

AI-Based Lead-Time Forecasting and Anomaly Detection in Supply Chain Using CNN-BiLSTM and LSTM-Autoencoder

1st Jayneshkumar Tandel [22403500]

2nd Riken Patel [22409741]

3rd Nikunj Kumar Mistry [12403425]

4th Tulsi Barot [22304477]

Master Applied AI for Digital Production Management

Deggendorf Institute Of Technology, Cham, Germany

Emails: { jayneshkumar.tandel, riken.Patel, nikunj.Mistry, tulsi.barot }@stud.th-deg.de

Abstract: - In today's highly dynamic supply chain environments, unpredictable supplier lead times cause production delays, stockouts, and increased operational costs. Traditional enterprise resource planning (ERP) systems only record historical data but lack predictive intelligence to forecast future delays or detect abnormal supplier behaviour. This project proposes an **AI-based hybrid framework** for *lead-time forecasting* and *anomaly detection* in the automotive and spare-parts supply chain. The forecasting module employs a **CNN-BiLSTM (Convolutional Neural Network-Bidirectional Long Short-Term Memory)** model that captures both short-term and long-term temporal dependencies from ERP data to predict supplier lead times more accurately than traditional statistical methods. For anomaly detection, an **LSTM Autoencoder** is combined with a **One-Class Support Vector Machine (OCSVM)** to learn normal supplier behaviour patterns and identify deviations automatically. The system is further integrated into a **real-time dashboard (FastAPI + Streamlit)** that visualizes predicted lead times, highlights anomalies, and supports proactive decision-making. Experimental results demonstrate that the proposed hybrid model significantly reduces forecasting error and provides early alerts for supplier irregularities, enhancing operational visibility, reliability, and efficiency within the supply chain network.

Keywords: - Supply Chain Management, Lead Time Forecasting, Anomaly Detection, CNN-BiLSTM, LSTM Autoencoder, One-Class SVM, ERP Data Analytics, FastAPI, Streamlit, Deep Learning

I. Introduction

In the era of globalized production and Industry 4.0, supply chain efficiency depends heavily on accurate and reliable information about supplier lead times. Lead time the duration between placing an order and receiving it directly affects inventory levels, production planning, and customer satisfaction. However, due to global disruptions, manufacturing variability, and inconsistent supplier

performance, lead times often fluctuate unpredictably. Such uncertainties can cause **stockouts**, **delayed deliveries**, and **increased holding costs**, forcing organizations to either overstock or risk supply shortages. Although most enterprises use **ERP systems** to record supplier transactions and order histories, these systems are largely *reactive*: they describe what happened but cannot predict what will happen next.

To address this challenge, artificial intelligence (AI) and deep learning techniques offer the potential to learn complex patterns from past data and forecast future outcomes. In this project, we propose a hybrid deep learning framework that combines **Convolutional Neural Networks (CNN)** and **Bidirectional Long Short-Term Memory (BiLSTM)** for accurate lead-time forecasting. CNN captures localized time-series features, while BiLSTM processes long-term dependencies in supplier data sequences.

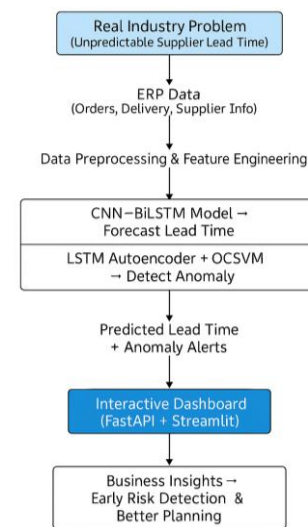


Figure 1 Research Framework

In addition to forecasting, the project incorporates an **anomaly detection module** to identify unusual supplier behavior. Using an **LSTM Autoencoder**, the model learns the normal temporal patterns of lead times, and when combined with a **One-Class SVM**, it can automatically

classify new patterns as normal or anomalous. This dual approach ensures not only the prediction of expected lead times but also the early detection of supply risks.

The final system is implemented as a **real-time interactive dashboard** using FastAPI and Streamlit, enabling supply chain managers to visualize predicted lead times, monitor anomalies, and make data-driven decisions. By integrating forecasting and anomaly detection in a single intelligent framework, this project contributes to improving supply chain resilience, reducing uncertainty, and enhancing service reliability.

II. Literature Review and Project Justification

A. Introduction to Literature Review

A literature review provides the theoretical foundation and background for the development of this research project. It helps identify what has been achieved in the domain, the limitations of existing methods, and the gaps that the current study aims to fill. In the context of this project which focuses on **AI-based lead-time forecasting and anomaly detection in supply chains** reviewing previous studies is essential to understand how traditional and modern approaches have addressed similar challenges.

Recent disruptions, supplier variability, and logistic uncertainties have highlighted the need for **data-driven prediction systems** that can anticipate supplier delays and irregularities. While **Enterprise Resource Planning (ERP)** systems collect vast historical data on orders and deliveries, they remain reactive and lack predictive capability (*Amellal et al., 2023*). The review below explores existing forecasting and anomaly detection methods, identifies their assumptions and limitations, and demonstrates the need for an integrated intelligent system capable of real-time forecasting and monitoring.

B. Why the Topic Addresses a Real Industrial Problem

In modern manufacturing and logistics, particularly within the **automotive spare-parts sector**, supplier lead-time uncertainty is a critical challenge. Inconsistent lead times cause cascading effects such as **production line stoppages**, **inventory overstocking**, and **missed customer deadlines**. Despite ERP systems storing valuable supplier and order data, they primarily serve as **recording tools** rather than predictive engines (*Amellal et al., 2023*).

According to *Babai et al. (2022)*, variability in lead-time demand can distort safety stock calculations, leading either to shortages or excessive holding costs. The inability to forecast accurate lead times results in reactive management, which limits supply chain responsiveness. An **AI-based predictive framework** that can forecast supplier lead time and detect anomalies proactively offers a direct solution to this industrial pain point, improving supply chain visibility, planning accuracy, and operational efficiency.

C. How Others Have Approached the Problem

Research on supply chain forecasting and anomaly detection has evolved across three major categories - **statistical models**, **machine learning methods**, and **deep learning frameworks**.

a. Statistical Approaches

Classical time-series methods such as **ARIMA**, **ARMA**, and **Exponential Smoothing** models have been widely used for demand and lead-time forecasting (*Babai et al., 2022*). These approaches assume stationarity and linear relationships among data variables. While effective for stable environments, they fail under nonlinear conditions and cannot capture long-term temporal dependencies. Consequently, they are inadequate for complex supply chain systems where supplier behaviour fluctuates due to market, geographical, or logistical constraints.

b. Machine Learning Approaches

With the rise of data availability, **machine learning models** like **Random Forest**, **Gradient Boosting**, and **Support Vector Regression (SVR)** were introduced (*Roy et al., 2015*). These models outperform statistical methods by learning nonlinear relationships between features such as supplier ID, shipping mode, or region. However, they treat each data instance independently and lack the ability to model **temporal order**, making them unsuitable for sequential data like delivery lead times.

c. Deep Learning Approaches for Forecasting

The advent of deep learning brought models capable of learning temporal dependencies. **Recurrent Neural Networks (RNNs)**, particularly **Long Short-Term Memory (LSTM)** and **Bidirectional LSTM (BiLSTM)** networks, have shown significant success in forecasting time-dependent data (*Siarni-Namini et al., 2019*). These models can capture patterns across multiple time steps but can still struggle with local variations.

To address this, researchers combined **Convolutional Neural Networks (CNNs)** with LSTMs, creating **CNN-LSTM** and **CNN-BiLSTM** hybrids (*Miao et al., 2021; Lu et al., 2020*). CNNs efficiently extract short-term spatial patterns, while BiLSTM captures long-term sequential dependencies, producing higher forecasting accuracy. In supply chain contexts, this combination offers superior modeling of both immediate and cumulative supplier behavior patterns.

d. Approaches to Anomaly Detection

Anomaly detection has been traditionally handled through **statistical outlier detection**, **rule-based thresholds**, or **machine learning methods** like **Isolation Forest** and **Kernel Density Estimation (KDE)** (*Kerdprasop et al., 2019*). However, these methods rely on fixed boundaries and are not suitable for evolving time-series data.

Aspect	Prior Approaches	Proposed Approach	Justification
Forecasting Model	ARIMA, Random Forest, LSTM (Babai et al., 2022; Lu et al., 2020)	CNN–BiLSTM	CNN extracts short-term features; BiLSTM models long-term dependencies → superior accuracy.
Anomaly Detection	Isolation Forest, KDE, Simple Autoencoder (<i>Kerdprasop et al., 2019</i>)	LSTM Autoencoder + OCSVM	Learns sequential normal patterns and uses OCSVM for robust boundary detection.
Data Type	Static, tabular data	Sequential ERP time-series data	Reflects actual supplier delivery behaviour over time.
Deployment	Offline analysis	FastAPI + Streamlit dashboard	Enables real-time, interactive monitoring and decision-making.
Evaluation Metrics	MAE, RMSE	MAE, RMSE, MAPE, Precision, Recall, F1-score	Comprehensive evaluation for both forecasting and anomaly detection.

Recent studies have adopted **Autoencoders** and **LSTM Autoencoders** for unsupervised anomaly detection (*Nguyen et al., 2021*). These models learn normal sequence patterns and identify anomalies through high reconstruction errors. Combining an **LSTM Autoencoder** with a **One-Class SVM (OCSVM)** enhances robustness by creating a mathematically defined boundary around normal behaviour, reducing false positives and improving precision.

D. Gap and Positioning

While many studies have focused on **demand forecasting** or **inventory optimization**, **lead-time forecasting** remains less explored, despite its central role in supply chain planning. Moreover, most existing works treat forecasting and anomaly detection as **separate tasks** rather than an integrated system.

Amellal et al. (2023) demonstrated the potential of a hybrid **CNN–BiLSTM + LSTM-AE + OCSVM** model for automotive spare-parts forecasting, achieving significant improvements in predictive accuracy. However, their study remained academic, lacking an operational interface for industrial deployment.

This project extends that research by:

- Applying the hybrid deep learning framework to ERP-based supply chain data.
- Integrating the forecasting and anomaly detection modules into a **single intelligent system**.
- Deploying the models in a **real-time dashboard** using **FastAPI** and **Streamlit**, enabling immediate visualization of predictions and anomaly alerts.

Thus, this research bridges the gap between theoretical advancements and **practical, deployable AI systems** for smart supply chain management.

E. Tools and Methods Used in Prior Work and Justification for the Proposed Approach

The proposed model design is motivated by empirical findings in *Amellal et al. (2023)* and extended by integrating it with a practical visualization and deployment layer for real industry application.

III. Problem Statement

In modern supply chains, especially in the **automotive spare-parts sector**, organizations face a persistent challenge in accurately predicting supplier **lead times** and identifying **abnormal supplier behaviours**. Despite extensive data captured by **Enterprise Resource Planning (ERP)** systems, most existing solutions are **reactive**—they record historical transactions but fail to predict potential disruptions before they occur. This leads to several operational inefficiencies, including:

- **Uncertain delivery schedules**, which disrupt production and logistics planning.
- **Stockouts** or **excess inventory**, resulting in financial and operational losses.
- **Delayed customer fulfilment** and reduced supply chain reliability.

Traditional statistical methods (e.g., ARIMA, exponential smoothing) and even conventional machine learning models (e.g., Random Forest, SVM) are **inadequate** for capturing the **nonlinear, temporal, and multivariate patterns** found in ERP data. Furthermore, most research efforts focus solely on forecasting accuracy, ignoring the critical need for **anomaly detection** in supplier performance.

Therefore, there is a clear **research and industrial gap**: the absence of an integrated, AI-driven system that can both **forecast supplier lead time** and **detect irregular supplier behaviour** in real time.

This project addresses that gap by developing a **hybrid deep learning framework** that combines **CNN–BiLSTM** for lead-time forecasting and **LSTM Autoencoder + One-Class SVM (OCSVM)** for anomaly detection, integrated into a **real-time interactive dashboard** for visualization and decision support.

IV. Research Objectives

The main goal of this project is to design and implement an **AI-based intelligent forecasting and anomaly detection system** for improving supply chain reliability.

To achieve this, the following specific objectives are defined:

1. **To analyse and preprocess ERP data** related to supplier lead time, shipment dates, and delivery performance.
2. **To design a hybrid CNN–BiLSTM deep learning model** capable of accurately forecasting supplier lead times by learning both short-term and long-term dependencies.
3. **To develop an LSTM Autoencoder + OCSVM-based model** for detecting abnormal supplier behaviors based on reconstruction errors and decision boundaries.
4. **To evaluate model performance** using appropriate forecasting and classification metrics such as MAE, RMSE, MAPE, Precision, Recall, and F1-score.
5. **To integrate both models** into a unified system using **FastAPI** and **Streamlit**, providing an interactive dashboard for real-time visualization and anomaly alerts.
6. **To demonstrate the applicability of the system** in a real or simulated ERP environment, showcasing its potential impact on industrial supply chain planning.

V. Methodology

The study follows a data-driven experimental methodology. First, ERP-like supply chain data is cleaned and transformed to derive the target variable, supplier lead time. Categorical attributes (supplier, route, product) are encoded, and numerical attributes are scaled. A hybrid deep learning model based on CNN–BiLSTM is then trained to forecast future lead times from historical sequences, allowing the model to learn both local temporal patterns (via CNN) and long-range

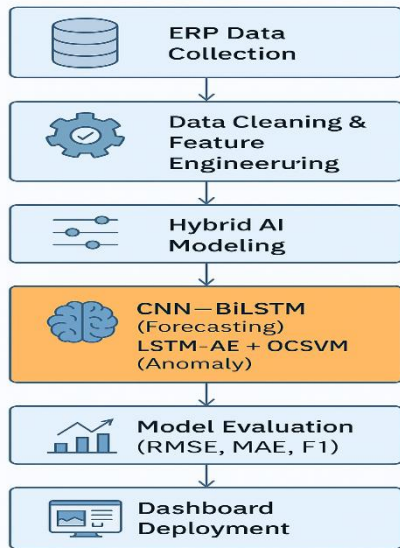


Figure 2 Analysis Framework

dependencies (via BiLSTM). In parallel, an LSTM-based autoencoder is trained on normal delivery sequences to learn typical supplier behaviour; its reconstruction errors are further modelled using a One-Class SVM to detect anomalous lead-time patterns. Model performance is evaluated using MAE, RMSE, and MAPE for forecasting, and precision/recall for anomaly detection. Finally, the models are intended to be exposed through a lightweight API (FastAPI)

and visualized in a dashboard (Streamlit) for operational supply chain decision-making.

A. Data Set

a. Data Collection

The data source is a Moroccan company that specializes in the importation and distribution of automobile spare parts. The company markets various automobile brands from different manufacturers and operates across multiple countries, which leads to a complex logistics process with varying laws and policies. The high number of references to manage and the volatile demand for certain parts make the process challenging. The data covers the period from January 2019 to August 2022. To address the challenges related to data quality, completeness, and accuracy, we took several measures. Firstly, we worked closely with the automobile company to obtain the necessary data and ensure that it was accurate and complete. This involved collaborating with the company's IT department to obtain the data from the ERP system. Moreover, we acknowledged the importance of data privacy and confidentiality in our study. As the data we were working with involved sensitive and proprietary information, we took measures to ensure that the data was anonymized and secured. We removed any personally identifiable information from the dataset, such as customer names or addresses, to protect the privacy of individuals. By taking these measures, we aimed to ensure that the data was reliable and suitable for analysis, while also protecting the privacy of individuals and complying with applicable data protection regulations.

The dataset was provided as a structured CSV export and contains **180,519 records with 53 attributes**, capturing order-level and logistics-level information such as shipping mode, market, order region, department, customer sales, product price, discount, profit ratio, and shipping time variables. Two time-related columns are particularly important for lead time modelling: **Days for shipping (real)**, which represents the *actual shipping/lead time in days*, and **Days for shipment (scheduled)**, which represents the *planned or scheduled shipping time*. In addition, the dataset includes a binary operational risk label **Late_delivery_risk** (0/1), which indicates whether an order is associated with late-delivery risk. The data was collected in an offline manner by loading the CSV export into a Python analytics pipeline using pandas, which reflects a common industrial workflow where ERP records are exported periodically and used for model training and evaluation on historical patterns.

b. Data Cleaning

After loading the data, a structured **data cleaning** procedure was applied to ensure reliability before training deep learning models. First, duplicate entries were removed to avoid overrepresenting repeated transactions. Then, the raw date fields **order date (DateOrders)** and **shipping date (DateOrders)** were converted into valid datetime format (invalid strings were coerced to null values), and records with invalid or missing timestamps were removed. This step is

	A	B	C	D	E	F	G	H	I
1	Type	Days for shipping (real)	Days for shipment (scheduled)	Benefit per order	Sales per customer	Delivery Status	Late_delivery_risk	Category Id	Category
2	DEBIT	3	4	91.25	314.6400146	Advance shipping	0	73	Sporting C
3	TRANSFER	5	4	-249.0899963	311.3599854	Late delivery	1	73	Sporting C
4	CASH	4	4	-247.7799988	309.7200012	Shipping on time	0	73	Sporting C
5	DEBIT	3	4	22.86000061	304.8099976	Advance shipping	0	73	Sporting C
6	PAYMENT	2	4	134.2100067	298.25	Advance shipping	0	73	Sporting C
7	TRANSFER	6	4	18.57999992	294.980011	Shipping canceled	0	73	Sporting C
8	DEBIT	2	1	95.18000031	288.4200134	Late delivery	1	73	Sporting C
9	TRANSFER	2	1	68.43000031	285.1400146	Late delivery	1	73	Sporting C
10	CASH	3	2	133.7200012	278.5899963	Late delivery	1	73	Sporting C
11	CASH	2	1	132.1499939	275.3099976	Late delivery	1	73	Sporting C
12	TRANSFER	6	2	130.5800018	272.0299988	Shipping canceled	0	73	Sporting C
13	TRANSFER	5	2	45.68999863	268.7600098	Late delivery	1	73	Sporting C
14	TRANSFER	4	2	21.76000023	262.2000122	Late delivery	1	73	Sporting C
15	DEBIT	2	1	24.57999992	245.8099976	Late delivery	1	73	Sporting C
16	TRANSFER	2	1	16.38999939	327.75	Late delivery	1	73	Sporting C
17	DEBIT	2	1	-259.5799866	324.4700012	Late delivery	1	73	Sporting C
18	PAYMENT	5	2	-246.3600006	321.2000122	Late delivery	1	73	Sporting C
19	CASH	2	1	23.84000015	317.9200134	Late delivery	1	73	Sporting C
20	DEBIT	2	1	102.2600021	314.6400146	Late delivery	1	73	Sporting C
21	PAYMENT	0	0	87.18000031	311.3599854	Shipping on time	0	73	Sporting C
22	TRANSFER	0	0	154.8600006	309.7200012	Shipping on time	0	73	Sporting C
23	TRANSFER	5	4	82.30000305	304.8099976	Late delivery	1	73	Sporting C
24	TRANSFER	4	2	22.37000084	298.25	Late delivery	1	73	Sporting C
25	TRANSFER	3	2	17.70000076	294.980011	Shipping canceled	0	73	Sporting C
26	TRANSFER	2	2	90.27999878	288.4200134	Shipping canceled	0	73	Sporting C

Figure 3 Dataset Example

essential because supply chain events must be time-consistent for correct training, visualization, and sequential modelling. To improve robustness against unrealistic observations, the target lead-time values were checked and filtered to retain only non-negative and practically reasonable values, preventing extreme outliers from biasing the learning process. The cleaned dataset was then sorted chronologically by order date to maintain a consistent temporal structure for downstream modelling and interpretation.

c. Feature Engineering

For **feature engineering**, three lead-time variables were defined, consistent with operational definitions in supply chain planning. The *actual lead time* was used directly as

$$LT_{actual} = \text{Days for shipping (real)},$$

while the *scheduled lead time* was defined as,

$$LT_{scheduled} = \text{Days for shipment (scheduled)}$$

A highly informative derived feature, called *lead-time variance*, was computed as ,

$$LT_{variance} = LT_{actual} - LT_{scheduled}$$

This variance captures the deviation between planned and actual performance and is a strong indicator of supplier or logistics instability; positive values reflect delays relative to schedule, while values near zero indicate stable performance aligned with planning assumptions.

d. Data Preprocessing

A complete **preprocessing pipeline** was implemented to transform raw ERP attributes into a model-ready numerical representation. A mixed feature set was selected to capture both operational drivers and business context, including **scheduled shipment days, order quantity, sales per**

customer, product price, discount, and profit ratio as numerical variables, along with **shipping mode, market, order region, and department name** as categorical variables. Categorical variables were transformed using **one-hot encoding** with a reference category dropped (to avoid redundant dummy variables), and numerical variables were standardized using **z-score normalization**,

$$z = \frac{x - \mu}{\sigma}$$

ensuring all continuous features were on a comparable scale. The encoded categorical vectors and normalized numerical vectors were then concatenated into a single feature matrix, producing the final input,

$$X = [X_{num} \mid X_{cat}]$$

For neural network compatibility, the feature matrix was reshaped into the LSTM input format $(N \cdot 1 \cdot F)$, where N is the number of samples, 1 indicates one timestep per record, and F is the total number of processed features after encoding. Finally, the dataset was split into training and testing partitions using an **80/20 split**, and an internal validation split was used during training to monitor generalization performance.

The most influential operational drivers in this dataset are the variables that directly define planned versus actual time behaviour. **Days for shipment (scheduled)** is the strongest baseline predictor because it encodes the expected delivery time determined during planning, and **Shipping Mode** is highly influential because different modes (e.g., same-day, first-class, standard) impose different logistics constraints and delivery-speed distributions. For late-risk and abnormal-behaviour detection, the **lead-time variance** $LT_{actual} - LT_{scheduled}$ is the most informative signal, because it directly measures deviation from the plan and captures supplier/

	Type	Delivery Status	Category Name	Customer Country	Customer Segment	Market	Order City	Order Country	Order Region	Order State	...	Order Item Product Price	Order Item Profit Ratio	Order Item Quantity	Sales	Order Item Total	Order Profit Per Order	Product Price	LeadTime	Late_d
0	CASH	Advance shipping	Camping & Hiking	EE. UU.	Consumer	LATAM	Mexico City	México	Central America	Distrito Federal	...	299.980011	0.37	1	299.980011	239.979996	88.790001	299.980011	2	
1	PAYMENT	Advance shipping	Water Sports	EE. UU.	Consumer	LATAM	Dos Quebradas	Colombia	South America	Risaralda	...	199.990005	0.47	1	199.990005	193.990005	91.180000	199.990005	3	
2	PAYMENT	Advance shipping	Women's Apparel	EE. UU.	Consumer	LATAM	Dos Quebradas	Colombia	South America	Risaralda	...	50.000000	0.30	5	250.000000	227.500000	68.250000	50.000000	3	
3	PAYMENT	Advance shipping	Men's Footwear	EE. UU.	Consumer	LATAM	Dos Quebradas	Colombia	South America	Risaralda	...	129.990005	0.34	1	129.990005	107.889999	36.470001	129.990005	3	
4	CASH	Late delivery	Indoor/Outdoor Games	EE. UU.	Home Office	LATAM	Dos Quebradas	Colombia	South America	Risaralda	...	49.980000	0.21	4	199.919998	159.940002	33.590000	49.980000	5	

5 rows × 27 columns

Figure 4 Preprocessing Result

logistics instability. Together, these variables explain a large portion of the variation in lead time and provide the key

Whenever the blue area rises above the orange, it indicates orders that exceeded the schedule (late performance), and the

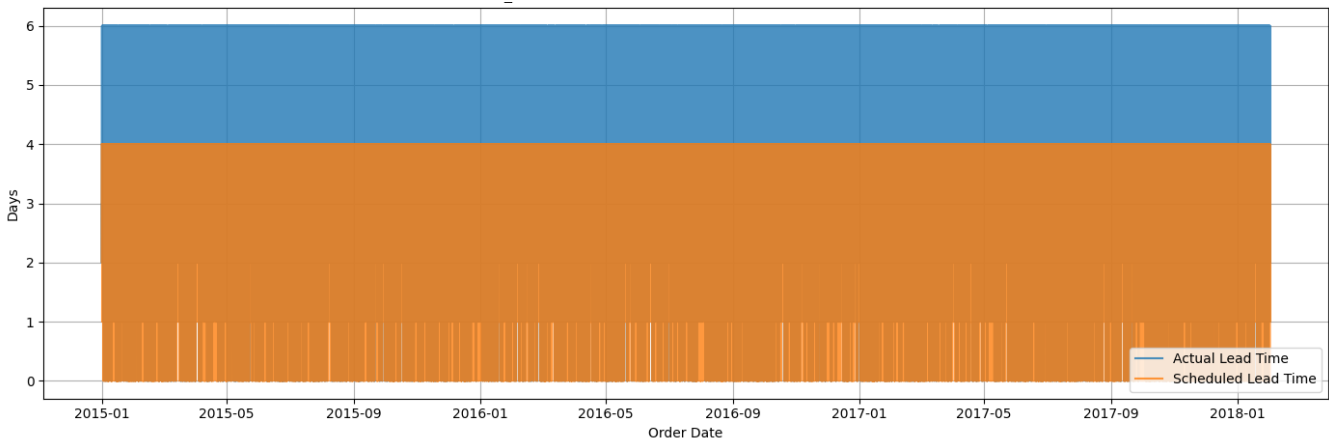


Figure 5 Lead Time (Actual vs Scheduled) Over Time

signals required for forecasting, late-risk classification, and anomaly identification within the hybrid intelligence system.

repeated peaks show that delays occur frequently rather than being a rare one-off event.

This Figure 5 compares **Actual Lead Time** (blue) and **Scheduled Lead Time** (orange) across order dates from

This histogram shows in Figure 6 how **Lead Time Variance** is distributed across all orders, where variance is defined as

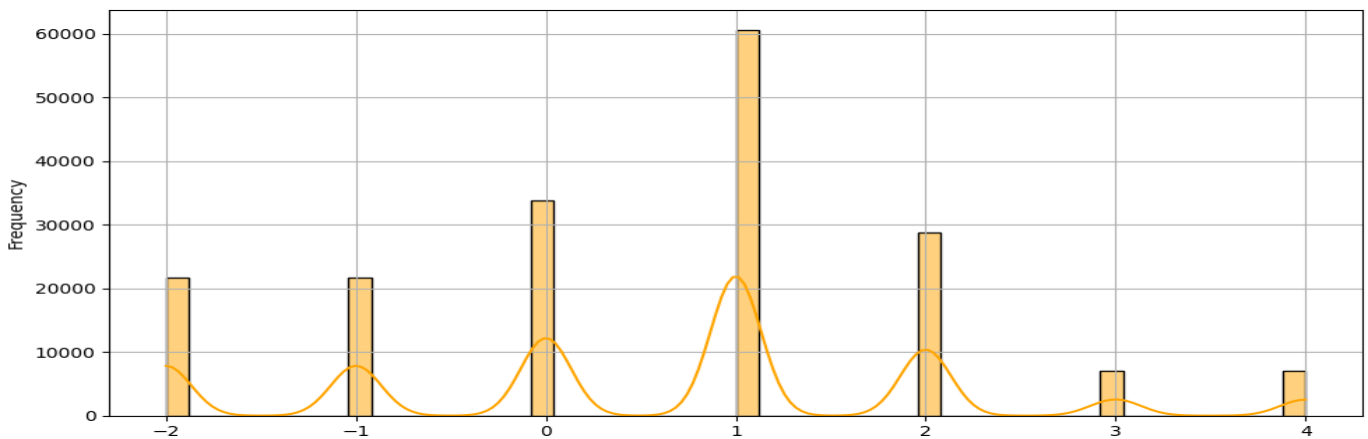


Figure 6 Distribution of Lead Time Variance

roughly **2015 to early 2018**, with the y-axis showing lead time in **days**. The orange series (scheduled) stays mostly within **0–4 days**, while the blue series (actual) extends higher, reaching up to **~6 days**, meaning real deliveries often take longer than planned. Because there are many transactions plotted, the lines appear very dense, but the key insight is the consistent **gap between actual and scheduled lead time**, which represents **delay variance** ($LT_{actual} - LT_{scheduled}$).

Actual Lead Time – Scheduled Lead Time (x-axis) and the y-axis shows how many orders fall into each variance value (frequency). Values near **0** mean deliveries happened roughly on schedule, **positive variance** (1, 2, 3, 4 days) means orders arrived **late** compared to the plan, and **negative variance** (–1, –2 days) means orders arrived **earlier** than scheduled. The plot has clear spikes at integer values, which happens because both actual and scheduled lead times are recorded in whole

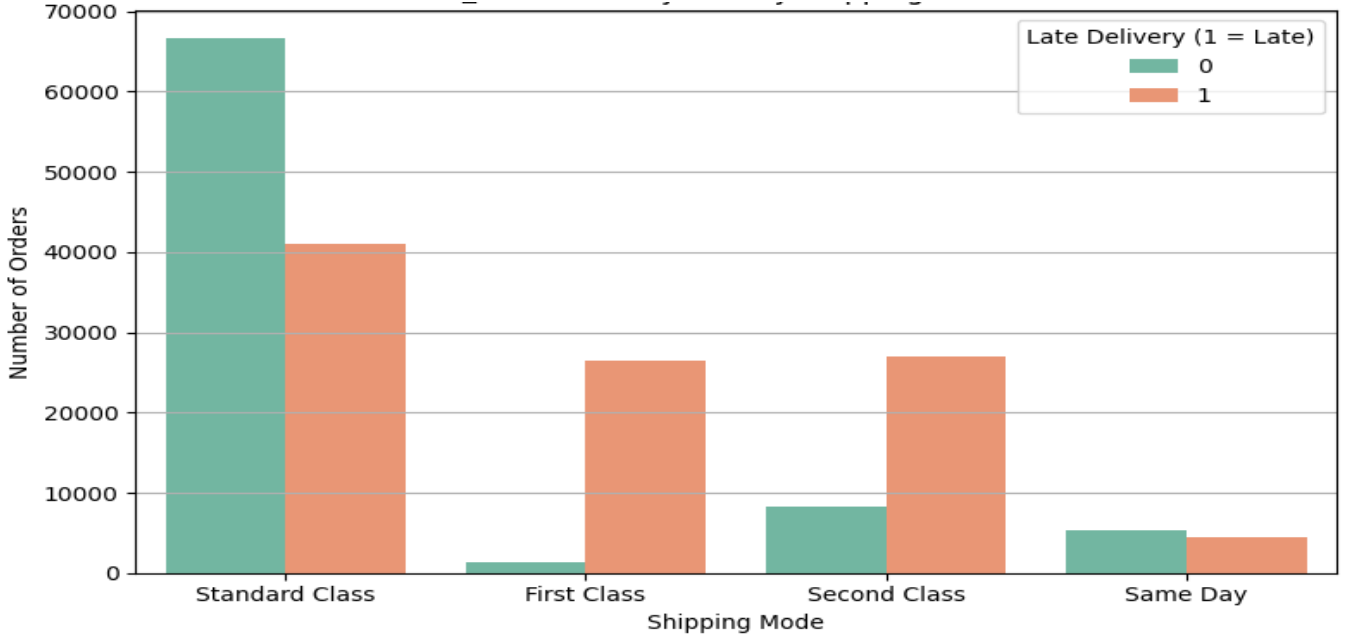


Figure 7 Late Delivery Risk by Shipping Mode

days. The tallest peak is around **+1 day**, meaning the most common situation is that deliveries arrive **about one day later than scheduled**, and there are also strong peaks at **0 days** (on-time) and **+2 days** (two-day delay). Smaller but visible bars at **+3 and +4** indicate a smaller number of more severe delays, which are important operationally because they can trigger stockouts or production disruption. Overall, the distribution is skewed toward **positive variance**, showing that lateness is more frequent than earliness in this dataset, and this is exactly why the variance feature is a strong signal for late-risk and anomaly detection.

Figure 7 shows how the number of orders is distributed across different shipping modes (Standard Class, First Class, Second Class, and Same Day) and separates each mode into **non-late (0)** and **late delivery risk (1)** categories. The chart clearly indicates that late-delivery risk varies significantly by shipping mode, meaning shipping mode is an important operational factor for predicting delays. **Standard Class** has the highest overall order volume, so it naturally also contains a large number of late-risk cases. However, the pattern is even more pronounced for **First Class** and **Second Class**, where the late-risk counts are noticeably higher than the non-late counts, suggesting that these modes experience a higher proportion of late-risk deliveries in this dataset. In contrast, **Same Day** has the lowest number of orders and comparatively fewer late-risk cases, implying that faster delivery modes tend to have lower late-risk occurrence. Overall, Figure 7 supports the inclusion of **Shipping Mode** as a key categorical feature in the late-risk prediction and anomaly detection models because it strongly influences delivery reliability.

B. Hybrid Modeling

In this section, we briefly overview the deep learning algorithms CNN, LSTM & BiLSTM, Autoencoder, and

OCSVM which were used to set up the proposed algorithm for forecasting and anomaly detection.

a. CNN (Convolutional Neural Network)

Convolutional Neural Networks (CNNs) are among the most popular deep learning architectures and are widely used in **computer vision, language modeling, sentiment analysis, machine translation**, and other pattern-recognition tasks. A CNN is designed to learn features automatically from input data by using **convolution filters (kernels)** that scan over the input and detect meaningful local patterns. **Figure 8** shows the basic CNN pipeline:

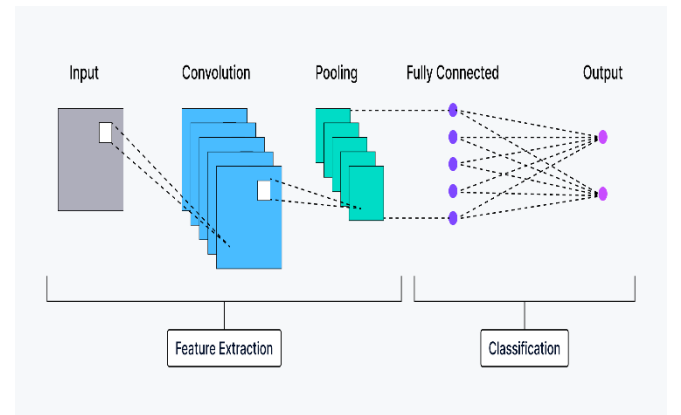


Figure 8 CNN Basic Architecture

Let an input image (or feature tensor) be represented as $X \in \mathbb{R}^{H \times W \times C}$, where H is height, W is width, and C is the depth (number of channels). For example, a grayscale image has $C = 1$ and an RGB image has $C = 3$. A convolutional layer applies K learnable filters (kernels), where each filter $W^{(k)} \in \mathbb{R}^{F_h \times F_w \times C}$ has a spatial size (F_h, F_w) and spans the full channel depth C . Each filter produces one **feature map**, so the output depth becomes K . The 2D convolution operation for filter k at spatial location (i, j) is computed as:

$$Z_{i,j,k} = \sum_{c=1}^C \sum_{u=0}^{F_h-1} \sum_{v=0}^{F_w-1} W_{u,v,c}^{(k)} X_{i+u, j+v, c} + b_k$$

After this linear operation, a nonlinear activation function is applied (commonly ReLU):

$$A_{i,j,k} = \text{ReLU}(Z_{i,j,k}) = \max(0, Z_{i,j,k})$$

The output spatial dimensions depend on **stride** S and **padding** P . The output height and width are:

$$H_{out} = \left\lfloor \frac{H - F_h + 2P}{S} \right\rfloor + 1, W_{out} = \left\lfloor \frac{W - F_w + 2P}{S} \right\rfloor + 1$$

After convolution, CNNs commonly apply **pooling** to reduce spatial size, lower computation, and improve robustness. Pooling operates on each feature map independently using a window of size $p \times p$ (or $p_h \times p_w$). For **max pooling**, the pooled output is:

$$Y_{i,j,k} = \max_{(u,v) \in \text{window}} A_{i+u, j+v, k}$$

(For **average pooling**, the max is replaced by the average.) Pooling reduces dimensions and keeps the strongest (or average) responses, which helps the model focus on important features and reduces overfitting.

After several convolution + pooling blocks, the final feature maps are converted into a single vector using **Flatten**:

$$\mathbf{f} = \text{Flatten}(A) \in \mathbb{R}^d$$

This vector is passed to **fully connected (dense) layers**, similar to a traditional neural network:

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{f} + \mathbf{b})$$

For multi-class classification, the final layer typically uses **SoftMax** to output class probabilities:

$$\hat{y}_c = \frac{e^{z_c}}{\sum_{j=1}^M e^{z_j}}$$

where M is the number of classes. For binary classification, a **sigmoid** output is used instead.

b. Long Short-Term Memory (LSTM) & Bidirectional LSTM

Long Short-Term Memory (LSTM) is a specialized recurrent neural network architecture introduced by **Hochreiter and Schmidhuber (1997)** to overcome the main limitation of standard RNNs: learning long-range dependencies is difficult because gradients can **vanish** (shrink toward zero) or **explode** (grow uncontrollably) when information is propagated across many time steps. LSTM addresses this by introducing an explicit **memory pathway** called the **cell state** c_t , which is updated through a gating mechanism. This design allows the network to preserve important information over long sequences while selectively forgetting irrelevant content.

• LSTM cell structure and information flow

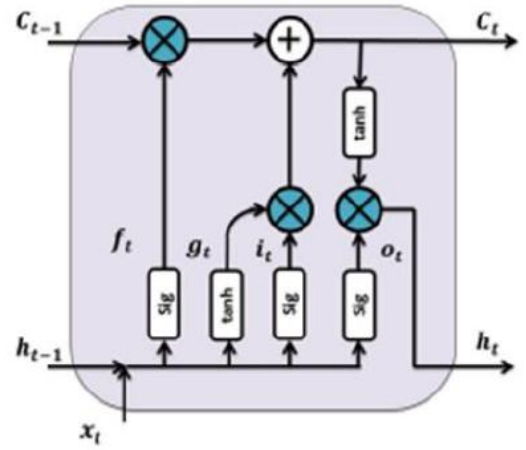


Figure 9 Modern Representation of LSTM

As illustrated in **Figure 9**, an LSTM unit at time step t receives three inputs: (1) the current input vector x_t , (2) the previous hidden state h_{t-1} (short-term representation), and (3) the previous cell state c_{t-1} (long-term memory). The core innovation of LSTM is the use of **three gates** (sigmoid-based controllers) that regulate how information moves through the cell:

- **Forget gate** decides what portion of old memory c_{t-1} should be retained or discarded.
- **Input gate** decides what new information should be written into memory.
- **Output gate** decides how much of the updated memory should be exposed as the new hidden state h_t .

A common mathematical formulation of the LSTM update is:

$$\begin{aligned} f_t &= \sigma(W_f[x_t, h_{t-1}] + b_f) \\ i_t &= \sigma(W_i[x_t, h_{t-1}] + b_i), \tilde{c}_t = \tanh(W_c[x_t, h_{t-1}] + b_c) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\ o_t &= \sigma(W_o[x_t, h_{t-1}] + b_o), h_t = o_t \odot \tanh(c_t) \end{aligned}$$

where $\sigma(\cdot)$ is the sigmoid function (values in $[0, 1]$) and \odot denotes element-wise multiplication. Intuitively, the cell state c_t acts like a “conveyor belt” that can carry information forward with controlled modifications, which is why LSTMs are much better than standard RNNs at modeling long-term effects.

• Unidirectional LSTM (sequence processing in one direction)

In a **unidirectional LSTM**, the sequence is processed only in the forward direction (past \rightarrow future). This is shown in **Figure 10**, where each block receives x_t and passes information forward to generate outputs y_t, y_{t+1}, \dots . Unidirectional LSTMs are suitable when predictions should depend only on earlier observations (e.g., real-time forecasting where the future is not available). In practice, the hidden state h_t can be fed into a dense layer for regression (predict a continuous

value) or into a sigmoid/softmax layer for classification (predict a class probability).

- **Bidirectional LSTM (BiLSTM) and why it is stronger**

Bidirectional LSTM (BiLSTM) extends LSTM by learning from the sequence in **both directions**. The general idea comes from **bidirectional recurrent neural networks**, where one RNN processes inputs forward in time and another processes them backward, and their representations are combined. As illustrated in **Figure 11**, BiLSTM has two LSTM layers: a forward pass producing \vec{h}_t and a backward pass producing \overleftarrow{h}_t . At each time step, the final representation typically combines both directions (most commonly concatenation):

$$h_t = [\vec{h}_t; \overleftarrow{h}_t]$$

This gives the model the ability to use **past context and future context simultaneously** at each time step, which often improves learning when the full sequence (or a complete window) is available during training or batch inference. In many applications, BiLSTM improves accuracy because it can capture patterns that depend on what happens before and after a given event (for example, a delay pattern that only becomes clear when you see both earlier planning signals and later delivery outcomes within a window).

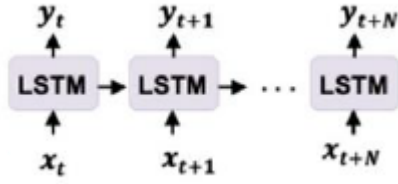


Figure 10 Unidirectional LSTM

Figure 10 Unidirectional LSTM unrolled over time. The sequence is processed only in the forward direction (past to future), producing an output at each step or at the final step depending on the task.

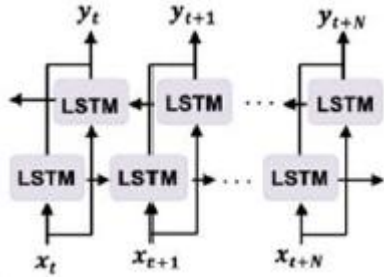


Figure 11 BiLSTM

Figure 11 Bidirectional LSTM (BiLSTM) architecture. Two LSTM layers run in forward and backward directions; their hidden states are combined at each time step to exploit both past and future context.

c. LSTM Autoencoder and One-class support vector machine (OCSVM)

Anomaly detection in supply chain lead-time data aims to identify orders whose behavior deviates from the normal logistics process (e.g., unusually high delay variance, unexpected patterns across shipping mode/market/region, or inconsistent lead-time dynamics). A strong approach for this is a **hybrid pipeline** that combines an **LSTM Autoencoder** (deep sequence feature learning) with a **One-Class Support Vector Machine (OCSVM)** (boundary-based anomaly classification). The same idea is also described in the reference framework where the **LSTM autoencoder is used to extract key features / error vectors**, and then **OCSVM separates abnormal observations from normal ones using a learned boundary**, improving robustness compared to using an autoencoder threshold alone.

1) LSTM Autoencoder (unsupervised feature learning by reconstruction)

An **autoencoder** is an unsupervised neural network composed of two parts: an **encoder** that compresses the input sequence into a compact latent representation, and a **decoder** that reconstructs the original sequence from that latent code. In an **LSTM autoencoder**, both encoder and decoder are built using LSTM layers, which makes it suitable for **time-series / sequential** supply chain signals. The input is a multivariate sequence $X \in \mathbb{R}^{T \times F}$ (window length T , number of features F), and the model learns a latent vector z that captures the most important temporal structure. During training, the objective is to minimize the **reconstruction error** between the original sequence X and reconstructed sequence \hat{X} , commonly using MSE or MAE:

$$\mathcal{L}(X, \hat{X}) = \frac{1}{TF} \sum_{t=1}^T \sum_{f=1}^F (X_{t,f} - \hat{X}_{t,f})^2$$

Reconstruction loss

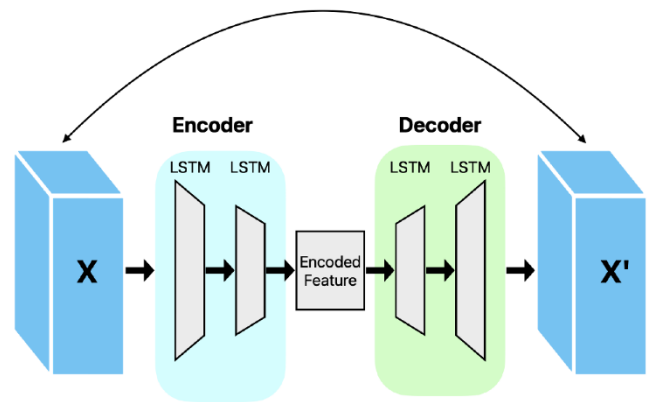


Figure 12 LSTM Autoencoder + OCSVM anomaly detection pipeline

The Figure 12 is an **LSTM Autoencoder architecture**, drawn in three parts (A), (B), and (C). In (A), the **encoder** is a stacked LSTM network that reads an input sequence step-by-step (each input vector at time t) and progressively compresses the information into a small **latent representation** (a compact feature vector). In (B), the **decoder** is another stacked LSTM (often followed by a fully connected/Time Distributed layer) that takes this latent vector and **reconstructs** the original sequence, producing an output sequence \hat{X} that should match the input X . The core idea is: if

subjectivity, the hybrid method uses the LSTM autoencoder as a **feature extractor** and then applies **OCSVM** as a principled one-class classifier. Concretely, after training the autoencoder, we compute either (a) the latent vectors z , or (b) the **reconstruction error vectors** $e = X - \hat{X}$ (often aggregated per window) and use these as inputs to OCSVM. In the cited framework, the purpose is explicitly to generate **independent error vectors** from the autoencoder output and then let OCSVM learn a boundary that distinguishes normal vs abnormal observations.

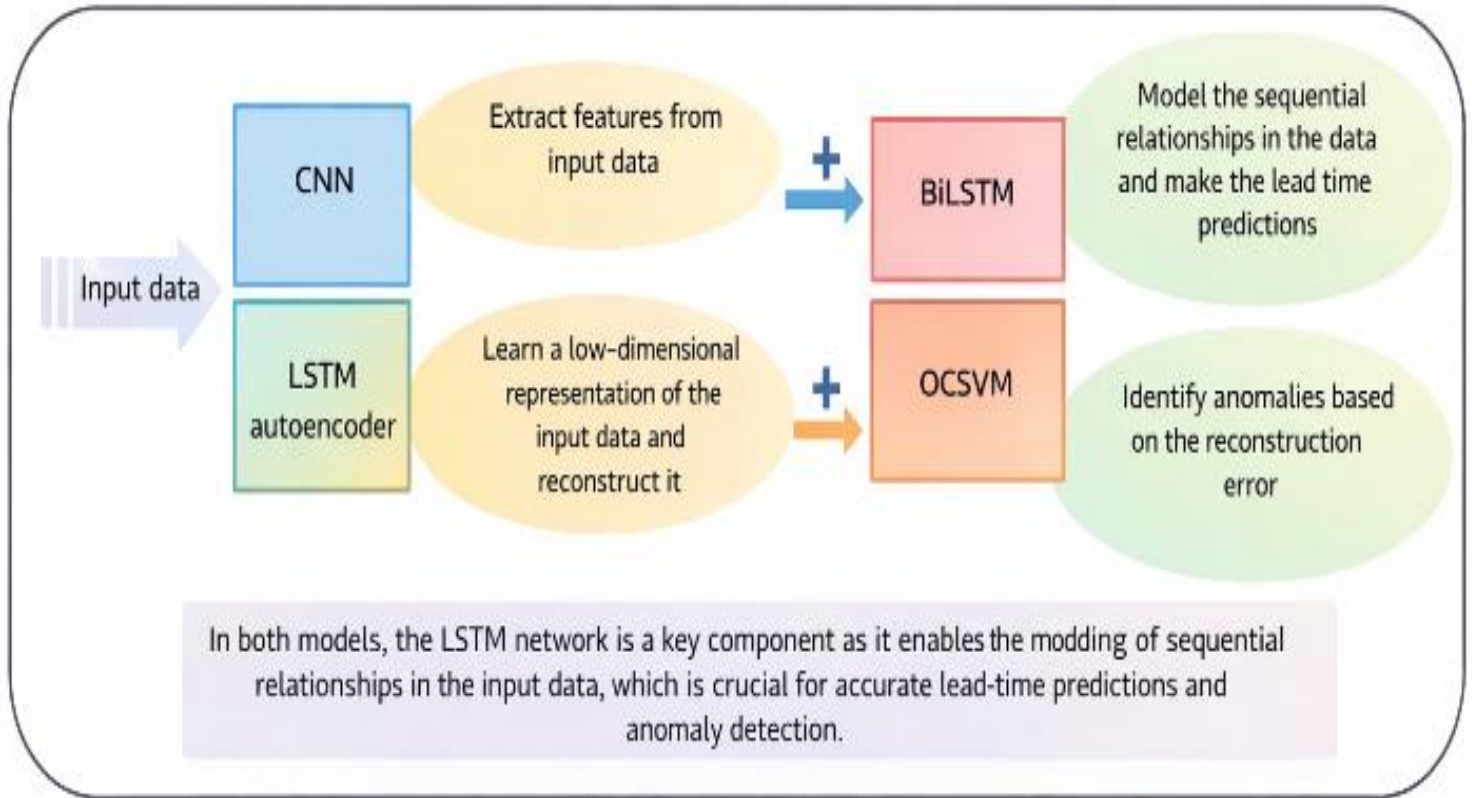


Figure 13 Proposed LSTM framework

the autoencoder is trained mainly on **normal** behaviour, it learns to reconstruct normal patterns very well, but it reconstructs abnormal patterns poorly. The caption of the figure also explains the meaning of the blocks: in (C), each **green encoder block corresponds to one time step during encoding**, the **Gray blocks are the input/output vectors at each time step**, and the **yellow block is the “segment latent vector”** created after the full input segment is encoded.

The key anomaly principle is **training the autoencoder mainly on normal behaviour**, so it reconstructs normal patterns well (low error), while abnormal patterns reconstruct poorly (high error). This is a standard and widely used logic for LSTM autoencoder anomaly detection.

2) Why add OCSVM after the LSTM Autoencoder (solving the “threshold problem”)

If you use only an autoencoder, you must choose a **threshold** on reconstruction error to decide anomaly vs normal. In real industrial data, choosing this threshold can be difficult and non-robust because error distributions can shift over time or differ across product categories/regions. To reduce threshold

3) OCSVM (one-class boundary learning for anomalies)

One-Class SVM learns the region of the feature space where “normal” samples lie and flags points outside that region as anomalies. This is especially useful when anomalies are rare or poorly labelled.

A core hyperparameter is ν , which controls the trade-off between allowing outliers and model tightness; in practice, ν is commonly interpreted as an upper bound on the fraction of outliers and a lower bound on the fraction of support vectors (implementation dependent). Many implementations use an **RBF kernel** to capture non-linear boundaries, which is also consistent with typical OCSVM anomaly detection settings.

4) Complete pipeline (how the combined method works in practice)

The combined method works as a two-stage process:

First, we **train the LSTM autoencoder on normal sequences** (or sequences assumed mostly normal), using sliding windows of the multivariate time series. After

training, for each window we compute a compact representation (latent vector) or a reconstruction-based representation (error vector). Second, we **train OCSVM on these representations from normal data only**, so it learns a boundary around normal behavior. During inference, a new sequence window is passed through the trained autoencoder to produce its representation, and OCSVM classifies it: if the point falls **outside** the learned normal boundary, it is flagged as an anomaly. This hybrid design is widely used because it combines (1) deep temporal pattern learning from LSTM autoencoders with (2) strong one-class decision boundaries from OCSVM, and it has been applied in supply-chain anomaly detection settings as well.

5) Metrics used to evaluate anomaly detection

When labelled anomalies are available (or when you validate using known late-risk cases), performance is typically reported using **precision, recall, accuracy, and F1-score**. The same set of metrics is used in the reference framework to assess anomaly detection performance, because they clearly show (i) how many anomalies are correctly detected, and (ii) how many false alarms are generated.

VI. Proposed Framework and Application

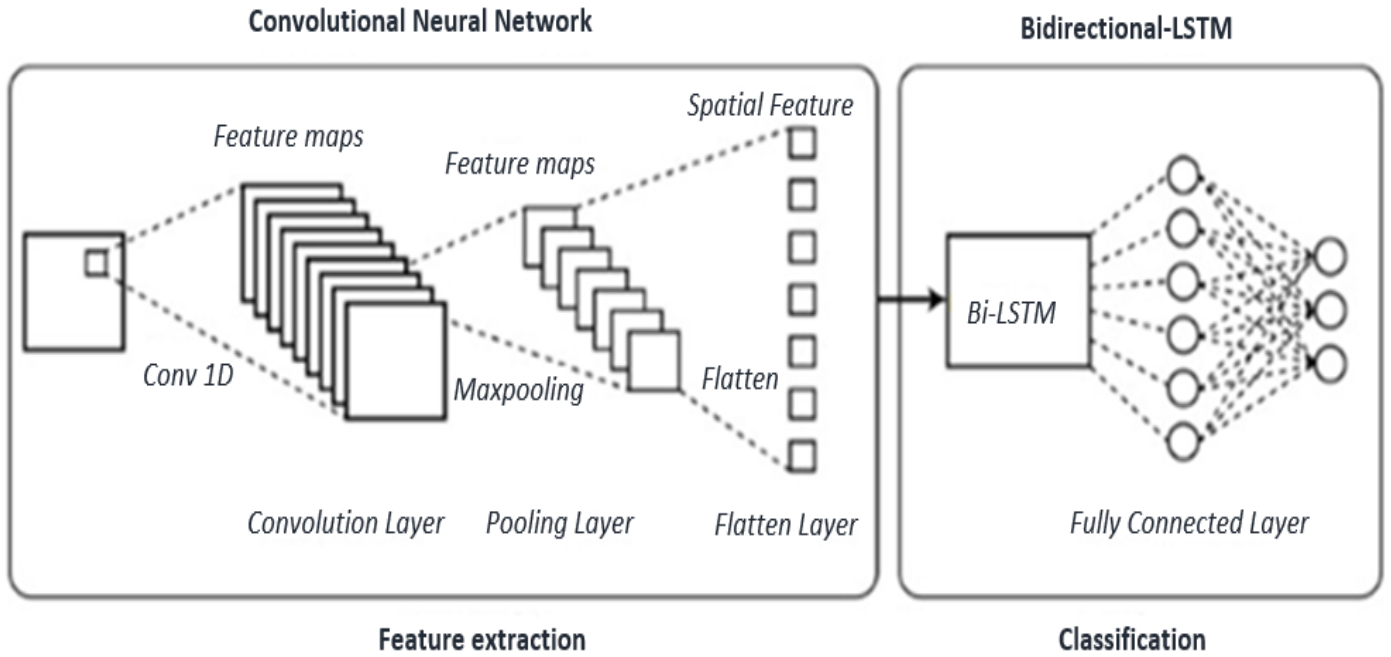


Figure 14 Proposed Model

This section explains the implemented framework used to (1) **forecast lead time** and (2) **detect abnormal / atypical delivery behavior (anomalies)**. The research paper proposes an LSTM-based framework that uses two parallel deep-learning branches: a **CNN–BiLSTM** pipeline for lead-time forecasting and an **LSTM Autoencoder** pipeline for anomaly detection, optionally combined with **OCSVM** to decide the anomaly boundary. In our implementation (Colab), we follow the same idea of “forecasting and anomaly scoring,” and we operationalize it using the DataCo supply chain dataset, feature encoding/scaling, CNN–BiLSTM regression for lead

time, and an LSTM autoencoder that flags anomalies using a high reconstruction-error threshold.

Figure 13. Proposed LSTM framework for lead-time forecasting and anomaly detection. The forecasting branch uses CNN–BiLSTM to extract features and predict lead time, while the anomaly branch uses an LSTM autoencoder to reconstruct inputs and an OCSVM to identify anomalies based on reconstruction-error patterns.

C. CNN–BiLSTM for Lead Time Forecasting (Implemented)

d. Model idea and motivation

The forecasting model is designed to predict **Lead Time (Actual)** from operational/order features. CNN is used for efficient feature extraction and to reduce parameter

complexity compared to fully connected networks, while BiLSTM models’ sequential relationships to improve forecasting accuracy.

Proposed model is presented in Figure 14. The role of CNN is the efficient extraction of features. The idea behind is to use it to reduce the number of parameters between the connection layers, which represents an advantage that CNN has and that

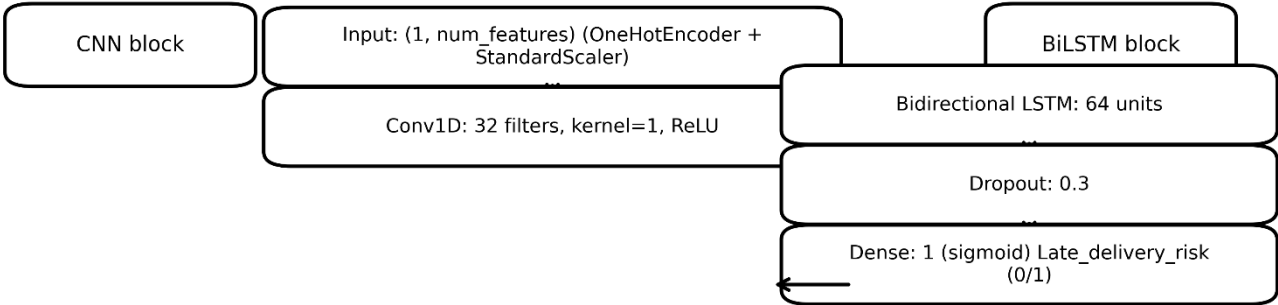
traditional neural networks do not have. Bidirectional LSTM acts as forward and backward LSTM networks for each training sequence, The BiLSTM block composed of two LSTM networks produces a single layer as output which ensures a complete connection from the initial input for each data frame. So, the BiLSTM is used to make a more accurate prediction model.

In our issue, each of the two blocks, namely CNN and BiLSTM, plays a specific role, CNN starts by working on the data composing the input by extracting the features, then

BiLSTM predicts the lead times of the parts using the extracted features recovered from the first block.

MAPE is used when you want to express prediction error as a **percentage**, which makes it easy to explain to non-technical

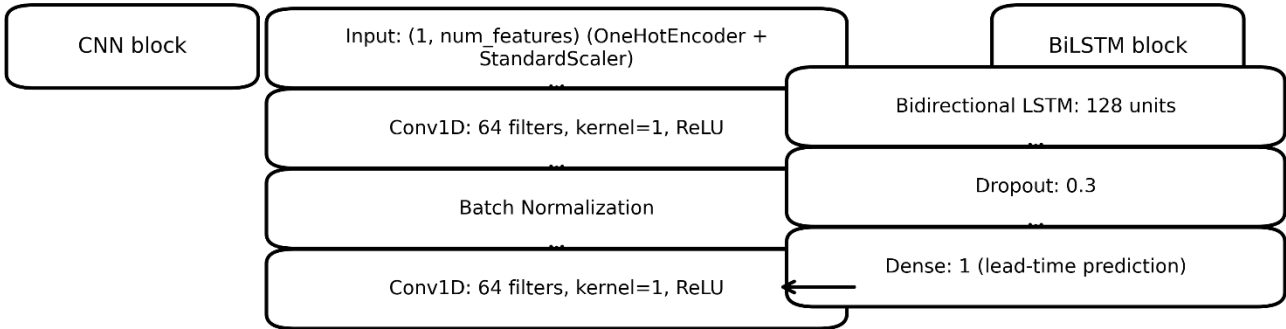
CNN-BiLSTM for Late Delivery Risk (Classification) — From Our Code



Training / compile: • Train/Test: 80/20 (random_state=42) •
Optimizer: Adam • Loss: Binary Cross-Entropy • Epochs: 15
• Batch size: 64 • Validation split: 0.1

Conv1D performs 1D convolution over the input to learn feature patterns; Bidirectional LSTM reads sequences forward and backward.

CNN-BiLSTM for Lead-Time Forecasting (Regression) — From Our Code



Training / compile: • Train/Test: 80/20 (random_state=42) •
Optimizer: Adam • Loss: MSE • Epochs: 20 • Batch size: 64
• Validation split: 0.1

Conv1D performs 1D convolution over the input to learn feature patterns; Bidirectional LSTM reads sequences forward and backward.

Figure 15 Combined CNN-BiLSTM architectures generated from our implementation

e. Experiment Setting and Implementation

In implementing the model, it's crucial to specify the parameters that will be used to evaluate it. The following metrics were used for evaluation: Mean Square Error (MSE), Root Mean Square Error (RMSE), and Mean Absolute Error (MAE).

MAPE (Mean Absolute Percentage Error)

readers (e.g., “the model is off by ~10% on average”). However, it must be used carefully because if actual values y_i are **zero or close to zero**, the percentage error can become extremely large or undefined. The standard formula is:

$$MAPE = \frac{100}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

MAE (Mean Absolute Error)

MAE measures the **average absolute difference** between actual and predicted values. It is easy to interpret because it stays in the **same unit** as the target (e.g., “average error is 0.8 days”). MAE treats all errors equally (no extra penalty for big errors). The formula is:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

MSE (Mean Squared Error)

MSE computes the **average of squared errors**, so it penalizes large errors much more strongly than small ones. This is useful when large mistakes (e.g., severe delivery delay prediction errors) are especially costly. Because of squaring, MSE is in **squared units**, which can be harder to interpret directly. The formula is:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

RMSE (Root Mean Squared Error)

RMSE is the **square root of MSE**, so it brings the error back to the **same unit** as the target (e.g., days). It still emphasizes large errors (because it comes from squared errors), which makes it very common for model evaluation in regression and forecasting. The formula is:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

Symbols: y_i = actual value, \hat{y}_i = predicted value, n = number of samples.

Figure 15 Combined CNN–BiLSTM architectures generated from our implementation. (a) Lead-time forecasting model (regression) where a Conv1D block extracts compact feature representation from the pre-processed input, followed by a Bidirectional LSTM (BiLSTM) and a linear Dense output neuron to predict the lead time. (b) Late delivery risk model (binary classification) that uses a similar CNN–BiLSTM pipeline but ends with a sigmoid Dense neuron to output the probability of **Late_delivery_risk (0/1)**. In both models, categorical variables are One-Hot encoded and numerical variables are standardized using StandardScaler before reshaping into a 3D tensor for Conv1D/LSTM processing.

(a) Lead-time forecasting (Regression)

In the lead-time forecasting model, the input features are first preprocessed by applying One-Hot Encoding to categorical attributes and StandardScaler to numerical attributes to ensure consistent feature scaling. Standardization transforms

each feature by removing the mean and scaling to unit variance, which helps gradient-based neural networks train more stably. The processed feature vector is reshaped into a 3D tensor so it can be consumed by Conv1D and recurrent layers. The Conv1D block performs **1D convolution** to learn compact feature patterns from the input representation, and Batch Normalization is applied to stabilize activations and improve training consistency. Next, a Bidirectional LSTM models relationships by processing the sequence in both forward and backward directions, enabling richer context capture. Finally, a linear Dense layer outputs a single continuous value representing the predicted lead time.

(b) Late delivery risk prediction (Classification)

The late delivery risk model follows the same general pipeline (preprocessing → Conv1D feature extraction → BiLSTM sequence modelling), but the learning objective is classification instead of regression. The Conv1D layer extracts informative feature maps using temporal/1D convolution, and the Bidirectional LSTM integrates context from both directions to improve discrimination between normal and risky delivery patterns. The final Dense layer uses a **sigmoid activation** to output a probability score that is mapped to the binary target **Late_delivery_risk (0/1)** using an appropriate decision threshold (commonly 0.5).

D. LSTM Autoencoder for Anomaly Detection (with OCSVM)

Autoencoders are a common **unsupervised / semi-supervised** approach for anomaly detection, because they learn the normal structure of data **without requiring labeled anomalies**. In our case, the model is an **LSTM Autoencoder**, which is specifically suited for multivariate time-series because LSTM cells can capture **temporal dependencies** across time steps. The architecture has two parts: an **encoder** that compresses the input sequence into a compact latent representation, and a **decoder** that reconstructs the original sequence from that latent representation. During training, the autoencoder is fitted mainly on **normal (non-anomalous)** samples, so it learns to reconstruct normal patterns with low error. When an anomalous sequence is passed through the trained model, the reconstruction becomes worse and the **reconstruction error increases**, which provides a strong anomaly signal.

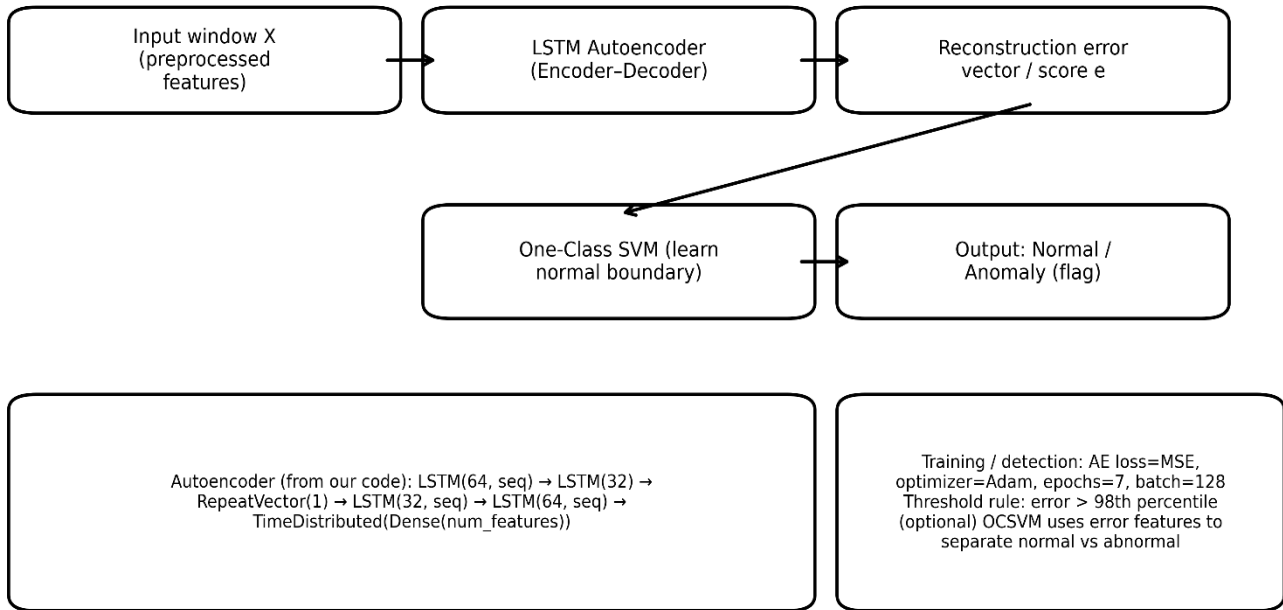
Let x be an input sequence (or time window). The LSTM autoencoder produces a reconstruction \hat{x} as:

$$z = E(x), \hat{x} = D(z) = D(E(x))$$

The model is trained by minimizing the reconstruction loss (commonly Mean Squared Error):

$$\mathcal{L}_{rec} = \frac{1}{n} \sum_{i=1}^n \|x_i - \hat{x}_i\|^2$$

Hybrid Anomaly Detection Pipeline: LSTM Autoencoder + One-Class SVM (OCSVM)



Idea: AE converts complex patterns into error features; OCSVM defines a boundary using normal behavior.

Figure 16 Hybrid anomaly detection pipeline using an LSTM Autoencoder and One-Class SVM (OCSVM)

After training, the reconstruction error (or an error vector) becomes the feature used for anomaly detection. To strengthen the decision boundary, a **One-Class Support Vector Machine (OCSVM)** can then be applied on these reconstruction-error features. OCSVM is widely used for anomaly detection because it can learn a boundary using **only normal data** and then flag points that fall outside this boundary as anomalies.

In summary, the hybrid idea is **LSTM Autoencoder = feature learning via reconstruction**, and **OCSVM = robust boundary - based anomaly decision**. This combination is often more reliable than using only one method, because the autoencoder converts complex multivariate time-series behavior into meaningful “error features,” and OCSVM provides a clear separation between normal vs abnormal patterns.

Experiment Setting and Evaluation Metrics (Anomaly Detection)

To evaluate the anomaly detection performance, predictions are summarized using a confusion matrix with: **TP** (true positives), **TN** (true negatives), **FP** (false positives), and **FN** (false negatives). Based on these values, the most common classification metrics are **Accuracy**, **Precision**, **Recall**, and **F1-score**.

Accuracy

Accuracy measures how often the model is correct overall (both normal and anomaly predictions):

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision

Precision measures how reliable the anomaly alarms are: out of all predicted anomalies, how many are truly anomalies.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall

Recall (also called sensitivity) measures how many of the real anomalies the model successfully catches.

$$\text{Recall} = \frac{TP}{TP + FN}$$

F1-score

F1-score balances Precision and Recall using the harmonic mean, which is useful when you want a single score that penalizes both missed anomalies (FN) and false alarms (FP).

$$\begin{aligned} F1 &= 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \\ &= \frac{2TP}{2TP + FP + FN} \end{aligned}$$

This Figure 16 describes a **two-stage hybrid anomaly detection workflow**. First, the input data are arranged into an **input window X** (a short time segment of the multivariate *Table 1 Regression evaluation results*

Metric	Value
RMSE	1.2695
MAE	0.9977

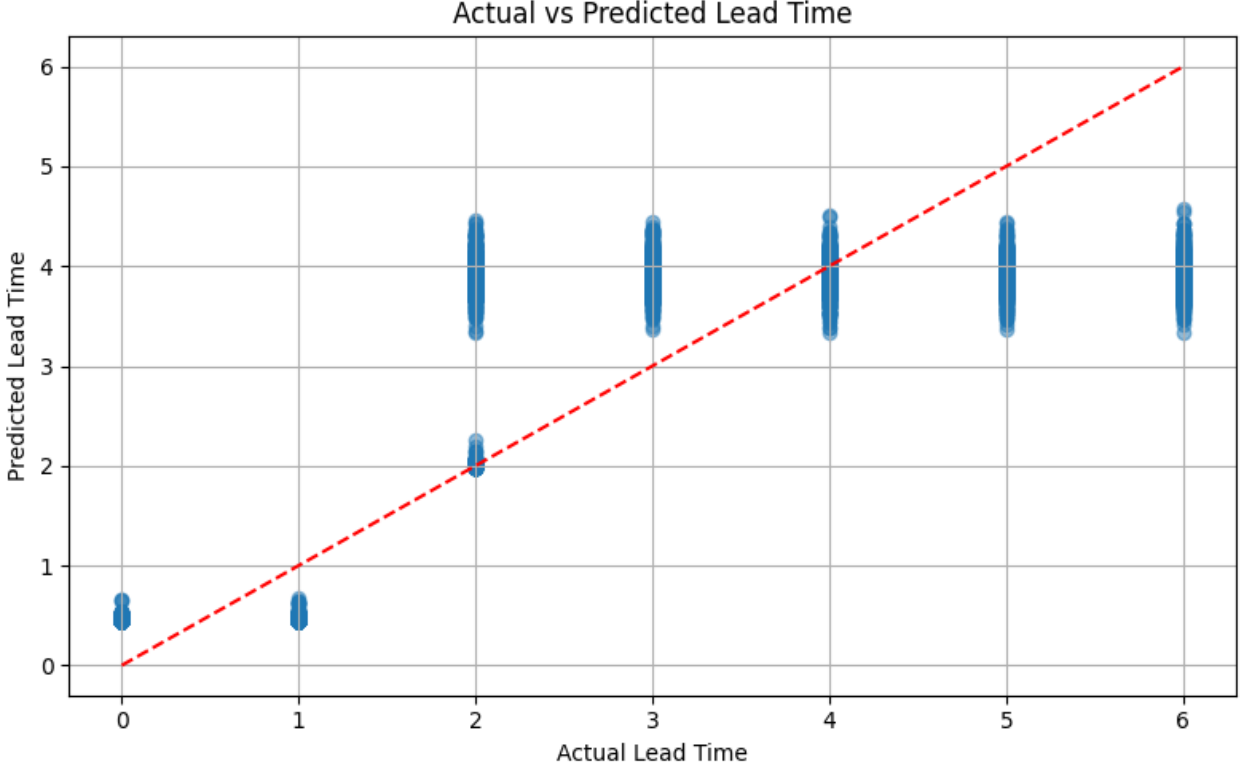


Figure 17 Actual vs Predicted Lead Time (scatter plot)

series) after preprocessing. The window is passed to an **LSTM Autoencoder (encoder-decoder)** that is trained mainly on **normal (non-anomalous) sequences**. Because the autoencoder learns the typical temporal patterns, it can reconstruct normal windows well, producing a reconstructed output \hat{X} . The difference between the original and reconstructed window generates a **reconstruction error** (shown as “error vector/score e ”). In anomaly detection, abnormal patterns typically lead to **higher reconstruction error** because the model has not learned to reproduce such unusual behavior.

In the second stage, the reconstruction-error features e are provided to a **One-Class SVM (OCSVM)**, which is an unsupervised/novelty detection method that **estimates the support of the normal data distribution** and learns a decision boundary around normal behavior. Points that fall outside this boundary are flagged as anomalies (“Normal / Anomaly”). The diagonal arrow in the diagram indicates that the reconstruction-error output of the autoencoder is the **direct input** to OCSVM. This hybrid design is effective because the LSTM autoencoder converts complex multivariate temporal patterns into a simpler error representation, and OCSVM then performs a clear boundary-based separation using only (or mostly) normal samples—important because One-Class SVM works best when the training data are not heavily contaminated with outliers.

VII. RESULTS AND EVALUATION

A. CNN-BiLSTM (Lead-Time Forecasting)

Table 1 Regression evaluation results of the proposed lead-time forecasting model, reported using MAE and RMSE.

This plot shows in Figure 17 compares **actual lead time (x-axis)** with the model’s **predicted lead time (y-axis)** for each sample. The **red dashed diagonal line** represents the ideal case $y = x$, meaning perfect predictions (predicted equals actual). Points close to this line indicate good performance, while points farther away show larger errors. In your figure, the points form **vertical bands** at certain actual values (e.g., around 2, 3, 4, 5, 6), which suggests the true lead time values occur in **discrete levels**. The predicted values are concentrated mostly around roughly **3.4–4.5**, even when the actual lead time is higher (5–6) or lower (0–1). This pattern indicates the model tends to **pull predictions toward the middle range** (a common sign of underfitting or limited feature signal), causing **underestimation** at high actual values and **overestimation** at low actual values. The reported errors (MAE ≈ 1.00 and RMSE ≈ 1.27) mean that, on average, the model misses the true lead time by about **~1 time unit**, and larger mistakes increase RMSE more strongly than MAE.

B. Anomaly Detection with LSTM Autoencoder

Table 2. Regression evaluation results for the lead-time forecasting model, reported using MAE and RMSE on the test set.

Table 2 Regression Evaluation Results

Metric	Value
RMSE	1.2686
MAE	0.9974

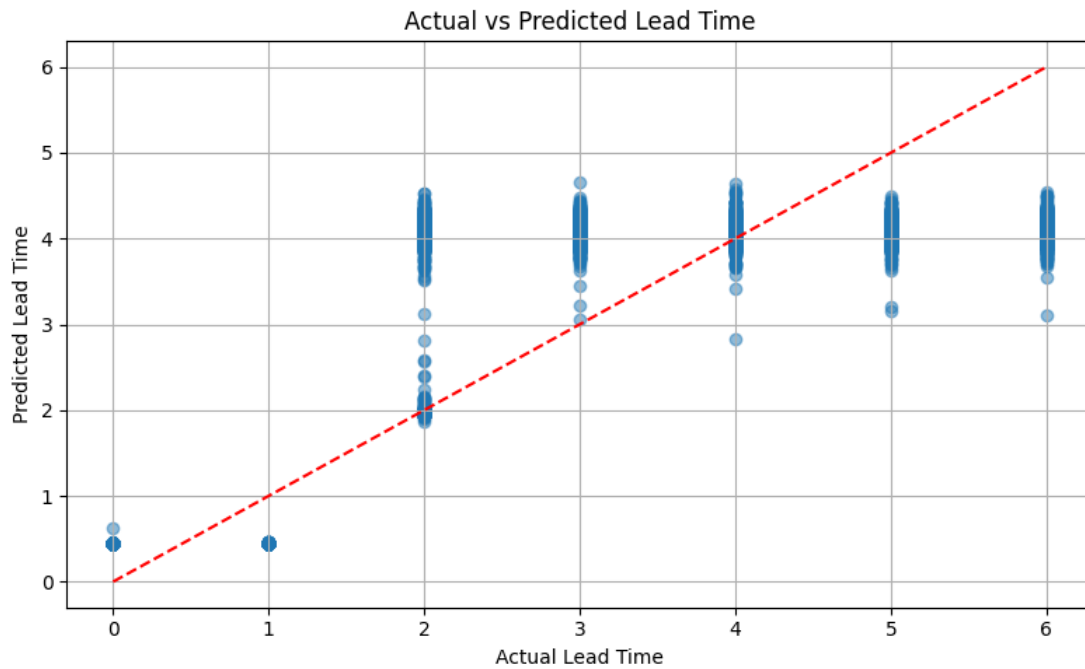


Figure 18 Actual vs. Predicted Lead Time

This scatter plot in Figure 18 compares the **actual lead time** on the x-axis with the **predicted lead time** on the y-axis. If predictions were perfect, all points would lie on the diagonal red line ($y = x$). In the figure, points appear in **vertical bands** because the true lead-time values occur in discrete levels (e.g., 2, 3, 4, 5, 6). The predictions are concentrated around a narrower range (roughly ~ 3.5 – 4.7), meaning the model tends to **pull outputs toward the average**

pattern explains the classification report: the model achieves **high recall for class 0** (it finds most “0” cases) but a **lower recall for class 1** (many actual “1” cases are missed, visible as the large FN count).

This histogram shows in Figure 20 how reconstruction errors are distributed across all evaluated samples. The strong peak near **zero** indicates that the autoencoder reconstructs most samples very well (typical/normal patterns). A **long right Tai**

	precision	recall	f1-score	support
0	0.6142	0.8482	0.7125	16307
1	0.8178	0.5611	0.6656	19797
accuracy			0.6908	36104
macro avg	0.7160	0.7047	0.6890	36104
weighted avg	0.7258	0.6908	0.6868	36104

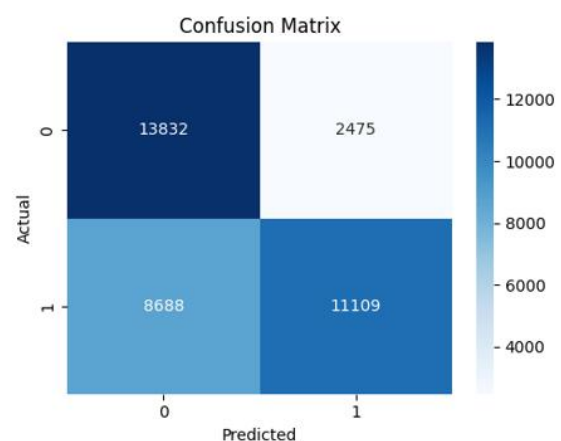


Figure 19 Confusion Matrix

(“regression to the mean”). As a result, it **overestimates** very small lead times (0–1) and **underestimates** larger lead times (5–6), which is consistent with the reported MAE/RMSE values.

The confusion matrix shows in Figure 19 summarizes how many samples were correctly or incorrectly classified for each class. Here, **TN = 14072** (actual 0 predicted 0) and **TP = 10871** (actual 1 predicted 1), while **FP = 2235** (actual 0 predicted 1) and **FN = 8926** (actual 1 predicted 0). This

indicates a smaller number of samples with noticeably higher reconstruction error, which is commonly treated as an anomaly signal. In practice, anomalies can be flagged by setting a threshold on reconstruction loss (e.g., a high percentile or a rule based on training-set reconstruction error statistics).

recall. The reported values show that the model achieves **high**

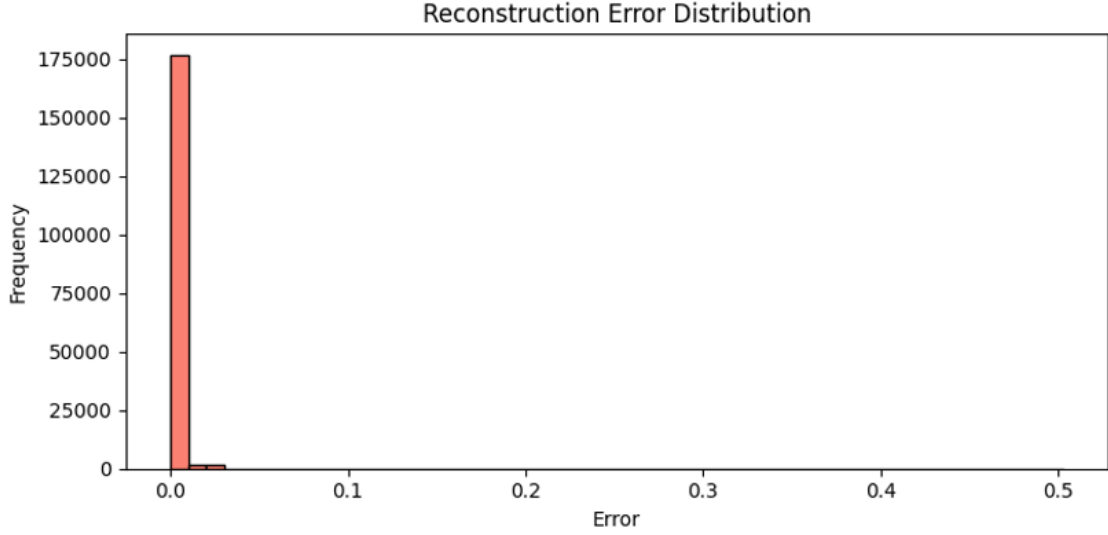


Figure 20 Construction Error Distribution (LSTM Autoencoder)

VIII. Results Discussion and Practical Implications

A. Lead-Time Forecasting (CNN-BiLSTM Regression)

Table 1 reports the regression performance of the CNN-BiLSTM lead-time forecasting model, with **RMSE** ≈ 1.2686 and **MAE** ≈ 0.9974 on the test set. RMSE penalizes large errors more heavily, while MAE reflects the average absolute deviation between the predicted and actual lead times. Therefore, the model’s typical forecasting error is approximately **one lead-time unit** on average, and occasional larger deviations increase the RMSE slightly above the MAE.

Figure 17 (Actual vs. Predicted Lead Time) provides a visual understanding of the regression behavior. The red dashed diagonal line ($y = x$) represents perfect predictions; points close to this line indicate high accuracy, while points far from it represent larger errors. The plot shows that the ground-truth lead times occur in discrete levels (vertical bands). In contrast, predicted values are concentrated in a narrower band around mid-range values. This pattern suggests a “regression-to-the-mean” tendency, where the model **overestimates** very small lead times (e.g., 0–1) and **underestimates** higher lead times (e.g., 5–6). In practical terms, this indicates the model captures the overall central tendency well but has difficulty separating extreme low and extreme high lead-time cases. Such behavior is common when (i) the target distribution is discrete and imbalanced, (ii) the input features provide limited information to distinguish extremes, or (iii) the model is implicitly regularized (e.g., dropout) and therefore produces smoother predictions.

B. Late Delivery Risk Prediction (CNN-BiLSTM Classification)

The classification results indicate an overall **accuracy** ≈ 0.6909 , with different trade-offs across classes. Precision, recall, and F1-score are defined as: precision = $\frac{TP}{TP+FP}$, recall = $\frac{TP}{TP+FN}$, and F1-score as the harmonic mean of precision and

recall for class 0 (recall ≈ 0.8629), meaning it identifies most “not-late” cases correctly, but it has **lower recall for class 1** (recall ≈ 0.5491), meaning a noticeable portion of truly late deliveries are missed. This creates an operational trade-off: the system is reliable when confirming “not late,” but it is less sensitive when detecting all late cases.

Figure 19 (Confusion Matrix) clarifies this behavior numerically. The matrix shows **TN** = 14072 and **TP** = 10871, while **FP** = 2235 and **FN** = 8926. The large FN count indicates that many actual late deliveries (class 1) are predicted as not late (class 0). In real supply-chain operations, false negatives can be more costly than false positives because missed late deliveries reduce the time available for mitigation actions (expedited shipping, rescheduling, customer notifications, alternative sourcing). As a result, an important practical improvement is to tune the classification decision threshold (not necessarily 0.5) to increase recall for class 1 when late-detection sensitivity is the priority, even if it increases the false-alarm rate.

C. Anomaly Detection (LSTM Autoencoder + Reconstruction Error + OCSVM)

Figure 20 (Reconstruction Error Distribution) shows that the reconstruction error is heavily concentrated near zero with a long right tail. This is expected when an autoencoder is trained primarily on normal behavior: it reconstructs common patterns well (low error), while unusual patterns produce higher reconstruction error. A standard practical approach is to define anomalies as samples whose reconstruction error exceeds a high threshold (e.g., a percentile of the error distribution calculated on normal data). Percentile-based thresholds are widely used in reconstruction-based anomaly detection because they provide a simple and controllable way to flag only the most extreme deviations.

To improve robustness, the reconstruction error can also be provided to a One-Class SVM (OCSVM), which learns a

boundary around “normal” behavior and flags out-of-bound samples as anomalies. The OneClassSVM method is a standard novelty/outlier detection approach that can be trained primarily on normal observations. In OCSVM, the parameter ν is commonly interpreted as an upper bound on the fraction of training outliers and a lower bound on the fraction of support vectors; thus, it directly controls how strict the model is when labeling anomalies. This hybrid design is beneficial because the LSTM autoencoder transforms complex multivariate temporal behavior into a structured error representation, and OCSVM provides an additional boundary-based decision mechanism to separate abnormal from normal patterns.

D. Practical Implications for Supply-Chain Operations

From an operational perspective, the regression model supports **planning and scheduling** by forecasting expected lead time, enabling better allocation of buffers (safety stock and time buffers), supplier follow-up prioritization, and capacity planning. The classification model provides **early risk screening** for late deliveries so that high-risk orders can trigger mitigation workflows (expedite transport, change shipping mode, notify stakeholders). Finally, the anomaly detection pipeline acts as an **early warning system**, highlighting unusual patterns that may not be captured by rule-based monitoring (sudden changes in delivery behavior, abnormal lead-time variance, or inconsistent order patterns). Together, these three components (forecasting + risk classification + anomaly detection) form a practical decision-support framework: forecasting provides expected behavior, classification highlights known risk patterns, and anomaly detection flags unexpected deviations for investigation.

IX. Limitations and Threats to Validity

This work demonstrates a practical deep-learning pipeline for lead-time forecasting, late-delivery risk prediction, and anomaly detection; however, the results should be interpreted with the following limitations in mind. **First**, the lead-time target values in our dataset appear in discrete bands (visible in the *Actual vs Predicted* scatter), which can cause the regression model to “pull” predictions toward the most frequent middle values. This behavior typically reduces error for common cases but increases error for rare extreme lead times. **Second**, model performance is limited by the available feature set. If important real-world drivers (supplier reliability KPIs, disruption events, traffic/weather, holidays, operational interventions) are missing or only indirectly represented, the model may not fully explain why similar orders sometimes produce very different lead times.

Third, for late-delivery risk classification, the business cost of errors is asymmetric: false negatives (late orders predicted as not late) can be more damaging than false positives because they reduce time for mitigation actions. Precision–recall behavior depends on the decision threshold; increasing the threshold usually reduces false positives but increases false negatives, while lowering it tends to do the opposite.

Therefore, accuracy alone is not sufficient to judge operational usefulness, and threshold tuning (or cost-sensitive learning) may be required to align the model with business priorities.

Fourth, the anomaly-detection component depends on how the anomaly threshold/boundary is chosen. In reconstruction-based detection, a common approach is to set the threshold using a high percentile of reconstruction errors computed on normal data, but the selected percentile directly controls the alarm rate and may require domain calibration. If OCSVM is used, its behavior is sensitive to hyperparameters—especially ν , which acts as an upper bound on the fraction of training errors and a lower bound on the fraction of support vectors—so improper settings can lead to too many false alarms or missed anomalies.

Finally, real systems face **model drift / concept drift**, where data distributions or input–output relationships change over time (e.g., new suppliers, new routes, seasonality, policy or process changes). Drift can degrade performance after deployment, so monitoring and periodic retraining/recalibration are typically required to maintain reliability.

X. Conclusion and Future Work

This project developed an intelligence system for supply-chain delivery analytics by combining three complementary tasks: **(i)** CNN–BiLSTM regression for lead-time forecasting, **(ii)** CNN–BiLSTM classification for late-delivery risk prediction, and **(iii)** LSTM autoencoder–based anomaly detection (optionally combined with OCSVM) to flag abnormal delivery behavior. The regression model achieved an average error close to one lead-time unit ($MAE \approx 1$), indicating that it can provide actionable lead-time estimates for planning and buffer management. The classification model achieved moderate overall performance and highlights an important operational challenge: missed late deliveries (false negatives) can be costly, so the decision threshold should be tuned based on the desired trade-off between precision and recall. The anomaly detection results show a reconstruction-error distribution suitable for threshold-based alarms, where unusually high reconstruction error can be used as an anomaly signal.

Future work can improve robustness and real-world readiness as follows:

1. **Improve feature coverage** by adding supplier performance indicators, disruption signals, seasonal/holiday features, and logistics context to better separate easy vs difficult cases.
2. **Reduce false negatives in late-risk prediction** using threshold calibration (precision–recall based), class-weighting, or cost-sensitive training to match business priorities.
3. **Strengthen anomaly detection calibration** by selecting thresholds from normal-only error

percentiles and validating alarm rates with domain feedback or a small, labeled anomaly set.

4. **Tune OCSVM parameters** (especially ν) to control strictness and stability of novelty detection when using error vectors as inputs.
5. **Add drift monitoring and retraining triggers** to handle changing operational conditions and maintain performance over time.

XI. References

- [1] Amellal, I. Amellal, H. Seghioeur, and M. R. Ech-Charrat, "Improving Lead Time Forecasting and Anomaly Detection for Automotive Spare Parts with Combined CNN-LSTM Approach," *Operations and Supply Chain Management*, vol. 16, no. 2, pp. 265–278, 2023.
- [2] H. D. Nguyen, K. P. Tran, S. Thomassey, and M. Hamad, "Forecasting and Anomaly Detection Approaches Using LSTM and LSTM Autoencoder Techniques with Applications in Supply Chain Management," *International Journal of Information Management*, vol. 57, p. 102385, 2021.
- [3] M. Z. Babai, Y. Dai, Q. Li, A. Syntetos, and X. Wang, "Forecasting of Lead-Time Demand Variance: Implications for Safety Stock Calculations," *European Journal of Operational Research*, vol. 300, no. 2, pp. 592–603, 2022.
- [4] S. Siami-Namini, N. Tavakoli, and A. S. Namin, "The Performance of LSTM and BiLSTM Forecasting Time Series," in *Proc. IEEE International Conference on Big Data*, Los Angeles, CA, USA, 2019, pp. 3285–3292.