latency for the components that use it.

8.8 Infrastructure Automation: Ansible

The cluster is provisioned using Ansible playbooks implementing Infrastructure as Code principles. The playbooks are organized in three layers:

1. **OS hardening**: Applies security configuration to Raspberry Pi OS, enables 64-bit cgroups v2 (required by Kubernetes), configures kernel parameters for network performance (TCP buffer sizes, connection tracking limits), and disables unnecessary services.

2. **K3s installation**: Bootstraps the K3s control plane on the primary (server) node, joins the remaining nodes as agents, installs MetalLB with an IP address pool drawn from the local subnet, and configures kubeconfig.

3. **ArgoCD bootstrap**: Installs ArgoCD from a pinned manifest and registers the root Application. From this point, application deployment is handled declaratively by ArgoCD via GitOps (Section 8.9) rather than by Ansible: EMQX, NATS, Benthos, VictoriaMetrics, Longhorn, and the observability stack are installed from version-pinned Helm values files committed to Git.

The playbooks are idempotent: running them on an already-configured cluster applies only the differences from the current state. A full cluster rebuild from bare hardware takes approximately 20 minutes.

8.9 GitOps with ArgoCD

Once Ansible has bootstrapped the cluster and installed ArgoCD, the entire cluster state-every Helm values file, every namespace, every NetworkPolicy-is declared in Git and reconciled continuously by ArgoCD. If a value is changed manually on the cluster, ArgoCD reverts it; if a commit changes a Helm values file, ArgoCD applies it within minutes. The deployment uses the App-of-Apps pattern: a root Application points to a Kustomization that lists every sub-application, and each sub-application deploys one component with `automated: {prune: true, selfHeal: true}`, so ArgoCD deletes resources that no longer appear in Git and corrects any manual drift. Sync waves enforce deployment order: MetalLB (wave 1) must exist before any `LoadBalancer` Service; Longhorn (wave 2) before any StatefulSet with a PVC; NATS (wave 3) before producers or consumers; and VictoriaMetrics (wave 4) before the consumer writes; EMQX, Benthos, the consumer, and the observability components follow. This ordering matters for reproducibility. Every cluster configuration in the benchmark is a tagged commit, so every performance difference can be traced to a specific file change rather than to ambient state drift-a property that proved decisive when attributing the four engineering fixes of Section 9.1.1 and is discussed further in Section 11.6.

8.10 Observability and Resource Budget

A full observability pipeline runs alongside the IoT data path: Prometheus scrapes Kubernetes, node, and kube-state metrics; Grafana serves pre-built dashboards (raw sensor readings, per-stage pipeline latency, node health, Longhorn volume status, message rate, and Raspberry Pi thermal/throttling metrics); Promtail ships pod and container logs to Loki as a DaemonSet on every node; and Tempo receives distributed traces via OTLP, with an OpenTelemetry Collector fanning data to both Tempo and Loki. All namespaces run default-deny ingress, and explicit NetworkPolicy objects permit only the required paths-the same policy mechanism whose DNS-egress rule caused the failure analysed in Section 9.1.1. Tables 4 and 5 report pod placement and per-pod memory in the five-node configuration. The placement is deliberately master-heavy: the master node (raspberrypi) hosts EMQX, Benthos, VictoriaMetrics, Prometheus, Loki, Tempo, ArgoCD, and CoreDNS, while the workers host consumer replicas and monitoring agents. A notable finding is that the observability stack consumed more memory than the data pipeline itself: three Grafana replicas alone used about 1.2 GB, whereas NATS and all consumer replicas together used under 60 MB.

Table 4: Pod placement in the 5-node configuration. The master carries all stateful pipeline components; workers host consumer replicas and monitoring agents.

Namespace	Component	Node
emqx	EMQX broker	raspberrypi (master)
benthos	Benthos	raspberrypi
nats	NATS server $\times 2$	raspberrypi, pi7
nats-consumer	Consumer $\times 3$	raspberrypi, pi7, pi2
victoriametrics	VictoriaMetrics	raspberrypi
monitoring	Grafana $\times 3$	raspberrypi, pi7, pi2
monitoring	Prometheus	raspberrypi
monitoring	Node Exporter $\times 5$	all nodes
logging	Loki, Tempo, OTel	raspberrypi
logging	Promtail $\times 5$	all nodes
argo	ArgoCD	raspberrypi
kube-system	CoreDNS	raspberrypi

Table 5: Memory usage by major pod during 5-node benchmarks. The observability stack (Grafana, Prometheus, OpenTelemetry Collector) consumes more RAM than the data pipeline.

Pod	Memory (MB)
Grafana (per replica, $\times 3$)	400
EMQX	228
OpenTelemetry Collector	186
ArgoCD Controller	177
MetalLB Speaker	130
VictoriaMetrics	109
Benthos	82
Prometheus Operator	80
Metrics Server	79
CoreDNS	76
NATS	26
NATS Consumer ($\times 3$, total)	32

9 Experimental Evaluation

9.1 Benchmark Design

The benchmark measures end-to-end performance of the IoT pipeline across four cluster configurations: 1-, 2-, 3-, and 5-node (all nodes are 8 GB Raspberry Pi 4B). Each configuration runs the same message flow-MQTT $\rightarrow$ EMQX $\rightarrow$ Benthos $\rightarrow$ NATS $\rightarrow$ consumer $\rightarrow$ VictoriaMetrics-but with different consumer implementations and pipeline tuning as the

cluster evolved. Changes at each step are cumulative: the 3-node configuration inherited everything from 2-node except what is explicitly listed in Table 7, and so on. A synthetic workload generator runs on a separate machine connected to the same Layer 2 switch to avoid competing for cluster resources. Each message contains a JSON payload with `device_id`, `ts`, `pm1`, `pm25`, `pm10`, `temp`, and `hum` fields-approximately 120 bytes. Each scenario runs for 60 seconds with a 30-second cooldown between tests. Eight scenarios vary client count (10 or 100) and target publish rate (100, 500, 1000, or 2000 msg/s total):

Scenario	Clients	Target (msg/s)
10c_100r	10	100
10c_500r	10	500
10c_1000r	10	1 000
10c_2000r	10	2 000
100c_100r	100	100
100c_500r	100	500
100c_1000r	100	1 000
100c_2000r	100	2 000

Table 6: Benchmark scenarios. The 1000 and 2000 msg/s scenarios were only achievable on the 5-node cluster; lower node counts saturate before reaching these targets.

Table 7: Cluster configurations tested, in order. Changes at each step are cumulative; everything not listed carried over from the previous configuration.

Config	Nodes	K3s	Changes from previous
1-node	1×Pi 4B	v1.34.3	Baseline. Python consumer. MQTT QoS 2.
2-node	2×Pi 4B	v1.34.3	Added one worker; pipeline and configuration unchanged.
3-node	3×Pi 4B	v1.34.3	Added second worker. Added `msg_id` label to all VictoriaMetrics metrics.
5-node	5×Pi 4B	v1.35.4	Added two more workers. Removed `msg_id` label (series per device: unbounded → 7). Compensated publisher timing drift. Fixed DNS egress NetworkPolicy. Aligned all nodes to K3s v1.35.4. Switched MQTT QoS 2 to QoS 0.

Latency measurement differs between configurations. On 1-, 2-, and 3-node clusters, latency is measured end-to-end from sensor publication timestamp to NATS consumer exit timestamp. On the 5-node cluster, latency is measured as inter-arrival time between consecutive messages at the consumer. The two methods are not directly comparable: end-to-end latency includes pipeline queueing, whereas inter-arrival time reflects throughput smoothness at the consumer. Results are presented separately and interpreted accordingly.

9.1.1 Incremental Engineering Fixes

Five engineering changes were applied incrementally across configurations and are necessary context for interpreting the results:

1. **Cardinality fix**: The 3-node configuration attached a `msg_id` label to each VictoriaMetrics metric, creating an unbounded number of unique time series (one per message). This caused VictoriaMetrics memory growth proportional to message count. The 5-node configuration removes `msg_id` from labels, bounding cardinality to 7 series per device regardless of message volume.

2. **Publisher timing correction**: Earlier configurations used a naive `usleep` loop for pacing, which accumulated drift under load and delivered fewer messages than the target rate. The 5-node publisher uses `clock_gettime`-compensated sleep, delivering the target rate with $<$1% deviation (e.g., 49.6 msg/s for a 50 msg/s per-device target).

3. **Network policy fix**: A DNS egress block in the Kubernetes network policy prevented the consumer from resolving the VictoriaMetrics service name. The consumer namespace's egress rule allowed DNS only to `kube-system`, but some lookups routed through a different namespace depending on the K3s version; the fix allowed DNS egress to any namespace on port 53. Without it the consumer failed silently with connection timeouts, and the bug was invisible during functional testing because DNS responses were cached across restarts.

4. **K3s version alignment**: Nodes had been upgraded to K3s at different times, leaving a mix of v1.34.3+k3s1 and v1.35.4+k3s1. The skew caused intermittent ClusterIP resolution failures because the kube-proxy iptables rules differed between versions. All nodes were aligned to v1.35.4+k3s1 in a single coordinated Ansible run.

5. **QoS downgrade**: MQTT QoS was switched from 2 to 0 for the 5-node runs, removing the four-way handshake that serialized per-connection delivery. As Section 9.5 shows (and the May 13 confirming run in Table 14 demonstrates), this protocol change-not the added nodes-was the dominant lever on the throughput ceiling.

Latency is measured by embedding a publication timestamp in the message payload and recording the VictoriaMetrics write confirmation timestamp; the difference is the end-to-end latency for that message (for the 1/2/3-node configurations).

9.2 Benchmark Campaign and Timeline

The benchmark comprises 184 runs spanning 62 days (March 19 to May 19, 2026). Table 8 summarizes the phases. Of the 184 runs, roughly 107 produced no usable data-a 58% waste rate driven by pipeline configuration failures in the early three-node phase and infrastructure breaks in May. Because every configuration was a tagged commit (Section 8.9), each phase boundary corresponds to a specific set of file changes rather than to ambient drift.

Table 8: Benchmark timeline by phase over the 62-day campaign (184 runs total).

Period	Phase	Key event
Mar 19-21	1-node baseline	3 scenarios completed
Apr 7-10	2-node initial	Good on Apr 7-8; degraded by Apr 9
Apr 15-27	3-node tuning	59$\times$ improvement within phase
May 13	5-node early	Accidental 2-node run; first QoS 0 test
May 14 12:14	5-node gold	Best run: 100% efficiency, all scenarios
May 14 13:34	DNS break	DNS egress block: P99 $\rightarrow$ 75.7 million ms
May 18	Degraded	50-70% throughput drop (SD-card wear)
May 19	Incomplete	VM cleared before run; 1 msg/scenario

The 80-minute window between the gold-standard run and the DNS failure (May 14, 12:14 vs. 13:34) is worth noting: nothing intentional changed in that gap. The DNS egress policy had been causing intermittent failures before it failed completely, and it was invisible during functional testing because DNS responses were cached across restarts.

9.3 Throughput Results

Table 9 reports achieved throughput across all four cluster configurations and all tested scenarios. Dashes indicate scenarios that were not run or where the cluster failed to sustain any measurable throughput.

Scenario	Target (msg/s)	1-Node (msg/s)	2-Node (msg/s)	3-Node (msg/s)	5-Node (msg/s)
10c_100r	100	-	48	100	100
10c_500r	500	24	39	458	496
10c_1000r	1 000	-	-	-	989
10c_2000r	2 000	-	-	-	1 966
100c_100r	100	23	44	96	100
100c_500r	500	42	67	447	498
100c_1000r	1 000	-	-	-	995
100c_2000r	2 000	-	-	-	1 839

Table 9: Achieved throughput (msg/s) across cluster sizes and scenarios. The 5-node cluster sustains 92-100% of target across all eight scenarios; the 1-node cluster reaches at most 4.7-22.9% efficiency.

Table 10 shows the corresponding efficiency (achieved as a percentage of target rate).

Scenario	1-Node	2-Node	3-Node	5-Node
10c_100r	-	48.5%	100.3%	99.7%
10c_500r	4.7%	7.9%	91.6%	99.2%
10c_1000r	-	-	-	98.9%
10c_2000r	-	-	-	98.3%
100c_100r	22.9%	43.7%	96.2%	100.0%
100c_500r	8.3%	13.4%	89.4%	99.7%
100c_1000r	-	-	-	99.5%
100c_2000r	-	-	-	92.0%

Table 10: Pipeline efficiency as percentage of target rate. The 5-node cluster maintains above 92% efficiency across all scenarios, including 2000 msg/s under 100 concurrent clients.

Three findings stand out from this comparison. The 1- and 2-node clusters are throughput-limited at roughly 24-67 msg/s regardless of target rate. The 1-node cluster with 100 clients at 500 msg/s target achieves only 42 msg/s (8.3% efficiency) and drops 91.7% of messages. The 2-node cluster improves throughput by roughly 2$\times$ over 1-node but still saturates well below 500 msg/s. These ceilings are attributable to QoS 2 protocol serialization combined with insufficient consumer throughput on the Go consumer implementation. The 3-node cluster breaks through the 50 msg/s ceiling decisively-reaching 447-458 msg/s at 500 msg/s target (89-92% efficiency)-but only after switching to the Python consumer and adding the third node. The 3-node configuration still has a cardinality problem from `msg_id` labels, which causes VictoriaMetrics memory to grow unboundedly and limits sustained operation. The 5-node cluster, with the cardinality fix, publisher timing correction, network policy fix, K3s version alignment, and the switch to QoS 0 applied, achieves 92-100% efficiency across all eight scenarios including 2000 msg/s. Zero messages are dropped in any 5-node scenario. Throughput scales near-linearly with target rate up to 1966 msg/s, after which a modest drop to 92% efficiency at 100c_2000r suggests the beginning of genuine resource saturation at the highest load.

9.4 Latency Results

Table 11 reports end-to-end latency (sensor timestamp to NATS consumer exit) for the 1- and 2-node configurations. Table 12 reports inter-arrival latency at the 5-node consumer. The two metrics are not directly comparable but together characterise the improvement in pipeline behaviour across generations.

Config	Scenario	Mean (ms)	P95 (ms)	P99 (ms)	Drop Rate
1-node	10c_500r	12 759	21 589	23 895	95.3%
1-node	100c_100r	360 627	761 823	895 192	77.1%
1-node	100c_500r	66 989 874	124 847 602	129 839 474	91.7%
2-node	10c_100r	157	226	330	-
2-node	10c_500r	157	232	333	-
2-node	100c_100r	179	315	624	-
2-node	100c_500r	598	2 864	3 062	-

Table 11: End-to-end latency for 1- and 2-node configurations. The 1-node cluster under high load accumulates latency in the tens of minutes due to message queuing; the 2-node cluster reduces this to sub-second for most scenarios.

Scenario	Mean (ms)	P50 (ms)	P90 (ms)	P95 (ms)	P99 (ms)
10c_100r	11.0	5	56	57	59
10c_500r	2.9	2	6	9	9
10c_1000r	1.9	1	4	5	6
10c_2000r	1.3	1	2	2	3
100c_100r	14.8	2	6	10	843
100c_500r	3.2	1	4	7	72
100c_1000r	1.7	1	3	3	5
100c_2000r	1.3	1	2	2	3

Table 12: Inter-arrival latency at the 5-node consumer (time between consecutive messages received). Mean latency decreases as target rate increases because higher throughput reduces inter-message gaps. P99 latency at 100c_100r (843 ms) reflects occasional bursts when 100 clients deliver at low per-client rates.

The progression across cluster generations is striking. The 1-node cluster under 100c_500r accumulates a P99 latency of 129 seconds-messages are backed up in the queue faster than they are consumed, and by the time they are processed the pipeline is minutes behind. The 2-node cluster reduces P99 to 3 seconds at the same scenario. The 5-node cluster at 100c_500r delivers P99 of 72 ms, a reduction of six orders of magnitude from the 1-node baseline. The 5-node mean inter-arrival latency decreases as target rate increases-from 11 ms at 100 msg/s to 1.3 ms at 2000 msg/s-which is arithmetically expected: higher throughput means less time between consecutive messages. The P99 spike at 100c_100r (843 ms) is a different phenomenon: with 100 clients each publishing at only 1 msg/s, messages arrive in irregular bursts depending on client timing alignment, causing occasional gaps at the consumer that inflate tail latency without affecting mean throughput. The mean end-to-end latency budget is not dominated by any single pipeline component. Table 13 decom-

poses per-stage latency on the 5-node gold-standard run using the in-message timestamp chain (publisher → EMQX → Benthos → NATS → consumer). Every individual hop completes in under 10 ms at P99; the dominant contribution to the end-to-end figure is the deliberate 2-second batch-accumulation window in the Python consumer, which trades a small constant latency for a large reduction in the number of HTTP requests to VictoriaMetrics. For comparison, the equivalent 1-node breakdown showed P99 of 1 000-2 000 ms in the NATS-to-consumer stage alone and 500 ms on the consumer-to-VM write under CPU saturation; those stages now complete in 3 and 8 ms respectively.

Table 13: Per-stage latency on the 5-node gold-standard run (10c_500r, May 14). Each individual stage stays under 10 ms at P99; the end-to-end total is dominated by the 2-second batch-accumulation window in the Python consumer.

Stage	**P50 (ms)**	**P99 (ms)**
Publisher → EMQX entry	<1	3
EMQX processing	<1	<1
EMQX → Benthos entry	<1	2
Benthos processing	<1	<1
Benthos → NATS	<1	<1
NATS → consumer	<1	3
Consumer → VM write	2	8
Total end-to-end	48	2 075

9.5 Bottleneck Analysis

The 1- and 2-node throughput ceilings have the same root cause: MQTT QoS 2 serialization limits per-connection throughput to $1000\,\text{ms}/20\,\text{ms} = 50$ msg/s, and the consumer on these configurations cannot drain the NATS queue fast enough to prevent backlog accumulation. The result is the characteristic pattern of a hard throughput ceiling independent of target rate, combined with exponentially growing latency as queue depth increases. That the ceiling was identical at 10 and 100 clients confirms it was a per-connection serialization constraint, not a resource limit: CPU and memory stayed below 60% on both nodes throughout. The 3-node improvement to 89-92% efficiency at 500 msg/s is attributable to two changes acting together: adding a third node distributes pod scheduling load, and switching to the Python consumer changes the downstream write path. However, the cardinality explosion from `msg_id` labels placed unbounded pressure on VictoriaMetrics, which would eventually exhaust memory during sustained operation. The 5-node cluster resolves every identified bottleneck:

- **Protocol serialization**: The QoS 2 four-way handshake serialized per-connection delivery. The 5-node runs switched to QoS 0, removing the handshake entirely; batch processing in the Python consumer then sustained the full target rate. The headline 5-node figures were obtained under QoS 0.

- **Cardinality**: Removing `msg_id` from metric labels bounds VictoriaMetrics to 7 series per device. A fleet of 100 devices produces 700 active time series-well within the configured limit of 3 000 000 and sustainable indefinitely.
- **Publisher drift**: The compensated publisher delivers the target rate to within 1%. At 50 msg/s per device the measured rate is 49.6 msg/s-the earlier apparent ceiling of exactly 50 msg/s in 2-node results was partly a publisher artefact.
- **DNS resolution**: The network-policy egress fix allows the consumer to resolve VictoriaMetrics by service name rather than requiring a hard-coded IP, which was silently causing connection failures in earlier 3-node runs.
- **Version skew**: Aligning all nodes to K3s v1.35.4+k3s1 removed the intermittent ClusterIP resolution failures caused by mismatched kube-proxy iptables rules.

A single data point isolates the protocol effect from the node-count effect. On May 13, before the five-node configuration was fully verified, the benchmark ran accidentally on only two active nodes-but with QoS 0 and the corrected pipeline. It reached 99.8-101.3% efficiency across all four sub-2000 msg/s scenarios (Table 14). Two correctly configured boards matched the five-node result, confirming that the protocol and configuration changes, not the added hardware, lifted the ceiling.

Table 14: May 13 accidental run on 2 active nodes with QoS 0 and the fixed pipeline. Compared with the April 2-node results (Table 11), the protocol and pipeline changes-not the node count-account for the improvement.

Scenario	Target	Achieved	Eff.	P99 (ms)
10c_100r	100	99.8	99.8%	135
10c_500r	500	494.0	98.8%	1 468
100c_100r	100	101.3	101.3%	282
100c_500r	500	498.8	99.8%	1 003

The remaining efficiency gap at 100c_2000r (92% vs. the 99%+ seen at lower rates) is the only scenario where genuine resource pressure is visible. At 2000 msg/s across 100 clients, each client publishes at 20 msg/s, and the aggregate write rate to VictoriaMetrics is approximately 10 000 samples/s (2000 messages $\times$ 5 metrics per message). This is within VictoriaMetrics's documented capacity on ARM64 hardware, but the combination of EMQX broker processing, NATS durability writes, and connection-layer overhead from 100 concurrent connections begins to create occasional write stalls that manifest as the 8% drop from target. This is a hardware limit, not an architectural one: it represents the actual ceiling of this specific five-node cluster under maximum load.

9.6 Run Variability, Resource Usage, and Storage Endurance

The gold-standard run was not the whole story. Table 15 shows per-run results for the 10c_500r scenario across all May 5-node runs: throughput and P99 varied by two orders of magnitude on the same cluster. The May 14 13:34 collapse is explained by the DNS egress block (Section 9.1.1); the May 18 degradation-roughly 66% of peak throughput with no configuration change-is most consistent with MicroSD write-endurance limits.

Table 15: 10c_500r results across all May 5-node runs, showing the impact of the DNS break and subsequent SD-card degradation.

Date / time	State	Throughput	P99 (ms)
May 13 16:02	2-node, QoS 0 (accidental)	494 msg/s	1 468
May 14 12:14	5-node gold	496 msg/s	2 075
May 14 13:34	DNS break	0.4 msg/s	75 679 000
May 14 13:59	Partial recovery	250 msg/s	1 444
May 18 12:30	Degraded	169 msg/s	1 988
May 18 16:06	Further degraded	160 msg/s	8 384
May 19 13:09	Incomplete (VM cleared)	111 msg/s	9

Table 16 reports master-node resource usage at idle versus peak. While the worker nodes had headroom, the master-which hosted EMQX, Benthos, VictoriaMetrics, Prometheus, Loki, Tempo, ArgoCD, and CoreDNS simultaneously-reached 91% CPU and a 1-minute load average of 15.8 on four cores, indicating a run queue that was backed up throughout the benchmark. Memory, however, never exceeded 56% in any configuration: memory was never the constraint.

Table 16: Master-node resource usage: idle vs. peak benchmark (5-node, 2000 msg/s). Load-average figures are for the BCM2711 quad-core processor.

Metric	Idle	Peak
CPU usage	15-25%	91%
Load average (1 min)	1.2	15.81
Memory used	35-40%	56%
Context switches/s	$\sim$500 K	$\sim$2 M
Disk writes (total)	$\sim$200 MB/day	192 GB

The 192 GB written to MicroSD over the 62-day campaign is a durability concern in its own right. Typical MicroSD endurance is 100-300 GB of total writes, and the progressive throughput degradation observed on May 18 is consistent with a card approaching its write limit. For sustained deployments, a USB 3.0 SSD (sub-millisecond fsync, far higher endurance) would remove both the tail-latency variance and the wear concern at roughly €25 per node.

10 PM2.5 Forecasting

10.1 Motivation and Problem Formulation

Monitoring PM2.5 concentrations in real time answers one question: what is the air quality right now? A separate and more actionable question is: what will it be tomorrow, or in ten days? Forecasts enable public health advisories, school and activity scheduling, and pre-emptive interventions such as traffic restriction on high-risk days. The spatial estimation model described in Section 4 produces a current field from sparse measurements. The forecasting model described here predicts the future trajectory of PM2.5 at a specific location from its recent history combined with meteorological context. The task is a univariate-output, multivariate-input time series regression problem: given a sequence of $L = 14$ consecutive daily observations of PM2.5, co-pollutants, meteorological variables, and engineered temporal features, predict the PM2.5 concentration on day $L + 1$. The model is trained and evaluated on data from the MOEPP (Ministry of Environment and Physical Planning) monitoring network in Tetovo, accessed via AQICN [33], which provides daily aggregated readings for PM2.5, PM_{10}, O_3, NO_2, SO_2, and CO.

10.2 Dataset

10.2.1 MOEPP Air Quality Data

The primary data source is the MOEPP air quality dataset for Tetovo, accessed via AQICN [33]. It provides daily measurements of six pollutant species: PM2.5 (the forecast target), PM_{10}, O_3, NO_2, SO_2, and CO. These readings come from a certified reference instrument operated by the Ministry of Environment and Physical Planning of North Macedonia, which distinguishes this dataset from the low-cost sensor network described elsewhere in this thesis. Reference instruments are subject to regular calibration against national standards and do not exhibit the humidity-induced measurement bias that affects optical particle counters. Figure 11 shows the PM2.5 and PM_{10} time series. The dominant feature is the winter seasonality: concentrations from November through March are substantially higher than the rest of the year, driven by residential wood combustion for heating under thermal inversion conditions. This is the signal the model must learn. Outside the heating season, PM2.5 drops to near-background levels and varies mainly with wind events and regional transport.

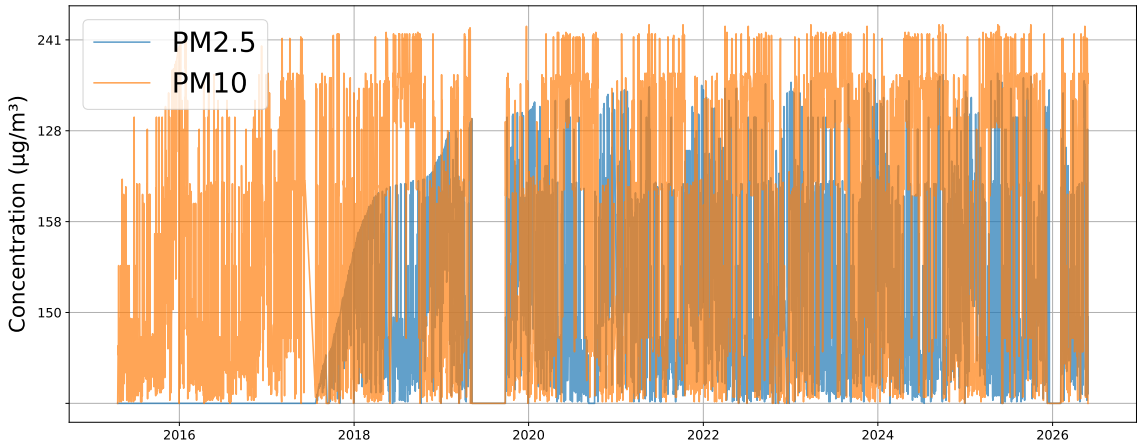


Figure 11: PM2.5 and PM_{10} daily concentrations from the MOEPP reference station in Tetovo. Winter peaks are driven by the combination of residential wood burning and valley-confined thermal inversions. The PM2.5/PM_{10} ratio is consistently high, indicating fine-mode combustion aerosol rather than coarse crustal dust.

Figure 12 shows the four gaseous co-pollutants. O_3 exhibits an inverse seasonal pattern-elevated in summer due to photochemical production under intense solar radiation-while NO_2, SO_2, and CO peak in winter alongside the particulate matter, consistent with combustion as the common source. These anti-correlations and co-variations are informative for the model: O_3 acts as a proxy for summer photochemical conditions, while the combustion gases reinforce the heating-season signal.



Figure 12: Time series of gaseous co-pollutants from the MOEPP dataset. O_3 peaks in summer; NO_2, SO_2, and CO peak in winter alongside PM2.5. These species provide complementary information about atmospheric conditions and emission sources that the model can exploit.

10.2.2 Meteorological Data

The on-site meteorological sensors in the low-cost network have known data quality problems-calibration drift, systematic gaps during cold periods when the housing condenses, and inconsistencies that appear during extreme weather events. Using this data as model input would introduce noise that is correlated with the periods of highest PM2.5 concentration, the worst possible place to have measurement errors in a forecasting context. Instead, meteorological inputs are sourced from Meteostat [34], an open-access platform that aggregates observations from official meteorological stations. Hourly records for the nearest Tetovo station are downloaded and aggregated to daily summaries. Figure 13 shows the raw hourly weather series; Figure 14 shows the distribution of temperature, humidity, and wind speed, confirming the expected seasonal ranges for a continental valley climate.

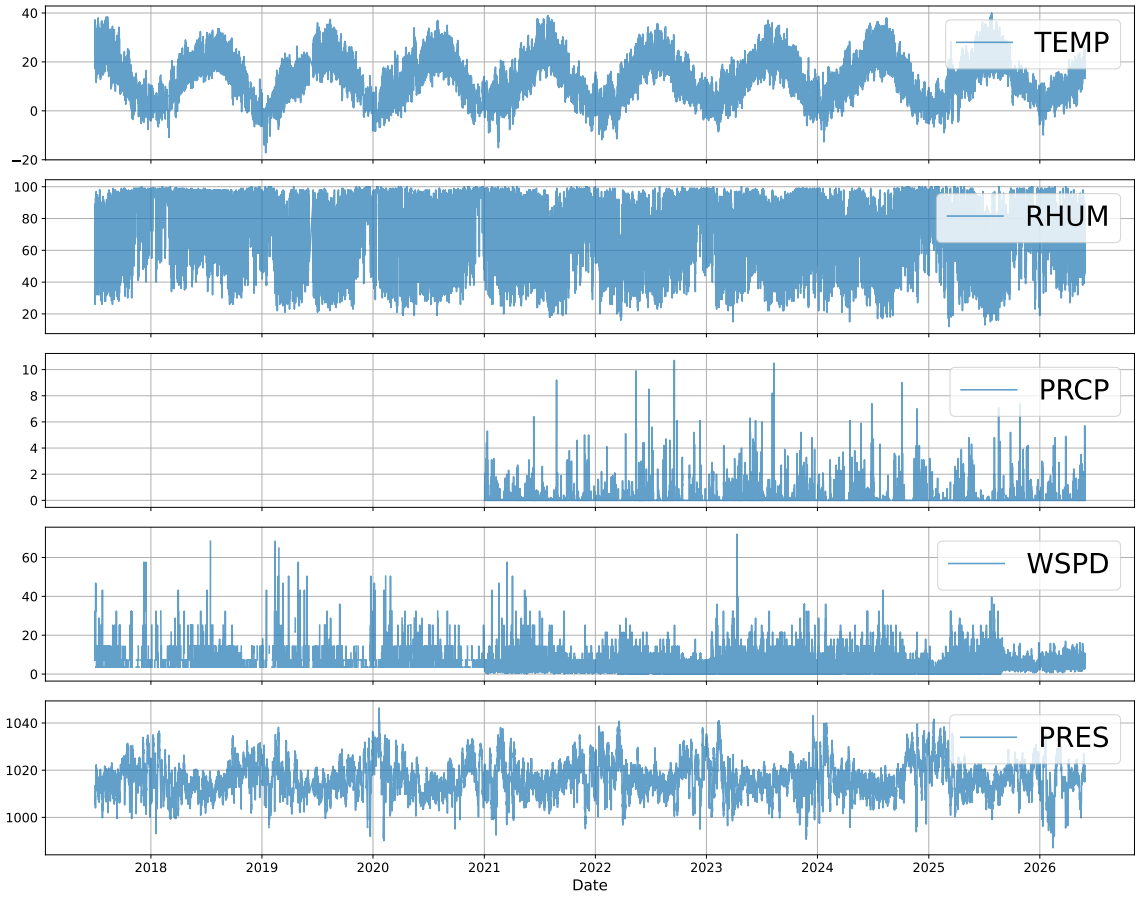


Figure 13: Raw hourly weather observations from the Meteostat station nearest to Tetovo: temperature, relative humidity, precipitation, wind speed, and atmospheric pressure. Hourly data is aggregated to daily summaries before merging with the air quality dataset.

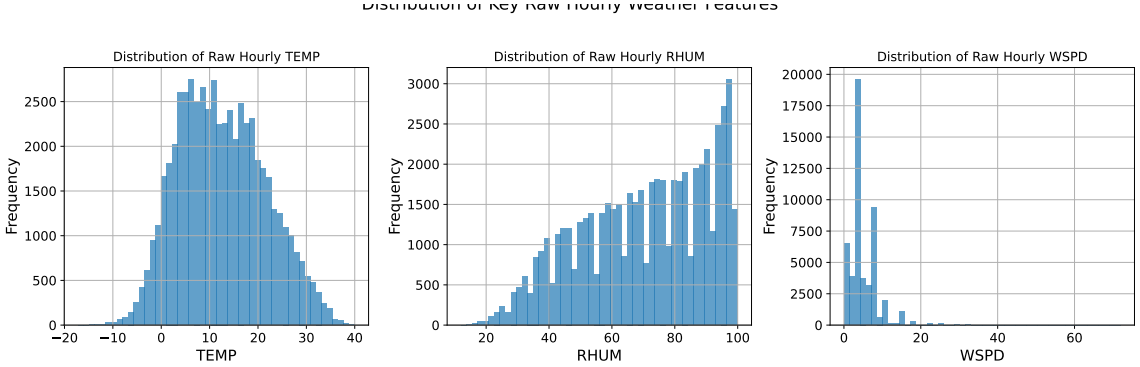


Figure 14: Distributions of key hourly weather features. Temperature spans roughly -10 to 35°C; humidity is bimodal with modes near 50% and 85%; wind speed is right-skewed with most hours below 10 km/h, consistent with the sheltered valley location.

The eight daily weather features used as model inputs are: `temp_mean`, `temp_min`, `temp_max` (°C); `rhum_mean` (%); `prcp_sum` (mm); `wspd_mean`, `wspd_max` (km/h); and `pres_mean` (hPa). Temperature and wind speed are the most physically important predictors: cold temperatures are associated with increased heating demand and with the formation of temperature inversions that trap pollutants, while high wind speed promotes mixing and dispersion. Atmospheric pressure complements wind speed: persistent high pressure suppresses vertical mixing and is frequently the meteorological precursor to severe PM2.5 episodes in Tetovo.

10.3 Data Pre-processing

10.3.1 Missing Value Imputation

Both datasets contain missing values from instrument outages, transmission failures, and quality-control filtering. Because the forecasting model requires a continuous daily index (sequences cannot span gaps without introducing undefined inputs), missing values must be filled rather than dropped. The imputation pipeline applies three steps in sequence. First, time-based linear interpolation fills gaps by fitting a straight line between the nearest valid values on either side, weighted by temporal distance. This preserves the local trend and is more appropriate than constant fills for a series with strong seasonal patterns-filling a winter gap with the annual mean would introduce a large artificial step in the series. Second, any values still missing after interpolation are filled forward from the last valid observation. Third, any values at the very start of the series (where forward fill has no preceding value) are filled backward. Mean or median imputation was considered and rejected: both assume the variable is stationary-that its expected value does not change over time-which is clearly false for a series with factor-of-ten amplitude differences between summer and winter. Figure 15 shows a zoomed view of the imputation effect across all six pollutant species. Gaps in the

raw data (shown as red dashed lines) are filled smoothly by the interpolated series (blue), with individual imputed points marked in green.

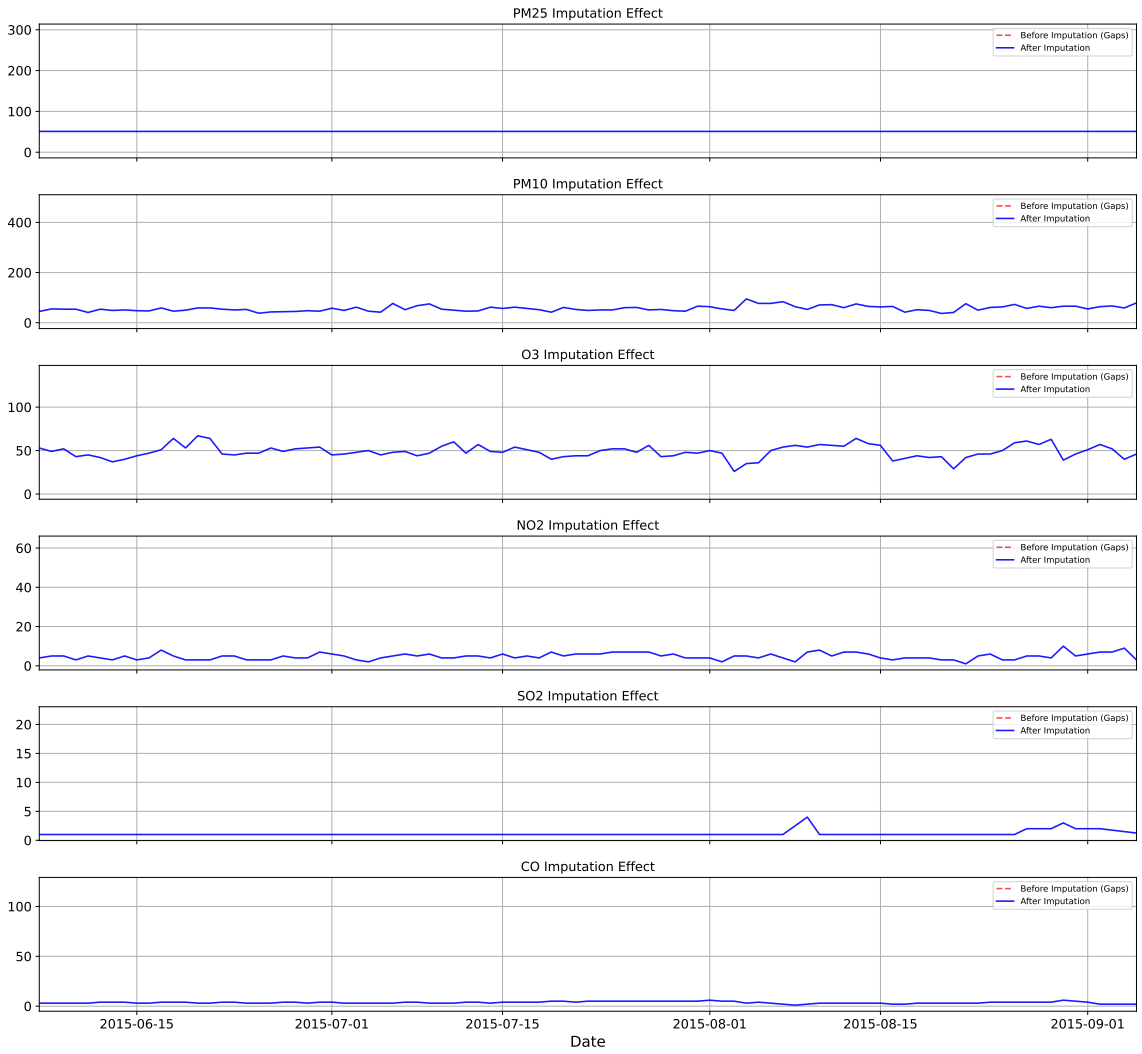


Figure 15: Zoomed view of the imputation effect on all six pollutant species. Red dashed lines show the raw data with gaps; blue lines show the imputed series; green crosses mark individually interpolated points. The interpolation preserves local trends without introducing step discontinuities.

Figure 16 shows the same comparison over the full time range, confirming that the imputed series retains the seasonal structure of the raw data.

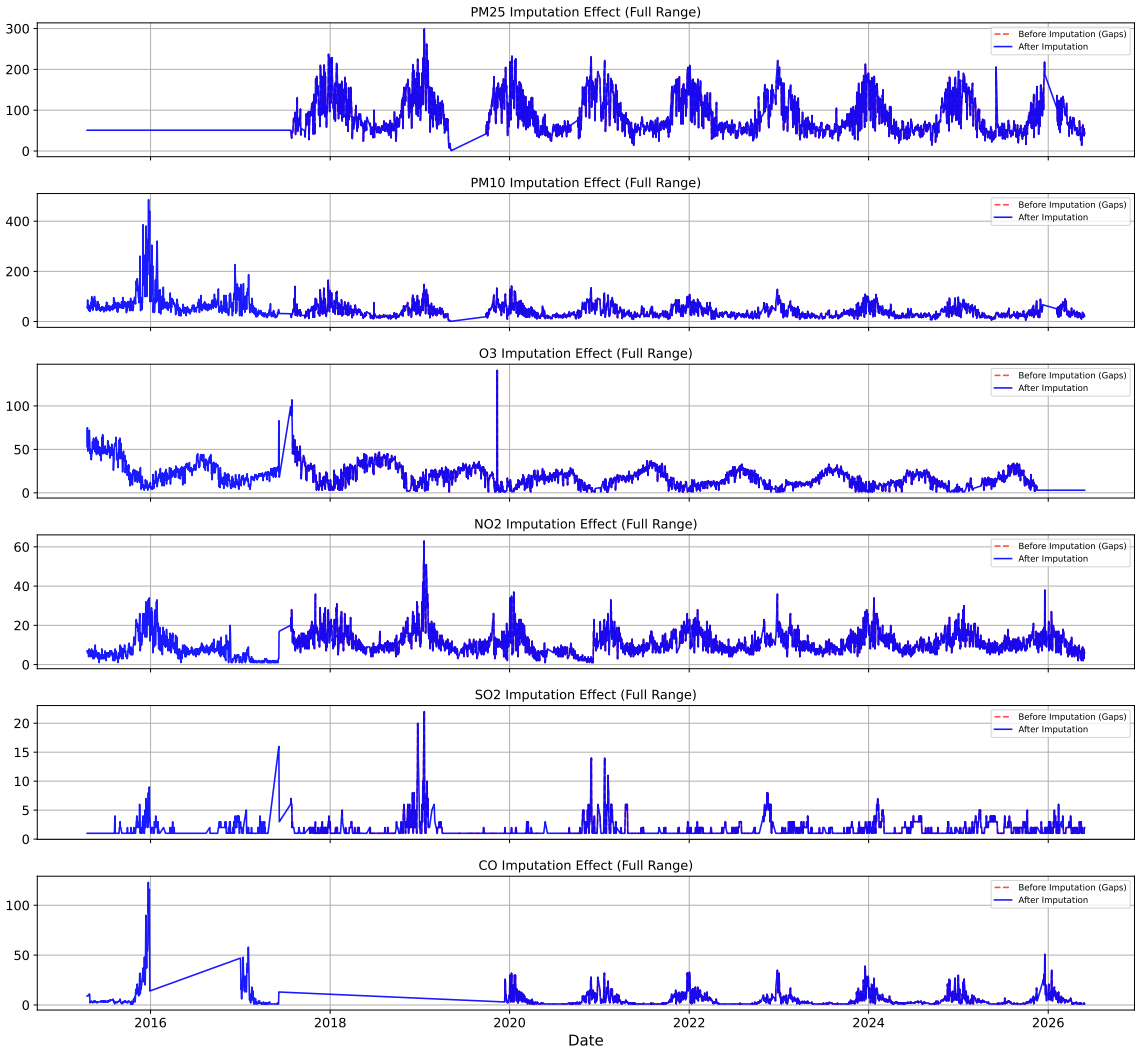


Figure 16: Full-range imputation effect across all pollutant species. The imputed series (blue) matches the raw data (red) where observations are available and interpolates smoothly across gaps. No spurious trends are introduced.

10.3.2 Merging and Alignment

The PM2.5 dataset and the aggregated daily weather data have different temporal coverage and are derived from independent sources. They are aligned using an inner join on the daily date index: only days with valid data in both sources are retained. Figure 17 shows the coverage overlap, making explicit which dates are lost in the merge.

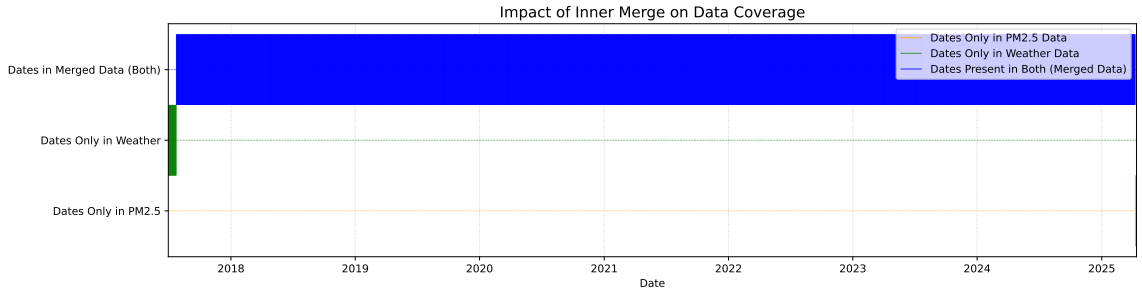


Figure 17: Impact of the inner join merge on date coverage. Orange lines mark dates present only in the air quality dataset; green lines mark dates present only in the weather dataset; blue lines mark dates retained in the merged dataset. The inner join is conservative: it discards any date without complete information from both sources.

After merging, the key variables are plotted together (Figure 18) to verify alignment and confirm that the seasonal PM2.5 pattern and the temperature cycle are phase-shifted as expected-PM2.5 peaks precisely when temperature is lowest.

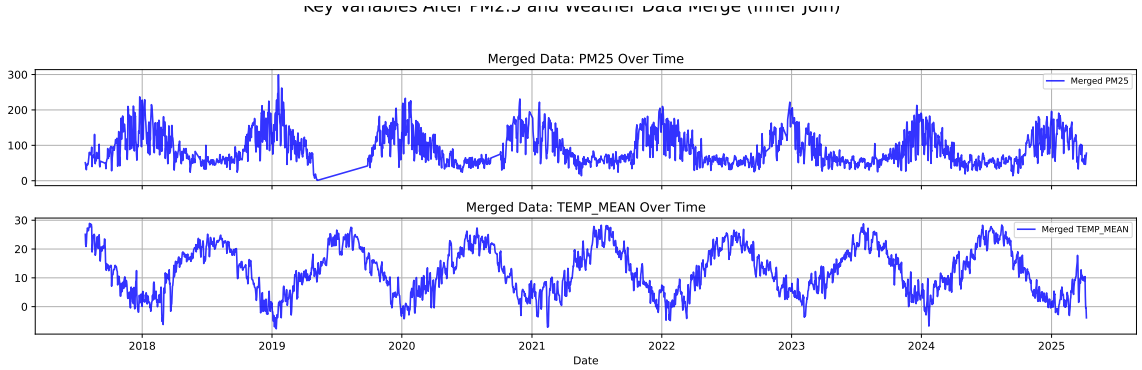


Figure 18: PM2.5 and mean temperature after merging. The anti-phase relationship is clear: PM2.5 peaks in winter when temperature is at its minimum. This is the primary signal the model must learn, and its presence in the merged dataset confirms the merge is correctly aligned.

10.4 Feature Engineering

Raw pollutant and meteorological measurements are not used directly. Three categories of engineered features are added to give the model access to temporal structure that the raw values alone cannot express.

10.4.1 Temporal Encoding

Calendar features-day of year, day of week, and month-are encoded using sine-cosine pairs rather than raw integers. Raw integer encoding imposes a false ordinal distance: it treats December 31 (day 365) and January 1 (day 1) as maximally distant in the feature space when they are in fact consecutive. Cyclical encoding maps each periodicity onto the unit circle, so nearby calendar positions are geometrically close regardless of where they fall in the calendar year:

$$\text{doy_sin} = \sin\left(\frac{2\pi d}{365.25}\right), \qquad \text{doy_cos} = \cos\left(\frac{2\pi d}{365.25}\right) \tag{8}$$

$$\text{dow_sin} = \sin\left(\frac{2\pi w}{7}\right), \qquad \text{dow_cos} = \cos\left(\frac{2\pi w}{7}\right) \tag{9}$$

$$\text{mon_sin} = \sin\left(\frac{2\pi (m-1)}{12}\right), \quad \text{mon_cos} = \cos\left(\frac{2\pi (m-1)}{12}\right) \tag{10}$$

where d is day of year (1-365), w is day of week (0-6), and m is month (1-12). Each sine-cosine pair jointly encodes position; no single component is meaningful in isolation.

10.4.2 Lag Features

For each pollutant and weather feature, lagged versions at delays $\tau \in \{1, 2, 3, 7\}$ days are appended:

$$x_{t-\tau}^{(k)} \quad \text{for feature } k,\ \tau \in \{1, 2, 3, 7\}$$

The 7-day lag specifically targets weekly periodicity in commuting and heating behaviour. All lags are computed with `shift(`τ`)` so that the lagged value at day t contains the observation from day $t - \tau$ only-no same-day information leaks into the lag features.

10.4.3 Rolling Statistics

Rolling window mean and standard deviation over $w \in \{3, 7\}$ days are computed for each pollutant and weather feature, with a one-day shift applied before rolling to prevent the current day's value from entering the window:

$$\mu_{w,t}^{(k)} = \frac{1}{w} \sum_{i=1}^{w} x_{t-i}^{(k)} \tag{11}$$

$$\sigma_{w,t}^{(k)} = \sqrt{\frac{1}{w-1} \sum_{i=1}^{w} \left(x_{t-i}^{(k)} - \mu_{w,t}^{(k)}\right)^2} \tag{12}$$

The rolling mean captures medium-term trend; the rolling standard deviation captures local volatility. High variance in recent PM2.5 or wind speed signals unstable atmospheric conditions associated with large day-to-day swings in concentration. After constructing lag and

rolling features, the first $\max(7, 7) = 7$ rows are dropped because they contain NaN values from the shifted windows. The final feature vector has 234 dimensions. Figure 19 illustrates representative engineered features: the original PM2.5 series alongside its 1-day lag; mean temperature with its 3-day rolling mean; and the cyclical day-of-year and day-of-week encodings.

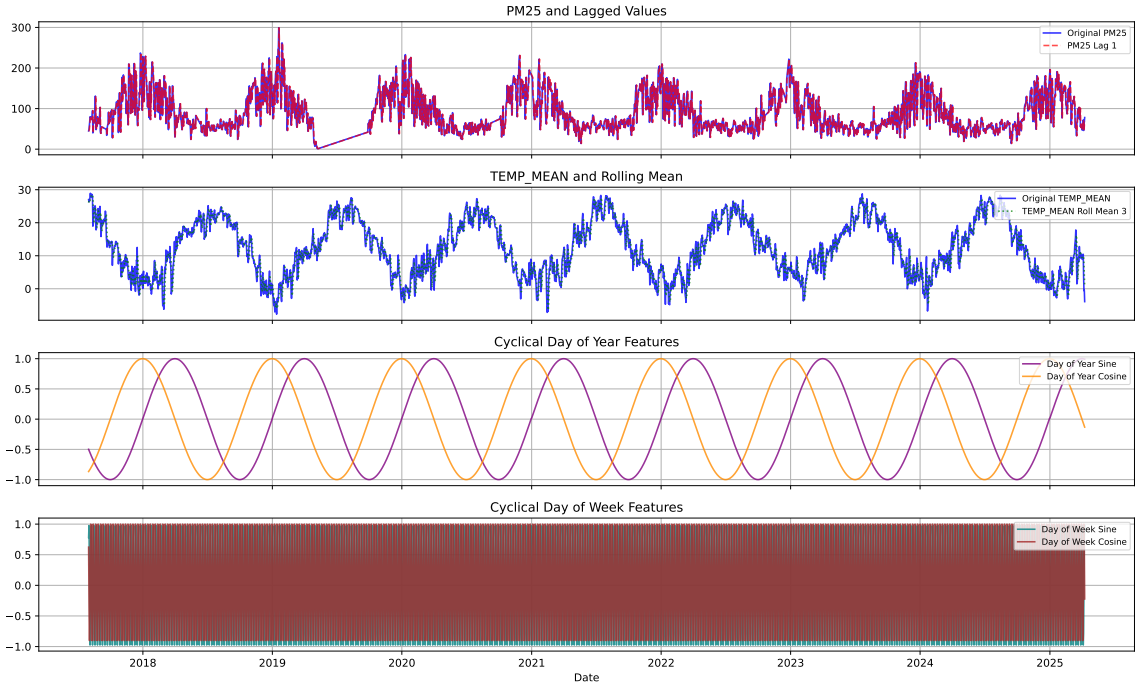


Figure 19: Examples of engineered features. Top: original PM2.5 and its 1-day lag (the lag shifts the series rightward by one day, providing the model with yesterday's reading as an explicit input). Second: mean temperature with its 3-day rolling mean, which smooths local noise while preserving the trend. Third and fourth: cyclical day-of-year and day-of-week sine-cosine pairs, which encode periodicity without imposing ordinal distances.

Figure 20 shows the effect of dropping the initial rows on the PM2.5 series length.

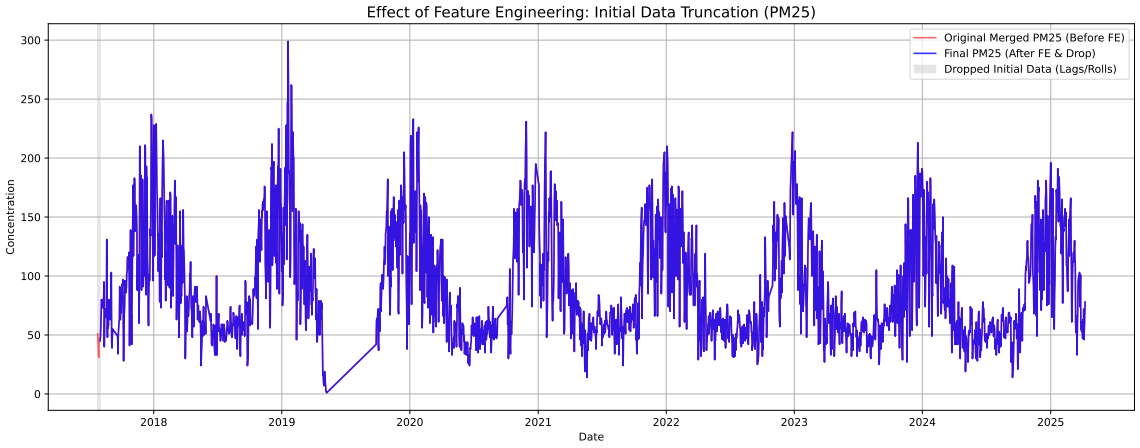


Figure 20: Effect of the initial row drop after feature engineering. The grey shaded region is removed from the dataset. This loss is small relative to the dataset length and unavoidable given the lag structure.

10.4.4 Sequence Construction

The feature-engineered dataframe is converted into fixed-length input-output pairs. A sliding window of $L = 14$ days produces input tensor $\mathbf{X}_t \in \mathbb{R}^{14\times 234}$ and target $y_t = \text{PM}_{2.5}(t+1)$:

$$(\mathbf{X}_t,\, y_t) = \big(\{x_{t-L+1}, \ldots, x_t\},\ \text{PM}_{2.5,\,t+1}\big)$$

The 14-day window was chosen based on the autocorrelation structure of the PM2.5 series: significant autocorrelation persists for approximately two weeks before dropping below a practically useful threshold. Figure 21 illustrates the windowing: a shaded 14-day input window leads to a single prediction for the following day.

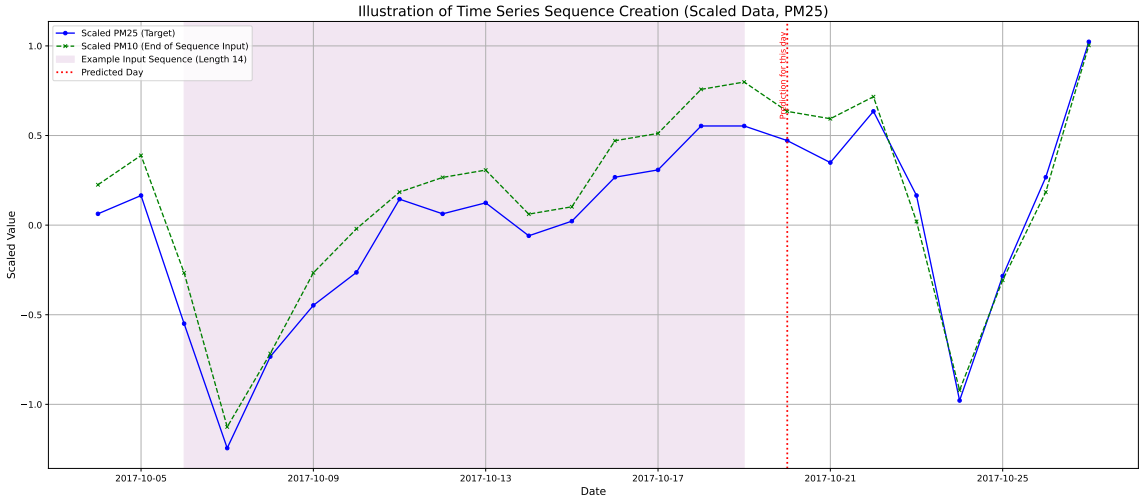


Figure 21: Illustration of the sliding window sequence construction. The purple shaded region is the 14-day input window for one training sample; the red dashed vertical line marks the day being predicted. The window slides forward by one day to produce the next sample.

10.5 Data Splitting and Normalization

The dataset is split chronologically into train (70%), validation (20%), and test (10%) sets. Random splitting would violate temporal ordering and allow information from later dates to appear in the training set, producing optimistic evaluation metrics. Chronological splitting ensures the model is always evaluated on data it has never seen. Feature scaling uses a `StandardScaler` fitted only on the training set and then applied to validation and test sets without refitting. A separate scaler is fitted for the PM2.5 target, allowing the model's output to be inverse-transformed independently for metric computation. Fitting the scaler on the full dataset before splitting would constitute data leakage, allowing test-set statistics to influence the normalization and bias evaluation metrics downward. Figure 22 shows the chronological split on the PM2.5 series.

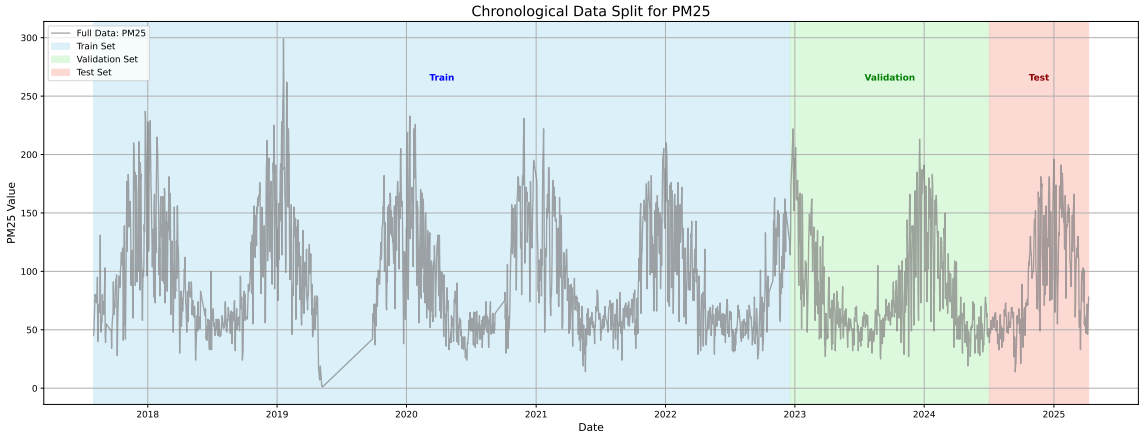


Figure 22: Chronological train-validation-test split on the PM2.5 series. The training set covers the majority of the heating seasons in the dataset; the test set covers August 2024 to April 2025, spanning one full seasonal cycle from summer baseline through winter peak and back.

Figures 23 and 24 show the distribution of PM2.5 and PM_{10} before and after scaling on the training set, confirming that the standardized features are centred at zero with unit standard deviation.

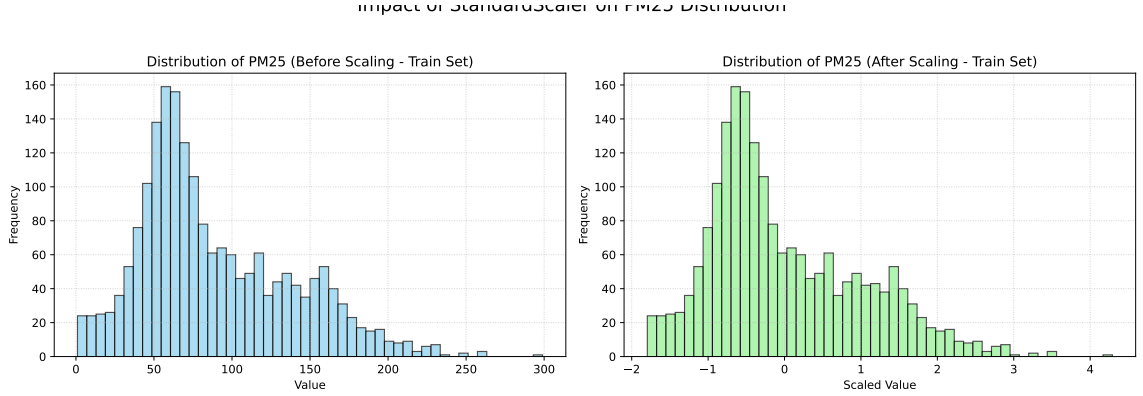


Figure 23: PM2.5 distribution before and after `StandardScaler` on the training set. The raw distribution is right-skewed due to winter peaks; the scaled distribution has mean 0 and standard deviation 1 but retains the same shape.

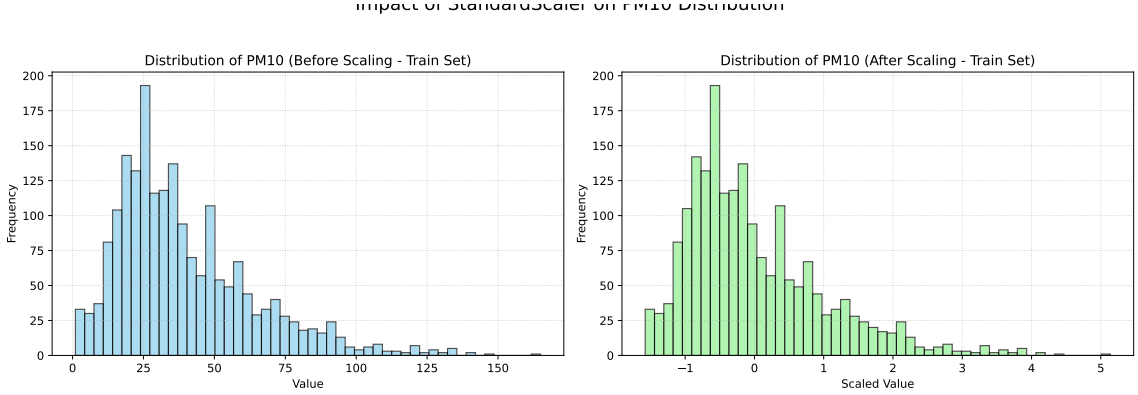


Figure 24: PM_{10} distribution before and after scaling. The strong positive correlation between PM_{10} and PM2.5 means both distributions share a similar skewed shape.

10.6 Model Architecture: CNN-BiGRU

10.6.1 Design Rationale

The forecasting model combines a one-dimensional convolutional block with two stacked bidirectional Gated Recurrent Unit (BiGRU) layers. This combination exploits complementary strengths: the convolutional block extracts local temporal patterns efficiently through shared-weight filters, while the BiGRU layers model long-range sequential dependencies across the full input window. The bidirectional configuration allows each BiGRU to condition on both the forward and backward context within the sequence, which is valid during training on historical data (where the full sequence is available) and improves feature representation.

10.6.2 Convolutional Block

The input tensor $\mathbf{X}_t \in \mathbb{R}^{14\times 234}$ is processed by a one-dimensional causal convolution with $F_c = 64$ filters and kernel size $k = 4$:

$$h_i^{(\text{conv})} = \text{ReLU}\left(\sum_{j=0}^{k-1} W_j \cdot x_{i-j} + b\right), \quad i = 1, \ldots, L$$

Causal padding ensures that the convolution at time step i uses only inputs from time steps $\leq i$, which is necessary to prevent information from future time steps within the window from influencing earlier representations. Batch normalization is applied after each convolutional layer to stabilize gradient magnitudes and reduce internal covariate shift. Max pooling with pool size 2 reduces the temporal dimension, discarding redundant high-frequency variation. Dropout with rate $p = 0.245$ is applied before the recurrent layers.

10.6.3 Bidirectional BiGRU Layers

The BiGRU is a recurrent unit with two gating mechanisms: the reset gate $\mathbf{r}_t$ controls how much of the previous hidden state influences the candidate activation, and the update gate $\mathbf{z}_t$ controls how much of the candidate replaces the current hidden state:

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1} + \mathbf{b}_r) \tag{13}$$

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{b}_z) \tag{14}$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h (\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h) \tag{15}$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t \tag{16}$$

where σ is the sigmoid function and $\odot$ denotes elementwise multiplication. The BiGRU is a simplified variant of the LSTM that combines the forget and input gates into a single update gate, reducing parameter count while retaining the ability to model long-range dependencies. For daily time series at 14-day window length, this simplification is appropriate. Two BiGRU blocks are stacked. The first uses $G_1 = 70$ units per direction and returns the full hidden state sequence (required for input to the second layer). The second uses $G_2 = 64$ units per direction and returns only the final hidden state. Each block is bidirectional, so the output dimension of the first block is $2 \times 70 = 140$ per time step; the output of the second block is $2 \times 64 = 128$. Batch normalization and dropout are applied after each BiGRU block.

10.6.4 Dense Output Layers

The final BiGRU hidden state passes through two fully connected layers with ReLU activation and dropout, followed by a single linear output unit:

$$\hat{y}_t = \mathbf{w}_{\text{out}}^{\top} \, \text{Dense}_2(\text{Dropout}(\text{Dense}_1(\mathbf{h}_{\text{final}})))$$

The first dense layer has 96 units and the second has 90 units. These layers apply nonlinear transformations to the compressed sequence representation before projecting to the scalar PM2.5 prediction. The complete model architecture is illustrated in Figure 25.

The full model hyperparameter configuration is summarized in Table 17.

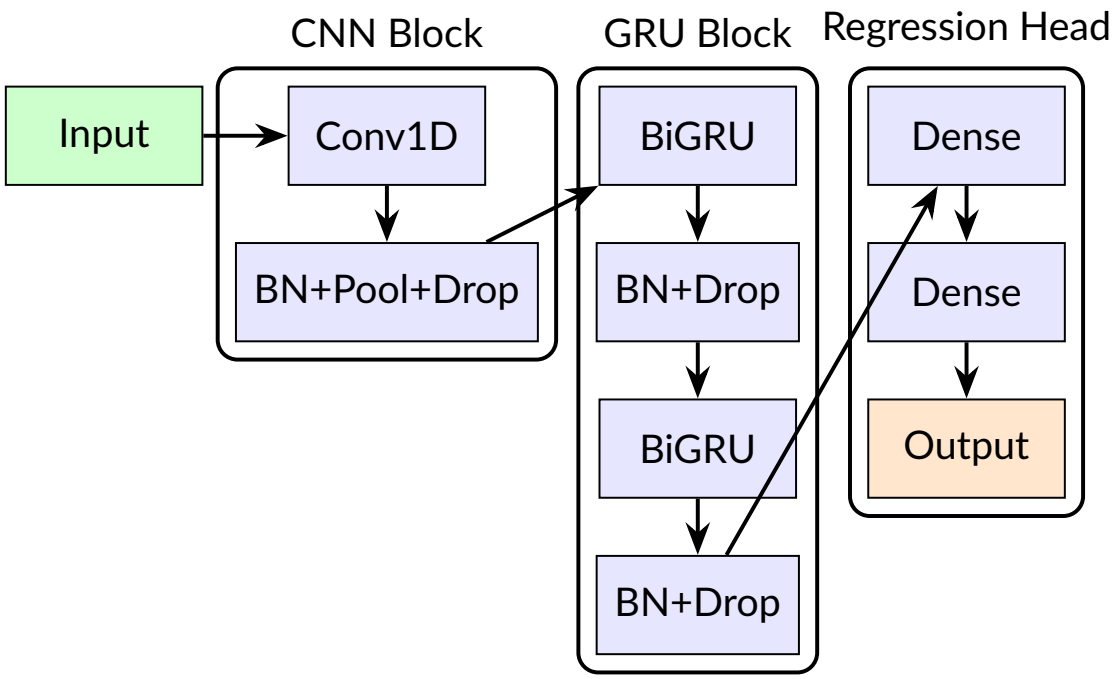


Figure 25: Architecture of the CNN-BiGRU model for PM2.5 forecasting. The input sequence passes through a causal convolutional block (batch norm, max pool, dropout), two stacked bidirectional BiGRU blocks, and finally two dense layers before the scalar output.

Parameter	Value
Input dimensions	14×234 (sequence length $\times$ features)
Output dimensions	1 (PM2.5, μg/m^3)
Conv1D filters	64
Conv1D kernel size	4
Conv1D padding	Causal
BiGRU units (layer 1)	70 per direction (bidirectional)
BiGRU units (layer 2)	64 per direction (bidirectional)
Dense units	96, then 90
Dropout rate	0.245
Batch normalization	After each conv and BiGRU block
Optimizer	AdamW ($\eta = 4 \times 10^{-4}$, weight decay $= 10^{-5}$)
Loss function	Mean Squared Error
Batch size	24
Max epochs	150
Early stopping patience	30 epochs (restore best weights)
LR reduction patience	15 epochs (factor 0.1, min 10^{-6})

Table 17: CNN-BiGRU model hyperparameters.

10.6.5 Federated Variant Architecture

The federated variant uses a different backbone: a CNN-BiLSTM with an additive attention mechanism as illustreted in 26, rather than the bidirectional BiGRU used in the centralised model. The convolutional block is identical. The recurrent layers use two stacked bidirectional LSTM units with hidden size H, followed by a Bahdanau et al. [35] style attention layer that computes a weighted sum over the encoder hidden states before the dense output head. This architecture was chosen for the federated setting because LSTM gating provides stronger gradient isolation across the longer effective sequences produced by FedAvg aggregation over heterogeneous clients.

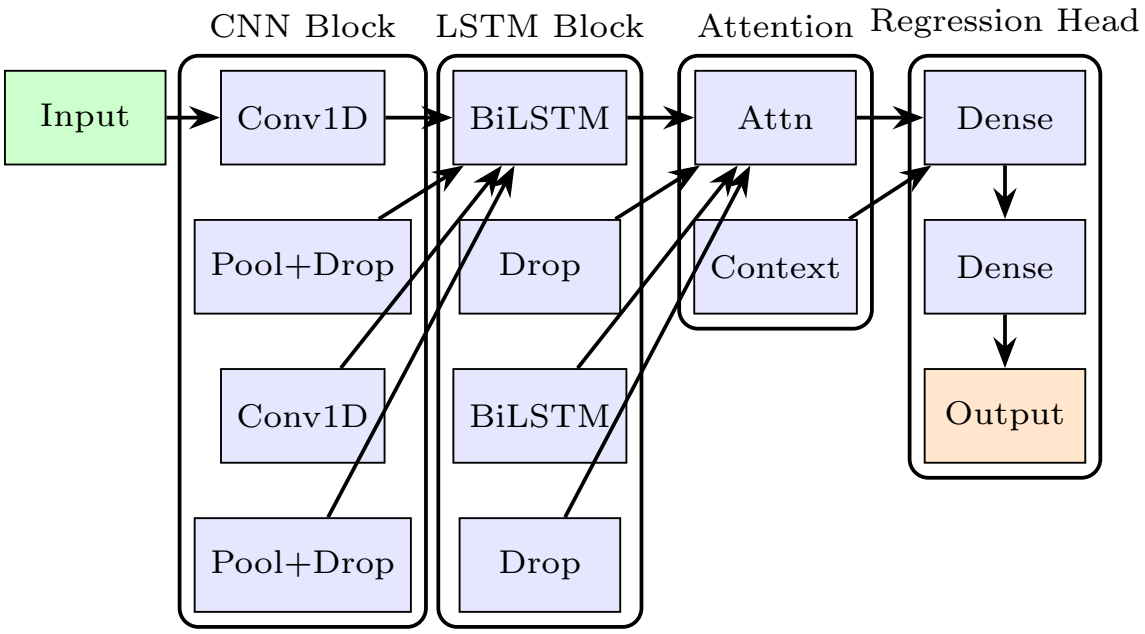


Figure 26: Architecture of the CNN-BiLSTM-AM model for PM2.5 forecasting. The input sequence passes through a causal convolutional block (batch norm, max pool, dropout), two stacked bidirectional LSTM blocks with additive attention, and finally two dense layers before the scalar output.

10.7 Training

The model is compiled with mean squared error as the loss function and trained using the AdamW optimizer [36], which adds decoupled weight decay regularization to the Adam update rule. Compared to L2 regularization added directly to the loss, decoupled weight decay applies regularization independently of the adaptive gradient scaling, which has been shown to improve generalization on sequence learning tasks. Two callbacks govern training dynamics. An early stopping callback monitors validation loss with a patience of 30 epochs and restores the weights from the best validation epoch at termination, ensuring the deployed model reflects the point of minimum generalization error rather than the last training epoch. A reduce-on-plateau callback reduces the learning rate by a factor of 10 when validation loss fails to improve for 15 consecutive epochs, allowing the optimizer to converge to a tighter minimum after large early steps have located the basin.

10.8 Federated Learning Variant

In addition to the centralized model trained on the MOEPP reference data, a federated variant was implemented to evaluate whether the architecture is suitable for privacy-preserving

training across the distributed low-cost sensor network. In the federated setting, the 9 sensor stations are treated as 9 independent clients. No raw data leaves the client; only model weight updates are communicated. Training follows the Federated Averaging (FedAvg) algorithm [32]. In each communication round: (1) the current global model weights are broadcast to all clients; (2) each client trains locally for a fixed number of epochs using AdamW on its private dataset; (3) the updated weight deltas are returned to the server; (4) the server aggregates them by weighted average, with each client's contribution weighted by its sample count; (5) the global model is updated and the next round begins. This process repeats until convergence. The federated approach is architecturally sound and has genuine operational advantages: it does not require centralizing raw sensor readings, which reduces bandwidth consumption and respects data ownership. Each new sensor station added to the network can participate in future training rounds without requiring the historical data of existing clients to be re-processed. The limitation identified here is not the federated architecture but the quality of the client data, as the discussion section explains.

10.9 Evaluation Metrics

Model performance is reported using three complementary metrics:

$$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n}|y_t - \hat{y}_t| \tag{17}$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2} \tag{18}$$

$$R^2 = 1 - \frac{\sum_{t=1}^{n}(y_t - \hat{y}_t)^2}{\sum_{t=1}^{n}(y_t - \bar{y})^2} \tag{19}$$

MAE reports the average absolute error in interpretable concentration units (μg/m^3). RMSE penalises large errors more heavily than MAE, making it sensitive to the tail of the error distribution. R^2 measures the fraction of variance in PM2.5 that the model explains; $R^2 = 1$ is perfect prediction, $R^2 = 0$ is no better than the mean prediction. Two application-level metrics are also computed: the fraction of predictions within ± 10 μg/m^3 (absolute tolerance), and the fraction within $\pm 15\%$ of the actual value on days where the concentration exceeds 1 μg/m^3.

10.10 Results

10.10.1 Feature Correlation Analysis

Before examining model predictions, the correlation structure of the input features is worth studying explicitly. Figure 27 shows the Pearson correlation matrix for PM2.5, the co-pollutants, and the meteorological variables.

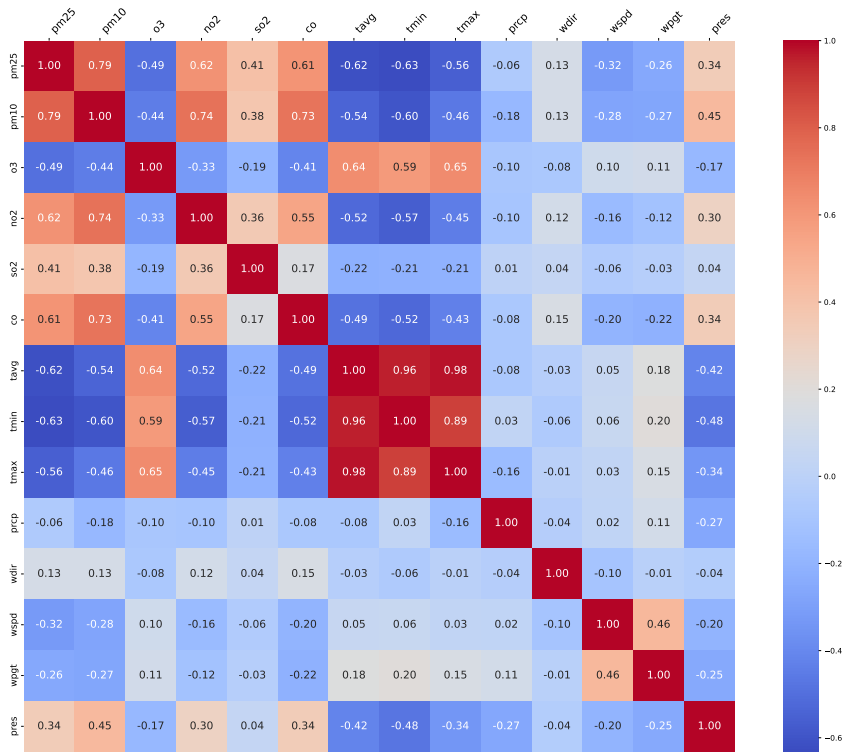


Figure 27: Pearson correlation matrix for PM2.5, co-pollutants, and meteorological variables. Values range from $+1$ (dark red, perfect positive correlation) to -1 (dark blue, perfect negative correlation). Strong inter-feature correlations confirm that the selected variables carry substantial joint predictive information about PM2.5.

Several correlations are directly interpretable in terms of the physical processes driving pollution in Tetovo. PM2.5 and PM_{10} are very strongly correlated ($r = 0.79$). This is expected: PM2.5 is a physical subset of PM_{10}, and both are generated by the same combustion sources. The high ratio of PM2.5 to PM_{10} confirms that the dominant aerosol in Tetovo is fine-mode combustion soot rather than coarse crustal dust. NO_2 and CO are strongly positively correlated with PM2.5 ($r \approx 0.62$ and 0.61 respectively). Both are primary combustion gases co-emitted with particulate matter from wood burning and traffic exhaust. When combustion activity is high-during cold, calm winter days-all three rise together. This co-variation makes NO_2 and CO useful predictive features even though the model is not explicitly given the emission inventory. O_3 is moderately negatively correlated with PM2.5 ($r \approx -0.49$). Ground-level ozone formation requires sunlight, high temperatures, and photochemical precursors. These conditions-summer, warm, sunny days-are the opposite of the conditions that produce high PM2.5, so the anti-correlation is physically meaningful rather than coincidental. Temperature variables (`temp_mean`, `temp_min`, `temp_max`) are strongly negatively correlated with PM2.5 ($r \approx -0.6$). This is the dominant meteorological relationship:

cold temperatures drive wood combustion for heating while simultaneously favouring the formation of thermal inversions that trap the resulting pollution near ground level. The three temperature variables are themselves very highly intercorrelated ($r > 0.96$), which creates multicollinearity. A linear model would be degraded by this; a deep learning model handles it without issue because the learned representations are not constrained to be linearly independent. Atmospheric pressure is positively correlated with PM2.5 ($r \approx 0.34$). Persistent high-pressure systems suppress vertical mixing, preventing pollutants from dispersing upward. Wind speed is negatively correlated ($r \approx -0.32$): stronger winds increase turbulent mixing and dilute near-surface concentrations. Both are physically expected and consistent with published literature on PM2.5 meteorological drivers in valley environments. The overall structure of the correlation matrix confirms two things. First, the selected features are genuinely informative about PM2.5-there are no features with near-zero correlation across the board. Second, the relationships are complex and non-linear (the negative correlation with temperature is driven by inversion dynamics, not a simple linear relationship), which justifies the choice of a deep learning architecture over a linear regression baseline.

10.10.2 Model Performance on MOEPP Test Set

The trained CNN-BiGRU model is evaluated on the held-out test set, which covers the period from August 2024 to April 2025-one full seasonal cycle spanning summer baseline through winter peak and back to spring. Table 18 reports the performance metrics.

Metric	Value	Interpretation
Mean Absolute Error (MAE)	12.85 μg/m^3	Average prediction error
Root Mean Squared Error (RMSE)	15.73 μg/m^3	Error penalising large deviations
R^2 (coefficient of determination)	0.87	87% of variance explained
Absolute accuracy ($\pm$10 μg/m^3)	-	Fraction within fixed tolerance
Relative accuracy ($\pm$15%)	-	Fraction within proportional tolerance

Table 18: CNN-BiGRU model performance on the MOEPP test set (August 2024 - April 2025). An R^2 of 0.87 indicates strong predictive capability across the full seasonal range.

An R^2 of 0.87 means the model accounts for 87% of the day-to-day variance in PM2.5 on unseen data. To put this in context: a model that always predicts the training-set mean would score $R^2 = 0$; a model that predicts perfectly would score $R^2 = 1$. A score of 0.87 represents a strong result for a daily environmental forecast in a complex valley environment, where a significant fraction of remaining variance is attributable to unobserved local factors-individual heating events, road conditions, episodic industrial emissions-that are not represented in the input features. The MAE of 12.85 μg/m^3 should be interpreted in the context of the concentration range. During summer, typical PM2.5 values are 5-20 μg/m^3; during winter peaks they reach 100-200 μg/m^3. An MAE of 12.85 μg/m^3 is approximately 10% of the winter peak concentration, which is acceptable for a public advisory system

where the goal is to distinguish “clean,” “moderate,” and “hazardous” days rather than to predict exact values. The RMSE (15.73) is higher than the MAE (12.85), which means the error distribution has a heavy right tail-there are occasional days where the model misses by substantially more than the average. These are likely the most extreme pollution episodes, where local conditions deviate sharply from what the meteorological inputs predict.

10.10.3 Prediction Analysis: MOEPP Test Set

Figure 28 shows the predicted versus actual PM2.5 for the full test period.

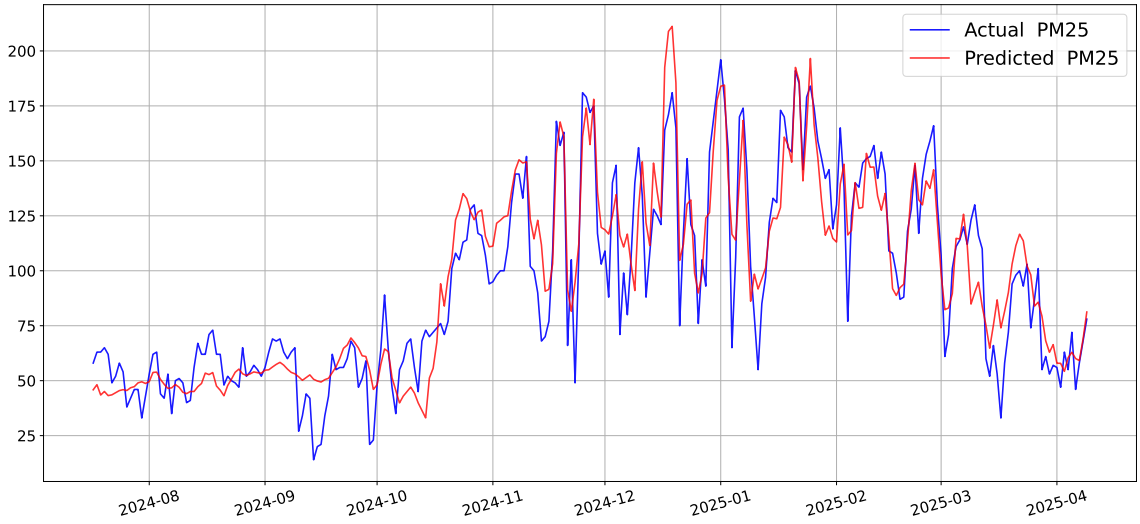


Figure 28: CNN-BiGRU model predictions (red dashed) versus actual PM2.5 (blue) on the MOEPP test set, covering August 2024 to April 2025. The model correctly captures the summer-to-winter seasonal transition, tracks the winter peak timing, and follows the spring decline. Predicted values are slightly smoother than actuals, which is characteristic of MSE-trained regressors.

Three aspects of the plot are worth noting. First, the model captures the large-scale seasonal transition accurately: the predicted series moves from low summer values through the autumn build-up and into the winter maximum at the correct time, without a systematic phase lag. This is the primary test of whether the temporal features and BiGRU memory are working as intended. Second, the model tracks day-to-day fluctuations reasonably well within the winter period, reflecting the learned influence of wind speed and pressure variations. Third, the predicted series is smoother than the actual series, particularly near the peaks. This is a known property of MSE-trained models: MSE is minimized by predicting the conditional mean of the target given the inputs, which smooths out the tails of the distribution. A model that never produces extreme predictions will have a lower expected squared error than one that sometimes predicts the extremes correctly and sometimes misses them

badly.

10.10.4 Comparison: Own Sensor Network

The same architecture trained on the low-cost sensor network dataset produces qualitatively different results. Figure 29 shows the centralized model's predictions over a 14-day horizon; Figure 30 shows the federated variant.

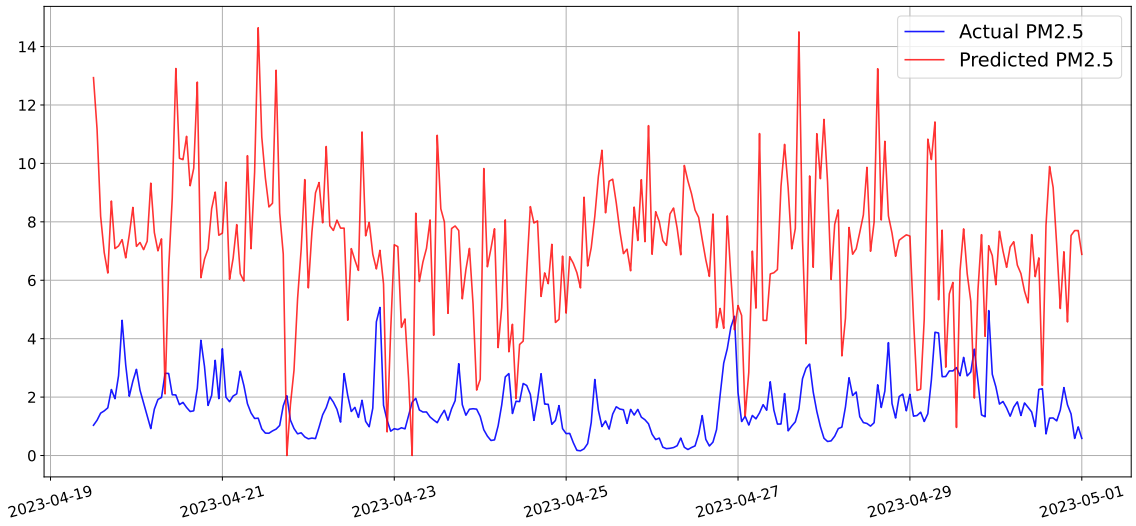


Figure 29: CNN-BiGRU predictions on the centralized own-sensor dataset (14-day horizon). Predicted values are systematically elevated relative to actuals. The model has learned to predict high because that is what the training data predominantly shows, due to humidity-induced overestimation in the optical particle counters.

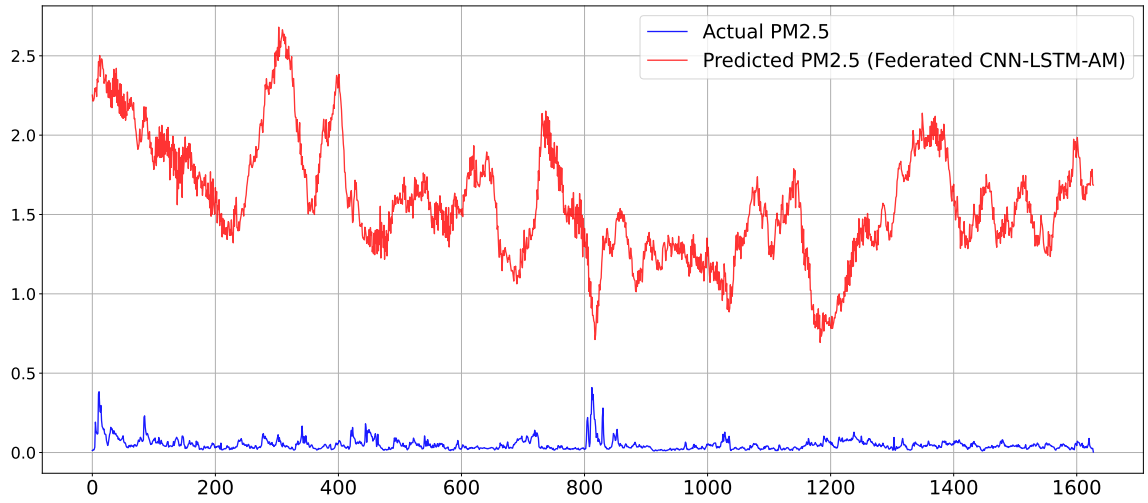


Figure 30: Federated learning variant predictions on the own-sensor dataset (7-day horizon). The systematic upward bias is similar to the centralized model, confirming that FedAvg aggregation does not correct for measurement errors that are common across all clients.

10.11 Discussion of Forecasting Results

The MOEPP model performs substantially better than the sensor-network models. The reason is data quality, not architecture.

The MOEPP data comes from a certified reference instrument with consistent calibration. The low-cost sensor data has known humidity-induced bias: optical particle counters overestimate PM2.5 when relative humidity is high, because hygroscopic growth increases the apparent scattering cross-section of aerosol particles without any corresponding increase in dry mass [17], [18]. In Tetovo, relative humidity routinely exceeds 70% during autumn and winter-the same period when PM2.5 is highest. The result is that the sensor readings are most biased precisely when they are most important. A model trained on that data learns to predict high; evaluated against reference data, it looks like it systematically overestimates.

This is a known failure mode. Hua et al. [37] showed that stratifying calibration by relative humidity category substantially reduces this bias in OPC-based networks. Mathieu-Campbell et al. **Mathieu2024Calibration** demonstrated the same effect for PurpleAir sensors under high-humidity conditions typical of the northeastern United States, finding that a humidity-stratified correction reduced mean absolute error by 30–50% compared to a single global correction. Casari and Po **Casari2024MitH** propose a more general framework-MitH-that estimates aerosol liquid water content from co-located meteorological measurements and uses it to correct OPC readings before any downstream modelling step. More recently, Guo et al. [38] demonstrated that aerosol liquid water estimation using EPA CSN network data can correct OPC readings across diverse humidity environments, which is relevant for

deployment in a region like the Polog Valley where humidity regimes vary sharply by season. The correction required is not exotic: it typically takes the form

$$\mathrm{PM}_{2.5}^{\text{corrected}} = \frac{\mathrm{PM}_{2.5}^{\text{raw}}}{1 + k\,(\mathrm{RH} - \mathrm{RH}_0)}$$

where k is a hygroscopic growth factor and RH_0 is a reference humidity. Applying such a correction before model training is the single most important pre-processing step for this deployment, and the literature cited above provides validated parameter estimates for comparable sensor types and climates.

The federated learning variant shows the same bias as the centralized model. That is expected: FedAvg aggregates weight updates from all clients but cannot correct for measurement errors that are common across all of them. When the humidity correction is applied, the federated architecture has real advantages-no raw sensor readings leave the client, which reduces bandwidth and simplifies data governance-and can continue learning as new sensor nodes are added without reprocessing historical data. The architecture is not the problem here. The training data is.

11 Discussion

11.1 The QoS Level Is the Architectural Decision

The benchmark makes the consequences of QoS choice concrete. MQTT defines three delivery semantics, and the choice between them determines throughput more than any hardware upgrade could. QoS 0 (fire-and-forget) sends a message once with no acknowledgment. There is no handshake, so throughput is limited only by network bandwidth and broker processing speed-easily hundreds or thousands of messages per second on this hardware. The cost is that messages can be silently lost: if the network drops a packet, the sender never knows. QoS 1 (at-least-once) adds a single acknowledgment: the broker responds with PUBACK when the message is stored. One round-trip of 10 ms limits per-connection throughput to 100 msg/s. Duplicate delivery is possible: if PUBACK is lost and the sender retransmits, the broker stores two copies of the same message. The application must handle deduplication, which for time-series data with a (`device_id`, `timestamp`) key is straightforward. QoS 2 (exactly-once) adds the second round-trip to prevent duplicates at the protocol level, halving per-connection throughput to 50 msg/s. The exactly-once guarantee matters when downstream deduplication is expensive or impossible-for example, financial transactions or control commands that must be executed exactly once. For environmental sensor data, where a duplicate measurement can be trivially discarded by comparing the timestamp to recent storage, QoS 2 provides no practical benefit over QoS 1. For this deployment, switching to QoS 1 would immediately raise the throughput ceiling from 50 msg/s to approximately 100 msg/s, with no hardware changes and no application logic changes other than adding timestamp-based deduplication in Benthos. The practical implication is that the current system can support twice the sensor fleet at the same hardware cost by changing a

single configuration parameter. In the benchmark, the headline 5-node figures (up to 1966 msg/s) were in fact obtained with QoS 0, and the May 13 confirming run (Table 14) shows that QoS 0 alone lifts a 2-node cluster to near-perfect efficiency. QoS 0 is appropriate for a benchmark whose purpose is to expose the hardware ceiling, but QoS 1 remains the recommended production setting: with timestamp-based deduplication in Benthos it restores the at-least-once guarantee while its pipelining removes the per-connection serialization that makes QoS 2 slow.

11.2 What Dominates Latency

In the early configurations, storage was the largest single latency contributor: VictoriaMetrics initially ran on a Longhorn-replicated volume, and the synchronous double-fsync to MicroSD added on the order of 200 ms to each write. Two changes removed this. Moving VictoriaMetrics (and NATS) to local-path storage eliminated the replication fsync, and batching writes in the Python consumer amortized the HTTP overhead. On the 5-node gold-standard run, the consumer-to-VictoriaMetrics write completes in 2 ms median and 8 ms P99 (Table 13); no single hop exceeds 10 ms at P99. What now dominates the end-to-end figure is the deliberate 2-second batch-accumulation window in the consumer-a latency the design accepts in exchange for a large reduction in request count. For air quality monitoring at 60-second sensor intervals, none of this is operationally significant: a measurement is already tens of seconds old when it arrives. The relevant latency for this application is query latency-how quickly a dashboard retrieves recent data-which VictoriaMetrics serves in under 50 ms on this hardware. Write-path latency would only matter for sub-second alerting, which this deployment does not require. Where it does matter, replacing MicroSD with a USB 3.0 SSD (sub-millisecond fsync) is a roughly €25-per-node fix.

11.3 Cloud-Native Overhead Is Acceptable

A reasonable concern when running Kubernetes on resource-constrained hardware is that the orchestration layer itself will consume a significant fraction of available resources, leaving little headroom for application workloads. The benchmark results do not support this concern. Under low load (50 msg/s sustained), the total cluster resource consumption is approximately 2.5 GB RAM per node and below 50% CPU across nodes. The remaining 5.5 GB RAM per node is available for additional workloads. This is sufficient to run the air quality model retraining pipeline (a periodic batch job triggered by Kubernetes CronJob), a small inference service for real-time PM2.5 estimation, or an expanded Benthos replica set for higher throughput after the QoS change. The picture changes only at the highest load: at 2000 msg/s the master node-which carries all stateful pipeline pods plus the observability stack-reaches 91% CPU (Table 16), while worker nodes retain headroom, which motivates the control-plane offload discussed in Section 13. K3s's minimal footprint compared to full Kubernetes is a real advantage here. The K3s binary consumes less than 300 MB at idle. Standard Kubernetes-with kube-apiserver, kube-controller-manager, kube-scheduler,

and etcd as separate processes-typically uses 1-2 GB just for the control plane components. That difference matters when total available RAM is 8 GB.

11.4 Longhorn Replication and Fault Tolerance

Longhorn's synchronous replication introduces write overhead but provides genuine fault tolerance. The trade-off is worth quantifying. Adding replication adds approximately 200 ms to mean write latency compared to what local storage would provide-which is precisely why VictoriaMetrics and NATS are kept on local-path storage and only EMQX, Loki, and Tempo use Longhorn. In exchange, the system survives a complete node failure without data loss for the Longhorn-backed components: a surviving node has a current replica of every such volume and can continue serving reads and writes immediately. (VictoriaMetrics and NATS, on local-path storage, instead recover through their own write-ahead logs.) For a deployment in a public building or outdoor enclosure, unexpected node failure is a realistic scenario-power disruption, hardware failure, physical damage. Automatic recovery without data loss and without manual intervention is a meaningful operational property. The 200 ms write overhead is the direct cost of that property.

11.5 Scaling the System

A single QoS 2 connection is limited to roughly 50 msg/s, but the 5-node cluster sustains up to 1966 msg/s aggregate (Section 9). Scaling to larger fleets requires addressing both the throughput ceiling and the storage capacity. The benchmarks answer the scaling question empirically rather than speculatively. The 5-node cluster sustains 1966 msg/s at 98.3% efficiency and 995 msg/s at 99.5% efficiency with zero message drops. For the air quality monitoring deployment-which requires on the order of tens to hundreds of msg/s depending on fleet size and reporting interval-the 5-node cluster provides capacity headroom of roughly 10-20$\times$. The throughput ceiling for this hardware generation appears to be approximately 1800-2000 msg/s sustained, at which point EMQX broker processing, NATS durability writes, and connection-layer overhead begin competing for the available I/O budget. Extending beyond this range requires either QoS 1 (eliminating one round-trip per message, which would raise the per-connection ceiling from 50 to 100 msg/s and reduce broker CPU load proportionally) or EMQX clustering with topic sharding across multiple broker instances. Storage capacity at 2000 msg/s with 5 metrics per message is approximately 10 000 samples/s. At VictoriaMetrics's measured compression ratio of 0.5 bytes per sample, this is approximately 0.43 GB/day-indefinitely sustainable with a 30-day retention policy on a 64 GB microSD, and comfortably within the 3 000 000 unique time-series limit when cardinality is properly bounded. The 192 GB written over the full benchmark campaign, however, is a reminder that sustained high-rate writes wear MicroSD cards; a production deployment should budget for USB 3.0 SSDs. Adding worker nodes extends both compute capacity and storage: Longhorn distributes volumes across all nodes, so adding a node increases the maximum replicated storage capacity and reduces per-node I/O load.

11.6 GitOps Made the Experiment Reproducible

An unplanned benefit of the ArgoCD setup was that it turned infrastructure management into version control. Every benchmarked cluster state corresponds to a tagged commit: the DNS fix is a one-line diff in the network-policy manifest, the K3s version change is a one-line diff in the Ansible inventory, and the `msg_id` label removal is a one-line diff in the consumer configuration. Each performance change could therefore be attributed to a specific file change rather than to ambient cluster drift. The 80-minute window between the gold-standard run and the DNS failure on May 14 is a clear example: with version control the responsible policy file was unambiguous, whereas otherwise it would have been tempting to blame the added load. Declaring all infrastructure state in Git from the start-and using a PVC rather than an EmptyDir for Prometheus-is worth doing before the first benchmark run.

11.7 Practical Design Guidance

Several recommendations follow directly from the benchmark and apply to anyone deploying a similar stack on constrained hardware:

1. **Choose the QoS level before deployment.** QoS 2 suits financial or safety-critical data; for environmental monitoring, QoS 1 gives the same durability guarantee at higher throughput, and QoS 0 is acceptable where occasional loss is tolerable. Changing QoS after deployment means updating every publisher and the broker.

2. **Audit metric labels for cardinality before writing any data.** Any label whose value set is unbounded (such as a per-message ID) belongs in a trace span or log entry, not a metric label; retrofitting this means rewriting all historical metrics.

3. **Test network policies under sustained load.** The DNS egress bug was invisible during functional testing because DNS responses were cached; it surfaced only when pod restarts triggered fresh lookups under load.

4. **Check version consistency across all nodes.** A K3s version skew that causes intermittent ClusterIP failures looks like flaky performance until `kubectl get nodes -o wide` reveals the mismatch.

5. **Use compensated sleep in publishers.** A naive `usleep(interval)` loop drifts under load; `clock_gettime`-based compensation holds the rate within 0.5% of target, so the benchmark measures the pipeline rather than the publisher's drift.

6. **Keep VictoriaMetrics and NATS on local-path storage.** Both maintain their own write-ahead logs; layering Longhorn replication on top only doubles SD-card I/O for no reliability gain.

7. **Watch SD-card write totals.** 192 GB over two months degraded a 64 GB card; budget for USB 3.0 SSDs or card replacement for sustained workloads.

8. **ARM64 is adequate for this workload.** The Cortex-A72 had CPU headroom at 2000 msg/s on the worker nodes; the binding limits were protocol, schema, and configuration, not compute.

12 Fault Tolerance

12.1 Architecture-Level Fault Containment

The pipeline is designed so that failure at any single component does not cause data loss and does not require manual intervention to recover. This property is achieved by combining Kubernetes pod restart policies, NATS JetStream's durable consumer model, and Longhorn's synchronous replication.

Node failure. If a node fails completely (power loss, hardware fault), Kubernetes detects the failure via the node status heartbeat timeout (default: 40 seconds). Pods running on the failed node are evicted and rescheduled on a surviving node. For components on Longhorn-replicated volumes (EMQX, Loki, Tempo), a surviving node already holds a current replica and serves it immediately, without a recovery step. NATS and VictoriaMetrics use local-path storage and recover through their own durability mechanisms instead: NATS JetStream replays from its on-disk stream log on restart, and VictoriaMetrics recovers from its write-ahead log. The downtime is bounded by the Kubernetes eviction timeout plus pod startup time, typically 60-120 seconds.

Pod failure. If an individual pod crashes (application error, OOM kill), Kubernetes restarts it according to its restart policy (Always, for all pipeline components). On restart, the pod reconnects to NATS JetStream and resumes from the last acknowledged message offset. Any messages that were in-flight at crash time-pulled from JetStream but not yet acknowledged-are redelivered. Benthos's idempotent write behavior (VictoriaMetrics ignores duplicate samples with the same timestamp) ensures that redelivery does not produce duplicate data.

NATS JetStream durability. NATS JetStream persists all stream messages to disk before acknowledging the publisher. This means that even if both NATS and Benthos crash simultaneously, the messages remain in the stream log and will be processed when the services recover. The stream is configured with a retention policy of 24 hours, providing a buffer against sustained consumer downtime.

EMQX session persistence. EMQX stores persistent sessions for QoS 1 and QoS 2 clients: if a client disconnects and reconnects, EMQX delivers any messages that were published during the disconnection. For sensor nodes that experience brief connectivity outages (e.g., a LoRaWAN link that drops for a few minutes during a storm), this prevents data loss at the protocol layer. Under QoS 0 this protocol-level buffering is not available, which is one

reason QoS 1 is preferred for production over the QoS 0 used to obtain the benchmark's peak-throughput figures.

12.2 Known Limitations

The current deployment has one single point of failure: EMQX runs as a single instance. If the EMQX pod is unavailable (during a restart, during the Kubernetes eviction window, or due to a node failure that happens to be the node EMQX is scheduled on), sensor nodes cannot deliver messages. Messages published during the outage are either buffered at the sensor (if the sensor has local storage) or lost. Addressing this requires running EMQX as a cluster, with at least two instances on different nodes and MetalLB routing connections to the available instance. EMQX supports native clustering with message routing between nodes. This is straightforward to configure but was not implemented in the current deployment, which prioritizes simplicity over full HA.

13 Limitations and Future Work

13.1 Limitations of the Air Quality Model

The spatial correction model has been calibrated and validated against synthetic data generated from known correction parameters. Validation against real PM2.5 measurements from the Tetovo reference station network is underway but not yet complete. Until validation is done against real field data, the correction term amplitudes and decay parameters should be treated as illustrative rather than operationally calibrated. The model currently treats correction terms as time-invariant. In reality, the road proximity effect is stronger during peak traffic hours than at night, and the industrial influence may vary with production schedules. Incorporating time-varying correction terms-for example, scaling Δ_{road} by a traffic activity index that varies by hour-would improve accuracy but requires traffic count data that is not currently available for the study area. The XGBoost model is trained on data from a single reference station. Station holdout cross-validation tests spatial generalization, but with only two or three stations available in the study area, the generalization test has limited statistical power.

13.2 Limitations of the LoRaWAN Planner

A field validation campaign on the SEEU campus (Section 7) revealed a systematic 12.4 dB optimistic bias in the propagation model's RSSI predictions, attributable to clutter loss parameters calibrated to average European urban environments rather than to the specific building construction and vegetation density of the study area. The bias does not affect spreading factor assignments for links with comfortable margins, but it reduces confidence in coverage predictions for devices operating near the sensitivity threshold at maximum

range. Field calibration of the clutter loss parameter against a broader set of RSSI measurements across diverse link geometries would eliminate this bias. The optimization assumes a fixed set of candidate gateway positions identified from the digital twin. Positions that are technically good but operationally infeasible-restricted roof access, no mains power supply-must be filtered out before the optimizer runs. This filtering is currently done manually and introduces a subjective element into the candidate set.

13.3 Limitations of the Edge Pipeline

Aggregate master-node resource usage was captured (Table 16), but per-pod, per-scenario resource profiling-CPU per process, memory per pod, per-device I/O throughput-was not collected systematically. The claim that compute is not the binding constraint therefore rests on master-node aggregates together with the indirect evidence discussed in Section 9: client-count invariance and the arithmetic match between the QoS 2 handshake model and the measured ceiling. A related gap is that the Prometheus StatefulSet used an EmptyDir volume pinned to a single worker, so when that worker went offline all historical infrastructure metrics were lost permanently and cannot be reconstructed. The fault tolerance properties described in Section 12 have been designed and reasoned about but not yet tested under fault injection. A systematic fault tolerance test would inject failures (process kill, node power-off, network partition) and measure recovery time and data loss. Three further limitations are specific to the benchmark methodology. First, the 3-node configuration changed both the consumer implementation (Go to Python) and the node count in a single step, so the 2-to-3-node improvement cannot be cleanly attributed to either change alone. Second, several 3-node runs produced negative average latencies (e.g., -114 ms, -214 ms); these reflect NTP clock drift between nodes rather than real negative time, arising when the consumer's clock runs slightly ahead of the publisher's. Third, the 5-node latency metric is inter-arrival time at the consumer, not the end-to-end sensor-to-storage latency used for the 1/2/3-node runs, so the two latency groups are not directly comparable. Finally, the lighter-weight stack was chosen partly for resource reasons that were not formally benchmarked against the alternatives. The repository includes version-pinned values files for Apache Kafka (with ZooKeeper), Apache IoTDB, and Apache NiFi; all three were evaluated and replaced because their JVM-based footprints (roughly 2 GB of heap for Kafka plus ZooKeeper, similar for IoTDB, multi-GB for NiFi) would consume most of the available memory on a Raspberry Pi node. A formal head-to-head comparison of the NATS/Benthos/VictoriaMetrics stack against the Kafka/NiFi/IoTDB stack on identical hardware remains future work.

13.4 Future Work

The most immediate next step is QoS 1 benchmarking to confirm the predicted $2\times$ throughput improvement. The change requires modifying one line of EMQX configuration and rerunning the benchmark suite. If the prediction holds, the benchmark will provide clean experimental evidence that protocol semantics, not hardware, determine throughput. Longer-

term directions include:

- **Field deployment and validation**: Deploy physical sensor nodes in Tetovo, connect them via the optimized LoRaWAN gateway placement, and operate the full pipeline with real sensor data. Compare model PM2.5 estimates against reference station measurements.
- **Propagation model calibration**: The field campaign identified a systematic 12.4 dB optimistic bias in the clutter loss model. A broader RSSI measurement campaign across diverse link geometries-varying distance, building density, and elevation difference-would allow the bias to be corrected with a calibrated loss factor rather than a generic published value. This would improve confidence in coverage predictions for marginal links at maximum range.
- **Fault tolerance testing**: Implement systematic fault injection (using tools such as chaos-mesh for Kubernetes-level failures) and measure recovery time, message loss, and system behavior under each failure mode.
- **Forecasting integration**: Deploy the CNN-BiGRU forecasting model as a Kubernetes CronJob that periodically retrains on the stored VictoriaMetrics data and generates multi-day PM2.5 forecasts. The edge hardware has sufficient spare capacity for periodic batch inference. Prior to deployment, humidity correction must be applied to sensor readings to remove the systematic bias identified in Section 10.11.
- **Sensor humidity correction**: Implement and validate an empirical hygroscopic correction function for the low-cost optical particle counters, fit against co-located reference instrument data. This is the single highest-priority step to improve forecasting model accuracy on the sensor network data.
- **Storage hardware upgrade**: Replace microSD cards with USB 3.0 SSDs and re-run the latency benchmark to quantify the improvement. The expected reduction in P99 latency is 50-70% based on the fsync latency difference between the two storage technologies, and the swap also removes the write-endurance concern raised by the 192 GB written during the benchmark campaign.
- **Multi-node scaling**: Expand to three or four nodes to measure horizontal scaling of the Benthos and VictoriaMetrics tiers. The expected throughput scaling is near-linear for Benthos (stateless) and sub-linear for VictoriaMetrics (shared write path).
- **Forecasting model extension**: Extend the CNN-BiGRU architecture with a Transformer-based attention mechanism and evaluate whether the attention weights provide interpretable explanations of which historical days are most predictive of future PM2.5 spikes.
- **EMQX clustering**: Deploy a multi-node EMQX cluster with topic sharding to remove the broker as a single point of failure (Section 12) and measure the throughput ceiling under clustered operation.

- **Control-plane offload**: Move stateful pipeline components (EMQX, VictoriaMetrics) off the master node using node-affinity rules. The master carried 91% CPU while workers averaged 50-65%; better distribution should raise the ceiling on the 100-client scenarios.
- **Unified end-to-end latency measurement**: Refactor the 5-node consumer to export sensor-to-write latency rather than inter-arrival time, so all four configurations can be compared on a single, directly comparable metric.

14 Conclusion

This thesis developed four components of an integrated air quality monitoring framework: a hybrid PM2.5 spatial estimation model, a GPU-accelerated LoRaWAN network planner, a cloud-native IoT pipeline on Raspberry Pi hardware, and a CNN-BiGRU forecasting model.

The spatial estimation model showed that a purely statistical approach breaks down when training data is spatially sparse. The XGBoost baseline predicts accurately in time but is blind to location-every point under the same meteorological conditions gets the same estimate. The spatial correction layer adds what the model cannot learn from data: road proximity gradients, industrial plume shapes, vegetation attenuation, elevation-driven inversion effects. The result is a pollution map with physical structure.

The LoRaWAN planner showed that gateway placement done on a simplified map produces wrong answers in a real city. Building geometry matters. The digital twin approach, derived from public DSM/DTM rasters, captures that geometry at city scale without a dedicated survey. NSGA-II finds the full Pareto front of coverage-energy trade-offs rather than a single solution, which is the useful output for an operator who has a budget constraint and needs to understand what it buys.

The edge pipeline benchmark produced two findings worth noting. First, the throughput ceiling on small clusters was not CPU or memory-it was the MQTT QoS 2 four-way handshake, which serializes per-connection delivery to roughly 50 msg/s regardless of what hardware is underneath. Switching to QoS 0, plus fixing four configuration bugs, took a 2-node cluster to near-perfect efficiency. The 5-node result (1966 msg/s, 98.3% efficiency, zero drops) was real, but the hardware was not the reason for it. Second, declaring all cluster state in Git and reconciling it with ArgoCD turned out to be as useful for debugging as for deployment-every performance change mapped to a specific commit, which made the DNS failure on May 14 immediately diagnosable rather than mysterious.

The forecasting model performed well on reference-quality data ($R^2 = 0.87$) and failed predictably on low-cost sensor data. The failure mode is specific: optical particle counters overestimate PM2.5 under high humidity, which in Tetovo means the worst readings are the most biased. A model trained on that data learns to predict high. The architecture is not the problem; the training data is. Humidity correction before training is the fix.

The thread running through all of it: the binding constraint is almost never what you would guess from the hardware spec. A protocol choice, a label cardinality bug, a sensor

calibration issue, a missing spatial correction-each of these had more impact on system performance than any hardware upgrade would have. That is the practical takeaway for anyone building something similar on constrained infrastructure.

References

[1] World Health Organization, *WHO global air quality guidelines: Particulate matter (PM2.5 and PM10), ozone, nitrogen dioxide, sulfur dioxide and carbon monoxide*, https://www.who.int/publications/i/item/9789240034228, 2021.

[2] N. Izairi et al., "Monitoring and analysis of, pm2.5 particles as air pollutants in the city of tetovo, republic of north macedonia," *Revista de Gestão Social e Ambiental*, 2024. DOI: 10.24857/rgsa.v18n12-074.

[3] R. Avis, "Causes and consequences of air pollution in north macedonia," 2022. DOI: 10.19088/k4d.2022.139.

[4] C. Belis et al., "Urban pollution in the danube and western balkans regions: The impact of major pm2.5 sources," *Environment International*, vol. 133, 2019. DOI: 10.1016/j.envint.2019.105158.

[5] M. Dimovska, D. Gjorgjev, O. Hänninen, and I. Rumrich, "Assessing health impact of air pollution in macedonian cities," *Biomedical Journal of Scientific Technical Research*, 2018. DOI: 10.26717/bjstr.2018.10.001887.

[6] M. Dimovska and R. Mladenovska, "Losing years of human life in heavy polluted cities in macedonia," *Open Access Macedonian Journal of Medical Sciences*, vol. 7, pp. 428 – 434, 2019. DOI: 10.3889/oamjms.2019.149.

[7] C. Belis, M. Ballocci, V. Matkovic, G. Millo, M. Jevtić, and R. V. Dingenen, "Costs of air pollution impact on health in the western balkans: Preliminary results," *The European Journal of Public Health*, vol. 32, 2022. DOI: 10.1093/eurpub/ckac129.144.

[8] M. Jevtic and V. Matkovic, "Health experts' voice for healthier choice – a call for zero air pollution in the western balkans," *The European Journal of Public Health*, vol. 32, 2022. DOI: 10.1093/eurpub/ckac129.145.

[9] B. Angelevska, V. Atanasova, and I. Andreevski, "Urban air quality guidance based on measures categorization in road transport," *Civil Engineering Journal*, vol. 7, pp. 253–267, 2021. DOI: 10.28991/cej-2021-03091651.

[10] E. Brattich et al., "How to get the best from low-cost particulate matter sensors: Guidelines and practical recommendations," *Sensors (Basel, Switzerland)*, vol. 20, 2020. DOI: 10.3390/s20113073.

[11] M. Vogt, P. Schneider, N. Castell, and P. Hamer, "Assessment of low-cost particulate matter sensor systems against optical and gravimetric methods in a field co-location in norway," *Atmosphere*, 2021. DOI: 10.3390/atmos12080961.

[12] Y. Zheng, F. Liu, and H.-P. Hsieh, "U-air: When urban air quality inference meets big data," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '13, Chicago, Illinois, USA: Association for Computing Machinery, 2013, 1436–1444, ISBN: 9781450321747. DOI: 10.1145/2487575.2488188. [Online]. Available: https://doi.org/10.1145/2487575.2488188.

[13] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16, San Francisco, California, USA: Association for Computing Machinery, 2016, 785–794, ISBN: 9781450342322. DOI: 10.1145/2939672.2939785. [Online]. Available: https://doi.org/10.1145/2939672.2939785.

[14] G. Hoek et al., "A review of land-use regression models to assess spatial variation of outdoor air pollution," *Atmospheric Environment*, vol. 42, no. 33, pp. 7561–7578, 2008, ISSN: 1352-2310. DOI: https://doi.org/10.1016/j.atmosenv.2008.05.057. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1352231008005748.

[15] L. Bai et al., "Long-term field evaluation of low-cost particulate matter sensors in nanjing," *Aerosol and Air Quality Research*, vol. 20, pp. 242 –253, 2020. DOI: 10.4209/aaqr.2018.11.0424.

[16] C. Malings et al., "Fine particle mass monitoring with low-cost sensors: Corrections and long-term performance evaluation," *Aerosol Science and Technology*, vol. 54, pp. 160 –174, 2018. DOI: 10.1080/02786826.2019.1623863.

[17] L. Crilley et al., "Effect of aerosol composition on the performance of low-cost optical particle counter correction factors," *Atmospheric Measurement Techniques*, 2019. DOI: 10.5194/amt-2019-370-supplement.

[18] A. D. Antonio, O. Popoola, B. Ouyang, J. Saffell, and R. L. Jones, "Developing a relative humidity correction for low-cost sensors measuring ambient particulate matter," *Sensors (Basel, Switzerland)*, vol. 18, 2018. DOI: 10.3390/s18092790.

[19] M. Si, Y. Xiong, S. Du, and K. Du, "Evaluation and calibration of a low-cost particle sensor in ambient conditions using machine-learning methods," *Atmospheric Measurement Techniques*, vol. 13, pp. 1693–1707, 2020. DOI: 10.5194/amt-13-1693-2020.

[20] V. Evagelopoulos, P. Begou, and S. Zoras, "In-depth study of pm2.5 and pm10 concentrations over a 12-year period and their elemental composition in the lignite center of western macedonia, greece," *Atmosphere*, 2022. DOI: 10.3390/atmos13111900.

[21] U. Raza, P. Kulkarni, and M. Sooriyabandara, "Low power wide area networks: An overview," *IEEE Communications Surveys Tutorials*, vol. 19, no. 2, pp. 855–873, 2017. DOI: 10.1109/COMST.2017.2652320.

[22] A. Augustin, J. Yi, T. Clausen, and W. M. Townsley, *A study of lora: Long range low power networks for the internet of things*, 2016. DOI: 10.3390/s16091466. [Online]. Available: https://www.mdpi.com/1424-8220/16/9/1466.

[23] C. Modesto, L. Mozart, G. Gonçalves, C. Nahum, B. Castro, and A. Klautau, *Lorawan gateway placement for network planning using ray tracing-based channel models*, 2026. arXiv: 2603.29105 [cs.NI]. [Online]. Available: https://arxiv.org/abs/2603.29105.

[24] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," 2, vol. 6, 2002, pp. 182–197. DOI: 10.1109/4235.996017.

[25] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12, Helsinki, Finland: Association for Computing Machinery, 2012, 13–16, ISBN: 9781450315197. DOI: 10.1145/2342509.2342513. [Online]. Available: https://doi.org/10.1145/2342509.2342513.

[26] M. Handosa, D. Gračanin, and H. G. Elmongui, "Performance evaluation of mqtt-based internet of things systems," in *2017 Winter Simulation Conference (WSC)*, 2017, pp. 4544–4545. DOI: 10.1109/WSC.2017.8248196.

[27] VictoriaMetrics Team, *VictoriaMetrics performance benchmarks*, https://victoriametrics.com/blog/, 2023.

[28] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *Internet of Things Journal, IEEE*, vol. 1, Jan. 2012. DOI: 10.1109/JIOT.2014.2306328.

[29] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. DOI: 10.1162/neco.1997.9.8.1735.

[30] K. Cho et al., *Learning phrase representations using rnn encoder-decoder for statistical machine translation*, 2014. arXiv: 1406.1078 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1406.1078.

[31] S. Zhou, W. Li, and J. Qiao, "Prediction of pm2.5 concentration based on recurrent fuzzy neural network," in *2017 36th Chinese Control Conference (CCC)*, 2017, pp. 3920–3924. DOI: 10.23919/ChiCC.2017.8027970.

[32] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, *Communication-efficient learning of deep networks from decentralized data*, 2023. arXiv: 1602.05629 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1602.05629.

[33] AQICN, *Air quality index: Tetovo, north macedonia*, https://aqicn.org/city/tetovo, 2025.

[34] J. Lamprecht, *Meteostat: Open weather and climate data*, https://meteostat.net, 2025.

[35] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, 2016. arXiv: 1409.0473 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1409.0473.

[36] I. Loshchilov and F. Hutter, *Decoupled weight decay regularization*, 2019. arXiv: 1711.05101 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1711.05101.

[37] J. Hua et al., “Improved pm2.5 concentration estimates from low-cost sensors using calibration models categorized by relative humidity,” *Aerosol Science and Technology*, vol. 55, pp. 600 –613, 2021. DOI: 10.1080/02786826.2021.1873911.

[38] Y. Guo et al., “Improving low-cost optical pm sensor accuracy in humid conditions via aerosol liquid water estimation using u.s. epa csn data,” *ACS Est Air*, vol. 3, pp. 326 –335, 2026. DOI: 10.1021/acsestair.5c00225.