Al-Farouk journal of sciences
Volume (2), Issue (3) (JUNE, 2026)
ISSN: 3135-2359

مجلة الفاروق للعلوم
المجلد (2) العدد 3 يونيو 2026
ردمد: 3135-2359

MATLAB/Simulink-Based Online CNN-LSTM Framework for EDFA Gain, OSNR, and ASE Noise Prediction with AI Decision Support

Marai M.Abousetta
m.abousetta@academy.edu.ly
Sujoud A.Adbeab
Libyan Academy For Postgraduate Studies
Email: sujoudtaher@gmail.com

تاريخ الاستلام: 2026/04/20 تاريخ المراجعة 20/ 05/ 2026 تاريخ القبول:2026/06/07- تاريخ النشر: 22/ 06/ 2026

Abstract—This paper reports a MATLAB/Simulink simulation study for predicting Erbium-Doped Fiber Amplifier (EDFA) gain, optical signal-to-noise ratio (OSNR), and amplified spontaneous emission (ASE) noise by using a hybrid convolutional neural network and long short-term memory (CNN-LSTM) model. The work was implemented in MATLAB R2022b. A dynamic data set was generated in MATLAB from six operating variables: input optical power, pump power, wavelength, temperature, active channel count, and span length. The trained network was saved and deployed in Simulink using a Predict block. Two Simulink models were built: an offline validation model using stored test signals and an online simulation model using runtime source blocks. The word online is used only in the Simulink simulation sense; the model is not connected to physical EDFA hardware, optical spectrum analyzers, or pump-control equipment. Output-specific evaluation produced RMSE values of 1.4581 dB for gain, 2.7702 dB for OSNR, and 1.5373 dBm for ASE noise. A rule-based AI decision-support layer was also added to generate status, pump-action, and alarm indicators. The study demonstrates a complete software workflow that can be used as a preliminary platform before measured EDFA data or hardware-in-the-loop testing are introduced.

Keywords—*EDFA, CNN-LSTM, MATLAB, Simulink, OSNR, ASE noise, optical communication, AI decision support.*

I. INTRODUCTION

Erbium-Doped Fiber Amplifiers (EDFAs) are important components in optical communication systems because they amplify optical signals directly in the optical domain, especially around the 1550 nm transmission window. In a wavelength-division multiplexed link, the amplifier output quality is affected by many operating conditions, including pump power, input power, channel loading, wavelength, and temperature. These interactions make EDFA monitoring a suitable problem for both physical modeling and data-driven prediction.

Traditional EDFA models can be accurate when the internal fiber and pump parameters are available. However, in many educational and early-stage research projects, the exact amplifier parameters are not available. A purely experimental data set may also be difficult to collect because it requires optical equipment, calibration, and repeated measurements. For this reason, the present work starts from a simulation-generated data set and focuses on building a complete implementation workflow.

The work was carried out as a MATLAB/Simulink project. MATLAB was used to generate the data, train the CNN-LSTM regression model, calculate RMSE values, and export result figures. Simulink was then used to validate the trained network in two forms. The first form used stored workspace signals and the second used runtime source blocks, which makes the model online inside the simulation environment. This distinction is important because it avoids overstating the scope of the work as a real-time hardware experiment.

The paper is written around the actual implementation files used in the project: D1.m for data generation and training, N1.mat for the trained network, D2.mat for validation signals, M1.slx for offline Simulink validation, M2.slx for online Simulink simulation, and R2.m for plotting the online outputs. This direct link to the MATLAB project makes the paper reproducible and keeps the discussion grounded in the performed work.

The main objectives are: (1) to build a dynamic EDFA-related data set, (2) to train a CNN-LSTM model for three-output regression, (3) to deploy the trained model in Simulink, (4) to add an AI decision-support layer, and (5) to evaluate the prediction error using output-specific RMSE. The overall workflow is shown in Fig. 1.

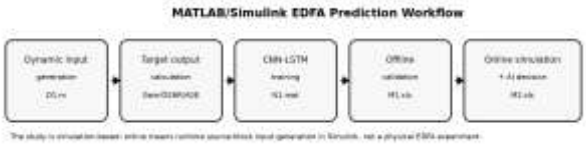

Fig. 1. Overall MATLAB/Simulink workflow used in the EDFA prediction project.

II. RELATED WORK

EDFA modeling has a long history in optical communication research. Giles and Desurvire presented a classical modeling approach for erbium-doped fiber amplifiers based on gain and noise behavior [1]. That type of model is valuable because it is connected to the underlying physical mechanism of amplification and ASE generation. The difficulty is that detailed device information is usually required to tune such models correctly.

A more recent review by Liu et al. explains that EDFA gain modeling remains challenging because gain depends on wavelength, channel loading, pump condition, and amplifier operating mode [2]. This is the same practical reason why the present simulation includes several inputs instead of using only one variable. Although the current study does not use real EDFA measurement data, it follows the idea that a single output such as gain is not enough to describe amplifier performance.

Deep learning has also been used for optical performance monitoring. Wang et al. used LSTM-based deep learning for OSNR and nonlinear noise estimation in optical fiber systems [3]. Their work supports the use of recurrent sequence learning for optical quantities that vary over time. In a different direction, Da Ros et al. demonstrated a neural-network EDFA gain model that could generalize across multiple physical devices [4]. These studies show that machine learning can be useful in optical-network modeling when enough representative data are available.

MATLAB and Simulink support this type of workflow because neural networks can be trained in MATLAB and then moved to block-level simulation. MathWorks documentation describes the Deep Learning Toolbox and the Predict block, which can load a pretrained network and produce predictions inside Simulink [5], [6]. The present work uses this capability in a student-scale EDFA simulation environment. The novelty is not a new deep-learning theory; it is the integration of data generation, training, Simulink prediction, and decision logic in one coherent project.

III. SIMULATION DATA SET

A. Input Variables

The MATLAB script D1.m generated 1500 time samples. The simulation stop time used later in Simulink was 1499 s, so the signal length and simulation time were consistent. Each sample contained six input variables. These variables were selected because they have direct or indirect influence on EDFA behavior. Input power and pump power affect amplification. Wavelength affects gain variation across the optical band. Temperature and span length represent environmental and link-related operating conditions. The number of active channels represents channel loading.

TABLE I
INPUT VARIABLES USED IN THE MATLAB DATA SET

Variable	MATLAB name	Range
Input power	Pin_dBm	-30 to -10 dBm
Pump power	Pump_mW	150 to 350 mW
Wavelength	Lambda_nm	1530 to 1565 nm
Temperature	Temp_C	15 to 45 C
Active channels	Channels	16 to 80
Span length	Span_km	60 to 100 km

The input signals were not kept constant. Step changes and time-varying changes were introduced to imitate load changes, operating-point movement, and network traffic variation. This produced a more realistic learning task than a purely static data set. It also allowed the Simulink source-block model to be tested with changing input values.

B. Target Output Calculation

The three target outputs were EDFA gain, OSNR, and ASE noise. Gain was treated as the amplification level between the input and output powers. ASE noise was calculated using a simplified optical-noise expression. OSNR was then obtained from the signal power relative to the ASE noise level. The equations were used to create repeatable training targets, not to claim an exact physical EDFA model.

The simplified target-generation stage is useful for verifying the AI pipeline. If measured EDFA data become available later, the same CNN-LSTM and Simulink workflow can be reused after replacing the synthetic target data. This makes the project expandable rather than limited to one data source.

C. Dynamic Target Signals

The generated dynamic outputs are shown in Figs. 2–4. Gain varies approximately between 23 dB and 28.5 dB. OSNR varies over a wider range because it is affected by both signal power and noise power. ASE noise remains negative in dBm and changes with the simulated amplification and noise conditions. These plots confirm that the data generation stage produced visible dynamics rather than constant output values.

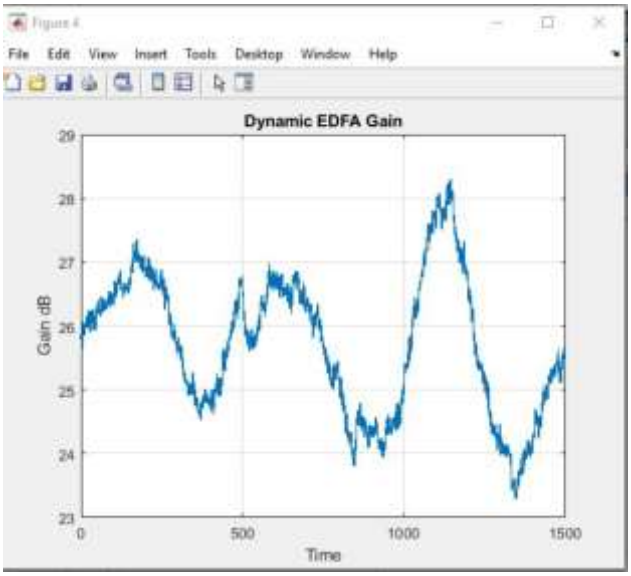

Fig. 2. Dynamic simulated EDFA gain generated in MATLAB.

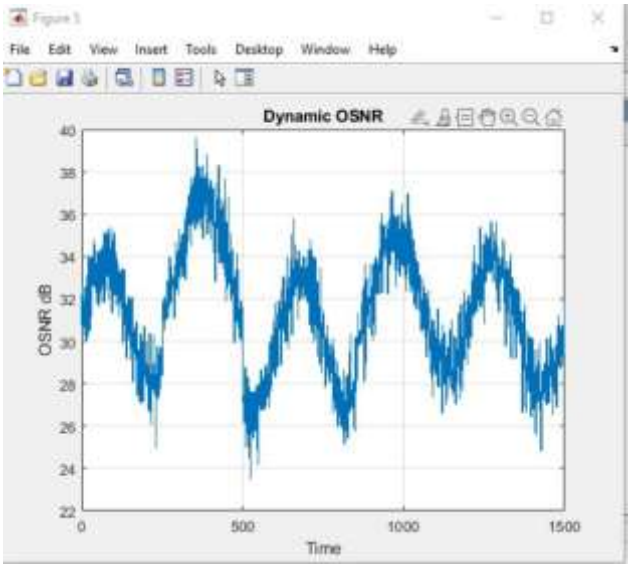

Fig. 3. Dynamic simulated OSNR generated in MATLAB.

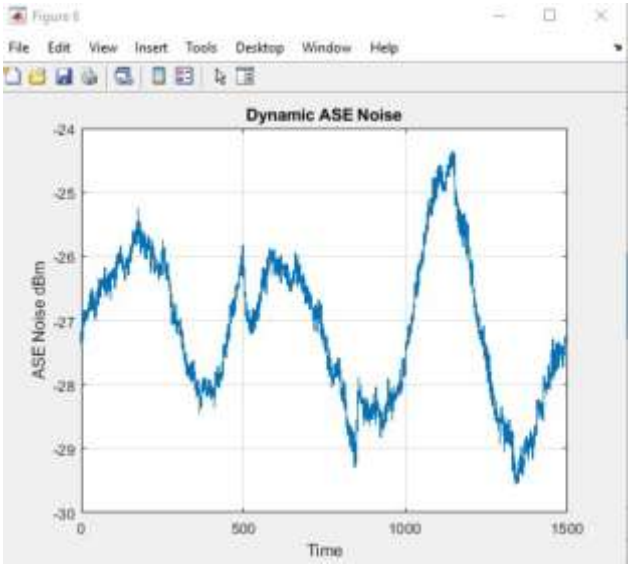

Fig. 4. Dynamic simulated ASE noise generated in MATLAB.

IV. CNN-LSTM MODEL AND TRAINING

A. Model Structure

The regression model combines convolutional layers and an LSTM layer. The convolutional part extracts local patterns from the six input variables over a short time window. The LSTM part keeps sequence information, which is important because EDFA-related outputs may depend on recent changes rather than only the current sample. The final fully connected layer produces three numerical outputs: gain, OSNR, and ASE noise.

A temporal window of 20 samples was used. For each training observation, the model received a short sequence rather than a single row of input values. This design choice made the model closer to an online monitoring system, where recent input history can influence the current prediction.

B. Training Settings

TABLE II
CNN-LSTM TRAINING CONFIGURATION

Setting	Value
Samples	1500
Input features	6
Target outputs	3
Temporal window	20 samples
Conv1D filters	32 and 64
LSTM units	64
Dropout	0.10
Data split	70% / 15% / 15%
Optimizer	Adam
Mini-batch size	64
Epochs	12
Learning rate	0.001
Saved network file	N1.mat

The model was trained for 12 epochs using the Adam optimizer. MATLAB reported that the maximum number of epochs was completed. The training-progress window shows that both RMSE and loss decreased sharply during the early iterations and then stabilized. The final validation RMSE shown in the training window was 3.1607. This is a global training indicator; later in the paper, the three output-specific RMSE values are reported separately.

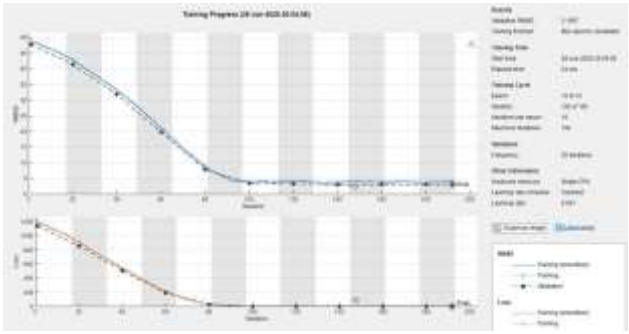

Fig. 5. MATLAB training-progress window for the CNN-LSTM model.

C. Reason for Using CNN-LSTM

A simple feedforward network can learn static input-output mapping, but it does not explicitly represent time history. A standard LSTM can model sequence behavior, but it may not extract local patterns from multivariable input windows as efficiently as a convolutional front end. The hybrid CNN-LSTM structure was therefore selected as a practical compromise. It is not claimed to be the only possible model, but it is suitable for this first MATLAB/Simulink implementation.

The model was saved after training in N1.mat. This saved file is important because the Simulink Predict block loads the trained network from the MATLAB environment. Without saving the trained network, the model could not be reused in the block-level simulation.

V. SIMULINK IMPLEMENTATION

A. Offline Validation Model

The first Simulink model was M1.slx. This model used stored workspace signals, named Uts and Yts, for offline validation. The input signal was passed through a MATLAB Function block that prepared the input window expected by the CNN-LSTM network. The Predict block then generated the predicted output vector. The predicted output and true output

were routed to logging blocks and a scope so that the two signals could be compared.

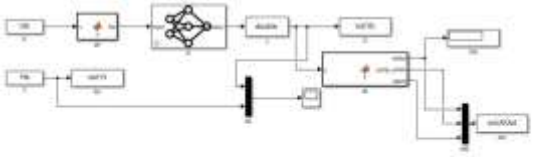

Fig. 6. Offline validation model M1.slx using stored workspace signals.

B. Online Source-Block Model

The second Simulink model was M2.slx. In this model, the stored input signal was replaced by source blocks. Some sources were sinusoidal and others were repeating stair sequences. The six signals were combined using a Mux block and then sent to the same windowing and Predict-block path. This design allowed the input values to be generated during the Simulink run.

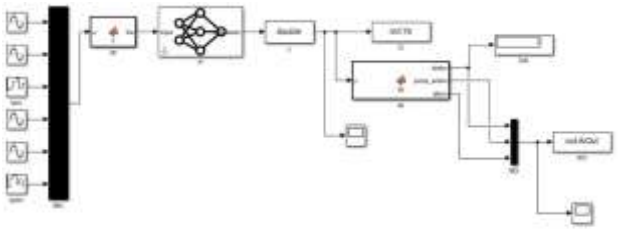

Fig. 7. Online Simulink model M2.slx with runtime source-block input generation.

The output of the Predict block was converted to double precision before logging and decision processing. This conversion was added because neural-network prediction outputs may be single precision, while the following MATLAB Function and To Workspace blocks were easier to handle in double precision. The output vector was saved as out.Yp and also sent to the AI decision-support block.

C. Project File Roles

TABLE III
MAIN PROJECT FILES AND THEIR ROLES

File	Role
D1.m	Generate data, train CNN-LSTM, save N1.mat and D2.mat
N1.mat	Saved trained CNN-LSTM network
D2.mat	Saved validation and test signals
M1.slx	Offline Simulink validation model
M2.slx	Online source-block Simulink model
R2.m	Plot online prediction outputs

This file organization is useful because each part of the project has a clear task. The training stage is not mixed with the online simulation stage, and the plotting stage can be repeated after each Simulink run. This also helped identify practical problems such as workspace-variable names and output orientation during plotting.

VI. DECISION-SUPPORT LAYER

A MATLAB Function block named AI was placed after the prediction output. Its purpose was to translate the predicted optical values into simple operational indicators. The block produced status, pump_action, and alarm. The decision layer used fixed thresholds. For example, low OSNR or high ASE conditions could trigger an alarm and suggest increasing or decreasing the pump action.

TABLE IV
AI DECISION-SUPPORT OUTPUT CODING

Output	Meaning
status	0 normal, 1 low OSNR, 2 high ASE, 3 low gain
pump_action	-1 decrease, 0 keep, 1 increase
alarm	0 no alarm, 1 warning

During the final M2 run, the display block showed status value 2. The decision curves in Fig. 8 appear constant because the tested input sequence kept the decision logic in a stable condition. This does not mean the decision layer is useless; it means the selected thresholds were crossed in the same way for most of the tested interval. In future work, these thresholds should be tuned using measured operating states.

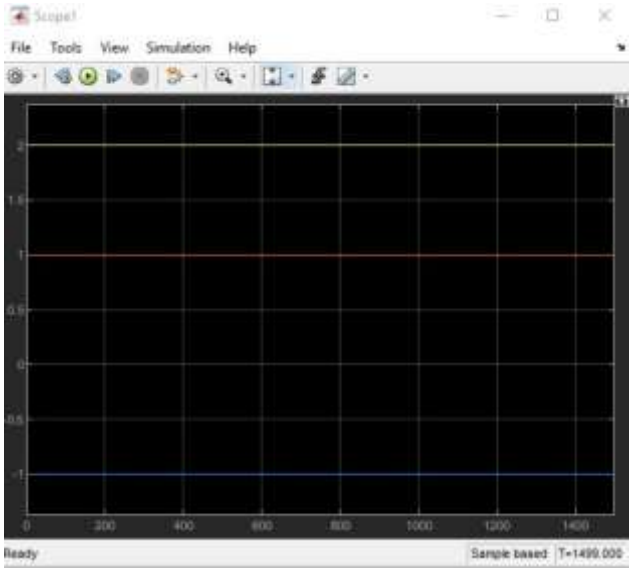

Fig. 8. AI decision-support outputs from the online Simulink model.

VII. RESULTS

A. Output-Specific RMSE

RMSE was calculated separately for each output because the units are different. Gain and OSNR are expressed in dB, while ASE noise is expressed in dBm. Combining the three outputs into one error number would hide which quantity is predicted more accurately. The values obtained from the MATLAB run are reported in Table V.

TABLE V
FINAL RMSE RESULTS

Output	RMSE
EDFA gain	1.4581 dB
OSNR	2.7702 dB
ASE noise	1.5373 dBm

The gain error is acceptable for a first simulation-based model. OSNR produced the largest RMSE, which is reasonable

because OSNR depends on both signal power and noise power. A small error in gain or ASE can appear as a larger OSNR error. ASE noise had an RMSE of 1.5373 dBm, which shows that the model followed the general trend but did not capture all rapid variations.

B. Online Prediction Plots

The online predicted plots are shown in Figs. 9–11. These figures were generated after running the M2 online model and exporting the logged Yp output. The online predicted curves are smoother than the dynamic target signals because the deployed network and input-window block produce a filtered regression response. This is expected in a neural-network regression model trained on simulated data.

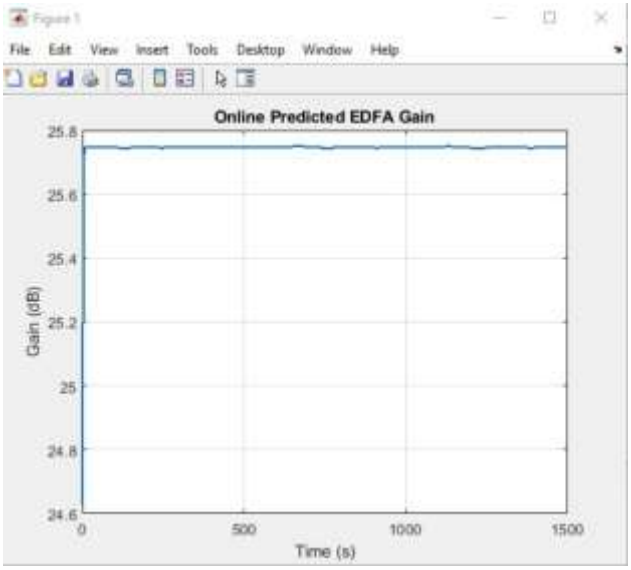

Fig. 9. Online predicted EDFA gain exported from the Simulink run.

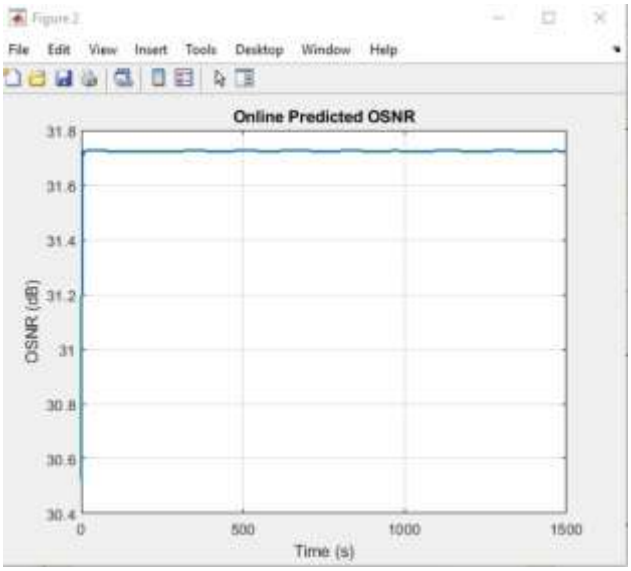

Fig. 10. Online predicted OSNR exported from the Simulink run.

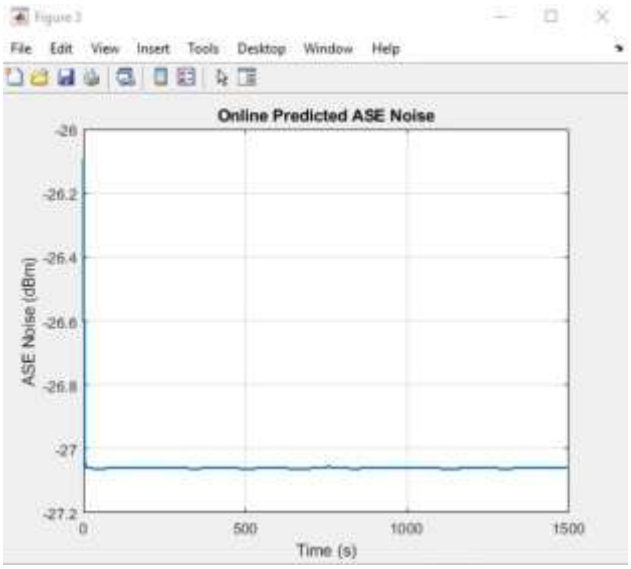

Fig. 11. Online predicted ASE noise exported from the Simulink run.

When the three outputs were plotted together on one axis, their variations appeared almost flat because the quantities have different scales and units. The combined output plot in Fig. 12 is still useful for confirming that the deployed Simulink block produces a three-element output vector. For detailed interpretation, each output should be inspected on its own axis.

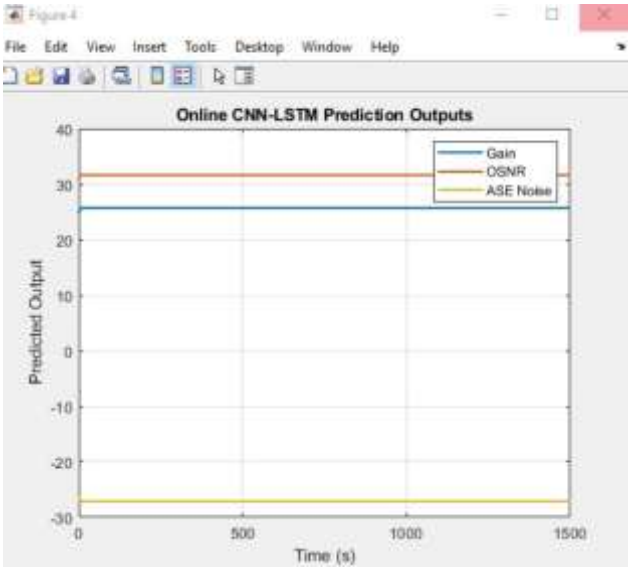

Fig. 12. Combined online CNN-LSTM prediction outputs on a common axis.

C. Interpretation

The results indicate that the workflow is technically correct: the data are generated, the model is trained, the network is saved, Simulink loads the network, predictions are logged, and the AI decision layer receives the predicted values. The main weakness is not the block connection but the quality and realism of the training data. Since the data were generated from simplified equations, the model can only learn those simulated relations. It cannot be expected to generalize to real EDFA hardware without retraining or validation using measured data.

The final RMSE values should therefore be understood as simulation results. They are useful for comparing future versions of the same project. For example, if measured or

OptiSystem-generated data are introduced later, the same RMSE table can be recalculated and compared with the current results.

VIII. PRACTICAL IMPLEMENTATION NOTES

Several practical issues appeared during the MATLAB/Simulink implementation. The first issue was signal scaling. When gain, OSNR, and ASE noise were plotted together on one scope, the curves appeared almost straight because they were displayed on the same vertical axis. This was not necessarily an error in the model. It was a visualization problem caused by plotting variables with different units and numerical ranges together. For this reason, the final analysis used separate figures for gain, OSNR, and ASE noise before showing the combined plot.

The second issue was workspace-variable naming. The plotting file R2.m originally expected the output to be stored as out.Yp. In some runs, Simulink exported the signal directly as Yp, while in other runs the signal was available inside a SimulationOutput object. This type of problem is common when moving from script-based MATLAB work to Simulink logging. The final plotting approach was adjusted to read the logged time series safely and then convert the data orientation when needed.

The third practical issue was the decision display. The display block DA showed a single status number at the end of the simulation. This value is useful for a quick check, but it should not be used alone to understand the complete decision behavior. The scope output is more informative because it shows status, pump_action, and alarm over the simulation interval. In the final run, the decision remained stable, so the AI output lines appeared constant. This is consistent with threshold-based logic.

TABLE VI
IMPLEMENTATION CHECKS USED DURING THE FINAL RUN

Check	Purpose
Yp output exists	Confirms that the Predict block was logged correctly
Separate output plots	Avoids misleading flat curves caused by common-axis scaling
AI scope output	Shows status, pump_action, and alarm over time
RMSE per output	Keeps dB and dBm errors separate
Simulation-only statement	Prevents confusing the work with a hardware EDFA experiment

These implementation notes are included because they describe real problems encountered during the project. They also make the paper more useful for repeating the work. A future researcher can keep the same model structure but change the data source, improve the target equations, or replace the decision thresholds without redesigning the whole Simulink diagram.

IX. LIMITATIONS AND RECOMMENDATIONS

The main limitation is that the project did not use laboratory measurements. No physical EDFA, optical spectrum analyzer, WDM transmitter, pump controller, or real-time acquisition hardware was connected. The online model is online only inside Simulink. This point must be stated clearly because it affects how the results should be interpreted.

The second limitation is the simplified target-generation model. It is suitable for software development, but it does not fully represent wavelength-dependent gain tilt, saturation dynamics, noise figure changes, or device-specific behavior. The third limitation is the rule-based decision layer. The AI block is useful as a monitoring demonstration, but it is not yet a trained controller.

The first recommendation is to collect measured EDFA data or generate more detailed data using an optical-system simulator. The second recommendation is to compare CNN-LSTM with ANN, standard LSTM, GRU, random forest, and support-vector regression using the same data split. The third recommendation is to tune the decision thresholds using labeled operating states. The fourth recommendation is to build a hardware-in-the-loop version after validating sampling time, signal scaling, and data format. Finally, the project should include more wavelength channels if the goal is to move from scalar prediction to full gain-spectrum prediction.

A measured-data extension should begin with a clear measurement plan. The input power, pump current or pump power, wavelength, temperature, and channel loading should be recorded together with the measured output spectrum. The data should be stored with the same sampling interval and the same variable names used in the current MATLAB project. Keeping the same structure will make it possible to reuse D1.m, M1.slx, M2.slx, and R2.m with fewer changes.

A second useful improvement is to report confidence intervals or repeated-run statistics. In the current work, one final RMSE value is reported for each output. In a stronger version of the study, the data set can be divided several times into different training and test splits. The mean and standard deviation of the RMSE would then show whether the model performance is stable or depends strongly on one random split.

A third improvement is to separate prediction from control. The CNN-LSTM block should be responsible only for estimating gain, OSNR, and ASE noise. The control or decision block should then be validated separately, either by expert-defined rules or by a labeled classifier. This separation will make the model easier to debug and safer to extend because prediction errors and decision errors can be analyzed independently.

Finally, any future paper based on physical measurements should add a calibration section. Optical power meters, optical spectrum analyzers, and wavelength sources have measurement uncertainty. If these uncertainties are not documented, the reported RMSE may look worse or better than the real model quality. Calibration information would therefore make the results more credible and closer to a publishable optical-network study.

X. CONCLUSION

This paper presented a MATLAB/Simulink-based CNN-LSTM framework for EDFA gain, OSNR, and ASE noise prediction with an AI decision-support layer. The work was based on a simulation data set generated in MATLAB and a

trained network deployed inside Simulink. The offline model M1.slx was used for stored-signal validation, while M2.slx implemented an online source-block simulation. The achieved RMSE values were 1.4581 dB for gain, 2.7702 dB for OSNR, and 1.5373 dBm for ASE noise.

The contribution of the work is a complete and reproducible software workflow rather than a hardware EDFA experiment. The results show that MATLAB and Simulink can be used to connect data generation, deep-learning prediction, output logging, and decision support in one project. The next step is to validate the same framework using measured EDFA data so that the model can move from simulation demonstration toward practical optical-network monitoring.

REFERENCES

[1] C. R. Giles and E. Desurvire, "Modeling erbium-doped fiber amplifiers," Journal of Lightwave Technology, vol. 9, no. 2, pp. 271–283, Feb. 1991.

[2] Y. Liu, X. Liu, L. Liu, Y. Zhang, M. Cai, L. Yu, and W. Hu, "Modeling EDFA gain: approaches and challenges," Photonics, vol. 8, no. 10, Art. no. 417, 2021, doi: 10.3390/photonics8100417.

[3] Z. Wang, A. Yang, P. Guo, and P. He, "OSNR and nonlinear noise power estimation for optical fiber communication systems using LSTM based deep learning technique," Optics Express, vol. 26, no. 16, pp. 21346–21357, 2018.

[4] F. Da Ros, U. C. de Moura, and M. P. Yankov, "Machine learning-based EDFA gain model generalizable to multiple physical devices," in Proc. European Conference on Optical Communication (ECOC), 2020, doi: 10.1109/ECOC48923.2020.9333297.

[5] MathWorks, "Deep Learning Toolbox Documentation," MathWorks, accessed Jun. 2026.

[6] MathWorks, "Predict responses using a trained deep learning network in Simulink," MathWorks, accessed Jun. 2026.

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

[8] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.

[9] S. Zhu, C. L. Gutterman, W. Mo, Y. Li, G. Zussman, and D. C. Kilper, "Machine learning based prediction of erbium-doped fiber WDM line amplifier gain spectra," in Proc. European Conference on Optical Communication (ECOC), 2018.

[10] BenHusein, A. H., Shakrum, F. M., & Elgdiri, E. M. (2026). A Comprehensive Performance Evaluation of a Wi-Fi-Based Wireless Sensor Network Using Raspberry Pi and ESP8266 Under Multi-Rate Traffic Conditions. *Al-Farooq Journal of Sciences*, *2*(3), 816-818.