Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 56-2 (2023) 8744–8751

Deep Learning-based Sequence Modeling for Advanced Process Control in Semiconductor Manufacturing

Filippo Dalla Zuanna ** **Natalie Gentner** *,** **Gian Antonio Susto** **

* *Infineon Technologies AG, Am Campeon 1-15, 85579 Neubiberg, Germany (e-mail: natalie.gentner@infineon.com).*

** *Department of Information Engineering, University of Padova, Via Gradenigo 6/B, 35131 Padova, Italy (e-mail: gianantonio.susto@unipd.it).*

Abstract: Semiconductor manufacturing is one of the most data-intensive industries in manufacturing. Many so-called Advanced Process Control (APC) approaches based on equipment or process data have been presented over the years to improve quality and efficiency, reduce waste and increase energy savings. While on the one hand, intensive production leads to large amount of data available for the development of Machine Learning-based technologies, on the other hand, the high-mix production, multiple processes and machines lead researchers and developers to deal with 'small data' scenarios. In this context, domain adaptation approaches are highly relevant to enhance scalability. In this work, we are extending a state-of-the-art Deep Learning architecture called Domain Adversarial Neural Network based Alignment Model (DBAM) by considering new sequence learning layers. In the real-world case study, the proposed architecture is shown to achieve higher accuracy, allow for better management in the absence of large amounts of data and make previously trained models reusable in similar scenarios.

Keywords: Deep Learning, Domain Adaptation, Etching, Industry 4.0, Process Control, Sequence Modeling, Semiconductor Manufacturing, Soft Sensing, Virtual Metrology

1. INTRODUCTION AND RELATED WORK

Due to new digital technologies and monitoring/control approaches, the industrial sector is nowadays facing a continuous increase in process complexity: while such new approaches promise to improve productivity and quality, the associated data complexity can be a heavy burden to bear. The semiconductor manufacturing (SM) sector is no exception because its processes are extremely complex and resource-intensive. Machine Learning (ML)-based technologies have emerged strongly in recent years, leading to improved production performances, increased quality and reducing waste and inefficiencies. This aspect is favored by state-of-the-art approaches and technologies that leverage computational resources and the collection of large amounts of data and methods to analyse collected data to extract useful information. While in past years, 'shallow' ML methods like regularization (Susto et al., 2012), support vector machines (Kang et al., 2016) or tree-based methods (Chen et al., 2020) dominated ML-based technologies in the semiconductor industry, recently Deep Learning (DL)-based approaches have become more popular. DL approaches have the advantage of typically guaranteeing higher accuracy (Shim and Kang, 2022) and more flexibility to deal with heterogeneous data sources (Maggipinto et al., 2022). Associated with DL-based approaches for SM, several works in the literature focus on two aspects and how to deal with them Dalle Pezze et al. (2021): (i) times-series data; (ii) scalability. While (i) is motivated by the wide diffusion of time-series data in SM equipment and processes, (ii) is motivated by the high number of machines, products and conditions in SM.

To deal with (ii), Transfer Learning (TL) has been recently applied in SM (Imoto et al., 2019): TL (James et al., 2013) is an ML research problem that aims at exploiting the knowledge gained by solving one problem and by applying it to a different but related one. In this context, Domain Adaptation (DA) (Zhang, 2019) refers to a particular case of TL, where the so-called domains corresponding to data sets under consideration follow different distributions. While DL-based adaptation methods that enable semi- or unsupervised learning exist, using labels can drastically reduce the task effort. Nevertheless, an effective model transfer is mandatory for productive rollouts and fab-wide usage. While descriptive features reduce data size and often model complexity, the usage of time series data

[1] Infineon Technologies AG is gratefully acknowledged for partially financing this research activity. This study was partially carried out within the MICS (Made in Italy – Circular and Sustainable) Extended Partnership and received funding from Next-GenerationEU (Italian PNRR – M4 C2, Invest 1.3 – D.D. 1551.11-10-2022, PE00000004). Moreover this study was also partially carried out within the PNRR research activities of the consortium iNEST (Interconnected North-Est Innovation Ecosystem) funded by the European Union Next-GenerationEU (Piano Nazionale di Ripresa e Resilienza (PNRR) – Missione 4 Componente 2, Investimento 1.5 – D.D. 1058 23/06/2022, ECS00000043).

10.1016/j.ifacol.2023.10.058

has two main advantages, namely reduced preprocessing effort combined with avoidance of information loss hence improved results

A perfect case study for (i) and (ii) is Virtual Metrology (VM), one of the most popular ML-based technology in SM. VM aims at predicting unmeasurable/costly quantities from sensors or information that is already available. VM is successfully developed in the literature for processes like chemical vapour deposition or etching (Schirru et al., 2012), where sensor data presents itself as a time-series. In many cases, metrology is placed after process equipment composed of two or more subtools (chambers) that process silicon wafers in parallel. Even if such subtools are built identically, the recorded data (in form of time-dependent raw sensor measurements) shows different distributions. This disparity can be caused by imperfections in the machines, non-identical settings/environments or different process histories. While VM literature is rich, there is still a lack of approaches presented in the literature that may deal naturally with both time-series data and TL.

This work aims to extend previous research (Gentner et al., 2021) that introduces the DANN-based Alignment Model (DBAM). It aligns two domains by projecting one input space to the other while allowing comparability of input data before and after the mapping. Since the comparison happens in the original feature space, it supports interpretability as a key aspect for acceptance and usage in production. DBAM is applied to time series data and used to showcase its potential for scalability. With the help of a VM case study and real-world data from an etching tool, we aim at introducing new sequence learning layers (LSTM, TCN, 1DCNN) and combinations of those to DBAM. Novel contributions of this work are:

- Presenting a common DA model approach for multiple subtools that is tested on different DL architectures suitable for time series data; by avoiding dedicated models for single tools, we avoid information waste, monitoring effort and increase effectiveness;
- Showcasing the power of time series suitable architectures for domain adaptation approaches, thanks to a VM case study on real-world SM data;
- Enhancing benchmark model selection and testing of mixed architecture model to make the best usage of each strength: for example usage of 1DCNN layers for feature creation followed by LSTM for additional time consideration.

The rest of the paper is organized as follows: Section 2 covers the methodology of DBAM and recaps DL architecture suitable for time series data. Section 3 describes the semiconductor etching data. Section 4 introduces the experimental design and presents experiments and results. Finally, section 5 draws conclusions and defines further research directions.

2. METHODOLOGY AND ARCHITECTURES

2.1 DBAM

DANN-Based Alignment Model (DBAM) (Gentner et al., 2021) bases its development on Domain Adversarial Neural Networks (DANN) (Ganin et al., 2016). DBAM adaptation

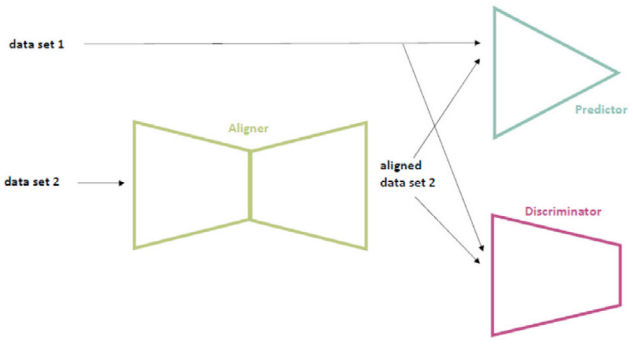

Fig. 1. DBAM system structure (Gentner et al., 2021)

is performed using an encoder model inspired by an autoencoder structure. This allows for the interpretability of the aligned data with the possibility of using different feature sizes between source and target domains. The DA is performed directly in the input space, being able to compare the samples of the two domains instead of relying only on latent features. DBAM is trained using an adversarial training approach (Gulrajani et al., 2017). The three main components of DBAM are (see Fig. 1):

1 A prediction model P (called predictor), that performs the modeling task;
2 An encoder model A (called aligner), that performs the adaptation from target to source domain;
3 A classifier model D (called discriminator), that distinguishes the domains during adversarial training.

As discussed in the following, the various components are designed based on modelling task and data type.

2.2 1DCNN and TCN

Temporal Convolutional Networks (TCN) (Bai et al., 2018) refer to a family of models that exploit the characteristics of Convolutional Neural Networks (CNN) and adapt them to one-dimensional input respective time sequences. The convolutions here are causal: there is no information leakage from future to past and output at time t is convolved only with elements from time t and earlier in the previous layer or input. The architecture can take a sequence of any length and map it to an output sequence of the same length, just as with a Recurrent Neural Network (RNN) model. TCN use a one-dimensional CNN (1DCNN) architecture where each feature map has the same length as the input layer. Zero padding of defined length is added to keep subsequent layers and the previous ones of the same length. TCN in addition implement dilated convolution and residual connections, as presented in WaveNet (van den Oord et al., 2016). A dilated convolution is a convolution where the filter is applied over an area larger than its length by skipping input values with a certain step. The network can perform the convolution on a coarser scale, like pooling or convolutions with a stride greater than 1, but the output has the same size as the input. Formally, let $x = x(t)$ be a time series input function and let $\omega(t)$ define a (finite) kernel and t describing the index of the corresponding one-dimensional grid. Then the dilated convolution operation s_d on element t of the sequence is defined as:

$$s_d(x(t)) = (x *_d \omega)(t) = \sum_{a+d\cdot s=t} x(a)\omega(s). \quad (1)$$

with $*_d$ dilated convolution and d is the dilation factor. The summation makes the skipping of some values visible and dilated convolution with dilation $d = 1$ yields the standard convolution. Stacked dilated convolutions enable networks to have very large receptive fields with just a few layers while preserving the input resolution throughout the network as well as computational efficiency. Usually, the dilation doubles up to a certain limit in a layer; this is repeated with the stacking of new layers. It has two benefits: first, exponentially increasing the dilation factor results in exponential receptive field growth with depth. Second, stacking these blocks further increases the model capacity and the receptive field size. Residual connections consist of a solution that permits the networks' layers to learn modifications to the identity mapping rather than the entire transformation by applying an activation function to the sum of the input and the transformed input. This has repeatedly been shown to benefit very deep networks since it preserves more information during the complete flow into the model.

2.3 LSTM

Long-Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) models are an extension of the more general Recurrent Neural Networks (RNN) and improve some of its aspects through optimizations of the information process. RNN focus on processing data that correlates in the time domain, being able to process any time series data type. Therefore, the model introduces recurrent connections that permit the elaboration of information at timestamp t along with the information taken from previous data. LSTM introduce some improvements in the management of the past information to overcome the lack of standard RNNs that limit their learning about short-term and not long-term dependencies (Shewalkar, 2019). This is done by adding a set of gated units to the basic structure of an RNN to manage the storage of the information, defining the LSTM memory cell and using notations in Fig. 2. In particular, the gates are of three

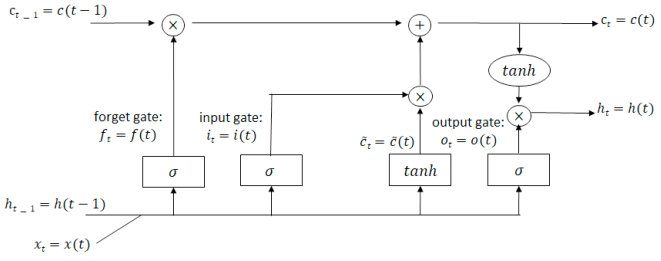

Fig. 2. LSTM cell structure.

.

different categories:

1 Input gates, which manage the input utilization from the memory cell;
2 Output gates, which weigh how much the current value stored in the memory cell is present in output;
3 Forget Gates, which manage how much the current value stored has to be forgotten.

The use of those gates drastically reduces the vanishing and exploding gradient problem which occurs in RNN. Gates also makes use of the sigmoid function σ to determine when to add or remove information.
First, the forget gate decides when to keep or discard information by looking at the current input $x(t) = x_t$ for timestep t and the previous output respective hidden representation from the previous step $h(t-1) = h_{t-1}$. Let f be the layer transformation that is parameterized by W_f. Then, the forget gate/layer output $f(t)$ is defined as (in form of a neural network layer output with sigmoid activation function)

$$f_t = f(t) = \sigma(W_f \cdot [h(t-1), x(t)] + b_f) \quad (2)$$

where W_f as parametrization is the weight matrix for the forget gate and b_f is the bias term. Next, the input gate determines what new information is going to be stored in the cell state. Which elements to store i_t is determined as follows: let i be the sigmoid layer from the input gate parameterized by W_i and the bias b_i. Let $\tilde{C}$ be the tanh layer that forms the input gate parameterized by W_c and bias b_c. Then, the input gate/layer outputs $i(t), \tilde{C}(t)$ are defined as (in form of a neural network layer output with sigmoid and tanh activation function)

$$i_t := i(t) = \sigma(W_i \cdot [h(t-1), x(t)] + b_i); \quad (3)$$

$$\tilde{C}_t = \tilde{C}(t) = tanh(W_c \cdot [h(t-1), x(t)] + b_c). \quad (4)$$

Then, the cell update with the new cell state $C(t)$ just multiplies the old state by the forgetting factor and adds the new input information:

$$C_t := C(t) = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t. \quad (5)$$

Finally, the output gate, which decides what to propagate forward in the output, is computed as follows. Let the output gate be defined as a sigmoid layer o parameterized by W_o and bias b_o. Then:

$$o_t := o(t) = \sigma(W_o \cdot [h(t-1), x(t)] + b_o); \quad (6)$$

$$h_t := h(t) = o_t \cdot tanh(C_t). \quad (7)$$

3. DATA DESCRIPTION

We consider a real-world case study related to an etching process. The dataset corresponds to the one in the reference work (Gentner et al., 2021). Data is collected over 2 years from two identical, parallel-running tools. All samples contain physical quantities and process information collected by sensors of the machines that operate during the process. A subset of samples gets data quantities added from a metrology tool that performs inline measurements on the wafer, e.g critical dimension and layer thickness. The latter is chosen as output/labels for the VM prediction task since it shows high correlations to the quantities' evolution during the etching process. The 33 measured quantities contained in the dataset correspond to the input features and are collected from two different equipment, corresponding to the source and the target domains respectively. The raw sensor signals see preprocessing steps: first constant features are deleted. Then quantile-based outlier removal is applied, and the data is normalized with a min-max scaler. Equal distributed upsampling of timestamps is applied to deal with time series length variations caused

by the endpoint detection present in the process. The data collection is limited to one process to avoid recipe-specific interference. A train-validation-test split 80:10:10 is done on both datasets. In summary, the datasets have the following dimensions, excluding the labels, in the format [samples : sequence length : features]:

- Train dataset A - [968 : 1024 : 33];
- Validation dataset A - [120 : 1024 : 33];
- Test dataset A - [121 : 1024 : 33];
- Train dataset B - [1017 : 1024 : 33];
- Validation dataset B - [127 : 1024 : 33];
- Test dataset B - [127 : 1024 : 33].

4. EXPERIMENTS AND RESULTS

4.1 VM prediction task

The VM regression uses given physical quantities measured during the etching process based on sensor process data to model the layer thickness. All the design choices are reported in the following to enhance reproducibility. The following hyperparameters are chosen based on reference (Gentner et al., 2021): the training epochs (fixed at 300), the Adam optimizer, the learning rate value (lr = 0.00001), the learning rate schedule (exponential decrease after 50 epochs), the schedule of the early stopping (which has min_δ = 0.00001 and $patience$ = 20) and the batch size (fixed at 32). The remaining hyperparameters are discussed in the following parts: each choice is based on a grid search approach and its validation results.

1DCNN Benchmark Model The baseline model (Gentner et al., 2021) is taken as the benchmark and the comparison starting point with some procedure characteristics used as a reference to train and test the TCN and LSTM enhanced models. The 1DCNN predictor model has the following elements: a 1DCNN layer with 16 filters followed by a batch normalization layer; a 1DCNN layer with 8 filters followed by a max-pooling layer with 4 as pool dimension and a batch normalization layer; a fully connected layer with 16 units applied to the flattened version of the features extracted previously, followed by a batch normalization layer; finally an output layer(1 fully connected unit) that use sigmoid as activation function. The 1DCNN layers have a kernel size of 17 and a stride of 2. The activation function used is the Exponential Linear Unit (ELU).

TCN predictor model The TCN predictor model is tested with two different configurations: the (i) first one can be considered the canonical architecture, which uses only the last part of the temporal features extracted by the TCN layers to make the regression prediction. Since the architecture has a limited receptive field in the last part of the sequence, latent features are not originated from the whole sequence limiting the capabilities of this configuration. The (ii) second configuration - which was the most accurate in our experiments - uses a series of TCN layers to extract a time sequence of features that is then reduced with a 1DCNN layer (Thill et al., 2021). The reduction using 1DCNN decreases the number of model parameters required when directly using a fully connected output layer. Here, the features processed are taken from the whole sequence, but the model is more complex. In both configurations, the activation function of the output layer is linear. Based on our experiments, the best configuration design was:

- Best architecture configuration: (ii).

With that configuration choice, the other hyper-parameters were chosen following Cao et al. (2021) parameters and by applying a grid search. The overall parameters tested are the following, where we highlighted the ones that obtained the best results:

- Activation function (for the 1DCNN and the fully connected layer): ReLU or **Leaky ReLU**;
- Number of TCN layers: **1** or 2;
- Number of dilated convolution blocks: **1**, 2 or 4;
- Number of filters: 16, 32, **64** or 128;
- Kernel size: 3, 15 or **50**;
- Dilated convolution depth: 4 or **6**.

The final TCN predictor model has the following elements: 1 TCN layer with the parameters highlighted above; a 1DCNN layer with kernel size and stride of 5 reducing by 5 the time dimension; a fully connected layer with 16 units applied to the flattened version of the features extracted previously; finally, an output layer (1 fully connected unit) to make the prediction.

LSTM predictor model The LSTM predictor model is tested for 3 configurations inspired by (Zhang et al., 2018; Mutegeki and Han, 2020). The first one (i) uses the last temporal features extracted by the LSTM layers to make the regression. In this case, there could be problems related to learning long-term dependencies meaning they could be obfuscated by short-term ones, resulting in a similar receptive field problem as for TCN. The second (ii) version uses stacked LSTM layers to extract a time sequence of features that is then reduced with a 1DCNN layer. The reduced time sequence is delivered to a fully connected layer with 16 units. The 1DCNN and fully connected layers have Leaky ReLU as fixed activation function. The third configuration (iii) reverses the LSTM and 1DCNN elements. Hence, the input is reduced in size before extracting the features due to the LSTM layers. This leads to simplification and lightening of the model as the 1DCNN levels are less demanding from a computational point of view while reducing dimensionality. In all configurations, the activation function of the output layer is linear. The other hyper-parameters are taken from (Reimers and Gurevych, 2017). Based on our experiments, the best configuration design in this case was:

- Best architecture configuration: (iii).

Hyperparameters included in the grid search are (the highlighted parameters are again the ones that obtained the best results):

- Number of LSTM layers: from **1** to 3;
- Number of units per layer: 25, **50** or 100;
- Dropout rate: **0**, 0.1 or 0.25.

The final LSTM predictor model has the following elements: a 1DCNN layer with kernel size and stride of 5 reducing by 5 the time dimension; 1 LSTM layer with the parameters highlighted above; a fully connected layer with 16 units applied to the flattened version of the features

extracted previously; finally an output layer (1 fully connected unit) to make the prediction.

4.2 Results on VM regression task

The analysis is performed using only the first division of the dataset and the statements are strengthened using Cross Validation (CV) at the end. The latter confirmed the statements written below.

Table 1. Predictors performance evaluation

Evaluation of the predictors				
Dataset	MAE	ME	EV	R2
Benchmark predictor				
Source (B) validation	0.0667	0.310	-	-
Source (B) test	0.0705	0.256	0.822	0.816
Target (A) test	0.227	0.484	**0.738**	-0.573
TCN predictor				
Source (B) validation	**0.0531**	0.185	-	-
Source (B) test	**0.0560**	**0.164**	0.895	**0.893**
Target (A) test	**0.129**	**0.357**	0.720	**0.377**
LSTM predictor				
Source (B) validation	0.0535	**0.177**	-	-
Source (B) test	0.0624	0.165	**0.900**	0.892
Target (A) test	0.141	0.338	0.689	0.302

Benchmark predictor results The performance metrics of the baseline model are shown in Table 1 and Fig. 3 visualize the predictions. The metrics are the mean absolute error (MAE), the maximal residual error (ME), the explained variance regression score (EV) and the R-square (coefficient of determination) regression score (R2).

TCN predictor results The improvements with using TCN architecture are presented in Table 1. The task is learned from the model better than the benchmark, with MAE and ME lower for the validation and the test source dataset. Therefore, the higher complexity of the TCN model leads to more stable training and better results on the learned regression task. The TCN architecture extracts more general properties since the results are also improved for the target domain, demonstrating a better generalization capability. Also, the predictions in Fig. 4 show this aspect. In this case, the target domain predictions (red dots) are closer to the optimal prediction line compared to the benchmark, further showing the claim regarding the best generalization properties. The source domain predictions (blue dots) show the same better performances. This is followed by the test sets (grey dots) that behave the same as the training ones. The complexity of the model seems the only drawback of the TCN architecture that proves to generalize better than the benchmark, along with the best results. However, this model doesn't solve the regression task for the target domain, even with its generalization properties.

LSTM predictor results Table 1 presents the again improved results; the MAE and ME are lower for the validation and the test source dataset. The extraction of more general properties takes place there too, with the LSTM architecture demonstrating better generalization capability than the benchmark, leading to better target test accuracy. This is supported by the visualization in Fig. 5. The better performances of the architecture compared to the benchmark are shown with the source domain predictions (blue dots) that are closer to the optimal prediction line and the same for the target domain predictions (red dots). The test sets (grey dots) behave consistently with the training sets. The LSTM model appears more balanced than the TCN model, with architecture more stable concerning parameter modification, including lower complexity while showing similar performance.

Prediction plots - Fig. 3, Fig. 4, Fig. 5: in the following, source train set (blue dots), target train set (red dots), source and target test sets (grey dots).

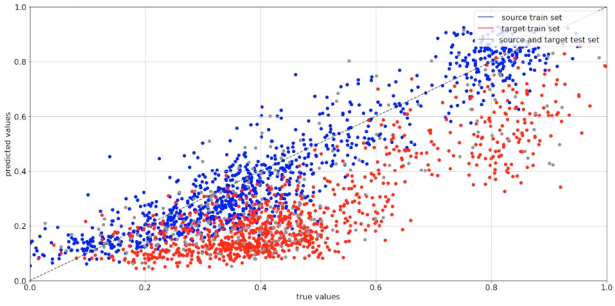

Fig. 3. Benchmark predictor

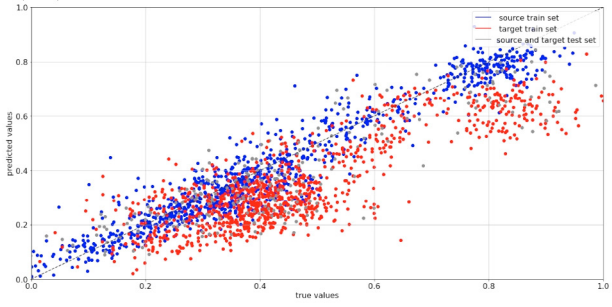

Fig. 4. TCN predictor

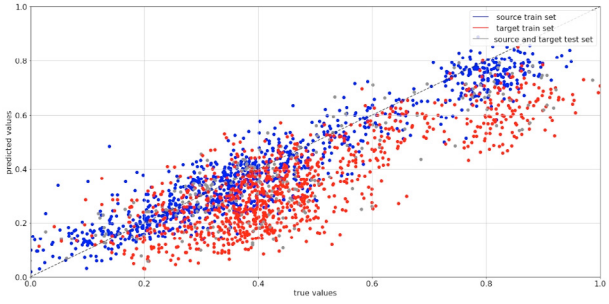

Fig. 5. LSTM predictor

4.3 Adversarial Domain Adaptation task

In Adversarial Domain Adaptation, the network objective is to standardize samples from the target domain to appear more similar to the source domain elements. The adversarial training procedure here is performed as in reference work (Gentner et al., 2021). The benchmark model is taken as the comparison starting point. In the present work, we test several changes from the benchmark DBAM by introducing TCN and LSTM layers. In our experiment, we performed ablation studies to evaluate the effect of each choice change: we first changed the predictor model keeping the adversarial elements as in the benchmark; we

then introduced the new aligner models and completed the study with the changes in the discriminator part. The parameters maintained from the reference are:

- Available data - same data used with 32 as batch size;
- Training epochs and early stopping criterion - fixed at 300 without early stopping;
- Optimizer - Adam;
- Learning rates and their schedule - fixed for the aligner at 0.00001 and the discriminator at 0.0001, without decreasing during training;
- Gradient penalty regularization (Gulrajani et al., 2017) - with a loss weight fixed to 10.

For better initialization, a warm-up training of the aligner is done as recommended. Regarding the training procedure, the following hyper-parameters were tested:

1 Adversarial loss weight (α): chosen from the set {0.1, 0.25, 0.5, 1, 5};
2 Discriminator training extra steps: chosen from the set {5, 10, 20}.

The parameter α is associated with the balance between the adversarial loss and the supervised predictor loss. The overall loss is equal to the weighted sum of the two. If α is lower than 1 the training is focused on finding an alignment that preserves the performance of the prediction. With $\alpha > 1$, the model focuses more instead on generalization and the quality of the domains' alignment, so the adversarial loss is multiplied by α to enhance it. This parameter is always tested, choosing the configuration that balances the training depending on the predictor characteristics. The benchmark uses 5 for this parameter.
The second hyperparameter considered is changed only if needed after the α results, otherwise kept at the original value of 20. This parameter balances the training ratio between the aligner and the discriminator, an aspect needed to keep the competition alive between the two parts, avoiding that one suppresses the ability of the other. The decreasing of this value is needed in the presence of a complex discriminator model that is more capable of its task and so needs less training to compete, being able to save time too. The latter could also be the reason for the decrease in this parameter if a compromise on training time is necessary.

4.4 DBAM elements

Benchmark aligner The 1DCNN aligner model, taken as the reference, has the following elements: a 1DCNN layer with 32 filters, 7 as kernel size and stride of 2; an average pooling layer with 2 as pooling dimension; a 1DCNN layer with 16 filters, 7 as kernel size and stride of 2; an upsampling layer with 3 as the upsampling parameter; a 1DCNN layer with 32 filters, 21 as kernel size and stride of 1; an upsampling layer with 3 as the upsampling parameter; a 1DCNN layer with 33 filters, 33 as kernel size and stride of 1. The 1DCNN layers have ReLU as an activation function, use the valid padding parameter in the convolution(which is not causal) and have also a dropout rate of 0.15. The last 1DCNN layer, instead, has a linear activation function and no dropout.

Benchmark discriminator The 1DCNN discriminator model, taken as the reference, has the following elements: two 1DCNN layers with 32 and 16 filters, respectively, both with 17 as kernel size, the stride of 1 and each one followed by a maximum pooling layer with 4 as pooling dimension; a fully connected layer with 512 units applied to the flattened version of the features extracted previously; a series of fully connected layers with 256, 128, 64, 32 units respectively; an output layer(1 fully connected unit) that use linear activation function. 1DCNN and the fully connected layers use Leaky ReLU as an activation function.

TCN aligner The TCN aligner autoencoder structure takes inspiration from (Thill et al., 2021). The base aligner model has the following elements and hyperparameters: 1 or 2 TCN layers with 8, 16 or 32 filters and 15 or 50 kernel size; an average or 1DCNN pooling layer with 16 as compression factor; an upsampling layer of 16 to reconstruct the length of the signal; finally TCN layers in the same number and characteristics as before with 33 filters(for the last one) to reconstruct the signal.

LSTM aligner The LSTM aligner autoencoder structure takes inspiration from Li et al. (2020). The final LSTM aligner model has the following elements: 1 LSTM layer with 50 units and no dropout; an average pooling layer with 16 as compression factor; a repeat layer(see Li et al. (2020)) of 16 to reconstruct the length of the signal; finally a LSTM layer with 33 units and no dropout to reconstruct the signal.

LSTM discriminator The LSTM discriminator architecture is composed by: a 1DCNN layer with 50 filters, 5 as kernel size and stride of 5 (divides by 5 the signal length); a 1DCNN layer with 50 filters, 3 as kernel size and stride of 3; an LSTM layer with 16 units and without dropout; a fully connected layer with 512 units applied to the flattened version of the features extracted previously; a series of fully connected layers with 256, 128, 64, 32 units respectively; an output layer(1 fully connected unit) that use linear activation function. 1DCNN and the fully connected layers use Leaky ReLU as an activation function.

4.5 Results for the Domain Adaptation task

The hyperparameters and architecture selection is performed using only the first division of the dataset and looking at validation results. In the end, the most important statements are strengthened using cross-validation once selected the final configuration, which confirms the selection.

Consideration of the new predictors The first study replaces the benchmark predictor with the TCN and LSTM models shown previously. In this case, the adversarial training components remain the same as in the benchmark, so we analyse the impact of the new models applied only in the prediction part. These models are named 'TCNpredictor-DBAM' and 'LSTMpredictor-DBAM'. The test results of the different configurations with the new predictor added are compared in Table 2. Inserting a new predictor improves overall performance. Additionally, the new predictors demonstrate improved readiness to facilitate data alignment (see alignment from LSTM in Fig. 8), as the required alpha parameter (0.1 for

TCN and 0.25 for LSTM) is less than the value of 5 used in the benchmark. This one benefits the prediction results, which are the best for the LSTM architecture, supported by visualizations of the test results in Fig. 7. The TCN configuration has difficulty in the alignment due to its better generalization properties (see Fig. 6).

Prediction plots - Fig. 6, Fig. 7: in the following, source train set (blue dots), target train set (red dots), source and target test sets (grey dots).

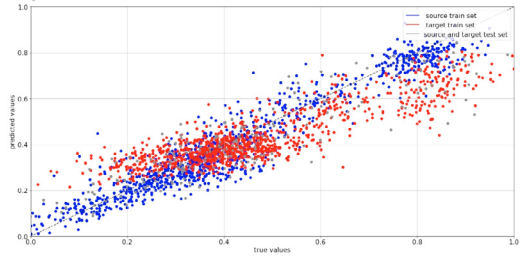

Fig. 6. TCNpredictor-DBAM: predictions plot

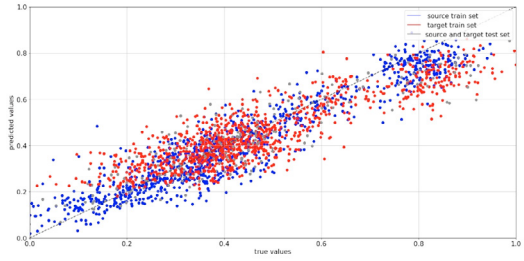

Fig. 7. LSTMpredictor-DBAM: predictions plot

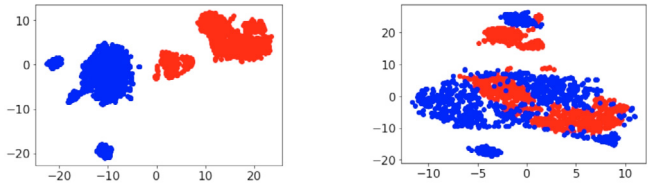

Fig. 8. LSTMpredictor-DBAM: t-SNE plot of train data (source blue dots, target red dots) before and after alignment

Consideration of the new aligners The second study is associated with replacing the benchmark aligner with corresponding TCN and LSTM models. In this case, the only component that remains the same as in the benchmark is the discriminator model. With this configuration, we analyze the impact of the new alignment architecture. These models are named 'TCNaligner-DBAM' and 'LSTMaligner-DBAM'. The TCN aligner proved not to work in this case, inserting a huge amount of noise in the signals hence losing the interpretability of the signals resulting in TCN not being suited for this part. These results conclude the analysis of the TCN-DBAM elements since the discriminator study without a functioning aligner is skipped. The configuration with the LSTM aligner works but does not improve the performance. Results are presented in Table 3. However, smoothing can be seen in the signals generated by the aligner in Fig. 9: with the LSTMaligner-DBAM configuration, the aligned signals are more regular at the expense of a greater computational complexity given by the more complicated aligner architecture.

Table 2. DBAMs test results with new predictor

DBAMs test results with new predictors				
DBAM model	MAE	ME	EV	R2
Benchmark DBAM	0.118	0.463	0.565	0.393
TCNpredictor-DBAM	0.0881	0.387	0.687	0.646
LSTMpredictor-DBAM	**0.0822**	**0.320**	**0.713**	**0.707**

Table 3. LSTMaligner-DBAM: test results

LSTMaligner-DBAM: test results				
DBAM model	MAE	ME	EV	R2
Benchmark DBAM	0.118	0.463	0.565	0.393
LSTMpredictor-DBAM	**0.0822**	**0.320**	**0.713**	**0.707**
LSTMaligner-DBAM	0.0856	0.342	0.693	0.691

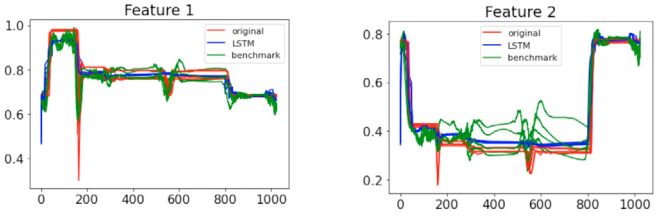

Fig. 9. LSTMaligner-DBAM: signals examples after training showing improved alignment using LSTM compared to benchmark.

Consideration of the new discriminator The last study replaced the benchmark discriminator with an LSTM-based model. This model, named 'LSTMdiscriminator-DBAM', enormously increases the required time in the DBAM training. This aspect can be improved by lowering the discriminator extra steps parameter. Furthermore, the discriminator is also more complex, so reducing its influence on training can better balance the competition between the aligner and become beneficial for training. The test results in Table 4 show that the configuration beats the benchmark also in this case. However, introducing the new discriminator doesn't improve the results compared to the LSTMaligner-DBAM configuration. Instead, the training is longer and more complicated, making this configuration not recommended.

Results recap Changing the predictor is the most beneficial improvement, significantly improving the performance for both TCN and LSTM configurations. However, while generalizing best, TCN appears more problematic in achieving a stable final configuration. Its properties make alignment more difficult using the benchmark aligner, with problems worsening with the addition of the TCN aligner. This aspect leads to the skipping of the TCN discriminator test. LSTM elements work well with introducing an aligner that generates more stable signals. However, prediction performance compared to the aligner benchmark

Table 4. Performance of various DBAM-based approaches

DBAM Model:	MAE	ME	EV	R2
Benchmark DBAM	0.118	0.463	0.565	0.393
LSTMaligner-DBAM	**0.0856**	0.342	**0.693**	**0.691**
LSTMdiscr.-DBAM	0.0917	**0.324**	0.676	0.663

is comparable, making the choice of the type indifferent if there is no preference for the shape of the signals. The LSTM discriminator also gives the same results as the benchmark counterpart. However, its training is longer and more complex, making its use not recommendable.

5. CONCLUSION

The main purpose of this work is to compare the architecture of the reference research (Gentner et al., 2021) with more sophisticated models based on TCN and LSTM, architectures developed to deal with temporal information. In the VM regression task, the results of the predictors show superiority over the benchmark, significantly improving the reliability of the estimate of the measure sought. For scalability, using an adversarial DA procedure shows its benefits; introducing the DBAM methodology that can standardize data distributions belonging to similar environments is crucial to permit the exploitation of all the capabilities of a dedicated prediction model while keeping features interpretable. LSTM-based models can solve this task, achieving similar accuracy compared to the benchmark elements and smoothing out adapted signals. Given the complexity of the considered architectures, we have reported all the design choices to provide useful guidelines for researchers and practitioners aiming at applying DBAM in their applications.

For future work, we envision implementing DBAM and the present extensions in more complex production settings; for example, taking as a reference the Virtual Metrology task in semiconductor manufacturing, we could include more tools or more recipes. We envision that more complex settings could further confirm the actual performance and benefits of more suitable time series DL architectures, such as the ones considered in this work.

REFERENCES

Bai, S., Kolter, J.Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *ArXiv*, abs/1803.01271.

Cao, Y., Ding, Y., Jia, M., and Tian, R. (2021). A novel temporal convolutional network with residual self-attention mechanism for remaining useful life prediction of rolling bearings. *Reliability Engineering System Safety*, 215, 107813.

Chen, C.H., Zhao, W.D., Pang, T., and Lin, Y.Z. (2020). Virtual metrology of semiconductor pvd process based on combination of tree-based ensemble model. *ISA Transactions*, 103, 192–202.

Dalle Pezze, D., Masiero, C., Tosato, D., Beghi, A., and Susto, G.A. (2021). Formula: A deep learning approach for rare alarms predictions in industrial equipment. *IEEE Transactions on Automation Science and Engineering*, 19(3), 1491–1502.

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V. (2016). Domain-adversarial training of neural networks.

Gentner, N., Carletti, M., Kyek, A., Susto, G.A., and Yang, Y. (2021). Dbam: Making virtual metrology/soft sensing with time series data scalable through deep learning. *Control Engineering Practice*, 116, 104914.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A.C. (2017). Improved training of wasserstein gans. *CoRR*, abs/1704.00028.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780.

Imoto, K., Nakai, T., Ike, T., Haruki, K., and Sato, Y. (2019). A cnn-based transfer learning method for defect classification in semiconductor manufact. *IEEE Trans. on Semiconductor Manufacturing*, 32(4), 455–459.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An introduction to statistical learning*, volume 112. Springer.

Kang, P., Kim, D., and Cho, S. (2016). Semi-supervised support vector regression based on self-training with label uncertainty: An application to virtual metrology in semiconductor manufacturing. *Expert Systems with Applications*, 51, 85–106.

Li, D., Li, L., Li, X., Ke, Z., and Hu, Q. (2020). Smoothed lstm-ae: A spatio-temporal deep model for multiple time-series missing imputation. *Neurocomputing*, 411. doi:10.1016/j.neucom.2020.05.033.

Maggipinto, M., Beghi, A., and Susto, G.A. (2022). A deep convolutional autoencoder-based approach for anomaly detection with industrial, non-images, 2-dimensional data: A semiconductor manufacturing case study. *IEEE Transactions on Automation Science and Engineering*.

Mutegeki, R. and Han, D.S. (2020). A cnn-lstm approach to human activity recognition. In *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, 362–366.

Reimers, N. and Gurevych, I. (2017). Optimal hyperparameters for deep lstm-networks for sequence labeling tasks. URL `https://arxiv.org/abs/1707.06799`.

Schirru, A., Susto, G.A., Pampuri, S., and McLoone, S. (2012). Learning from time series: Supervised aggregative feature extraction. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, 5254–5259.

Shewalkar, A. (2019). Performance evaluation of deep neural networks applied to speech recognition : Rnn, lstm and gru. *Journal of Artificial Intelligence and Soft Computing Research*, Vol. 9, No. 4, 235–245.

Shim, J. and Kang, S. (2022). Domain-adaptive active learning for cost-effective virtual metrology modeling. *Computers in Industry*, 135, 103572.

Susto, G.A., Schirru, A., Pampuri, S., and Beghi, A. (2012). A predictive maintenance system based on regularization methods for ion-implantation. In *2012 semi advanced semiconductor manufacturing conference*, 175–180. IEEE.

Thill, M., Konen, W., Wang, H., and Bäck, T. (2021). Temporal convolutional autoencoder for unsupervised anomaly detection in time series. *Applied Soft Computing*, 112, 107751.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A.W., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *ArXiv*, abs/1609.03499.

Zhang, J., Li, Y., Tian, J., and Li, T. (2018). Lstm-cnn hybrid model for text classification. In *2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, 1675–1680.

Zhang, L. (2019). Transfer adaptation learning: A decade survey. *CoRR*, abs/1903.04687.