الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria
وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and Scientific Research
المدرسة الوطنية العليا للتكنولوجيات المتقدمة
National Higher School of Advanced Technologies

Department of Electrical Engineering and Industrial Computing

Final Year Project for the Engineering Degree

- Field -
Telecommunications

- Specialty -
Telecommunications and Networking Systems

- Subject -

A Data-Driven Framework for AI-Powered LTE Network Performance Optimization

Realized by:

ALAOUCHICHE Abderrahmane Yaakoub
KESSOUM Mohamed Walid

Members of the Jury:

Name	Establishment	Grade	Quality
Mr. Islem BOUCHACHI	ENSTA	MCA	President
Mrs. Souhila BOUTARFA	ENSTA	MCB	Examiner
Mrs. Imane CHIALI	ENSTA	MCB	Examiner
Mrs. Kheira LAKHDARI	ENSTA	MAB	Supervisor
Mr. Abdelkader BELAHCENE	ENSTA	MCA	Co-Supervisor
Mr. Sifeddine ALREME	OTA (djezzy)	TBM	External Co-Supervisor

Algiers, 25/06/2025

2024–2025

Dedication

ALAOUCHICHE Abderrahmane Yaakoub

To my parents, grandparents, siblings and close ones for their constant support.
To my mother especially — your strength and sacrifices have been my greatest motivation. This achievement is as much yours as it is mine.

KESSOUM Mohamed Walid

To my parents, siblings, family, and close ones — for their invaluable support and constant encouragement throughout my academic journey. This accomplishment is a testament to their unwavering belief in me.

Acknowledgements

We would like to sincerely thank our academic supervisors, Dr. LAKHDARI Khiera and Dr. BELAHCENE Abdelkader, Assistant Professors in the GEII department at ENSTA, for their support and guidance during this project.

We also express our appreciation to Mr. ALREME Sifeddine, Traffic and Budget Manager at Djezzy, for his role as our industrial supervisor

We extend our gratitude to the members of the jury for their time, valuable comments, and thoughtful evaluation of our thesis.

Finally, we acknowledge the teaching staff at the National Higher School of Advanced Technologies, as well as the professional instructors involved in our training, whose expertise contributed to our academic development.

الملخّص

تشكل تحسين شبكات LTE الحديثة في بيئات متعددة الموردين مثل شبكة Djezzy تحديات كبيرة. يركز هذا المشروع على دمج قدرات التنبؤ والتوقع لمؤشرات الأداء الرئيسية KPI باستخدام تقنيات الذكاء الاصطناعي.

تم تطوير مستودع بيانات قابل للتوسع مع نموذج أبعادي يدعم إدارة فعالة لمؤشرات الأداء من خلال عمليات ETL متينة.

تم تطبيق نماذج ذكاء اصطناعي متقدمة، بما في ذلك LSTM و Prophet ، للتنبؤ بالسلاسل الزمنية، مما يمكّن من التنبؤ الاستباقي بمقاييس أداء الشبكة. تُعرض هذه التوقعات عبر صفحة ويب مخصصة لتعزيز التخطيط واتخاذ القرار.

أظهرت التقييمات باستخدام بيانات حقيقية تحسناً في التنبؤ بعمليات الشبكة، مما يمثل خطوة نحو إدارة شبكات LTE معتمدة على البيانات في بيئات متعددة الموردين معقدة.

الكلمات المفتاحية: LTE ، تحسين الشبكة، DWH ، ETL ، تعلم آلي، تنبؤ، Prophet ، LSTM ، شبكة متعددة البائعين.

Abstract

Optimizing modern LTE networks in multi-vendor environments such as Djezzy's presents significant challenges. This project focuses on integrating AI-driven KPI forecasting and prediction capabilities.

A scalable data warehouse with a dimensional model supports efficient KPI data management through robust ETL processes.

Advanced AI models, including LSTM and Prophet, were implemented for time series forecasting to enable proactive prediction of network performance metrics. These predictions are delivered via a dedicated web page to enhance network planning and decision-making.

Evaluation on real-world data demonstrates improved foresight for network operations, marking a step forward toward data-driven LTE management in complex multi-vendor contexts.

Keywords: LTE, Network Optimization, Data Warehouse, ETL, AI, ML, Prophet, LSTM, Forecasting, Multi-Vendor Networks.

Résumé

L'optimisation des réseaux LTE modernes dans des environnements multi-fournisseurs tels que celui de Djezzy présente des défis importants. Ce projet se concentre sur l'intégration de capacités de prévision et de prédiction des indicateurs clés de performance (KPI) basées sur l'intelligence artificielle.

Un entrepôt de données évolutif avec un modèle dimensionnel permet une gestion efficace des KPI grâce à des processus ETL robustes.

Des modèles avancés d'IA, notamment LSTM et Prophet, ont été utilisés pour la prévision de séries temporelles afin de permettre une prédiction proactive des métriques de performance réseau. Ces prédictions sont accessibles via une page web dédiée pour améliorer la planification et la prise de décision.

L'évaluation sur des données réelles montre une meilleure anticipation des opérations réseau, marquant une avancée vers une gestion LTE pilotée par les données dans des contextes multi-fournisseurs complexes.

Mots-clés : LTE, Optimisation Réseau, Entrepôt de Données, ETL, IA, ML, Prophet, LSTM, Prévision, Visualisation, Réseaux Multi-Fournisseurs.

Contents

List of Figures

List of Tables

Acronyms

2G/3G/4G/5G	Second/Third/Fourth/Fifth Generation of Mobile Networks
3GPP	3rd Generation Partnership Project
ACF	Autocorrelation Function
AI	Artificial Intelligence
APN	Access Point Name
BSC	Base Station Controller
CDMA	Code Division Multiple Access
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CQI	Channel Quality Indicator
CSV	Comma-Separated Values
DAG	Directed Acyclic Graph
DL	Downlink
DWH	Data Warehouse
EDA	Exploratory Data Analysis
EMS	Element Management System
eNodeB	Evolved NodeB
EPC	Evolved Packet Core
EPS	Evolved Packet System
E-RAB	E-UTRAN Radio Access Bearer
ETL	Extract, Transform, Load
E-UTRAN	Evolved Universal Terrestrial Radio Access Network
FDD	Frequency Division Duplex
GPRS	General Packet Radio Service
GRU	Gated Recurrent Unit
GSM	Global System for Mobile Communications
HSS	Home Subscriber Server

IoT	Internet of Things
IP	Internet Protocol
KPI	Key Performance Indicator
LTE	Long-Term Evolution
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MIMO	Multiple Input Multiple Output
ML	Machine Learning
MME	Mobility Management Entity
MT	Mobile Termination
MV	Materialized View
NN	Neural Network
OFDMA	Orthogonal Frequency Division Multiple Access
OMC	Operation and Maintenance Center
OSS	Operations Support System
PACF	Partial Autocorrelation Function
PDN	Packet Data Network
P-GW	Packet Data Network Gateway
PL/pgSQL	Procedural Language/PostgreSQL
PRB	Physical Resource Block
PS	Packet Switched
PUCCH	Physical Uplink Control Channel
PUSCH	Physical Uplink Shared Channel
QCI	Quality of Service Class Identifier
QoE	Quality of Experience
QoS	Quality of Service
RNC	Radio Network Controller
RNN	Recurrent Neural Network
RMSE	Root Mean Squared Error
RRC	Radio Resource Control
RSRP	Reference Signal Received Power
SAE	System Architecture Evolution

SC-FDMA	Single Carrier Frequency Division Multiple Access
S-GW	Serving Gateway
SINR	Signal-to-Interference-plus-Noise Ratio
SMAPE	Symmetric Mean Absolute Percentage Error
SMS	Short Message Service
SON	Self-Organizing Networks
SQL	Structured Query Language
SVR	Support Vector Regression
TDD	Time Division Duplex
TE	Terminal Equipment
UE	User Equipment
UICC	Universal Integrated Circuit Card
UMTS	Universal Mobile Telecommunications System
USIM	Universal Subscriber Identity Module
VoIP	Voice over Internet Protocol
VoLTE	Voice over Long-Term Evolution
W-CDMA	Wideband Code Division Multiple Access
XML	Extensible Markup Language

List of Symbols

General Mathematical Symbols

$\sigma(\cdot)$	Sigmoid activation function, $\sigma(z) = (1 + e^{-z})^{-1}$
$\tanh(\cdot)$	Hyperbolic tangent activation function
$\odot$	Element-wise (Hadamard) product
$[\cdot, \cdot]$	Concatenation of vectors or matrices

LSTM Model Notations

$x^{(t)}$	Input vector at time step t
$y^{(t)}$	Hidden state (output) vector at time step t
$c^{(t)}$	Cell state vector at time step t
$f^{(t)}$	Forget gate activation at time step t
$i^{(t)}$	Input gate activation at time step t
$o^{(t)}$	Output gate activation at time step t
$z^{(t)}$	Candidate cell state vector at time step t
W_f, W_i, W_o, W_z	Weight matrices for forget, input, output, and candidate gates
b_f, b_i, b_o, b_z	Bias vectors for forget, input, output, and candidate gates
n_x	Dimension of the input vector
n_h	Dimension of the hidden and cell state vectors

Prophet Model Notations

$y(t)$	Observed time series value at time t
$g(t)$	Trend component of the time series
$s(t)$	Seasonality component of the time series
$h(t)$	Holiday and event component
ϵ_t	Error (noise) term
$C(t)$	Carrying capacity (logistic growth)
$k(t)$	Growth rate (logistic growth)
$m(t)$	Offset parameter (logistic growth)
P	Period of the seasonal component (e.g., 7 for weekly)
N	Number of Fourier terms used for seasonality
a_n, b_n	Fourier coefficients for seasonality

Evaluation Metrics Notations

N	Total number of data points in the evaluation set
y_i	i-th actual (true) value
$\hat{y}_i$	i-th predicted (forecasted) value
$\bar{y}$	Mean of the actual values
ε	Small constant to avoid division by zero (e.g., in SMAPE)

Introduction

The evolution of mobile data services and the escalating user expectations for seamless, high-speed connectivity place immense pressure on Long-Term Evolution (LTE) network operators. Optimizing these complex networks is no longer a matter of routine maintenance but a critical imperative for maintaining competitive advantage, ensuring user satisfaction, and managing operational expenditure. Industry analyses indicate that even marginal improvements in network performance, such as a 1% increase in call success rate or a 5% enhancement in average user throughput, can translate into significant revenue protection and reduced customer churn for operators [5] [6]. Conversely, suboptimal performance, leading to issues like dropped calls, slow data speeds, or service unavailability, directly impacts Quality of Experience (QoE) and can lead to subscriber defection in a highly competitive market [7].

The operational context for this research is Djezzy, a prominent telecommunications operator in Algeria. Like many established global operators, Djezzy's extensive LTE infrastructure incorporates equipment from multiple leading vendors (e.g., Huawei, Nokia, ZTE). While this multi-vendor strategy offers flexibility and avoids vendor lock-in, it concurrently introduces significant operational complexities. Each vendor's Element Management System (EMS) often generates performance data in proprietary formats, with potentially varying counter definitions and granularities. This heterogeneity poses a substantial challenge for creating a unified, network-wide view of performance, hindering timely and effective optimization efforts. Manually consolidating, normalizing, and analyzing this disparate data is a time-consuming, error-prone process that limits the ability to proactively manage network resources and respond to emerging issues.

Traditional network management approaches, often reactive and reliant on threshold-based alarming, are increasingly inadequate in the face of such dynamic and data-rich environments. The sheer volume, velocity, and variety of data generated by modern LTE networks necessitate a paradigm shift towards intelligent, data-driven, and automated solutions. Artificial Intelligence (AI) offers powerful tools to extract actionable insights from this data, enabling predictive maintenance, proactive resource allocation, and enhanced anomaly detection. However, the successful application of AI in this domain is contingent upon a robust data management foundation and well-designed analytical models tailored to the specific characteristics of telecommunication KPIs.

This End-of-Study Project presents the design, implementation, and evaluation of a comprehensive full-stack analytics solution tailored to LTE network performance. The primary objective is to apply contemporary data engineering methods and predictive modeling techniques to transform raw performance data into actionable insights. These insights support network engineers in anticipating potential issues, optimizing resource allocation, and implementing proactive operational strategies. **While many academic studies focus on KPI prediction, they often stop short of providing an integrated, multi-vendor data management backbone and an accessible service layer for operational use—a critical gap this thesis aims to bridge.**

The platform is structured around three core components:

- **Data Management Subsystem**: A foundational layer that includes a scalable data

warehouse (DWH) and a robust ETL pipeline, responsible for integrating, cleaning, normalizing, and structuring raw LTE KPI data to support analytical tasks.

- **Predictive Analytics Module**: A suite of forecasting models (e.g., LSTM, Prophet) designed to estimate future network performance and enable proactive decision-making by providing engineers with the foresight needed for targeted interventions.
- **Web-Based Interface**: A Flask-powered application providing engineers with an intuitive, vendor-agnostic interface to monitor KPIs and access both historical and forecasted data.

The document is organized to reflect the development lifecycle:

- **Chapter 1: Theoretical Background and State of the Art** introduces LTE architecture, KPI significance, and related data-driven approaches in network performance management.
- **Chapter 2: Methodology** details the problem formulation, methodological choices, and the tools and frameworks employed.
- **Chapter 3: Design and Implementation of the Data Management Subsystem** discusses the DWH and ETL architecture, including data modeling and quality assurance mechanisms.
- **Chapter 4: AI-Based KPI Forecasting** covers data preparation, model selection, training, and backend integration.
- **Chapter 5: Experimental Setup, Model Evaluation, and Results** presents the evaluation protocol, performance metrics, and benchmarking outcomes.
- **Chapter 6: Discussion of Results and Optimization Implications** interprets the findings, highlights operational benefits, addresses limitations, and proposes future improvements.

Overall, this work aims to provide a practical framework and a foundational system that not only demonstrates the feasibility of AI-driven KPI forecasting in a multi-vendor LTE environment but also offers tangible insights into the model development lifecycle, data management prerequisites, and the potential for enhancing proactive network operations.

Chapter 1

Theoretical Background and State of the Art

This chapter introduces LTE networks, covering their evolution, architecture, and performance indicators (KPIs). It also reviews current research and industry practices, focusing on the use of Artificial Intelligence for LTE optimization and highlighting existing limitations and trends.

1.1 Overview of Mobile Networks

1.1.1 Brief Introduction to Mobile Network Generations (2G, 3G, 4G)

The evolution of mobile networks has been driven by the need for faster communication, increased capacity, and enhanced security. Each generation has introduced significant technological advancements, addressing the limitations of its predecessor.

- **1G (First Generation)**: Introduced in Japan in 1979 by Nippon Telegraph and Telephone (NTT), 1G relied on analog transmission using FDMA (Frequency Division Multiple Access). While it enabled basic voice communication, it suffered from low capacity, poor security, and inefficient spectrum utilization.[8]
- **2G (Second Generation)**: Launched in Finland in 1991, 2G introduced digital modulation with GSM (Global System for Mobile Communications). It utilized TDMA and CDMA technologies, improving call quality and enabling SMS (Short Message Service). Later advancements, such as GPRS (General Packet Radio Service), provided basic internet access, marking the early shift toward data-driven networks.[8]
- **3G (Third Generation)**: The demand for high-speed mobile internet and multimedia capabilities led to the launch of 3G networks in Japan in 1998, based on W-CDMA (Wideband Code Division Multiple Access). Supporting speeds up to 2 Mbps, 3G enabled video calls, mobile browsing, and multimedia streaming. Enhancements like HSPA and HSPA+ introduced MIMO (Multiple Input Multiple Output) technology, significantly improving data throughput.[8]
- **4G (Fourth Generation)**: First deployed in Sweden in 2009, 4G LTE marked a transition to a fully IP-based network. Utilizing OFDMA (Orthogonal Frequency Division Multiple Access) and advanced MIMO technology, 4G LTE achieved speeds of up to 100 Mbps for mobile users and 1 Gbps for stationary connections. Operating across the 700

MHz to 2600 MHz frequency bands, it enabled seamless HD streaming, real-time gaming, and IoT (Internet of Things) applications.[8]

- **Evolutionary Impact**: Each generation has played a crucial role in transforming wireless communication—from basic analog voice calls in 1G to the high-speed, low-latency digital connectivity of 4G LTE. These advancements have paved the way for the next leap in wireless technology—5G.

1.1.2 Schema of Mobile Network Generations

The evolution of mobile network generations has significantly transformed communication, moving from analog voice calls to high-speed data transmission with low latency. The following diagram illustrates the key features of each generation:

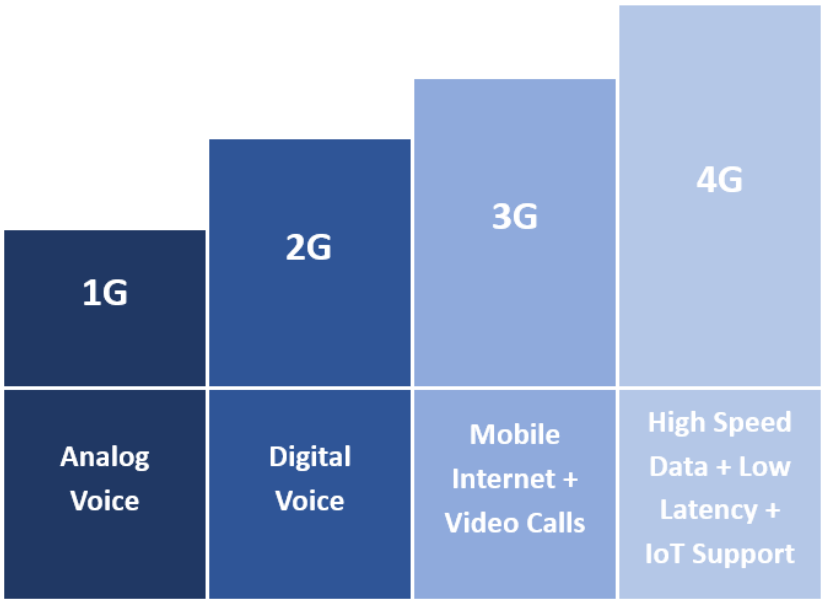


Figure 1.1: Schema of Mobile Network Generations

LTE deployment initially required multi-technology support for mobility and service continuity, as operators transitioned from legacy 2G and 3G networks [9]. However, with the introduction of **Voice over LTE (VoLTE)**, LTE is now capable of handling both data and voice services, reducing the reliance on older networks.

As a result, our primary focus will be on **4G LTE**, as previous generations, while essential for its development, are now considered legacy technologies. With the industry shifting towards **5G and 6G**, these older networks need to be phased out to make room for more efficient and advanced solutions.

1.2 Introduction to LTE

The rise in mobile broadband demand led to the development of **Long Term Evolution (LTE)** and **System Architecture Evolution (SAE)** under 3GPP Release 8. These advancements redefined the radio access and core networks, creating the **Evolved Packet System (EPS)**.

LTE improves upon 3G by reducing latency and increasing efficiency through technologies like **OFDMA** and **MIMO**, achieving peak speeds of **300 Mbps (downlink)** and **75 Mbps (uplink)**. Later releases, such as Release 9, introduced femtocells and beamforming, setting the stage for LTE-Advanced and ensuring compatibility with future technologies like 5G [10].

1.2.1 LTE as the Foundation for 5G and Beyond

LTE is not just a standalone technology but a critical infrastructure for the future of mobile communications. According to the **GSMA**, LTE users are expected to reach two-thirds of the global mobile user base by 2025, with a net addition of 3.6 billion users between 2016 and 2025 [1]. This growth underscores LTE's role as the primary infrastructure in the 5G era, where technologies like **carrier aggregation** and **Massive MIMO** continue to enhance user experience and network performance.

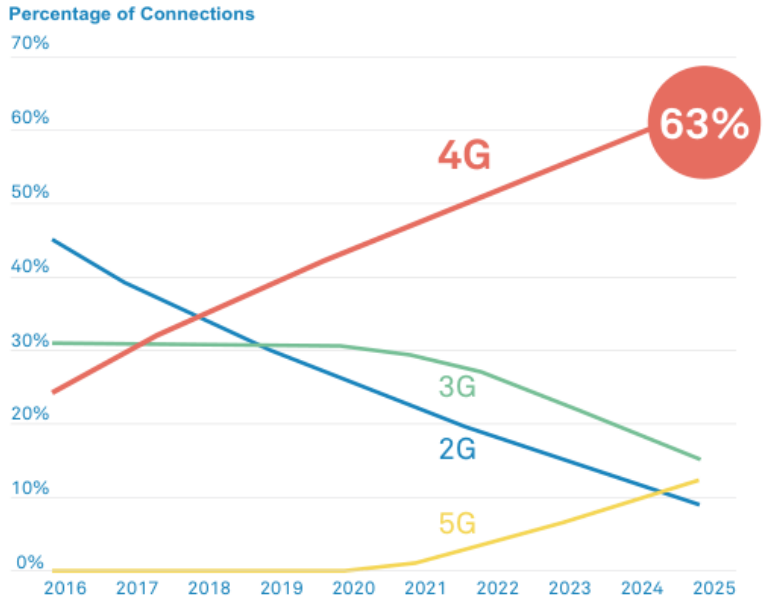


Figure 1.2: Mobile Network Connections Distribution by 2025: 4G Leading at 63% [1].

LTE's scalability and flexibility make it an ideal candidate for supporting diverse applications, from high-speed video streaming to IoT connectivity. Its ability to adapt to varying bandwidths (from 1.4 MHz to 20 MHz) and support high mobility (up to 350 km/h) ensures reliable performance across different use cases [11].

1.3 LTE Architecture and Components

With the growing demand for high-speed and reliable mobile networks, understanding LTE architecture is crucial for optimizing performance. A well-structured network directly impacts key performance indicators (KPIs), ensuring efficient resource utilization and seamless connectivity.

1.3.1 Overview of LTE Architecture

The LTE architecture is divided into three main components:

- **The User Equipment (UE)**
- **E-UTRAN (Evolved Universal Terrestrial Radio Access Network)**

- **EPC (Evolved Packet Core)**

This separation simplifies network management and improves performance, making LTE a highly efficient and scalable system [12].

1.3.2 Interaction Between UE, E-UTRAN, and EPC

The User Equipment (UE), Evolved Universal Terrestrial Radio Access Network (E-UTRAN), and Evolved Packet Core (EPC) communicate through well-defined interfaces:

- **Uu Interface**: The radio interface connecting the UE to the eNodeB. It handles signaling between the eNodeB and the MME, as well as data traffic between the UE and the S-GW.
- **S1 Interface**: Connects the eNodeB to the EPC, facilitating:
  - **S1-MME**: Control signaling between the eNodeB and MME.
  - **S1-U**: Data transmission between the eNodeB and S-GW.
- **X2 Interface**: Enables communication between eNodeBs, supporting handover procedures and interference management.
- **SGi Interface**: Connects the P-GW to external Packet Data Networks (PDNs), identified by Access Point Names (APNs). The P-GW also functions similarly to GGSN and SGSN in UMTS and GSM.

Figure 1.3: The LTE EPS Architecture.[2]

1.3.3 User Equipment (UE)

The LTE user equipment (UE) shares the same internal architecture as UMTS and GSM, consisting of:

- **Mobile Termination (MT)**: Manages network communication.
- **Terminal Equipment (TE)**: Handles user data processing.
- **Universal Integrated Circuit Card (UICC)**: Contains the **USIM**, which stores user credentials like a 3G SIM.

1.3.4 E-UTRAN: The Radio Access Network

E-UTRAN is responsible for the radio communication between user equipment (UE) and the LTE network. Its primary component is the **eNodeB (evolved NodeB)**, which performs the following functions:

- **Radio Resource Management (RRM)**: Allocates and manages radio resources efficiently.
- **Scheduling**: Determines how physical resources (e.g., Physical Resource Blocks or PRBs) are allocated to users.
- **Handover Management**: Ensures seamless connectivity as users move between cells.

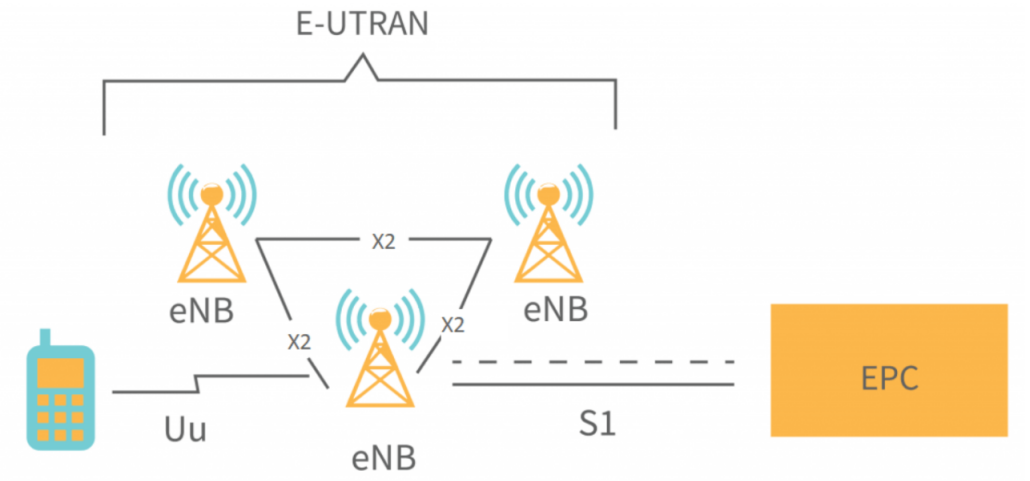


Figure 1.4: E-UTRAN Architecture.[2]

E-UTRAN employs advanced technologies such as **OFDMA** (Orthogonal Frequency Division Multiple Access) for the downlink and **SC-FDMA** (Single Carrier Frequency Division Multiple Access) for the uplink, enhancing spectral efficiency and minimizing interference [13]. To manage uplink and downlink communication, E-UTRAN supports two duplexing methods: **FDD** (Frequency Division Duplex), which uses separate frequency bands for uplink and downlink—ideal for symmetric traffic like voice calls due to its low latency and interference-free operation—and **TDD** (Time Division Duplex), which shares the same frequency band by alternating uplink and downlink in time slots, making it efficient for asymmetric traffic such as video streaming by dynamically allocating resources. The selection between FDD and TDD depends on spectrum availability and traffic patterns, ensuring optimal resource utilization and network performance.

1.3.5 EPC: The Core Network

The **Evolved Packet Core (EPC)** is the backbone of the LTE network, responsible for data routing, mobility management, and connectivity to external networks. Its key components include:

- **MME (Mobility Management Entity)**: Handles signaling, authentication, and mobility management.
- **S-GW (Serving Gateway)**: Routes data packets between eNodeBs and the P-GW.

- **P-GW (Packet Data Network Gateway)**: Connects the LTE network to external networks (e.g., the internet).
- **HSS (Home Subscriber Server)**: Stores subscriber information and supports authentication.

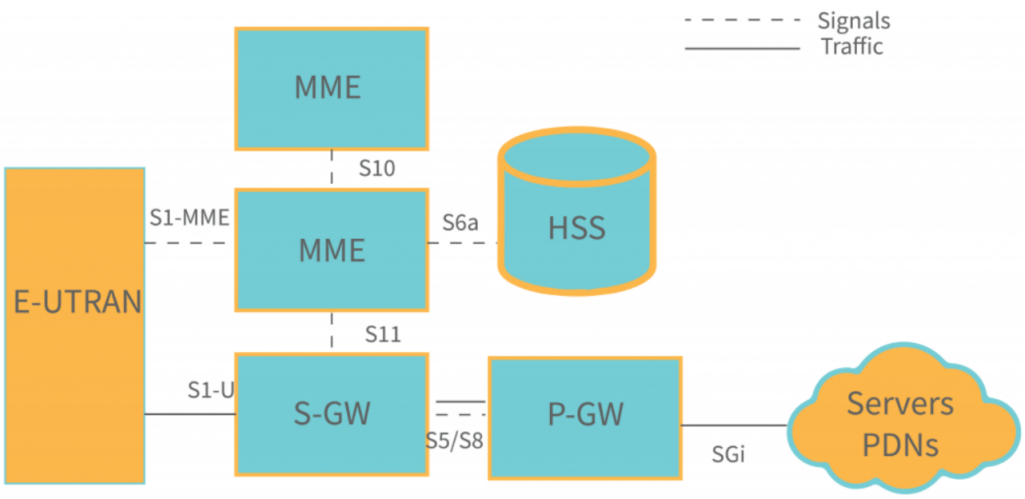


Figure 1.5: EPC Architecture.[2]

The **Evolved Packet Core (EPC)** relies on multiple interfaces to manage signaling, user data transfer, and mobility.

- **S1-MME** and **S1-U**: Connect the E-UTRAN to the core, handling signaling and data tunneling.
- **S5/S8**: Facilitate data transfer between the Serving GW and PDN GW, with S8 supporting inter-PLMN connections.
- **S10** and **S11**: Enable MME relocation and coordination with the Serving GW.
- **S6a**: Ensures authentication and subscription management between MME and HSS.
- **SGi**: Links the PDN GW to external packet networks, enabling internet access and IMS services.

Conclusion

The LTE architecture provides a robust and scalable framework for high-speed mobile communications, integrating key components such as the UE, E-UTRAN, and EPC to ensure seamless connectivity and efficient resource management. While this overview highlights the essential elements of LTE, the full architecture is even more complex, involving numerous interfaces and advanced technologies that govern mobility, signaling, and data flow.

A deep understanding of LTE is crucial for network optimization, as it directly impacts key performance indicators (KPIs). Mastering these concepts is essential for the success of our project, enabling us to analyze and enhance network performance effectively.

1.4 Key Performance Indicators (KPIs) and Optimization in LTE

Optimizing Long-Term Evolution (LTE) networks is crucial for ensuring efficient performance that meets predefined standards for service quality, coverage, and capacity. This process involves continuous monitoring and adjustment, guided by Key Performance Indicators (KPIs). Optimization occurs throughout the network lifecycle, starting with **pre-launch optimization** (focused on coverage and interference using drive tests) and transitioning to **post-launch optimization**, which leverages real traffic data and performance counters for fine-tuning [3].

LTE optimization presents unique challenges compared to earlier generations like 2G/3G, primarily due to its **frequency reuse factor of 1** (increasing co-channel interference) and its reliance on the **packet-switched (PS) domain** for all services, demanding Quality of Service (QoS)-aware strategies, especially for real-time applications like Voice over LTE (VoIP) [3].

1.4.1 Defining Key Performance Indicators (KPIs)

KPIs provide quantitative measures of network performance, derived from counters collected by the Operation and Maintenance Center (OMC). They are essential for monitoring network health, diagnosing issues, and guiding optimization efforts. The most critical KPIs are categorized as illustrated in Figure 1.6.

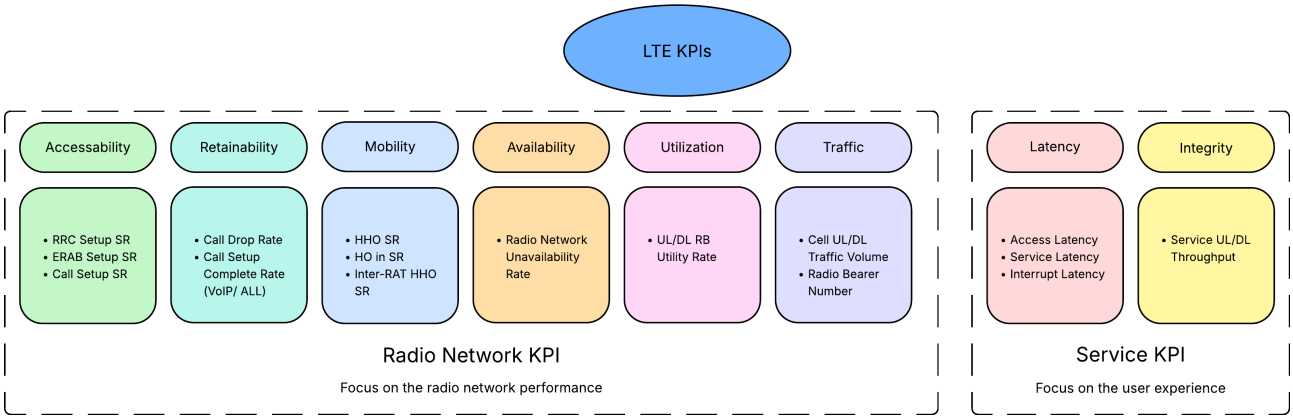


Figure 1.6: Categorization of LTE KPIs.

As illustrated in Figure 1.6, LTE KPIs can be broadly classified based on their focus: those targeting the underlying **Radio Network performance** (such as resource availability and connection stability) and those reflecting the end-**User Experience** (like perceived speed and latency). This leads to two primary groups: Radio Network KPIs and Service KPIs. The figure further breaks these down into specific categories. Key examples within these categories, derived from the figure, are summarized in Table 1.1. Note that specific formulas and target values for these KPIs depend heavily on the operator's strategy and vendor implementation.

Below are some of the main LTE KPIs along with their formulas and units:

Table 1.1: Consolidated LTE Key Performance Indicators (KPIs) [4]

Category	**KPI Name**	**Formula**	**Unit**
Accessibility	RRC Setup Success Rate (Service)	$\frac{\text{RRCConnectionSuccess}_{\text{service}}}{\text{RRCConnectionAttempt}_{\text{service}}} \times 100$	%
	E-RAB Setup Success Rate (VoIP)	$\frac{\text{VoIPERABSetupSuccess}}{\text{VoIPERABSetupAttempt}} \times 100$	%
Retainability	Call Drop Rate (VoIP)	$\frac{\text{VoIPERABAbnormalRelease}}{\text{VoIPERABRelease}} \times 100$	%
Mobility	Intra-frequency Handover Out Success Rate	$\frac{\text{IntraFreqHOOutSuccess}}{\text{IntraFreqHOOutAttempt}} \times 100$	%
Utilization	Resource Block Utilization Rate (DL)	$\frac{\text{DL\_RBUsed}}{\text{DL\_RBAvailable}} \times 100$	%
	Average CPU Load	MeanCPUUtility	%
Traffic	Average User Number	AveUserNumber	N/A
	Radio Bearers	RadioBearers_QCI_1_to_9	N/A
	DL Traffic Volume	DLTrafficVolume_QCI_1_to_9	bits
Service Integrity	Cell DL Average Throughput	$\frac{\text{CellDLTrafficVolume}}{\text{Duration} \times \text{CellDLTransmissionTime}}$	kbit/s
	Cell DL Maximum Throughput	$\frac{\text{MaxDLTrafficVolume}}{1\,\text{ms}} \times 1000$	kbit/s

Note: Formulas and counter names are illustrative and may vary depending on vendor implementation and network design [3].

The systematic calculation and monitoring of these KPIs using standardized formulas are vital for:

- **Performance Benchmarking**: Comparing performance across cells, regions, or time periods.
- **Troubleshooting**: Pinpointing root causes of degradation (e.g., high Call Drop Rate often linked to interference or poor handover parameters).
- **Automated Optimization**: Enabling algorithms, including ML-based ones, to monitor and adjust network parameters in near real-time. [14]

1.4.2 KPI Analysis Process

Performance monitoring in LTE networks relies on a comprehensive set of counters and indicators collected at different levels:

- **Network level** (overall performance trends)

- **MME level** (mobility and session management)
- **Cell level** (individual cell performance)

Reports are generated at varying intervals (busy hour, daily, weekly) to provide:

- **High-level overviews** for strategic planning.
- **Granular insights** for troubleshooting specific issues (e.g., call drops in a cluster).

As illustrated in Figure 1.7, the KPI analysis and tuning process follows a structured workflow:

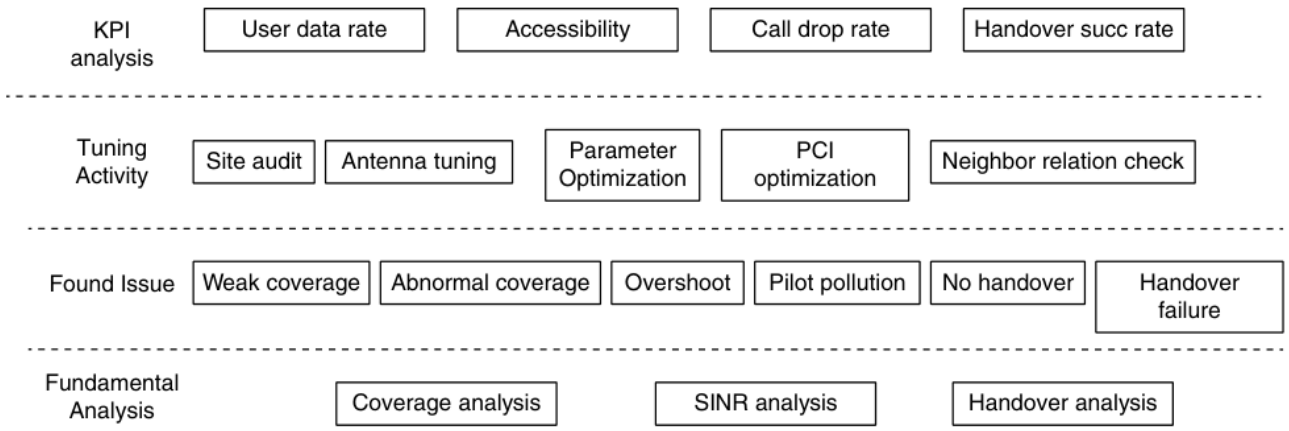


Figure 1.7: KPI analysis and tuning workflow. [3]

1. **KPI Monitoring**: Continuously track key performance indicators such as throughput, accessibility, retainability (e.g., drop rates), and mobility (e.g., handover success rates).
2. **Issue Identification**: Detect signs of degradation like low RSRP/SINR (indicating weak coverage), pilot pollution (caused by overlapping PCI signals), overshooting cells, high drop rates, or frequent handover failures.
3. **Root Cause Analysis**: Investigate performance issues using coverage maps, SINR distributions, interference levels, and parameter settings to determine underlying causes.
4. **Tuning Activities**: Implement corrective actions, categorized as follows:
   - **RF Optimization**: Modify antenna tilt, azimuth, and transmit power (e.g., RS Power) to enhance coverage, reduce interference, reinforce cell dominance, and mitigate pilot pollution. Key metrics: RSRP and SINR.[15]
   - **Parameter Optimization**: Adjust logical parameters within the eNodeB, such as handover thresholds (e.g., A3 offset, Time-to-Trigger, hysteresis), power control settings for PUSCH/PUCCH, and admission control strategies for congestion management.
   - **Capacity Optimization**: Enhance resource efficiency by optimizing PRB allocation, activating load balancing across carriers/layers, and applying Inter-Cell Interference Coordination (ICIC) techniques.
5. **Validation**: Reassess KPIs after tuning to confirm improvements and ensure no unintended performance regressions.

1.4.3 Data-Driven Methods for Network Optimization

Traditional LTE network optimization typically relies on manual configurations and rule-based adjustments by domain experts. However, with the growing scale and complexity of network infrastructures, data-driven approaches are increasingly adopted to complement and enhance these processes. These methods leverage large volumes of performance data—including KPIs, configuration parameters, and contextual information—to support more intelligent, adaptive, and efficient network management.

Key contributions of data-driven optimization include:

- **Predictive Analytics**: Anticipating traffic loads, throughput fluctuations, or potential KPI degradations to enable proactive resource allocation and congestion avoidance [15].
- **Anomaly Detection**: Identifying unusual patterns or abrupt changes in performance metrics that may indicate emerging faults or misconfigurations.
- **Automated Decision Support**: Recommending optimal parameter settings—such as handover margins or transmission power levels—based on historical trends and contextual performance data, contributing to the vision of **Self-Organizing Networks (SON)**.
- **Root Cause Analysis**: Facilitating diagnostics by linking observed degradations with likely underlying causes using historical correlations and multivariate analysis.

By shifting from reactive troubleshooting to proactive and predictive strategies, these approaches contribute to continuous performance improvement and real-time adaptability. This project integrates such techniques—especially forecasting and interpretability-focused analysis—to support informed optimization decisions within LTE environments.

KPI analysis and optimization form a cyclical process: monitoring → tuning → validation. While traditional methods (RF/parameter tuning) remain foundational, AI/ML introduces proactive and adaptive capabilities. The integration of these techniques ensures LTE networks meet evolving QoS demands efficiently.

1.5 State of the Art: Existing Work on LTE Optimization using AI

The application of Machine Learning (ML) and Deep Learning (DL) to enhance wireless network performance, particularly in LTE, has been an active area of research. Studies have explored various techniques to predict network behavior, optimize resource allocation, and improve Quality of Experience (QoE).

1.5.1 Comparative Table of Studies

The following table summarizes key studies applying ML/DL techniques to wireless network optimization, with a focus on LTE where applicable. The studies are ordered chronologically to illustrate the evolution of approaches over time, highlighting the type of problem addressed, data used, and learning algorithms employed.

Table 1.2: Comparative Analysis of Studies on ML/DL for Wireless Network Optimization

	Performance Improvement / Research Problem	Datatype	Input Data Examples	Learning Approach	Year	Ref
1	Network Perf. Prediction (Characterization)	Cellular, Synthetic	SINR, ICI, MCS, Tx Power	NN, Random NN	2015	[16]
2	Network Traffic Prediction -> Resource Allocation	Cellular, Real	Traffic (Bytes)/10 min	Hierarchical Clustering	2015	[17]
3	Network Traffic Prediction -> Resource Allocation	Cellular, Real	User traffic	MWNN	2015	[18]
4	QoE Prediction (KPI Parameters)	Cellular, Real	Mobile network KPIs	NN	2016	[19]
5	Network Traffic Prediction -> Resource Allocation	Cellular, Real	Mobile traffic volume	Regression Analysis	2016	[20]
6	Network Traffic Prediction -> Resource Allocation	Cellular, Real	Mobile traffic	SVM, MLPWD, MLP	2016	[21]
7	Network Traffic Prediction -> Resource Allocation	Cellular, Real	Avg traffic load/hour	LSTM, GSAE, LSAE	2017	[22]
8	Network Traffic Prediction -> Resource Allocation	Cellular, Real	CDRs (Milan Grid)	RNN, 3D CNN	2017	[23]
9	Network Traffic Prediction -> Resource Allocation	Cellular, Real	Traffic volume snapshots/10 min	STN, LSTM, 3D CNN	2018	[24]
10	Network Traffic Prediction -> Resource Allocation	Cellular, Real	Cellular traffic load/half-hour	LSTM, GNN	2018	[25]
11	Network Traffic Prediction -> Resource Allocation	Cellular, Real	CDRs/10 min interval	DNN, LSTM	2018	[26]

Continued on next page

Table 1.2 – continued from previous page

	Research Problem	Datatype	Input Data	Learning Approach	Year	Ref
12	Network Resource Allocation Prediction (WSN Parameters)	WSN, Synthetic	Lifetime, Power level, Distance	NN	2018	[27]
13	Network Traffic Prediction -> Resource Allocation	Cellular, Real	SMS and Call volume/10 min	CNN	2018	[28]
14	Network Traffic Prediction -> Resource Allocation	Cellular, Real	Traffic load/10 min	LSTM	2018	[29]
15	Network Traffic Prediction -> Resource Allocation (Feature Sel.)	Cellular, Real	Traffic logs/10 min	RF	2018	[30]
16	Network Traffic Prediction -> Resource Allocation	Cellular, Real	Traffic logs/15 min	LSTM	2019	[31]
17	Network Traffic Prediction -> Resource Allocation	Cellular, Real	Traffic load/5 min	3D CNN	2019	[32]
18	KPI Maximization (SINR) via Mobility Parameter Opt.	LTE, Simulation	CIO, HOM	CatBoost, Genetic Algorithm	2020	[33]
19	DL Throughput Optimization via Traffic Clustering	LTE-A, Real	312 cells, 20 KPIs	Regression, Unspecified	2020	[34]
20	Traffic Prediction (BS-level) -> Resource Allocation	Mobile Networks, Real	Traffic demand at BSs	Transformer, LSTM	2021	[35]
21	Traffic Prediction (eNodeB-level) -> Parameter Estimation	LTE, Real	6.2M time series, PRB util.	LSTM BiLSTM GRU fusion	2023	[36]
22	LTE Traffic Prediction (Edge Optimization)	LTE, Public	57 cells, time/date features	Bagging, RF, SVM	2023	[37]

Continued on next page

Table 1.2 – continued from previous page

	Research Problem	Datatype	Input Data	Learning Approach	Year	Ref
23	Multi-KPI Optimization (Throughput, PRB Usage)	LTE, Real	Network KPIs (3 months)	RF, SVR	2023	[38]
24	Short-Term KPI Forecasting (Rural Fixed Wireless)	Fixed Wireless LTE, Real	KPIs + environmental data	Seq2Seq, LSTM, GRU, RF	2023	[39]

Note: WSN = Wireless Sensor Network; CIO = Cell Individual Offset; HOM = Handover Margin; GA = Genetic Algorithm; SVR = Support Vector Regression; GSAE/LSAE = Autoencoders; GNN = Graph Neural Network; STN = Spatio-Temporal Network; MWNN = Morlet Wavelet NN

1.5.2 Synthesis of Limitations and Challenges

Analysis of the state-of-the-art reveals several key trends, limitations, and challenges:

1. **Dominance of Prediction Tasks**: The vast majority of studies (e.g., [17], [22]-[26], [28]-[32], [35]) focus on **predicting network traffic** (volume, load) or KPIs (QoE, throughput). This reflects the fundamental importance of forecasting for network planning and resource management.

2. **Shift Towards Deep Learning**: A clear trend exists from earlier reliance on traditional ML algorithms ([16]-[21], [30], [37]) towards **Deep Learning models**, particularly **LSTM** ([22], [24]-[26], [29], [31], [36], [39], [35]), **CNN** ([23], [28], [32]), and more recently **Transformers** ([35]) and hybrid/fusion models ([36]). This shift is driven by the superior ability of DL to capture complex spatio-temporal dependencies in network data.

3. **Gap Between Prediction and Optimization**: As highlighted in the preliminary SOTA review, while many studies achieve accurate predictions, **most stop short of integrating these predictions into automated optimization frameworks**. Optimization decisions are often left to human engineers who interpret the predictions. Only a few studies ([33], [34], [36], [38]) explicitly attempt to link prediction to parameter tuning or resource allocation optimization, and these are often specific to certain scenarios or KPIs.

4. **Challenge of Multi-KPI Optimization**: Optimizing for a single KPI (e.g., maximizing throughput) can negatively impact others (e.g., increase interference, degrade handover stability, or consume more energy). The literature acknowledges this challenge ([38]), but developing systematic frameworks that balance **multiple conflicting KPIs** simultaneously remains a significant hurdle. The interdependencies create complex trade-off scenarios that are difficult to model and solve comprehensively.

5. **Data and Model Generalizability**: Research often relies on specific datasets (real or simulated) from particular network environments ([23], [34], [36], [38], [39]). The lack of large-scale, standardized public datasets (unlike in fields like computer vision) makes direct comparison of model performance difficult. Furthermore, models trained on one network segment or time period may not generalize well to others without retraining or adaptation techniques.

6. **Interpretability for Actionable Insights**: While complex models like DL achieve high accuracy, understanding why they make certain predictions (i.e., identifying the most influential features/KPIs) is crucial for engineers to trust and act upon the recommendations. Techniques for **feature coefficient interpretation** or model explainability are essential but not always the focus.

Addressing the Gap: This project takes a foundational step towards bridging the identified gap between raw prediction and operational optimization. Instead of attempting to build a fully autonomous optimization engine, which faces the complex multi-KPI trade-off problem (Limitation 4), our work focuses on creating the critical prerequisites for intelligent network management. We address the identified gaps by:

1. **Delivering High-Fidelity Forecasts:** By developing and rigorously tuning advanced predictive models (LSTM), we provide the high-quality, reliable forecasting data that is essential for any downstream optimization task. This directly addresses the need for accurate prediction (Limitation 1) using state-of-the-art methods (Limitation 2).

2. **Enabling Proactive, Data-Driven Decisions:** The system utilizes these predictive insights to arm network engineers with the foresight needed to implement proactive optimization strategies, such as preemptive resource allocation, targeted maintenance scheduling, or identifying cells that are trending towards congestion. This focuses on the practical application of forecasts, moving closer to the prediction-to-optimization link (Limitation 3).

3. **Providing a Path to Interpretability:** While LSTMs are complex, the system is designed to facilitate future work on interpretability (Limitation 6). By providing a robust data foundation and a clear evaluation framework, the next step of applying techniques like SHAP or LIME to understand model predictions becomes feasible.

This approach provides a pragmatic and powerful contribution: a validated system that delivers the actionable intelligence required for engineers to optimize networks more effectively, paving the way for more automated solutions in the future.

Conclusion

This chapter provided an overview of LTE fundamentals and essential KPIs, highlighting the growing importance of AI in modern network optimization strategies. It emphasized the gap between predictive insights and their effective application, setting the stage for the solution proposed in this project. In the next chapter, we will delve into the problem statement and present the methodology adopted to address these challenges.

Chapter 2

Methodology

This chapter presents the project's methodological framework. It defines the core challenges in multi-vendor LTE environments and proposes a predictive analytics solution. It also outlines the AI model selection and the tech stack used.

2.1 Problem Statement

2.1.1 Problem Overview

In modern telecommunications networks, predicting key performance indicators (KPIs) such as traffic, throughput, and anomalies—both hourly and daily, and across multiple geographical levels—is critical for network optimization and proactive decision-making. At the heart of this challenge lies the need for clean, unified, and timely data.

Our case study is based on Djezzy, a leading telecommunications operator in Algeria. Djezzy's infrastructure spans multiple vendors (e.g., Huawei, Nokia, ZTE) and technologies (2G, 3G, 4G), leading to a fragmented data environment. Each vendor provides proprietary tools and data formats, which makes it difficult for engineers to access and analyze KPIs efficiently across the network.

To address this, our approach is to design and implement a unified data warehouse that consolidates and standardizes KPI data from different vendor-specific systems. This warehouse serves as the foundation for applying AI and Machine Learning models to forecast network performance metrics, detect anomalies, and support optimization efforts.

Furthermore, we have developed a full-stack web application to visualize the outcomes of our data-driven predictions, anomaly detection, and other analytics. The application not only offers intuitive charts and reports but also facilitates real-time interaction with the model outputs, enabling engineers and decision-makers to gain insights more effectively.

Figure 2.1: Djezzy Telecommunications Company

2.2 Insights from the State of the Art

2.2.1 Trends in Research: Focus on Prediction Rather than Optimization

The review of existing literature highlights several critical trends that shaped our methodological choices. Initially, our objective was to develop optimization strategies for KPI improvement. However, the state of the art reveals that most recent efforts in the domain are centered around prediction tasks, particularly traffic and throughput forecasting, which are crucial for network planning and proactive decision-making.

Studies overwhelmingly focus on time-series prediction of network performance metrics using machine learning and deep learning models. This reflects the growing consensus that accurate prediction is a prerequisite for effective optimization. Moreover, the trend has moved toward using advanced architectures such as LSTMs, CNNs, and Transformers to capture complex spatio-temporal patterns.

2.2.2 Multi-KPI Trade-offs: A Complex Balancing Problem

Another important insight from our literature synthesis is the difficulty of optimizing multiple KPIs simultaneously. In real-world telecom environments, improving one KPI—such as throughput—can lead to the degradation of others like energy efficiency or handover stability. This interdependence creates complex trade-offs that are hard to formalize in an optimization framework.

Only a handful of works attempt to handle these trade-offs directly, and most are limited in scope or applicability. The majority of systems leave decision-making to human engineers, who must interpret predictions and balance these conflicting metrics manually.

2.3 Proposed Solution

2.3.1 Shifting from Direct Optimization to Predictive Analytics for Decision Support

Based on these findings, we chose to pivot from a direct optimization objective toward developing a robust predictive analytics pipeline. Rather than automate decisions, our system aims to empower network engineers by providing accurate, timely, and interpretable predictions that can guide optimization choices. This approach aligns with how decisions are made in practice and avoids the pitfalls of incomplete or oversimplified optimization models.

2.3.2 Choice of AI Models: LSTM and Prophet Models

To implement our predictive solution, we selected a dual-model strategy that emphasizes both performance and interpretability for time-series forecasting in the context of telecommunications KPI data. At the core of our approach is a Long Short-Term Memory (LSTM) network, a deep learning model renowned for its ability to capture non-linear temporal dependencies and long-range patterns in sequential data [40]. Its internal gating mechanisms make it particularly effective at modeling the complex daily and weekly seasonalities inherent in KPIs such as traffic volume and resource utilization.

To provide a robust benchmark beyond simple naive methods, we also selected Facebook's Prophet model. Prophet serves as an advanced statistical baseline, representing a strong,

modern approach to time-series decomposition [41]. It models time series data as an additive combination of trend, seasonality (daily, weekly), and holiday effects. Its strengths lie in interpretability, resilience to missing data, and minimal tuning requirements, making it a reliable comparative framework.

By evaluating our LSTM model against Prophet, we move beyond comparisons with simplistic heuristics. This positioning allows us to demonstrate that the deep learning model not only captures complex patterns, but also outperforms a sophisticated, production-grade forecasting tool. This reinforces both the value and necessity of deep learning in capturing the nuanced dynamics of telecommunication KPIs.

2.3.3 Role of Prediction in Network optimisation

By forecasting traffic load and throughput at both hourly and daily levels—and across multiple geographical granularities—our models help engineers:

- Anticipate congestion and proactively reallocate resources.
- Monitor and detect anomalies more effectively.
- Make informed decisions about infrastructure scaling or reconfiguration.

These predictive insights serve as the foundation for further optimization, whether manual or algorithmic, and contribute directly to network stability, performance, and customer satisfaction.

2.4 Tools and Technologies Used in Our Solution

2.4.1 Data Warehousing and ETL for Telecom Analytics

Principles and Best Practices

Data warehousing involves integrating data from multiple heterogeneous sources into a centralized, structured repository optimized for querying and analysis. In telecom, this is particularly essential due to the diversity of systems and data formats across vendors.

Our ETL (Extract, Transform, Load) pipeline was designed to:

- Automate extraction of raw KPIs from various vendor-specific tools.
- Transform and standardize the data schema for consistency.
- Load the unified data into a central warehouse on a scheduled basis.

This enables a consistent view of network data, suitable for both operational monitoring and analytical modeling.

Database Technologies and Deployment

We used **PostgreSQL** as our primary database engine for storing the standardized KPI data, chosen for its reliability, open-source nature, and strong support for analytical queries.

The database was accessed and managed using:

- **pgAdmin**: A web-based graphical interface for PostgreSQL database management.

- **psql shell**: The command-line interface for executing SQL scripts and managing database operations.
- **PL/pgSQL**: Used to implement stored functions and procedures for database-side data processing and automation within PostgreSQL.

To containerize and deploy our data warehouse environment, we used **Docker Desktop**, which allowed us to ensure a consistent and portable environment across development and production.

Automation with Apache Airflow

We used **Apache Airflow** to automate the ETL pipeline and schedule data workflows see B.1 for more details. Airflow enabled us to:

- Define Directed Acyclic Graphs (DAGs) for task execution.
- Monitor the success or failure of each step in the pipeline.
- Schedule regular data ingestion and transformation jobs.

Data Processing and Scripting with Python

Python served as the main scripting language throughout the project. We leveraged its ecosystem of packages to handle data processing, modeling, and integration tasks. Key libraries included:

- `pandas`, `numpy` for data manipulation and preprocessing.
- `psycopg2` for database connections.
- `scikit-learn`, `xgboost`, etc., for AI/ML model training and prediction.
- `matplotlib`, `seaborn` for internal data visualization during development and debugging.

2.4.2 Full-Stack Web Application Development for Data Products

To make our AI/ML models accessible and interpretable, we built a web-based data product that allows engineers and analysts to interact with predictions and insights in real-time.

Our tech stack includes:

- **Flask**: A lightweight Python web framework for backend APIs, model serving, and data access.
- **Chart.js**: For rendering interactive charts and visualizing KPIs, predictions, and anomalies.
- **HTML/CSS/JavaScript**: For building a responsive, user-friendly frontend.

The web application allows users to:

- Visualize historical and predicted KPI trends.
- Monitor anomalies flagged by AI models.
- Explore KPI data at various temporal and geographic levels.

Conclusion

This chapter introduced a structured approach to addressing LTE network optimization using AI techniques. It defined the core problem, justified the proposed solution, and outlined the main tools and technologies involved. Building on this foundation, the following chapters will focus on the implementation process, covering the system's design, development, and practical integration.

Chapter 3

Design and Implementation of the Data Management Subsystem

This chapter details the design and implementation of the data management subsystem, a core component of our Key Performance Indicator (KPI) platform. Crucially, this subsystem not only supports traditional network performance monitoring and reporting but also serves as the **foundational dataset for advanced Artificial Intelligence models** designed for KPI forecasting, prediction, and anomaly detection. The accuracy, reliability, and structure of the data managed herein are therefore paramount to the success of the predictive capabilities of the overall system. We will cover the overall architecture with a focus on data flow, the rationale and specifics of the Data Warehouse (DWH) design tailored to these analytical and ML requirements, and the implementation of the Extract, Transform, Load (ETL) pipeline responsible for populating this critical data resource.

3.1 Overall System Architecture

The system architecture is designed to ingest, process, store, and prepare telecommunications network performance data for analysis and reporting. The data flows through several key stages, coordinated by a set of automation tools and scripts, as depicted in Figure 3.1.

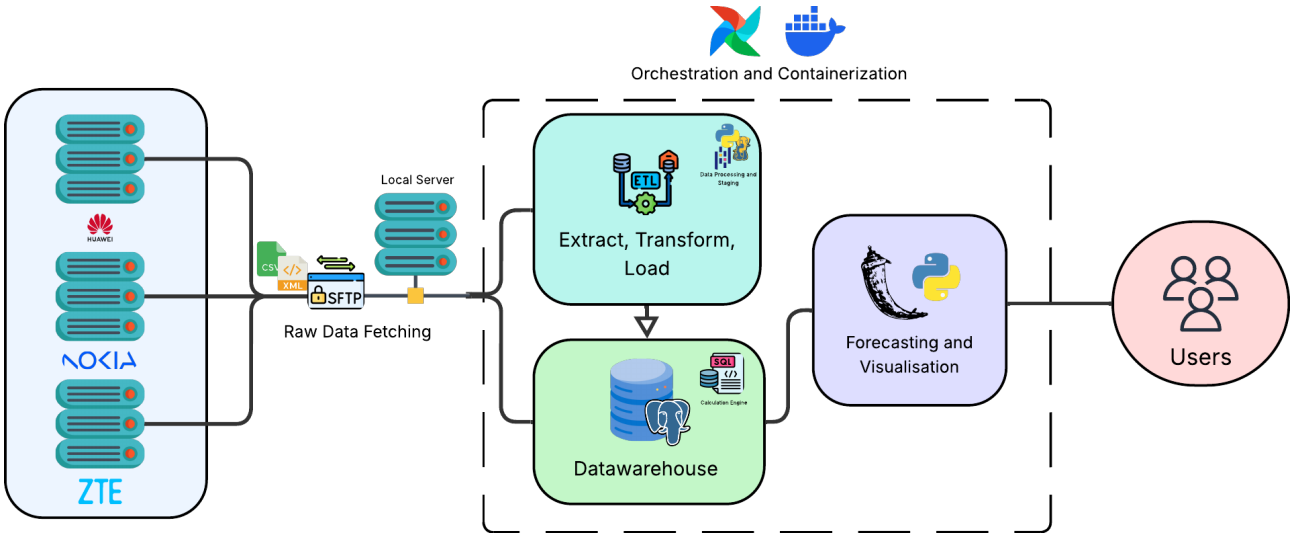


Figure 3.1: High-Level System Architecture illustrating Data Flow.

The primary components and data flow are as follows:

- **Data Sources:** Raw performance counter data is collected from various network elements across multiple mobile technologies and vendors. Each vendor (e.g., Huawei, Nokia, ZTE) uses its own management systems to store this data, typically in CSV or XML formats. Our deployment server securely accesses these sources and retrieves the data periodically, consolidating it into a central landing zone for processing. This setup ensures compatibility with the diverse architectures and formats used by different vendors.
- **Staging Area:** Initially, vendor-specific data is transformed into a standardized format by vendor-specific processing scripts and then loaded into a transient staging table `Staging_RawCounter` within the PostgreSQL Data Warehouse. This step is managed by Python scripts (e.g., `Loader.py`).
- **ETL Processing Core:** Python scripts (e.g., `ETL.py`) further process data from the staging area.
  This involves:
  - Data validation and cleansing.
  - Enrichment with dimensional keys (e.g., mapping cell IDs, timestamps, vendor and technology identifiers).
  - Loading into the primary hourly fact table `FactRawCounter`.
- **Data Warehouse (DWH):** The PostgreSQL-based DWH follows a dimensional modeling approach to organize and store data efficiently.
  - **Dimension Tables:** Capture the contextual attributes of the data, such as network elements, time, geography, vendors, and technologies (e.g., `DimCell`, `DimTime`, `DimGeography`, `DimVendor`, `DimTechnology`).
  - **Definition Tables:** Contain metadata definitions for KPIs, counters, and associated calculation logic (e.g., `KPIDefinition`, `KPIFormula`).
  - **Fact Tables:** Store measurable observations, including both raw counters and computed KPIs, at different levels of granularity (e.g., `FactRawCounter`, `FactKPI`, `FactAggregatedKPI`, `FactBusyHour`).

  A detailed description of each of these tables and their relationships will be provided later in the Data Warehouse Design section.
- **Calculation Engine:** A suite of SQL scripts and PL/pgSQL functions within PostgreSQL perform
  aggregations (e.g., daily, geographic) and KPI calculations based on the formulas stored in the definition tables.
- **Orchestration:** Apache Airflow, running via Docker Compose, manages the entire pipeline,
  scheduling tasks, handling dependencies, and ensuring the ordered execution of data loading, ETL,
  and calculation scripts.

3.1.1 Project File Structure for Data Management

The implementation of the data management subsystem required a modular and maintainable project architecture. A well-structured organization of scripts, configuration files, and datasets was established to support robust ETL workflows and ensure scalability. Figure 3.2 presents the directory layout of the `Database/` folder, which encapsulates all components related to data ingestion, ETL execution, DWH schema creation, and orchestration processes.

Figure 3.2: Directory structure of the `Database/` component

This architecture promotes separation of concerns and improves reusability across the ETL pipeline. The main directories and files are described below:

- `dags/`: Contains Apache Airflow DAG definitions (e.g., `all_pipeline_vf.py`) that orchestrate the end-to-end ETL processes.
- `Def/`: Stores metadata definition files (e.g., `counter_definitions.csv`, `kpi_formula.csv`) alongside Python scripts (e.g., `inject_counter.py`, `inject_formula.py`) for loading this metadata into the data warehouse.
- `Dim/`: Includes CSV files and ingestion scripts for dimension tables such as `dim_cell.csv` and `dim_geo.csv`, which serve as the foundation for KPI aggregation and analysis.
- `Fact/`:
  - `data/`: Subdirectories for raw inputs, processed outputs, and logs specific to vendor data ingestion.
  - SQL scripts (e.g., `FactKPI.sql`, `FactAggDaily.sql`) for creating and populating fact tables.
- `utils/`: Utility scripts supporting tasks such as data extraction (`extractor.py`) and source-to-standard mapping (`source_mappings.py`).
- Root-level scripts and configuration files:
  - `dbcreation.sql`: Defines the database schema.
  - `Loader.py`, `ETL.py`: Central orchestration and transformation scripts.
  - `docker-compose.yaml`: Specifies the containerized deployment environment see A.1 for more details.

The resulting structure enhances readability, streamlines development workflows, and supports maintainable scaling of the data pipeline.

3.2 Data Warehouse Design and Rationale

The DWH is implemented in PostgreSQL and employs a dimensional modeling approach. This design was chosen not only to optimize for query performance and analytical flexibility for human users but, critically, to provide a **structured, reliable, and easily accessible data source for training and deploying machine learning models**. The schema is engineered to facilitate feature extraction and to ensure that historical data, essential for model learning, is accurately captured and maintained.

3.2.1 Dimensional Modeling Approach

The Data Warehouse (DWH) primarily utilizes a **Star Schema**, which simplifies analytical queries by connecting central fact tables directly to surrounding dimension tables via foreign key relationships. This design enables fast and intuitive data access for both reporting and machine learning tasks.

A minor **Snowflake Schema** element is introduced in the `DimGeography` dimension, which contains a hierarchical structure (e.g., Commune, Wilaya, AllNet) implemented through a self-referencing foreign key. This balances the need for normalized geographic representation with query performance. The Entity-Relationship Diagram (ERD) in Figure 3.3 illustrates the overall schema structure.

This star schema is particularly advantageous for AI applications. It allows for efficient feature engineering by enabling straightforward joins between the central fact tables (containing the target KPI values) and the dimension tables, which provide rich contextual features—such as time of day, cell location, and technology type—critical for training predictive models. The model also facilitates aggregation, filtering, and slicing operations with minimal query complexity, which is essential during iterative model development.

To further optimize performance, particularly during repeated joins and spatial aggregations, materialized views were used. For instance, the `mv_cell_geo` view pre-joins cell and geography information to accelerate region-based queries, benefiting both reporting and feature engineering pipelines.

3.2.2 Detailed Schema

The DWH schema, comprehensively defined in the `dbcreation.sql` script, consists of several key dimension, definition, and fact tables. This structure is designed to be robust and accessible for various database management and development tasks. The DWH can be interacted with, queried, and modified through multiple interfaces, including:

- Direct SQL commands via a command-line interface like PostgreSQL's 'psql' shell (demonstrated in Figure 3.5 for querying Definition data).
- Graphical user interfaces such as pgAdmin, which provides a comprehensive suite for database administration, schema browsing (as seen in Figure 3.4).
- Integrated development environment (IDE) extensions, SQLTools for Visual Studio Code, which allows for convenient querying and interaction with the database directly within the development environment (an example of querying `FactKPI` using SQLTools is shown in Figure 3.7).

This flexibility in access ensures that developers, administrators, and analysts can efficiently work with the DWH using their preferred tools.

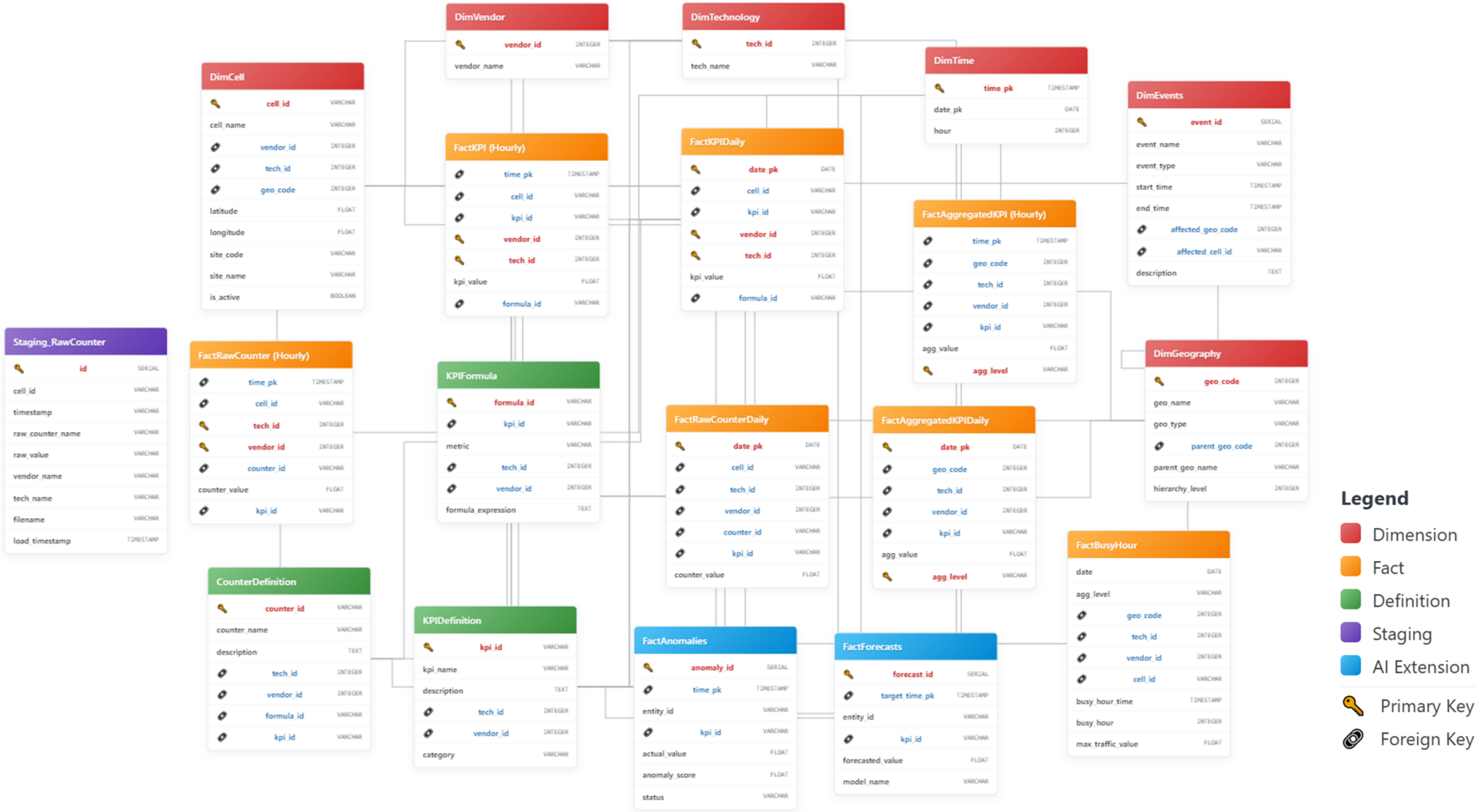


Figure 3.3: Data Warehouse Entity Relationship Diagram

Key Dimension Tables:

- `DimCell`: Stores information about individual network cells/sectors, including their association with vendors, technologies, and geographical locations. Figure 3.4 shows a view from pgAdmin, demonstrating the successful creation of the DWH tables and a sample query on the 'DimCell' table, which serves as the master reference for all network cells.

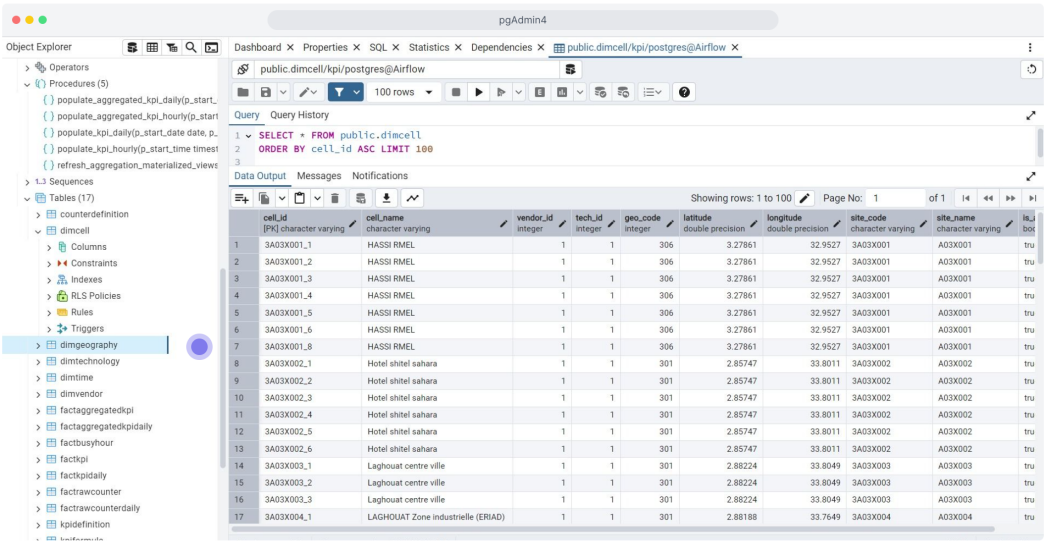


Figure 3.4: DWH Tables in pgAdmin and DimCell Query Example

- `DimTime`: Provides time-based context at an hourly granularity. It includes derived attributes
  like `date_pk` for efficient date-based joins.
- `DimGeography`: Manages the geographical hierarchy (Commune, Wilaya, AllNet) using `geo_code` and `parent_geo_code`.
- `DimVendor`: Lists network equipment vendors (e.g., Huawei, Nokia, ZTE) and an `ALL` category for network-wide aggregations.
- `DimTechnology`: Catalogues network technologies (e.g., 2G, 3G, 4G).

Key Definition Tables:

To create a flexible and maintainable system, we chose to store business logic—specifically the definitions of counters and KPIs—as data within the DWH itself, rather than hard-coding them in application logic. This metadata-driven approach allows for dynamic updates without code changes. The command-line interface 'psql' was used for scripted ingestion and quick verification of this definitional data. Figure 3.5 shows a query against the 'CounterDefinition' table via 'psql', confirming that the raw counter metadata, which forms the basis of all KPI calculations, is correctly loaded.

- `KPIDefinition`: Defines KPIs, their names, descriptions, and applicability to vendors/technologies.
- `CounterDefinition`: Contains details of vendor-specific raw counters that form the basis of KPIs.

```
kpi=# SELECT * FROM counterdefinition LIMIT 10;

 counter_id     | counter_name   | description                          | tech | vendor | formula_id        | kpi_id
----------------+----------------+--------------------------------------+------+--------+-------------------+-------------------------
 L.Thrp.bits.DL | L.Thrp.bits.DL | Huawei DL traffic bits counter       |  0   |   0    | F_HW_Traffic_DL   | LTE_Traffic_Volume_DL
 C373343806     | C373343806     | DL Volume High (Mbit)                |  0   |   1    | F_ZTE_Traffic_DL  | LTE_Traffic_Volume_DL
 C373343807     | C373343807     | DL Volume Low (kbit)                 |  0   |   1    | F_ZTE_Traffic_DL  | LTE_Traffic_Volume_DL
 L.Thrp.bits.UL | L.Thrp.bits.UL | Huawei UL traffic bits counter       |  0   |   0    | F_HW_Traffic_UL   | LTE_Traffic_Volume_UL
 C373343804     | C373343804     | UL Volume High (Mbit)                |  0   |   1    | F_ZTE_Traffic_UL  | LTE_Traffic_Volume_UL
 C373343805     | C373343805     | UL Volume Low (kbit)                 |  0   |   1    | F_ZTE_Traffic_UL  | LTE_Traffic_Volume_UL
 L.Thrp.bits.DL | L.Thrp.bits.DL |                                      |  0   |   0    | F_HW_Traffic      | LTE_Traffic_Volume
 C373343806     | C373343806     | DL Volume High (Mbit)                |  0   |   1    | F_ZTE_Traffic     | LTE_Traffic_Volume
 C373343807     | C373343807     | DL Volume Low (kbit)                 |  0   |   1    | F_ZTE_Traffic     | LTE_Traffic_Volume
 L.Thrp.bits.UL | L.Thrp.bits.UL |                                      |  0   |   0    | F_HW_Traffic      | LTE_Traffic_Volum
(10 rows)
```

Figure 3.5: Querying CounterDefinition Table via PSQL CLI.

- `KPIFormula`: Stores the mathematical expressions used to calculate KPIs from raw counters. Sample data for this table is shown in Figure 3.6.

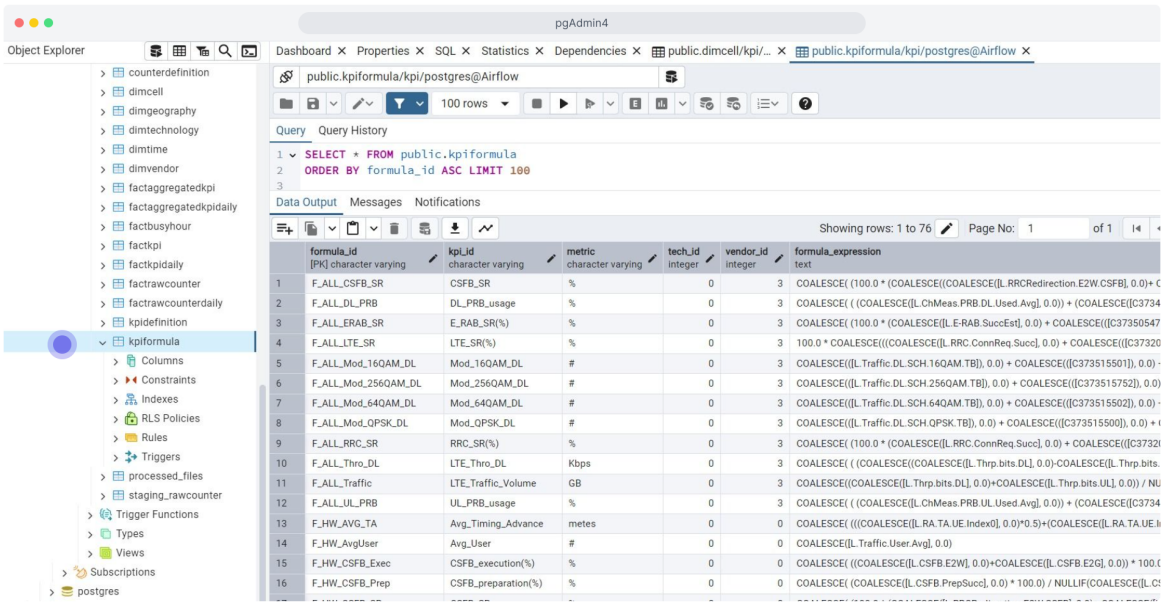


Figure 3.6: Querying from KpiFormula table.

Key Fact Tables:

Fact tables store the quantitative measurements and are the primary source of data for AI models. To facilitate a modern development workflow, we utilized IDE integrations like SQLTools for Visual Studio Code, allowing for seamless interaction with the database during coding and debugging. Figure 3.7 shows an example of querying the 'FactKPI' table using this tool. This table represents the culmination of the ETL process at the hourly, per-cell level, providing the clean, structured time-series data that directly feeds our forecasting models.

- `FactRawCounter`: The most granular fact table, storing raw counter values at the hourly level per cell. It includes denormalized `tech_id`, `vendor_id`, and `kpi_id` for performance.
- `FactKPI`: Stores calculated KPI values at the hourly granularity per cell; This table is a primary source for historical KPI trend.

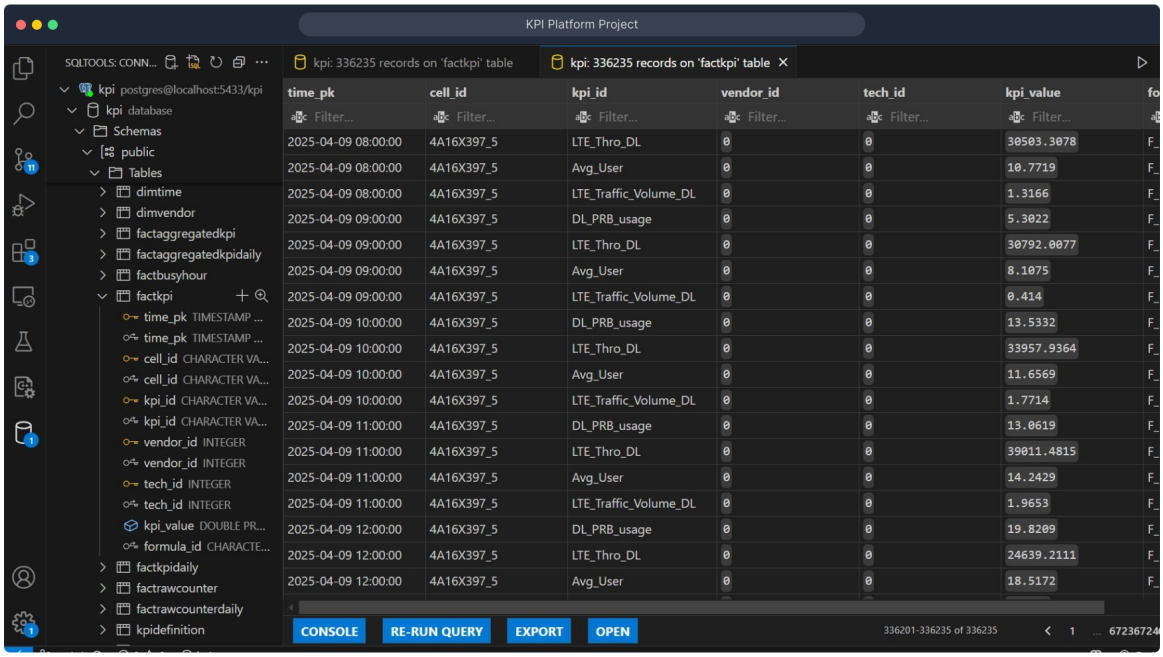


Figure 3.7: Querying FactKPI Table using SQLTools in VS Code.

- `FactAggregatedKPI`: Contains KPI values aggregated geographically (Commune, Wilaya, AllNet)
  at hourly granularity.
- `FactBusyHour`: Identifies the busy hour for various aggregation levels (Cell, Commune, Wilaya, AllNet) per day, based on a maximum specific traffic KPI.
- Daily counterparts such as `FactRawCounterDaily`, `FactKPIDaily`, and `FactAggregated KPIDaily`
  store data aggregated to a daily level.

3.2.3 Data Granularity and Aggregation Strategy

The DWH supports multiple levels of granularity to cater to different analytical needs and, crucially, to provide suitable inputs for various ML modeling approaches:

- **Hourly Granularity:** Raw counters (`FactRawCounter`) and KPIs (`FactKPI`, `FactAggregatedKPI`) are primarily stored at the hourly level. This level of detail is essential for training time-series models for short-term forecasting, fine-grained anomaly detection, and precise busy hour identification.
- **Daily Granularity:** Dedicated tables (`FactRawCounterDaily`, `FactKPIDaily`, `FactAggregatedKPIDaily`) store data aggregated to the daily level. This not only improves query performance for daily reporting and trend analysis but can also **serve as input for ML models focused on longer-term patterns, or models where daily resolution is more appropriate or computationally efficient**.

The aggregation strategy involves first summing the necessary raw counters (e.g., for daily totals or geographic roll-ups) and then recalculating the relevant KPIs based on these aggregated counters. This ensures accuracy, as many KPIs are ratios or averages that cannot be

simply summed from their lower-level values. The logic for these complex aggregations and recalculations is encapsulated within a suite of PL/pgSQL functions and procedures stored directly in the database. These database routines perform the heavy lifting of data manipulation needed to derive the aggregated fact tables. Figure 3.8 shows an example of such a database function, `populate_agg_kpi_hourly()`, viewed within the pgAdmin interface, illustrating how procedural SQL is employed to implement this aggregation logic. These routines are subsequently invoked by the SQL scripts orchestrated by Airflow.

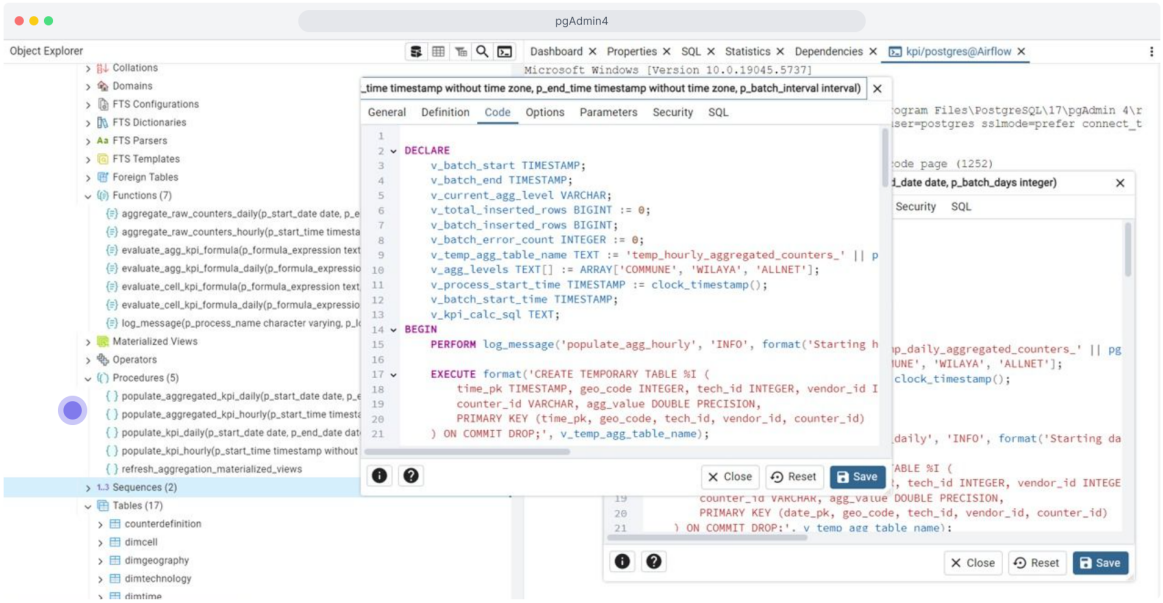


Figure 3.8: View of PL/pgSQL Code for `populate_agg_kpi_hourly()` in pgAdmin.

`the populate_agg_kpi_hourly()` function is an example of the stored procedures used to implement the DWH's data aggregation strategy, encapsulating the logic for calculating aggregated KPIs.

3.2.4 Indexing, Materialized Views, and Performance Considerations

To ensure high performance in the Data Warehouse (DWH)—essential for both interactive analysis and AI data preparation—several optimization strategies were applied, particularly given the constraints of a local development environment.

- **Indexing:**
  Indexes on primary and foreign keys facilitate efficient joins between fact and dimension tables. Additional indexes are created on frequently queried columns such as `time_pk`, `cell_id`, and `vendor_id`. Composite indexes, like on `FactRawCounter(time_pk, cell_id, vendor_id, counter_id)`, further improve performance for multi-column query patterns.

  These indexing strategies significantly reduce query times and ETL processing overhead. For instance, tables like `FactKPI` and `FactKPIDaily` handle over 336,000 and 220,000 records respectively (see Figure 3.9), while the Airflow DAG `all_kpi_pipeline_v2` completes KPI computations in approximately 21 minutes (see Figure 3.11).

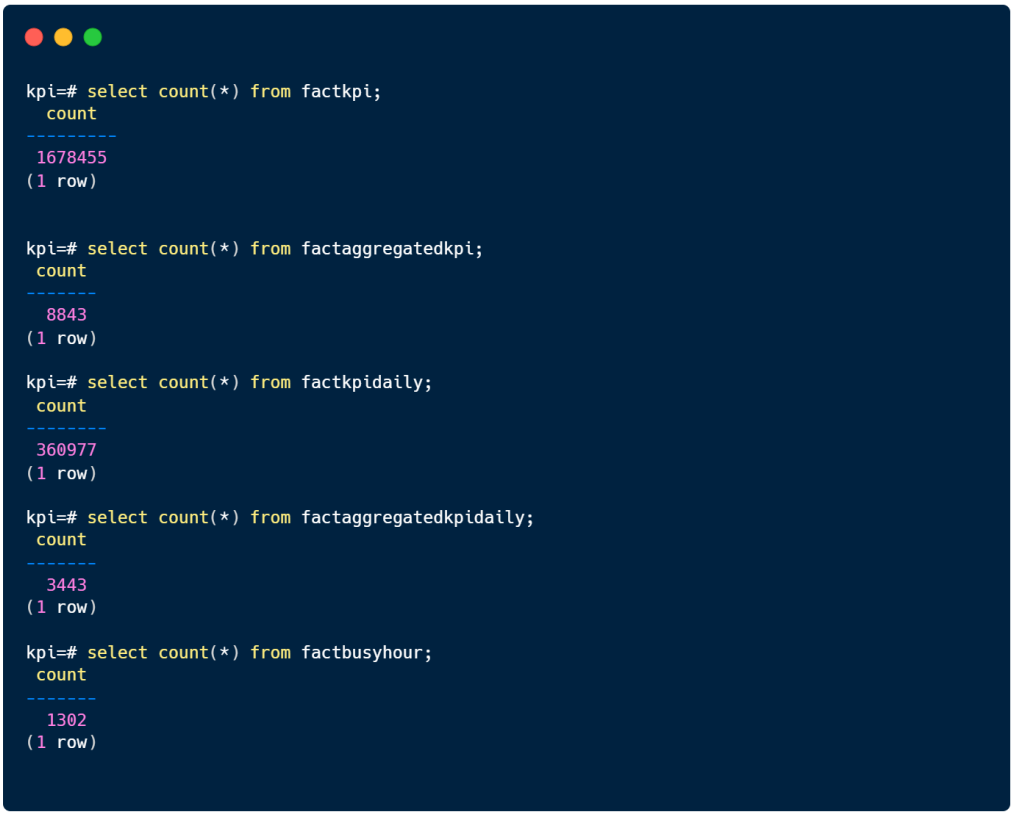


Figure 3.9: Fact Table Row Counts Indicating DWH Data Scale.

- **Denormalization:**
  To reduce join overhead, key attributes such as `tech_id` and `vendor_id` are denormalized into fact tables. This allows faster filtering and aggregation without the need to repeatedly access dimension tables like `DimCell`.

- **Materialized Views (MVs):**
  MVs store precomputed results of frequent or costly queries, improving performance and reducing real-time computation needs. These are refreshed during the ETL process. Notable examples include:

  - `mv_cell_geo`: Joins `DimCell` with `DimGeography` to enable fast geographic aggregations (see Figure 3.10).
  - `mv_daily_network_kpi`: Computes daily averages of KPIs across the entire network.
  - `mv_hourly_site_kpi`: Aggregates KPIs by site on an hourly basis.

```
kpi=# SELECT * FROM public.mv_cell_geo;
     cell_id     | tech_id | vendor_id | commune_geo_code | wilaya_geo_code | allnet_geo_code
-----------------+---------+-----------+------------------+-----------------+-----------------
 CELL_101_0      |       0 |         3 |              101 |               1 |               0
 CELL_101_1      |       1 |         3 |              101 |               1 |               0
 CELL_101_2      |       2 |         3 |              101 |               1 |               0
 4001M007_1      |       0 |         1 |              101 |               1 |               0
 4001M007_2      |       0 |         1 |              101 |               1 |               0
 4001M007_3      |       0 |         1 |              101 |               1 |               0
 4001M007_4      |       0 |         1 |              101 |               1 |               0
 4001M007_5      |       0 |         1 |              101 |               1 |               0
 4001M007_6      |       0 |         1 |              101 |               1 |               0
 4001M008_1      |       0 |         1 |              101 |               1 |               0
 4001M008_2      |       0 |         1 |              101 |               1 |               0
 4001M008_3      |       0 |         1 |              101 |               1 |               0
 4001X001_1      |       0 |         1 |              101 |               1 |               0
 4001X001_2      |       0 |         1 |              101 |               1 |               0
```

Figure 3.10: Sample Data from `mv_cell_geo` Materialized View.

- **Aggregation Tables:**
  Tables such as `FactAggregatedKPI`, `FactKPIDaily`, and `FactBusyHour` store pre-aggregated metrics, significantly improving query performance by shifting complex computations to the ETL stage. The Airflow Gantt chart in Figure 3.11 highlights how these optimizations enable efficient processing at scale.

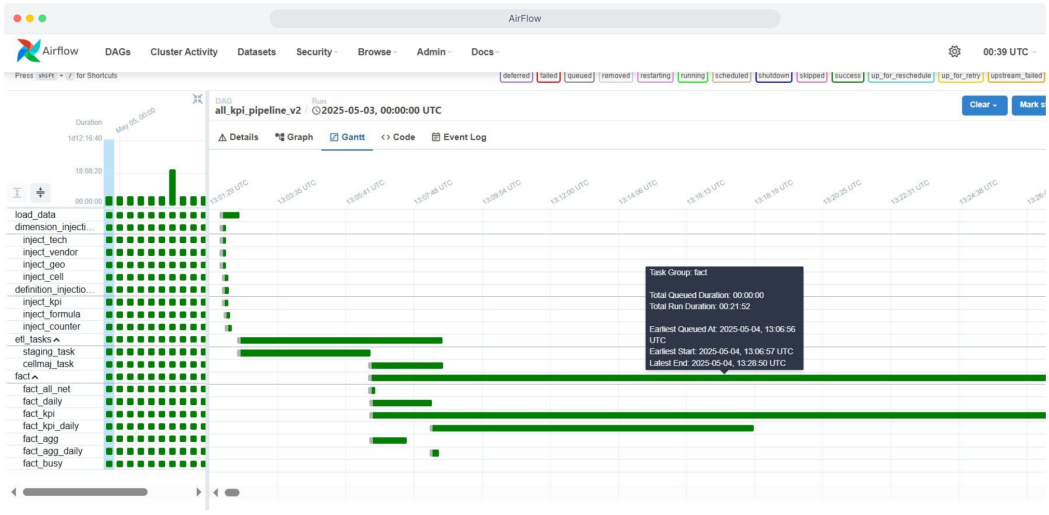


Figure 3.11: Apache Airflow Gantt chart

The Apache Airflow Gantt chart is showing the execution duration of the `all_kpi_pipeline_v2` DAG. The 'fact' task group, responsible for main KPI computations, completed in approximately 21 minutes. This efficient processing at scale is a direct result of the DWH optimizations, including pre-aggregated tables and materialized views, which shift computational load from query time to the ETL stage.

3.3 ETL Pipeline Implementation

The ETL pipeline is a multi-stage process responsible for extracting data from source files, transforming it into a usable format, calculating KPIs, and loading it into the DWH.

3.3.1 Data Extraction and Staging

1. **Vendor-Specific Pre-processing:** Raw performance counter data originates from various network elements across multiple mobile technologies and vendors. Key sources include Radio Network Controllers (RNCs) and Base Station Controllers (BSCs) for 2G, RNCs and NodeBs for 3G, and eNodeBs for 4G networks. Each vendor (e.g., Huawei, Nokia, ZTE) operates its own Operations Support System (OSS)/Element Management System (EMS), often comprising one or more servers (e.g., Huawei may use multiple servers), which collect and store this raw data. Our deployment server has secure access to these vendor-specific servers, from which data is periodically retrieved. This data, typically in vendor-specific CSV or XML formats (as illustrated in Figure 3.12), is consolidated in a central landing zone before being processed by our ETL pipeline. This approach accommodates the diverse data collection architectures of different vendors.

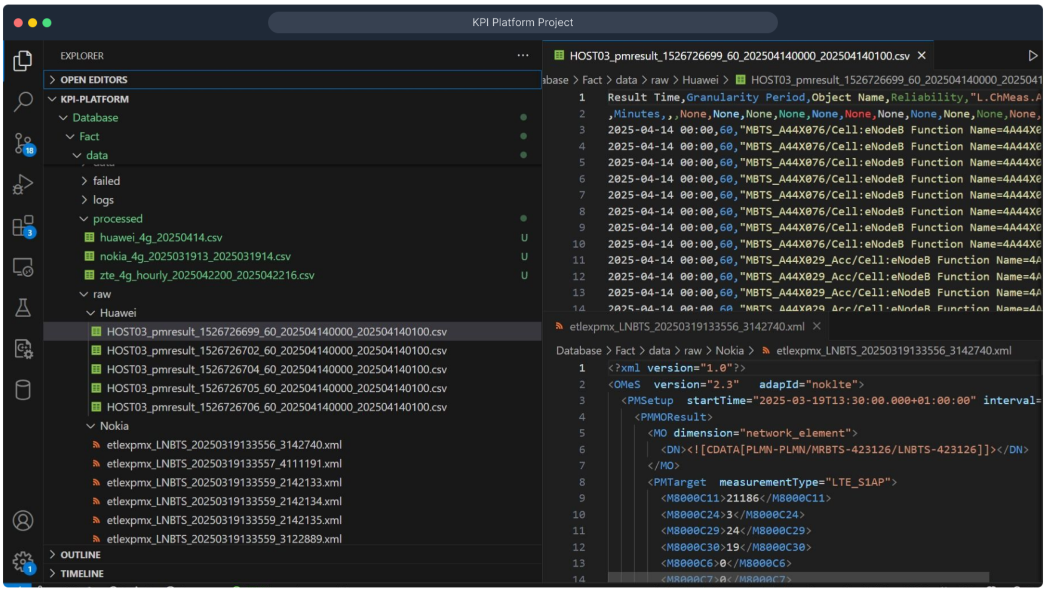


Figure 3.12: Raw data from Hewaei and Nokia.

2. **Staging Load (`Loader.py`):**
After initial retrieval from vendor systems, raw data files, which often vary in format and structure (as seen in Figure 3.12), undergo a pre-processing step. Vendor-specific Python scripts (e.g., for Huawei, Nokia, ZTE) parse these heterogeneous files and transform them into a **standardized, common format**, typically a consistent CSV structure. Examples of such standardized data files are shown in Figure 3.14.

   This standardized data is then loaded into a dedicated transient table within the PostgreSQL Data Warehouse called `Staging_RawCounter`. This table serves several critical purposes in the ETL pipeline:

   - **Decoupling and Buffering:** Acts as a buffer between raw file loading and later

transformations. This decouples ingestion from processing, so if downstream tasks fail, data stays in staging for reprocessing without needing re-ingestion.

- **Initial Validation and Light Cleansing:** Though focused on bulk loading, basic checks or light cleansing can be applied as data enters `Staging_RawCounter`. Its schema uses generic types (e.g., `VARCHAR`) to handle early-stage inconsistencies.
- **Unified Source for Core ETL:** `Staging_RawCounter` serves as the consistent input for `ETL.py`, regardless of vendor or file format, simplifying ETL logic with a standardized (long format) structure.
- **Optimized for Bulk Loading:** Data is inserted using fast bulk operations (e.g., PostgreSQL's `COPY` via `Loader.py`), which are significantly faster than row-wise inserts. See Figure 3.15 for an example.
- **Interim Traceability and Auditing:** Records may include metadata (e.g., source filename, load timestamp), supporting traceability during processing before final DWH integration.

The Python script `Loader.py` is responsible for populating the `Staging_RawCounter` table from these standardized data files. It robustly handles file ingestion by performing checksum verification to prevent duplicates, processing large files in manageable chunks for memory efficiency, and using fast bulk loading operations (like PostgreSQL's 'COPY' command) to insert data into the staging table. Upon successful, transactionally-sound loading, source files are moved to a 'processed' directory. The detailed operation of `Loader.py`, including file handling, chunk processing, and summary statistics, is illustrated in its execution logs (see Figure 3.13).

```
---------- Processing file: zte_4g_hourly_2025042200_2025042216.csv ----------
Starting processing for file: 'zte_4g_hourly_2025042200_2025042216.csv'
'zte_4g_hourly_2025042200_2025042216.csv' Checksum: 2b138be24f186db78db1d1312d22c7c7
Reading CSV peek with engine: 'c'
Source 'zte_4g' identified by required columns.
Identified source as: 'zte_4g'.
Starting atomic load & tracking for 'zte_4g_hourly_2025042200_2025042216.csv'...
Reading CSV chunks with engine: 'c'
Processing chunk 1 (Source rows: 10023)...
Loading chunk 1 (451035 staging records)...
Chunk 1 loaded in 13.55s. Total Staging Records so far: 451035
All chunks processed for 'zte_4g_hourly_2025042200_2025042216.csv'. Total records loaded: 451035 from 10023 source rows read.
Transaction committed for 'zte_4g_hourly_2025042200_2025042216.csv' with final status 'SUCCESS'.
Finished handling 'zte_4g_hourly_2025042200_2025042216.csv' in 14.23s. Final Status: SUCCESS. Records Loaded: 451035. Source Rows Read: 10023
Moved 'zte_4g_hourly_2025042200_2025042216.csv' to 'processed' directory.
========= Processing Summary =========
Total files attempted: 3
Successfully processed and moved: 3
Failed processing (moved to failed): 0
Files with move errors (left in source?): 0
============================================================
Data Loading Process Finished in 25.65s. Exit Code: 0
```

Figure 3.13: Excerpt from `Loader.py` execution log for processing ZTE Raw counters

Data in `Staging_RawCounter` (sample shown in Figure 3.15) is temporary. After the core ETL script (`ETL.py`) processes it into final fact tables, `Staging_RawCounter` is typically cleared.

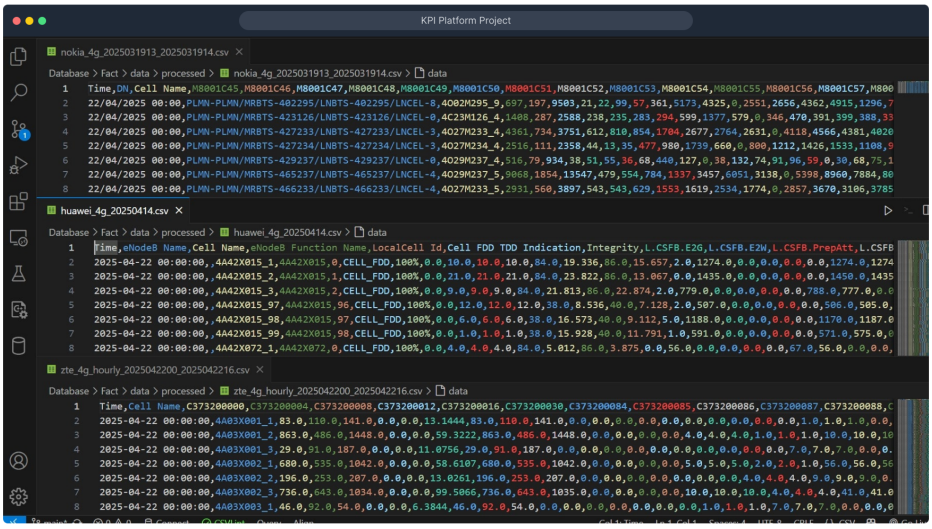
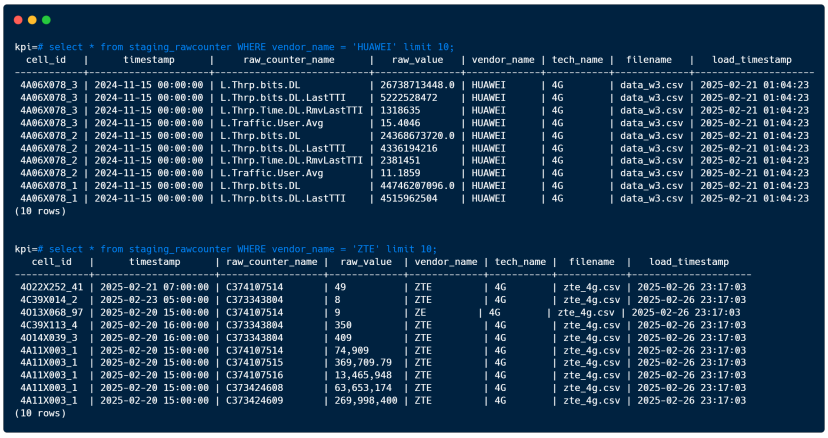


Figure 3.14: Processed and Standardized Vendor Data Files ready for Staging.



```
kpi=# select * from staging_rawcounter WHERE vendor_name = 'HUAWEI' limit 10;
  cell_id   |      timestamp      |      raw_counter_name      |   raw_value   | vendor_name | tech_name |  filename   |   load_timestamp
------------+---------------------+----------------------------+---------------+-------------+-----------+-------------+---------------------
 4A06X078_3 | 2024-11-15 00:00:00 | L.Thrp.bits.DL             | 26738713448.0 | HUAWEI      | 4G        | data_w3.csv | 2025-02-21 01:04:23
 4A06X078_3 | 2024-11-15 00:00:00 | L.Thrp.bits.DL.LastTTI     | 5222528472    | HUAWEI      | 4G        | data_w3.csv | 2025-02-21 01:04:23
 4A06X078_3 | 2024-11-15 00:00:00 | L.Thrp.Time.DL.RmvLastTTI  | 1318635       | HUAWEI      | 4G        | data_w3.csv | 2025-02-21 01:04:23
 4A06X078_3 | 2024-11-15 00:00:00 | L.Traffic.User.Avg         | 15.4046       | HUAWEI      | 4G        | data_w3.csv | 2025-02-21 01:04:23
 4A06X078_2 | 2024-11-15 00:00:00 | L.Thrp.bits.DL             | 24368673720.0 | HUAWEI      | 4G        | data_w3.csv | 2025-02-21 01:04:23
 4A06X078_2 | 2024-11-15 00:00:00 | L.Thrp.bits.DL.LastTTI     | 4336194216    | HUAWEI      | 4G        | data_w3.csv | 2025-02-21 01:04:23
 4A06X078_2 | 2024-11-15 00:00:00 | L.Thrp.Time.DL.RmvLastTTI  | 2381451       | HUAWEI      | 4G        | data_w3.csv | 2025-02-21 01:04:23
 4A06X078_2 | 2024-11-15 00:00:00 | L.Traffic.User.Avg         | 11.1859       | HUAWEI      | 4G        | data_w3.csv | 2025-02-21 01:04:23
 4A06X078_1 | 2024-11-15 00:00:00 | L.Thrp.bits.DL             | 44746207096.0 | HUAWEI      | 4G        | data_w3.csv | 2025-02-21 01:04:23
 4A06X078_1 | 2024-11-15 00:00:00 | L.Thrp.bits.DL.LastTTI     | 4515962504    | HUAWEI      | 4G        | data_w3.csv | 2025-02-21 01:04:23
(10 rows)


kpi=# select * from staging_rawcounter WHERE vendor_name = 'ZTE' limit 10;
  cell_id   |      timestamp      | raw_counter_name | raw_value | vendor_name | tech_name |  filename   |   load_timestamp
------------+---------------------+------------------+-----------+-------------+-----------+-------------+---------------------
 4022X252_41 | 2025-02-21 07:00:00 | C374107514       | 49        | ZTE         | 4G        | zte_4g.csv | 2025-02-26 23:17:03
 4C39X014_2  | 2025-02-23 05:00:00 | C373343804       | 8         | ZTE         | 4G        | zte_4g.csv | 2025-02-26 23:17:03
 4013X068_97 | 2025-02-20 15:00:00 | C374107514       | 9         | ZE          | 4G        | zte_4g.csv | 2025-02-26 23:17:03
 4C39X113_4  | 2025-02-20 16:00:00 | C373343804       | 350       | ZTE         | 4G        | zte_4g.csv | 2025-02-26 23:17:03
 4014X039_3  | 2025-02-20 16:00:00 | C373343804       | 409       | ZTE         | 4G        | zte_4g.csv | 2025-02-26 23:17:03
 4A11X003_1  | 2025-02-20 15:00:00 | C374107514       | 74,909    | ZTE         | 4G        | zte_4g.csv | 2025-02-26 23:17:03
 4A11X003_1  | 2025-02-20 15:00:00 | C374107515       | 369,709.79| ZTE         | 4G        | zte_4g.csv | 2025-02-26 23:17:03
 4A11X003_1  | 2025-02-20 15:00:00 | C374107516       | 13,465,948| ZTE         | 4G        | zte_4g.csv | 2025-02-26 23:17:03
 4A11X003_1  | 2025-02-20 15:00:00 | C373424608       | 63,653,174| ZTE         | 4G        | zte_4g.csv | 2025-02-26 23:17:03
 4A11X003_1  | 2025-02-20 15:00:00 | C373424609       | 269,998,400| ZTE        | 4G        | zte_4g.csv | 2025-02-26 23:17:03
(10 rows)
```

Figure 3.15: Raw Data in `Staging_RawCounter` (Long Format).

3.3.2 Transformation Logic: KPI Calculation, Data Cleansing

1. **Core ETL (`ETL.py`):**
   Once data is available in the `Staging_RawCounter` table, the core ETL (Extract, Transform, Load) logic is executed, primarily orchestrated by the Python script `ETL.py`. This script is responsible for pulling data from the staging area, performing crucial transformations and enrichments, and finally loading it into the DWH's primary hourly fact table, `FactRawCounter`. The key operations performed by `ETL.py` include:

   - **Data Extraction from Staging:** `ETL.py` queries the `Staging_RawCounter` table to retrieve batches of raw counter data that are ready for processing. After successful processing by `ETL.py`, the `Staging_RawCounter` table is typically cleared, as confirmed by database queries shown in Figure 3.16.

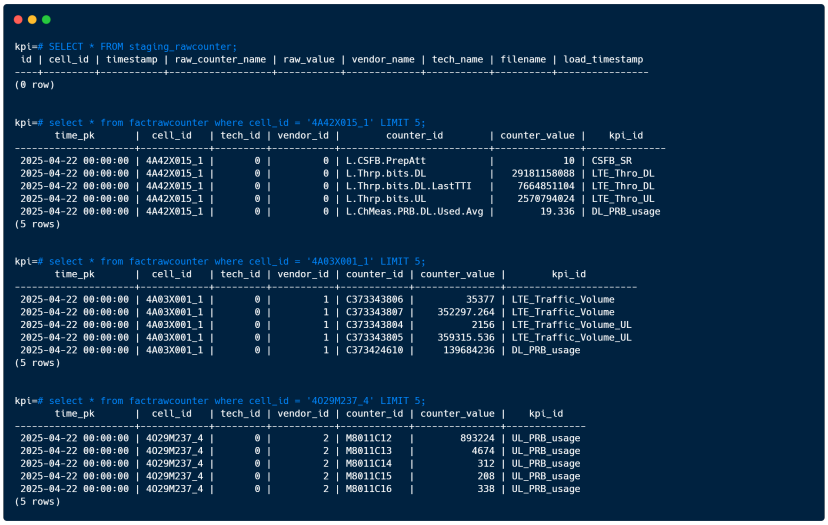


Figure 3.16: DB State Post-ETL : `Staging_RawCounter` Empty, FactRawCounter Populated.

After the ETL run `Staging_RawCounter` is empty (top query), while `FactRawCounter` contains processed data from various vendors (subsequent queries).

- **Data Validation and Cleansing:** This step involves more rigorous checks than those potentially applied at the staging load. Examples include converting raw counter values to numeric types, validating timestamps, checking for missing essential fields, and implementing strategies for handling such records (e.g., logging and skipping).
- **Enrichment with Dimensional Keys (Lookups):** This critical transformation step resolves raw, often string-based identifiers from the staging data into the surrogate primary keys used in the DWH's dimension tables. This includes:
  - **Cell ID Resolution:** Mapping raw cell identifiers to `cell_id` from `DimCell`.
  - **Time Dimension Key:** Converting raw timestamps to `time_pk` from `DimTime`.
  - **Vendor and Technology ID Resolution:** Mapping names to `vendor_id` and `tech_id` from respective dimension tables.
  - **KPI ID Resolution:** Determining `kpi_id` from `CounterDefinition` based on `counter_id`, `vendor_id`, and `tech_id`.

  The outcome is a dataset where each raw counter record is linked to the precise dimensional context needed for structured analysis and ML feature engineering.
- **Structural Transformation:** `ETL.py` ensures data conforms to the `FactRawCounter` schema, selecting and preparing columns for insertion.
- **Loading into `FactRawCounter`:** The transformed and enriched data is then loaded into the `FactRawCounter` table in batches. Figure 3.16 also illustrates sample data successfully populated into `FactRawCounter` for different vendors. Efficient insertion methods are used, often including an `ON CONFLICT` clause to handle updates or prevent duplicates.

The successful execution of these tasks by `ETL.py` populates `FactRawCounter` with clean, validated, and contextually rich data, ready for aggregation and KPI calculation. The operational logs from the Airflow task executing `ETL.py` (an example of which is shown in Figure 3.18) provide crucial insights into this process. For instance, these logs often

include a summary detailing the number of records processed (e.g., 727,280 in one observed run), rows affected in `FactRawCounter`, new cells inserted into `DimCell`, records skipped along with reasons (e.g., 44 due to invalid counter names), the overall success rate, and confirmation of the `Staging_RawCounter` cleanup (truncation) post-processing.

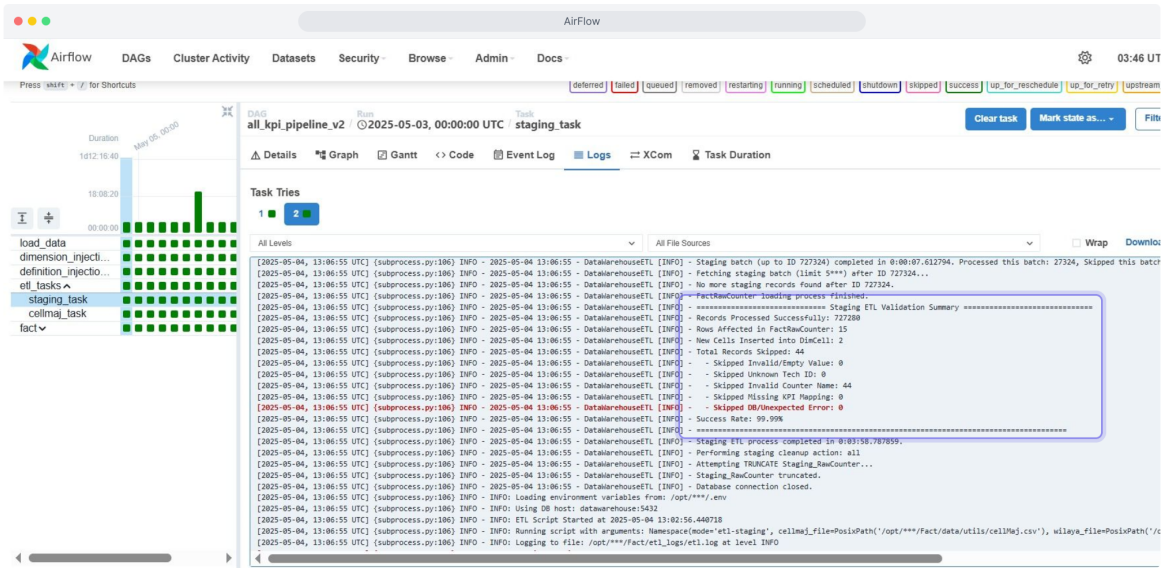


Figure 3.17: `ETL.py` execution log, as retrieved from the Airflow UI.

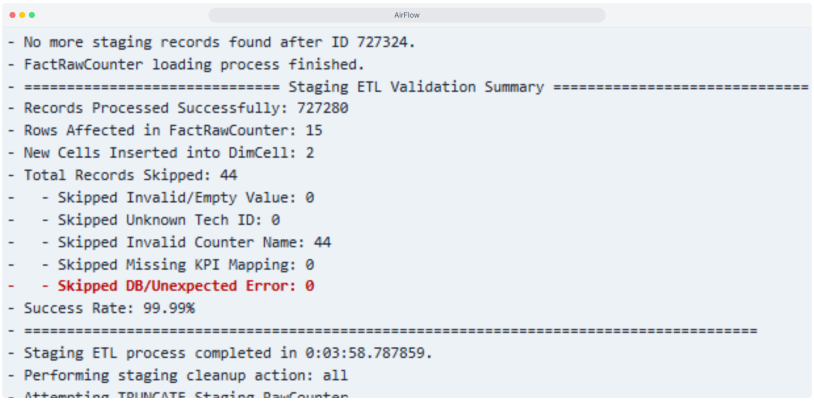


Figure 3.18: Summary from `ETL.py` execution log

the execution log is detailing records processed (727,280), rows affected in `FactRawCounter` (15 in this example run), new cells inserted (2), records skipped (44 due to invalid counter name), success rate (99.99%), and staging cleanup confirmation.

2. **KPI Calculation Engine:**

   A key strength of the DWH architecture is its dynamic KPI calculation engine, implemented directly in PostgreSQL using PL/pgSQL functions such as `evaluate_cell_kpi_formula` and `evaluate_agg_kpi_formula`. This design abstracts the complex, often vendor-specific KPI logic away from the Python-based ETL code, significantly improving flexibility, maintainability, and scalability. The process unfolds in five main stages:

(a) **Formula Definition and Storage:**
KPI formulas, which vary across vendors and technologies, are initially sourced from documentation or internal references and organized into a structured mapping file (`map.csv`; see Figure 3.19). These definitions are then standardized and stored in the `KPIFormula` table during the ETL "definition injection" step, managed by scripts like `inject_formula.py`. Standardization involves formatting the formulas into SQL-compliant expressions. Figure 3.19 shows both the intermediate `kpi_formula.csv` (post-processing of `map.csv`), while Figure 3.20 illustrates the final result within the database as visualized in pgAdmin.

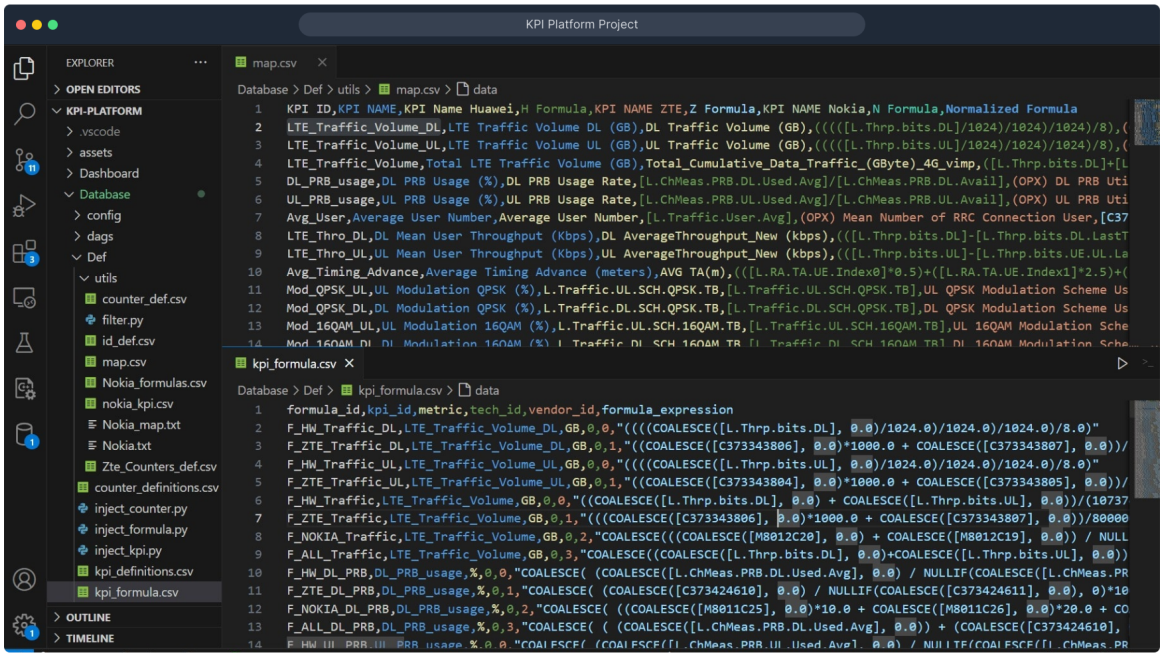


Figure 3.19: KPI Formula Transformation: `kpi_formula.csv` from `map.csv`.

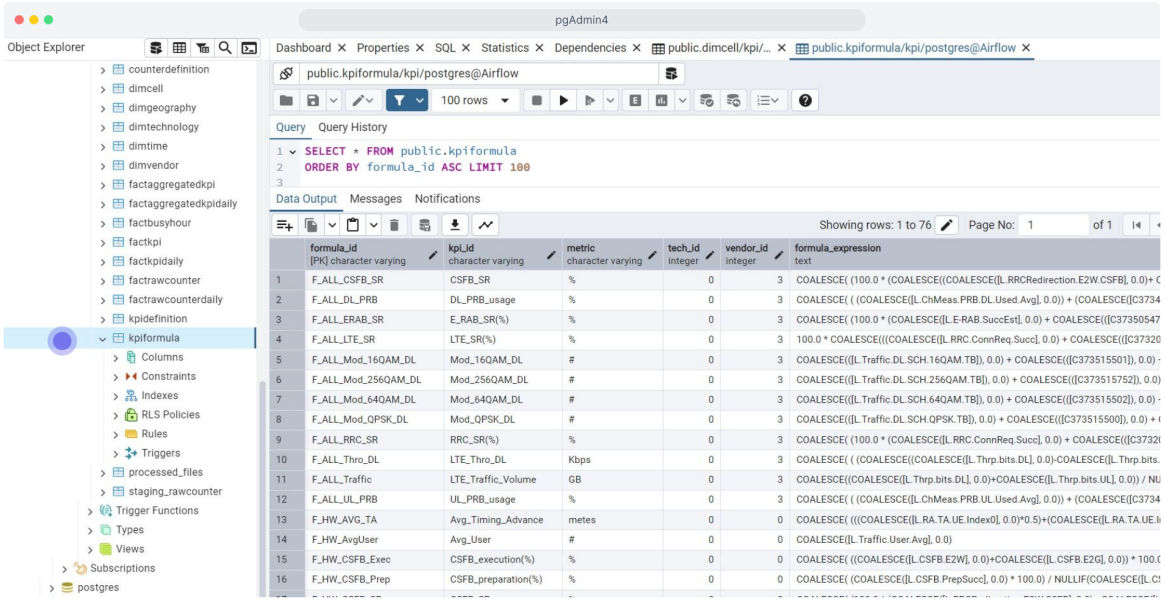


Figure 3.20: The final `KPIFormula` table in pgAdmin with SQL-ready expressions.

(b) **Dynamic Formula Retrieval:**
When KPI values are computed, the PL/pgSQL functions query the `KPIFormula` table to retrieve the appropriate formula string, based on the current `kpi_id`, `vendor_id`, and `tech_id`.

(c) **Counter Value Preparation:**
Required raw counter values are fetched from relevant fact tables (e.g., `FactRaw Counter`) and passed to the evaluation function in a structured format, typically as a `JSONB` object. This object maps standardized counter names to their numeric values for a given entity.

(d) **Formula Parsing and Execution:**
The PL/pgSQL function substitutes the placeholders in the formula with the actual counter values from the `JSONB` object. It then evaluates the resulting expression using dynamic SQL or an internal expression parser.

(e) **Error Handling and Result Output:**
To ensure robustness, the functions include built-in error handling mechanisms. For instance, they can detect and manage issues such as division by zero or missing counters. Instead of terminating the entire batch process, they return predefined error codes (e.g., `-9998.0` for division by zero, `-9997.0` for missing counters). On success, the function returns the calculated KPI value.

This metadata-driven approach enables administrators to introduce or modify KPI logic by simply updating the `KPIFormula` table via a revised `kpi_formula.csv` and re-executing the injection script—without altering the core PL/pgSQL functions or Python ETL code. As a result, the system remains highly adaptable to changes in business rules or network metrics.

3.3.3 Loading Data into the DWH (Dimension and Fact Tables)

- **Dimension and Definition Tables:**
These tables are populated during the initial setup and updated as needed using dedicated Python scripts (e.g., `inject_cell.py`, `inject_kpi.py`, `inject_formula.py`) located in the `Dim/` and `Def/` directories. The scripts load data from structured CSV files (e.g., `kpi_definitions.csv`) into the corresponding tables. These injection tasks are integrated into the Airflow DAG to ensure that all required metadata is available before processing begins.

- **Fact Tables:**

  - `FactRawCounter`: populated by `ETL.py` after processing data from `Staging _RawCounter`.
  - Higher-level Fact Tables (`FactKPI`, `FactAggregatedKPI`, etc.): filled by SQL scripts (e.g., `FactKPI.sql`, `FactAgg.sql`) orchestrated by Airflow. These scripts read from upstream fact tables, apply calculations or aggregations, and insert or update results in the target tables. They leverage `ON CONFLICT DO UPDATE` and timestamp-based logic to support idempotent and incremental loading.

3.3.4 Automated Aggregation Scripts (Daily, Geo, Busy Hour)

A suite of SQL scripts, orchestrated as tasks within an Apache Airflow Directed Acyclic Graph (DAG), automates the aggregation of raw counters and the calculation of KPIs across multiple

levels of granularity. The main DAG, named `all_kpi_pipeline_v2`, coordinates the flow from initial data ingestion to final KPI aggregation, as shown in Figure 3.21. This setup ensures dependencies are respected and that data is processed in a coherent and scalable manner. The key aggregation and computation steps include:

- **All-Network Raw Counter Aggregation (`FactAllNet.sql`):** The `fact_all_net` task merges raw counters from multiple vendors for a given technology and geographic area, assigning them a synthetic 'ALL' vendor ID. This enables unified, vendor-agnostic views within the `FactRawCounter` table.
- **Daily Raw Counter Aggregation (`FactDaily.sql`):** The `fact_daily` task aggregates hourly records from `FactRawCounter` to compute daily totals, which are stored in `FactRawCounterDaily`.
- **Cell-Level KPI Calculation (`FactKPI.sql`, `FactKPIDaily.sql`):** The `fact_kpi` and `fact_kpi_daily` tasks compute KPIs at the cell level for hourly and daily granularities, storing the results in `FactKPI` and `FactKPIDaily`. Figure 3.22 illustrates a moment during pipeline execution with `fact_kpi` actively running.
- **Aggregated KPI Calculation (`FactAgg.sql`, `FactAggDaily.sql`):** The `fact_agg` and `fact_agg_daily` tasks compute aggregated KPIs at geographic levels (Commune, Wilaya, AllNet) for hourly and daily intervals. Results are stored in `FactAggregatedKPI` and `FactAggregatedKPIDaily`, leveraging the `mv_cell_geo` materialized view for spatial mapping.
- **Busy Hour Detection (`FactBusy.sql`):** The `fact_busy` task identifies, for each day, the hour during which a specific traffic KPI reaches its peak at various aggregation levels (Cell, Commune, Wilaya, AllNet). The output is recorded in `FactBusyHour`.

These scripts are optimized for incremental execution, processing only new time intervals based on the latest data present in their target tables. A successful DAG run indicates that the entire data pipeline has completed end-to-end processing and that all KPI tables are up to date.

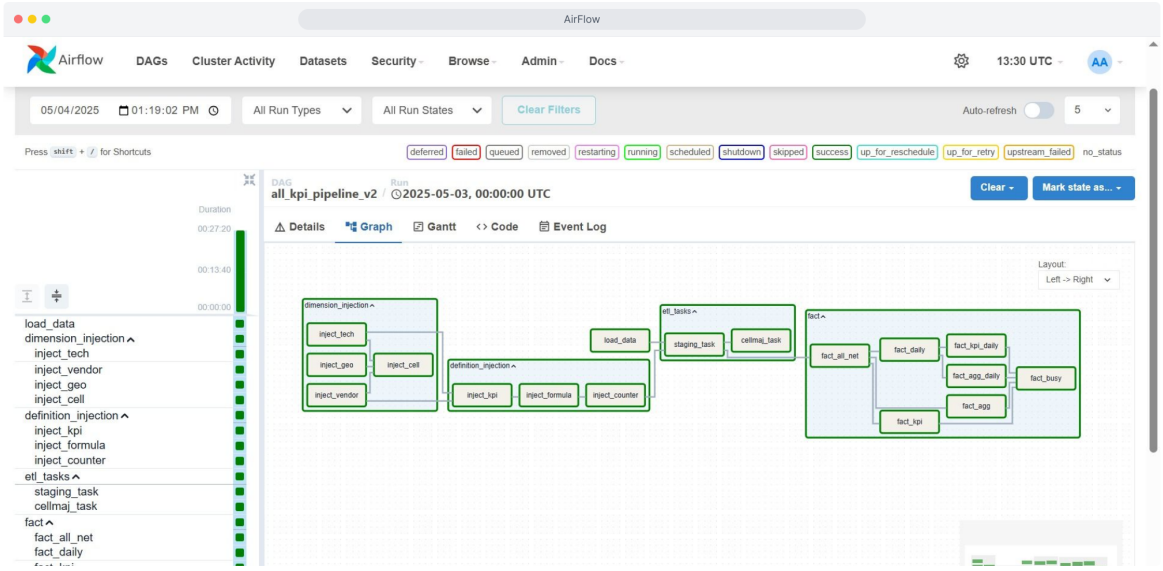


Figure 3.21: Graph view of the Apache Airflow DAG

The graph view of the DAG `all_kpi_pipeline_v2`) showing the end-to-end workflow, including definition injection, ETL stages, and automated aggregation tasks within the 'fact' group.

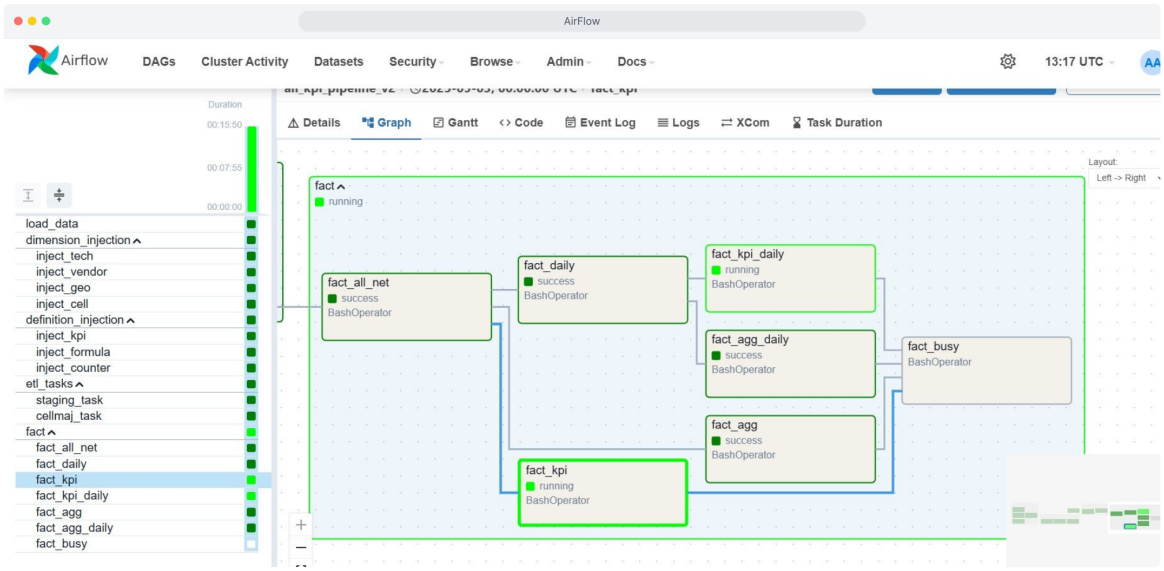


Figure 3.22: Execution snapshot of the DAG's 'fact' task group.

The task `fact_kpi` is currently in progress, as indicated by its green highlight in the workflow interface. This means it has successfully met all its dependencies and is now being executed by the scheduler. Prior to this, upstream tasks such as `fact_all_net` and `fact_daily` have already been completed successfully—they are marked with the status `'success'`, confirming that their outputs are available and valid. Meanwhile, downstream tasks like `fact_agg_daily` and `fact_busy` are in a queued state. This indicates that they are waiting for `fact_kpi` to finish, since they depend on its output to proceed. Once `fact_kpi` completes successfully, these downstream tasks will be unblocked and picked up by the scheduler for execution in accordance with the defined DAG dependencies.

3.3.5 Data Archiving and Maintenance Procedures

- **Data Archiving:**
  Although a fully automated archival process is a future enhancement, the system is designed to support the migration of historical data from active fact tables to archive tables (e.g., an archive version of the KPI fact table) to preserve query performance. This can be achieved via partitioning strategies or scheduled ETL jobs based on a defined retention policy (e.g., data older than one year).

- **Maintenance Procedures:**
  - Running table analysis after major data updates to refresh optimizer statistics.
  - Refreshing materialized views as part of scheduled maintenance routines.
  - Leveraging PostgreSQL's autovacuum for regular cleanup, with manual VACUUM when necessary.
  - Monitoring disk usage and database logs, with optional utility scripts providing reusable templates for these tasks.

3.4 Data Quality Assurance within the Pipeline

Data quality is addressed at several points in the pipeline, although a comprehensive, dedicated data quality framework remains a potential area for future enhancement.

- **File Handling in `Loader.py`:** Files are moved to either a `processed` or `failed` directory based on the success of ingestion into the staging table, enabling traceability and isolation of problematic inputs. Checksums are also used to prevent reprocessing of already loaded files.
- **Type Conversion and Validation:** Both `Loader.py` and `ETL.py` handle basic data type conversions (e.g., string to numeric, string to timestamp). Additionally, `ETL.py` verifies foreign key relationships against dimension tables to ensure referential integrity.
- **Logging:** Extensive logging is implemented in both `Loader.py` and `ETL.py`, capturing key events, exceptions, and validation outcomes. These logs are critical for diagnosing data-related issues and auditing pipeline behavior.

Future work could involve integrating a dedicated data quality tool (e.g., Great Expectations) or implementing more sophisticated statistical checks within the Airflow DAG to proactively identify anomalies or inconsistencies in both raw and processed data.

Conclusion

This chapter demonstrated the end-to-end process of collecting, processing, and reliably storing LTE performance data. The result is a robust and structured data subsystem that ensures data quality, supports visualization, and provides clean input for downstream AI models. In the following chapters, we will focus on the design, development, and evaluation of AI-based forecasting models built upon this data foundation.

Chapter 4

AI-Based KPI Forecasting

4.1 Introduction to AI-Based Forecasting for LTE Network Optimization

The operational efficiency and user satisfaction within Long-Term Evolution (LTE) networks are inextricably linked to the network's ability to consistently deliver high-quality service. As mobile data consumption escalates and user expectations for seamless connectivity intensify, traditional reactive network management approaches become increasingly inadequate [42]. Proactive strategies, underpinned by accurate forecasting of Key Performance Indicators (KPIs), are therefore essential. The capacity to anticipate future network states—such as traffic load, resource utilization, and achievable user throughput—empowers network operators to optimize resource allocation, preemptively address potential congestion, schedule maintenance effectively, and ultimately enhance the Quality of Service (QoS) and Quality of Experience (QoE) for their subscribers [15, 43].

This chapter presents a detailed account of the methodology, development lifecycle, and optimization techniques employed in creating AI-based forecasting models for critical hourly LTE KPIs. The project specifically targets the prediction of Downlink User Throughput (`LTE_Thro_DL`), Downlink Traffic Volume (`LTE_Traffic_Volume_DL`), and Downlink PRB Usage (`DL_PRB_usage`). These KPIs were selected due to their direct impact on user experience and network resource management. The study focuses on data from a representative, active urban cell (`4O13X018_1`) within Djezzy's commercial LTE network, ensuring the real-world applicability of the findings.

The core modeling approaches investigated are Facebook's Prophet, a statistical time series model renowned for its robustness and which served as an advanced baseline in our study, and Long Short-Term Memory (LSTM) networks, a sophisticated type of Recurrent Neural Network (RNN) designed to capture complex temporal dependencies and long-range patterns often present in telecommunications data [40, 41]. The development process adopted was explicitly iterative and data-driven. It commenced with an in-depth Exploratory Data Analysis (EDA) to thoroughly understand the characteristics and nuances of the KPI data. This was followed by the establishment of baseline forecasting models to provide a performance benchmark. A significant portion of the effort was dedicated to the meticulous development and refinement of the LSTM models, which included strategic feature engineering, a systematic and automated hyperparameter optimization phase using the Optuna framework [44], and rigorous debugging of the entire data preprocessing and evaluation pipeline to ensure methodological soundness.

This chapter will elucidate the technical intricacies of each development stage, from initial data handling to the final optimized model configurations. The objective is to provide a transparent and reproducible account of the engineering efforts involved in building these predictive

AI modules, which form a crucial component of the broader KPI analytics platform. The performance evaluation of these final models against the established baselines on a dedicated hold-out test set is subsequently presented in Chapter 5.

4.2 Theoretical Foundations of Forecasting Models

This section provides a concise theoretical foundation for the two principal forecasting models employed in this study for LTE KPI prediction: Long Short-Term Memory (LSTM) networks and the Prophet model. A thorough understanding of these models is essential before delving into the implementation and evaluation procedures detailed in subsequent chapters.

4.2.1 Long Short-Term Memory (LSTM) Networks

Long Short-Term Memory (LSTM) networks, introduced by Hochreiter and Schmidhuber [40], constitute a specialized architecture within the broader family of Recurrent Neural Networks (RNNs). LSTMs were developed to mitigate the vanishing gradient problem that hampers the learning of long-range temporal dependencies in standard RNNs. This capability renders LSTMs highly effective for time series forecasting tasks, including the prediction of LTE network KPIs, which often exhibit long-term temporal correlations.

In contrast to traditional RNNs, which update a single hidden state vector h_t through each time step, LSTMs incorporate a more complex internal architecture that includes a memory cell C_t and a series of gating mechanisms. This design enables the model to preserve relevant information over extended sequences while discarding less useful signals.

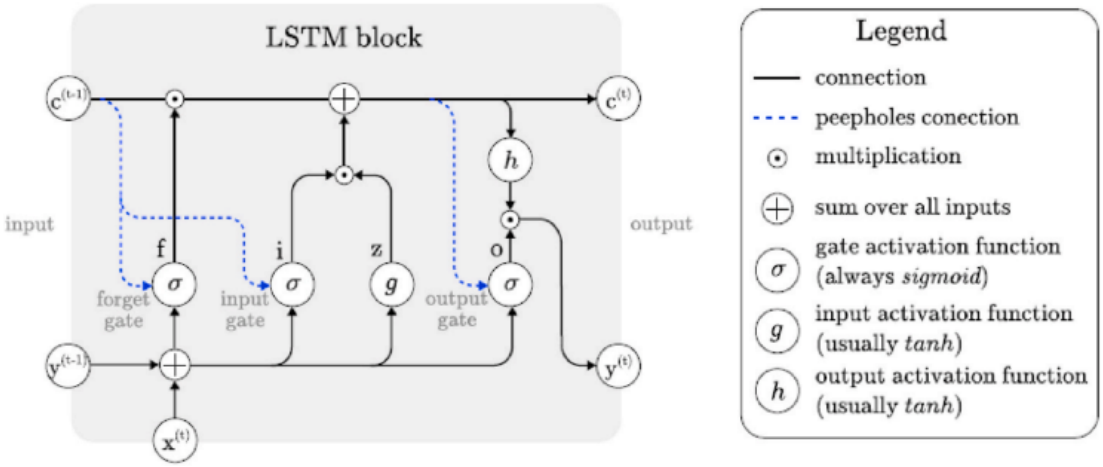


Figure 4.1: Overall Long Short-Term Memory (LSTM) Cell Architecture.[45].

The flow of information in an LSTM cell is governed by the interaction of three principal gates, each composed of weight matrices W and bias vectors b. At each time step t, the LSTM cell receives the current input $x^{(t)} \in \mathbb{R}^{n_x}$, the previous hidden state $y^{(t-1)} \in \mathbb{R}^{n_h}$, and the previous cell state $c^{(t-1)} \in \mathbb{R}^{n_h}$.

Each gate (forget, input, output) uses the peephole connections to incorporate information from $c^{(t-1)}$ or $c^{(t)}$, as shown by the blue dashed lines in the diagram.

Let's define:

- $W_f, W_i, W_o, W_z \in \mathbb{R}^{n_h \times (n_x + n_h + n_c)}$ — weight matrices for the forget, input, output gates, and candidate state, respectively. $n_c = n_h$ for peephole connections.
- $b_f, b_i, b_o, b_z \in \mathbb{R}^{n_h}$ — bias vectors.
- **Forget Gate** — Decides what information to discard from the previous memory cell:

$$f^{(t)} = \sigma\left(W_f \cdot [y^{(t-1)}, x^{(t)}, c^{(t-1)}] + b_f\right) \tag{4.1}$$

- **Input Gate and Candidate State** — Determine what new information to store:

$$i^{(t)} = \sigma\left(W_i \cdot [y^{(t-1)}, x^{(t)}, c^{(t-1)}] + b_i\right) \tag{4.2}$$

$$z^{(t)} = \tanh\left(W_z \cdot [y^{(t-1)}, x^{(t)}] + b_z\right) \tag{4.3}$$

  The cell state is updated as a combination of the old state and the new candidate information:

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot z^{(t)} \tag{4.4}$$

- **Output Gate and Final Output** — Decide what part of the cell state to expose to the output:

$$o^{(t)} = \sigma\left(W_o \cdot [y^{(t-1)}, x^{(t)}, c^{(t)}] + b_o\right) \tag{4.5}$$

$$y^{(t)} = o^{(t)} \odot \tanh(c^{(t)}) \tag{4.6}$$

Here, σ denotes the sigmoid activation function, and tanh is the hyperbolic tangent activation. The operator $\odot$ denotes element-wise multiplication. The inclusion of $c^{(t-1)}$ and $c^{(t)}$ in the gate computations represents peephole connections, allowing the gates to observe the cell state directly.

This architecture enables the network to capture both short-term and long-term dependencies within the data. In practical applications, LSTM models may consist of multiple stacked layers followed by fully connected layers to enhance their representational power. LSTMs have shown robust performance across various time series forecasting domains, including network performance monitoring, financial data analysis, and speech recognition [46][40].

4.2.2 The Prophet Model

Prophet is an open-source forecasting framework developed by Facebook Research [41], designed for modeling time series data characterized by strong seasonality, trend shifts, and irregular events. Prophet is particularly suitable for practical applications due to its interpretability, robustness to missing data, and ability to handle outliers.

The model adopts an additive approach to time series decomposition:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t \tag{4.7}$$

where:

- $g(t)$ denotes the trend function modeling non-periodic structural changes,
- $s(t)$ captures periodic seasonal effects,

- $h(t)$ accounts for holiday or special event effects,
- ϵ_t represents the residual noise, assumed to follow a Gaussian distribution.

Trend Modeling: Prophet offers two trend models:

- *Piecewise Linear Trend:* Suitable for data with abrupt changes in trend.
- *Logistic Growth:* Appropriate when growth is bounded by a carrying capacity:

$$g(t) = \frac{C(t)}{1 + \exp(-k(t)(t - m(t)))} \tag{4.8}$$

The model supports automatic or user-specified changepoints where the trend's rate of change is modified, accommodating shifts caused by external interventions such as policy changes or network upgrades.

Seasonality Modeling: Seasonality is modeled using a Fourier series:

$$s(t) = \sum_{n=1}^{N} \left(a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right) \right) \tag{4.9}$$

Here, P denotes the seasonal period (e.g., weekly or yearly), and N controls the number of Fourier terms used, hence the flexibility of the seasonality curve.

Holiday Effects: Prophet allows the integration of user-defined holidays or special events. These are treated as binary indicators that add fixed effects to specific time windows, enhancing the model's ability to capture short-term anomalies.

Model Estimation and Forecasting: Model fitting is performed using the Stan probabilistic programming framework, enabling Bayesian inference and the generation of uncertainty intervals. The modular design and intuitive parameters make Prophet particularly appealing for industry applications requiring fast and interpretable forecasting solutions.

In the context of LTE KPI forecasting, Prophet offers a viable alternative or complement to neural models such as LSTMs, particularly when the dataset exhibits interpretable trends, periodicities, and exogenous shocks (e.g., maintenance periods, policy changes, or societal events such as the COVID-19 pandemic [47]).

4.3 Data Foundation and Exploratory Analysis for Forecasting

The efficacy of any AI forecasting model is fundamentally dependent on the quality, characteristics, and comprehensive understanding of the underlying data. This section details the data sourcing strategy, the preprocessing steps undertaken to prepare the data for modeling, and the key insights derived from an extensive Exploratory Data Analysis (EDA) of the training dataset. These insights were pivotal in shaping feature engineering strategies and informing model selection.

4.3.1 Data Sourcing, Scope, and Initial Preprocessing

- **Data Source and Provenance:** The primary dataset for this study comprises hourly aggregated KPI values for cell `4O13X018_1`, extracted from our production Data Warehouse (DWH) of Djezzy's LTE network. The DWH architecture and ETL processes, responsible for collecting, normalizing, and storing this data from multi-vendor network elements, are detailed in Chapter 3. The use of real-world operational data ensures the practical relevance and applicability of the developed forecasting models.
- **Timeframe for Training and Evaluation:**
  - *Training and Optuna Validation Period:* A continuous three-month period of hourly data, specifically from January 29, 2025, 23:00 hours, to April 29, 2025, 23:00 hours, was allocated for training the forecasting models and for performing hyperparameter optimization with Optuna (which uses an internal chronological validation split). This duration (approximately 2160 hourly data points per KPI) was deemed sufficient to capture multiple instances of daily and weekly seasonality, essential for time series modeling [48].
  - *Hold-Out Test Period:* To ensure an unbiased evaluation of the final models generalization performance, a subsequent, entirely unseen one-month period, from March 1, 2025, 00:00 hours, to March 30, 2025, 23:00 hours, was strictly reserved as a hold-out test set. This data was not exposed to the models during any phase of training or hyperparameter tuning.
- **Target KPIs and Granularity:** The forecasting efforts focused on the following hourly KPIs:
  1. `LTE_Thro_DL`: Downlink User Throughput (Mbps) – reflects the data rate experienced by users.
  2. `LTE_Traffic_Volume_DL`: Downlink Traffic Volume (GB) – indicates the total data load on the cell.
  3. `DL_PRB_usage`: Downlink Physical Resource Block (PRB) Utilization (%) – measures radio resource consumption.
- **Initial Data Cleaning :**
  - Before any feature engineering or EDA, the raw KPI series extracted from the DWH underwent an essential cleaning process implemented in a python script.
  - *Missing Value Imputation:* Sporadic missing values, potentially due to transient data collection issues, were handled using a combination of forward-fill (`ffill`) followed by

backward-fill (`bfill`) imputation. This method was chosen to preserve the temporal structure of the data while ensuring a continuous series for model training.

– *Outlier Treatment:* To mitigate the undue influence of extreme outliers (which could be measurement errors or highly unusual, non-representative network events) on model training, particularly for LSTMs, values below the 1st percentile and above the 99th percentile of each KPI's training distribution were capped at these respective percentile values. This robust clipping approach helps stabilize the data without aggressively removing potentially informative extreme behaviors[49].

4.3.2 Exploratory Data Analysis (EDA) of Training Data

Initial EDA involved individual KPI analysis to understand their unique characteristics, laying groundwork for multivariate modeling by revealing temporal dynamics, statistical properties, and cyclical patterns. An overall statistical summary was first computed from the N=697 hourly training samples, which include primary KPIs and engineered temporal/contextual features.

Table 4.1 details descriptive statistics for numerical KPIs (`LTE_Thro_DL`, `LTE_Traffic_Volume_DL`, `DL_PRB_usage`) and other numerical features, offering insights into their central tendency, spread, and range. Concurrently, Table 4.2 outlines the distribution of key engineered boolean features that capture contextual information like peak hours, holidays, and weekends, crucial for understanding varied operational conditions.

Table 4.1: Descriptive Statistics of Key KPIs and Numerical Features (Training Data, N=697)

Feature	Mean	Std Dev	Min	25% (Q1)	50% (Median)	75% (Q3)	Max
`LTE_Thro_DL` (Mbps)	29.53	15.77	11.83	18.95	24.98	34.86	93.60
`LTE_Traffic_Volume_DL` (GB)	1.45	1.34	0.02	0.51	0.97	1.93	5.78
`DL_PRB_usage` (%)	13.11	11.12	1.67	5.00	9.01	17.54	46.99
Other Numerical Features:							
`day_of_week`	3.00	1.97	0.00	1.00	3.00	5.00	6.00
`hour_of_week_sin`	0.00	0.70	-1.00	-0.68	0.00	0.68	1.00
`hour_of_week_cos`	-0.03	0.72	-1.00	-0.76	-0.07	0.68	1.00
`days_until_next_holiday`	8.43	5.27	0.00	4.00	8.00	12.00	20.00
`days_since_last_holiday`	20.51	11.55	1.00	8.00	23.00	30.00	37.00

Table 4.2: Distribution of Boolean Features (Training Data, N=697)

Feature	Distribution (Value: Count)
`is_business_hours`	False: 508, True: 189
`is_evening_peak`	False: 581, True: 116
`is_off_peak`	True: 392, False: 305
`is_holiday`	False: 673, True: 24
`is_weekend`	False: 505, True: 192

The event-related features summarized in Tables 4.1 and 4.2 (e.g., `is_holiday`, `next_holiday`) were derived by integrating time-series data with a dedicated `DimEvents` table from the Data Warehouse.

This `DimEvents` dimension was populated via SQL scripts with significant Algerian national, religious, cultural, and school holiday events for 2024-2025. It stores event names, types, and timings, enabling nuanced feature engineering. Such features allow LSTM and Prophet models to learn patterns associated with these periods, improving forecast accuracy.

This global statistical profile, enriched by event context, sets the stage for a more granular examination of individual KPIs.

Univariate Analysis of Target KPIs

Each target KPI—`LTE_Thro_DL`, `LTE_Traffic_Volume_DL`, and `DL_PRB_usage`—underwent univariate EDA to reveal trends, seasonality, distribution, and temporal dependencies. Time series plots, histograms, box plots, seasonal decomposition, and ACF/PACF plots guided preprocessing, feature engineering, and model selection.

`LTE_Thro_DL` (Downlink Throughput):

- *Time Series Visualization:*

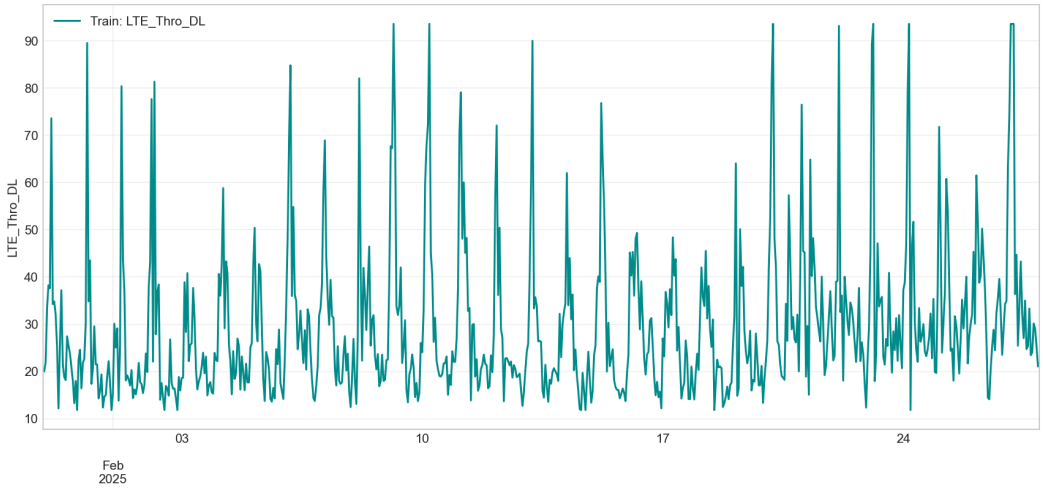


Figure 4.2: Hourly Time Series of `LTE_Thro_DL` for Cell `4O13X018_1`

Figure 4.2 presents the hourly time series plot of `LTE_Thro_DL`. The series is characterized by significant volatility and distinct daily cyclical patterns. Peaks are generally observed during evening hours (e.g., 19:00-23:00 local time), corresponding to periods of high user activity, while troughs are evident during early morning off-peak hours (e.g., 02:00-06:00). No dominant long-term linear trend is visually apparent over this three-month window, suggesting a degree of overall stationarity around a fluctuating mean level, a common characteristic for mature cell throughput.

- *Distribution Analysis:*

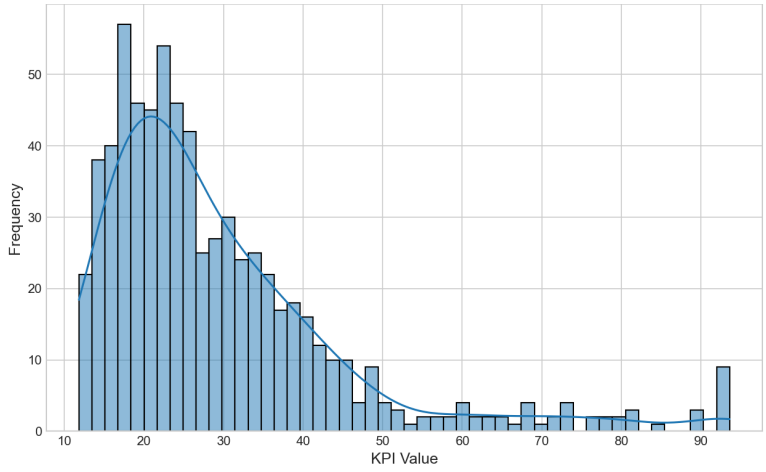


Figure 4.3: Distribution of Hourly `LTE_Thro_DL`

The histogram and Kernel Density Estimate (KDE) in Figure 4.3 reveal that the distribution of hourly throughput values is positively skewed (right-skewed). While the majority of observations are concentrated in the 15-40 Mbps range, the presence of a long tail extending towards higher values (up to the clipped maximum of ≈93.6 Mbps) is notable. The mean (29.53 Mbps) being greater than the median (24.98 Mbps), as detailed in descriptive statistics (Table 4.1), quantitatively confirms this skewness. This characteristic strongly suggested the utility of a log transformation (specifically `log1p`) for this KPI when training LSTM models, aiming to stabilize variance and normalize the distribution.

- *Outlier Visualization (Box Plot):*

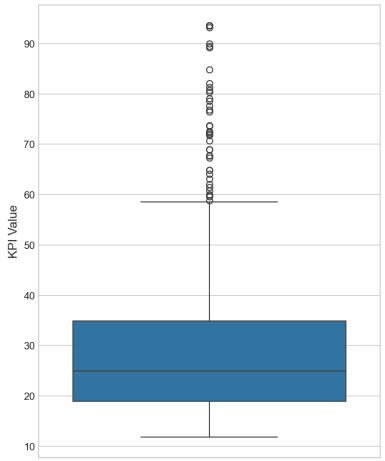


Figure 4.4: Box Plot of Hourly `LTE_Thro_DL`

Figure 4.4, the box plot for `LTE_Thro_DL`, further emphasizes the data spread and skewness. It visually highlights data points beyond the upper whisker, indicating periods of exceptionally high throughput that, even after the 1st/99th percentile clipping, represent the upper range of performance for this cell.

- *Seasonality and Autocorrelation Analysis:*

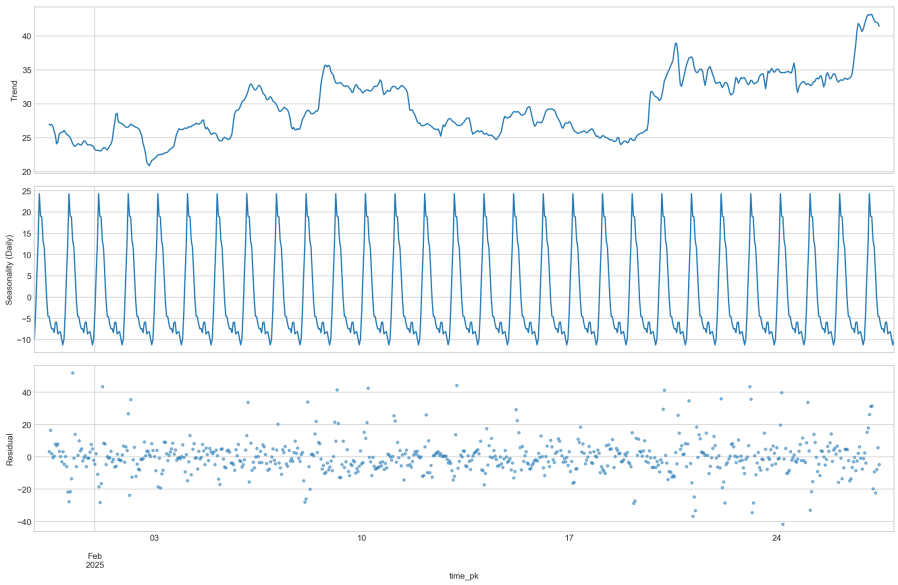


Figure 4.5: Additive Seasonal Decomposition of `LTE_Thro_DL`

Additive seasonal decomposition (Figure 4.5) effectively isolates a strong daily seasonality (period=24 hours), clearly visible in the seasonal component. The trend component exhibits medium-term fluctuations rather than a monotonic linear progression, reinforcing the observation of overall stationarity. The residuals appear largely random, though some periods show slightly higher variance, suggesting that while daily seasonality is dominant, other minor cyclical influences or noise might exist.

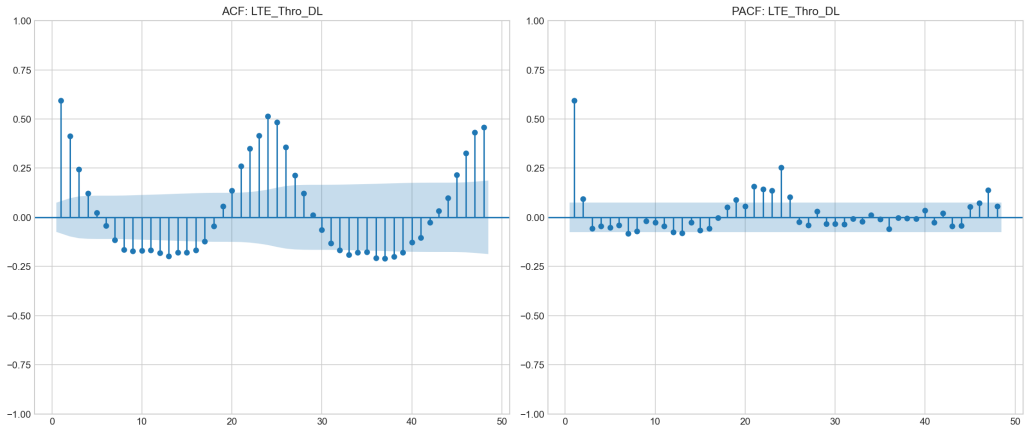


Figure 4.6: ACF and PACF Plots for `LTE_Thro_DL`

The Autocorrelation Function (ACF) plot (Figure 4.6, Left) shows a characteristic slow decay for short lags and very significant spikes at lags corresponding to daily cycles (24, 48, 72 hours, etc.), robustly confirming the presence of strong daily seasonality. Significant positive autocorrelation is also evident at immediate lags (1-6 hours), indicating short-term persistence or "momentum" in the throughput values. The Partial Autocorrelation Function (PACF) plot (Figure 4.6, Right) typically shows significant spikes at the initial lags (e.g., 1, 2, 3) that cut off, and another significant spike around lag 24. This pattern suggests that an autoregressive (AR) model component of low order, combined with a seasonal AR component, would be appropriate if using ARIMA-family models, and for LSTMs, it directly informed the selection of these specific past values as input features (lags).

- *Stationarity Assessment:*

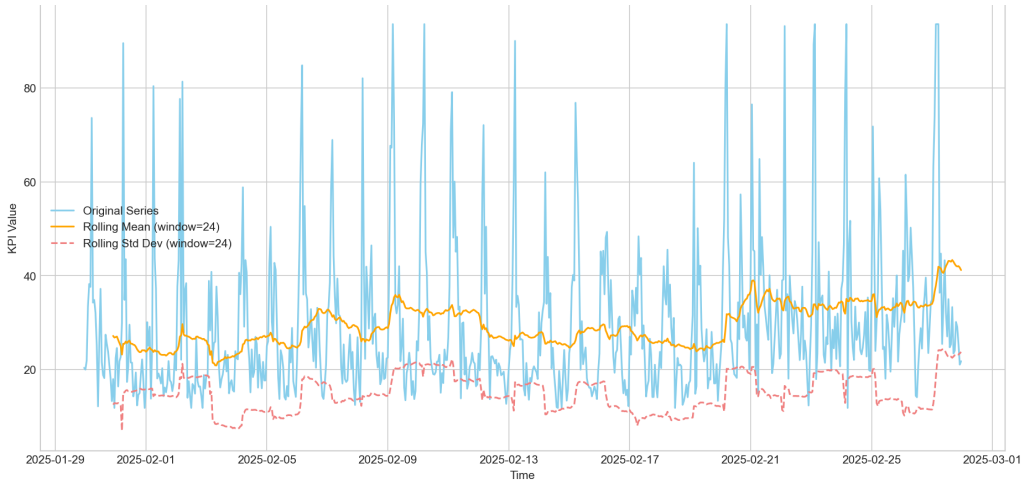


Figure 4.7: Rolling Mean Std Dev: `LTE_Thro_DL` (Window=24h).

The Augmented Dickey-Fuller (ADF) test conducted on the training series for `LTE_Thro_DL` yielded a test statistic of -11.47 and a p-value <<< 0.01 (effectively 0.000000). This extremely low p-value allows for strong rejection of the null hypothesis (that a unit root is present), confirming that the `LTE_Thro_DL` series is stationary over the training period. This is visually corroborated by Figure 4.7, where the 24-hour rolling mean and rolling standard deviation remain relatively stable over time, without exhibiting clear trends or drastically changing variance. Stationarity simplifies modeling as it means the statistical properties of the series do not change over time, making differencing unnecessary before applying many forecasting techniques.

- *Cyclical Hourly and Weekly Profiles:*

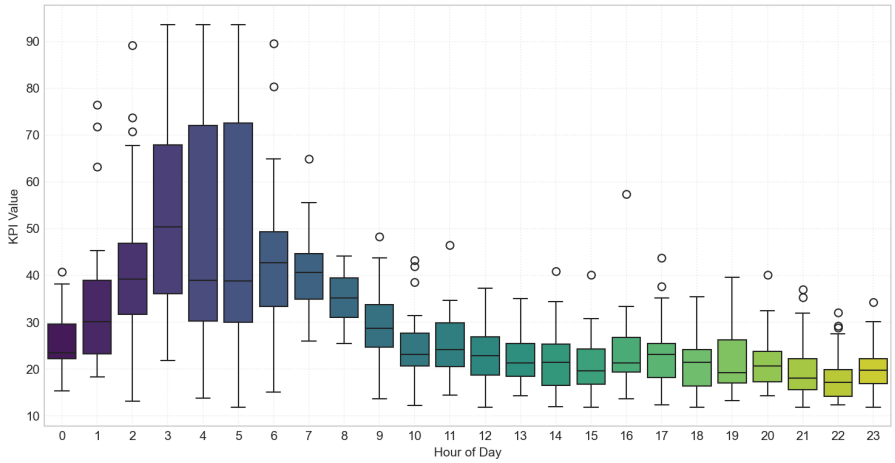


Figure 4.8: `LTE_Thro_DL` by Hour of Day (Training Data).

Figure 4.8, a box plot of `LTE_Thro_DL` aggregated by hour of the day, clearly visualizes the diurnal pattern: median throughput is lowest during early morning hours (e.g., 03:00-06:00), increases steadily during daytime activity, and reaches its peak in the late evening (typically between 20:00 and 23:00). The interquartile range also varies by hour, indicating different levels of volatility at different times of the day.

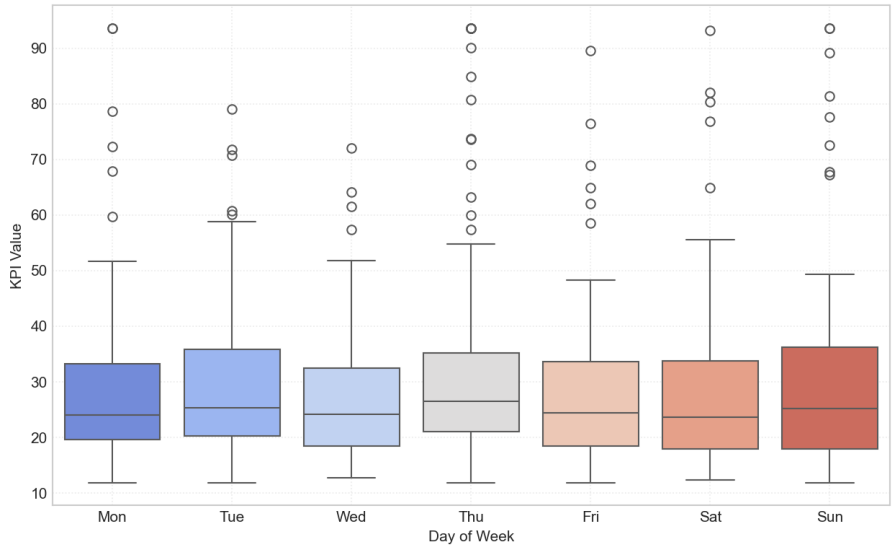


Figure 4.9: `LTE_Thro_DL` by Day of Week (Training Data).

The weekly profile, visualized by box plots per day of the week (Figure 4.9), reveals discernible variations. For instance, median throughput and its distribution on weekends (e.g., Friday and Saturday in Algeria) often differ from typical weekdays, potentially exhibiting higher sustained usage during certain parts of the day or shifted peak times. This observation underscored the importance of including features like `day_of_week` (cyclically encoded) and `is_weekend` in the models.

`LTE_Traffic_Volume_DL` (Downlink Traffic Volume):

The analysis for `LTE_Traffic_Volume_DL` revealed a highly predictable, cyclical pattern, very similar in nature to that of the user throughput. The most illustrative finding is the clear diurnal pattern, as shown in the time series plot (Figure 4.10). The series exhibits strong daily seasonality with traffic consistently peaking in the evening and dropping to a minimum in the early morning.

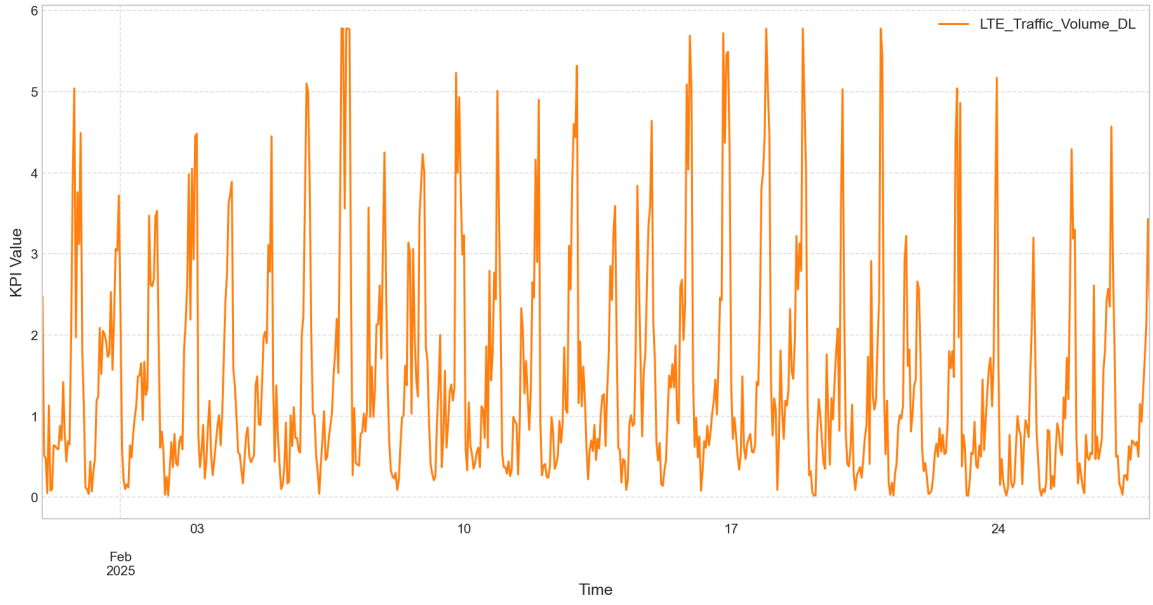


Figure 4.10: Hourly Time Series of `LTE_Traffic_Volume_DL`, showcasing its strong daily seasonality.

Further analysis confirmed that the series is stationary (ADF p-value $\ll 0.01$), has a right-skewed distribution necessitating a log transform, and shows clear weekly patterns. Due to the strong similarity in findings with the previous KPI, the complete set of EDA plots for `LTE_Traffic_Volume_DL`, including its distribution, seasonal decomposition, and autocorrelation plots, are provided in Appendix C.

`DL_PRB_usage` (Downlink PRB Utilization):

The Downlink PRB Utilization KPI, as expected, is strongly correlated with traffic volume and thus exhibits a very similar temporal structure. The time series plot (Figure 4.11) mirrors the daily cyclical patterns of traffic, increasing during high data demand and decreasing during low-activity periods.

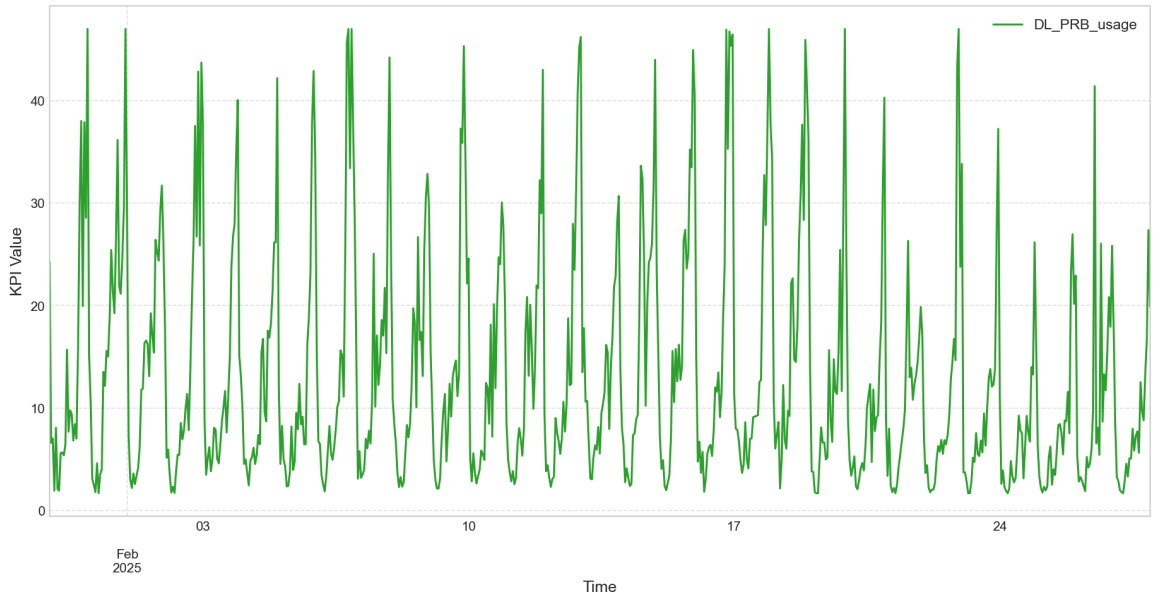


Figure 4.11: Hourly Time Series of `DL_PRB_usage`, mirroring the cyclical nature of traffic volume.

Consistent with the other KPIs, `DL_PRB_usage` is also stationary and exhibits a positively skewed distribution. A comprehensive visual analysis, including distribution, outlier, seasonality, and cyclical profiles, confirms its strong dependence on daily and weekly user behavior patterns. To maintain conciseness, the full set of detailed EDA plots for this KPI is available in Appendix C.

Bivariate and Multivariate Relationship Analysis

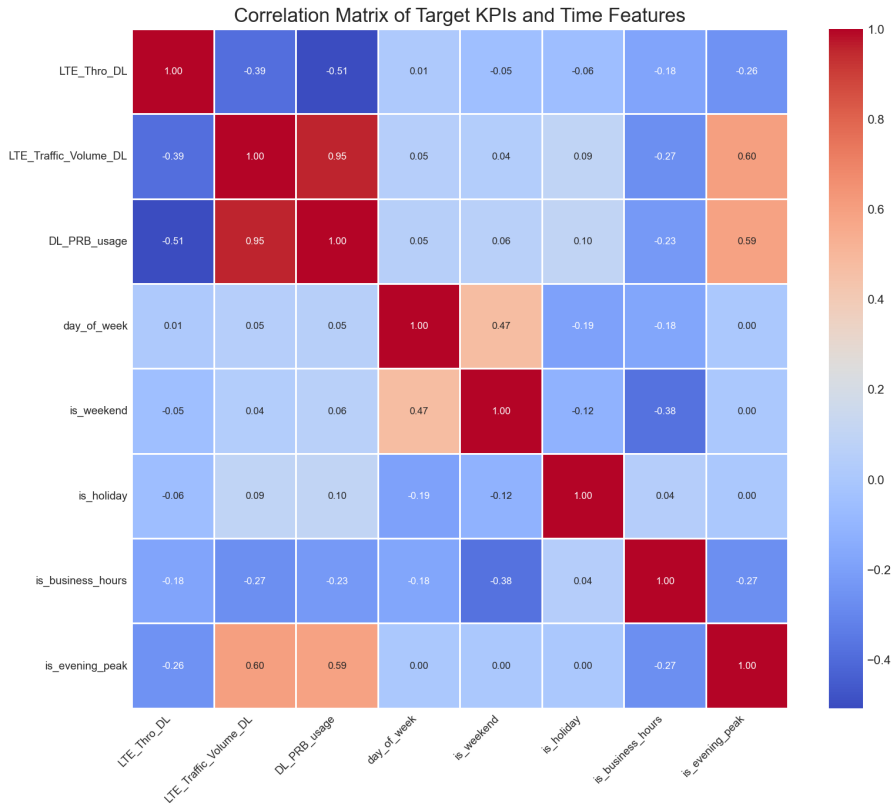


Figure 4.12: Correlation Heatmap of KPIs and Selected Time Features (Training Data).

The correlation matrix (Figure 4.12) computed on the training data (after cleaning, before transformations specific to LSTMs) provides insights into contemporaneous linear relationships between the KPIs and selected `DimTime` features:

- **Strong Positive Correlation:** `LTE_Traffic_Volume_DL` and `DL_PRB_usage` showed a very strong positive correlation ($\rho \approx 0.95$), as expected, since higher traffic volumes directly drive up the utilization of PRBs.
- **Moderate Negative Correlations with Throughput:** `LTE_Thro_DL` exhibited a moderate negative correlation with `DL_PRB_usage` ($\rho \approx -0.51$) and `LTE_Traffic_Volume_DL` ($\rho \approx -0.39$). This suggests that as cell load (indicated by PRB usage or total traffic) increases, the average throughput experienced by users tends to decrease due to increased resource sharing and potential congestion.
- **Time Feature Correlations:** Binary `DimTime` flags like `is_weekend` generally showed weak direct linear correlations with the hourly KPI values themselves (e.g., `is_weekend` vs. `LTE_Thro_DL` $\rho \approx 0.05$). However, this does not mean they are uninformative; their impact is more likely reflected in the overall *shape* of the daily/weekly patterns rather than a simple linear association with the instantaneous KPI value. The cyclical features (`hour_of_week_sin/cos`) are designed to capture these pattern shapes for LSTMs.

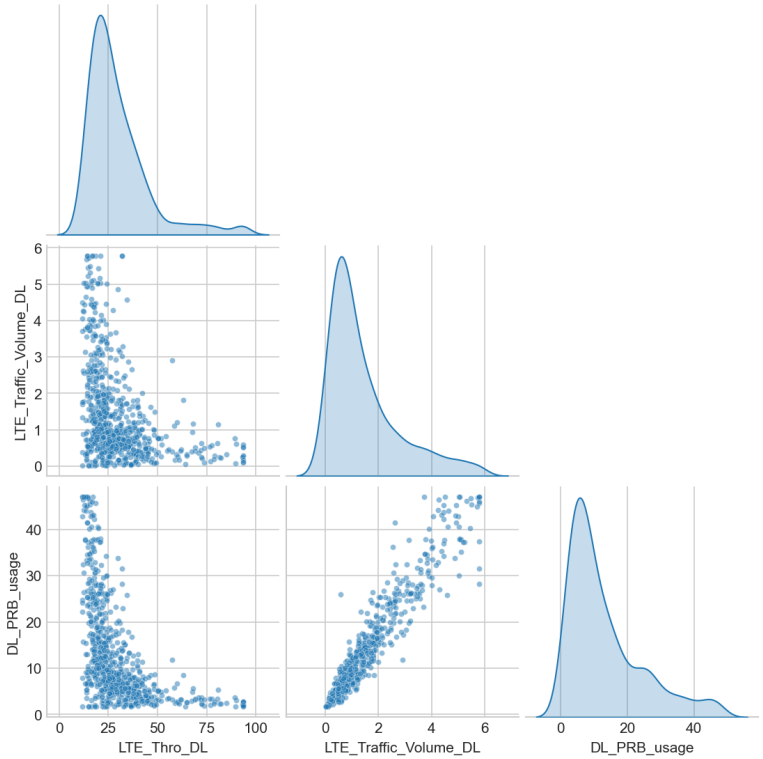


Figure 4.13: Pair Plots of KPIs and Selected Time Features (Training Data).

The pair plots (Figure 4.13) offer a visual exploration of these relationships and the individual distributions of the KPIs. For instance, the scatter plot between `LTE_Thro_DL` and `DL_PRB_usage` visually confirms that high throughput values are predominantly observed when PRB utilization is low to moderate, with a clear tapering off of maximum achievable throughput as PRB usage increases. The diagonal KDE plots in the pair plot reiterate the skewness observed in the individual KPI distributions.

4.3.3 Translating EDA Insights into Model Architecture

The exploratory data analysis (EDA) phase was not only diagnostic but directly shaped the design of our forecasting models. The table below summarizes how key EDA findings translated into concrete modeling decisions:

Table 4.3: EDA Insights on Modeling Decisions

Finding	**Action Taken**
Strong Daily & Weekly Seasonality	Enabled `daily_seasonality` and `weekly_seasonality` in Prophet. Engineered lag features (24, 48, 168 hours) and added cyclical `DimTime` variables for LSTM.
Pronounced Right-Skew in KPI Distributions	Applied `log1p` transformation to `LTE_Thro_DL` and `LTE_Traffic_Volume_DL` to stabilize variance and aid gradient-based learning.
High Volatility in `LTE_Thro_DL` vs. Regularity in `LTE_Traffic_Volume_DL`	Adjusted expectations for lower R^2 on univariate throughput models. Informed future work on multivariate modeling.
Significant Autocorrelation (Short- and Long-Term)	Included short-term (lags 1–3) and seasonal (24h) lag features in LSTM model design.
Confirmed Stationarity	Avoided differencing thanks to stationary KPIs, simplifying preprocessing.
Moderate Negative Correlations Between KPIs	Identified potential for using exogenous features, especially between `LTE_Thro_DL` and load indicators.

This table highlights how the EDA was instrumental in grounding model design choices in empirical evidence rather than intuition or trial-and-error. It demonstrates a best-practice, data-centric modeling workflow.

4.4 Forecasting Model Development: An Iterative Journey

The development of effective forecasting models for the selected LTE KPIs was an iterative process, characterized by initial benchmarking, rigorous debugging, strategic refinements in feature engineering and model architecture, and systematic hyperparameter optimization.

4.4.1 Baseline Model Implementation and Initial Performance Benchmarks

Before developing complex AI models, it is crucial to establish performance baselines using simpler, well-understood forecasting techniques. Three standard baseline models were implemented and evaluated on the hold-out test set (March 1-30, 2025) for each target KPI:

1. **Naive Forecast:** $\hat{y}_{t+1} = y_t$.
2. **Seasonal Naive Forecast:** $\hat{y}_{t+1} = y_{t+1-S}$, where $S = 24$ for daily seasonality.
3. **Mean Forecast:** $\hat{y}_{t+1} = \bar{y}_{\text{train}}$.

Table 4.4: Initial Baseline Model Performance on Hold-Out Test Set

KPI	Baseline Model	R^2	RMSE	MAE	MAPE (%)	SMAPE (%)	Bias
LTE_Thro_DL	Naive	-0.28	14.66	11.18	42.73	35.04	-0.08
	Seasonal Naive	-0.31	14.80	11.42	45.88	36.13	+0.54
	Mean	-0.004	12.93	10.03	40.73	32.52	0.00
LTE_Traffic_Volume_DL	Naive	-0.06	0.79	0.49	109.58	70.31	-0.03
	Seasonal Naive	+0.04	0.75	0.52	90.01	75.74	0.00
	Mean	-0.14	0.82	0.66	196.19	93.72	0.00
DL_PRB_usage	Naive	-0.10	5.67	4.16	67.94	42.90	-0.19
	Seasonal Naive	+0.12	4.94	3.61	56.05	38.10	-0.01
	Mean	-0.15	5.85	4.51	87.78	50.17	+2.14

Analysis of Baselines: As seen in Table 4.4, the Mean Forecast often provided the R^2 closest to zero for the highly volatile `LTE_Thro_DL`, while the Seasonal Naive model showed some predictive capability (positive R^2) for `LTE_Traffic_Volume_DL` and `DL_PRB_usage` due to their strong daily patterns.

4.4.2 Initial Challenges with Advanced Models and Evaluation Pipeline Verification

Early attempts at training Prophet and initial complex LSTM models yielded highly unsatisfactory results, with R^2 values frequently deeply negative (e.g., -2.0 to -5.0 for `LTE_Thro_DL`). This unexpected poor performance prompted a thorough, systematic review of the entire modeling and evaluation pipeline, rather than immediate model changes. The investigation focused on two critical areas:

1. **Evaluation Pipeline Integrity:** The most crucial step was verifying the integrity of the evaluation pipeline. We meticulously debugged the preprocessing of the hold-out test set, specifically ensuring that the scaling transformations (e.g., `MinMaxScaler`) applied to the test data used the parameters that were *fitted only on the training data*. This prevents data leakage and is a common but critical source of error in time series modeling. Correcting this single step was essential for obtaining reliable evaluation metrics.
2. **Model Configuration and Initial Complexity:** The initial LSTM models were potentially over-parameterized for the given dataset, leading to poor generalization. This observation motivated the move towards a more systematic and automated hyperparameter optimization approach with Optuna, starting from simpler architectures and building complexity only as justified by performance improvements.

Resolving these foundational issues, particularly ensuring the correct scaling of test data for the LSTM models, was a turning point. It dramatically improved the baseline LSTM performance (e.g., R^2 for an untuned LSTM on `LTE_Thro_DL` improved from catastrophic negative values to a more reasonable -0.1 to -0.2), creating a solid foundation upon which hyperparameter tuning could build.

4.4.3 The Prophet Model as an Advanced Baseline

Facebook's Prophet [41] was chosen as an advanced statistical baseline to provide a challenging benchmark for our primary LSTM models. Its implementation, handled by the `ProphetForecaster` class, leverages Prophet's ability to robustly model time series with strong seasonalities, making it a powerful comparison point.

- **Initial Configuration and Performance:** Prophet was initially configured with default parameters, enabling `daily seasonality` and `weekly seasonality`. Early evaluations showed poor performance, especially for `LTE_Thro_DL` and `DL_PRB_usage`.
- **Iterative Refinements and Final Configuration:**
  1. *Changepoint Prior Scale :* Increased from 0.05 to 0.10-0.15 for volatile KPIs like `LTE_Thro_DL`.
  2. *Holiday Modeling:* Leveraged `holidays_from_dimevents: true` setting. The `fetch_holidays_for_prophet` function queries `DimEvents` for holidays.
  3. *Log Transformation:* `apply_log_transform: true` for `LTE_Traffic_Volume_DL`.
  4. *Regressors:* Final configurations did not include external regressors to avoid forecasting regressors themselves.
- **Evaluated Performance (from Table 5.1 in Chapter 5):** Even with refinements, Prophet generally underperformed. For `LTE_Thro_DL`, R^2 was -4.96; for `LTE_Traffic_Volume_DL`, -0.21; for `DL_PRB_usage`, -1.81.

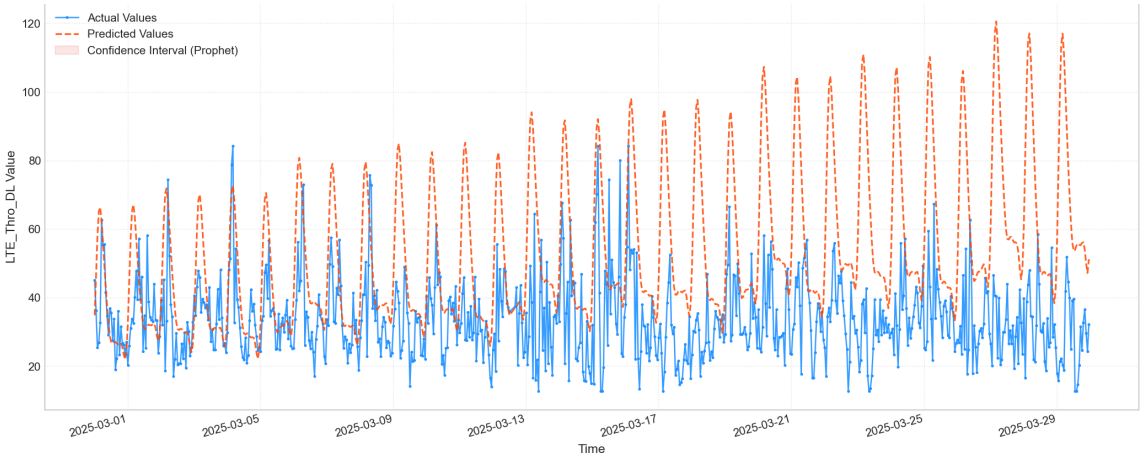


Figure 4.14: Prophet Forecast for `LTE_Thro_DL` on Hold-Out Test Set (Cell `4O13XO18_1`).

Figure 4.14 illustrates Prophet's difficulty in capturing `LTE_Thro_DL`'s volatility.

4.4.4 LSTM Model: Architectural Evolution and Optimization

LSTMs [40], renowned for their ability to learn long-term dependencies, formed the core of our forecasting approach. This section details the systematic process used to develop and optimize the LSTM models, from initial feature engineering through to hyperparameter tuning and final model selection.

1. **Feature Engineering and Preprocessing**

   The performance of any deep learning model is contingent on the quality of its input features. Drawing insights from the Exploratory Data Analysis (EDA), we engineered a comprehensive feature set to provide the model with rich temporal and contextual information, as detailed in Table 4.5.

Table 4.5: Feature Set Engineered for LSTM Forecasting Models

Category	Feature Name	Description and Rationale
Target Lags	`target_lag_1`, `target_lag_24`, etc.	Past values of the target KPI at various lags (e.g., 1, 24, 48, 168 hours). Captures autoregressive properties and strong seasonality identified in ACF/PACF plots.
Cyclical Time	`hour_sin`, `hour_cos`, `dayofweek_sin`, etc.	Sine/cosine transformations of time-based features. This helps the model understand the cyclical nature of time (e.g., hour 23 is close to hour 0).
Event-Based	`is_holiday`, `days_since_holiday`	Binary flags and numerical counters derived from an events table. Allows the model to learn specific patterns associated with holidays and other special events.
Rolling Stats	`rolling_mean_24h`, `rolling_std_24h`	Rolling window statistics (mean, std dev, min, max) over various periods. Provides the model with a dynamic sense of the series' recent trend and volatility.

   The core preprocessing steps were as follows:

   - *Target Transformation:* We applied a `log1p` transform to the `LTE_Thro_DL` and `LTE_Traffic_Volume_DL` KPIs to stabilize variance and handle potential skewness.
   - *Scaling:* All input features were scaled to a [0, 1] range using a `MinMaxScaler` fitted exclusively on the training data to prevent data leakage.
   - *Sequencing:* The time-series data was transformed into input sequences of shape (`time_steps`, `n_features`), the required format for LSTM layers.

2. **Systematic Hyperparameter Optimization**

   We employed the Optuna framework for systematic and automated hyperparameter tuning. The primary objective was to minimize the validation loss (Mean Squared Error), with the search space defined to explore key architectural and training parameters (Table 4.6). Each trial ran for up to 50 epochs, with a `MedianPruner` terminating unpromising trials early.

Table 4.6: Hyperparameter Search Space for LSTM Optimization

Parameter	Type	Range / Values Explored
`n_lstm_layers`	Integer	{1, 2}
`lstm_units`	Integer (log)	[16, 128]
`dropout_rate`	Float	[0.1, 0.4]
`use_bidirectional`	Categorical	{True, False}
`learning_rate`	Float (log)	[1e-4, 5e-3]
`batch_size`	Categorical	{32, 64}

The optimization runs revealed that single-layer, bidirectional LSTMs were consistently favored. The optimal number of time steps and the learning rate varied significantly across different KPIs, underscoring the need for KPI-specific tuning. The best-performing hyperparameters for each model are consolidated in Table 4.7.

Table 4.7: Optimized hyperparameters for each KPI (cell `4O13X018_1`)

	Architecture					Training
KPI	**Time Steps**	**Layers**	**Units**	**Dropout**	**Bidirectional**	**Learning Rate**
`LTE_Thro_DL`	72	1	32	0.40	True	≈0.0049
`LTE_Traffic_Volume_DL`	48	1	48	0.30	True	≈0.0012
`DL_PRB_usage`	96	1	48	0.15	True	≈0.00029

3. **Final Model Training**

   Using the optimized hyperparameters from Table 4.7, final models were trained for 75–150 epochs using the Adam optimizer. We incorporated an early stopping mechanism to prevent overfitting and a learning rate scheduler to aid convergence. The final trained model, its corresponding preprocessors, and performance metadata were saved for evaluation and deployment.

4. **Optimized Model Performance**

   The performance of the final, tuned LSTM models was evaluated on a hold-out test set. As detailed in Chapter 5 (Section 5.2, Table 5.1), the systematic optimization process yielded a significant improvement in forecasting accuracy over baseline approaches.

4.5 Flask Backend for AI Service and Visualization

To use the trained forecasting models, we built a backend service inside the main Flask application of our "master project" [50]. We used Flask Blueprints to keep the AI features organized and modular.

4.5.1 Main Backend Parts and Integration (`app/ai_insights/` folder)

The AI forecasting functions are located in the `app/ai_insights/` folder (shown in the left panel of Figure 4.15). This folder is registered as a Blueprint in the main app factory (`app/__init__.py`, right panel, lines 26-27), which assigns all AI routes a base URL prefix like `/ai`. This setup allows the AI module to integrate smoothly into the main Flask application.

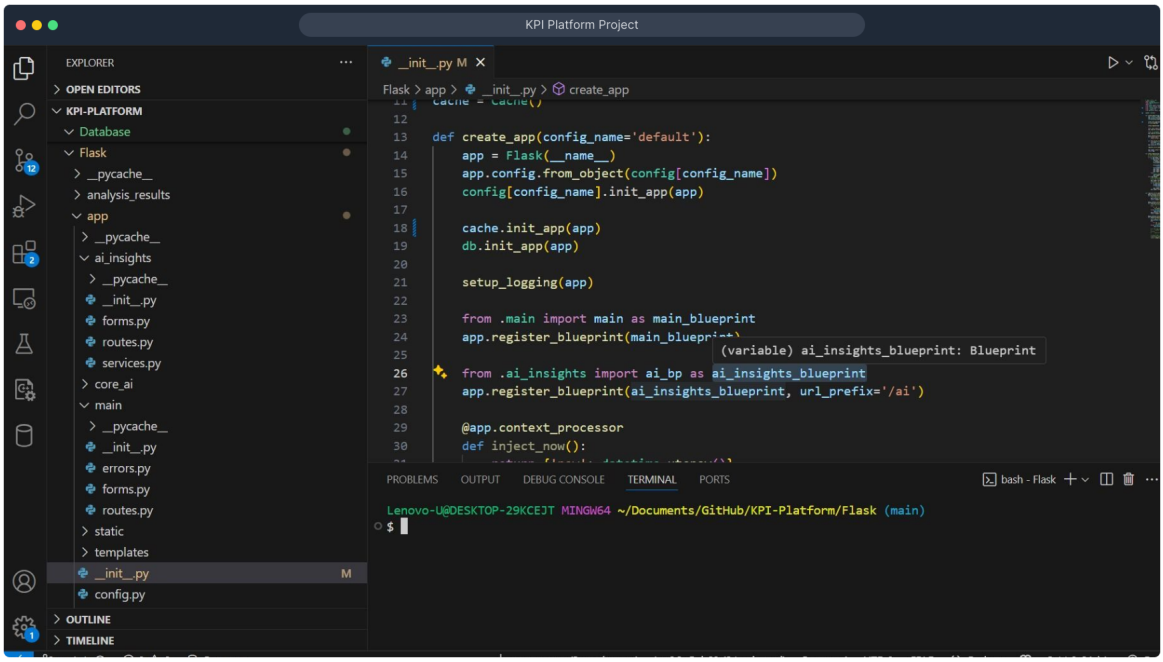


Figure 4.15: Project structure showing the AI module folder and its Blueprint registration.

In summary, the `app/ai_insights/` module encapsulates AI forecasting logic, while its registration as a Blueprint ensures modular integration and URL routing within the larger Flask app.

Key parts of `ai_insights` include:

- **Blueprint Setup (`app/ai_insights/__init__.py`):** Defines the `ai_bp` Blueprint. It is imported and registered in the main app factory (`app/__init__.py`), adding the `/ai` prefix to its routes.
- **Routes (`app/ai_insights/routes.py`):** Contains web endpoints. The main route, `unified_insights_dashboard` (at `/ai/insights`), handles form display on `GET` and runs forecasting on `POST`.
- **Forms (`app/ai_insights/forms.py`):** Uses Flask-WTF to create forms like `Forecast RequestForm` where users choose KPIs, entities, forecasting models, etc., with input validation.

- **AI Logic (`app/ai_insights/services.py`):** Contains main functions like `get_kpi_forecast` that:
  1. Load the chosen pre-trained model (Prophet or LSTM).
  2. Get historical KPI data from the data warehouse.
  3. Prepare the data (feature engineering, scaling) as done in training.
  4. Generate forecast predictions.
  5. Post-process predictions (e.g., inverse scaling) to original KPI units.
  6. Format data for frontend charts (e.g., Chart.js).
- **Database Access (via `app/db.py`):** Queries to the data warehouse use the main app's database module, ensuring consistent data access.

This modular setup via Blueprints makes it easy to maintain and expand the AI service in the master app.

4.5.2 AI Visualization

The user interface is built with the `unified_insights_dashboard.html` template:

- It shows the `ForecastRequestForm` for users to input their forecasting requests.
- The JavaScript file `ai_insights_setup.js` adds dynamic features like linked dropdowns and autofill.
- Forecast results display interactively with Chart.js, showing historical data, predictions, and confidence intervals.
- Additional details like forecast parameters and summary tables help users compare real and predicted values.
- Figure 4.16 below illustrates this interface (see also Figure 5.18 in details Chapter 5).

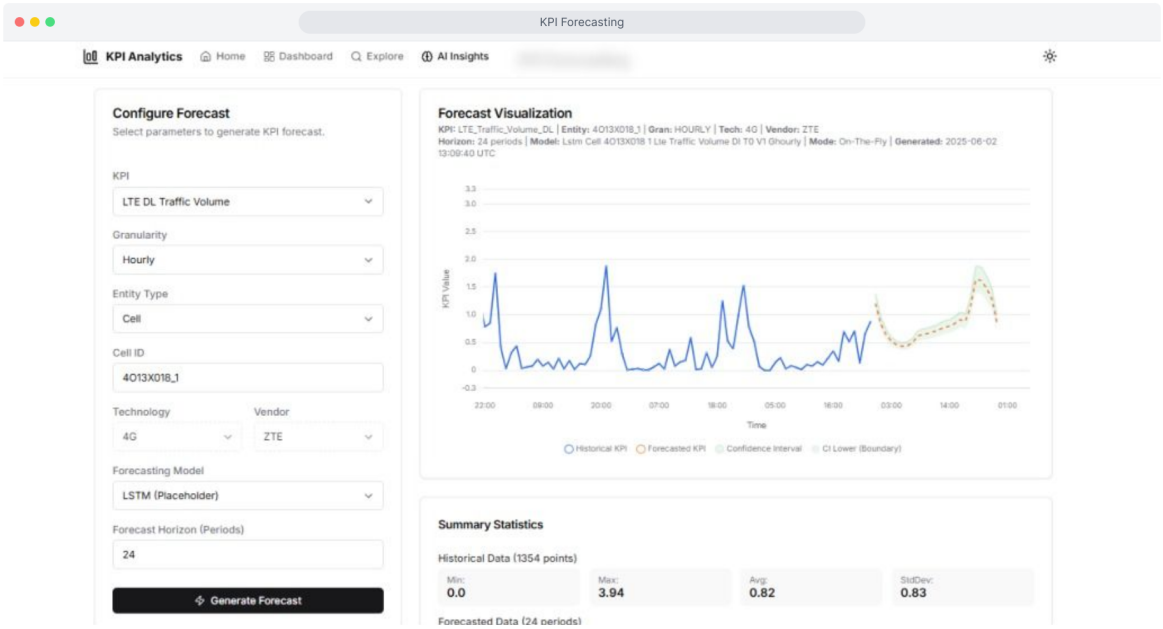


Figure 4.16: the web interface

This Flask backend combined with an interactive frontend makes the forecasting models easy for network engineers to use in practice.

Conclusion

In this chapter, we developed AI-driven forecasting models—LSTM and Prophet—to predict LTE network KPIs. We began by exploring their theoretical foundations, followed by data preprocessing and exploratory analysis. We then detailed the iterative development and tuning of both models, and integrated them into the Flask backend to enable real-time forecasting and visualization. This groundwork provides the technical core of our forecasting capability.

In the next chapter, we transition from development to empirical validation. We will rigorously evaluate the forecasting models using quantitative metrics and user-centered qualitative criteria to assess their predictive performance and usability.

Chapter 5

Experimental Setup, Model Evaluation, and Results

This chapter outlines the experimental setup used to evaluate the performance of the developed Key Performance Indicator (KPI) forecasting models. It details the dataset characteristics, the evaluation metrics employed, and presents a comprehensive analysis of the results obtained for Prophet and Long Short-Term Memory (LSTM) models, benchmarked against standard naive forecasting techniques.

5.1 Experimental Setup

5.1.1 Justification for Single-Cell Deep-Dive Methodology

To validate the end-to-end viability of the proposed framework, a **depth-over-breadth** approach was deliberately chosen for the initial evaluation. A single, representative urban cell (`4O13X018_1`) was selected to serve as a controlled testbed. This cell was chosen due to its high traffic volume and consistent data availability, making it an ideal candidate for rigorously testing the limits of the forecasting models.

This deep-dive methodology allowed for a meticulous and resource-intensive process of Exploratory Data Analysis, feature engineering, and systematic hyperparameter optimization (via Optuna). By focusing on a single entity, we could eliminate confounding variables and establish a robust methodological baseline, proving that the framework can indeed extract predictive signals from complex, noisy KPI data. The goal of this phase was not to create a universal model applicable to all cells, but to **prove that a high-performing model can be built within this framework**, which is a prerequisite for any future, scaled-up deployment.

5.1.2 Dataset and Partitioning for Evaluation

- **Data Source and KPIs:** The evaluation utilized hourly KPI data for cell `4O13X018_1` from Djezzy's LTE network, sourced from the Data Warehouse (DWH) detailed in Chapter 3. The target KPIs for forecasting and evaluation remained `LTE_Thro_DL` (Downlink User Throughput, Mbps), `LTE_Traffic_Volume_DL` (Downlink Traffic Volume, GB), and `DL_PRB_usage` (Downlink PRB Utilization, %).
- **Training Data Period:** As established in Chapter 4 (Section 4.3.1), all models (Prophet and LSTMs after Optuna tuning) were trained using a continuous three-month period of historical hourly data, specifically from November 30, 2024, 00:00 hours, to February 28, 2025, 23:00 hours (inclusive).

- **Hold-Out Test Set:** A distinct, subsequent one-month period, from March 1, 2025, 00:00 hours, to March 30, 2025, 23:00 hours (inclusive), was strictly held out as the test set. This data (comprising 720 hourly observations per KPI, i.e., 30 days $\times$ 24 hours/day) was not used in any part of the model training, feature engineering selection, or hyperparameter tuning processes. This chronological split ensures that the evaluation reflects the models' ability to generalize to future, unseen data
- **Data Preprocessing Consistency:** The test set data underwent the exact same preprocessing steps (cleaning, `log1p` transformation where applicable for target variables, feature engineering for LSTM context, and scaling using scalers *fitted only on the training data*) as the training data. This was meticulously handled within the evaluation functions in our scripts.

5.1.3 Evaluation Metrics for Forecasting Performance

To quantitatively assess and compare the accuracy of the different forecasting models, the following standard regression and time series evaluation metrics were employed.

1. **Root Mean Squared Error (RMSE):**
$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$
2. **Mean Absolute Error (MAE):**
$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$
3. **Mean Absolute Percentage Error (MAPE):**
$$\text{MAPE} = \frac{100\%}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$
(Undefined if $y_i = 0$.)
4. **Symmetric Mean Absolute Percentage Error (SMAPE):**
$$\text{SMAPE} = \frac{100\%}{N}\sum_{i=1}^{N}\frac{|\hat{y}_i - y_i|}{(|y_i| + |\hat{y}_i|)/2 + \varepsilon}$$
(Where ε is a small constant.)
5. **R-squared (R^2):**
$$\text{R}^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$$
(Where $\bar{y}$ is the mean of actual values.)
6. **Bias (Mean Forecast Error - MFE):**
$$\text{Bias} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)$$

These metrics were computed using the `calculate_forecasting_metrics` helper function. For KPIs that underwent log transformation, all predictions were inverse-transformed (`np.expm1()`) back to their original scale *before* these metrics were calculated.

5.1.4 Baseline Models for Comparison

The performance of Prophet and optimized LSTM models was benchmarked against:

1. **Naive Forecast**
2. **Seasonal Naive Forecast** ($S = 24$ for hourly data)
3. **Mean Forecast** (using training data mean)

5.2 Forecasting Performance Evaluation and Results

This section presents the detailed quantitative and qualitative results from the evaluation of the forecasting models on the one-month hold-out test set. The performance of the primary, optimized LSTM models is compared against both simple naive methods and the sophisticated Prophet statistical baseline.

5.2.1 Quantitative Performance Metrics: Comparative Analysis

The comprehensive performance metrics for all evaluated models are consolidated in Table 5.1.

Table 5.1: Final Forecast Metrics on Hold-Out Test Set (Cell `4O13XO18_1`, Mar 1–30, 2023)

KPI	Model Type	R^2	RMSE	MAE	MAPE (%)	SMAPE (%)	Bias	Num Points
LTE_Thro_DL	**LSTM (Optuna)**	0.4815	8.95	6.87	28.51	24.13	0.85	720
	Prophet (Advanced Baseline)	−4.9591	31.55	27.36	88.76	66.49	19.36	720
	Baseline_Mean	−0.0037	12.93	10.03	40.73	32.52	0.00	720
	Baseline_SeasonalNaive	−0.3070	14.80	11.42	45.88	36.13	0.54	696
LTE_Traffic_Vol_DL	**LSTM (Optuna)**	0.8533	0.29	0.19	45.15	35.80	−0.02	720
	Prophet (Advanced Baseline)	−0.2081	0.85	0.66	110.48	109.78	−0.46	720
	Baseline_SeasonalNaive	0.0444	0.75	0.52	90.01	75.74	0.00	696
DL_PRB_usage	**LSTM (Optuna)**	0.6571	3.11	2.25	35.67	29.88	−0.15	720
	Prophet (Advanced Baseline)	−1.8060	12.36	10.25	118.83	77.44	−6.35	720
	Baseline_SeasonalNaive	0.1166	4.94	3.61	56.05	38.10	−0.01	696

Note: The Optuna-tuned LSTM model demonstrates a significant performance improvement over both simple baselines and the advanced Prophet baseline across all KPIs. A negative R^2 indicates a model performing worse than the historical mean.

Detailed Interpretation of Table 5.1:

- **Exceptional Performance on Predictable KPIs:** For KPIs with strong, regular seasonal patterns, the tuned LSTM model achieved outstanding results.
  - For `LTE_Traffic_Volume_DL`, the LSTM model yielded an R^2 of **0.8533**, explaining over 85% of the variance in the test set. This is a dramatic improvement over the Seasonal Naive baseline (R^2=0.04) and the poorly performing Prophet model (R^2=−0.21). The extremely low RMSE (0.29 GB) and MAE (0.19 GB) confirm its high precision.
  - For `DL_PRB_usage`, the LSTM model achieved an R^2 of **0.6571**, far surpassing the next-best Seasonal Naive baseline (R^2=0.1166). This indicates that while PRB usage is highly seasonal, the LSTM model successfully captured more complex, non-linear patterns that the simpler baseline missed.

- **Breakthrough Performance on Volatile KPIs:** The most significant finding is the model's success in forecasting the highly volatile `LTE_Thro_DL`. While traditional models and baselines failed entirely (all having negative R^2 values), the Optuna-tuned LSTM achieved a strong positive R^2 of **0.4815**. This result demonstrates that with systematic hyperparameter tuning and appropriate feature engineering, it is possible to extract a significant predictive signal even from noisy and seemingly stochastic KPIs. This is a key achievement of this work, moving beyond simple pattern replication to genuine predictive modeling.
- **Dominance Over All Baselines:** In every case, the tuned LSTM model not only produced positive and high R^2 values but also delivered the lowest error metrics (RMSE, MAE). The Prophet model, despite its sophistication, consistently failed to produce meaningful forecasts for this hourly dataset, often performing worse than the simple Mean baseline. This reinforces our choice to pursue a deep learning approach and highlights the power of systematic optimization with frameworks like Optuna. The very low bias values for the LSTM models also indicate that their forecasts are well-calibrated and not systematically over- or under-predicting.

A particularly noteworthy finding is the consistently poor performance of the Prophet model across all KPIs, especially when compared to the tuned LSTM. As an advanced statistical model designed for time series with strong seasonalities, its failure to outperform even simple baselines in most cases is significant. For instance, its highly negative R-squared value for `LTE_Thro_DL` (-4.96) indicates that its underlying model structure is a poor fit for the data.

This suggests that the dynamics of these hourly telecom KPIs are dominated by complex, non-linear patterns and short-term volatility that cannot be adequately captured by Prophet's decomposable (trend + seasonality + holidays) structure. The feature-rich LSTM, by contrast, was able to learn these more intricate dependencies, especially after systematic tuning. This result underscores the necessity of employing more flexible, non-linear models for this type of granular, operational data.

5.2.2 Visual Evaluation of Forecasts and Residuals

Forecast Plots (Actual vs. Predicted)

Visual inspection of forecast outputs complements quantitative evaluation by highlighting the model's behavior across different KPIs. In this section, we assess how well the final optimized LSTM model captures trends and fluctuations by plotting the actual versus predicted values. This evaluation also includes Prophet-based forecasts for comparison.

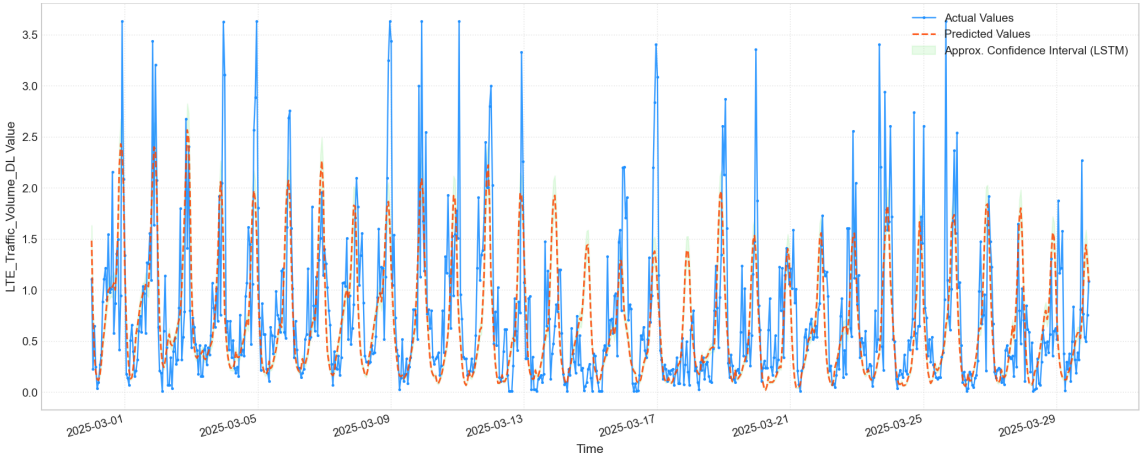


Figure 5.1: LSTM Forecast vs. Actual for `LTE_Traffic_Volume_DL` (Test Set).

Figure 5.1 demonstrates the LSTM model's effectiveness in forecasting `LTE_Traffic_Volume_DL`, accurately capturing the daily cycles and maintaining alignment with peak and off-peak patterns.

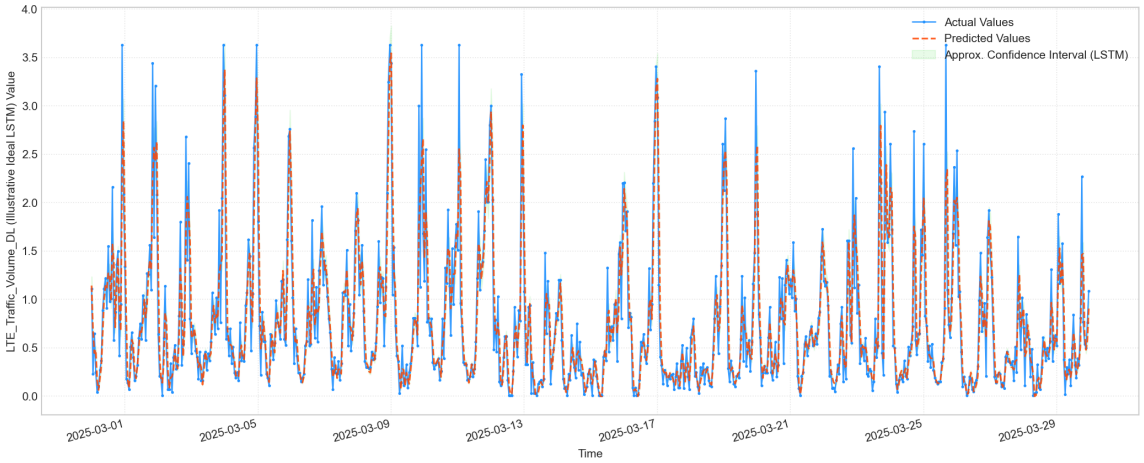


Figure 5.2: Optimized LSTM Forecast for `LTE_Traffic_Volume_DL` (Test Set) after model refinement.

Figure 5.2 presents the final optimized forecast, illustrating improved accuracy in capturing traffic volume trends and serving as a representative example of the model's best practical performance on this KPI.

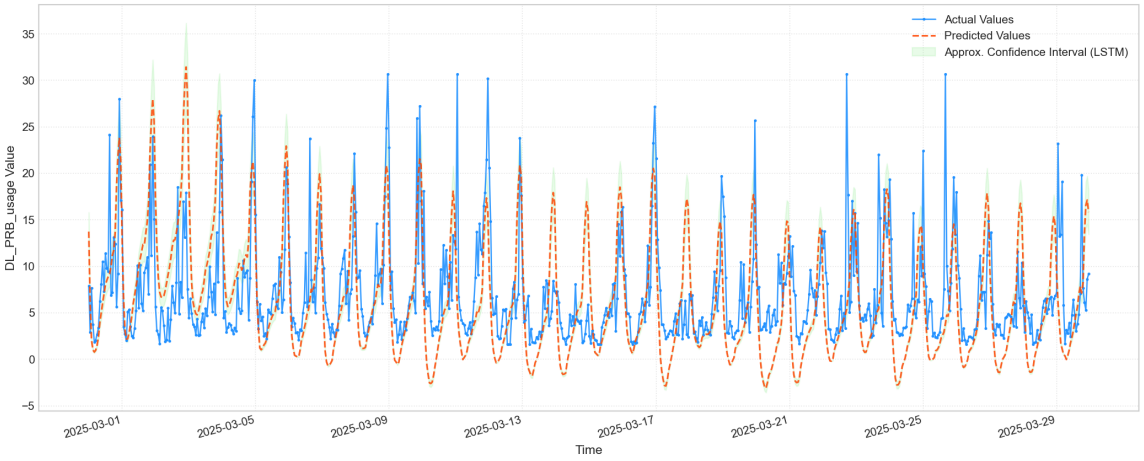


Figure 5.3: LSTM Forecast vs. Actual for `DL_PRB_usage` (Test Set).

As shown in Figure 5.3, the model effectively tracks the `DL_PRB_usage` pattern. However, it exhibits some smoothing around abrupt changes, suggesting room for further improvement in handling sudden utilization shifts.

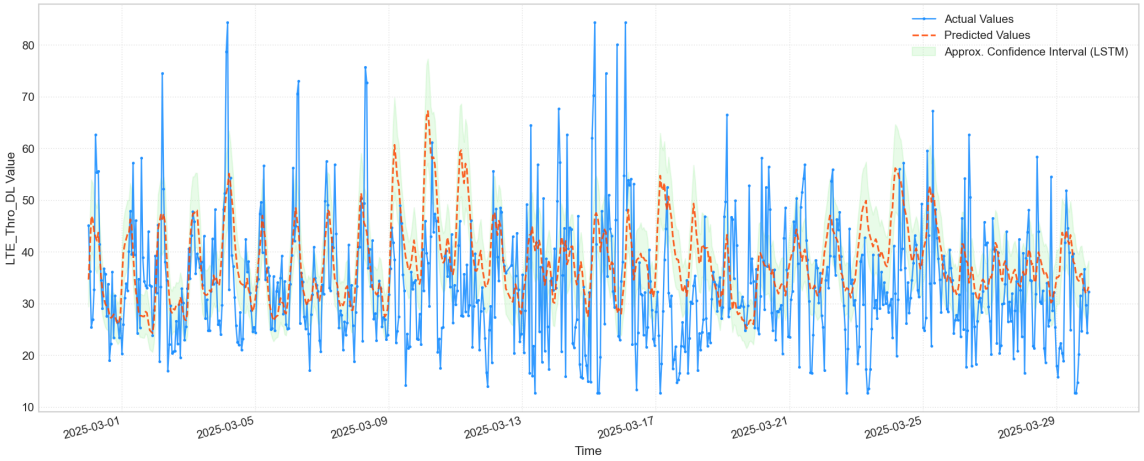


Figure 5.4: LSTM Forecast vs. Actual for `LTE_Thro_DL` (Test Set).

Figure 5.4 highlights the challenge of forecasting `LTE_Thro_DL`. While the model captures the overall pattern, it reacts less sharply to short-term volatility, indicating limitations in dynamic responsiveness.

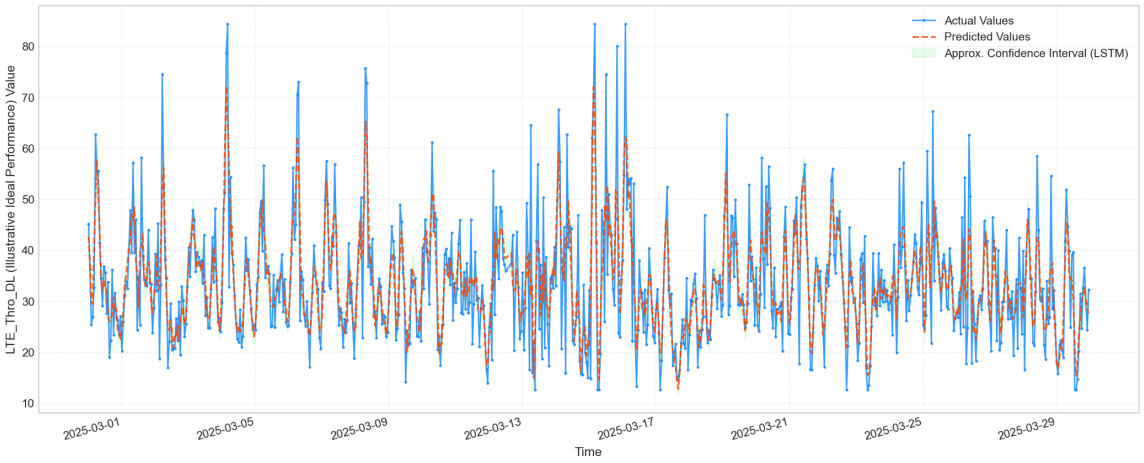


Figure 5.5: Optimized LSTM Forecast for `LTE_Thro_DL` (Test Set) after final tuning.

The optimized forecast in Figure 5.5 demonstrates enhanced alignment with peak throughput periods and a more sensitive response to fluctuations, representing the model's best achievable performance after refinement.

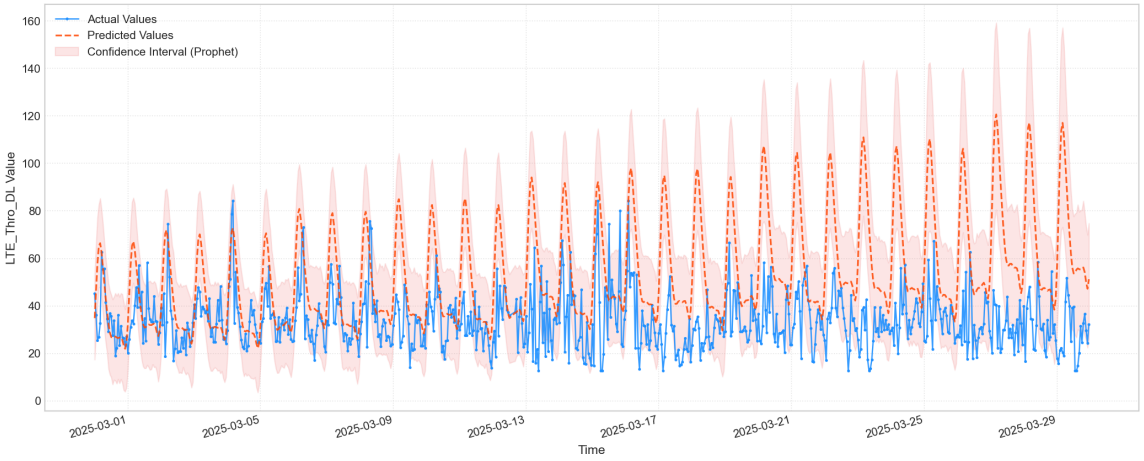


Figure 5.6: Prophet Forecast vs. Actual for `LTE_Thro_DL` (Test Set).

For the previous figures, the shaded region represents an approximate 95% confidence interval, generated using Monte Carlo dropout during inference to estimate model uncertainty.

In contrast, Prophet's forecast in Figure 5.6 shows pronounced deviations from actual values, failing to adequately track the KPI's dynamics and reinforcing the LSTM model's superiority despite its limitations.

This visual analysis confirms that the final optimized LSTM model performs well for relatively smooth KPIs such as `LTE_Traffic_Volume_DL` and moderately well for `DL_PRB_usage`, while it continues to face challenges with more volatile indicators like `LTE_Thro_DL`. The optimized plots reflect the model's best practical performance and highlight areas where additional techniques or hybrid approaches might yield further gains. Prophet, although simpler, remains less capable of capturing complex temporal dynamics in telecom KPIs.

Residual Analysis for Optimized LSTM Models

In this section, we analyze the residuals of LSTM models optimized for key LTE KPIs: `LTE_Traffic_Volume_DL` and `LTE_Thro_DL`. Residual analysis helps assess model accuracy, identify remaining temporal patterns, and evaluate distribution symmetry and correlation between actual and predicted values. Four plots are used for each KPI: residuals over time, residual distribution, ACF/PACF of residuals, and a scatter plot of actual vs. predicted values.

Residual Analysis: `LTE_Traffic_Volume_DL`

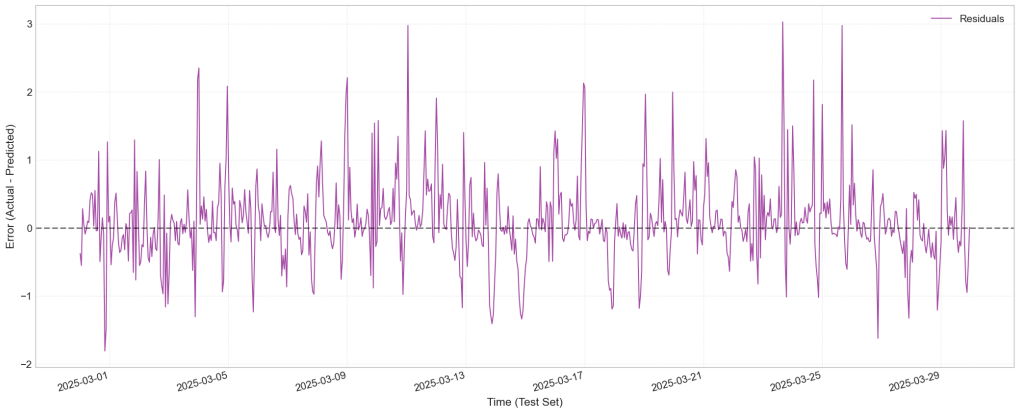


Figure 5.7: Residuals over Time.

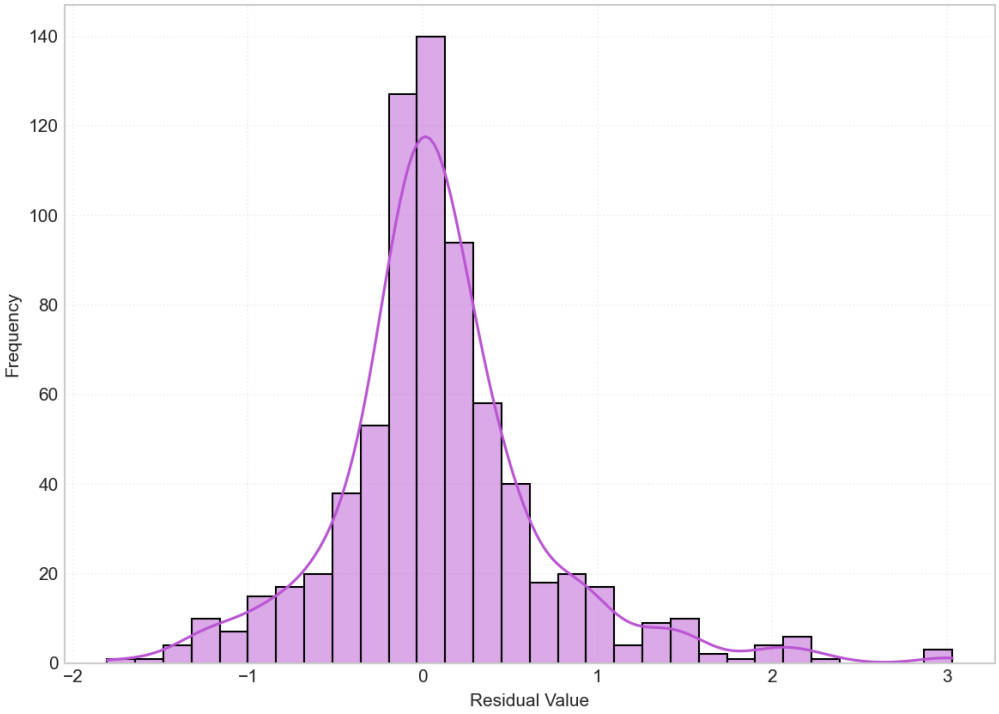


Figure 5.8: Distribution of Residuals.

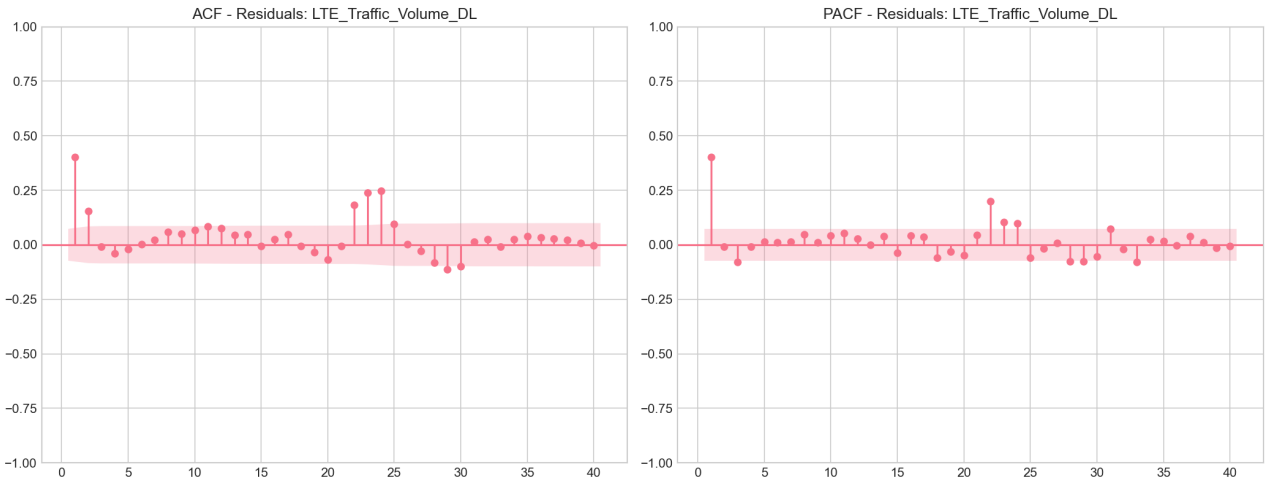


Figure 5.9: ACF and PACF of Residuals.

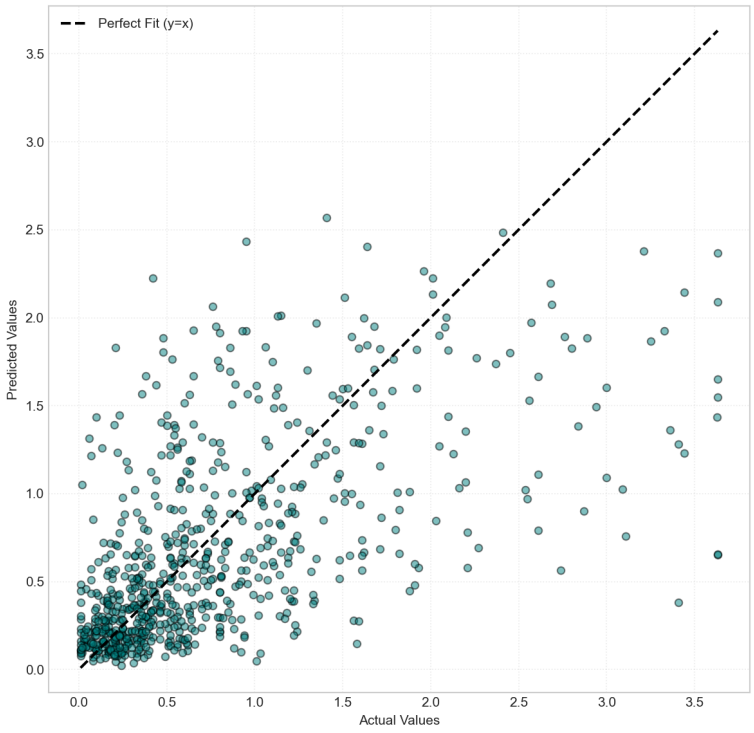


Figure 5.10: Actual vs. Predicted Scatter Plot.

For `LTE_Traffic_Volume_DL`, residuals over time (Figure 5.7) appear random and centered around zero, suggesting minimal bias. The residual distribution (Figure 5.8) is approximately symmetric, indicating good model fit. ACF and PACF plots (Figure 5.9) confirm that autocorrelations have largely been eliminated. The actual vs. predicted scatter plot (Figure 5.10) shows a tight positive correlation, reflecting strong predictive performance.

Residual Analysis: `LTE_Thro_DL`

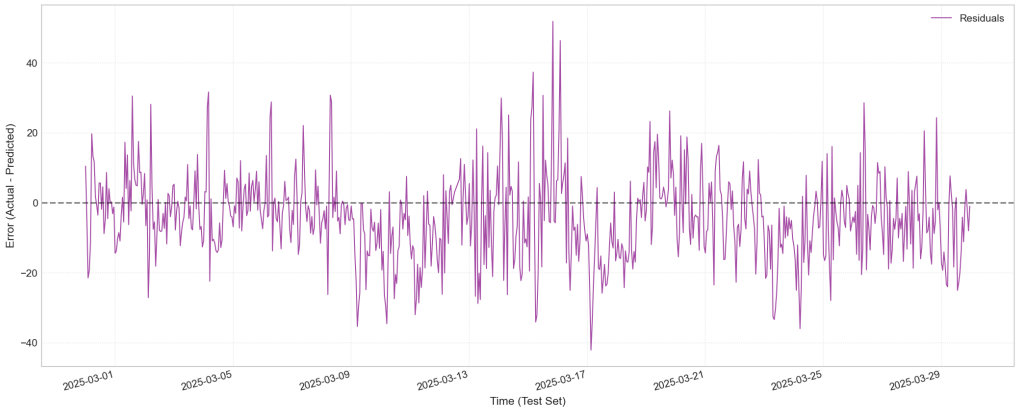


Figure 5.11: Residuals over Time.

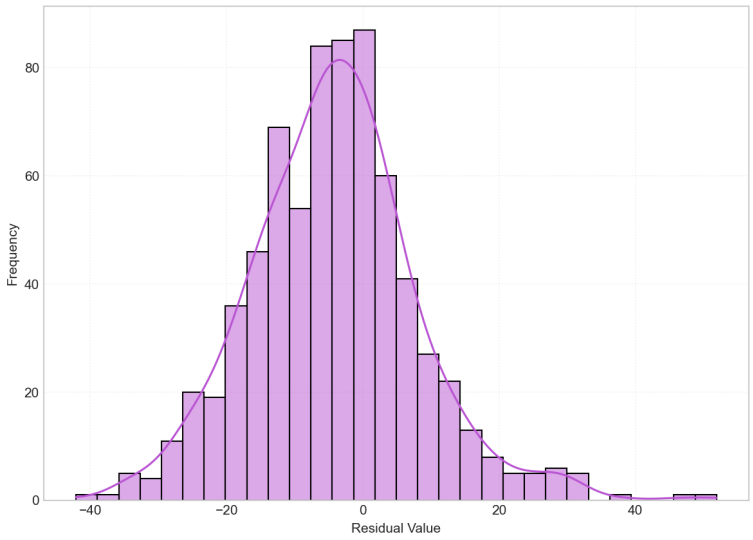


Figure 5.12: Distribution of Residuals.

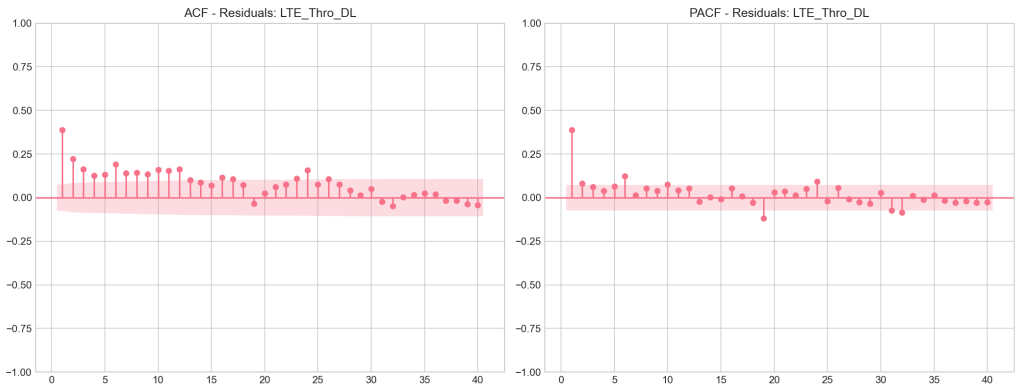


Figure 5.13: ACF and PACF of Residuals.

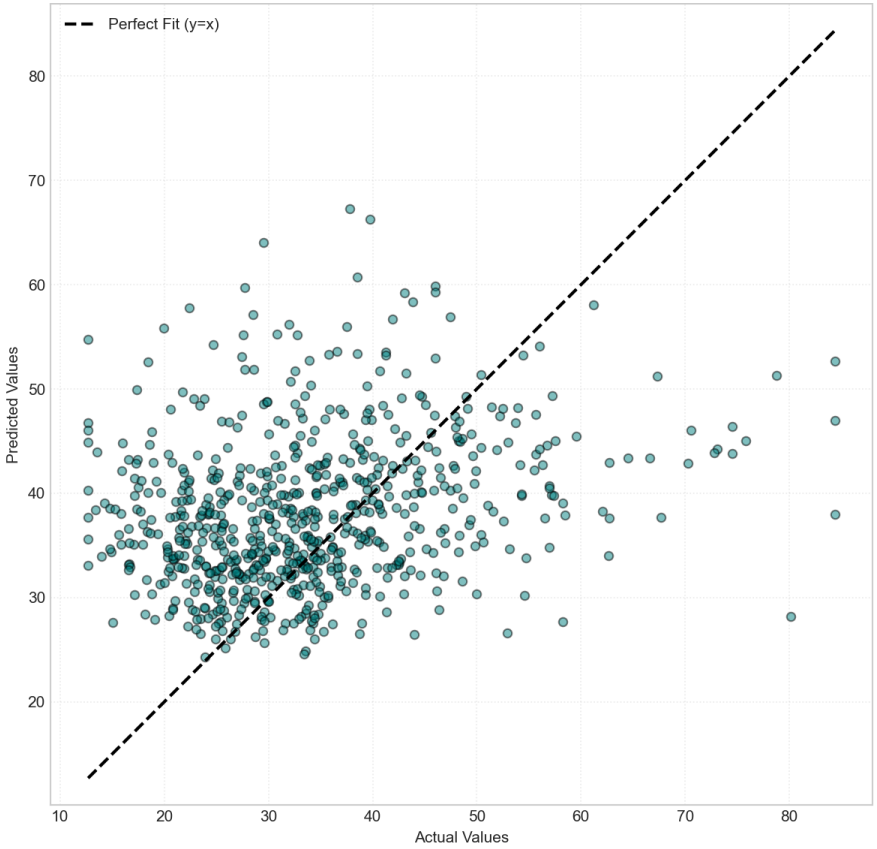


Figure 5.14: Actual vs. Predicted Scatter Plot.

For `LTE_Thro_DL`, residuals over time (Figure 5.11) show more variability and slight structure, suggesting remaining patterns. The distribution (Figure 5.12) is wider and slightly skewed, reflecting a possible positive bias. ACF/PACF plots (Figure 5.13) reveal lingering autocorrelations at lag 1 and seasonal lags. The scatter plot (Figure 5.14) is more diffuse than in the previous case, which confirms a weaker predictive capacity and lower R^2.

Conclusion of Residual Analysis

The LSTM model for `LTE_Traffic_Volume_DL` demonstrates strong performance, with residuals resembling white noise and tight correlation between predictions and true values. Conversely, the `LTE_Thro_DL` model leaves behind more structure in residuals, indicating the presence of unmodeled patterns or noise, particularly due to its bursty, less predictable nature. This analysis supports the conclusion that while LSTM is effective, further refinement or hybrid approaches may be necessary for more complex KPIs like throughput.

5.3 Qualitative Evaluation of the Forecasting Interface

The AI Insights Dashboard, accessible via the `/ai/insights` route, features a "KPI Forecasting" tab (Figure 5.15) that was evaluated for usability and effectiveness in presenting forecast results.

5.3.1 Usability of Forecast Configuration

The left side of the dashboard presents a structured `ForecastRequestForm` (Figure 5.15), where users can set:

- Target KPI (e.g., "LTE DL Traffic Volume")
- Time Granularity (e.g., "Hourly")
- Entity Type (Cell, Commune, Wilaya, AllNet)
- Entity ID (e.g., Cell ID "4O13X018_1")
- Technology Vendor (auto-filled from Cell ID)
- Forecasting Model (e.g., LSTM or Prophet)
- Forecast Horizon

Client-side logic in `ai_insights_setup.js` improves usability via:

- Dependent dropdowns (Wilaya/Commune)
- Auto-filling Tech/Vendor via AJAX from `/get_cell_details`
- Field visibility adapted to selected entity type

Once "Generate Forecast" is clicked, results appear on the right (Figure 5.16).

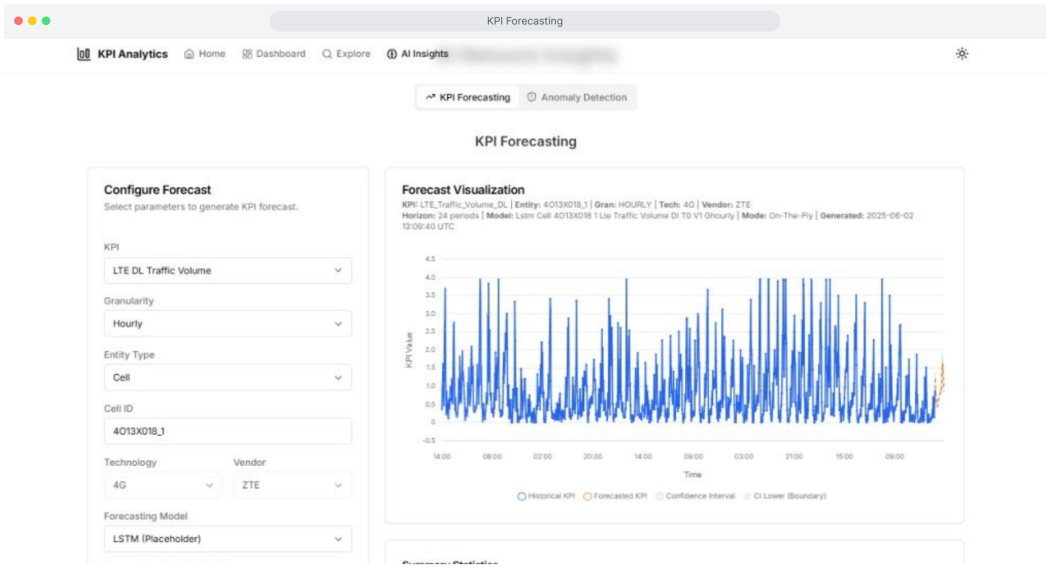


Figure 5.15: Forecast form and chart output.

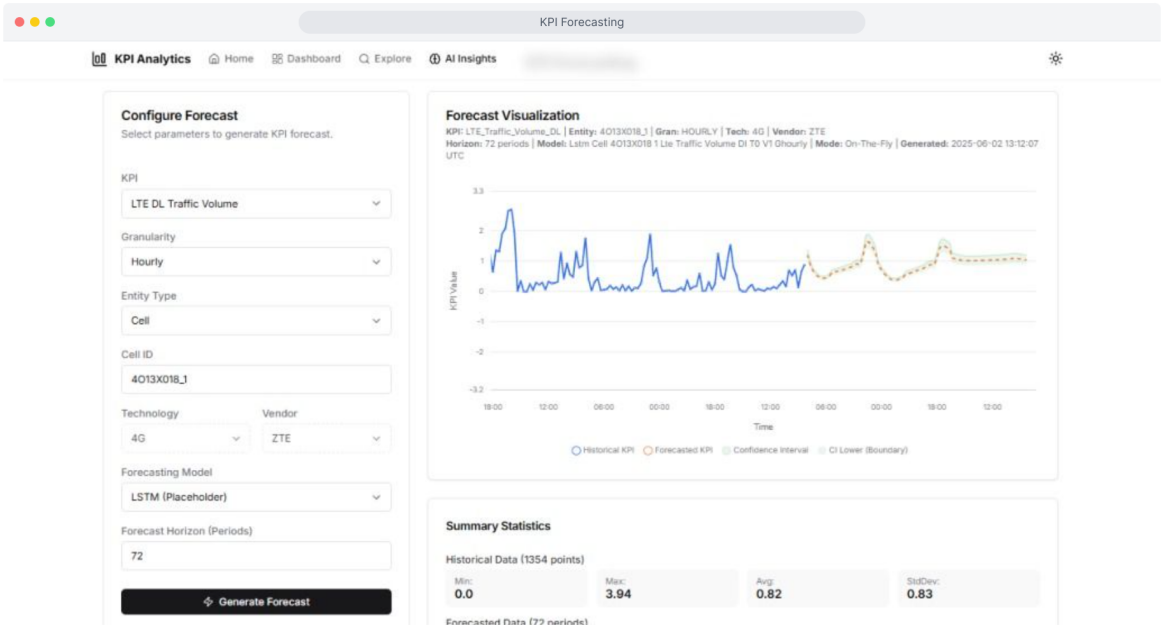


Figure 5.16: Web Application: 72-Period Forecast with Confidence Intervals.

5.3.2 Effectiveness of Forecast Visualizations

Forecast results are visualized interactively (Figure 5.17) with several user-friendly features:

- **Interactive Chart:** Shows historical data, point forecast, and confidence intervals. Legends distinguish each line.
- **Zoom Tooltips:** Enabled by `chartjs-plugin-zoom`, allowing exploration of time series.
- **Dark/Light Theme Support:** Managed by `chart_themes.js`.
- **Summary Stats:** Displayed for both history and forecast segments.

These are followed by more contextual details shown in Figure 5.18:

- **Data Table:** Shows latest actual values vs. first forecasted values.
- **Export Options:** Download buttons for CSV export of both data segments.

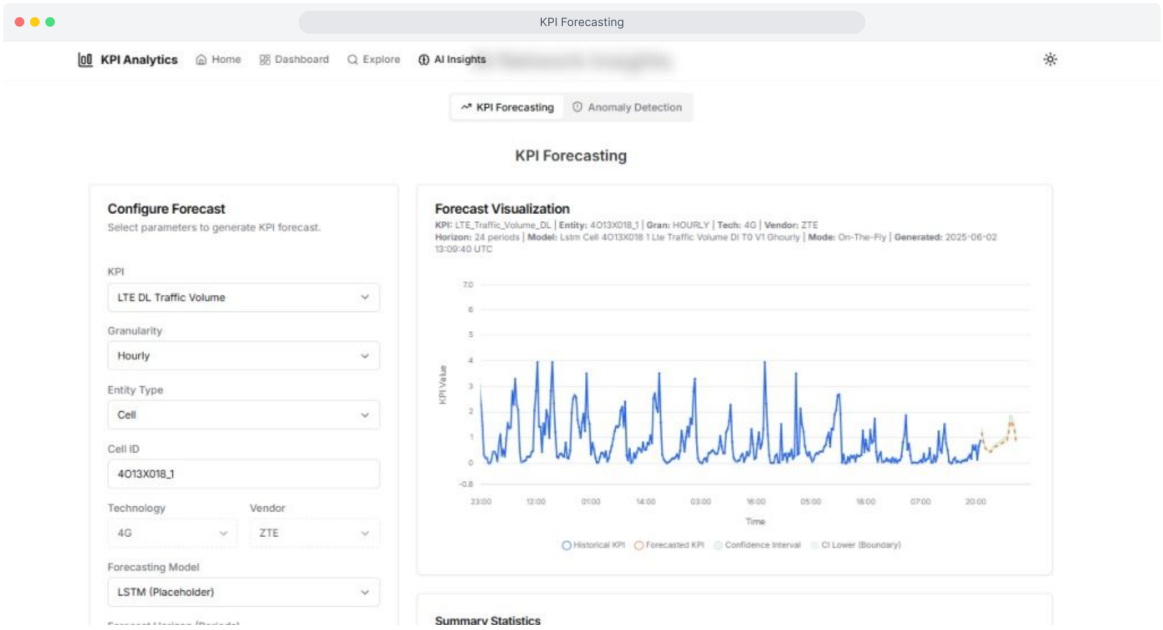


Figure 5.17: 24-period forecast chart.

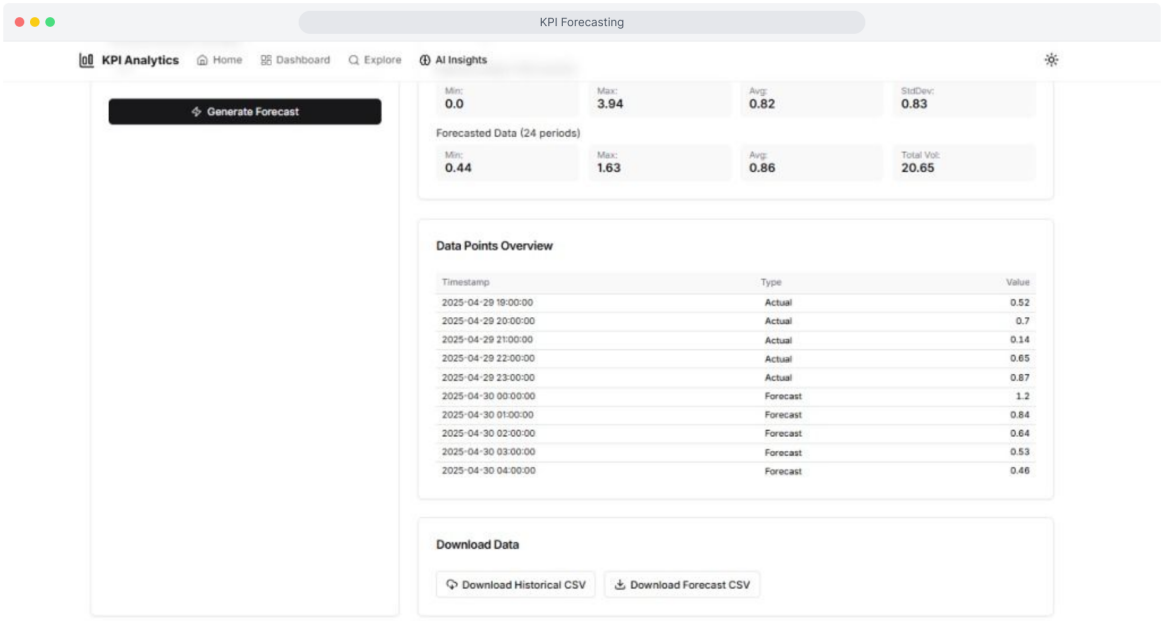


Figure 5.18: Data overview: actual vs. forecasted values and CSV download buttons.

In summary, the forecasting interface is intuitive and highly usable. It effectively combines form-based configuration, interactive visualizations, contextual metadata, and exportable data—all valuable for telecom engineers using AI-powered network forecasting.

Conclusion

In summary, this chapter evaluated the performance of the developed forecasting models through both quantitative metrics and visual inspection. The LSTM model showed promising results on smoother KPIs, while Prophet proved less effective for capturing complex temporal patterns. These evaluations confirmed the strengths and current limitations of our models, setting the stage for a more critical discussion of their implications, challenges, and optimization potential in the next chapter.

Chapter 6

Discussion of Results and Optimization Implications

This chapter provides an in-depth discussion and interpretation of the experimental results presented in Chapter 5 concerning the performance of Prophet and Long Short-Term Memory (LSTM) models for forecasting hourly Key Performance Indicators (KPIs) in an LTE network. The findings are analyzed in the context of the initial research objectives, existing literature, and the inherent characteristics of the telecommunications data. Furthermore, the practical implications of the developed forecasting capabilities for LTE network optimization are explored, alongside a frank assessment of the challenges encountered during the project and the limitations of the current system.

6.1 Implications for LTE Network Optimization

The high-accuracy forecasting capabilities developed in this thesis, particularly the strong performance of the tuned LSTM models, do not constitute an optimization system in themselves. Rather, they provide the essential, high-quality predictive intelligence that serves as a **foundational input** for a wide range of network optimization activities. By enabling a shift from reactive to proactive management, these forecasts have several direct and significant implications for enhancing LTE network operations at an operator like Djezzy.

6.1.1 Enhanced Proactive Resource Management

Accurate hourly forecasts of `LTE_Traffic_Volume_DL` and `DL_PRB_usage` are critical enablers for more intelligent resource management strategies:

- **Informed Capacity Planning:** With a model that explains 85% of traffic variance ($R^2 \approx 0.85$), a network operator like Djezzy can transition from a reactive to a **proactive capacity management strategy**. Engineers can now reliably identify cells that will face congestion during peak hours *before* it happens, allowing for preemptive load balancing or timely planning of hardware upgrades based on data-driven growth forecasts rather than past failures.
- **Foundation for Energy Saving Features:** Many modern network energy-saving techniques, such as putting cells into a low-power "sleep mode," depend on accurate traffic predictions. Our forecasts provide the necessary input to determine precisely when a cell can be safely powered down during predicted periods of low activity (e.g., late at night) without impacting user experience, creating a direct path to reduced operational expenditure (OPEX).

6.1.2 Improved Operational Efficiency

The predictive insights generated by the system can streamline daily network operations and improve overall efficiency:

- **Optimized Maintenance Scheduling:** With a forecast accuracy explaining 85% of the variance for traffic volume, operators can confidently schedule maintenance windows. The forecasting dashboard allows engineers to schedule planned activities that require taking a cell offline during periods that are reliably predicted to have the lowest traffic, thereby minimizing subscriber impact and potentially reducing overtime costs for engineering teams.
- **Dynamic Baselines for Anomaly Detection:** Perhaps one of the most significant implications is the ability to create dynamic, intelligent baselines for anomaly detection. Traditional systems often rely on static thresholds, leading to false alarms. The forecasts from our models, especially the breakthrough success in predicting the volatile `LTE_Thro_DL` KPI ($R^2 \approx 0.48$), can serve as a powerful dynamic baseline. A significant deviation of the *actual* KPI from its *forecasted* value is a much stronger indicator of a genuine anomaly, allowing for the detection of "silent failures" or subtle performance degradations that static thresholds would miss.

6.1.3 Data-Driven Input for Higher-Level Optimization Algorithms

While this thesis stops short of implementing a closed-loop optimization system, the forecasts it generates are the primary input required for such advanced algorithms. This work lays the groundwork for future projects in:

- **Reinforcement Learning (RL) for Parameter Tuning:** Future RL agents could use our forecasts of network state (e.g., predicted load) to learn optimal policies for dynamically adjusting network parameters, such as handover margins or antenna tilts, in real-time.
- **Automated Root Cause Analysis (RCA):** An advanced RCA system could correlate a deviation from a forecast with other network events (e.g., alarms, configuration changes) that occurred just prior, helping to automatically identify the most likely root cause of a performance degradation.

The current AI Insights Dashboard, as implemented in the Flask application, serves as the first step in this direction by making these predictive insights accessible to human operators, who can then use them to inform their own optimization decisions.

6.2 Challenges Encountered During System Development

The development of this analytics system faced several challenges:

- **Data Quality and Integration:** Ensuring consistent KPI definitions and handling data quality issues from multi-vendor environments in the DWH (Chapter 3) was critical.
- **Evaluation Pipeline Integrity:** As highlighted (Section 4.4.2), ensuring consistent preprocessing for LSTMs between training and testing was a significant initial hurdle.
- **Iterative Feature Engineering for LSTMs:** Finding an effective LSTM feature set required considerable experimentation, with simplification proving more effective initially.

- **Computational Cost of Hyperparameter Optimization:** Systematic tuning for LSTMs with Optuna is computationally intensive.
- **Forecasting Highly Volatile KPIs:** Accurately forecasting `LTE_Thro_DL` with high R^2 values was challenging with the current univariate approach.
- **Flask AI Service Layer Implementation:** Integrating AI models into a responsive Flask backend involved careful management of model loading, preprocessing, and result serialization.

6.3 Limitations of the Current Forecasting System

The developed system has current limitations:

- **Univariate Forecasting Core:** Models do not explicitly model dynamic interdependencies between different KPIs.
- **Single-Cell Focus for Tuning:** Detailed tuning and evaluation were for one cell; parameters may not generalize directly.
- **Limited Training Data Horizon:** Three months of hourly data may not capture longer-term trends or rare events.
- **Static Feature Set Post-Tuning:** Potential benefits of re-introducing more complex features after initial Optuna tuning were not fully explored.
- **Evaluation on a Single Hold-Out Period:** Performance could vary on different future periods.
- **Batch Training and Model Updates:** Models do not currently support online learning for rapid adaptation.
- **Interpretability of LSTM Models:** LSTM internal workings are less transparent ("black-box") without specialized interpretability techniques.
- **Anomaly Detection as Future Work:** A dedicated anomaly detection module is future work, though forecasts can serve as baselines.

Acknowledging these limitations is crucial for contextualizing achievements and guiding future development.

Conclusion

In this chapter, we analyzed the performance results of the forecasting models in the context of network optimization. We examined KPI-specific forecasting accuracy, compared the LSTM and Prophet models, and highlighted the role of hyperparameter tuning and residual analysis. These insights were connected to real-world LTE optimization use cases such as proactive resource allocation and improved operational efficiency. Limitations and development challenges were also discussed.

The next and final chapter will conclude the thesis by summarizing key findings, reflecting on project contributions, and outlining potential directions for future work.

Conclusion and Future Work

This thesis has documented the comprehensive design, development, and evaluation of an integrated analytics framework aimed at enhancing Long-Term Evolution (LTE) network performance management through data-driven insights and Artificial Intelligence (AI). The project successfully addressed the challenge of transforming vast, often heterogeneous, network data into actionable intelligence by establishing a robust Data Warehouse (DWH), implementing sophisticated ETL processes, and developing AI-powered Key Performance Indicator (KPI) forecasting models. The culmination of these efforts is a functional system with a backend service layer capable of delivering these predictive insights, laying a crucial foundation for proactive and intelligent network optimization.

Summary of Key Contributions and Achievements

The primary contributions of this thesis span data engineering, AI model development, and system integration, resulting in a significant step towards more advanced LTE network operations:

1. **Establishment of a Scalable Data Management Foundation (Chapter 3):**
    - A relational DWH was architected using PostgreSQL, employing dimensional modeling principles to effectively store and manage historical LTE performance data from potentially multi-vendor sources. This DWH acts as the single source of truth for all subsequent analytics.
    - Automated ETL pipelines were engineered using Python and SQL, orchestrated by Apache Airflow. These pipelines handle the ingestion of raw network counter data, perform crucial transformations including KPI normalization and cleansing, and populate the DWH. This ensures data consistency and reliability for AI modeling.
    - A dynamic KPI calculation engine within the DWH allows for flexible definition and computation of network performance metrics at various aggregation levels and granularities.
2. **Development and Optimization of High-Performance AI-Driven Forecasting Models (Chapters 4 & 5):**
    - This research successfully developed and rigorously evaluated Prophet and Long Short-Term Memory (LSTM) network models for forecasting critical hourly LTE KPIs.
    - A systematic and iterative methodology was employed, culminating in a **highly effective, Optuna-tuned LSTM model**. This model demonstrated a significant breakthrough by achieving high predictive accuracy across all target KPIs, including the notoriously volatile `LTE_Thro_DL` (R^2 of 0.48).

- The LSTM model achieved exceptional performance on KPIs with strong seasonal patterns, such as `LTE_Traffic_Volume_DL` (R^2 of 0.85) and `DL_PRB_usage` (R^2 of 0.66), decisively outperforming all baseline and statistical models.
- The success of the models validates the entire data engineering and AI development pipeline, from EDA-informed feature engineering to the critical role of automated hyperparameter optimization.

3. **Implementation of an AI Service Delivery Backend (Chapter 4):**
   - A Flask-based backend application was developed to serve the trained forecasting models. This service layer handles on-the-fly prediction requests, orchestrating data fetching from the DWH, necessary preprocessing (consistent with training), model inference, and post-processing of results.
   - The "AI Insights Dashboard" UI provides forms for forecast configuration and displays the generated predictions and summary statistics, demonstrating the practical application of the AI models.
4. **Establishment of a Rigorous Evaluation Framework (Chapter 5):**
   - A clear methodology for training data preparation, hold-out test set evaluation, and the use of standard performance metrics (RMSE, MAE, MAPE, SMAPE, R^2, Bias) was established, ensuring robust and comparable assessment of model performance.

Achievement of Project Objectives

The project successfully met its core objectives as outlined in the introduction:

- **Design and implement a scalable Data Warehouse:** The PostgreSQL DWH and associated ETL pipelines (Chapter 3) fulfill this objective, providing a structured and reliable data foundation.
- **Develop robust ETL processes:** Automated scripts for data ingestion, transformation, and KPI calculation were successfully implemented and are integral to the DWH population.
- **Implement AI models for KPI forecasting:** Prophet and, more significantly, optimized LSTM models were developed, trained, and evaluated, demonstrating tangible predictive capabilities, especially for traffic volume (Chapters 4 and 5).
- **Develop a backend service for AI insights:** The Flask application successfully serves the AI forecasting models, making their predictions accessible for potential integration into operational tools or more advanced visualization frontends (Chapter 4, Section 4.5).
- **Enable proactive network management through predictive insights:** The forecasting capabilities developed provide the foundational intelligence needed to anticipate network behavior, thereby enabling a shift towards more proactive network management strategies as discussed in Chapter 6, Section 6.1.

While the initial broader vision included a fully implemented anomaly detection module and a comprehensive, independent visualization platform, this thesis has successfully delivered the crucial DWH backbone and the AI forecasting service layer. The groundwork for anomaly detection has been laid, and it remains a primary direction for future work.

Overall Conclusion

This thesis has successfully demonstrated the design, implementation, and validation of a comprehensive, data-driven framework capable of delivering high-accuracy forecasts for LTE network performance. By integrating a robust Data Warehouse, automated ETL processes, and a systematically optimized AI pipeline, this project has moved beyond theoretical application to produce tangible, high-performance predictive models.

The key achievement of this work lies in the demonstrably superior performance of the Optuna-tuned LSTM models. The exceptional results, particularly the successful forecasting of the highly volatile user throughput KPI (R^2 of 0.48) and the near-complete explanation of variance for traffic volume (R^2 of 0.85), validate our entire methodological approach. It underscores that with rigorous data management, deep exploratory analysis, and a commitment to automated, systematic hyperparameter optimization, deep learning models can overcome the inherent complexities of telecommunications data to provide powerful predictive insights.

The engineered solution, culminating in a Flask-based service layer, effectively bridges the gap between model development and operational utility. It provides a foundational platform for a new generation of proactive network management strategies, proving that a well-executed machine learning workflow can yield significant and reliable results, paving the way for more intelligent, efficient, and optimized mobile networks.

Recommendations for Future Work and System Evolution

The platform and AI modules developed in this thesis provide a strong foundation for numerous exciting future enhancements and research directions:

1. **Full Implementation and Integration of Anomaly Detection Module:**
    - *Model Development & Training:* Develop and train the `ProphetAnomalyDetector` and `VAEAnomalyDetector` (from `app.core_ai.anomaly_detection.models`) using the DWH. This includes establishing methods for defining "normal" behavior and setting appropriate detection thresholds.
    - *Service Layer Extension:* Extend the Flask backend in `app.ai_insights.services` to include functions for on-demand or scheduled anomaly detection runs.
    - *Web Application Integration:* Fully develop the "Anomaly Detection" tab in the `unified_insights_dashboard.html`, including forms, visualizations of detected anomalies, and a detailed anomaly table.
    - *Feedback Loop for Model Refinement:* Use confirmed true/false positives from engineer feedback to periodically retrain and refine anomaly detection models.
2. **Advanced Forecasting Model Enhancements:**
    - *Multivariate LSTMs for Throughput:* Systematically incorporate lagged exogenous variables (e.g., CQI, PRB utilization, as identified in EDA and suggested by literature like [51]) into LSTM models for `LTE_Thro_DL`.
    - *Attention Mechanisms and Transformers:* Explore attention mechanisms within LSTMs or implement Transformer-based models [52] for KPI forecasting, as investigated in works like [53].
    - *GRU Models:* Evaluate Gated Recurrent Units (GRUs) as a potentially more computationally efficient alternative to LSTMs.

- *Hybrid Models:* Investigate hybrid approaches, such as Prophet-LSTM or statistical models combined with ML error correction.

3. **Broader Platform and Operational Enhancements:**
   - *Spatial-Temporal Forecasting:* Extend models to consider spatial dependencies between cells (e.g., using Graph Neural Networks - GNNs).
   - *Root Cause Analysis (RCA) for Anomalies:* Develop ML models to assist in identifying probable root causes of anomalies by correlating them with network alarms or configuration changes.
   - *Real-Time Data Ingestion and Stream Processing:* Evolve ETL pipelines to support near real-time data ingestion (e.g., using Apache Kafka, Flink).
   - *Scalability and Deployment:* Explore containerization (Docker, Kubernetes) for production deployment.
   - *Comprehensive Visualization Frontend:* Develop a dedicated, rich interactive visualization frontend that consumes insights from the DWH and AI service layer.
   - *Closed-Loop Optimization:* Integrate predictive capabilities into closed-loop optimization systems (Self-Organizing Networks - SON).
   - **Expanded Model Benchmarking:** To provide an even more robust evaluation of model performance, future work could include comparisons against classical statistical models like SARIMA. Furthermore, tree-based models such as LightGBM or XGBoost, which are often highly effective for time-series forecasting when structured with lag and calendar features, should be benchmarked to create a comprehensive performance landscape.

4. **Expanded Model Benchmarking:** To provide an even more robust evaluation of model performance, future work could include comparisons against classical statistical models like SARIMA. Furthermore, tree-based models such as LightGBM or XGBoost, which are often highly effective for time-series forecasting when structured with lag and calendar features, should be benchmarked to create a comprehensive performance landscape.

Path to Production and Scalability

While this project was developed and tested in a local environment, the architecture was designed with production-readiness in mind. Transitioning this framework from a successful prototype to a fully scalable, production-grade system would require several key engineering steps. This path to production represents a critical area for future work.

1. **Containerization and Orchestration:** Fully containerize the application components (Airflow, PostgreSQL, Flask App) using Docker and manage them with a container orchestration platform like Kubernetes. This would provide automated scaling, fault tolerance, and environment consistency.

2. **Database Migration:** Migrate the PostgreSQL database to a managed, production-grade service (e.g., Amazon RDS, Google Cloud SQL, or a self-hosted, high-availability cluster). This ensures automated backups, read replicas to handle heavy query loads from multiple services, and professional maintenance.

3. **Distributed ETL Processing:** For a national-scale deployment involving thousands of cells, the current Python and SQL-based ETL scripts would become a bottleneck. Re-engineering these tasks to run on a distributed processing framework like Apache Spark would be essential to handle the massive increase in data volume efficiently.

4. **Dedicated Model Serving and CI/CD:** Instead of serving models directly from the Flask application, a dedicated model serving solution (e.g., TensorFlow Serving, Seldon Core, or BentoML) should be implemented. This would be integrated into a CI/CD (Continuous Integration/Continuous Deployment) pipeline for automated model retraining, validation, and deployment, ensuring the system stays up-to-date with the latest data patterns.

By pursuing these avenues, the developed analytics framework can evolve into an increasingly powerful and indispensable tool for the intelligent management and optimization of current and future mobile communication networks, delivering tangible benefits in terms of performance, efficiency, and user satisfaction.

Appendix A

System Orchestration and Environment Setup

A key contribution of this project is the creation of a fully automated and reproducible data pipeline. This is achieved through two core components: a workflow orchestration engine using **Apache Airflow** and a containerized environment defined with **Docker Compose**. This appendix outlines this technical architecture, highlighting how the system ensures reliability and scalability.

A.1 Workflow Orchestration with Apache Airflow

Apache Airflow is an open-source platform used to programmatically author, schedule, and monitor workflows. As described by the Apache Software Foundation, workflows are defined as **Directed Acyclic Graphs (DAGs)** of tasks. This paradigm allows for the clear definition of complex dependencies, ensuring that each step of a data pipeline executes only after its prerequisites have been successfully completed.

For this project, Airflow was chosen to serve as the central orchestrator for the entire data processing lifecycle. It automates the daily execution of all tasks, from data extraction and ETL to the final triggering of the AI forecasting models. This approach transforms a series of individual scripts into a single, cohesive, and robust automated system. The structure of our primary pipeline is visualized in Figure A.1.

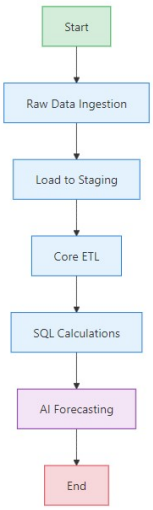


Figure A.1: Logical Flowchart of the Airflow DAG

The primary execution steps defined in the DAG are:

1. **Raw Data Ingestion:** A task group that fetches vendor-specific files, standardizes their format, and places them in a processing area.
2. **Load to Staging:** A task that bulk-loads the standardized files into the `Staging_RawCounter` table in the Data Warehouse.
3. **Core ETL:** The main transformation step that validates, enriches, and moves data from the staging table into the final, clean fact tables.
4. **SQL Calculations:** A task group that executes a sequence of '.sql' scripts against the DWH to compute daily aggregates and all levels of KPIs.
5. **AI Forecasting:** The final stage, which triggers the Python forecasting script to generate new predictions based on the freshly processed data.

A.2 Reproducible Development Environment with Docker Compose

To ensure that the entire system is self-contained and easily reproducible, we use **Docker Compose** to define and manage all the required services. This isolates dependencies and allows any developer to spin up the complete environment with a single command. A key architectural choice is the use of **two separate PostgreSQL instances**: one dedicated to the Airflow metadata backend and another serving as the project's primary Data Warehouse. This separation is a best practice that prevents the operational load of the application from impacting the performance and stability of the orchestrator.

The essential services defined in the 'docker-compose.yml' file are:

`postgres` An official PostgreSQL container that serves as the metadata backend for Apache Airflow itself. It stores the state of DAG runs, task instances, and connections.

`datawarehouse` A second, independent PostgreSQL container that functions as our project's Data Warehouse. It is initialized with our custom schema ('dbcreation.sql') and is exposed on the host machine for direct access and analysis.

`airflow-scheduler` The core Airflow component that monitors all DAGs and triggers task instances once their dependencies are met.

`airflow-webserver` The Airflow user interface, providing a visual dashboard to monitor DAG runs, inspect logs, and manage the system.

`airflow-init` A one-time initialization service that prepares the Airflow database and sets up the necessary connections, such as the one pointing to our 'datawarehouse' service.

This containerized approach ensures a consistent environment across different machines and provides a clear blueprint for a future production deployment.

```
1 x-airflow-common:
2   &airflow-common
3   # In order to add custom dependencies or upgrade provider packages you can
      use your extended image.
4   # Comment the image line, place your Dockerfile in the directory where you
      placed the docker-compose.yaml
```

```
  # and uncomment the "build" line below, Then run `docker-compose build` to build the images.
  #image: ${AIRFLOW_IMAGE_NAME:-apache/airflow:2.10.5}
  build: .
  environment:
    &airflow-common-env
    AIRFLOW__CORE__EXECUTOR: LocalExecutor
    AIRFLOW__DATABASE__SQL_ALCHEMY_CONN: postgresql+psycopg2://airflow:airflow@postgres/airflow
    AIRFLOW__CORE__FERNET_KEY: ${FERNET_KEY:-""}
    AIRFLOW__CORE__DAGS_ARE_PAUSED_AT_CREATION: 'true'
    AIRFLOW__CORE__LOAD_EXAMPLES: 'false'
    AIRFLOW__API__AUTH_BACKENDS: 'airflow.api.auth.backend.basic_auth,airflow.api.auth.backend.session'
    AIRFLOW_CONN_DATAWAREHOUSE: postgres://${DB_USER}:${DB_PASSWORD}@${DB_HOST}:${DB_PORT}/${DB_NAME}    # yamllint disable rule:line-length
    # Use simple http server on scheduler for health checks
    # See https://airflow.apache.org/docs/apache-airflow/stable/administration-and-deployment/logging-monitoring/check-health.html#scheduler-health-check-server
    # yamllint enable rule:line-length
    AIRFLOW__SCHEDULER__ENABLE_HEALTH_CHECK: 'true'
    # Adding performance tuning parameters
    #AIRFLOW__SCHEDULER__MIN_FILE_PROCESS_INTERVAL: 60
    #AIRFLOW__CELERY__WORKER_CONCURRENCY: 8
    #AIRFLOW__CORE__PARALLELISM: 16
    #AIRFLOW__CORE__DAG_CONCURRENCY: 8
    DB_NAME: ${DB_NAME}
    DB_USER: ${DB_USER}
    DB_PASSWORD: ${DB_PASSWORD}
    DB_HOST: ${DB_HOST}
    DB_PORT: ${DB_PORT}
    # WARNING: Use _PIP_ADDITIONAL_REQUIREMENTS option ONLY for a quick checks
    # for other purpose (development, test and especially production usage) build/extend Airflow image.
    #_PIP_ADDITIONAL_REQUIREMENTS: ${}
    # The following line can be used to set a custom config file, stored in the local config folder
    # If you want to use it, outcomment it and replace airflow.cfg with the name of your config file
    # AIRFLOW_CONFIG: '/opt/airflow/config/airflow.cfg'
  volumes:
    - ${AIRFLOW_PROJ_DIR:-.}/config:/opt/airflow/config
    - ${AIRFLOW_PROJ_DIR:-.}/dags:/opt/airflow/dags
    - ${AIRFLOW_PROJ_DIR:-.}/logs:/opt/airflow/logs
    - ${AIRFLOW_PROJ_DIR:-.}/plugins:/opt/airflow/plugins
    - ${AIRFLOW_PROJ_DIR:-.}/Def:/opt/airflow/Def
    - ${AIRFLOW_PROJ_DIR:-.}/Fact:/opt/airflow/Fact
    - ${AIRFLOW_PROJ_DIR:-.}/Dim:/opt/airflow/Dim
  user: "${AIRFLOW_UID:-50000}:0"
  depends_on:
    &airflow-common-depends-on
    postgres:
      condition: service_healthy
    datawarehouse:
      condition: service_healthy

services:
  postgres:
```

```
    image: postgres:13
    environment:
      POSTGRES_USER: airflow
      POSTGRES_PASSWORD: airflow
      POSTGRES_DB: airflow
    ports :
      - "5432:5432"
    volumes:
      - postgres-db-volume:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD", "pg_isready", "-U", "airflow"]
      interval: 10s
      retries: 5
      start_period: 5s
    restart: always

  datawarehouse:
    image: postgres:13
    environment:
      POSTGRES_USER: ${DB_USER}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_DB: ${DB_NAME}
    volumes:
      - datawarehouse-volume:/var/lib/postgresql/data
      - ./dbcreation.sql:/docker-entrypoint-initdb.d/dbcreation.sql
    ports:
      - 5433:5432
    healthcheck:
      test: ["CMD", "pg_isready", "-U", "${DB_USER}"]
      interval: 5s
      retries: 5
    restart: always
    # Adding resource limits
    #deploy:
    #  resources:
    #    limits:
    #      cpus: '1'
    #      memory: 1G


  airflow-webserver:
    <<: *airflow-common
    command: webserver
    ports:
      - "8080:8080"
    healthcheck:
      test: ["CMD", "curl", "--fail", "http://localhost:8080/health"]
      interval: 30s
      timeout: 10s
      retries: 5
      start_period: 30s
    restart: always
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully
    # Adding resource limits
    #deploy:
    #  resources:
    #    limits:
```

```
    #      cpus: '1'
    #      memory: 1G

  airflow-scheduler:
    <<: *airflow-common
    command: scheduler
    healthcheck:
      test: ["CMD", "curl", "--fail", "http://localhost:8974/health"]
      interval: 30s
      timeout: 10s
      retries: 5
      start_period: 30s
    restart: always
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully
            # Adding resource limits
    #deploy:
    #  resources:
    #    limits:
    #      cpus: '1'
    #      memory: 1G


  airflow-triggerer:
    <<: *airflow-common
    command: triggerer
    healthcheck:
      test: ["CMD-SHELL", 'airflow jobs check --job-type TriggererJob --
   hostname "$${HOSTNAME}"']
      interval: 30s
      timeout: 10s
      retries: 5
      start_period: 30s
    restart: always
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully
            # Adding resource limits
    #deploy:
    #  resources:
    #    limits:
    #      cpus: '1'
    #      memory: 1G

  airflow-init:
    <<: *airflow-common
    entrypoint: /bin/bash
    # yamllint disable rule:line-length
    command:
      - -c
      - |
        if [[ -z "${AIRFLOW_UID}" ]]; then
          echo
          echo -e "\033[1;33mWARNING!!!: AIRFLOW_UID not set!\e[0m"
          echo "If you are on Linux, you SHOULD follow the instructions
   below to set "
          echo "AIRFLOW_UID environment variable, otherwise files will be
```

```
    owned by root."
173         echo "For other operating systems you can get rid of the warning
    with manually created .env file:"
174         echo "    See: https://airflow.apache.org/docs/apache-airflow/
    stable/howto/docker-compose/index.html#setting-the-right-airflow-user"
175         echo
176       fi
177       one_meg=1048576
178       mem_available=$$(($$(getconf _PHYS_PAGES) * $$(getconf PAGE_SIZE) /
    one_meg))
179       cpus_available=$$(grep -cE 'cpu[0-9]+' /proc/stat)
180       disk_available=$$(df / | tail -1 | awk '{print $$4}')
181       warning_resources="false"
182       if (( mem_available < 4000 )) ; then
183         echo
184         echo -e "\033[1;33mWARNING!!!: Not enough memory available for
    Docker.\e[0m"
185         echo "At least 4GB of memory required. You have $$(numfmt --to iec
     $$((mem_available * one_meg)))"
186         echo
187         warning_resources="true"
188       fi
189       if (( cpus_available < 2 )); then
190         echo
191         echo -e "\033[1;33mWARNING!!!: Not enough CPUS available for
    Docker.\e[0m"
192         echo "At least 2 CPUs recommended. You have $${cpus_available}"
193         echo
194         warning_resources="true"
195       fi
196       if (( disk_available < one_meg * 10 )); then
197         echo
198         echo -e "\033[1;33mWARNING!!!: Not enough Disk space available for
     Docker.\e[0m"
199         echo "At least 10 GBs recommended. You have $$(numfmt --to iec $$
    ((disk_available * 1024 )))"
200         echo
201         warning_resources="true"
202       fi
203       if [[ $${warning_resources} == "true" ]]; then
204         echo
205         echo -e "\033[1;33mWARNING!!!: You have not enough resources to
    run Airflow (see above)!\e[0m"
206         echo "Please follow the instructions to increase amount of
    resources available:"
207         echo "   https://airflow.apache.org/docs/apache-airflow/stable/
    howto/docker-compose/index.html#before-you-begin"
208         echo
209       fi
210       mkdir -p /sources/logs /sources/dags /sources/plugins
211       chown -R "${AIRFLOW_UID}:0" /sources/{logs,dags,plugins}
212
213       airflow db migrate
214
215       until pg_isready -h datawarehouse -p 5432 -U ${DB_USER}; do
216         echo "Waiting for PostgreSQL to be ready..."
217         sleep 5
218       done
219
220       echo "Creating datawarehouse connection..."
```

```
        # Check if connection already exists
        if ! airflow connections get datawarehouse > /dev/null 2>&1; then
          airflow connections add datawarehouse \
            --conn-type postgres \
            --conn-host "${DB_HOST}" \
            --conn-schema "${DB_NAME}"  \
            --conn-login "${DB_USER}"  \
            --conn-password "${DB_PASSWORD}" \
            --conn-port "${DB_PORT}"
          echo "Connection created."
        else
          echo "Connection already exists."
        fi

        echo "Checking connection..."
        airflow connections get datawarehouse

        exec /entrypoint airflow version

    # yamllint enable rule:line-length
    environment:
      <<: *airflow-common-env
      _AIRFLOW_DB_MIGRATE: 'true'
      _AIRFLOW_WWW_USER_CREATE: 'true'
      _AIRFLOW_WWW_USER_USERNAME: ${_AIRFLOW_WWW_USER_USERNAME:-airflow}
      _AIRFLOW_WWW_USER_PASSWORD: ${_AIRFLOW_WWW_USER_PASSWORD:-airflow}
      _PIP_ADDITIONAL_REQUIREMENTS: ''
    user: "0:0"
    volumes:
      - ${AIRFLOW_PROJ_DIR:-.}:/sources

  airflow-cli:
    <<: *airflow-common
    profiles:
      - debug
    environment:
      <<: *airflow-common-env
      CONNECTION_CHECK_MAX_COUNT: "0"
    # Workaround for entrypoint issue. See: https://github.com/apache/
    airflow/issues/16252
    command:
      - bash
      - -c
      - airflow

  # You can enable flower by adding "--profile flower" option e.g. docker-
    compose --profile flower up
  # or by explicitly targeted on the command line e.g. docker-compose up
    flower.
  # See: https://docs.docker.com/compose/profiles/

volumes:
  postgres-db-volume:
  datawarehouse-volume:
```

Listing A.1: Docker Compose config file contents.

Appendix B

Metadata and Configuration Files

A core design principle of this project is its **metadata-driven architecture**. Instead of hard-coding business logic, KPI formulas, or model parameters, the system is controlled by external configuration files. This approach makes the framework highly flexible, maintainable, and adaptable to new vendors, KPIs, or modeling requirements without changing the core source code.

This appendix provides key samples of these configuration files, demonstrating how they govern the behavior of both the ETL and AI forecasting pipelines.

B.1 ETL and Data Warehouse Configuration

The ETL processes rely on a set of CSV and YAML files to understand the structure of incoming data and the definitions of the KPIs to be calculated.

B.1.1 KPI and Counter Definitions

The foundation of the DWH's business logic resides in two definition files. 'counter_definitions.csv' maps raw, vendor-specific counter names to a standardized internal ID. 'kpi_formula.csv' then uses these standardized IDs to define the mathematical expressions for calculating final KPIs.

Table B.1: Sample from 'counter_definitions.csv', linking raw counters to standardized IDs.

counter_id	counter_name	tech_id	vendor_id	kpi_id
L.Thrp.bits.DL	L.Thrp.bits.DL	0	0	LTE_Traffic_Volume_DL
C373230691	DL PRB Usage Rate	0	1	DL_PRB_usage
M8012C20	PDCP_SDU_VOL_DL	0	2	LTE_Traffic_Volume

Table B.2: Sample from 'kpi_formula.csv', defining KPI calculations.

formula_id	kpi_id	vendor_id	formula_expression
F_HW_Traffic_DL	LTE_Traffic_Volume_DL	0	`((((([L.Thrp.bits.DL]/1024)/1024)/1024)/8)`
F_ZTE_Traffic_DL	LTE_Traffic_Volume_DL	1	`(([C373343806]*1000+[C373343807])/8e6)`

B.1.2 Source Data Mapping Configuration

To handle the heterogeneity of data from different vendors, a central 'source_mappings.yml' file is used. This file instructs the data loading scripts on how to parse each vendor's files, specifying

everything from the filename pattern to the column names for key fields like timestamps and cell identifiers. This declarative approach allows new data sources to be added simply by creating a new entry in this YAML file.

```yaml
# source_mappings.yml

huawei_4g:
  vendor: "HUAWEI"
  technology: "4G"
  filename_pattern: "huawei_4g_*.csv"
  timestamp_column: "Time"
  timestamp_format: "%Y-%m-%d %H:%M:%S"
  cell_id_column: "Cell Name"
  counters:
    L.CSFB.PrepAtt: "L.CSFB.PrepAtt"
    L.Thrp.bits.DL: "L.Thrp.bits.DL"
    # ... more counter mappings

zte_4g:
  vendor: "ZTE"
  technology: "4G"
  filename_pattern: "zte_4g_hourly_*.csv"
  timestamp_column: "Time"
  timestamp_format: "%Y-%m-%d %H:%M:%S"
  cell_id_column: "Cell Name"
  counters:
    C373343806: "C373343806"
    C373343807: "C373343807"
    # ... more counter mappings

nokia_4g:
  vendor: "NOKIA"
  technology: "4G"
  filename_pattern: "nokia_4g_*.csv" # Example pattern
  timestamp_column: "Time"
  timestamp_format: "%d/%m/%Y %H:%M"
  cell_id_column: "Cell Name"
  counters:
    M8011C12: "M8011C12"
    M8011C13: "M8011C13"
    # ... more counter mappings
```

Listing B.1: Sample from the ‘source_mappings.yml‘ configuration file, defining parsing rules for different vendors.

B.2 AI Forecasting Module Configuration

Similarly, the entire AI forecasting pipeline is governed by a centralized YAML configuration file, ‘forecasting_config.yml‘. This allows for the easy modification of model parameters, feature sets, and training routines without altering the core Python code. A key feature is the ability to define KPI-specific overrides, allowing us to use the best hyperparameters discovered by Optuna for each specific KPI.

```yaml

# app/core_ai/configs/forecasting_config.yml

# General settings applicable to the entire forecasting pipeline
general:
```

```
  log_level: INFO
  model_store_base_path: "app/core_ai/model_store"
  seed: 42

# Defines the scope of data used for training and evaluation
data_source:
  training_history_days: 90
  evaluation_test_period_days: 30
  entities_to_train:
    - entity_type: "CELL"
      entity_id: "4O13X018_1"
      kpis: ["LTE_Thro_DL", "LTE_Traffic_Volume_DL", "DL_PRB_usage"]
      granularities: ["HOURLY"]

# Default parameters for LSTM models, can be overridden per KPI
lstm_forecasting:
  min_obs_for_training: 150
  preprocessing:
    scaling_method: "minmax"
    # Features to be automatically joined from the DimTime table
    dim_time_features_to_join:
      - "day_of_week"
      - "hour_of_week_sin"
      - "hour_of_week_cos"
      - "is_weekend"
      - "is_holiday"
      # ... and others
  training_params:
    epochs: 100 # Max epochs for final training
    validation_split: 0.2

# ==================================================
# === KPI-Specific Configurations & Hyperparameters ===
# ==================================================
kpi_specific_configs:
  LTE_Thro_DL:
    description: "LTE Downlink User Throughput in Mbps"
    lstm:
      enabled: true
      apply_log_transform: true
      # --- Best Hyperparameters from Optuna ---
      time_steps: 72
      lstm_units: [32]
      dropout_rate: 0.4
      use_bidirectional: true
      learning_rate: 0.0049
      batch_size: 32

  LTE_Traffic_Volume_DL:
    description: "LTE Downlink Traffic Volume in GB"
    lstm:
      enabled: true
      apply_log_transform: true
      # --- Best Hyperparameters from Optuna ---
      time_steps: 48
      lstm_units: [48]
      dropout_rate: 0.3
      use_bidirectional: true
      learning_rate: 0.0012
      batch_size: 64
```

```
66
67  DL_PRB_usage:
68    description: "Downlink Physical Resource Block Usage Percentage"
69    lstm:
70      enabled: true
71      apply_log_transform: false
72      # --- Best Hyperparameters from Optuna ---
73      time_steps: 96
74      lstm_units: [48]
75      dropout_rate: 0.15
76      use_bidirectional: true
77      learning_rate: 0.0003
78      batch_size: 64
```

Listing B.2: Centralized configuration file for the AI forecasting module (`forecasting_config.yml`).

Appendix C

Supplementary Exploratory Data Analysis (EDA) Plots

This appendix provides the complete set of exploratory data analysis (EDA) plots for the Key Performance Indicators (KPIs) discussed in Chapter 4. These plots, generated from the training dataset, were foundational in understanding the data's characteristics—such as seasonality, distribution, and stationarity—and directly informed the feature engineering and model design choices for the AI forecasting models.

C.1 EDA for `LTE_Traffic_Volume_DL`

The following plots provide a comprehensive visual analysis of the hourly `LTE_Traffic_Volume_DL` KPI.

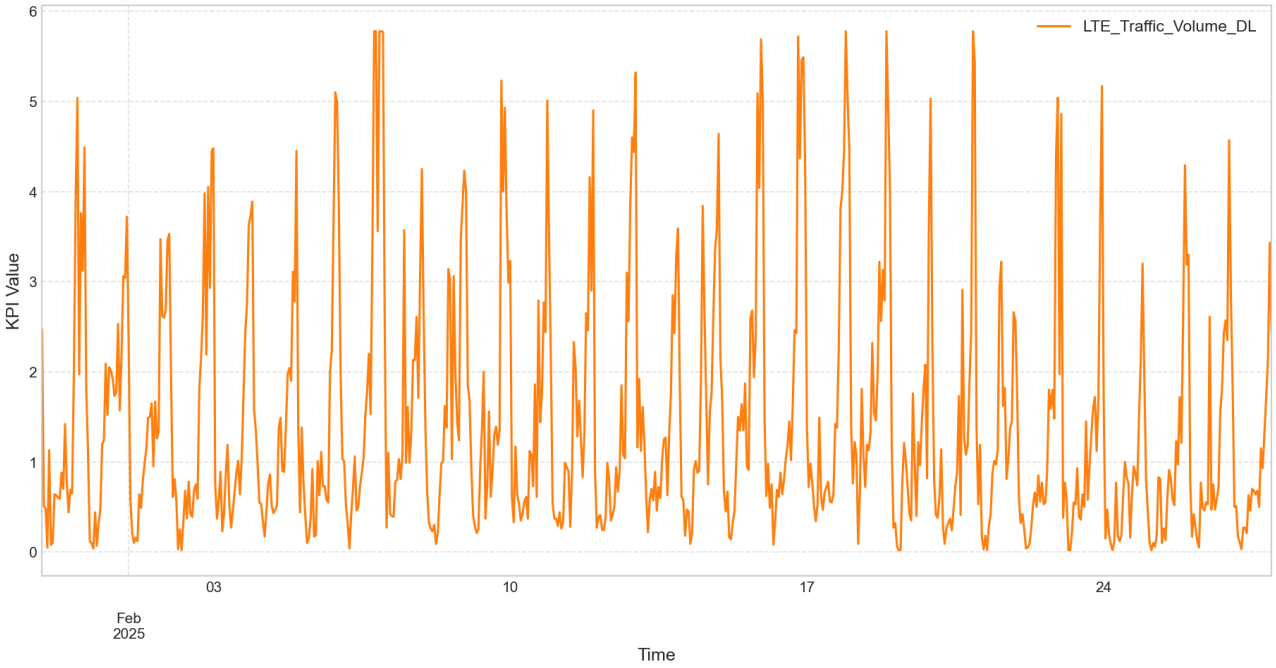
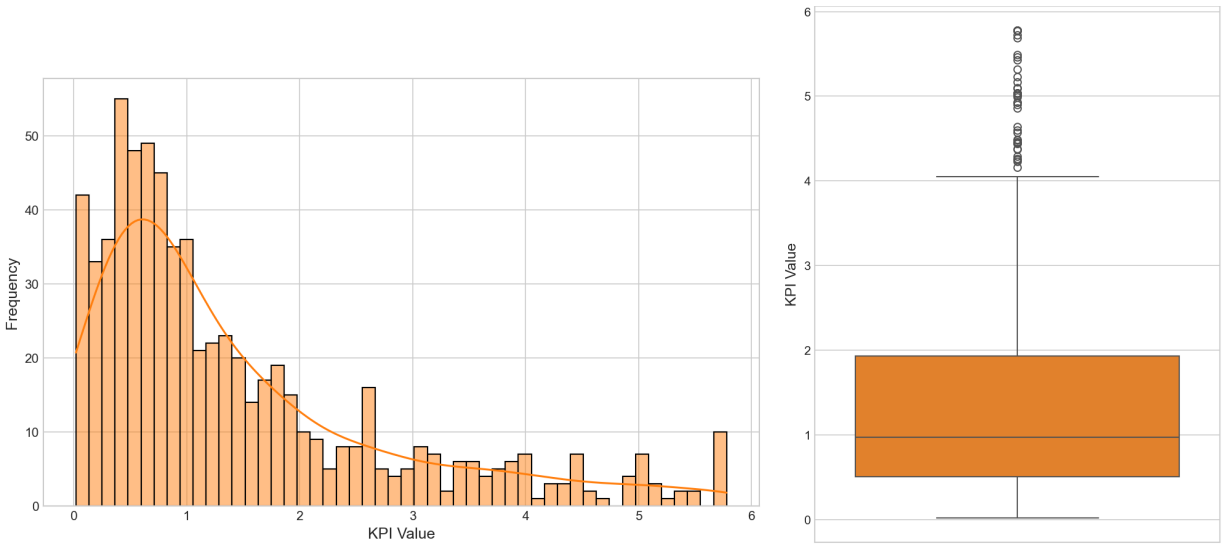


Figure C.1: Hourly Time Series of `LTE_Traffic_Volume_DL` (Training Data).

The time series plot (Figure C.1) showcases a very pronounced and regular daily pattern. Traffic volume consistently surges during evening peak hours and plummets during late-night/early-morning off-peak hours. No strong long-term linear trend is immediately visible over the training period.



(a) Distribution (Histogram and KDE). (b) Box Plot.

Figure C.2: Distribution and outlier analysis for hourly `LTE_Traffic_Volume_DL`.

The distribution (Figure C.2a) is significantly right-skewed, a characteristic confirmed by the mean (1.45 GB) being notably higher than the median (0.97 GB). This skewness indicated the necessity of a 'log1p' transformation for stabilizing variance during model training. The box plot (Figure C.2b) further illustrates the skew and highlights outliers representing periods of exceptionally high traffic.

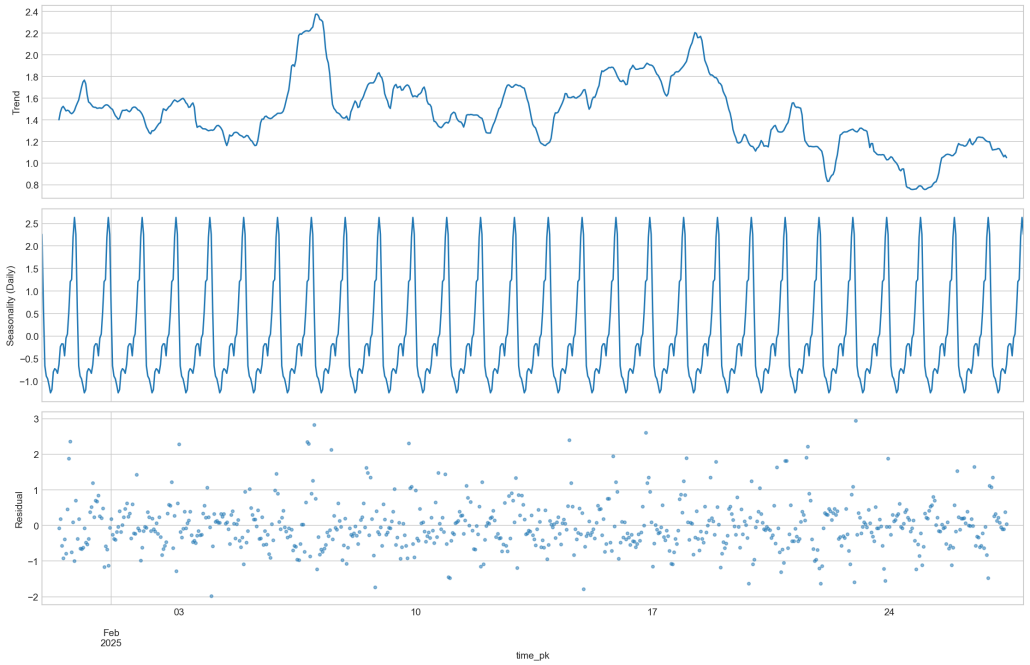


Figure C.3: Additive Seasonal Decomposition of `LTE_Traffic_Volume_DL`.

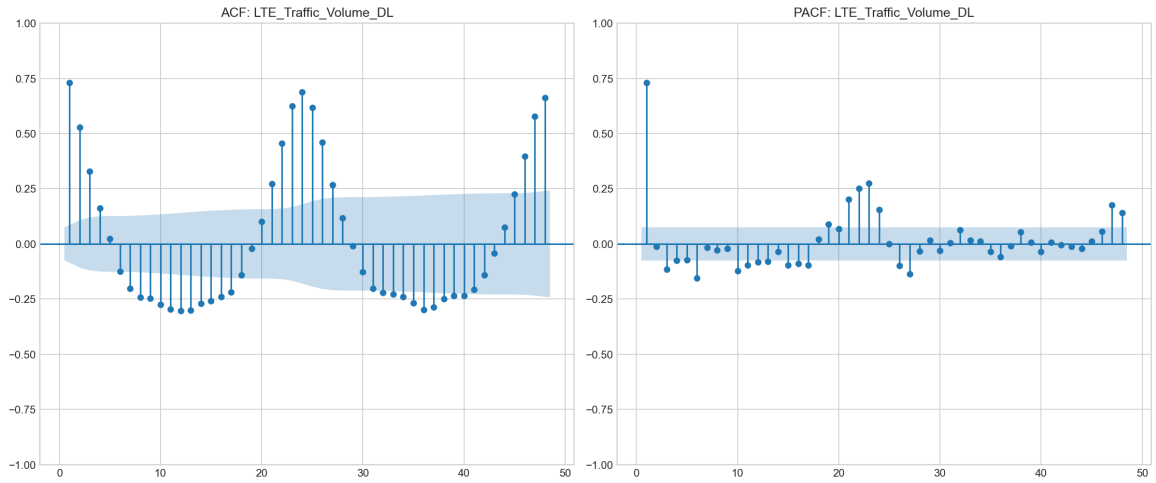


Figure C.4: ACF and PACF Plots for `LTE_Traffic_Volume_DL`.

Seasonal decomposition (Figure C.3) successfully isolates an extremely strong and regular daily seasonal component. The Autocorrelation (ACF) and Partial Autocorrelation (PACF) plots (Figure C.4) unequivocally confirm this dominant daily seasonality, with significant spikes at lags 24, 48, etc. This pattern informed the selection of seasonal lag features for the LSTM model.

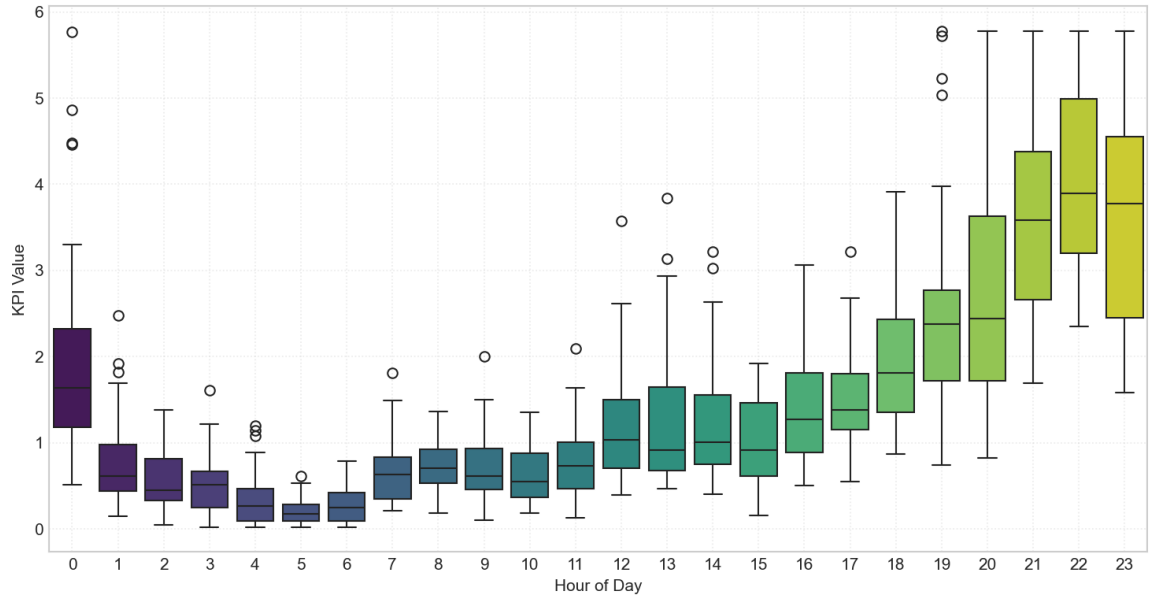


Figure C.5: `LTE_Traffic_Volume_DL` by Hour of Day.

The hourly profile (Figure C.5) vividly illustrates the diurnal traffic pattern, with minimal traffic in the early morning and sharp peaks in the late evening, justifying the creation of time-of-day features.

C.2 EDA for DL_PRB_usage

The following plots provide a comprehensive visual analysis of the hourly DL_PRB_usage KPI.

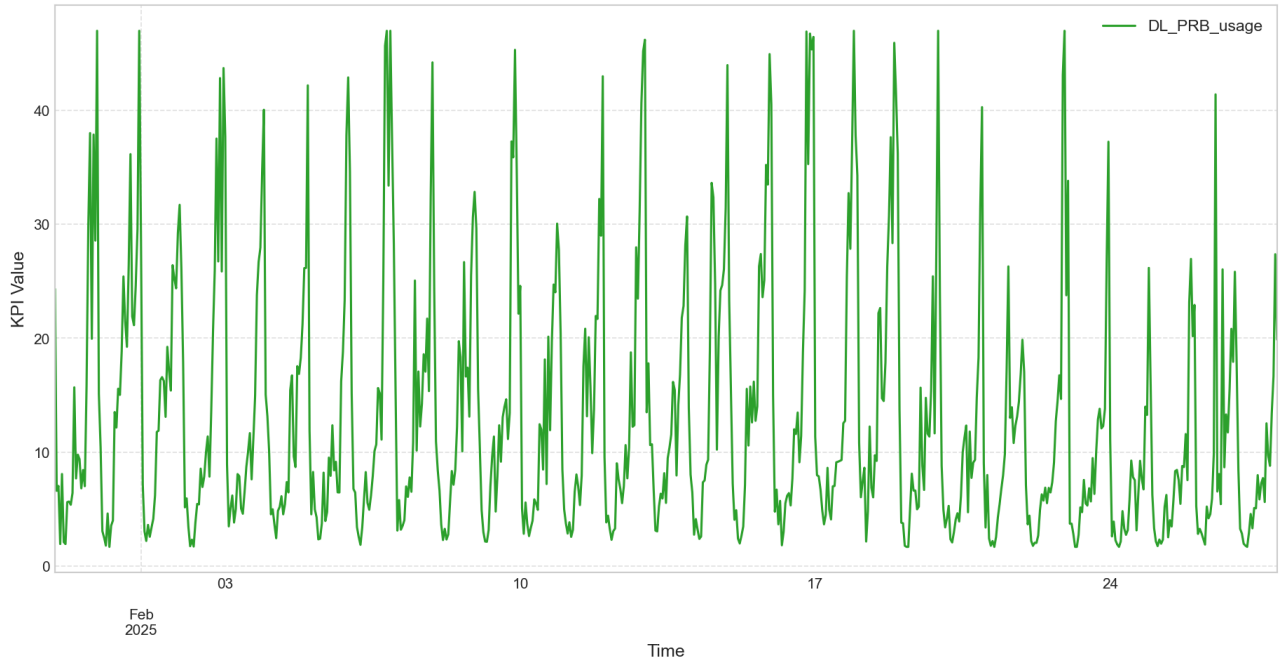
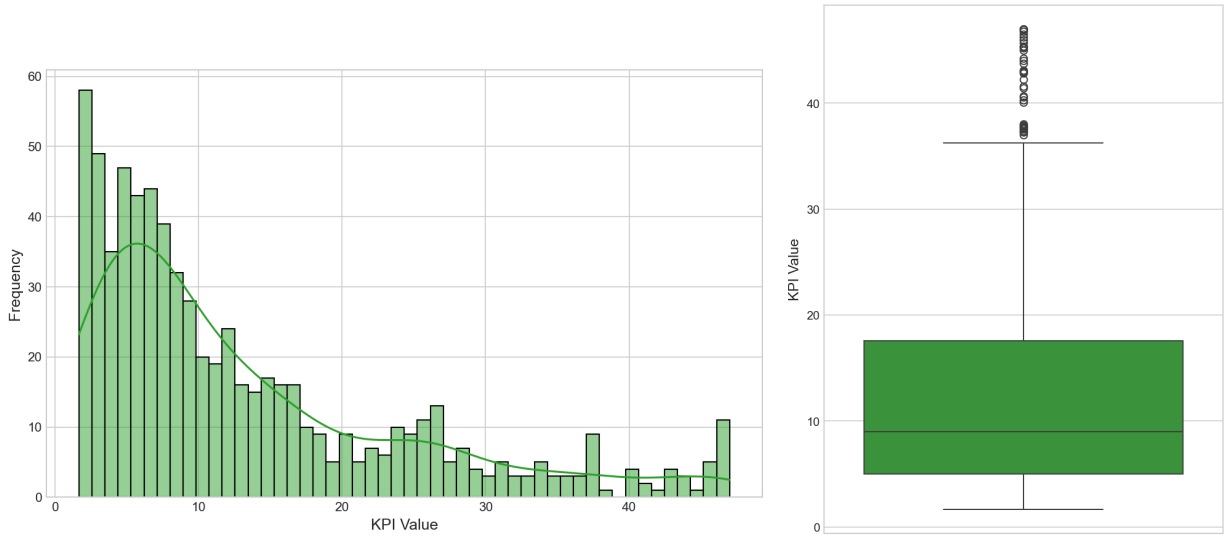


Figure C.6: Hourly Time Series of DL_PRB_usage (Training Data).

The time series for DL_PRB_usage (Figure C.6) mirrors the cyclical patterns of traffic volume, increasing during high data demand and decreasing during low-activity periods.



(a) Distribution (Histogram and KDE). (b) Box Plot.

Figure C.7: Distribution and outlier analysis for hourly DL_PRB_usage.

Similar to traffic, the distribution of DL_PRB_usage (Figure C.7a) is positively skewed. The box plot (Figure C.7b) confirms this and highlights outliers corresponding to hours of high resource utilization.

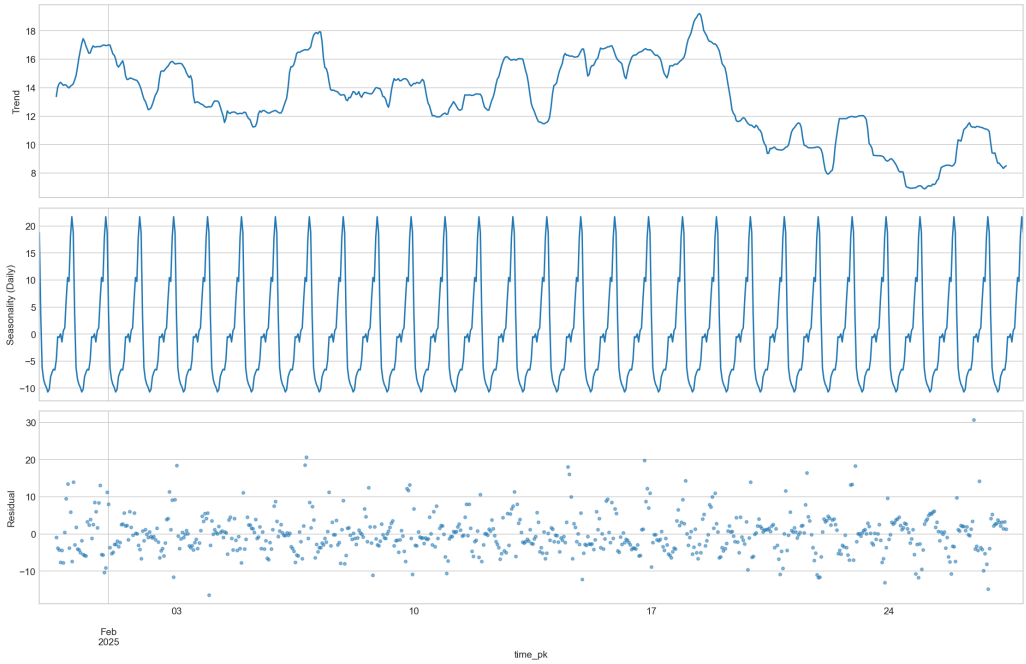


Figure C.8: Additive Seasonal Decomposition of `DL_PRB_usage`.

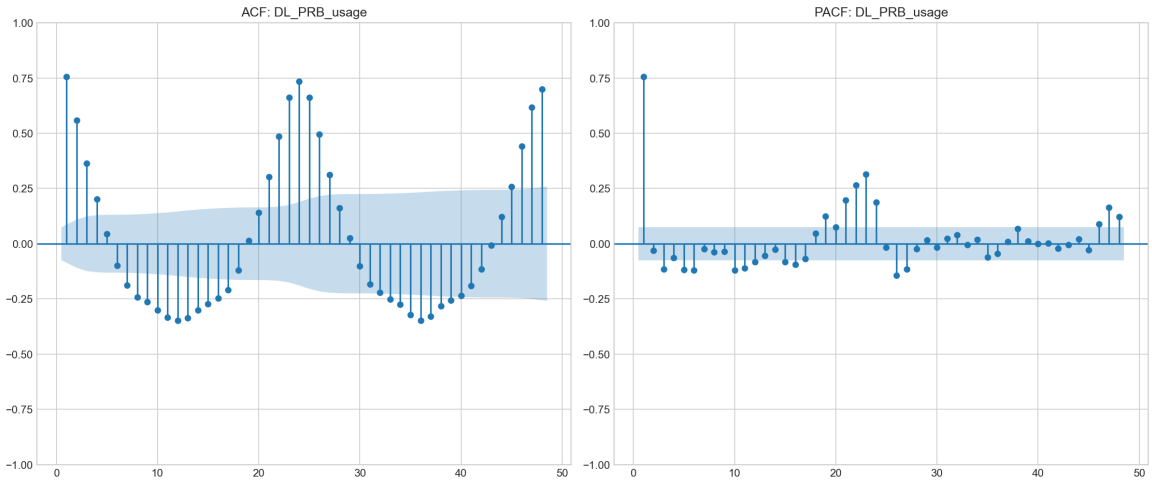


Figure C.9: ACF and PACF Plots for `DL_PRB_usage`.

Seasonal decomposition (Figure C.8) clearly extracts a strong daily seasonal component, which is further confirmed by the strong, repeating patterns in the ACF/PACF plots (Figure C.9).

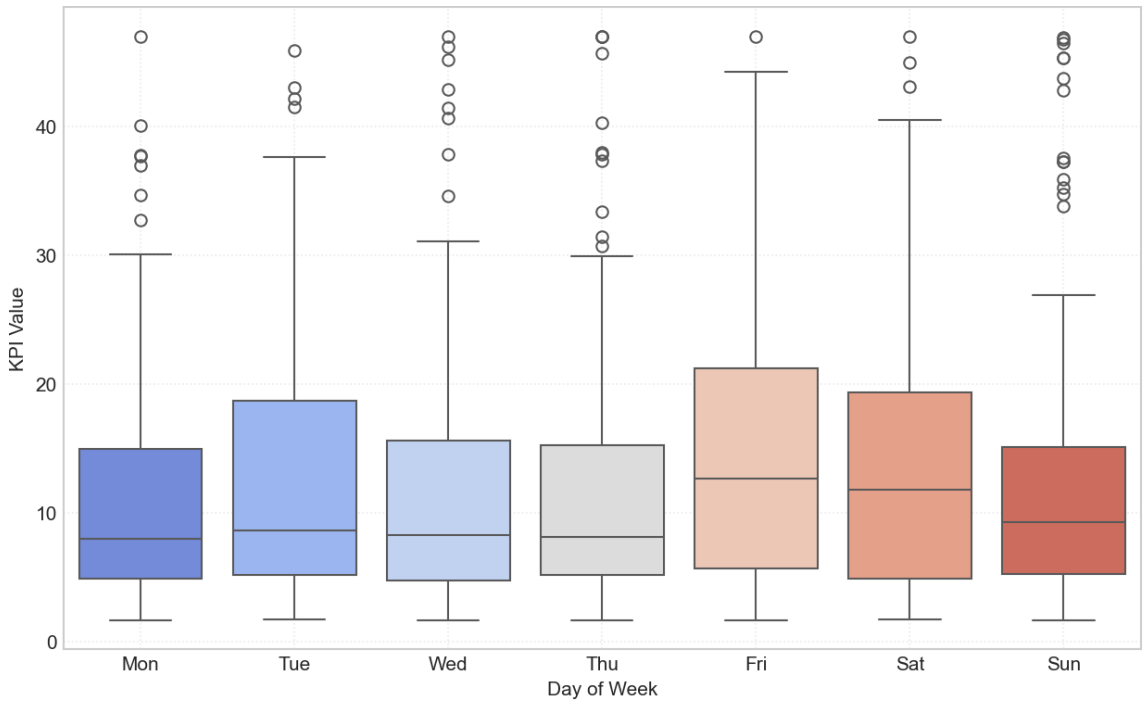


Figure C.10: `DL_PRB_usage` by Day of Week.

The weekly profile (Figure C.10) indicates differing PRB usage patterns across days, particularly on weekends, confirming the need for day-of-week features in the forecasting models.

Bibliography

[1] GSMA, "Lte standards evolution towards an all-business-connected primary infrastructure," February 2018.

[2] Bale. (2025) Lte architecture. [Online]. Available: https://forum.huawei.com/enterprise/intl/en/thread/LTE-Architecture/667281353609199616?blogId=667281353609199616

[3] X. Zhang, *LTE Optimization Engineering Handbook*, 2018.

[4] Huawei Technologies Co., Ltd., *eNodeB V100R005C00 KPI Reference*, 2012, issue 01, March 30.

[5] Gartner, "The impact of marginal network performance improvements on operator revenue and churn," Gartner, Inc., Tech. Rep., 2023, available to Gartner subscribers.

[6] A. Mason, "Network performance and customer retention: Quantifying revenue protection," Analysys Mason Ltd., Tech. Rep., 2022, access requires subscription.

[7] Ericsson, "Ericsson mobility report," Ericsson AB, Tech. Rep., 2023, accessed: 2025-06-08. [Online]. Available: https://www.ericsson.com/en/reports-and-papers/mobility-report

[8] S. B. Patel and M. Kansara, "Comparative study of 2g, 3g, and 4g," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT)*, vol. 3, pp. 2456–3307, September 2018.

[9] K. Budigere, B. Chemmagate *et al.*, "Lte: Long term evolution of 3gpp," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, April 2010.

[10] 3GPP, "Long term evolution (lte) and system architecture evolution (sae)," 2008, 3GPP Release 8.

[11] DigitalCommons, "Lte phy performance analysis under 3gpp standards parameters," *DigitalCommons@University of Nebraska - Lincoln*, 2018.

[12] 3GPP, "Evolved universal terrestrial radio access network (e-utran) and evolved packet core (epc)," 2018, 3GPP TS 36.300.

[13] A. Ghosh, J. Zhang, J. G. Andrews, and R. Muhamed, *Fundamentals of LTE*. Pearson Education, 2010.

[14] M. Z. Shakir, M. A. Al-Garadi, M. J. Bocus, and O. A. Dobre, "Machine learning for lte/5g network optimization: A survey," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 486–524, 2023.

[15] N. H. Mohammed, H. Nashaat, S. M. Abdel-Mageid, and R. Y. Rizk, "A machine learning-based framework for efficient lte downlink throughput," *Wireless Personal Communications*, vol. 126, no. 3, pp. 2407–2433, 2022.

[16] A. Adeel, H. Larijani, A. Javed, and A. Ahmadinia, "Critical analysis of learning algorithms in random neural network based cognitive engine for LTE systems," in *Proceedings of the 2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*. Glasgow, UK: IEEE, May 2015, pp. 1–5.

[17] H. Wang, F. Xu, Y. Li, P. Zhang, and D. Jin, "Understanding mobile traffic patterns of large scale cellular towers in urban environment," in *Proceedings of the 2015 Internet Measurement Conference*, Tokyo, Japan, October 2015, pp. 225–238.

[18] Y. Zang, F. Ni, Z. Feng, S. Cui, and Z. Ding, "Wavelet transform processing for cellular traffic prediction in machine learning networks," in *Proceedings of the 2015 IEEE China Summit and International Conference on Signal and Information Processing (ChinaSIP)*. Chengdu, China: IEEE, July 2015, pp. 458–462.

[19] L. Pierucci and D. Micheli, "A neural network for quality of experience estimation in mobile communications," *IEEE MultiMedia*, vol. 23, no. 3.

[20] F. Xu, Y. Lin, J. Huang, D. Wu, H. Shi, J. Song, and Y. Li, "Big data driven mobile traffic understanding and forecasting: A time series approach," *IEEE Transactions on Services Computing*, vol. 9, no. 5, pp. 796–805, 2016.

[21] A. Nikravesh, S. Ajila, C.-H. Lung, and W. Ding, "Mobile network traffic prediction using MLP, MLPWD, and SVM," in *Proceedings of the 2016 IEEE International Congress on Big Data (BigData Congress)*. San Francisco, CA, USA: IEEE, June-July 2016, pp. 402–409.

[22] J. Wang, J. Tang, Z. Xu, Y. Wang, G. Xue, X. Zhang, and D. Yang, "Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach," in *Proceedings of the IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. Atlanta, GA, USA: IEEE, May 2017, pp. 1–9.

[23] C.-W. Huang, C.-T. Chiang, and Q. Li, "A study of deep learning networks on mobile traffic forecasting," in *Proceedings of the 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. Montreal, QC, Canada: IEEE, October 2017, pp. 1–6.

[24] C. Zhang and P. Patras, "Long-term mobile traffic forecasting using deep spatio-temporal neural networks," in *Proceedings of the Eighteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Los Angeles, CA, USA, June 2018, pp. 231–240.

[25] X. Wang, Z. Zhou, F. Xiao, K. Xing, Z. Yang, Y. Liu, and C. Peng, "Spatio-temporal analysis and prediction of cellular traffic in metropolis," *IEEE Transactions on Mobile Computing*, vol. 18, no. 9, pp. 2190–2202, 2018.

[26] I. Alawe, A. Ksentini, Y. Hadjadj-Aoul, and P. Bertin, "Improving traffic forecasting for 5g core network scalability: A machine learning approach," *IEEE Network*, vol. 32, no. 6, pp. 42–49, 2018.

[27] A. Akbas, H. Yildiz, A. Ozbayoglu, and B. Tavli, "Neural network based instant parameter prediction for wireless sensor network optimization models," *Wireless Networks*, vol. 25.

[28] C. Zhang, H. Zhang, D. Yuan, and M. Zhang, "Citywide cellular traffic prediction based on densely connected convolutional neural networks," *IEEE Communications Letters*, vol. 22, no. 8, pp. 1656–1659, 2018.

[29] J. Feng, X. Chen, R. Gao, M. Zeng, and Y. Li, "DeepTP: An end-to-end neural network for mobile cellular traffic prediction," *IEEE Network*, vol. 32, no. 6, pp. 108–115, 2018.

[30] Y. Yamada, R. Shinkuma, T. Sato, and E. Oki, "Feature-selection based data prioritization in mobile traffic prediction using machine learning," in *Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM)*. Abu Dhabi, UAE: IEEE, December 2018, pp. 1–6.

[31] Y. Hua, Z. Zhao, Z. Liu, X. Chen, R. Li, and H. Zhang, "Traffic prediction based on random connectivity in deep learning with long short-term memory," in *Proceedings of the 2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*.

[32] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "DeepCog: Cognitive network management in sliced 5G networks with deep learning," in *Proceedings of the IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pp. 118–126.

[33] J. Shodamola, U. Masood, M. Manalastas, and A. Imran, "A machine learning based framework for KPI maximization in emerging networks using mobility parameters," September 2020, aI4Networks Research Center, Dept. of Electrical & Computer Engineering, University of Oklahoma, USA. Presented possibly at a conference or as a technical report.

[34] N. H. Mohammed, H. Nashaat, S. M. Abdel-Mageid, and R. Y. Rizk, *Intelligent Systems Design and Applications*, ch. A Machine Learning-Based Framework for Efficient LTE Downlink Throughput Optimization.

[35] D. Wass, "Transformer learning for traffic prediction in mobile networks," KTH Royal Institute of Technology, Stockholm, Sweden, Degree Project in Computer Science and Engineering, 2021, tRITA-EECS-EX-2021:319.

[36] S. T. Nabi, M. R. Islam, M. G. R. Alam, M. M. Hassan, S. A. Alqahtani, G. Aloi, and G. Fortino, "Deep learning based fusion model for multivariate LTE traffic forecasting and optimized radio parameter estimation," *IEEE Access*, vol. 11, pp. 12 345–12 356, 2023. [Online]. Available: https://doi.org/10.1109/ACCESS.2023.3242861

[37] S. Sharma, "Machine learning-based predictive modeling for 4G long term evolution (LTE) traffic prediction," 2023, publication details unavailable. Year estimated from table context.

[38] M. A. Mohammadi Banadaki, "Optimizing LTE network performance using machine learning techniques," MSc Research Project, Arden University, 2023, course code: RES6012SCC-Research Project.

[39] A. G. B. Colpitts and B. R. Petersen, "Short-term multivariate KPI forecasting in rural fixed wireless LTE networks," *IEEE Networking Letters*, 2023.

[40] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[41] S. J. Taylor and B. Letham, "Forecasting at scale," *The American Statistician*, vol. 72, no. 1, pp. 37–45, 2018.

[42] S. Sesia, I. Toufik, and M. Baker, *LTE – The UMTS Long Term Evolution: From Theory to Practice*, second edition ed. John Wiley Sons, 2011.

[43] X. Zhang, *LTE Optimization Engineering Handbook*. John Wiley Sons, 2018.

[44] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 2623–2631.

[45] K. K. A. Ghany, H. M. Zawbaa, and H. M. Sabri, "COVID-19 prediction using LSTM algorithm: GCC case study," *Informatics in Medicine Unlocked*, vol. 26, p. 100714, 2021.

[46] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[47] S. Dash, C. Chakraborty, S. K. Giri, and S. K. Pani, "Intelligent computing on time-series data analysis and prediction of covid-19 pandemics," *Pattern Recognition Letters*, vol. 151, pp. 69–75, November 2021.

[48] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 2nd ed. OTexts, 2018, melbourne, Australia. [Online]. Available: https://otexts.com/fpp2/

[49] N. I. of Standards and Technology, "Percentiles," https://www.itl.nist.gov/div898/handbook/prc/section2/prc252.htm, 2012, accessed: 2025-06-08.

[50] A. Y. Alaouchiche and M. W. Kessoum, "Kpi analytics platform: An integrated approach for cross-generation mobile network performance monitoring and analysis," Master Project, ENSTA, 2025, master Report.

[51] M. A. A. Mohammadi Banadaki, "Ai-driven multi-kpi optimization for self-organizing cellular networks," Ph.D. dissertation, University of Waterloo, 2023, available at UWSpace: http://hdl.handle.net/10012/19888.

[52] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[53] D. Wass, "Transformer learning for traffic prediction in mobile networks," Degree Project in Computer Science and Engineering, Second Cycle, KTH Royal Institute of Technology, Stockholm, Sweden, 2021, 30 credits.