DLSTM-SCM– A Dynamic LSTM-based Framework for Smart Supply Chain Management

Seyf Eddine Hasnaoui, Mohammed Amine Boudouaia, Samir Ouchani, Abdelatif Rahmoun

DLSTM-SCM– A Dynamic LSTM-Based Framework for Smart Supply Chain Management

Seyf Eddine HASNAOUI [a,b], Mohammed Amine BOUDOUAIA [b], Samir OUCHANI [b], and Abdelatif RAHMOUN [a]
[a] ESI Engineering School, Sidi bel Abbès, Algeria; [b] CESI LINEACT, Aix-en-Provence, France
{sehasnaoui, aboudouaia, souchani}@cesi.fr; a.rahmoun@esi-sba.dz

***Abstract*—**
In the retail industry, SCM holds significant importance as it ensures the efficient movement of goods from suppliers to customers. In this intricate and fast-paced environment, the availability of accurate information and data is crucial. The purpose of this paper is to develop a framework that enhances forecasting accuracy and efficiency in supply chain operations within the retail industry. By analyzing the latest research and advancements in the field, this paper seeks to contribute valuable insights into the potential of deep learning for supply chain management. The ultimate goal is to provide retailers with a reliable tool that empowers them to make informed decisions based on accurate predictions, thereby optimizing their supply chain operations and better meeting customer demands in the dynamic retail landscape. DLSTM-SCM, the framework developed in this paper, updates dynamically the deployed LSTM models to predict the upcoming day's sales using historical sales data in addition to statistical features like lagging and shifting to enhance forecasting precision. The efficacy of DLSTM-SCM is demonstrated through its performance on real benchmarks, where it yielded significant improvements compared to existing methods.

***Index Terms*—Time Series, Demand Forecasting, Supply Chain Management, RNN, Deep Learning, Sales Forecasting, LSTM.**

I. Introduction

With the advent of new technologies and the growing complexity of global trade. Supply Chains (SC) have become increasingly important and complex. The supply chain is the connected series of activities that is concerned with planning, coordinating, and controlling material, parts and finished goods from supplier to customer [1]. For example, the COVID-19 pandemic has highlighted the value of resilient supply chains and the requirement for organizations to be able to quickly adjust to changing conditions.

Existing solutions in SCM, including time series forecasting and demand forecasting used in [2] and [3], have made significant strides. However, they contend with notable limitations. They often struggle to capture intricate hierarchical patterns within SC datasets, which can be complex and multi-tiered. Additionally, these solutions frequently overlook the inclusion of crucial statistical features, compromising their predictive accuracy. Bridging these gaps represents a promising direction for enhancing the efficacy of Deep Learning (DL) applications in SCM. Future solutions should focus on addressing these limitations to optimize SC operations and decision-making effectively [4].

This paper tackles the limitations identified earlier in SCM solutions by proposing the implementation of a dynamic model that incorporates a broader set of features, enabling it to effectively track and decipher complex patterns. Its primary contribution is the development of **DLSTM-SCM**, a dynamic tool that effectively captures complex hierarchical dataset patterns. **DLSTM-SCM** leverages DL techniques and incorporates essential statistical features like lagging and shifting to deliver highly accurate forecasting results. Thus **DLSTM-SCM** empowering more precise decision-making and optimizing SC operations.

In the upcoming sections, we will cover key aspects of **DLSTM-SCM**. Section II reviews related work of using AI in the supply chain context. Section III details **DLSTM-SCM**. In Section IV, we present the run experiment results. Finally, we conclude the paper in Section V while providing the future directions related to **DLSTM-SCM**

II. Related Work

This section explores and discusses the utilization of deep learning approaches in the supply chain domain and their profound impact on decision-making and overall performance enhancements.

Amani and Sarkodie [5] automated meat production regulation and monitoring by leveraging Industry 4.0 principles and addressing Sustainable Development Goal 12. For meat supply chain management, they employed a classifier based on Deep Convolutional Neural Networks and Particle Swarm Optimization algorithms. The dataset, sourced from a Turkish store, was used to classify meat photos as fresh or rotting. Their method achieved 100% accuracy in meat authenticity classification, surpassing the 99.62% reported by Ulucan et al. [6].

Chong et al. [2] applied real-time series data to soft-Markov Decision Process based inventory management. Their numerical results highlight the effectiveness of Deep Reinforcement Learning for accurately forecasting product demand within a significant supply chain management system. The study's credibility was reinforced by the explicit depiction of stock level variations in response to demand fluctuations. Additionally, the experimental findings underscored the importance of entropy optimization, which enabled the agent to navigate episodes with extreme demand history variations.

Oyewola et al. [3] aimed to enhance SCM efficiency while reducing manual intervention and associated costs. They have demonstrated that Bayesian Optimization with the Tree Parzen Estimator and AllkNN can optimize deep learning models, such as LSTM and One Dimensional Convolutional Neural Network (1D-CNN). The obtained results indicated that combining the Tree Parzen Estimator with 1D-CNN and AllkNN can enhance the accuracy of supply shipment price datasets. The data samples were sourced from Kaggle, although it is worth noting that the industry sample may not fully represent the entire sector.

Tan et al. [7] examined deep learning software ecosystems, focusing on TensorFlow and PyTorch. Their analysis compared package usage within these ecosystems, revealing similarities, except for a higher prevalence of research-related projects in PyTorch. This suggests that PyTorch is preferred by academics due to its ease of use and suitability for rapid prototyping, while TensorFlow is more geared towards AI product development. Utilizing Generalized Additive Models, they uncovered a nonlinear relationship between the number of authors and downstream projects, which is inversely associated with the number of dependent packages.

Wang et al. [8] introduced a framework for forecasting commodity demand in the retail supply chain. Their approach leverages vertical federated learning to address data security and privacy concerns in the context of new retail. The authors developed a vertical federated LSTM model to securely encrypt and integrate data from multiple departments, resulting in a new dataset that preserves data privacy throughout the entire supply chain demand forecasting system. The results demonstrated that Fed-LSTM outperforms existing Federated Learning-based machine learning methods, ensuring the confidentiality of business data and providing an effective sales forecasting tool.

Xu and He [9] applied a Deep Belief Network to assess financial credit risk in online supply chain management, specifically targeting small and medium-sized firms in the automobile industry. Their approach utilized a Deep Learning Neural Network comprising Restricted Boltzmann Machines and a SOFTMAX classifier to evaluate credit risk. They collected comprehensive data from 100 small and medium-sized businesses, totaling 300 datasets. The method was validated using a wind dataset, achieving an evaluation accuracy of 96.04%, surpassing the performance of other methods such as Support Vector Machines and Logistic Regression.

The studied SCM works, offer insights into DL's effectiveness but exhibit limitations. They excel in specific tasks like demand forecasting and credit risk assessment but often employ limited datasets and overlook the use of comprehensive features, potentially hindering broader applicability and superior results. These studies underscore the need for more complex datasets and a richer feature set to enhance the versatility and accuracy of DL approaches in SCM. Compared to the studied studies, **DLSTM-SCM** relies on a dynamic model known for its adaptability. It excels in adjusting and optimizing predictions as supply chain data evolves, **DLSTM-SCM** is also enriched with an extensive feature set on complex dataset. This combination has notably improved result accuracy, setting a new standard for SC applications.

III. **DLSTM-SCM** FRAMEWORK

To predict future sales, **DLSTM-SCM** uses a potent prediction model that makes use of deep neural networks, particularly LSTM. To accurately anticipate future sales performance, the prediction model learns patterns, trends, and dependencies by examining prior sales data. Figure 1 illustrates **DLSTM-SCM**'s workflow described below.

1) Initially, it takes historical sales data as input.
2) Then, **DLSTM-SCM** preprocesses it (e.g., date lists, time sequences), configures model parameters, and undergoes training/testing.
3) Evaluation metrics (RMSE, loss, validation loss) are then applied.
4) Further, statistical features like lag and shift enhance predictions, with iterative updates for optimization.
5) Ultimately, **DLSTM-SCM** delivers accurate sales forecasting, aiding supply chain management.

In the following, we detail each step of **DLSTM-SCM**.

A. Data Preprocessing

During the analysis, we look more on the correlation between entities such as stores. For example, Figure 2 shows the highest correlation between the store 1 and 2 (CA_1 and CA_2), which are located in the same state for California based on Walmart dataset. This strong correlation suggests that these two stores share similar sales patterns and may be influenced by similar factors in their respective regions.

The steps follow detail the preprocessing procedures we have considered within **DLSTM-SCM**.

1) Date List Creation for Time Series Analysis: In order to present a Time Series with the right dates, we made a "dates list" that will be helpful. We assigned a variable named "date_index" to the value of the column "date" from the calendar date frame (Algorithm 1).

Algorithm 1 Data Processing.

Require: $calendar\_df$: Calendar DataFrame
Require: $sales\_train\_validation\_df$: Sales Train Validation DataFrame
1: $date\_index = calendar\_df['date']$ {Extract 'date' column from the calendar DataFrame}
2: $dates = date\_index[0 : 1913]$ {Select the first 1913 dates}
3: $dates\_list = []$
4: **for** each $date$ in $dates$ **do**
5: $\_date$ = strptime$(date, '\%Y - \%m - \%d')$.date() {Convert date string to datetime format}
6: APPEND $\_date$ to $dates\_list$ {Add the formatted date to the list}
7: **end for**

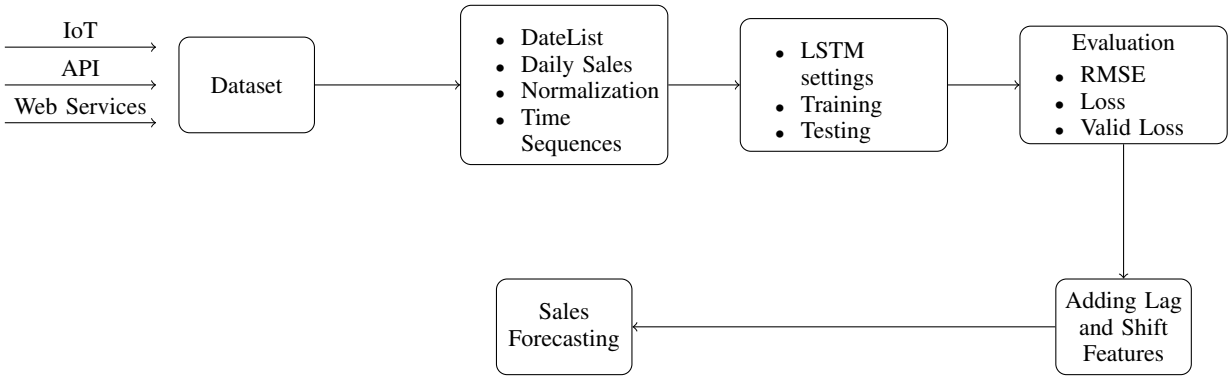

Fig. 1. **DLSTM-SCM** Framework Work-Flow.

Fig. 2. Correlation between Stores Sales

2) Daily Item Sales Creation: We follow a number of processes to produce a dataframe that represents daily item sales. We begin by including a column that functions as a unique identifier by fusing item IDs and store IDs. Then, we made a transpose matrix, where the rows stand for days and the columns for specific goods. We used item-store IDs in place of the product numbers. The index is specified to be a list of dates, and datetime is selected as the index type. These changes provide a structured dataframe, which makes it easier to analyze daily item sales since it has dates as the index and item-store combinations as column names.

3) Data Normalization: we used MinMaxScaler to normalize our data within a feature range of -1 to 1 (Equation 1). This specific normalization process enhances the accuracy and reliability of our predictions by scaling numerical attributes while maintaining their distinctive characteristics.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{1}$$

4) Time Sequences Creation: We employed a technique for creating data windows or sequences (Algorithm 2). Each sequence in our example had 28 successive samples which represents 4 weeks. These sequences will be used to anticipate the sales for the next day. Thanks to the design of these sliding windows, the RNN model can then detect temporal correlations and patterns in the data, improving the accuracy of our predictions.

Algorithm 2 Sliding Windows for Sequence Data.

Require: $data$: Input data array
Require: seq_length: Length of the sliding window sequence
1: $x \leftarrow []$ {Initialize empty array for input sequences}
2: $y \leftarrow []$ {Initialize empty array for output values}
3: **for** i **from** 0 **to** (LENGTH($data$) − seq_length − 2) **do**
4: $\_x \leftarrow$ SUBARRAY($data, i, i +$ seq_length $- 1$) {Create input sequence}
5: $\_y \leftarrow$ ELEMENT_AT($data, i +$ seq_length) {Get corresponding output value}
6: APPEND $\_x$ to x {Append input sequence to x}
7: APPEND $\_y$ to y {Append output value to y}
8: **end for**
9: **return** x, y

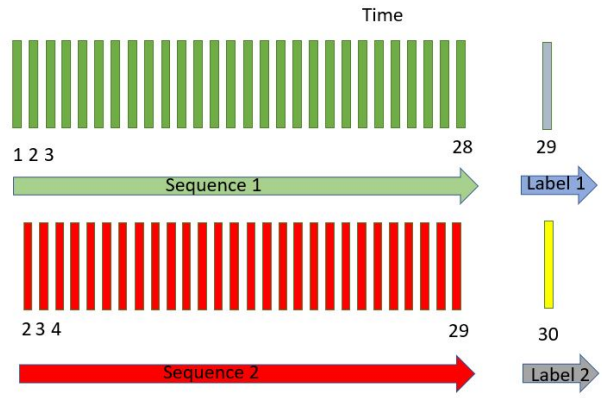

Fig. 3. Creation of weekly Time Sequences.

B. LSTM Modeling

LSTM is a type of RNN that is specifically designed to handle sequential data, such as time series, speech, and text. LSTM networks are capable of learning long-term dependencies in sequential data, which makes them well-suited for tasks such as language translation, speech recognition, and time series forecasting [10]. This dependency is retained by the cells and the memory manipulations are done by the gates

(Figure 4).

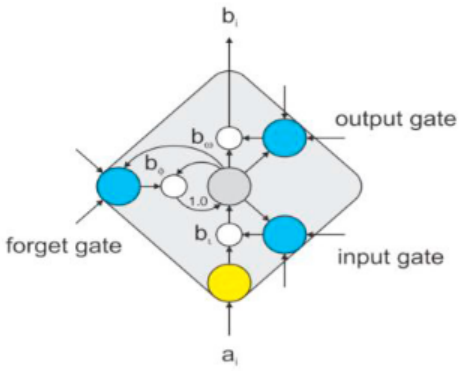

Fig. 4. LSTM Architecture [11].

In LSTM, three gates are distinguished.

(a) **Forget Gate:** The forget gate filters cell state information by processing current input x_t and previous cell output h_{t-1} through weight matrices and biases. An activation function generates a binary output: 0 for discarding information and 1 for retaining it for future use. The forget gate equation is:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

where:

- W_f represents the weight matrix for the forget gate.
- $[h_{t-1}, x_t]$ denotes the concatenation of the previous hidden state and the current input.
- b_f is the bias associated with the forget gate.
- σ represents the sigmoid activation function.

(b) **Input gate:** The input gate augments the cell state with pertinent data. It employs the sigmoid function to control information relevance, akin to the forget gate, using inputs h_{t-1} and x_t. Next, a vector is constructed via the tanh function, outputting a range from -1 to +1, encompassing potential values from h_{t-1} and x_t. Finally, the vector and controlled values are multiplied to yield the essential information. The equations for the input gate operations are as follows:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\hat{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t$$

Where:

- W_i is the weight matrix associated with the input gate.
- W_c is the weight matrix for creating the vector.
- b_i is the bias related to the input gate.
- b_c is the bias for vector creation.
- σ represents the sigmoid activation function.
- $\odot$ denotes element-wise multiplication.
- $\tanh$ signifies the hyperbolic tangent activation function.

(c) **Output Gate:** The output gate extracts valuable information from the current cell state for presentation as output. It begins by creating a vector through the application of the tanh function on the cell state. Subsequently, the sigmoid function is employed to regulate information and filter values to remember, involving inputs h_{t-1} and x_t. Lastly, the vector and controlled values are multiplied to produce an output sent to the next cell as both output and input. The equation for the output gate is:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

DLSTM-SCM provides the layouts of the three models for predicting future sales.

The LSTM models takes several parameters into account:

- Num-classes: This represents the number of outputs.
- Input size: a single input consisting of 28 samples.
- Hidden layers: This indicates the number of hidden layers within each LSTM cell.
- Num layers: In our current configuration, this can be increased if needed.

1) Single Layer Model: This module builds a straightforward LSTM model with a single LSTM layer and a dense input layer. In the single-layer model, a solitary input composed of 28 samples was employed. The model featured a singular layer, one class, and a hidden size of 512 (Algorithm 3). The parameters are carefully chosen to strike a balance between model complexity, training efficiency, and performance. Fine-tuning these parameters based on the specific dataset and problem at hand is crucial to achieving optimal results in LSTM modeling.

Algorithm 3 Single Layer LSTM Training.

Require: Hyperparameters

1: $E \leftarrow 500$, $L \leftarrow 1e-3$, $I \leftarrow 1$, $H \leftarrow 512$, $N \leftarrow 1$, $C \leftarrow 1$ {Set hyperparameters}
2: Initialize LSTM: $L \leftarrow \text{LSTM}(C, I, H, N)$, $L \leftarrow \text{LSTM}(C, I, H, N).to(device)$ {Initialize LSTM model}
3: Set criterion, optimizer, scheduler: $M \leftarrow$ MSE Loss, $O \leftarrow$ Adam optimizer$(L.parameters(), lr = L)$, $S \leftarrow$ LR Scheduler$(O, patience = 500, factor = 0.5, min_lr = 1e-7, eps = 1e-8)$ {Set loss, optimizer, and scheduler}
4: **for** e **in** 1 **to** E **do**
5: Set LSTM mode: $L.train()$ {Set LSTM model to training mode}
6: Forward pass: $O \leftarrow L(X.to(device))$
7: Zero gradients: $O.zero\_grad()$ {Zero the gradients}
8: Calculate loss: $L \leftarrow M(O, Y.to(device))$
9: Backpropagate gradients: $L.backward()$ {Backpropagate gradients}
10: Update model: $O.step()$ {Update model parameters}
11: **end for**

2) Multiple Layer Model: To enhance our model's performance, we increased the training epochs to 700 and added two extra LSTM layers while maintaining the number of classes and hidden size constant (Algorithm 4). The inclusion of multiple LSTM layers enables the model to capture intricate sequential patterns, enhancing its predictive abilities. These adjustments unlock the potential for improved forecasting accuracy and deeper data insights through the utilization of multiple LSTM layers. These hyperparameters play a critical role in shaping the LSTM model's architecture and behavior, directly impacting its predictive capabilities.

Algorithm 4 Multiple Layer LSTM Training.

Require: Hyperparameters

$E \leftarrow 700$, $L \leftarrow 1e-3$, $I \leftarrow 1$, $H \leftarrow 512$, $N \leftarrow 2$, $C \leftarrow 1$ {Set hyperparameters}

Initialize LSTM: $L \leftarrow \text{LSTM2}(C, I, H, N)$, $L \leftarrow \text{LSTM2}(C, I, H, N).to(device)$ {Initialize LSTM model}

Initialize weights: L.apply(init_weights)

Set criterion, optimizer, scheduler: $M \leftarrow$ MSE Loss, $O \leftarrow$ Adam optimizer$(L.parameters(), lr = L, weight_decay = 1e-5)$, $S \leftarrow$ LR Scheduler$(O, patience = 100, factor = 0.5, min_lr = 1e-7, eps = 1e-8)$ {Set loss, optimizer, and scheduler}

for e **in** 1 **to** E **do**

  Set LSTM mode: $L.train()$ {Set LSTM model to training mode}

  Forward pass: $O \leftarrow L(X.to(device))$ {Perform forward pass}

  Zero gradients and clip: $O.zero\_grad()$, clip_grad_norm_$(L.parameters(), 1)$ {Zero gradients and apply gradient clipping}

  Calculate loss: $L \leftarrow M(O, Y.to(device))$ {Calculate loss}

  Backpropagate gradients: $L.backward()$

  Adjust learning rate: $S.step(L)$ {Adjust learning rate using scheduler}

  Update model: $O.step()$ {Update model parameters}

end for

3) Adding More Layers and Features:

In our pursuit of better predictive performance, we introduced delays and rolling windows using data frames for seamless implementation. This allowed us to enhance our model by incorporating additional features and refining our training approach. We calculated means and standard deviations within rolling windows (Algorithm 5). However, this introduced NaN (Not-a-Number) values, which were replaced with zeros. Despite the data's multidimensionality, these changes improved model performance to match the previous version. Notably, we now preserve the best model based on the lowest validation loss, marking a significant improvement.

IV. Experimental Results

Algorithm 5 Shifting Data.

Require: DataFrame DF {Input DataFrame}

$start\_time \leftarrow$ time.time() {Record start time}

for i **in** $[1, 7, 14, 28, 365]$ **do**

  Print 'Shifting:', i {Print current shift value}

  $DF['lag\_{+str(i)}] \leftarrow DF['sales'].transform(\lambda x : x.shift(i))$ {Shift 'sales' column by i positions}

end for

In order to evaluate the efficiency of **DLSTM-SCM**, we used the Walmart dataset, which contains hierarchical sales data, to assess how well our framework performed. To assess the model's effectiveness, we ran trials using RMSE. Each of the three models we suggested includes convolutional layers in addition to the classification layers used by the others. We utilized a rolling window to train the models, and a dropout is used to avoid over-fitting. Throughout the process of creating the models for our various implementations, PyTorch was employed.

Figure 5 illustrates the outcomes of the single-layer model, where a noticeable reduction in both measures loss and validation loss. As the model learns, the training loss decreases, but after approximately 300 epochs, the validation loss starts to rise, hinting at possible overfitting. To address this, consider regularization or architectural adjustments. Monitoring this divergence helps determine the right number of epochs or early stopping.

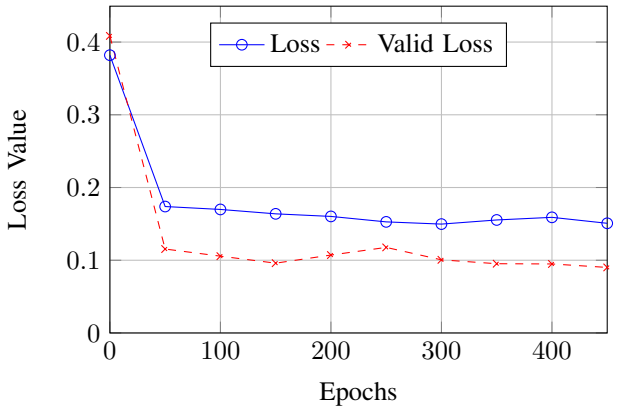

Fig. 5. Single Layer Model Results

Figure 6 presents the performance results of a multiple-layer model during training. Both training and validation losses follow a decreasing trend. However, maintaining a consistent gap between the training and validation losses is essential to ensure the model's generalization performance. Compared to the single-layer model results, the multiple-layer model shows a relatively lower validation loss, indicating improved predictive performance. This demonstrates the potential benefits of adding more layers to the model for capturing complex patterns in the data.

The third model depicted in Figure 7 demonstrates the most promising results among the three configurations. It displays

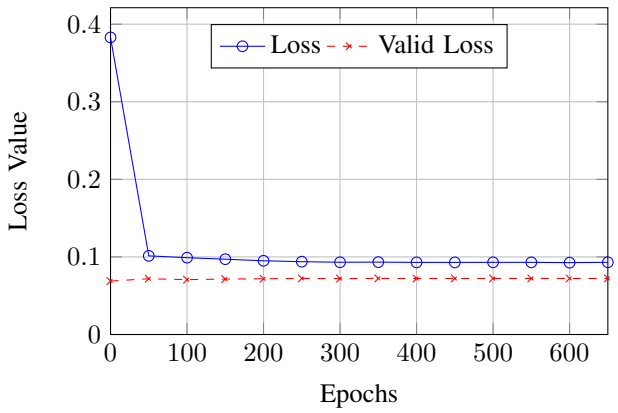

Fig. 6. Multiple Layer Model Results

the results of the statistical features-based model during its training process. Notably, both the training and validation losses exhibit a decreasing trend, indicating effective learning. Interestingly, the statistical features-based model shows significantly lower losses compared to the previous models. This suggests that the model's predictive performance is notably superior to the earlier models.

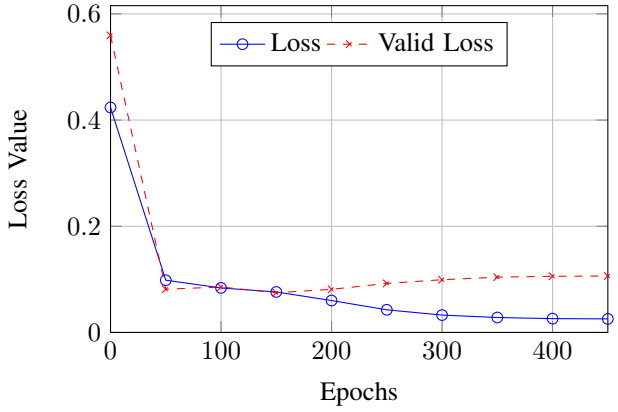

Fig. 7. Statistical Features Based Model Results

V. Conclusion

In this paper, **DLSTM-SCM** framework for enhancing supply chain performance through time series forecasting has been proposed. We initially employed a single-layer model, which provided a foundational understanding of the problem. However, recognizing the potential for improved accuracy, we ventured into more advanced territory, integrating a multi-layer LSTM model. Building upon this foundation, we introduced a third model enriched with lag features. The outcomes exceeded our expectations, demonstrating the promise of **DLSTM-SCM** for sales forecasting and supply chain optimization while updating online the LSTM model. Looking ahead, our future work will delve deeper into these advancements, exploring the application of G3RU models to further refine and enhance our forecasting capabilities. With each step, we continue to advance **DLSTM-SCM** for supply chain forecasting, driving improved performance and efficiency in this critical domain.

References

[1] Graham C Stevens. Integrating the supply chain. *international Journal of physical distribution & Materials Management*, 19(8):3–8, 1989.

[2] Ji Won Chong, Wooju Kim, and Jun Seok Hong. Optimization of apparel supply chain using deep reinforcement learning. *IEEE Access*, 10:100367–100375, 2022.

[3] David Opeoluwa Oyewola, Emmanuel Gbenga Dada, Temidayo Oluwatosin Omotehinwa, Onyeka Emebo, and Olugbenga Oluseun Oluwagbemi. Application of deep learning techniques and bayesian optimization with tree parzen estimator in the classification of supply chain pricing datasets of health medications. *Applied Sciences*, 12(19):10166, 2022.

[4] Chaitanya Ingle, Dev Bakliwal, Jayesh Jain, Preeyesh Singh, Preeti Kale, and Vaibhav Chhajed. Demand forecasting: Literature review on various methodologies. In *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pages 1–7. IEEE, 2021.

[5] Mohammad Amin Amani and Samuel Asumadu Sarkodie. Mitigating spread of contamination in meat supply chain management using deep learning. *Scientific Reports*, 12(1), March 2022.

[6] Oktay Ulucan, Dilber Karakaya, and Mehmet Turkan. Meat quality assessment based on deep learning. In *2019 Innovations in Intelligent Systems and Applications Conference (ASYU)*. IEEE, 2019.

[7] Xin Tan, Kai Gao, Minghui Zhou, and Li Zhang. An exploratory study of deep learning supply chain. In *Proceedings of the 44th International Conference on Software Engineering*, pages 86–98, 2022.

[8] Hexu Wang, Fei Xie, Qun Duan, and Jing Li. Federated learning for supply chain demand forecasting. *Mathematical Problems in Engineering*, 2022:Article ID 4109070, 8 pages, 2022.

[9] Rong-Zhen Xu and Meng-Ke He. Application of deep learning neural network in online supply chain financial credit risk assessment. In *2020 international conference on computer information and big data applications (CIBDA)*, pages 224–232. IEEE, 2020.

[10] GeeksforGeeks. Deep learning: Introduction to long short-term memory, 2023.

[11] Afan Galih Salman, Yaya Heryadi, Edi Abdurahman, and Wayan Suparta. Single layer & multi-layer long short-term memory (lstm) model with intermediate variables for weather forecasting. *Procedia Computer Science*, 135:89–98, 2018.