ML-Based Edge Application for Detection of Forced Oscillations in Power Grids

Sergio A. Dorado-Rojas, Shunyao Xu, Luigi Vanfretti
Electrical, Computer, and Systems Engineering
Rensselaer Polytechnic Institute
Troy, NY, USA
{dorads, xus, vanfrl}@rpi.edu

M. Ilies I. Ayachi, Shehab Ahmed
Electrical and Computer Engineering
King Abdullah University of Science and Technology
Thuwal, Saudi Arabia
{mohammed.ayachi, shehab.ahmed}@kaust.edu.sa

***Abstract*—This paper presents a Machine Learning (ML) solution deployed in an Internet-of-Things (IoT) edge device for detecting forced oscillations in power grids. We base our proposal on a one-dimensional (1D) and two-dimensional (2D) Convolutional Neural Network (CNN) architecture, trained offline and deployed on an Nvidia Jetson TX2. Our work also shows the advantages of optimizing the CNNs models, after training, using TensorRT, a library for accelerating deep learning inference in real-time. Both real-world and synthetic measurement signals are employed to validate the applicability of the proposed approach.**

***Index Terms*—Convolutional neural networks, forced oscillations, NVIDIA Jetson TX2, TensorRT, real-time detection**

I. Introduction

Power system oscillations are generally classified as free and forced. *Free oscillations* appear as the system's response, for instance, to accommodate a change of loads. Free oscillations are *structural* to the system dynamics. In contrast, *forced oscillations* occur when exogenous stimuli with a rich enough spectral component (e.g., cycle-limited control actuation [1], or periodic disturbances [2]) excite the system, thus producing oscillating modes [3]. While different control systems such as the Power System Stabilizer (PSS) aim at attenuating the effects of free oscillations, forced oscillations could lead to system-wide cascading outages if the excited modes are unstable [4]. The only remedial action is to disconnect the equipment that causes the oscillation or drastically reduce its output power, as in the case of wind farms [5].

Several detection methods have been proposed given the adverse potential of forced oscillations on the grid (e.g., [6]). In [7], a novel multi-delay self-coherence method using data measured by Phasor Measurement Units (PMUs) is designed not only to detect but also to locate the source of a forced oscillation. The reader is referred to [8] for a comprehensive review of such techniques. Meanwhile, [9] provides a suppression control method that can automatically induce a power injection into the power grid to compensate for the impact of the forced oscillation.

This work was funded in part by the New York State Energy Research and Development Authority (NYSERDA) under grant agreement numbers 137951 and 137940, and in part by the Center of Excellence for NEOM Research at King Abdullah University of Science and Technology.

Most oscillation detection proposals consist of a chain of signal processing stages such as noise removal and filtering (e.g., [5]). While some of these techniques have proven effective, their computational efficiency is constrained by the complexity of intermediate calculations. Overcoming this time requirement is crucial if such algorithms are deployed as real-time detection pipelines. Even more importantly, if they are to be deployed at the edge. Some recent contributions have addressed the online performance of detection methods (e.g., [10], [11]). For instance, the detection algorithm in [12] takes 1.7 s to process and label a forced oscillation. In, [13], the detection time is about 350 ms. From the commercial point of view, [14] presents a set of detection methods including patented solutions now implemented in commercial relays. In summary, it is natural to question if most signal processing-based solutions are also feasible for real-time deployment.

Machine Learning (ML) algorithms show promising potential as data-driven methods for oscillation detection (see [15], [16]) and efficient real-time deployment while harnessing Graphical Processing Unit (GPU) power. Nowadays, ML models can be easily optimized offline using existing data, which is known as *training*. The trained model can be deployed in Internet-of-Things (IoT) devices at the edge to process measurements directly. IoT devices are equipped with a Graphics Processor Unit (GPU). GPUs allow ML models to be executed in real-time efficiently [17]. Like the NVIDIA Jetson TX2, IoT devices have been proven effective for real-time ML-based solutions. Successful case studies arise from applications such as depth reconstruction from images [18], and face recognition [19] among others. That being said, the purpose of this work is to make the case that ML-based models are feasible solutions for real-time pipelines for forced oscillation detection at the edge, e.g., as part of a new type of protective relay.

Previous works regarding ML for oscillation detection have focused on developing and deploying the algorithm *on a computer*, either for offline or real-time detection. Such "server-centered service" adds the requirement of a communication network for ambient data collection. An example of this is shown in [10] where an oscillation detection method based on an improved XGboost algorithm and random power system measurements is introduced. The

trained model is applied to online oscillation detection of a power system, with the algorithm running on a computer. So, if the algorithm were to be deployed for real-time detection, data must be streamed through a communication network. Likewise, in [11], an ML algorithm based on regularized exponential forgetting is proposed. The solution is suitable for non-stationary data analysis. The model is deployed on a conventional computer, so measurements such as currents, voltages, and angle differences must be transmitted through a communication network to apply it to real-time ambient data.

Data transmission from client to server introduces further delays into the overall detection process, and consequently, practically reduces the computational efficiency of any algorithm. Therefore, deploying ML-based oscillation detection algorithms on edge devices becomes more relevant. As an advantage, it bypasses the communication stages directly and can work with measurement data on sites near potential sources (e.g., wind farms [14]). To fill this gap, this paper introduces a method and presents an ML-based approach for oscillation detection at the edge deployed on an NVIDIA Jetson TX2. The proposed method can detect forced oscillations accurately using real-time measurements directly while efficiently processing the data with the built-in GPU. We must underline that the core of the contribution is proving the potential of edge devices as means for real-time inference.

The paper is organized as follows: Section II describes the procedure to train 1D- and 2D-Convolutional Neural Networks (CNNs). In Section III, we present how the trained models are downloaded to an NVIDIA Jetson TX2 for real-time execution using `TensorRT`. The inference results on ambient data from a wind farm streamed in real-time are discussed in Section IV. Finally, Section V concludes the work.

II. CNN Model Description and Training

A. Foundations of CNNs

CNNs are models inspired by the human eye's mechanism to extract visual details. CNNs mimic the structure of receptive fields by allocating specialized neurons to detect features in specific parts of an image. So, a CNN swipes an image looking for local similarities and then filters out the high-level or global features [20]. Feature extraction allows CNNs to discriminate raw data into several categories (i.e., similar images have similar characteristics). Therefore, they exhibit outstanding performance on image classification tasks.

We seek to assess the potential of CNNs for forced oscillation detection in two flavors: one-dimensional (1D) and two-dimensional (2D) CNNs. The former architecture uses the measurement data as an input time series. At the same time, the latter sees the ambient data as *images*.

Both CNN models are developed in Python using the TensorFlow framework and the Keras API. As we will see in Section III, TensorFlow offers a direct path to deploy trained models in hardware using the Software Development Kit (SDK) `TensorRT`.

B. Training and Validation Data Set Construction

To build training and testing data sets, we used measurements from a wind farm in Oklahoma [5]. The information comes from PMU recordings of voltage, current, and frequency during various oscillation episodes, which utility workers classified. In Fig. 1, an example event spanning $\approx$ 12.5 min $=$ 750 s and features a forced oscillation is shown. We divide the signal into 1 s windows, each window containing exactly 31 samples (there is a one-sample overlap between consecutive windows). By doing so, we generate $\approx$ 750 instances from each recording at a particular location, where an instance corresponds to a 1 s window. Examples of training instances can be seen in the 1 s frames in Fig. 1.

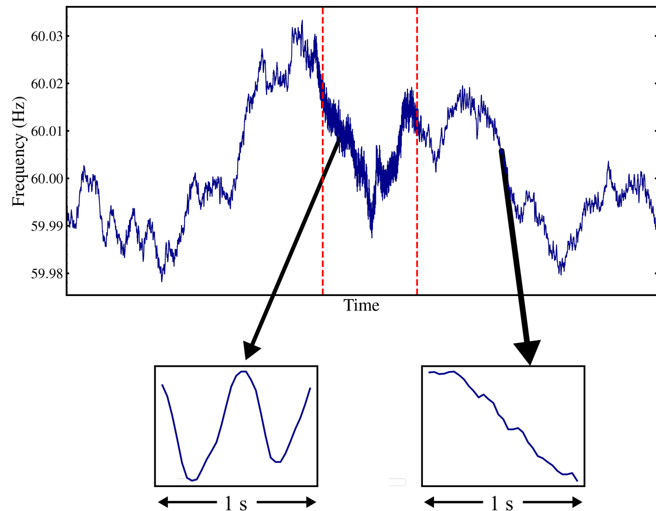

Fig. 1. Sample of training PMU data (dotted vertical lines indicate the event inset and offset).

C. 1D-CNN Model

A 1D-CNN performs a temporal convolution on the input data through several kernels. Besides the dimensionality of the input, the principles behind the operation of a 1D-CNN are the same as conventional 2D-CNNs. This model is included since it is more intuitive to the power engineer because the underlying data set is a one-dimensional time series. Table I presents a summary of the CNN architecture. All kernels used have a size of 3.

TABLE I
1D-CNN Model Summary

n_{layer}	**Layer Type**	**Output Shape**	$n_{\text{parameters}}$
1	`conv1d`	`(None, 29, 64)`	256
2	`conv1d`	`(None, 27, 64)`	12352
3	`max_pooling1D`	`(None, 13, 64)`	0
4	`dropout`	`(None, 13, 64)`	0
5	`flatten`	`(None, 832)`	0
6	`dense`	`(None, 100)`	83300
7	`dense`	`(None, 2)`	202
	Trainable Parameters:		96110

The input is first processed by two convolutional layers (layers 1 and 2), each carrying out a weighted convolution operation with 64 filters. A `ReLu` function is employed as activation in both layers. After the first two activations, a `maxpooling1D` (layer 3) improves the robustness of the network to noise by effectively decreasing the number of features and selecting the most prominent ones. A `dropout` layer (layer 4) is added right after the pooling layer to prevent further overfitting (i.e., it restricts the network from "memorizing" patterns seen in the training instances). The flatten layer (layer 5) converts the multi-dimensional tensor to a one-dimensional vector. This vector is passed to a fully connected layer (`dense` , layer 6) with 100 neurons and a ReLu activation. Finally, another fully connected layer (layer 7) with a softmax activation σ is used to give a probabilistic interpretation to the network output y. Let $\mathbf{z} = [z_1 \quad z_2]^T$ be the input to the last layer, then

$$y = \text{argmax}\left[\sigma\left(\begin{bmatrix} z_1 \\ z_2 \end{bmatrix}\right)\right] = \text{argmax}\begin{bmatrix} \frac{e^{z_1}}{e^{z_1}+e^{z_2}} \\ \frac{e^{z_2}}{e^{z_1}+e^{z_2}} \end{bmatrix} \quad (1)$$

The output encoding is as follows: `1` means an oscillation detected from the input data. At the same time, `0` represents that no oscillation is identified from the passed time window, and thus the power system is safe. The loss function for parameter optimization corresponds to a categorical cross-entropy, as commonly done in classification problems. A visual illustration of the 1D-CNN model is presented in Fig. 2. Training and validation results are shown in Fig. 3.

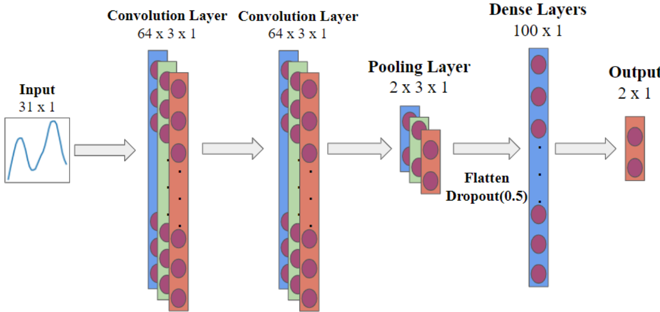

Fig. 2. Illustration of the 1D-CNN model.

D. 2D-CNN Model

In contrast with the 1D-CNN model, the 2D-CNN model takes images as inputs. The operation, however, is similar to the one of the 1D-CNN network, and thus, a detailed explanation is therefore omitted.

The plots (as RGB images) of the time series data are used as training and validation inputs, such as the one-second windows in Fig. 1. The architecture is illustrated in Fig. 4.

By inspection, it is straightforward to notice that the 2D-CNN model is more complex than the 1D-CNN, having a significantly larger number of parameters (cf., $n_{\text{parameters}}^{\text{2D-CNN}} = 2222690$ and $n_{\text{parameters}}^{\text{1D-CNN}} = 96110$) and more layers (cf., $n_{\text{layers}}^{\text{2D-CNN}} = 19$ and $n_{\text{layers}}^{\text{1D-CNN}} = 7$). For this reason, the number of training instances has to be considerably larger to achieve significant performance.

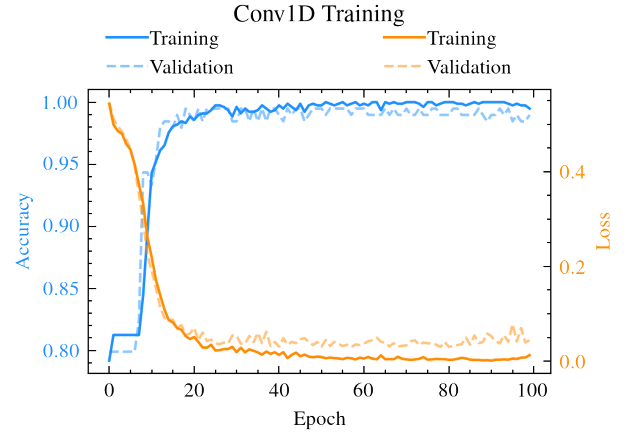

Fig. 3. Training and validation results for 1D-CNN.

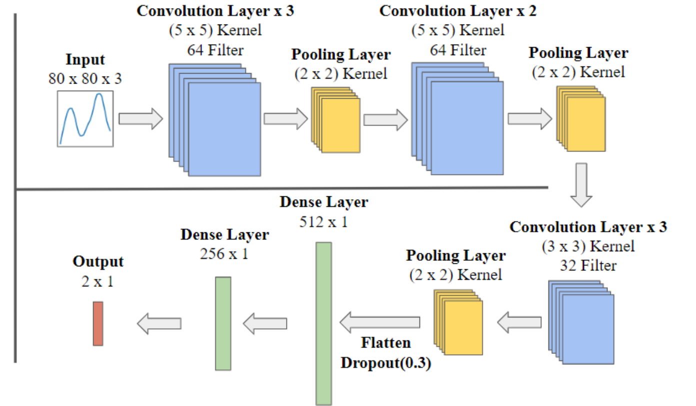

Fig. 4. Architecture of 2D CNN Model.

Fortunately, the image-based approach of the 2D-CNN allows performing data augmentation. In this case, random transformations of the images (such as rotations and noising) are carried out to expand the number of training and validation instances, important for this problem due to the low number of training instances and a large number of training parameters. Augmentation is performed automatically by the TensorFlow function `ImageDataGenerator` .

Training and validation results can be seen in Fig. 5. Notice that, since the number of parameters is larger, the model needs to be trained for more epochs. For either architecture, given the low number of data instances for this problem, a large number of epochs would be initially thought of as ideal to maximize data utilization. The number of epochs is such that the network finds a "sweet spot" where inference performance is not constrained by overfitting. So, the number of epochs is set to be a maximum of ≈ 300. The resulting model will therefore be used for inference.

III. CNN Models Optimized by TensorRT

Once the CNN models have been trained offline (i.e., on a computer or server), the next step is to prepare them for

deployment in a target for real-time inference. The library `TensorRT` is used to convert the model from frameworks such as TensorFlow/Keras and PyTorch to CUDA-compatible code, to be deployed to the target device. The target for our study is the NVIDIA Jetson TX2. For this work, the CNNs were developed in TensorFlow, so the trained models can be exported either as `*.hdf5` files or `*.h5` files. The former format is preferred since fewer intermediate steps are required to import the model within the `TensorRT` library.

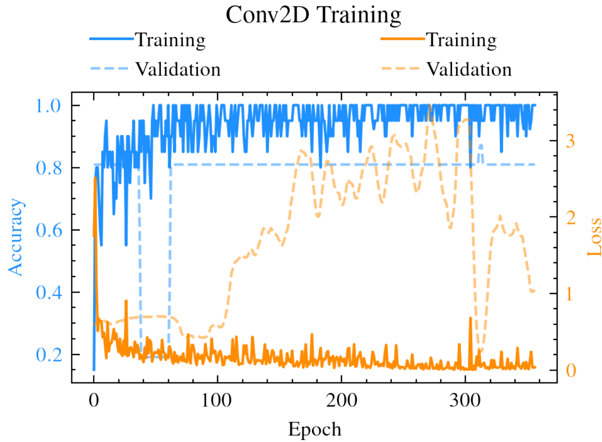

Fig. 5. Training and validation results for 2D-CNN.

The conversion and code optimization process is outlined in Fig. 6. As mentioned before, the first step is to save the trained models in a suitable format such as `.hdf5` (for the model weights; so the model would have to be rebuilt and saved) or `.h5` (for the full model). The trained model is then saved to the `TensorRT`-compatible format `.pb`. Once in `TensorRT`, the model could be deployed directly to the target, but an additional optimization process can be carried out to improve real-time performance.

`TensorRT` counts with a set of optimization routines to improve inference performance (i.e., the time it takes to complete inference provided inputs to the model). As part of the actions, while optimizing the code, layers with unused outputs are removed, operations with similar parameters are combined, and subsequent layers are blended into one (for instance, a convolution operation and an activation function are merged into a single computational layer). Altogether, the optimization routine yields a model with a less computational burden on the target and faster real-time inference.

Table II shows a comparison of the inference time when the model is deployed on an offline computer and the NVIDIA Jetson device with and without the code optimization routines of `TensorRT`. We observe that, without optimization, the average inference time per sample is within the same order of magnitude for both the Jetson TX2 edge device and the offline computer. However, the edge device infers $\approx$ 10x faster thanks to the optimized code. This result speaks highly of the feasibility of the NVIDIA Jetson TX2 for edge deployment of a real-time oscillation detection tool: the CNNs detect a forced oscillation using 1 s. windows within 10 ms., approximately 3x faster than the algorithm in [13]. Also, note that the inference time for the 2D-CNN is faster than that of the 1D-CNN. This is due to the most extensive use of optimized linear algebra routines in image convolution operations.

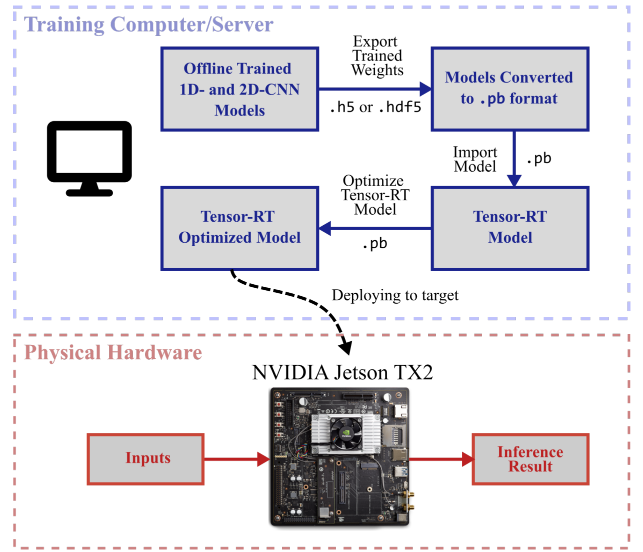

Fig. 6. Outline of CNN model development for deployment on NVIDIA Jetson TX2 to perform real-time inference.

TABLE II
AVERAGE INFERENCE TIME USING DIFFERENT HARDWARE DEVICES

Hardware/Model	1D-CNN	2D-CNN
Windows PC Intel i7-7700HQ 2.80 GHz NVIDIA GeForce GTX 1060 (TensorFlow/Keras Model)	96.931 ms	67.312 ms
NVIDIA Jetson TX2 Non-optimized by TensorRT	74.290 ms	38.379 ms
NVIDIA Jetson TX2 Optimized by TensorRT	**9.787 ms**	**3.410 ms**

IV. DETECTION OF FORCED OSCILLATIONS

The performance of the CNN-based forced oscillation detection method on the NVIDIA Jetson TX2 is evaluated with two different experiments. On the one hand, ambient data *never seen neither during training nor validation* are used as inputs. On the other hand, synthetic waveforms emulating oscillations created by a signal generator are fed to the NVIDIA Jetson's I2C input ports through a bespoke analog to digital conversion board. Note that the time series require additional preprocessing (for instance, generating the plots from the time series data for the 2D-CNN) in the Jetson's CPU before performing inference in the device's GPU. From the top two plots in Fig. 7, we observe that the CNNs succeed at detecting the oscillation and keep providing correct predictions while the event is active. However, accuracy is not 100%, as expected. A simple running window algorithm can be used to discard false positives by keeping track of the CNN recent predictions (e.g., if most of the inferences in the last 2 s are `0`, then the CNN's `1` output should be discarded). Both the 1D- and 2D-CNN detect the oscillations using measurements at different grid locations where they could be deployed.

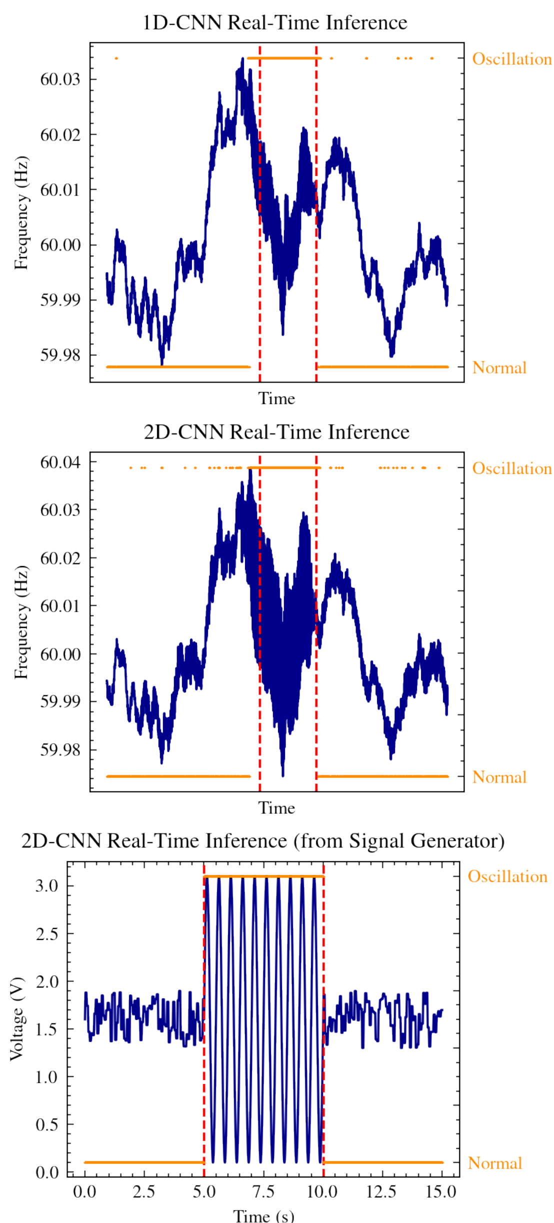

Fig. 7. Real-time inference results from ambient data (top two figures) and synthetic signal generation (bottom plot).

The plot at the bottom of Fig. 7 illustrates how oscillations are detected when an external signal is applied to the Jetson TX2 board using a signal generator. The forced change results from superimposing a sinusoid on a noisy signal. Accuracy improves compared to the ambient data experiment since the signal is not as "challenging" as those from the grid. The result shows that CNNs can learn the patterns of an oscillation using data from one system and then identify such events in another (i.e., *transfer learning* [20]).

V. Conclusions

We introduced an ML-based approach for detecting forced oscillations in power grids using an IoT edge device, an NVIDIA Jetson TX2. Two NN models, 1D- and 2D-CNNs, respectively, were trained using the TensorFlow/Keras framework. Then, the trained model code was optimized using the `TensorRT` library for real-time execution at the edge. Optimized code has proven to be faster than offline execution on the hardware. We evaluated the performance of the proposed CNNs on two different experiments: using real-world ambient data and feeding the NN input with oscillation signals created from a signal generator. The pipeline has proven to be a practical and feasible solution for oscillation detection in IoT-based monitoring systems based on the observed results.

References

[1] P. B. Reddy and I. A. Hiskens, "Limit-induced stable limit cycles in power systems," in *2005 IEEE Russia Power Tech*, Jun. 2005, pp. 1–5.
[2] W. Xuanyin, L. Xiaoxiao, and L. Fushang, "Analysis on oscillation in electro-hydraulic regulating system of steam turbine and fault diagnosis based on PSOBP," *Expert Syst. Appl.*, vol. 37, no. 5, pp. 3887–3892.
[3] R. Xie and D. Trudnowski, "Distinguishing features of natural and forced oscillations," in *2015 IEEE Power Energy Society General Meeting*.
[4] V. Jain, S. T. Nagarajan, and R. Garg, "Study of forced oscillations in two area power system," in *2018 2nd IEEE International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES)*.
[5] L. Vanfretti, M. Baudette, J. L. Domínguez-Garcíaz, A. White, M. S. Almas, and J. O. Gjerdeóy, "A PMU-based fast real-time sub-synchronous oscillation detection application," in *2015 IEEE 15th International Conference on Environment and Electrical Engineering (EEEIC)*, pp. 1892–1897.
[6] J. Follum and J. W. Pierre, "Detection of periodic forced oscillations in power systems," *IEEE Trans. Power Syst.*, vol. 31, no. 3, pp. 2423–2433.
[7] F. Ghorbaniparvar and H. Sangrody, "PMU application for locating the source of forced oscillations in smart grids," in *2018 IEEE Power and Energy Conference at Illinois (PECI)*.
[8] M. Ghorbaniparvar, "Survey on forced oscillations in power system," *Journal of Modern Power Systems and Clean Energy*, vol. 5, no. 5.
[9] D. J. Trudnowski and R. Guttromson, "A strategy for forced oscillation suppression," *IEEE Trans. Power Syst.*, vol. 35, no. 6, pp. 4699–4708.
[10] W. Hu, J. Liang, Y. Jin, F. Wu, X. Wang, and E. Chen, "Online evaluation method for low frequency oscillation stability in a power system based on improved XGboost," *Energies*, vol. 11, no. 11, p. 3238, Nov. 2018.
[11] D. N. Sidorov, Y. A. Grishin, and V. Šmidl, "On-line detection of inter-area oscillations using forgetting approach for power systems monitoring," in *2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE)*, vol. 3, Feb. 2010, pp. 292–295.
[12] H. Cho, S. Oh, S. Nam, and B. Lee, "Non-linear dynamics based sub-synchronous resonance index by using power system measurement data," *IET Gener. Transm. Distrib.*, vol. 12, no. 17, pp. 4026–4033, 2018.
[13] H. Khalilinia and V. Venkatasubramanian, "Subsynchronous resonance monitoring using ambient high speed sensor data," *IEEE Trans. Power Syst.*, vol. 31, no. 2, pp. 1073–1083, Mar. 2016.
[14] Wind SSO Task Force, "Wind energy systems Sub-Synchronous oscillations: Events and modeling," IEEE Power & Energy Society, Tech. Rep. PES-TR80, Jul. 2020.
[15] J. Liu, W. Yao, J. Wen, H. He, and X. Zheng, "Active power oscillation property classification of electric power systems based on SVM," *J. Appl. Math.*, vol. 2014, May 2014.
[16] M.-I. Ayachi, L. Vanfretti, and S. Ahmed, "A PMU-Based machine learning application for fast detection of forced oscillations from wind farms," Dec. 2020.
[17] "Speed up TensorFlow inference on GPUs with TensorRT," https://blog.tensorflow.org/2018/04/speed-up-tensorflow-inference-on-gpus-tensorRT.html, accessed: 2021-10-26.
[18] A. Bokovoy, K. Muravyev, and K. Yakovlev, "Real-time vision-based depth reconstruction with NVidia jetson," in *2019 European Conference on Mobile Robots (ECMR)*, Sep. 2019, pp. 1–6.
[19] M. Goyal, N. D. Reeves, S. Rajbhandari, and M. H. Yap, "Robust methods for Real-Time diabetic foot ulcer detection and localization on mobile devices," *IEEE J Biomed Health Inform.*
[20] A. Géron, *Hands-on Machine Learning with Scikit Learn and Tensorflow Concepts*, 2nd ed. O'Reilly, 2019.