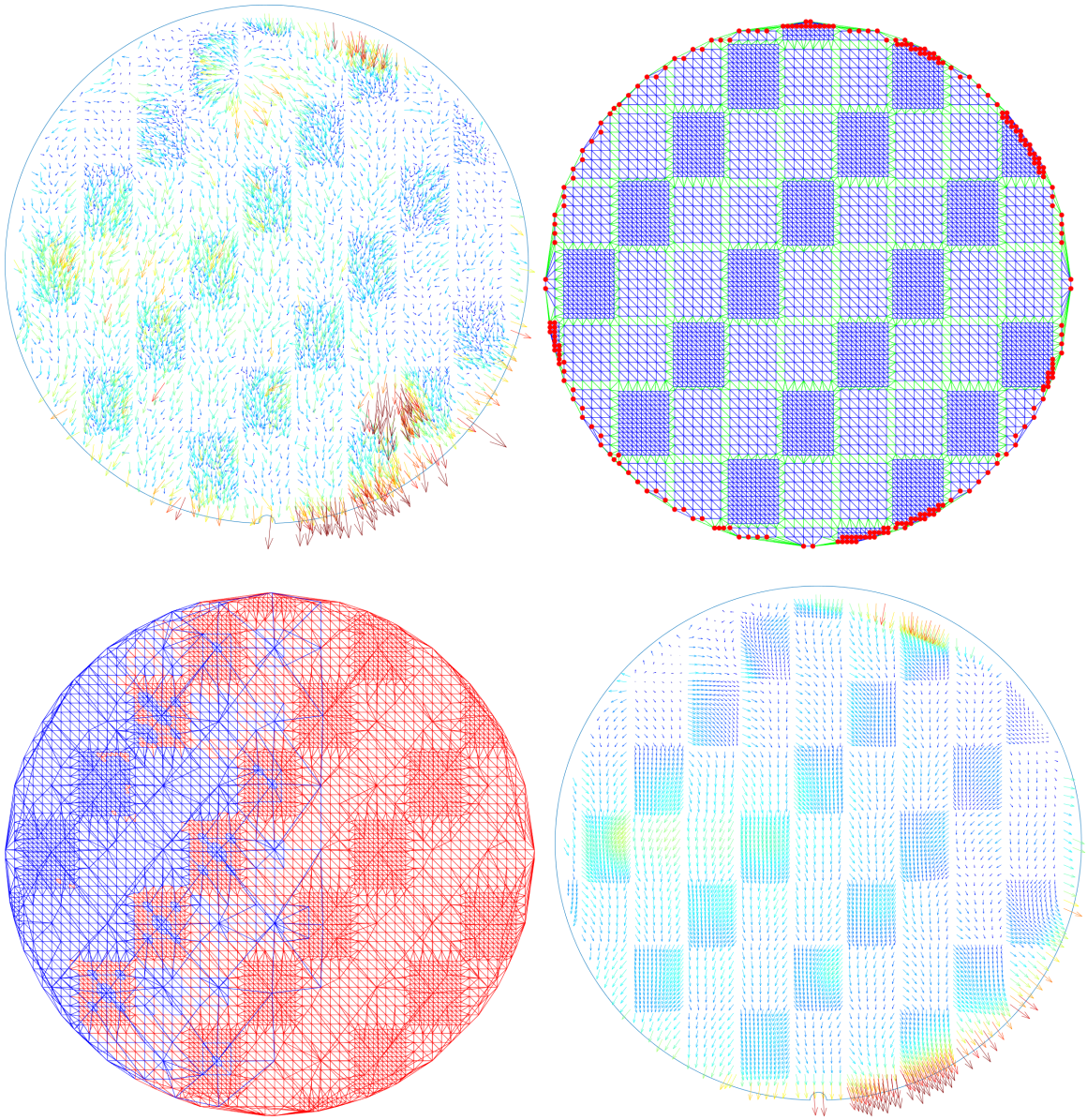
Creating a machine learning-based outlier removal algorithm that incorporates a priori knowledge of the physics

T.M. Kamminga



TU Delft

ASML

Creating a machine learning-based outlier removal algorithm that incorporates a priori knowledge of the physics

by

T.M. Kamminga

To obtain the degree of Master of Science,
in Applied Mathematics,
at the Delft University of Technology,
to be defended on Wednesday July 3, 2024 at 3:00 PM.

Student number:	4957431	
Project duration:	November 23, 2023 – July 3, 2024	
Thesis committee:	Prof. dr. ir. M.B. van Gijzen ,	TU Delft
	Dr. A. Heinlein,	TU Delft, supervisor
	Dr. ir. G.N.J.C. Bierkens,	TU Delft
	Ir. L. Bekker	ASML

TU Delft

Abstract

This thesis explores the development of a machine learning model aimed at the removal of outliers in overlay measurements. These overlay measurements give the placement error of the patterns used to manufacture semiconductors. Reducing this error is crucial for achieving precision in positioning during the semiconductor manufacturing processes. Some points in the overlay measurements are influenced by contaminants and other stochastic effects which results in measurements that do not represent the true consistent placement error. The goal is to mark these measurement points as outliers. Instead directly labeling outliers, the problem was approached as a signal-denoising task where outliers, which can have various shapes and sources, are treated as noise to be removed from the true overlay signal. The core objective then became to create a model that can effectively denoise overlay measurements in a data-driven manner. Given the absence of noise-free overlay measurements, the research adopts the Noise2Noise approach, training our deep neural network with noisy measurement pairs to learn the denoising process without direct access to clean data.

The model is built upon a message-passing neural network (MPNN) architecture, a variant of a graph neural network where information is passed to neighboring vertices of a graph. This architecture allows the model to learn directly from the measurements, which are represented by the vertices of the graph, without interpolating the data and also allows us to incorporate physics-based information into the model. This physics-based information was included as a vertex- and edge encoding that marked the exposure field and the wafer borders, respectively, locations we want our model to fit differently. To validate the model's performance, a synthetic overlay dataset was constructed, emulating key properties of the real dataset and providing known ground truth overlay states. This synthetic dataset enabled evaluation of our model through ablation studies.

The model trained on the synthetic overlay dataset achieved a high average R^2 score of 0.976, closely approximating noise-free overlay measurements, despite being trained solely on noisy overlay measurements. Ablation studies indicated that the physics-based encodings slightly improved model performance. Additionally, using a graph that included longer range connections significantly increased the accuracy of the model while not increasing the number of parameters in the model. In one example the model with longer range connections outperformed its counterpart which used twice the number of message passing steps and thus also almost twice the number of parameters. Adding a random rotation data augmentation strategy enhanced the model's accuracy by preventing overfitting to the training set.

On the real overlay dataset, the model demonstrated a 30% lower mean squared error in predicting noise-free overlay measurements compared to the noisy inputs. On the synthetic dataset, this measure was 96%. This performance gap is likely due to the predictability of the noise-free overlay in the synthetic dataset, though some of the gap may be attributed to overfitting. The results suggest that despite not having access to clean overlay targets, the model can learn to denoise overlay measurements based solely on patterns recognized from the noisy overlay data and with minimal manual priors inserted into the model.

Overall, the thesis concludes that the machine learning model, trained using the Noise2Noise approach, provides an effective solution for denoising overlay measurements. "Future work should focus on refining the model and its training strategies to try to bridge the performance gap between synthetic and real datasets, and on exploring if integration of the denoising model into the overlay calibration pipeline can improve calibration stability in a significant manner.

Preface

This document marks the end of my life as a student as which I have learned a lot, have had amazing experiences, and where I grown up to to become the person I am now. In six years of studying applied mathematics I enjoyed learning abstract mathematical concepts but also finding their practical applications. While I spent many years in the books, I really enjoyed this last year of my masters where I was able to use my built up knowledge to dive into applications of state of the art A.I. models.

I could not be in the role I am now without the support I got, for whom I will now dedicate a small part of this thesis. First of all I would like to thank my colleagues as ASML who received me into the team with open arms and where always there when I had any questions. I would especially like to thank Lucas Bekker, who was my supervisor at ASML, and with whom I have had many insightful conversations and who greatly helped me with the planning for the thesis. I would also like the Alexander Heinlein, my university supervisor, who always made time in his very busy schedule to give great advice during our meetings. I really like the research area he is in and can't wait to see what the future brings for the topic. I would also like to thank my family and friend who where always there for me. I would especially like to thank my girlfriend who's support was extraordinary.

T.M. Kamminga
Delft, June 2023

Contents

1

Scanner overlay

As this master thesis covers a method for removing outliers in the overlay error measurements on ASML scanners, we will first elaborate on what these overlay error measurements are. Once a context of these overlay error measurements and the outliers that occur in them has been established, we will formulate the mathematical notation that will be used for these measurements in this report.

1.1. Introduction to scanner overlay

Semiconductor manufacturing

ASML is a manufacturer that specializes in the development of photolithography machines that are used to make computer chips. These machines are called TwinScans or, more generally, scanners. Photolithography is a manufacturing technique where a pattern from a blueprint, known as a mask, is projected on a light-sensitive coating called photoresist. The pattern on the exposed photoresist is then used to construct the components of the computer chip. Continued improvements to the lithography process have enabled semiconductor manufacturers to produce ever more complex computer chips with billions of transistors. Photolithography is an important step in a many-step process used to manufacture modern chips. Constructing a single chip layer on a silicon wafer involves a manufacturing procedure consisting of deposition, photoresist coating, exposure, developing, etching, implantation, and stripping. A short summary of these steps is given in Figure 1.1. ASML's machines are only involved in the exposure step.

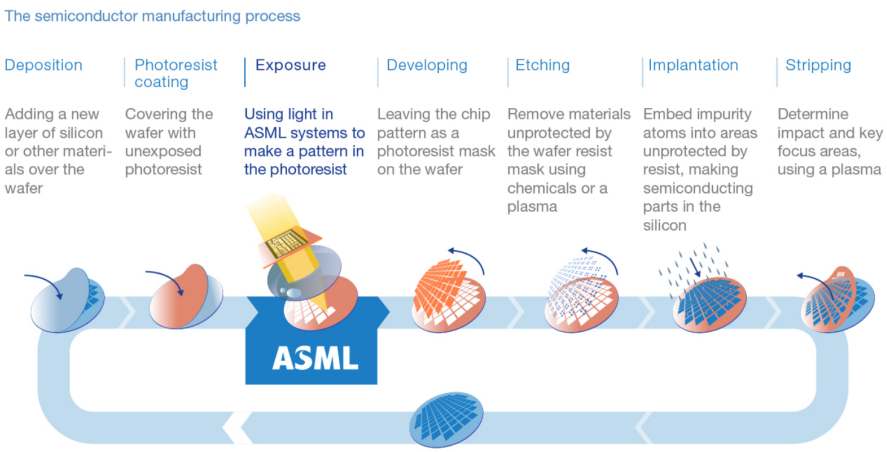


Figure 1.1: A summary of the semiconductor manufacturing process, taken from [1]. ASML develops the TwinScan machines that perform the exposure step.

Layer on layer

The manufacturing steps mentioned above are repeated tens of times so that the 2D patterns produced can be stacked to form complex 3D structures that make up the components and connections of a chip. Over time, the size of these structures has shrunk to the nanometer level. Many challenges have been overcome to enable the constant miniaturization, and many remain unsolved. One of those challenges is to stack these layers on top of each other accurately. After every exposure, the wafer is removed from the scanner, developed, and then exposed again. To ensure proper connections between consecutive layers, every part of the layer has to be placed in the exact right place, on top of the previous layer, with nanometer precision. In Figure 1.2, we can see the 2D metal layers of a chip stacked on top of each other; if these layers were not accurately placed at exactly the right locations, the missing connections would hamper the performance of the resulting chip or even make the chip non-functional.

If a TwinScan scanner had a consistent placement error, no problem would arise if the entire chip was made on this single scanner. Every layer would have the same deviation, meaning the layers stack properly and are connected in the right places. Chip manufacturers, however, use many different scanners to manufacture a single chip. This means we want to calibrate the scanners so that all parts of the pattern lay exactly in the correct place, independently of which scanner was used. The amount of deviation, in the xy-direction, of the pattern over the whole wafer is called machine-to-machine overlay error. Reducing the machine-to-machine overlay error is a continuing challenge.

Before a scanner is delivered to the customer, it has already undergone an extensive calibration procedure that reduces the overlay error to an agreed baseline. Over the time span of days or weeks, overlay performance drift drifts from this initial calibration [9, 43]. This drift can come from multiple sources, such as temperature fluctuations or degradation of the chuck on which the wafer is placed during exposure.

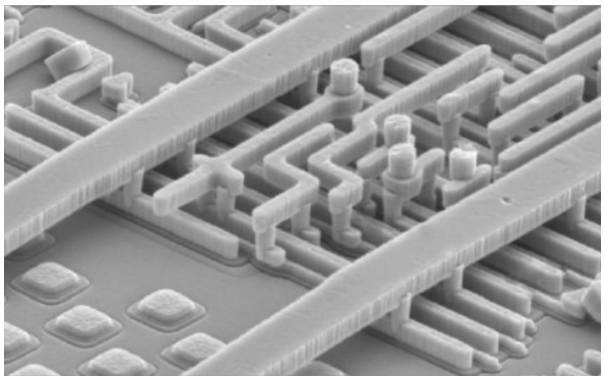


Figure 1.2: An image produced by a scanning electron microscope showing the internal structures of a computer chip. We can see that the structures are fabricated using stacked 2D layers. These layers must be properly placed, which is done, in part, by minimizing the overlay error. Taken from [25]

.

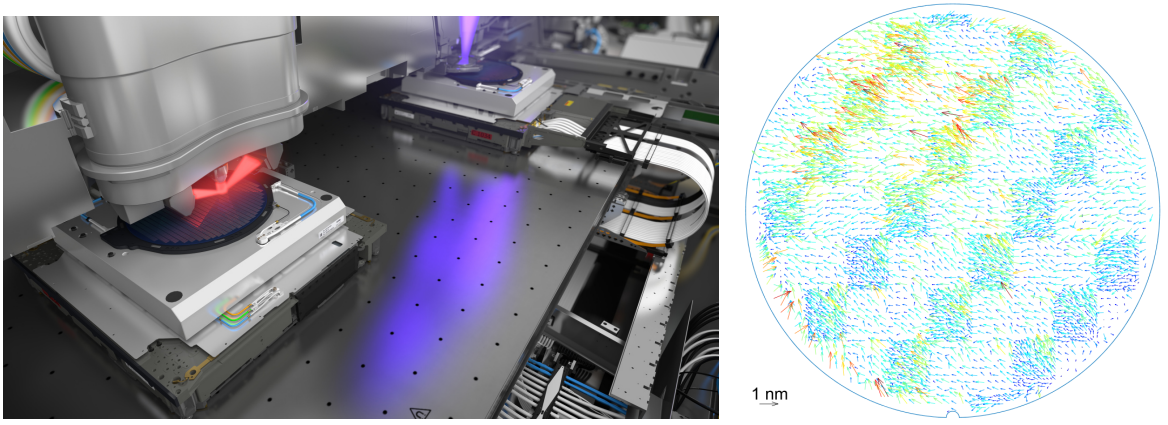
Measuring overlay

Since a low overlay error is important for producing functioning computer chips, reducing the overlay error is the next logical step. To do this, we must first know the current state of the overlay error; we can then use this state to tune how the pattern is projected on the wafer and end up with a lower overlay error. To measure the current overlay, monitor wafers with many markers printed on their surface are exposed every few days during production [9, 43]. The exact location of these markers has been accurately measured before the monitor exposure job and is thus known. On top of these markings, a new photoresist layer is added, which is then exposed by the scanner we want to calibrate. This new layer features similar markers placed on top of the old ones. Because of the scanner's overlay error, any pair of old and new markers will not be placed exactly on top of each other. The difference between the marker of which the location is known and the newly placed marker is measured, resulting in dx and dy values representing the local overlay shift compared to the baseline. We thus get such a dx and dy value for each measurement location that represents the deviation from the correct location.

Overlay can be measured in two ways. The first possibility is to expose the monitor wafer in the scanner and then also read out the resulting marks in the scanner. This read-out process is visible in

illustration figure 1.3a. The other possibility is to still expose the pattern in the scanner but then read out the overlay in a separate ASML product called the Yieldstar metrology system. Because the readout process is done in a separate machine, the scanner can continue production, reducing cost [9, 43]. This is the preferred process and is used for our overlay measurements.

Either procedure results in measurements in the form of a vector field with an xy-overlay vector for every measured marker. Figure 1.3b shows the structure of this output data. This illustration is synthetic overlay data created using the process described in section 4.3 and not an actual overlay measurement, as these measurements are sensitive customer data.



(a) A wafer being measured in the scanner [2].

(b) An illustration showing the structure of the overlay data.

Figure 1.3: Two images showing the overlay measurement process and generated output data, respectively. The output data is not an actual wafer measurement but is generated using the process described in section 4.3.

Outliers in overlay data

The overlay calibration process aims to reduce the scanner's overlay error. The input of the overlay calibration model is the overlay data generated by measuring the monitor wafers. This overlay data is generated by a process consisting of many steps that can potentially add inaccuracies to the data. These inaccuracies result in a measured overlay that does not purely correspond to a wrongly calibrated scanner but has other sources.

One reason for these inaccuracies could be a backside contamination, where a particle on the underside of the wafer deforms the wafer [6]. Another reason could be a particle on top of the marker, resulting in an inaccurate readout of the marker. A further reason for inaccurate readings could be slight temperature fluctuations in the scanner when a wafer is exposed. Because the measurements are done at a nanometer scale, every small disturbance can change the overlay measurements.

The measured overlay can thus be seen as a stochastic process giving information on the machine's true overlay. Because these stochastic overlay measurements are the input of the overlay calibration model, the resulting calibration is also stochastic. This means that the calibration can have unwanted variance. A couple of methods are used to reduce this variance.

One variance reduction method is to measure a batch of multiple wafers. This batch can then be used to average out the overlay and thus lower the calibration's variance. Exposing, developing, and measuring overlay on the monitor wafers is an expensive process; therefore, the batch size is preferably kept as small as possible. A second method to reduce the calibration variance is to remove outliers in the data. These outliers are overlay vectors on the wafer classified as errors stemming from the overlay measurement process and deemed not to correspond to the actual machine overlay calibration. This can be points influenced by contamination or other faults in the process. The current implementation of the overlay calibration model already has an outlier removal mechanism. The assessment was, however, made that this outlier removal algorithm could be improved using the vast amounts of data available from previous overlay measurements. This thesis's assignment is the challenge of improving the outlier removal model based on the available data.

1.2. Formalizing scanner overlay measurements

To discuss the overlay measurements mathematically, we will formalize the mathematical notation we use for the measurement data and for the measurement sampling process.

The overlay measurements

The overlay measurements are generated by reading out markers projected by the scanner on a monitor wafer; a silicon wafer which is preprocessed for these overlay measurements and is not directly used to manufacture semiconductors. The markers on the monitor wafers are projected on this wafer by repeating a reticle exposure. The reticle is an exchangeable plate that has the magnified pattern of the fields we want to project edged in to it's surface. The reticle absorbs part of the light during a projection such that the remaining light beam exposes the pattern on the photoresist coating on top of the wafer. Each full exposure of the reticle results in one exposed field on the wafer. These fields are repeated over the wafer, as indicated by the black rectangles in Figure 1.4a. To save measurement time, not every marker is read out, but some fields are read out densely and others are read out more sparsely. The exact pattern is classified, but Figure 1.4a shows the layout used for our synthetic dataset of section 4.3. To keep track of the measurement locations, we will use the tensor

$$\mathbf{u}_i = \begin{pmatrix} u_{i,1} \\ u_{i,2} \\ \vdots \\ u_{i,M_i} \end{pmatrix}, \quad u_{i,j} = (u_{i,j}^x, u_{i,j}^y), \tag{1.1}$$

where $\mathbf{u}_i \in \mathbb{R}^{M_i \times 2}$ and $u_{i,j} \in \mathbb{R}^2$. A single measurement location on the wafer $u_{i,j}$ is given by the the x and y coordinates $u_{i,j}^x$ and $u_{i,j}^y$, where the origin of the coordinate system is the center of the wafer. The subscript i denotes which monitor wafer batch the locations belong to and j to which marker on the wafer. To distinguish between the single locations $u_{i,j}$ and the collection of all locations $\mathbf{u}_i$, the latter will be written in bold. Since for each monitor wafer measured some of the measurements of the makers can fail, resulting in no measured overlay value at that measurement point, the number of successful measurements M_i and thus the locations of all successful measurements in $\mathbf{u}_i$ can vary with i. Since the wafer has a radius of 0.15 meters and $u_{i,j}$ is measured relative to the center, for all i and j, we have $||u_{i,j}||^2 \leq 0.15m$.

For all the overlay errors at the marker locations $u_{i,j}$, on the ith wafer, we get overlay value in the x and y direction, which will denote as

$$\mathbf{x}_i = \begin{pmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,M_i} \end{pmatrix}, \quad x_{i,j} = (x_{i,j}^{dx}, x_{i,j}^{dy}), \tag{1.2}$$

where just as with the locations $\mathbf{x}_i \in \mathbb{R}^{M_i \times 2}$ and $x_{i,j} \in \mathbb{R}^2$. The overlay errors $x_{i,j}^{dx}, x_{i,j}^{dy}$ are measured in nanometers and of the order $1 \times 10^{-9}m$, significantly smaller than the values of $\mathbf{u}_i$. The most common way of visualizing the measured overlay is as a vector plot, as shown in Figure 1.4b. Here, the vectors $x_{i,j}$ have been colored according to their length, and the size of the vectors has been greatly magnified to make overlay error visible. Each vector in the vector plot represents the shift in location of the projected compared to the correct location. This figure is a synthetic overlay measurement made according to the method of section 4.3.

When comparing two overlay measurements in $\mathbb{R}^{M_i \times 2}$ we will mostly use the L^2 loss. We define this L^2 loss between two overlay measurements $\mathbf{x}_i$ and $\mathbf{y}_i$ as

$$L^2(\mathbf{x}_i, \mathbf{y}_i) := \left\| \begin{pmatrix} x_{i,1}^{dx} \\ x_{i,1}^{dy} \\ \vdots \\ x_{i,M_i}^{dx} \\ x_{i,M_i}^{dy} \end{pmatrix} - \begin{pmatrix} y_{i,1}^{dx} \\ y_{i,1}^{dy} \\ \vdots \\ y_{i,M_i}^{dx} \\ y_{i,M_i}^{dy} \end{pmatrix} \right\|^2. \tag{1.3}$$

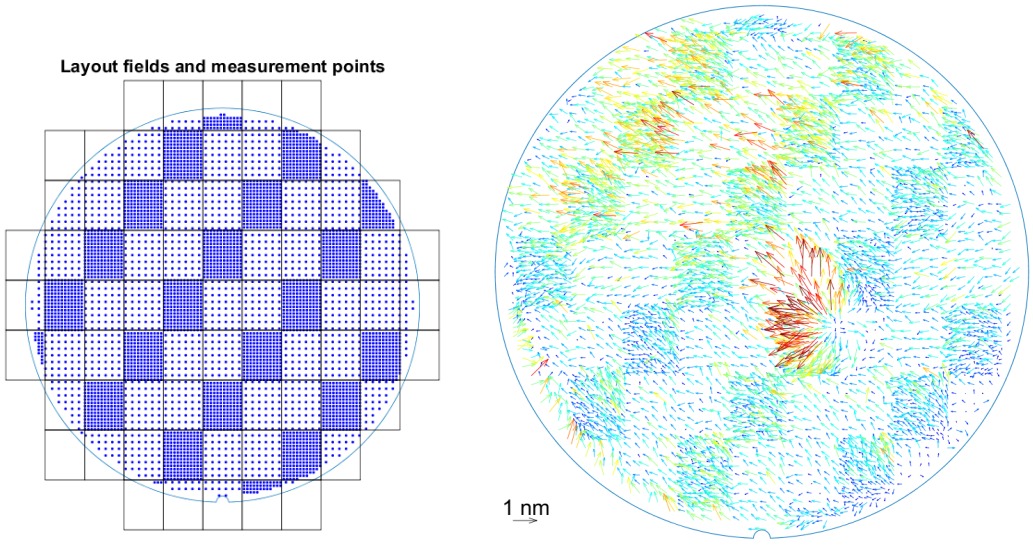


Figure 1.4: The image (a) shows a measurement layout on the wafer where the black rectangles represent the fields projected and the blue points the measurement points represent the places $\mathbf{u}_i$ where the markers have been read out for the overlay error values. Image (b) shows an example of the resulting overlay errors $\mathbf{x}_i$ at the locations $\mathbf{u}_i$ represented by a vector plot. The layout and values are from the synthetic dataset as described in section 4.3.

Sampling the overlay measurements

The drift in the overlay is sampled with a frequency in the order of every few days, with the exact frequency depending on the customer's preference. Since errors, such as contaminations or faults in the wafer development, can occur during the overlay measurement process, most measurements are done using four monitor wafers. There are two different chucks on which the four monitor wafers are developed; two wafers for each chuck are thus exposed under nearly the same circumstances in just a few minutes. Because this time span is significantly shorter than the days during which the overlay error drift occurs, we will assume that these two overlay measurements are sampled from the same distribution of possible overlay measurements that can occur at that moment in time with that scanner on that specific chuck.

If resources were unconstrained, we could measure the overlay error more accurately by taking an average $\overline{\mathbf{x}_i}$ of many overlay measurements representing the current machine state. We expect this average to converge to the true noise-free overlay, which we will denote as $\mathbf{y}_i$. From now on, we will assume that the distribution from which we sample the overlay measurements $\hat{\mathbf{x}}_i$ and $\tilde{\mathbf{x}}_i$ is conditional on this true noise-free machine overlay state $\mathbf{y}_i$. First, $\mathbf{y}_i$ is sampled from all the possible machine overlay states, and then the two overlay measurements measured on the same chuck are sampled conditional on $\mathbf{y}_i$. For these two noisy overlay measurements we will use the notation $\hat{\mathbf{x}}_i$ and $\tilde{\mathbf{x}}_i$. The sampling process is then given as

$$\mathbf{y}_i \sim p(\mathbf{y}), \quad \hat{\mathbf{x}}_i, \tilde{\mathbf{x}}_i \sim p(\mathbf{x}|\mathbf{y}_i). \tag{1.4}$$

Here $\hat{\mathbf{x}}_i$ and $\tilde{\mathbf{x}}_i$ are assumed to be independent conditional on $\mathbf{y}_i$ and identically distributed because we assume no drift in the scanner's overlay error in the short time frame of the two measurements. We assume the expectation of the noisy overlay measurements is equal to the ground truth overlay $\mathbf{y}_i$, that is $\mathbb{E}\,\hat{\mathbf{x}}_i|\mathbf{y}_i = \mathbb{E}\,\tilde{\mathbf{x}}_i|\mathbf{y}_i = \mathbf{y}_i$.

The i in the subscript of the variables indicates that the pair $\hat{\mathbf{x}}_i$ and $\tilde{\mathbf{x}}_i$ are sampled from the same scanner state at the locations $\mathbf{u}_i$. The full dataset will thus be composed of the triples $(\hat{\mathbf{x}}_i, \tilde{\mathbf{x}}_i, \mathbf{u}_i)$ with $1 \leq i \leq N$. For the synthetic dataset, the true noise-free overlay $\mathbf{y}_i$ is also available, so we end up with the dataset consisting of the quadruples $(\hat{\mathbf{x}}_i, \tilde{\mathbf{x}}_i, \mathbf{u}_i, \mathbf{y}_i)$ with $1 \leq i \leq N$. For the synthetic dataset, $N = 4000$; for the real data, N is considerably larger but classified.

2

Geometric deep learning

In the next chapters, we will use various deep-learning techniques and neural network architectures. This chapter will give an introduction to these methods. It is based on the book "Geometric Deep Learning Grids, Groups, Graphs, Geodesics, and Gauges" [8] that tries to unify many deep-learning techniques in a foundational geometric way. We will focus specifically on convolutional neural networks and graph neural networks as they are of interest to the rest of this report.

2.1. The objective

We will first focus on supervised machine learning. In supervised machine learning, we are given the input variables x_i and want to use these variables to predict the target variables y_i. We assume these variables to come as N i.i.d. samples $\mathcal{D} = (x_i, y_i)_{i=1}^N$ from the underlying data distribution. To predict the target variables y_i using the input data x_i, we choose a function f_θ from a parameterized function class $\mathcal{F} = \{f_{\theta \in \Theta}\}$. A common class of such functions are neural networks, where $\theta \in \Theta$ represents the chosen network weights. To measure how well a function from the parametric family predicts the target variables, we define the expected loss

$$\mathcal{R}(f_\theta) := \mathbb{E}_{x,y} L(f_\theta(x), y), \tag{2.1}$$

which is based on a loss function L that quantifies how close $f_\theta(x)$ is to y. Common loss functions are the $L_1(y, y') := |y - y'|$ and the $L_2(y, y') := (y - y')^2$ loss. Because in most learning problems, the distribution of x and y is unknown and we have only a finite set of samples from this distribution, we estimate the expected loss with the empirical loss

$$\mathcal{R}_{emp}(f_\theta) := \frac{1}{N} \sum_{i=1}^{N} L(f_\theta(x_i), y_i), \tag{2.2}$$

which we can minimize with respect to θ. Finding a function f_θ that minimizes the empirical loss and does not overfit the input data given, meaning that it also performs well on unseen data, is one of the main challenges in the machine learning field. If the model f_θ gives a low empirical loss on data it was fitted on but a high empirical loss on unseen, the model is over-fitted. It is common to split the dataset into a training and validation set to test if this is the case. The model is fitted using the training set, and the performance of this fitted model is then measured using the unseen validation set.

2.2. Neural networks

With the advent of large high-quality datasets, increased computing hardware performance, and the availability of proper software, it has been possible to construct and fit ever more complex functions f_θ, with thousands to billions of parameters, that can learn complicated patterns and make accurate predictions. A popular class of such functions are artificial neural networks. These neural networks are made up of linear and nonlinear maps, defined by matrix multiplication with weights matrices W and

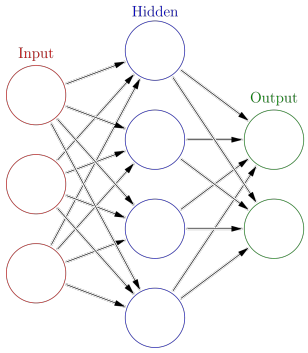


Figure 2.1: A graphical depiction of a relatively shallow three-layer multilayer perceptron (MLP) with an input layer, a single hidden layer, and an output layer. Every node in a layer is "connected" to every other node in the next layer by a weight from the weight matrix W [18]. This MLP has an input dimension of 3, a single hidden dimension of 4, and an output dimension of 2. Following the notation of equation 2.3 this simple MLP is equal to $f^{\text{connected}}_{\theta=\{W^{\text{out}},b^{\text{out}}\}} \circ f^{\text{connected}}_{\theta=\{W^{\text{in}},b^{\text{in}}\}}(x)$, with $x \in \mathbb{R}^3, W^{\text{in}} \in \mathbb{R}^{3\times4}, b^{\text{in}} \in \mathbb{R}^4, W^{\text{out}} \in \mathbb{R}^{4\times2}$, and $b^{\text{out}} \in \mathbb{R}^2$.

bias vectors b, where a non-linear activation function $a(\cdot)$ is applied to each element of the resulting vector. This construction allows the composed function to make non-linear predictions and gives a large parameter space that can be tuned. A fully connected neural network layer can be written as

$$f^{\text{connected}}_{\theta=\{W,b\}}(x) = a(W^T x + b). \tag{2.3}$$

The fully connected layer has input size n and output size m for the corresponding in- and output vectors. To conform to these size we choose $b \in \mathbb{R}^m$ and $W \in \mathbb{R}^{n\times m}$. A common pick for the non-linear activation functions are the $\tanh(\cdot)$ function, the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, or the ReLu function $ReLu(X) = \max(0, x)$. The main criteria for an activation function is that it is differentiable almost everywhere, such that, using backpropagation [28], the gradient of the empirical loss with respect to the parameters of the network can be found analytically. This gradient is then used to minimize the empirical loss and thus fit the network to the data.

2.3. Multilayer perceptrons

A common way of building larger neural network-based functions is to stack multiple fully connected neural network layers to form a multilayer perception (MLP). MLPs are made up of three or more layers. An input layer, one or more hidden layers, and an output layer. A simple illustration of such an MLP is shown in Figure 2.1, where we can see that because of the matrix multiplication $W^T x$ every node from a layer is "connected" with every node in the next layer by a single weight from the weight matrix W. We define an MLP with input dimension n^{in}, k hidden layers with dimension n^{hidden}, and output dimension n^{out} using the notation of equation 2.3 as

$$f^{\text{MLP}}_\theta(x) = f^{\text{connected}}_{\theta=\{W^{\text{out}},b^{\text{out}}\}} \circ \underbrace{f^{\text{connected}}_{\theta=\{W^{\text{hidden}}_{k-1},b^{\text{hidden}}_{k-1}\}} \circ \ldots \circ f^{\text{connected}}_{\theta=\{W^{\text{hidden}}_1,b^{\text{hidden}}_1\}}}_{k-1\times} \circ f^{\text{connected}}_{\theta=\{W^{\text{in}},b^{\text{in}}\}}(x), \tag{2.4}$$

with $W^{\text{in}} \in \mathbb{R}^{n^{\text{in}}\times n^{\text{hidden}}}, b^{\text{in}} \in \mathbb{R}^{n^{\text{in}}}, W^{\text{hidden}}_1, \ldots, W^{\text{hidden}}_{k-1} \in \mathbb{R}^{n^{\text{hidden}}\times n^{\text{hidden}}}, b^{\text{hidden}}_1, \ldots, b^{\text{hidden}}_{k-1} \in \mathbb{R}^{n^{\text{hidden}}}, W^{\text{out}} \in \mathbb{R}^{n^{\text{hidden}}\times n^{\text{out}}}$, and $b^{\text{out}} \in \mathbb{R}^{n^{\text{out}}}$. Because $f^{\text{MLP}}_\theta(x)$ is a composition of almost everywhere differentiable functions, we can calculate the gradients of $f^{\text{MLP}}_\theta(x)$ almost everywhere with the chain rule, in turn, used for the loss minimization method.

2.4. Stochastic gradient descent

Our objective is to fit out model's parameters to our data by minimizing the empirical loss. Many possible first-order gradient optimization methods to achieve this task exist. The simplest one used in deep learning is stochastic gradient descent (SGD). We start by picking a learning rate $\eta \in \mathbb{R}_{>0}$ and the initial values for our parameters θ. We can then write the empirical loss as a function of the parameters

θ, so

$$Q(\theta) := \frac{1}{N}\sum_{i=1}^{N} Q_i(\theta) = \mathcal{R}_{emp}(f_\theta), \quad \text{with} \quad Q_i(\theta) := L(f_\theta(x_i), y_i). \tag{2.5}$$

In the standard, and not stochastic, gradient descent method, we minimize $Q(\theta)$ by updating the parameters θ to their new values θ' with the step

$$\theta' = \theta - \eta \nabla Q(\theta) = \theta - \frac{\eta}{N}\sum_{i=1}^{N} \nabla Q_i(\theta). \tag{2.6}$$

Since when using neural networks with many parameters and large datasets, the calculation of $\nabla Q(\theta)$ is computationally and memory-wise very expensive operation, it is common to approximate the gradient on the full dataset $\nabla Q(\theta)$ by the gradient on a single sample $\nabla Q_i(\theta)$. The iteration step then becomes

$$\theta' = \theta - \eta \nabla Q_i(\theta). \tag{2.7}$$

This greatly reduces the computation and memory resources needed for a single iteration, allowing for minimizing the loss for datasets that do not fit into memory. So this means there is no limit to the dataset size we can fit our function f_θ on. The ordering of the samples for which we calculate $\nabla Q_i(\theta)$, the gradient of the loss on a single data point x_i and target y_i, is chosen randomly for each sweep through the data, also called an epoch. This means that during a single epoch, $\nabla Q_i(\theta)$ is calculated once for every data pair x_i and y_i in the dataset for the value of θ at that iteration step, but the ordering of i is random. This random order gives the algorithm its name: stochastic gradient descent.

The gradient of the loss on a single data sample with respect to the parameters $\nabla Q_i(\theta)$, can however be a high variance estimator for $\nabla Q(\theta)$, leading to worse or no convergence. To remedy this, it is common to use mini-batches. We again randomly order $\{1, 2, \dots, N\}$, but now partition the results into (almost) equally sized subsets $S_1, \dots, S_K$ representing our data batches. The lower variance estimator estimator formed by the average gradients of these data batches is then used in the iteration step. A single iteration is then be written as

$$\theta' = \theta - \frac{\eta}{|S_k|}\sum_{i \in S_k} \nabla Q_i(\theta). \tag{2.8}$$

While the expectation of the outcome of stochastic gradient descent optimization, with the right learning rate η, can be proven to converge to the global minimum for convex functions [20], this is not the case for our non-convex problem of empirical loss minimization. The method can get stuck in a local minimum. Experience has however shown that convergence to a sufficient parameter set θ which gives a loss close to the minimum of $Q(\cdot)$ is often reached. Reaching the exact minimum of the empirical loss on the training set is often even undesirable because overfitting can lead to poor performance on the validation set.

2.5. The curse of dimensionality

Designing a neural network model to learn a particular task efficiently and effectively is a hard task. Using MLPs on large inputs such as images can quickly result in a large parameter count for the model. If there are a relatively high number of parameters in the model compared to the size of the dataset, overfitting can become a significant problem. Efficient neural network models often utilize priors on the input data in the model. These priors then decrease the information the model has to learn from the data. Many strategies exist to use the input data structure to reduce the needed model size, and the authors of [8] split them into two fundamental principles: symmetry and scale separation.

Starting with symmetry, if, for example, we have as input a picture of a dog, we can create hundreds of versions of this picture by slightly translating this picture in all directions. If our model is translation-equivariant, meaning that a translation of the input results in an output with the same translation, we do not have to learn from all these examples but can use just one. Thus, we see in this example that we can use the structure or symmetry in the input data, adapt the model accordingly, and reduce the amount of data needed to effectively learn a task, thereby reducing the parameter count needed to learn this task.

The second fundamental principle proposed by the authors of [8] is to have scale separation in the model. The authors use the example of a Fourier transform which can separate different frequency signals from a measurement. A signal in the input important to the model's prediction, could occur at various frequencies. Our model should be able to interpret effectively signals at these different frequencies. If we relate this to a model classifying an image as either of a cat or a dog, we should use the high-frequency structure of the fur and the low-frequency shape of the animal to accurately determine if the picture is of a cat or a dog. The usual way to incorporate scale separation in a neural network-based model is to have separate parts in the model that interpret different frequencies of signals, which are then combined to form the output of the model.

2.6. Convolutional neural networks

Many fields in machine learning use input data with a lattice data structure which can be indexed by integer vectors. An example of such data with such a lattice data structure is images. Convolutional neural networks (CNNs) have excelled at many tasks for this data structure type. The success of these models can be explained by the model criteria stated in the previous section.

Convolutional neural networks are based on the multidimensional convolution operation. This convolution between two M dimensional functions x and h taking values on a discrete lattice, a repeating arrangement of points, produces another function also defined on this M dimensional discrete lattice. This operation is written as

$$f(n_1, n_2, ..., n_M) = x(n_1, n_2, ..., n_M) \overset{M}{*} h(n_1, n_2, ..., n_M), \tag{2.9}$$

with $n_1, n_2, ..., n_M \in \mathbb{Z}$ representing the discrete values of x, h and f. The operation is defined as

$$f(n_1, n_2, ..., n_M) = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} ... \sum_{k_M=-\infty}^{\infty} h(k_1, k_2, ..., k_M) x(n_1 - k_1, n_2 - k_2, ..., n_M - k_M). \tag{2.10}$$

The multidimensional convolution operation can be seen as multiplying the signal of x with a filter h. To limit the size of this filter, it is common to only define the values of h for an equally sized M-dimensional array and set all values outside of this M-dimensional array to 0. The values in this M-dimensional array $C_\theta \in \mathbb{R}^{2K+1 \times \cdots \times 2K+1}$ are the parameter set for the convolution operation

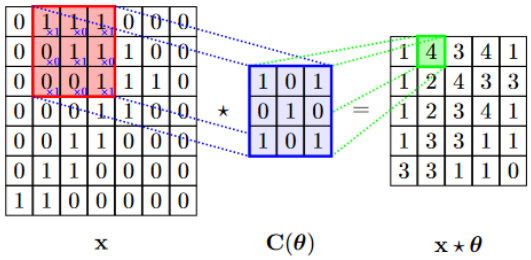
$$f^{\text{conv}}_{\theta=C_\theta}(x(n_1, ..., n_M)) = a\left(\sum_{k_1=-K}^{K} ... \sum_{k_M=-K}^{K} C_\theta(k_1, ..., k_M) x(n_1 - k_1, ..., n_M - k_M) \right), \tag{2.11}$$

with $K \in \mathbb{N}_0$ and some activation function $a(\cdot)$ that is differentiable almost everywhere. If we take M equal to 2, K equal to 1, and use the identity function for $a(\cdot)$, we get the operation illustrated in Figure 2.2a. Implementations of CNNs also often use "channels," representing multiple arrays with the same dimensions, such as for different color values in image data. This more general convolution operation then allows for different numbers of input and output channels. For the sake of brevity, we will ignore these channels in the CNN operation and use 1 input- and output channel as in equation 2.11.

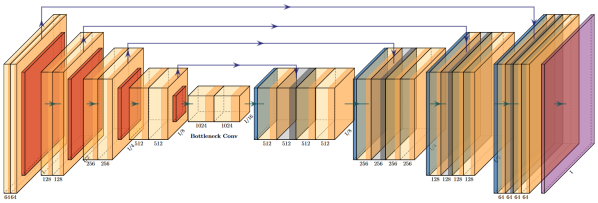
The multidimensional convolution operation has the same output dimension as the input dimension, and by choosing K small, the operation processes only local information. The operation has considerably fewer parameters than a fully connected layer with the same input and output sizes would have because it shares the parameters of C_θ over the whole input. The output on the convolution operation is translation equivariant as any translation to input results in an output equal to the equally translated output made with the untranslated input. These priors in CNN models use the data structure of the input data to reduce the necessary amount of parameters needed to learn a task. Using this convolutional layer, we no longer have every node in the input connected to the output with a single weight; instead, in a single CNN layer there are only connections to local data defined by the convolution filter; however, by stacking multiple convolutional layers, information can still travel further distances over the input domain.

The U-Net architecture [41] is an example of using scale separation in a CNN model. The very successful and popular architecture uses multiple convolutional layers together with up- and downsampling layers to operate at multiple resolution levels, as seen in Figure 2.2b. The coarser grids can capture the low frequency signals and the finer grids the higher frequency signals.

State-of-the-art CNN-based architectures, such as U-NET, use additional techniques, including skip connections [45] and batch normalization [23] to achieve state-of-the-art results. These CNN-based architectures show how the principles of data symmetry and scale separation from the previous section can be used to build models that make prior assumptions on the input data structure to improve model performance.



(a) The convolution of the 3×3 filter $C(\theta)$ on the input x resulting in the output $x \star \theta$ [8].



(b) An illustration of the U-Net architecture which is a state-of-the-art CNN architecture that uses multiple convolutional layers together with scale separation [8].

Figure 2.2: The convolution operation on the left and the popular CNN-based architecture U-Net [41] on the right.

2.7. Graph neural networks

While CNN-based structures have been shown to incorporate effective priors for data sampled from an n dimensional lattice, not every data format naturally translates well to such a lattice. This observation has led to the increasing popularity of graph neural networks (GNNs), which perform their calculations on graphs instead of on a lattice. The same lessons learned from CNNs, such as the benefits of spatial equivariance and scale separation, can also be used for GNNs. Many types of graph neural networks exist, but the most general [8] and the one used in the upcoming chapters is the message-passing neural network [5]. Such a message-passing neural network will receive as the input data a graph and a set of feature vectors. The edges of the graph indicate some relation between the vertices. The feature vectors represent information on the edge or vertex of the graph. These feature vectors can be linked to either a unique vertex of the graph or a unique edge of the graph. Our message-passing neural network does not alter the graph but incrementally updates the feature vectors on the vertices and edges.

The input graph and feature vectors

Each data sample is represented by an undirected graph $G_i = (V_i, E_i)$ composed of a set of vertices V_i and a set of edges E_i containing unordered pairs of vertices $\{u, v\}$ with $u, v \in V_i$. Two vertices $u, v \in V_i$ are neighbors if $\{u, v\} \in E_i$. We then define the neighborhood of the vertex v as the set of all neighbors of v i.e. $\mathcal{N}_v = \{u \mid \{u, v\} \in E_i\}$. For each vertex $u \in V$, we choose a corresponding vertex feature vector $x_u \in \mathbb{R}^{n_{\text{vertex}}}$. To include the edge features, for every ordered edge (u, v) which has $\{u, v\} \in E$, we choose an edge feature vector $e_{u,v} \in \mathbb{R}^{n_{\text{edge}}}$. The vertex feature vectors should include information about the vertex it is linked to, and the edge vectors should include information about the relation between the two vertices its edge connects.

In our definition of message-passing in the report given by the algorithm 1, we thus have an undirected graph G_i, but directed edge features $e_{u,v}$. It is also possible to define this algorithm for undirected edges features by taking $e_{u,v}$=$e_{v,u}$ for all $\{u, v\} \in E_i$ or choose a directed graph G_i and use the directed in-neighborhood $\mathcal{N}_v^{\text{in}} = \{u \mid (u, v) \in E_i\}$ for the message-passing step. These alternative definitions will, however, not align with our eventual model.

The message-passing step

Now that we have defined the input graph and the node- and edge feature vectors, we can define the message-passing step. There are multiple ways to do this message-passing step with edge features. The implementation we will use will update both the vector feature vectors x_v and the edge feature vectors $e_{u,v}$ as described in [5]. Like with CNNs, the message-passing step uses local information to update the input data while keeping the same data format. Where CNNs use a convolutional filter to gather this local information, the message-passing step uses the neighborhood of a node to gather local

information. Again, just like with CNNs, the parameters are stored in an operation applied repeatedly over the whole input. This form of parameter sharing reduces the need for a higher parameter count.

The pseudo-code for the message-passing step with both vertex- and edge feature updates can be seen in algorithm 1. First, the edge feature vectors $e_{u,v}$ are updated using the learnable function $\psi_\theta(\cdot)$ with as input the vertex feature vectors x_u and x_v of the vertices the edge connects, together with the old edge feature vector $e_{u,v}$. Following this update of edge feature vectors, the vertex feature vectors are updated. This is done with the learnable function $\phi_\theta(\cdot)$, taking the input arguments of the previous vertex feature vector x_v and the output of a permutation invariant function $\oplus$. This permutation invariant function aggregates all the updated edge features $e_{u,v}$ that point to the vertex u. The function $\oplus$ needs to be permutation invariant as there should be no ordering of the neighbors. The most common options for the permutation invariant function are the sum, mean, or max functions. The learnable functions $\psi_\theta(\cdot)$ and $\phi_\theta(\cdot)$ are often implemented by concatenating all the input vectors and then applying the multi-layer perceptron of equation 2.4. This then allows you to choose the output dimension of both the vertex- and node feature vectors freely.

Algorithm 1: The message-passing step, with both node- and edge feature vector updates.

$G_i = (V_i, E_i)$

for $\{u, v\} \in E_i$ **do**
  $e'_{u,v} \leftarrow \psi_\theta(x_u, x_v, e_{u,v})$
  $e'_{v,u} \leftarrow \psi_\theta(x_v, x_u, e_{v,u})$
end
for $v \in V_i$ **do**
  $x'_v \leftarrow \phi_\theta\left(x_v, \oplus_{u \in \mathcal{N}_v} e'_{u,v}\right)$
end

The spread of information

In Figure 2.3, we see an illustration of the message-passing step. The feature vector x_0 is updated by first calculating the updated edge feature vectors for all the neighbors seen in green and then combining these values to update the feature vector x_0 itself. In a single message-passing step, the feature vector x_v is updated with information of the neighboring vertex feature vectors $\{x_u | u \in \mathcal{N}_v\}$. A network architecture that utilizes the message-passing step is commonly made up of multiple consecutive message-passing steps, where every step has its parameters for the learnable functions $\psi_\theta(\cdot)$ and $\phi_\theta(\cdot)$. If we want every vertex feature vector to have the opportunity to receive information from every other vertex feature vector, we would thus need the number of message-passing steps to be equal to or bigger than the graph's diameter. This diameter of the graph is defined as the maximum distance between two nodes in the graph $\max_{u,v \in V} d(u, v)$. This distance $d(u, v)$, in turn, is defined as the number of edges in the shortest path from u to v. The number of message-passing steps influences both how far information can travel over the graph and the number of parameters in the model as every message-passing step has a set of parameters for the learnable functions $\psi_\theta(\cdot)$ and $\phi_\theta(\cdot)$.

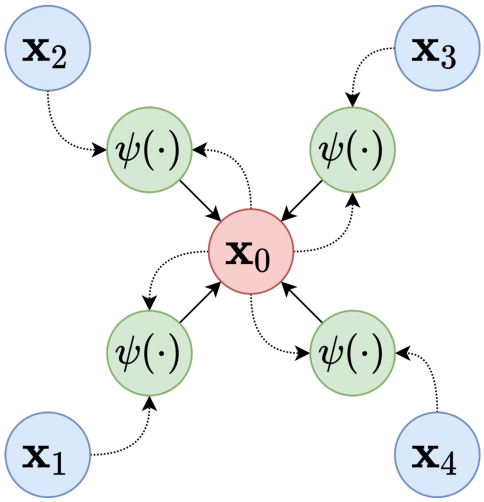


Figure 2.3: An illustration of the message-passing step of algorithm 1, where the feature vector x_0, shown in red, is updated using the updated edge feature vectors of all edges connecting x_0 to its neighbors, shown in green. These edge feature vectors are updated using the learnable function $\psi_\theta(\cdot)$ and its outputs are combined by the premutation invariant function $\oplus$ to a single vector. This single vector and the old feature vector of x_0 are the inputs for the learnable function $\phi_\theta(\cdot)$. The output of this function gives the new value of the feature vector x_0. This illustration was sourced from [13].

3

Related work

To better understand the work related to our objective of developing a machine learning-based method for removing outliers in overlay data, we summarize work related to our topic. We have split the work we found into three sections. The first section is on works that use overlay data. The second section looks broader into machine learning methods to remove outliers, which is further split up into a review of outlier classification methods and methods for denoising data. Here, we find a training method we would like to use. Since the reference neural network could not be directly used, we looked into neural network architectures compatible with our overlay data structure in the third section. We summarize the results of the related work chapter and how this has influenced our model design. In the last section of this chapter, we split up the central objective of this thesis, into several sub-questions.

3.1. Literature related to machine learning methods for overlay outliers

There are a limited number of works available that use overlay in a machine learning or outlier context. The most relevant paper to our problem of outlier removal in overlay data is [40], which discusses statistical methods for outlier removal. The authors highlight the need for better techniques due to the increased sensitivity of higher-order overlay modeling to large outliers. They compare the accuracy of two common outlier removal methods used for overlay data and their own robust regression model. All methods involve removing outliers based on residuals after fitting a linear model. Their method shows increasing accuracy but does not leverage the large data sets available, which we envision could be improved with machine learning methods. Additionally, we found [42], which explores a deep learning method for predicting overlay using dense measurements. The authors enhance their model by incorporating leveling measurements as inputs to their neural network, which provide a height map of the wafer, increasing the R^2 from the original values of 0.72 to 0.81.

3.2. Machine learning methods for outlier removal

Anomaly detection

For our study into related work for machine learning-based outlier removal models, we will use the systematic literature review ”Machine Learning for Anomaly Detection: A Systematic Review” [36], which reviews papers covering the detection of anomalies in datasets. We will use the term outliers interchangeably with anomalies in a dataset. In this overview, the authors define an anomaly as ”the problem of finding patterns in data that do not conform to expected behavior” [11]. The review authors use a further split of these anomalies into three classes [11]: point anomalies, contextual anomalies, and collective anomalies. Point anomalies are data points that are anomalies if you compare them to the full dataset, contextual anomalies are anomalies if you compare them to other points that are close in some metric, and collective anomalies are sets of multiple associated data instances that are anomalies compared to the full dataset. Our overlay dataset can have all three types of outliers. If a single marker is contaminated, the single overlay value could differ significantly from its neighborhood and form a point anomaly. Overlay values could also be within the normal overlay range but constitute an anomaly

in the context of neighboring overlay values. Suppose the backside of the wafer is contaminated with a particle. In that case, an area may be misprinted, leading to overlay values deviating in multiple measurement points from the true machine overlay, forming a collective anomaly.

Where statistical anomaly detection techniques build a statistical model for ordinary behavior of the dataset and judge anomalies accordingly, the review's authors describe machine learning techniques for anomaly detection as the effort to "automate the process of knowledge acquisition from examples" [7]. The authors of the review analyzed 290 papers discussing machine learning techniques for anomaly detection and label them with the three categories [16, 11]; supervised anomaly detection, semi-supervised anomaly detection and unsupervised anomaly detection. Here, the category supervised anomaly detection is used when the outliers get labeled, semi-supervised anomaly detection when the non-outliers get labeled (s.t., there is, in turn, only a single label for outliers), and the category unsupervised anomaly detection is used when no labeling is used in the training set. We will only consider unsupervised techniques for our outlier dataset as there is currently no labeling of the overlay dataset that is more accurate than the current outlier detection technique. Manual labeling could, in turn, introduce a bias stemming from the labeling process. The review's authors further categorize all the 290 papers into what machine learning model was used, and we can conclude that a wide range of models apply to anomaly detection. Popular classification methods include support vector machines, Bayesian networks, various types of neural networks, and K-nearest neighbors. Other popular machine learning methods used are different types of clustering or different regression types. Most of these methods are then verified by calculating scores such as true positive rate, true negative rates, and/or accuracy on a labeled validation set. For our method to calculate such scores, we would thus need to label a validation set for outliers with the possibility of adding bias based on this labeling.

Image denoising

In a way, the overlay data is similar to image data, where every wafer measurement represents an image, and every measurement location represents a pixel with a color value determined by the overlay value. There are some differences, such as the measurement locations having a sparse layout, there being only two values for each measurement location, and the values not being limited in size. However, machine learning methods for outliers in image data could still be interesting because of the similarities to our overlay data.

In the image domain, the term outlier or anomaly is most of the time not used; instead, classically, the assumption was that there is a clean image y sampled from the distribution of clean images $p(y)$, to which independent identically distributed (I.I.D.) Gaussian noise is added [14], to form the noisy image $x = y + z, z \sim N(0, \sigma I)$. The success of a denoising function $f(\cdot)$ is then measured by how low the mean squared error (MSE) for the method is, defined as $\mathbb{E}(||f(x) - y||_2^2)$. While it is easy to filter the noise and form a smoother image, it is hard to filter out the noise z and retain the detail from the clean image y. Because the distribution of the noise $x|y$ is known, most classical denoising methods try to approximate the distribution of clean images $p(y)$ to form, using the Bayes rule, the distribution of a clean image y given a noisy image x [35], so

$$p(y|x) \sim p(x|y) \cdot p(y). \tag{3.1}$$

Once the approximation for the likelihood of $p(y|x)$ of clean images conditional on the noisy images is constructed, we can approximate the clean image y using maximum likelihood estimation. The most difficult part of this classic Bayesian method is to find an approximation for the prior $p(y)$, the distribution of clean images. Over time, these approximations have used various techniques [14], such as energy regularization, linear approximation techniques such as principal component analysis, and nonlinear approximation, such as ones using wavelets. The increasing complexity of these approximations, together with other non-Bayesian new classical methods, led to better but diminishing results, which, in 2009, resulted in the question, "Is Denoising Dead?" [12] being posed.

This question ended up being answered with a resounding no as the rise of deep learning methods, where a highly over-parameterized function $f_\theta(\cdot)$ is used to achieve a low empirical loss on a (very) large dataset, allowed for a new solution for the image denoising problem. Instead of approximations for $p(y)$ by which we try to minimize using the Bayesian method the mean squared error, we minimize

the empirical loss on the training set

$$\mathcal{R}_{emp}(f_\theta) := \frac{1}{N}\sum_{i=1}^{N} L(f_\theta(x_i), y_i) \tag{3.2}$$

as before given in equation 2.2 for the loss of a supervised learning task. Minimizing this loss can then be achieved by tuning the parameter set θ of a parameterized function f_θ using optimization methods like the batched stochastic gradient descent algorithm of equation 2.8. If for the loss function L we chose the L^2 loss, we are directly minimizing an estimator for the mean squared error $\mathbb{E}(||f_\theta(x) - y||_2^2)$. We call this method supervised learned denoising as we use, in the training set, pairs of noisy image x_i and target clean images y_i. Because the success of this method depends in large part on the effectiveness of the parameterized function f_θ, the development of ever better-performing function families f_θ, like CNN-based models, has led to increased performance of supervised learned denoising methods, where they now significantly outperform any classic denoising method for Gaussian additive noise on common benchmark datasets [14].

Since this method of supervised learned denoising does not use any assumptions on the distribution of the noise $p(x|y)$, it works for a wide range of noise distributions, not just I.I.D. Gaussian noise. This property of learned denoising methods is especially useful to us since collective anomalies, as defined in the previous sections, can be seen as highly correlated noise and not at all as I.I.D. noise.

A remaining challenge that stops us from using the supervised learned denoising method is the need for noisy and clean signal pairs. Multiple methods for training a denoising neural network without clean signals have been developed [34, 32, 4]. All these methods share the fact that certain assumptions on the noise distribution have to be made to approximate the clean signal without using these clean directly signals. The method that most closely aligns with our assumptions on the overlay dataset is the training Noise2Noise method [34]. The Noise2Noise method trains an image-denoising neural network by using pairs of noisy images $\hat{x}_i, \tilde{x}_i$ sampled from the distribution $p(x|y_i)$ of noisy images with the same underlying clean image y_i, thus no longer needing the actual clean images y_i. The authors successfully train denoising models on synthetic and real-world image datasets consisting of these noisy image pairs. The Noise2Noise method performs only slightly worse than the usual supervised learned denoising method. In the Noise2Noise method the authors replace the clean targets y_i in equation 3.2 with the second noisy sample $\tilde{x}_i$ from the noisy pair sample pair $\hat{x}_i, \tilde{x}_i$ to minimize the training loss

$$\frac{1}{N}\sum_{i=1}^{N} L(f_\theta(\hat{x}_i), \tilde{x}_i). \tag{3.3}$$

The authors of the Noise2Noise paper show that given that the loss L is chosen as the L^2 loss, $\mathbb{E}(\hat{x}_i|\tilde{x}_i) = y_i$, and infinite data equation 3.2 and equation 3.3 have the same minimum for θ [34]. These sampling conditions are identical to the sampling conditions we made in section 1.2. Thus, minimizing this loss allows training a neural network for denoising without needing a distribution for the clean images $p(y)$, a distribution for the noise $p(x|y)$, or clean samples y_i. Instead, these distributions are indirectly "learned" from the data.

The authors of the Noise2Noise paper give multiple examples of their denoising method applied to real image datasets. Given different assumptions on the noise of the dataset, different norms are used in the training loss. If the noise is made to be mean zero or assumed to be mean zero, so $\mathbb{E}(\hat{x}_i|\tilde{x}_i) = y_i$, the L^2 norm in the loss of equation 3.3 leads to the best denoising neural network. If, instead, the noise is created by replacing some pixels with random values, the noise is no longer mean zero as the average of these random pixel values adds some bias. In these situations, the L^1 norm is shown to be the correct norm to use in the training loss if less than 50% of the pixels are corrupted, and the L^0 norm works when more than 50% of the pixels are corrupted. In an example of Monte Carlo-generated images, heavy tails of the noise distribution lead to problems in convergence during training. To remedy these convergence problems a relative L^2 loss function, defined as $(f_\theta(\hat{x}_i) - \tilde{x}_i)^2/(\hat{x}_i + \epsilon)^2$ with ϵ small, is used. For a dataset of MRI images, a different special loss function is developed based on how MRI images are generated. From these examples, we can conclude that while supervised and Noise2Noise training for denoising can lead to denoising models that perform almost equally well compared to the supervised counterpart, special attention to the noise distribution and accompanying norm used during

training with the Noise2Noise method should be given to lead to this equal performance. Either a bias is introduced by the wrong loss function which does not correspond to the noise distribution, or a loss function can destabilize the convergence of the optimization method.

3.3. Learning on spatially sparse data

We deemed the Noise2Noise method of the previous section to be a promising method for detecting the noise caused by outliers in the overlay data. This noise detection of measurements could then be used to form an outlier removal model that is interpretable and based on the data without manual labeling. One issue preventing the immediate implementation of the Noise2Noise model used by the original authors is that their model is based on convolutional neural networks. As seen in section 2.6 these CNN models use a learnable convolutional filter that uses local data structures. This convolutional filter assumes the input data is laid out in a lattice structure, which is true for the images used in the Noise2Noise paper but not for our overlay data. To our knowledge, the Noise2Noise method has only been used with CNN-based models. One option would be to interpolate the overlay data to a lattice structure. While certainly possible, this would introduce biases caused by the interpolation process. Any interpolation would smooth the overlay measurements between measurement points, while the real overlay pattern is not expected to be smooth, especially on field edges. We prefer a model that works with spatially sparse data, so measurements in an arbitrary spatial layout. We would also like to choose a model architecture that allows us to insert a priori information about the exposure process, such as the field layouts and the location of the edge of the wafer.

The point cloud approach

One approach would be to interpret the measured locations as a 2D point cloud with the overlay values attached to the points in the point cloud. Many deep learning architectures have been developed for tasks like point cloud shape classification, point cloud object detection, and point cloud segmentation [19]. One of the first model architectures for learning on 3D point clouds is PointNet [38]. The main issue this method tackles is that a point cloud has no natural ordering for its data points like an image or piece of text has, and to normal neural network architectures like an MLP, the ordering of the input vector matters. This problem is solved by separating the local features of the data point and the global features of the total point cloud. The local features are processed for every point, and from this, the global feature is created by combining the local feature vectors using a permutation invariant function $\oplus$. For the permutation invariant function $\oplus$, the authors used the max pool operation that takes the maximum values at each location in the vector of all the feature vectors. These local and global feature vectors are then concatenated and used to make a final prediction for each data point. Using this permutation invariant function causes the global features and the predicted scores to be identical for every ordering of the input data points. PointNet++ [39] improves on the PointNet architecture by not just considering the information at the points and the global information of the point cloud but also at intermediate levels. They develop a method that iteratively selects a subset of the points in the point cloud and executes a Pointnet layer on all the points in a ball around every point in this subset.

The practice of combining the feature vectors of points that are in some sense nearby with a permutation invariant function $\oplus$ is very similar to the message-passing neural network (MPNN) of section 2.7. The PointNet++ architecture can, while it was originally not, be implemented using a message-passing neural network where the nodes in the graph are the data points, and directed edges go from all the points in the ball to the point in the center of the ball.

Learning on meshes

The MeshGraphNet model [37], uses such a message-passing neural network, and defines a graph with both vertex and edge feature vectors, on which it uses the message-passing steps of section 2.7 to predict the next step in physics simulations. The reasoning for this approach is that physics simulations are often done on meshes, graphs with locations in space for the nodes, because these meshes can capture complicated geometries and allow for variable resolution such that locations of interest can have a higher resolution. Meshes can also describe complex objects like a piece of cloth or a deformed steel beam. Learning physics simulations directly on a mesh passes on these advantages compared to learning physics simulations with CNNs on a uniform mesh.

The MeshGraphNet model is tailored to time-dependent physics problems and trained to predict the

first and, when necessary, second-order derivatives of a physics simulation for a single time step. These predicted derivatives are then used in a forward-Euler integrator to generate the values for the next time step in the approximation of the simulation. By iteratively going through this process, predictions are made over multiple time steps. Every time step has as input the graph, the node feature vectors, and the edge feature vectors. For the graph, the same mesh is used that was for the target physics simulation. The current values of the simulation are stored on the node feature vectors concatenated with a one-hot encoding for the node type encoding, for example, information about whether the node is fixed in space or part of a surface in the physics simulation. The edge feature vectors are determined by the relative positioning of the nodes in space and the distance between the nodes. By using only relative positioning information, the model is made to be location invariant, preventing overfitting to a certain location in the simulation and forcing the model to learn the general relations for node interaction at different distances. The authors use, for the graph neural network model, an Encode-Process-Decode architecture [5], which is made up of multiple steps of the message-passing step of algorithm 1. The MeshGraphNet outperformed the same model implemented with the CNN-based neural network architecture U-NET [41] for the task of learning physics simulations, and further outperformed the same model implementation using Graph Convolutional Network layers [30].

As discussed in section 2.7, the spread of information on a graph in a message-passing neural network depends on the number of message-passing steps used. Thus, when the message-passing number is lower than the graph diameter, the possibility arises that necessary information is missing for an accurate approximation of the physics simulation. The authors of the MeshGraphNet also noticed this effect and showed that the error generally reduces as the number of message-passing steps is increased. This problem is even more noticeable when the number of nodes in the mesh becomes very large, on the order of thousands of nodes. Two works in the literature tackle this issue. Both methods add longer-range connections to the input graphs. [33] create a single ”multimesh” comprising 7 different meshes with an increasing node count from 12 to 40,962 nodes, then interlinked at the intersecting nodes. [17] also uses meshes with different node counts but does not connect them directly. Instead, the information flows between the different meshes using up- and down-sample graphs on which message-passing steps are performed. The first solution is the easiest to implement, while the second solution has the potential to be more computationally efficient since multiple message-passing steps can be performed on a computationally cheap coarse mesh for every message-passing step on a computationally expensive fine mesh.

The original MeshGraphNet paper aims to learn to predict physics simulations, meaning it learns the numerical approximation of the solution to a partial differential equation with initial conditions. However, the model is in no way designed to work only on simulation data. [27, 33] give an example of this by learning to predict medium-range weather based on reanalyzed weather data, which is based on real weather measurement data. Their model produces better weather predictions than the state-of-the-art weather forecasts on some metrics, and [33] uses a higher node count graph with long-range connections to outperform state-of-the-art weather forecasts on 90% of the metrics used. These two models show that data-based machine learning physics models have the potential to outperform hand-crafted partial differential equation-based models while, for this example, also using orders of magnitudes less energy.

3.4. The research gap

From the available public literature studied, we found [40] as the only paper discussing outlier removal methods for overlay data. This paper states the importance of removing outliers in the overlay dataset, and the methods used are interesting but do not leverage the large amount of data available. [42] does learn to predict overlay from a large dataset using physics-based biases but has a considerably different goal and could not directly be used to find outliers in the overlay data.

In the systematic review of [36], we found many machine learning-based methods that classify outliers, of which many are supervised and semi-supervised, leading to the need for manual labeling and introducing the risk of adding a human bias. Because the overlay dataset is similar to an image dataset, we looked into image denoising techniques, which could, in turn, be used to create an unsupervised overlay outlier removal technique. Many classic image denoising methods make strict assumptions about the image’s noise distribution and estimate the prior of all clean images [35]. Recently developed supervised learned denoising techniques drop the need to know the noise distributions and approxi-

mate the image distributions manually, but can learn to denoise given a large dataset of noisy-clean image pairs [14]. This would still not fit our data as we do not have any ”clean” overlay measurements, so we shortly discussed three unsupervised learning-based denoising methods [34, 32, 4], of which we deemed the Noise2Noise method [34] to fit our dataset best. It only requires the minimal assumption that the noise is mean zero, while some caution should be taken when selecting the loss function. Since the authors of the Noise2Noise paper apply the training method to image data, which is laid out in a uniform 2D grid, a CNN-based architecture is utilized. To our knowledge, this denoising method has not been utilized for overlay data.

The overlay data has a sparse layout of the measurement points on the wafer, and we thus choose an alternative neural network architecture since we do not want to interpolate the smooth overlay data to a uniform grid. One option would be to use an architecture designed for point clouds [39, 39]. These two architectures are similar to message-passing neural networks that define a convolution operation based on the neighborhood of a node in a graph, and using such a message-passing neural network allows for incorporating a priori physics-based biases into the model. A message-passing neural network model shown to learn and generalize physics simulations effectively is the MeshGraphNet model [37]. The MeshGraphNet model uses an MPNN-based Encode-Process-Decode architecture [5] with a numerical integrator step and an input graph that encodes the geometry of the input mesh using the relative edge encoding such that the model is spatially equivariant, in turn reducing the chance of overfitting. The node- and edge feature vectors can be encoded with extra a priori information. This means that we can use these encodings to mark the edge of an exposure field and the edge, locations where the overlay error is expected to be non-continuous and of increased amplitude respectively [31]. It will be interesting to see if these encodings will increase denoising performance. The temporal numerical integrator of the MeshGraphNet model will not be necessary since we are not predicting in time steps and do not use an underlying partial differential equation. The large number of nodes in the input graph could lead to a lower model accuracy [17] since information spread could then become limited by the number of message-passing steps. We would like to investigate if a graph with longer distance connections between nodes could increase performance. We will create a synthetic dataset with a ground truth denoised target we would like to approximate to verify all these model configurations.

Research questions

The central objective of this assignment is to make an effective outlier removal model based on the available overlay data. We have devised a possible method to perform such a task from related works. We have summarized our findings and will give more details of the devised method in chapter 4. To make the central objective more attainable, we have split the objective into several research questions stated below.

1. For the synthetic overlay dataset of section 4.3:
    - Does the model correctly predict the ground truth overlay?
    - Which loss function is best used for training the model with the Noise2Noise method?
    - Does the wafer edge encoding in the input graph lead to better denoising results?
    - Does the field edge encoding in the input graph lead to better denoising results?
    - Does an input graph with further connections lead to better denoising results?
2. For the real overlay dataset:
    - Does the model output an overlay prediction that is a lower mean squared error estimator for the second overlay measurement, exposed under the same conditions?
    - Can the model be used for effective outlier removal?

4

Methodology

This chapter will describe our model, designed to remove outliers in overlay measurements based on historical overlay data. We start with the Noise2Noise loss, which we mathematically show can approximate learned denoising with clean overlay measurements $\mathbf{y}_i$, without using these clean overlay measurements. After we define the loss, we will go into the message-passing neural network model and the data structure we will use as the input for this model. When we fit our model to the data, we will do this on a synthetic overlay dataset we created and the real overlay dataset. We elaborate on how the synthetic overlay dataset was created and how the real dataset was gathered. In the final section of this chapter, we will discuss how we fit the model to the data.

4.1. The Noise2Noise method

As discussed in section 3.4, we want to train a parameterized function $f_\theta(\cdot)$ that takes as input a noisy overlay measurement $\mathbf{x}_i$ performed on a test wafer and predicts, as accurately as possible, the clean noise-free overlay measurement $\mathbf{y}_i$ representing the actual machine overlay state. Because we do not have any clean noise-free overlay measurements $\mathbf{y}_i$, we will instead use the Noise2Noise method that can achieve the previously stated goal with a dataset that is made up of the noisy measurement pairs $\hat{\mathbf{x}}_i, \tilde{\mathbf{x}}_i$, where these noisy measurements are sampled according to the sampling procedure of 1.2. This sampling procedure assumes that we first sample the clean overlay measurement $\mathbf{y}_i$ from the distribution of all clean overlay measurements $p(\mathbf{y})$ and then sample $\hat{\mathbf{x}}_i, \tilde{\mathbf{x}}_i$ from the noisy overlay distribution conditional on the clean noise measurement $p(\mathbf{x}|\mathbf{y}_i)$. Thus we have

$$\mathbf{y}_i \sim p(\mathbf{y}), \quad \hat{\mathbf{x}}_i, \tilde{\mathbf{x}}_i \sim p(\mathbf{x}|\mathbf{y}_i). \tag{4.1}$$

To explain the Noise2Noise method, suppose we have infinite data and use the L^2 norm for the loss as defined in equation 1.3. The method describes that we should choose for θ the minimum of the expectation

$$\mathbb{E}_{(\hat{\mathbf{x}},\tilde{\mathbf{x}})}\left[L^2\left(f_\theta(\hat{\mathbf{x}}), \tilde{\mathbf{x}}\right)\right]. \tag{4.2}$$

Using the rule of total expectation, this is equal to

$$\mathbb{E}_{\hat{\mathbf{x}}}\left[\mathbb{E}_{\tilde{\mathbf{x}}|\hat{\mathbf{x}}}\left[L^2\left(f_\theta(\hat{\mathbf{x}}), \tilde{\mathbf{x}}\right)\right]\right]. \tag{4.3}$$

The class of minimizers for the number z in the expectation $\mathbb{E}_x L(z, x)$ is known as M-estimators [22]. When the loss function equals the L^2 loss function, this expectation is minimized by taking $z = \mathbb{E}_x x$. Thus this means that the expectation of equation 4.3 and in turn the expectation of equation 4.2 is minimized when $f_\theta(\hat{\mathbf{x}}) = \mathbb{E}_{\tilde{\mathbf{x}}|\hat{\mathbf{x}}}\tilde{\mathbf{x}} = \mathbb{E}\tilde{\mathbf{x}}|\hat{\mathbf{x}}$. Because the analytic distribution of $\hat{\mathbf{x}}_i$ and $\tilde{\mathbf{x}}_i$ is not known, and we only observe a finite number of samples from this distribution, we approximate the expression of equation 4.2 with its estimator

$$\frac{1}{N}\sum_{i=1}^{N} L^2\left(f_\theta(\hat{\mathbf{x}}_i), \tilde{\mathbf{x}}_i\right). \tag{4.4}$$

Given infinite data, so $N \to \infty$, we expect this estimator to converge to the same value as equation 4.2. This estimator is in that case minimized when $f_\theta(\hat{\mathbf{x}}_i) = \mathbb{E}\tilde{\mathbf{x}}_i|\hat{\mathbf{x}}_i$. Because of the assumed sampling process of equation 4.1, conditioning $\tilde{\mathbf{x}}_i$ on $\hat{\mathbf{x}}_i$ is identical to conditioning $\tilde{\mathbf{x}}_i$ on $\mathbf{y}_i$, so $\mathbb{E}\tilde{\mathbf{x}}_i|\hat{\mathbf{x}}_i = \mathbb{E}\tilde{\mathbf{x}}_i|\mathbf{y}_i$. In section 1.2, we made the further assumption that the noise in the overlay measurement, caused by outliers and measurement errors, is mean zero, thus $\mathbb{E}\tilde{\mathbf{x}}_i|\mathbf{y}_i = \mathbf{y}_i$. Putting this all together, we can conclude that following our assumptions, equation 4.4 is minimized for infinite data when

$$f_\theta(\hat{\mathbf{x}}_i) = \mathbf{y}_i. \tag{4.5}$$

Turning this around, the closer equation 4.4 is to its minimum, achieved myy minimizing the loss with respect to θ, the closer we expect $f_\theta(\hat{\mathbf{x}}_i)$ to be to $\mathbf{y}_i$.

Since the number of nodes in an overlay measurement M_i can vary for each wafer pair i, it is hard to compare the $L^2\left(f_\theta(\hat{\mathbf{x}}_i), \tilde{\mathbf{x}}_i\right)$ between wafers. That is why we will use for the training loss the mean squared error where the mean is taken over all the measurement nodes so

$$MSE(f_\theta(\hat{\mathbf{x}}), \tilde{\mathbf{x}}) := \frac{1}{\sum_{i=1}^{N} M_i} \sum_{i=1}^{N} L^2\left(f_\theta(\hat{\mathbf{x}}_i), \tilde{\mathbf{x}}_i\right). \tag{4.6}$$

Since this is just a constant multiple of equation 4.4, it will not change the optimization problem.

If we find the function that perfectly denoises $\tilde{\mathbf{x}}_i$ s.t. $f_\theta(\tilde{\mathbf{x}}_i) = \mathbf{y}_i$, the mean squared error of equation 4.6 will not be zero. Instead this loss function is zero if the function $f_\theta(\cdot)$ predicts, from the first noisy sample $\tilde{\mathbf{x}}_i$, the second noisy sample $\hat{\mathbf{x}}_i$ s.t. $f_\theta(\tilde{\mathbf{x}}_i) = \hat{\mathbf{x}}_i$. In the objective of equation 4.3, it is not possible to attain this equality if $Var(\hat{\mathbf{x}}_i) > 0$, and thus it is, in turn, impossible to achieve zero loss. For finite data the function can overfit to the specific noise samples and thus achieve zero loss. To measure how well the function $f_\theta(\cdot)$ is denoising we will instead use, when available, the ground truth mean squared error

$$MSE(f_\theta(\hat{\mathbf{x}}), \mathbf{y}) := \frac{1}{\sum_{i=1}^{N} M_i} \sum_{i=1}^{N} L^2\left(f_\theta(\hat{\mathbf{x}}_i), \mathbf{y}_i\right), \tag{4.7}$$

which is equal to zero when we find a perfect denoising function $f_\theta(\cdot)$.

4.2. The model architecture

In the previous section, we stated the training loss we want to use to fit the function $f_\theta(\cdot)$. We will in this section elaborate how we implement the function $f_\theta(\cdot)$. In section 3.4, we decided we wanted to use a message-passing neural network (MPNN) with an Encode-Process-Decode architecture [5], where the input graph has edge and vertex encodings that represent both the physical layout of the measurement points and physics-based prior information for the relation between the nodes. For this model, we must define an input graph $G_i = (V_i, E_i)$ consisting of the nodes V_i and edges E_i on which the message-passing steps will be performed. For each node $v \in V_i$, we also need to define a node feature vector $\mathbf{v}_v$ and further, for every edge $\{v, u\} \in E$, we must define two directed edge feature vectors $\mathbf{e}_{u,v}$ and $\mathbf{e}_{v,u}$. This input graph and the feature vectors are then processed in the MPNN to output overlay measurement for every measurement location.

The input graph

For each measurement $\hat{\mathbf{x}}_i$, we need to construct a graph $G_i = (V_i, E_i)$, which is then used by our message-passing neural network. The graph nodes will represent the measurement points. So for a wafer measurement pair i, we define the nodes of the input graph as $V_i = [M_i]$. We have great freedom to choose the set of edges E_i. An edge in the graph represents a connection between the nodes, so ideally, the most related nodes should be connected with an edge. At the same time, too many edges could make training and inference slow without much higher accuracy. We will assume that measurements close to each other on the wafer are more related, so should share an edge. We would also like G_i to be a connected graph, meaning that there is a path of edges connecting every pair of nodes in the graph. One option would be to use a k-nearest neighbors graph, which is a graph in which two vertices $v, u \in V$ are connected by an edge if the distance between v and u is among the k-th smallest distances from v to other nodes from V. The resulting graph is not guaranteed to

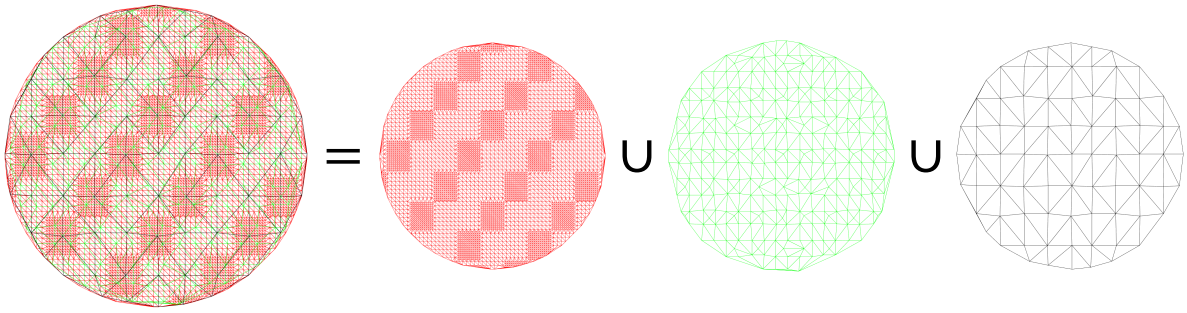


Figure 4.1: The graph used for the message-passing neural network, where the edges E_i of this graph are a union of the edge sets E_i^{fine}, E_i^{medium}, and E_i^{coarse} shown in red, green, and black respectively. These edge sets are, in turn, Delaunay triangulations of the vertex sets V_i, V_i^{medium}, and V_i^{coarse} where the measurement locations $\mathbf{u}_i$ are used for the triangulation. This graph with both short and long edges was chosen such that, in theory, information can be gathered both locally and from further distances using a minimal number of message-passing steps.

be connected, and to not deviate too far from the meshes of the MeshGraphNet paper, which inspired our model, we will instead use a Delaunay triangulation [24]. This triangulation is often used to create meshes in finite element method solvers. Other ways of constructing the graphs where not tested.

Inspired by [33], we will construct the graph from multiple graphs that have a different amount of nodes and thus also different length edges. The longer edges in the resulting graph should increase the speed of information spread and in turn, the model's accuracy even if fewer message-passing steps are used [17]. The finest mesh created will be the Delaunay triangulation of all the vertices, where we will denote this set of edges as $E_i^{\text{fine}} = Delaunay(V_i, \mathbf{u}_i)$, where V_i denotes the set of vertices used for the triangulation and $\mathbf{u}_i$ contains the location of all the measurement points on the wafer. The second medium mesh will be created using a strict subset of V_i. This subset $V_i^{\text{medium}} \subset V_i$ contains, for each field exposed 5 nodes chosen in an X-shape with one measurement point in the middle of the field and 4 near the corners. The resulting set of edges $E_i^{\text{medium}} = Delaunay(V_i^{\text{medium}}, \mathbf{u}_i)$, for the synthetic dataset, can be seen in the green mesh of Figure 4.1. The last set of edges is made by first defining $V_i^{\text{coarse}} \subset V_i^{\text{medium}}$, the measurement points closest to the center of each field, and then creating $E_i^{\text{coarse}} = Delaunay(V_i^{\text{coarse}}, \mathbf{u}_i)$. The mesh created using this method is visible in black in Figure 4.1. The full process defined in formulas is thus:

$$V_i = [M_i] \tag{4.8}$$
$$V_i \supset V_i^{\text{medium}} \supset V_i^{\text{coarse}} \tag{4.9}$$
$$E_i^{\text{fine}} = Delaunay(V_i, \mathbf{u}_i) \tag{4.10}$$
$$E_i^{\text{medium}} = Delaunay(V_i^{\text{medium}}, \mathbf{u}_i) \tag{4.11}$$
$$E_i^{\text{coarse}} = Delaunay(V_i^{\text{coarse}}, \mathbf{u}_i) \tag{4.12}$$
$$E_i = E_i^{\text{fine}} \cup E_i^{\text{medium}} \cup E_i^{\text{coarse}} \tag{4.13}$$
$$G_i = (V_i, E_i) \tag{4.14}$$

The vertex and edge feature vectors

Using the message-passing step of section 2.7 we can encode for every node $v \in V$ a feature vector $\mathbf{v}_v$, and for every edge $\{u, v\} \in E$ two directional feature vectors $\mathbf{e}_{v,u}$ and $\mathbf{e}_{u,v}$. These feature vectors will contain data we want the MPNN to use for its predictions. The overlay vectors $\hat{x}_{i,j} \in \mathbb{R}^2$ form the main input for the denoising MPNN $f_\theta(\cdot)$ and each overlay vector $\hat{x}_{i,v}$ is added to the vertex feature vector $\mathbf{v}_v$ of its measurement index v. The authors of the MeshGraphNet model further use the vertex feature vectors to store the types of vertex, such as which nodes are stationary in the simulation. The MeshGraphNet model uses the edge feature vectors to store the geometry of the input mesh, as the graph on which the calculations are done does not have a spatial geometry of itself. We will use their way of encoding the mesh geometry and similarly give a special encoding to vertex and edge feature vectors we want our model to fit differently.

We will start with encoding the geometry of the mesh. A possible approach would be to take the coordinates of the measurement points, $u_{i,v} \in \mathbb{R}^2$ for $v \in V_i$, and directly concatenate them into the vertex feature vectors $\mathbf{v}_v$. This would allow the model to fit to specific coordinates on the wafer instead of recognizing patterns that repeat over the whole wafer, which could lead to overfitting. The alternative approach proposed by the authors of MeshGraphNet is to only use relative position information of vertices connected by an edge. This is done by concatenating relative position information between two nodes $\{u, v\} \in E$ into the edge feature vectors $\mathbf{e}_{u,v}$ and $\mathbf{e}_{v,u}$. The relative position information we will use for the edge feature vector $\mathbf{e}_{u,v}$ is the relative location $u_{i,u} - u_{i,v} \in \mathbb{R}^2$ and the Euclidean distance between the measurement points $||u_{i,u} - u_{i,v}||_2 \in \mathbb{R}$, as used in the MeshGraphNet model. This way of encoding ensures that our model is location invariant, meaning it will make the same prediction, independent of the location if all other variables are the same. This should prevent overfitting as the MPNN learns to make accurate predictions that generalize to different locations.

For our edge and vertex feature vectors with a special encoding, we have selected 2 areas that we would like to mark. The first area is the edge of the wafer, where we expect higher variance in the overlay error [31]. The second area marked is the borders between the exposed fields where we expect the overlay error to be non-continuous [31]. We have decided to mark the edge of the wafer by using a one-hot encoding for the vertex feature vectors close to the border defined as:

$$k_v^{\text{wafer edge}} = \begin{cases} 1 & \text{if } ||u_{i,v}||_2 > 0.141 \\ 0 & \text{else} \end{cases} \tag{4.15}$$

This value of 0.141 was chosen such that only vertices near the wafer edge got a special encoding as can be seen in Figure 4.2. We have decided to mark the borders between fields by giving a one-hot encoding to the edges conditional to if this edge crosses a field border, so we write this as:

$$k_{u,v}^{\text{field border}} = \begin{cases} 1 & \text{if } u \text{ and } v \text{ are from the same exposure field} \\ 0 & \text{else} \end{cases} \tag{4.16}$$

These two encodings have been visualized in the mesh of Figure 4.2.

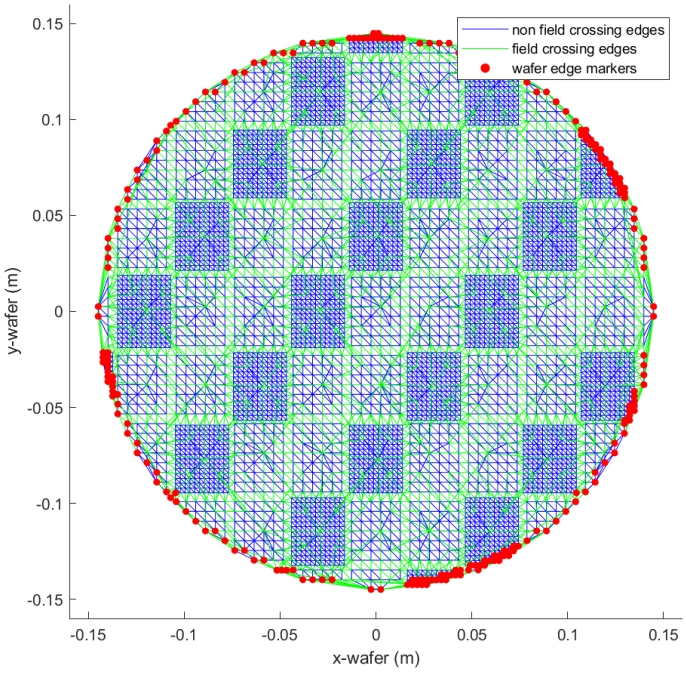


Figure 4.2: A visualization of the wafer edge vertex encoding $k_v^{\text{wafer edge}}$ and the field border encodings $k_{u,v}^{\text{field border}}$ on the mesh of Figure 4.1.

The total input for the model is summarized in Table 4.1, where all the features are concatenated into the feature vectors s.t. $\mathbf{v}_v \in \mathbb{R}^3$ and $\mathbf{e}_{v,u} \in \mathbb{R}^4$.

Table 4.1: The features which concatenated together form the vertex feature vectors $\mathbf{v}_v$ for $v \in V$ and edge feature vectors $\mathbf{e}_{v,u}$ and $\mathbf{e}_{u,v}$ for $\{v, u\} \in E$.

Feature vector	Features
$\mathbf{v}_v$	$\hat{x}_{i,v}, k_v^{\text{wafer edge}}$
$\mathbf{e}_{v,u}$	$u_{i,u} - u_{i,v}, \|\|u_{i,u} - u_{i,v}\|\|_2, k_{v,u}^{\text{field border}}$

From input to output

Now that we have defined for each overlay measurement $\hat{\mathbf{x}}_i$, how we construct the computation graph for our MPNN $G_i = (V_i, E_i)$, how we encode the vertex features $\{\mathbf{v}_v\}_{v \in V_i}$, and how we encode the edge features $\{\mathbf{e}_{v,u}, \mathbf{e}_{u,v}\}_{\{u,v\} \in E_i}$, we will define how our model $f_\theta(\cdot)$ uses these inputs to give a prediction. The architecture we chose is an adaptation of the MeshGraphNet model [37], which uses, in turn, an Encoder-Processor-Decoder message-passing neural network architecture proposed by [5]. For our adaptation of the MeshGraphNet model, we changed the input dimensions to our vertex- and edge feature vector size and the output size to be equal to that of the denoised overlay measurement, namely $\mathbb{R}^{2 \times M_i}$. We further removed the forward Euler step of the MeshGraphNet model as we are not predicting the next time step of a known differential equation. The resulting architecture is described in the pseudo-code of Algorithm 2 and can be divided into the encoder step, the processor step, and the decoder step.

The encoding step takes the vertex- and edge feature vectors from respectively $\mathbb{R}^3$ and $\mathbb{R}^4$ to their embedding in $\mathbb{R}^{128}$, where this embedding dimension is the same as in the MeshGraphNet model. The encoder is made up of a single message-passing step of section 2.7. For the permutation invariant function $\oplus$ we use the sum $\sum$ is used as this is also the function used in the MeshGraphNet model. For the learnable functions $\psi_\theta(\cdot)$ and $\phi_\theta(\cdot)$ we use $f_{\theta_{0,1}}^{\text{MLP}} : \mathbb{R}^{10} \to \mathbb{R}^{128}$ and $f_{\theta_{0,2}}^{\text{MLP}} : \mathbb{R}^{131} \to \mathbb{R}^{128}$. These functions first concatenate all their inputs to a single vector and then perform the Multi-Layer Perceptron of section 2.3 with a hidden dimension of 128 and use the $ReLu(x) = max(0, x)$ function as activation function. In our implementation, the multilayer perceptrons also have a layer normalization step [3], which has been shown to increase the speed of convergence during training. In the pseudo-code, all parameter sets θ are given indexing to indicate which parameters are used in the MLP.

The processing step uses N^{MPS} repeated message-passing steps to iteratively update the vertex- and edge embedding vectors $\mathbf{e}'_{u,v} \in \mathbb{R}^{128}$ and $\mathbf{v}'_v \in \mathbb{R}^{128}$. The learnable functions are implemented in the same way as for the encoder step, but now with differing input dimensions, so for all $l \in [N^{\text{MPS}}]$ we have $f_{\theta_{l,1}}^{\text{MLP}} : \mathbb{R}^{384} \to \mathbb{R}^{128}$ and $f_{\theta_{l,2}}^{\text{MLP}} : \mathbb{R}^{256} \to \mathbb{R}^{128}$. A second difference with the message-passing step in the encoder is that the embedding vectors are updated by adding the output of the MLP to the last value of the embedding vector. These iterative updates are called residual connections in the deep learning literature and have been shown to help with the convergence of ”deep” neural networks with many consecutive layers [21]. The number of message-passing steps N^{MPS} for our model is 20. This number was chosen after the results of section 5.1.

The decoder step consisting of a single MLP $f_{\theta_{l+1}}^{\text{MLP}} : \mathbb{R}^{128} \to \mathbb{R}^2$ which uses the vector embeddings $\mathbf{v}'_v$ to predict the overlay values x_v^{out} at each measurement location index $v \in V$. These vectors x_v^{out} are the entries in the output tensor $f_\theta(\cdot)$. The set of all weights θ in $f_\theta(\cdot)$ are all the weights in the Multi Layer Perceptrons, so $\theta = \{\theta_{0,1}, \theta_{0,1}, \theta_{1,1}, \theta_{1,2}, \ldots, \theta_{N^{\text{MPS}},1}, \theta_{N^{\text{MPS}},2}, \theta_{N^{\text{MPS}}+1}\}$. Our model with 20 message-passing steps has 3.8 million tunable parameters.

We implemented our model using the PyTorch Geometric [15] Python package for training large Graph Neural Networks. We started with an open-source PyTorch Geometric implementation of the MeshGraphNet model [26] and made our model changes to this implementation.

Algorithm 2: Our overlay denoising model which uses a Encoder-Processor-Decoder message-passing neural network architecture [5].

// The input of our model is the graph $G_i = (V_i, E_i)$, used for the message passing steps, and the vertex- and edge feature vectors $\{\mathbf{v}_v\}_{v \in V_i}$ and $\{\mathbf{e}_{v,u}, \mathbf{e}_{u,v}\}_{\{u,v\} \in E_i}$

// The encoder step consisting of a single message passing step embedding the edge and vertex feature vectors.

for $\{u, v\} \in E_i$ **do**

$\mathbf{e}'_{u,v} \leftarrow f^{\text{MLP}}_{\theta_{0,1}}(\mathbf{v}_u, \mathbf{v}_v, \mathbf{e}_{u,v})$

$\mathbf{e}'_{v,u} \leftarrow f^{\text{MLP}}_{\theta_{0,1}}(\mathbf{v}_v, \mathbf{v}_u, \mathbf{e}_{v,u})$

end

for $v \in V_i$ **do**

$\mathbf{v}'_v \leftarrow f^{\text{MLP}}_{\theta_{0,2}}\left(\mathbf{v}_v, \sum_{u \in \mathcal{N}_v} \mathbf{e}'_{u,v}\right)$

end

// The processor step consisting of N^{MPS} message passing step with residual connection.

for $l \in \{1, \ldots, N^{MPS}\}$ **do**

for $\{u, v\} \in E_i$ **do**

$\mathbf{e}'_{u,v} \leftarrow \mathbf{e}'_{u,v} + f^{\text{MLP}}_{\theta_{l,1}}(\mathbf{v}'_u, \mathbf{v}'_v, \mathbf{e}'_{u,v})$

$\mathbf{e}'_{v,u} \leftarrow \mathbf{e}'_{u,v} + f^{\text{MLP}}_{\theta_{l,1}}(\mathbf{v}'_v, \mathbf{v}'_u, \mathbf{e}'_{v,u})$

end

for $v \in V_i$ **do**

$\mathbf{v}'_v \leftarrow \mathbf{v}'_v + f^{\text{MLP}}_{\theta_{l,2}}\left(\mathbf{v}'_v, \sum_{u \in \mathcal{N}_v} \mathbf{e}'_{u,v}\right)$

end

end

// The decoder step consisting of a single MLP that outputs the predicted overlay value for each vertex.

for $v \in V_i$ **do**

$x^{\text{out}}_v \leftarrow f^{\text{MLP}}_{\theta_{N^{\text{MPS}}+1}}(\mathbf{v}'_v)$

end

Data augmentation during training

Significant overfitting occurred during training on synthetic and real datasets, so we added the option to randomly augment the data during training. Due to the relative location encoding of the edge feature vectors, the model is location equivariant. Translations of the input data will not change the feature vectors and, in turn, the model's output. However, The model is not rotation equivariant, which means that rotating the overlay data will not guarantee the output is the same as the original output rotated by the same angle. If we want our model to learn to give rotation equivariant predictions, we can achieve this by performing random rotation data augmentation during training.

Specifically, we have implemented this by, for each wafer measurement pair i, sampling the rotation angle ϕ and using this angle for the 2D rotation matrix R, so

$$\phi \sim U(0, 2\pi), \quad R = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}. \tag{4.17}$$

We then use this rotation matrix to rotate, for all measurement locations $j \in [M_i]$, both the location relative to the center of the wafer $u_{i,j}$, the input overlay vector $\hat{x}_{i,j}$, and the target overlay vector $\tilde{x}_{i,j}$, so the new values for vectors are

$$u^*_{i,j} = Ru_{i,j}, \quad \hat{x}^*_{i,j} = R\hat{x}_{i,j}, \quad \tilde{x}^*_{i,j} = R\tilde{x}_{i,j}. \tag{4.18}$$

4.3. The synthetic overlay dataset generation

Noise2Noise training allows a neural network to denoise real-life measurements for which no ground truth denoised measurement version is known. However, this lack of ground truth complicates the validation process since the model output can not be directly compared to any ground truth noise-free target. Another problem is that overlay measurements are sensitive customer information and cannot be published here. To remedy these two concerns, we have created a synthetic overlay dataset with an interfield, intrafield, and radial effect in the overlay data we also expect in our real overlay dataset [31]. This synthetic overlay dataset can then be used to verify parts of our model and is used for all overlay plots in this thesis.

The synthetic overlay sampling process

Following the sampling process described in section 1.2, we will first sample the ground truth machine overlay state $\mathbf{y}_i$ and thereafter sample the noisy overlay measurements $\hat{\mathbf{x}}_i$ and $\tilde{\mathbf{x}}_i$ conditional on $\mathbf{y}_i$, thus as

$$\mathbf{y}_i \sim p(\mathbf{y}), \quad \hat{\mathbf{x}}_i, \tilde{\mathbf{x}}_i \sim p(\mathbf{x}|\mathbf{y}_i). \tag{4.19}$$

where in this the section sampling method for the distributions $p(\mathbf{y})$ and $p(\mathbf{x}|\mathbf{y}_i)$ is defined for the synthetic dataset. For the ground truth noise-free overlay value $\mathbf{y}_i$ we will split the overlay into three distinct commonly observed overlay error sources called the interfield, intrafield, and radial overlay error [31]. A method for sampling these three overlay sources will be defined, after which they are added together. So the sampling process for $\mathbf{y}_i$ in the synthetic dataset is

$$\mathbf{y}_i^{\text{interfield}} \sim p(\mathbf{y}^{\text{interfield}}), \tag{4.20}$$
$$\mathbf{y}_i^{\text{intrafield}} \sim p(\mathbf{y}^{\text{intrafield}}), \tag{4.21}$$
$$\mathbf{y}_i^{\text{radial}} \sim p(\mathbf{y}^{\text{radial}}), \tag{4.22}$$
$$\mathbf{y}_i = \mathbf{y}_i^{\text{interfield}} + \mathbf{y}_i^{\text{intrafield}} + \mathbf{y}_i^{\text{radial}}. \tag{4.23}$$

Once $\mathbf{y}_i$ has been sampled, we can continue to sample the noisy overlay measurements $\hat{\mathbf{x}}_i$ and $\tilde{\mathbf{x}}_i$. The noise added to these samples has been split up into global noise $\mathbf{n}_i^{\text{global}}$ representing larger defects, and local noise $\mathbf{n}_i^{\text{local}}$ representing measuring uncertainty at each measurement point. The global noise $\mathbf{n}_i^{\text{global}}$ is independent of $\mathbf{y}_i$, but the local noise $\mathbf{n}_i^{\text{local}}$ is dependent both on $\mathbf{y}_i$ and on $\mathbf{n}_i^{\text{global}}$. So the sampling process for $\hat{\mathbf{x}}_i$ and $\tilde{\mathbf{x}}_i$ in the synthetic dataset is

$$\hat{\mathbf{n}}_i^{\text{global}} \sim p(\mathbf{n}^{\text{global}}), \qquad \tilde{\mathbf{n}}_i^{\text{global}} \sim p(\mathbf{n}^{\text{global}}), \tag{4.24}$$
$$\hat{\mathbf{n}}_i^{\text{local}} \sim p(\mathbf{n}^{\text{local}}|\mathbf{y}_i, \hat{\mathbf{n}}_i^{\text{global}}), \qquad \tilde{\mathbf{n}}_i^{\text{local}} \sim p(\mathbf{n}^{\text{local}}|\mathbf{y}_i, \tilde{\mathbf{n}}_i^{\text{global}}), \tag{4.25}$$
$$\hat{\mathbf{x}}_i = \mathbf{y}_i + \hat{\mathbf{n}}_i^{\text{global}} + \hat{\mathbf{n}}_i^{\text{global}}, \qquad \tilde{\mathbf{x}}_i = \mathbf{y}_i + \tilde{\mathbf{n}}_i^{\text{global}} + \tilde{\mathbf{n}}_i^{\text{global}}. \tag{4.26}$$

The overlay measurements are done at different locations $\mathbf{u}_i$ on the wafer. The photolithography process exposes fields on the wafer. Each field contains many markers spread out over the field. Not every marker is read in the measurement process; some fields are more densely read out than others. The precise layout is classified, but we will use a self-made layout of 5167 measurement locations spread out over alternating densely and sparsely sampled fields, as shown in Figure 4.4. The synthetic dataset can, however, be created for any layout specified. Unlike with the real dataset, the locations $\mathbf{u}_i$ for the synthetic dataset are all the same for each batch $1 \leq i \leq N$

This model for generating the synthetic overlay data was designed to have the core components of an overlay measurement and representative defects that are hard to distinguish from the signal ground truth overlay. The synthetic overlay data has not been fitted to real overlay data but has been created by trial and error. There are considerable differences between the real and the synthetic datasets. Two main differences are that y_i seems relatively too smooth in the synthetic dataset and that the noise in the real dataset is made up of significantly more diverse types of shapes coming from many differing sources. Since the model used to denoise real overlay data will not be trained on synthetic data this should not impact its performance.

The noise pattern

For the stochastic patterns in the overlay dataset, we would like to use a smooth signal with details at different coarseness levels that can be sampled from a randomized procedure. For this, we implemented a fractal-like noise pattern by [44]. Their algorithm adds together uniformly sampled random matrices, which are to an increasing level using cubic interpolation and then magnified. The code for these patterns can be read in appendix C. A sample of these noisy matrices and the final result from addition can be seen in Figure 4.3. The values in the resulting array can be negative and positive and are normalized by dividing by the standard deviation of the array.

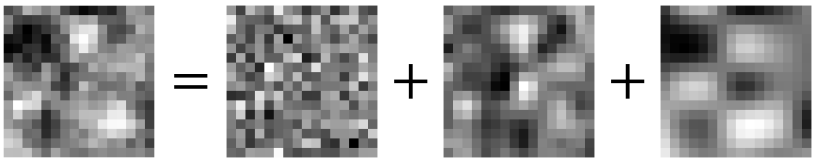


Figure 4.3: A graphical representation of how the fractal-like pattern is created by adding up the different matrices sampled from a uniform distribution increasingly augmented using interpolation and magnification. The code for these samples is available in appendix C.

Ground truth overlay

The ground truth overlay $\mathbf{y}_i$ in our synthetic overlay model comprises three sources of overlay error. The first modeled source of overlay error is the so-called interfield overlay error. This is defined as the overlay error that occurs over the whole wafer and can, for example, come from the degradation of the chuck on which the wafer is placed. To simulate this interfield overlay error, we sample two fractal-like noise matrices and interpolate these linearly at the measurement locations $\mathbf{u}_i$ to end up with the synthetic interfield overlay values $dx_{i,j}$ and $dy_{i,j}$. These values are then combined for all measurement points on the wafer to form the ground truth interfield overlay $\mathbf{y}_i^{\text{interfield}}$ for each wafer pair i. Figure 4.4 illustrates this process and the resulting pattern. All scaling factors and full implementation of the process are available in appendix C.

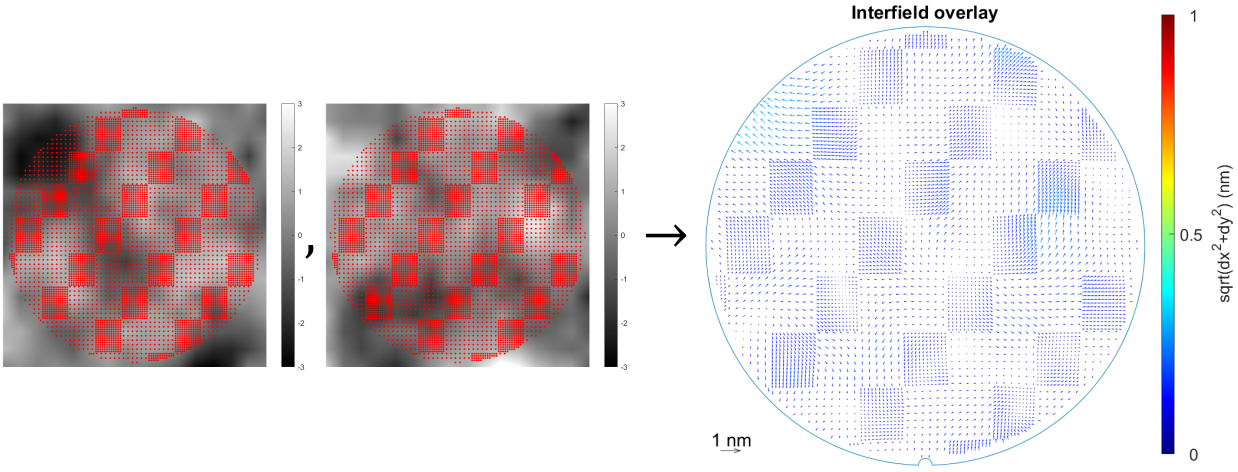


Figure 4.4: The interfield overlay $\mathbf{y}_i^{\text{interfield}}$ represents a continuous deformation over the whole wafer. This illustration shows how we sample the overlay values for the dx and dy directions and the resulting overlay pattern.

The second source of ground truth overlay error modeled is intrafield overlay $\mathbf{y}_i^{\text{intrafield}}$. This overlay pattern represents a deviation in how every field is printed and can, for example, come from drift in the calibration of the optics. Because this overlay error is the same for every field, it is sampled from two noise patterns and repeated for every field. The process of generating the intrafield overlay pattern and the resulting overlay is depicted in Figure 4.5.

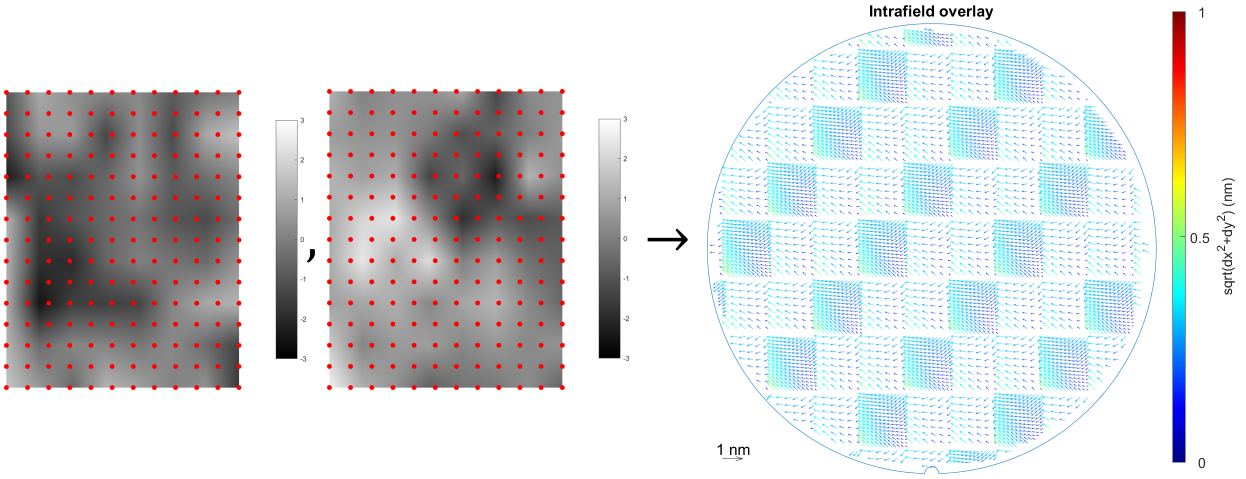


Figure 4.5: The intrafield overlay $\mathbf{y}_i^{\text{intrafield}}$ is made up of the overlay error that occurs repeatedly for each field that is exposed on the wafer. Thus, to construct the synthetic intrafield overlay, we sample the $dx_{i,j}$ and $dy_{i,j}$ from two noise arrays and repeat this overlay for each field of the wafer as illustrated above.

The last overlay pattern incorporated in the ground truth overlay is a radial overlay pattern $\mathbf{y}_i^{\text{radial}}$. Because the wafer is clamped and this wafer clamp tends to wear faster on the edge than in the center, bigger overlay values are expected near the edge of the wafer. This results in overlay measurements that mostly point inward or outward, orthogonal to the wafer edge. To simulate this effect, we first create outward pointing vectors by taking the distance vectors of the measurement locations $u_{i,j}$, multiplying them by their length raised to the 16th power, and then normalizing their lengths to $[0, 1]$, defined as

$$r_{i,j} = u_{i,j} \cdot \frac{\left(||u_{i,j}||_2\right)^{16}}{\max_{1 \le j \le n} \left(||u_{i,j}||_2\right)^{16}}, \tag{4.27}$$

where $u_{i,j}$ is the vector of the measurement location relative to the center of the wafer. The 16th power was chosen so that the spread of radial overlay was similar to the pattern on real wafers. The resulting vectors are then multiplied by the noise pattern to make their length and in- or outward orientation stochastic. Figure 4.6 illustrates this process and the resulting overlay pattern.

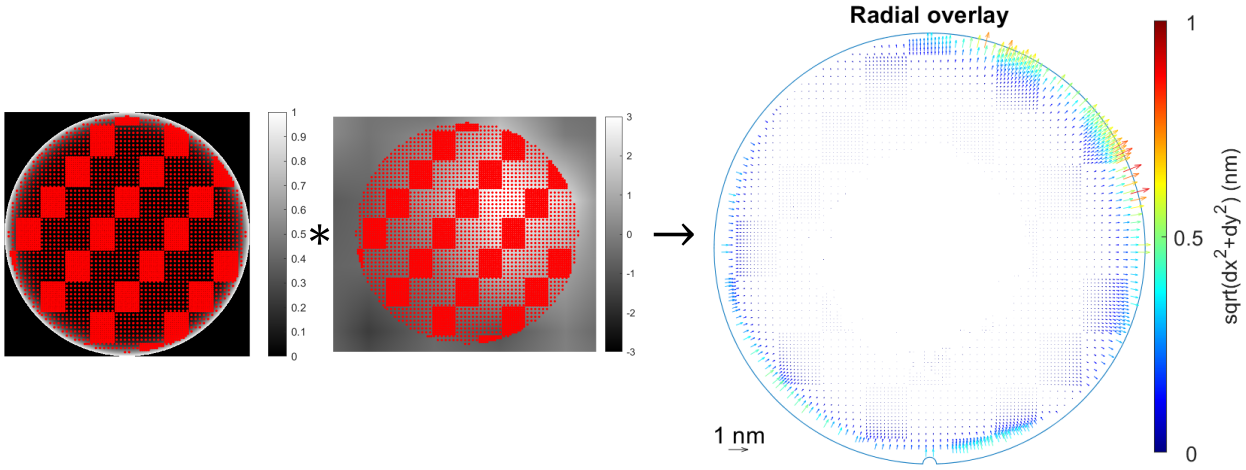


Figure 4.6: An illustration of how the ground truth radial overlay pattern $\mathbf{y}_i^{\text{radial}}$ is created. First, outward vectors $r_{i,j}$ are created according to equation 4.27. These vectors are then multiplied by an interpolated noise pattern and scaled to the nanometer level to create the pattern visible on the right.

The interfield, intrafield, and radial overlay pattern are subsequently added together to form a single ground truth overlay $\mathbf{y}_i$ sample, so

$$\mathbf{y}_i = \mathbf{y}_i^{\text{interfield}} + \mathbf{y}_i^{\text{interfield}} + \mathbf{y}_i^{\text{radial}}. \tag{4.28}$$

An illustration of this addition and the resulting overlay pattern can be seen in Figure 4.7. The parameters used to weigh each overlay pattern and all the code for the synthetic data generation process are available in appendix C.

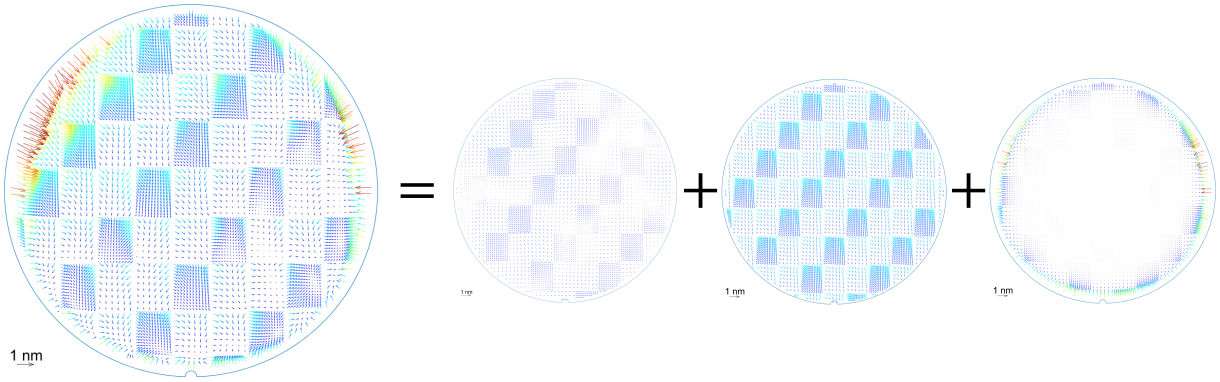


Figure 4.7: The ground truth overlay representing the scanner state is then created by adding up the interfield overlay pattern, the intrafield overlay pattern, and the radial overlay pattern.

Added Noise

After the ground truth has been established, the two samples of synthetic overlay data can be created by adding the noise $\mathbf{n}_i$ to the ground truth overlay. Firstly the global noise $\mathbf{n}_i^{\text{global}}$ is generated. This global noise spans multiple measurement points and represents contaminations during exposure. Because these errors can come from contaminations on the underside of the wafer [6], they often form "bumps" in the overlay, resulting in outward-pointing overlay vectors surrounding the contamination point. These defects can come in many different shapes and sizes, and we would thus like $\mathbf{n}_i^{\text{global}}$ also to have a wide range of possible shapes and sizes, such that the model can demonstrate it can even recognize complicated stochastic defect patterns.

First, the number of global outliers is determined by sampling a $Geo(0.3)$ distribution, after which each global outlier gets a $x-$ and $y-$coordinate sampled from $U(-0.15, 0.15)$. The basis shape for the global outliers is the bivariate normal probability density function with covariance matrix $(0.0025+\sigma)\cdot I_2$, $\sigma \sim Exp(0.01)$ and mean vector equal to the $x-$ and $y-$coordinate of the outlier. For the radial pattern, the gradient vectors at the measurement locations on the probability density function are used which is then combined with two samples of the fractal-like noise multiplied by the bivariate normal density function. The resulting outlier pattern is then scaled by an amplitude sampled from $0.4 + Exp(0.5)$. The Matlab code that produced these outliers is available in appendix C. Figure 4.8 shows two examples of synthetic global errors $\mathbf{n}_i^{\text{global}}$, where the examples were chosen because they had a large amount of these synthetic defects. The amount of outliers is over-represented in the synthetic dataset. This should mean fewer samples are needed to train the model, greatly lowering the required computational resources.

After adding these defects, we generate the local noise $\mathbf{n}_i^{\text{local}}$, which is independently sampled for each overlay measurement point j. This is done by sampling two bivariate normal distributions. The first bivariate normal sample is added to the second sample which is multiplied by the length of the ground truth measurement together with the defects from the global noise to create radius-dependent noise. This was done because, without this radius-dependent noise, large measurements in the ground truth would be relatively undisturbed by noise and ended up too smooth when compared to the real overlay measurements. Thus, we can write the local noise as

$$n_{i,j}^{\text{local}} = z_1 + ||y_{i,j} + n_{i,j}^{\text{global}}||_2 \cdot z_2, \qquad z_1 \sim N(\mathbf{0}, \sigma_1 \cdot I_2), z_2 \sim N(\mathbf{0}, \sigma_2 \cdot I_2), \tag{4.29}$$

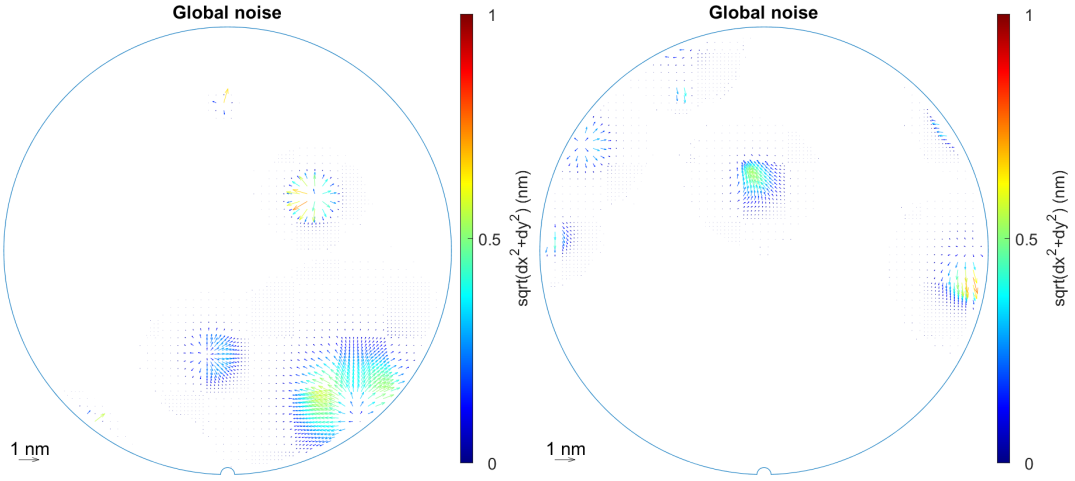


Figure 4.8: Two samples of synthetic global outliers $\mathbf{n}_i^{\text{global}}$. The defects in the overlay measurement process form global overlay errors, which affect multiple overlay measurement points. These samples were chosen because of their relatively high numbers of outliers to illustrate the different shapes and sizes.

where $||y_{i,j} + e_{i,j}^{\text{global}}||_2$ is the length of the ground truth overlay samples and global noise artifacts added together at each measurement location j.

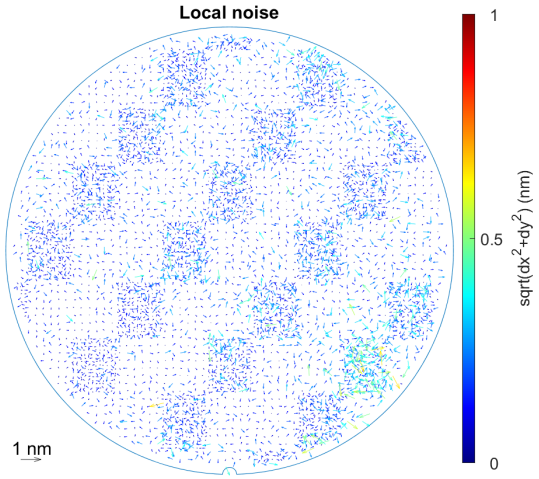


Figure 4.9: A sample $\mathbf{n}_i^{\text{local}}$ from the local noise. We can see that in the bottom right, the variance seems to be higher, caused by larger values of $||y_{i,j} + n_{i,j}^{\text{global}}||_2$ in this area. This local noise pattern was generated using the global noise depicted in the left illustration of Figure 4.8.

Resulting overlay

The resulting overlay sample is generated by adding up the global noise, local noise, and the ground truth overlay, so

$$\mathbf{x}_i = \mathbf{y}_i + \mathbf{n}_i^{\text{global}} + \mathbf{n}_i^{\text{local}}. \tag{4.30}$$

Figure 4.10 shows this process and the resulting overlay sample. Just like with the real overlay dataset, we then produce two samples $\tilde{\mathbf{x}}_i$ and $\hat{\mathbf{x}}_i$ from a single ground truth machine overlay state $\mathbf{y}_i$ by sampling two different global and local noise patterns

$$\tilde{\mathbf{x}}_i = \mathbf{y}_i + \tilde{\mathbf{n}}_i^{\text{global}} + \tilde{\mathbf{n}}_i^{\text{local}},$$
$$\hat{\mathbf{x}}_i = \mathbf{y}_i + \hat{\mathbf{n}}_i^{\text{global}} + \hat{\mathbf{n}}_i^{\text{local}}.$$

Two overlay patterns and the ground truth overlay pattern used to generate them can be seen in Figure 4.11.

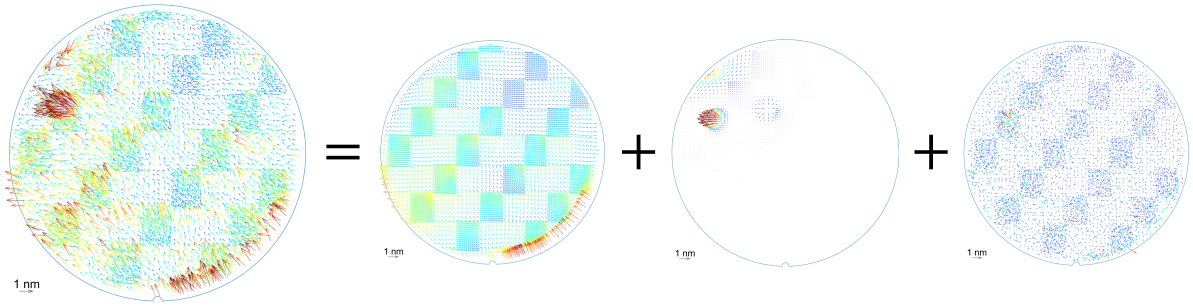


Figure 4.10: A sample from the synthetic overlay distribution is created by adding the global and local noise samples to the ground truth overlay pattern.

The synthetic dataset has been generated using $N = 3400$ samples. It is made up of the quadruples $(\mathbf{y}_i, \tilde{\mathbf{x}}_i, \hat{\mathbf{x}}_i, \mathbf{u}_i)$ with $1 \leq i \leq N$ where $\mathbf{u}_i$ contains all measurement locations on the wafer and in the synthetic dataset are identical for every i. We split this dataset into a training and validation set containing 2800 and 600 data quadruples, respectively.

As we have seen in section 4.1 the requirement for the Noise2Noise method with the L^2 loss to converge to $\mathbf{y}_i$ is that $\mathbb{E}\,\hat{\mathbf{x}}_i|\tilde{\mathbf{x}}_i = \mathbf{y}_i$. Because in the overlay sampling process $\mathbf{y}_i$ is sampled before $\tilde{\mathbf{x}}_i$, and furthermore, $\tilde{\mathbf{x}}_i$ and $\hat{\mathbf{x}}_i$ are independent conditioned on $\mathbf{y}_i$, we have $\mathbb{E}\hat{\mathbf{x}}_i|\tilde{\mathbf{x}}_i = \mathbb{E}\hat{\mathbf{x}}_i|\mathbf{y}_i$. In the synthetic overlay dataset, this requirement is thus satisfied if $\mathbb{E}\,\mathbf{n}_i^{\text{local}}|\mathbf{y}_i = \mathbf{0}$ and $\mathbb{E}\,\mathbf{n}_i^{\text{global}}|\mathbf{y}_i = \mathbb{E}\,\mathbf{n}_i^{\text{global}} = \mathbf{0}$.

The first statement is clearly true since $\mathbf{n}_i$ is generated using samples of mean zero bivariate normal distributions. Since the sampling method for $\mathbf{n}_i^{\text{global}}$ is rather complicated, there is no concise way of proving the defects are mean zeros. The generation method is, however, not made to be biased in a certain overlay direction; furthermore, empirically, we see that $\overline{\mathbf{n}_i^{\text{global}}} \to \mathbf{0}$. Thus, it seems safe to assume for the synthetic dataset, we have $\mathbb{E}\,\hat{\mathbf{x}}_i|\tilde{\mathbf{x}}_i = \mathbf{y}_i$.

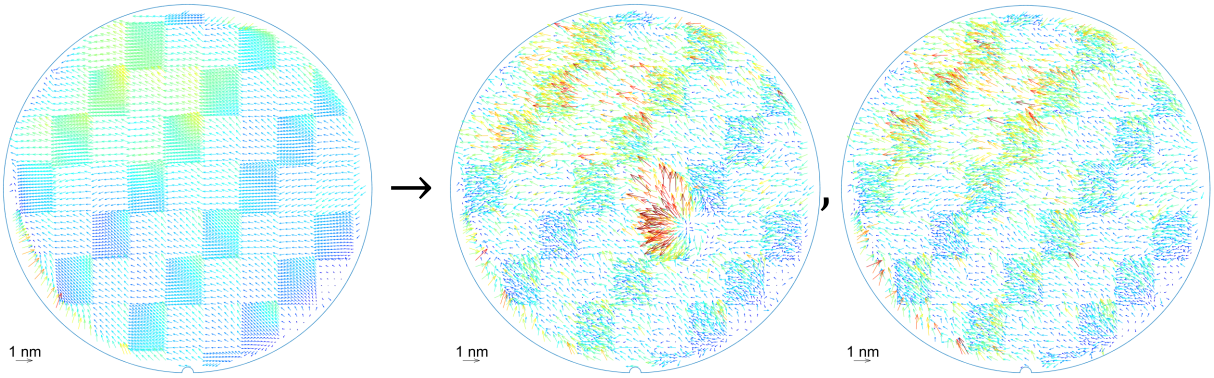


Figure 4.11: A single ground truth overlay pattern is used to create two samples of synthetic overlay. Here, the ground truth overlay represents the true machine overlay state, and the two synthetic samples represent the overlay measured on two test wafers exposed in the machine.

4.4. The real overlay dataset

The real overlay dataset is constructed from all overlay measurements exposed in the same measurement batch and measured on the same chuck. We can further double the dataset size by alternating the two noisy overlay measurements $\hat{\mathbf{x}}_i$ and $\tilde{\mathbf{x}}_i$, which has been shown to increase the performance of the resulting denoiser trained with the Noise2Noise loss [10]. Unlike the synthetic dataset, ground truth overlay machine states $\mathbf{y}_i$ are not expected to be independently distributed. A limited amount of scanners are used to expose the monitor wafer on which the overlay measurements are performed.

Since these scanners drift in overlay performance over time, a time correlation is expected. We will shuffle these measurements during training and then assume the resulting shuffled dataset is I.I.D. To validate that the denoising neural network generalizes to unseen scanners, we split the dataset into a training and validation dataset by scanner. So, overlay measurements in the validation set come from scanners not in the training set. The training and validation contain, respectively, roughly 80% and 20% of the monitor wafer measurements.

After generating the synthetic dataset we found that the real overlay dataset contains some very large outliers which are orders of magnitudes larger than those in the synthetic dataset and which, if left in, dominate the loss of equation 4.4. We found that the model, during training, learned to denoise these specific measurements in the training set without lowering the loss on the validation set, thus overfitting to these large overlay measurements. This is why we will mask these measurements in the loss on the real dataset. The contribution of a measurement point j to the loss is multiplied by zero if either

$$|\hat{x}_{i,j}^{dx}| > 4\hat{\sigma}_i^{dx}, \quad |\hat{x}_{i,j}^{dy}| > 4\hat{\sigma}_i^{dy}, \quad |\tilde{x}_{i,j}^{dx}| > 4\tilde{\sigma}_i^{dx}, \quad |\tilde{x}_{i,j}^{dy}| > 4\tilde{\sigma}_i^{dy}. \tag{4.31}$$

Here $\hat{\sigma}_i^{dx}$ is the standard deviation of all dx values of the overlay measurement $\hat{\mathbf{x}}_i$. Because of this masking technique, the neural network will not be trained to denoise these extremely large overlay values, but we can reject them automatically. To store which values are masked we will use the variables $m_{i,j}$ for $1 \leq i \leq N$ and $1 \leq i \leq M_i$, where

$$m_{i,j} = \begin{cases} 1 & \text{if } |\hat{x}_{i,j}^{dx}| > 4\hat{\sigma}_i^{dx} \text{ or } |\hat{x}_{i,j}^{dy}| > 4\hat{\sigma}_i^{dy} \text{ or } |\tilde{x}_{i,j}^{dx}| > 4\tilde{\sigma}_i^{dx} \text{ or } |\tilde{x}_{i,j}^{dy}| > 4\tilde{\sigma}_i^{dy} \\ 0 & \text{else.} \end{cases} \tag{4.32}$$

Masking the loss of these values is preferred over not using them as, when masked, the large values are still part of the input and can thus be interpreted by the model. While we see our masking method in no way as the only or perfect solution to our overfitting problem, it did reduce our overfitting problem on the real dataset.

4.5. Training our model

During training with the Noise2Noise method, we used the loss of equation 4.6. Which we now equivalently will write as the sum of the vectors wise losses

$$MSE(f_\theta(\hat{\mathbf{x}}), \tilde{\mathbf{x}}) := \frac{1}{\sum_{i=1}^{N} M_i} \sum_{i=1}^{N} \sum_{j=1}^{M_i} || \left(f_\theta(\hat{\mathbf{x}}_i)\right)_j - \tilde{\mathbf{x}}_j ||^2, \tag{4.33}$$

where with $\left(f_\theta(\hat{\mathbf{x}}_i)\right)_j$ we denote the jth measurement vector in the output $f_\theta(\hat{\mathbf{x}}_i)$. For the real dataset we will use the masked loss which we define as

$$MSE(f_\theta(\hat{\mathbf{x}}), \tilde{\mathbf{x}}) := \frac{1}{\sum_{i=1}^{N} \sum_{j=1}^{M_i} m_{i,j}} \sum_{i=1}^{N} \sum_{j=1}^{M_i} m_{i,j} \cdot || \left(f_\theta(\hat{\mathbf{x}}_i)\right)_j - \tilde{\mathbf{x}}_j ||^2, \tag{4.34}$$

where $m_{i,j}$ is 0 if the measurement is masked and otherwise 1. This loss is equivalent to equation 4.33 if the mask is all ones.

To minimize these losses, we use the Adam optimization method [29], which is similar to the batched stochastic gradient descent method of equation 2.8 in that it is a first-order gradient-based optimization method, but has shown to outperform batched SGD method in many example datasets. For this method, we used the learning rate equal to $1\mathrm{e}{-4}$ and a batch size of 8. This batch size was chosen to be as large as possible while not using more VRAM than our GPU allowed for. Our models are trained on a single NVIDIA V100 32GB GPU, where training time was about 1 day on the synthetic overlay dataset and multiple days on the real overlay dataset.

5

Results

The results of our model have been split up into the results of the model on the synthetic overlay dataset of section 4.3 and on the real overlay dataset of section 4.5. Since, for the synthetic dataset, the ground truth noise-free overlay state we want to approximate is known, we can test the effectiveness of our model's design decisions with multiple ablation studies on this dataset. To compare the model's performance of the two datasets, we prove a lower bound for the Noise2Noise loss, which we can then use to normalize the loss for both datasets. This result is then used to compare the model trained on the synthetic dataset, where we have a ground truth noise-free overlay state, to the model trained on the real dataset, where we do not have this ground truth noise-free overlay state.

5.1. Results on the synthetic overlay dataset

For the loss during training, we use the mean squared error $MSE(f_\theta(\hat{\mathbf{x}}), \tilde{\mathbf{x}})$ between the predictions of our model $f_\theta(\hat{\mathbf{x}}_i)$ and the noisy samples $\tilde{\mathbf{x}}_i$. In our synthetic dataset, we also have access to the ground truth overlay samples $\mathbf{y}_i$, so can also calculate the ground truth mean squared error $MSE(f_\theta(\hat{\mathbf{x}}), \tilde{\mathbf{y}})$ between the model predictions $f_\theta(\hat{\mathbf{x}}_i)$ and the ground truth overlay samples $\mathbf{y}_i$. To measure if our model generalizes well outside of the training set, to which we have fitted our model, we will only give these measures for the validation set, to which our model has not been fitted. The mean squared error number does not have a clear interpretation; thus, instead, we will use the R^2 value of the model on the validation set to illustrate the performance of our model. The R^2 value is the proportion of the variation of the residuals of your model and the variance of the variable you are predicting and is defined as

$$R^2 = 1 - \frac{\sum_i (y_i - f_i)^2}{\sum_i (y_i - \overline{y})^2}. \tag{5.1}$$

So if your function f perfectly predicts y_i for every value of i, the R^2 value is equal to 1, and if the function is just the mean of all the values of y_i, the R^2 is 0. Our model's output $f_\theta(\hat{\mathbf{x}})$ and ground truth samples $\mathbf{y}_i$, consist of M_i measurement points, and each measurement point has a dx and dy value we predict. Because on the synthetic dataset, the values for M_i are all the same, we can, for each prediction point, calculate the R^2 value and then take the average of all these values to end up with an average R^2 statistic. For our model on the validation set, this average R^2 statistic equals 0.977. From this, we can conclude that the model has learned to predict overlay measurements, on average, close to the ground truth overlay measurements without ever seeing these ground truth noise-free overlay measurements.

When we train our model, the model parameters are constantly updated, leading to changing model performance over iterations, which we can see in the loss graph of figure 5.2. Because the validation set is finite and the mean squared error is just an estimator of the expected risk, additional variance in the resulting score is introduced. During training, we save the model parameters every 5 epochs of roughly 300 total epochs. Whenever we calculate the R^2 metric, we pick the saved parameter set with the lowest loss value. When comparing models, we will mostly show the loss plots as they better illustrate the variance of the model's performance over iterations.

We can alternatively study our model output using example samples. In Figure 5.1, we see a representative sample $\hat{\mathbf{x}}_i$ from the validation set, our denoising function $f_\theta(\cdot)$ applied to this sample, the corresponding ground truth overlay value $\mathbf{y}_i$, and the difference between the prediction and the ground truth $\mathbf{y}_i - f_\theta(\hat{\mathbf{x}}_i)$. Our model removed both the normal independent noise as the two simulated outliers in the top and bottom right of the input $\hat{\mathbf{x}}_i$. When we look at the prediction error $\mathbf{y}_i - f_\theta(\hat{\mathbf{x}}_i)$, we see that the error is relatively small compared to the signal, but that this error is largest at the two locations of the two simulated outliers. This is somewhat expected as the model here has to estimate the signal convoluted by these outliers. Three more examples of the model denoising synthetic overlay measurements are available in appendix B.

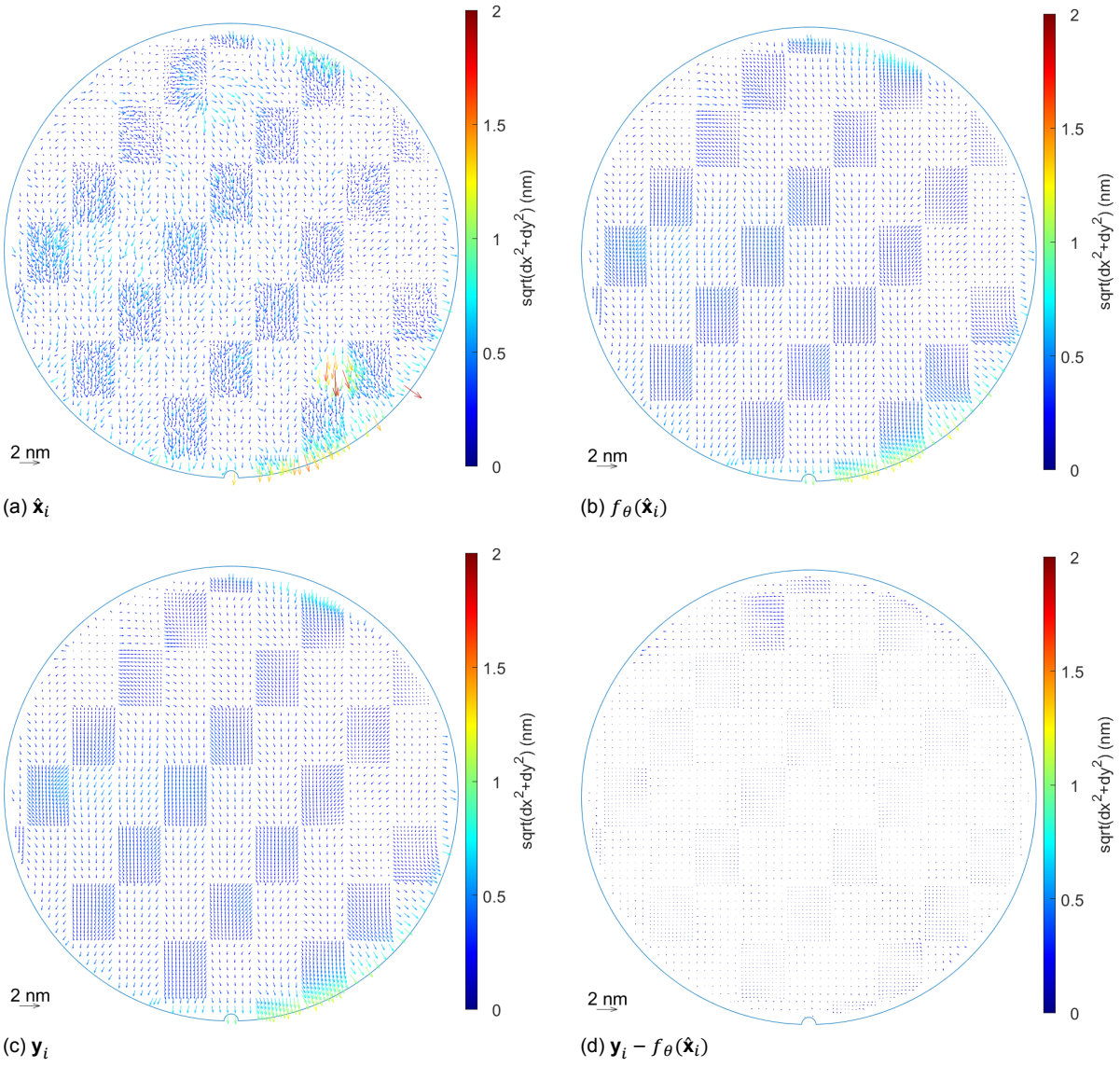


Figure 5.1: Our denoising model applied to a noisy overlay sample. Figure (a) shows the noisy input sample $\hat{\mathbf{x}}_i$, (b) shows the output of our model $f_\theta(\hat{\mathbf{x}}_i)$ on this sample, (c) shows the ground truth overlay $\mathbf{y}_i$ from which the sample $\hat{\mathbf{x}}_i$ was created, and (d) shows the error between the prediction and ground truth overlay defined as $\mathbf{y}_i - f_\theta(\hat{\mathbf{x}}_i)$. The sample used is from the validation set and is representative of our model's performance, where the error is often largest at the locations of the artificial outliers.

Comparing Noise2Noise training with supervised training

We trained our model using the Noise2Noise loss of equation 4.6 so, by minimizing $MSE(f_\theta(\hat{\mathbf{x}}), \tilde{\mathbf{x}})$ on the training set. For our synthetic dataset, we have access to the ground truth noise-free samples $\mathbf{y}_i$ and can alternatively train our model by minimizing the ground truth mean squared error $MSE(f_\theta(\hat{\mathbf{x}}), \mathbf{y})$ of equation 4.7 on the training set. The performance of two models during training with these two

different losses, which we will call Noise2Noise training and supervised training, can be seen in figure 5.2. We can see that both models converge to a loss relatively close to 0, but the model trained using the supervised method is consistently better than the one trained using the Noise2Noise method. The average R^2 value for the model trained with the Noise2Noise loss is, as stated before, equal to 0.977. The average R^2 value is slightly higher at 0.981 for the model trained with the supervised method. We see two possible explanations for this slightly worse performance of the model trained with the Noise2Noise method.

The first explanation is that it could be that the conditions of the Noise2Noise method discussed in section 4.1 are not met because $\mathbb{E}\tilde{\mathbf{x}}|\mathbf{y}_i \neq \mathbf{y}_i$ and thus we converge to $\mathbb{E}\tilde{\mathbf{x}}|\mathbf{y}_i$ instead of $\mathbf{y}_i$. We discussed this condition for synthetic data in section 4.3, reasoned why we thought it would be true, and observed that it seemed to be true empirically because the average of $\tilde{\mathbf{x}}|\mathbf{y}_i$ converged to $\mathbf{y}_i$, but have not proven equality.

The second explanation, which we think is more likely, is that we do not converge to the supervised solution because this is only guaranteed for infinite data. The authors of the Noise2Noise method show that the expected squared difference between the solution of the Noise2Noise method converges to 0 if $N \to \infty$ but that this value is non-zero if $N < \infty$. We thus expect this performance gap to shrink for a larger dataset.

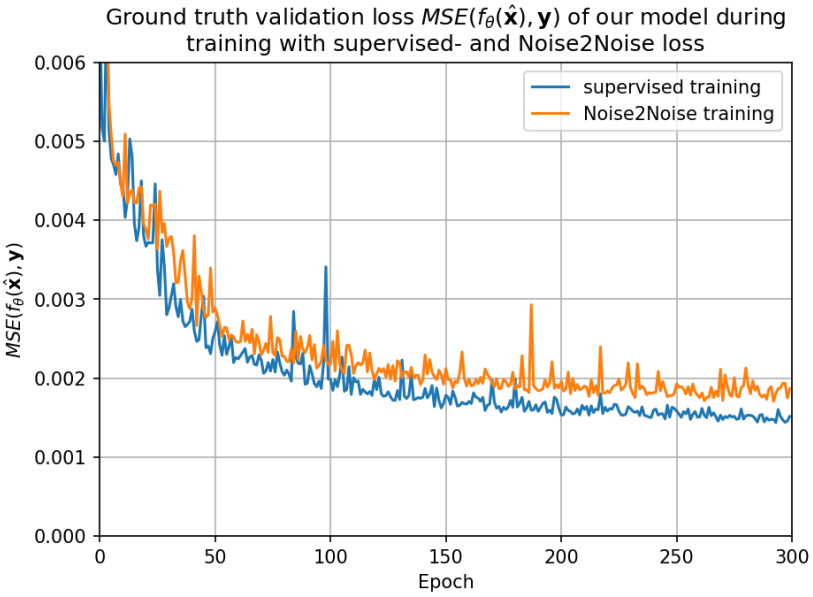


Figure 5.2: The loss curve during training of our model. Here we compare two different training methods, namely one model trained on the Noise2Noise loss using the two noisy samples pairs $\hat{\mathbf{x}}_i, \tilde{\mathbf{x}}_i$, and one trained using a supervised loss using the noisy-clean measurement pairs $\hat{\mathbf{x}}_i, \mathbf{y}_i$.

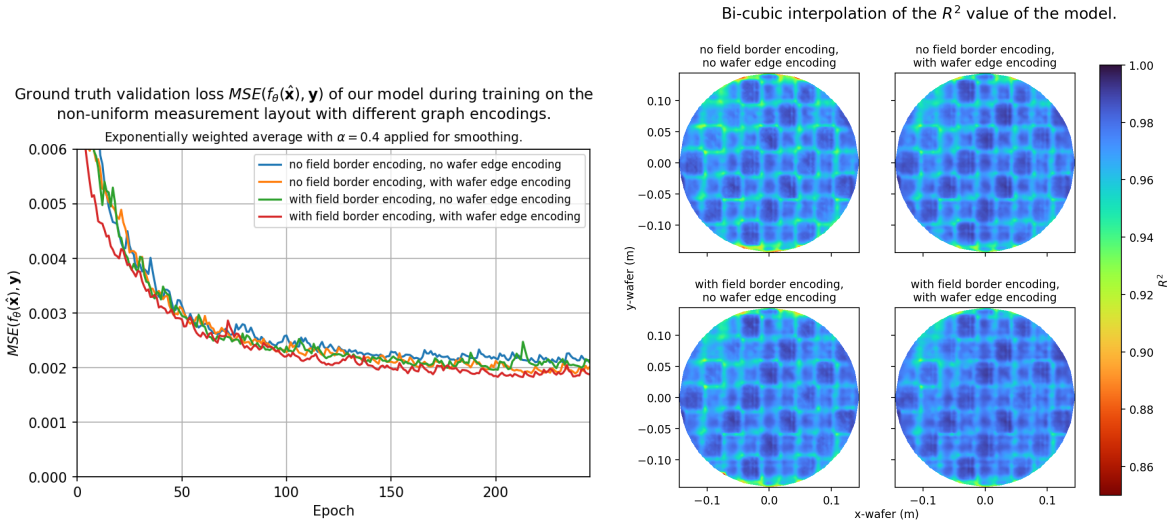
The effect of the a priori physics-based information

In section 4.2, we discussed how we encoded the edge- and vertex feature vectors, which form the input for our message-passing neural network. To the encodings used in the MeshGraphNet model, which inspired our model, we added two additional encodings that incorporate our a priori physics-based assumptions of the overlay data. The wafer edge encoding of equation 4.15 added a one-hot encoding to the vertex feature vectors equal to one for vertices close to the edge where we expect higher variance measurements. The field border encoding of equation 4.16 added a one-hot encoding to the edge feature vectors equal to one if that edge crossed a field border.

To test if these encodings achieve the desired effect and increase the accuracy of our model, we trained our model in four different configurations with the two encoding types enabled or disabled. Figure 5.3a shows the ground truth mean squared error of these four different model configurations during training. While the model with both of the two encodings is on average the best performing, and the model without any of the two encodings is on average the worst performing, these results are not conclusive as even after smoothing is applied, the differences are about as big as the noise levels in

the loss. While the extra encodings do increase the length of the feature vectors and thus the number of parameters in the MLPs of the encoding step of algorithm 2, the increase to the total parameter count is 0.006% at most.

Since the encodings are applied to specific locations on the wafer, we would expect accuracy to increase at these specific locations when the encodings are added to the model input. To investigate if this is the case, we calculated the R^2 score for all the nodes and averaged the value for the dx and dy prediction to get a score at every node. Using bilinear interpolation, we then interpolated these values to create Figure 5.3b. In this figure, we see that the error at some of the more isolated nodes along the top and bottom of the edge of the mesh seems to be reduced by the wafer edge encoding. There further seems to be a slight reduction in the error on the field border for the models with field border encoding enabled. These results are, however, again not conclusive.

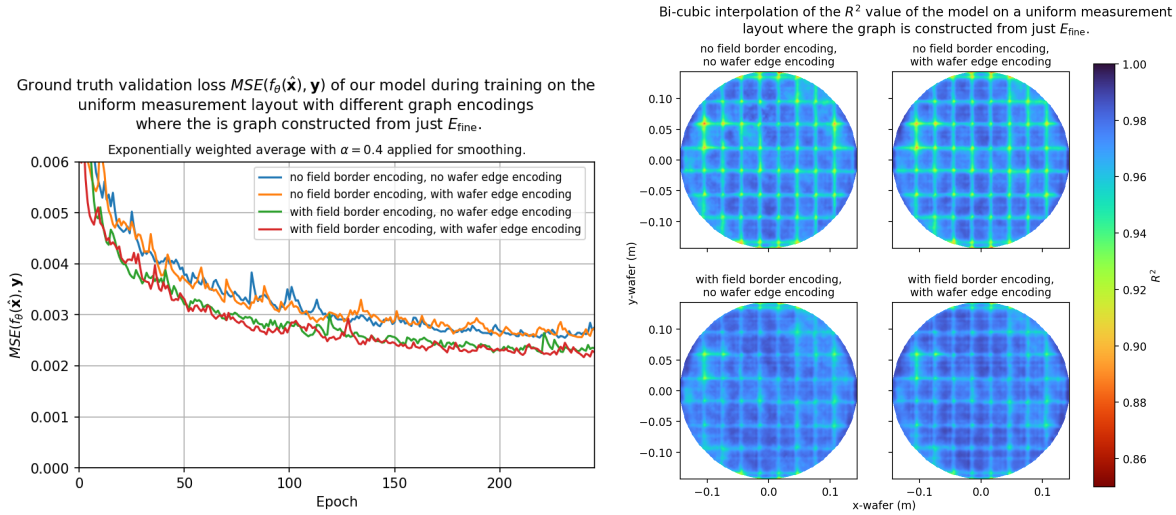


(a) (b)

Figure 5.3: The performance of four models trained on the synthetic dataset with and without the wafer border- and wafer edge encoding visualized by the ground truth validation loss (a), and a bi-cubic interpolation of the R^2 score at each measurement point on the validation set (b).

We hypothesized that removing the field border encodings did not affect the model's accuracy significantly because the model could retrieve the field borders from alternative information in the computation graph and the feature vectors. Our measurement layout, visible in Figure 1.4a, with its alternation densely- and sparsely sampled fields, together with how the edge sets E_i^{medium} and E_i^{coarse} of section 4.2 are constructed using the field layout, could enable the model to retrieve the field border information independently of the field border encoding. To test this hypothesis, we created a synthetic dataset following the procedure of section 4.3 but now with a uniform, evenly spaced measurement layout and fitted to this data, our model that uses just the edges of E_i^{fine} for its computation graph. The results with this setup can be seen in Figure 5.4. Figure 5.4a shows, as we expected, a larger loss reduction for the model with field border encodings, and Figure 5.4b affirms the effect further as the models with field border encodings have a clear lower error near the field borders. The error reduction along the edge of the wafer we attributed to the wafer edge encoding seems to have disappeared with this setup. The model could have learned to recognize the wafer edge as the number of neighbors for each vertex is smaller here, and this is potentially more consistently the case for the more uniform graph of the uniform measurement layout.

We could conclude from this experiment that the a priori physics-based encodings are not as necessary in our model because the model has alternative ways of retrieving the encoded information. The encodings do, however, seem to increase accuracy somewhat. Still, more importantly, for our real dataset, the computation graph is not the same for each measurement pair in the real data as some measurements may be missing in a sample. We would thus prefer our model to fit our encodings instead of the computation graph, which is no longer constant on the real data. Because of this, we have decided to include the encodings in the model.



(a) (b)

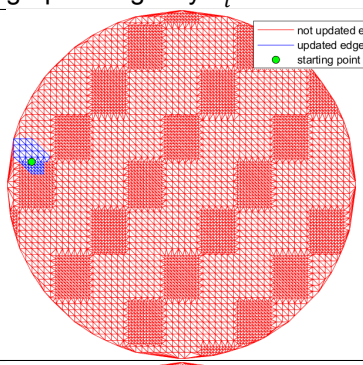
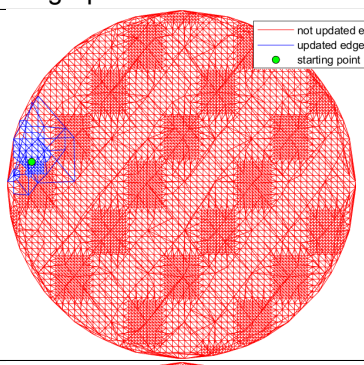
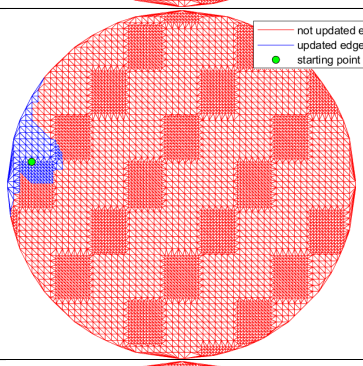
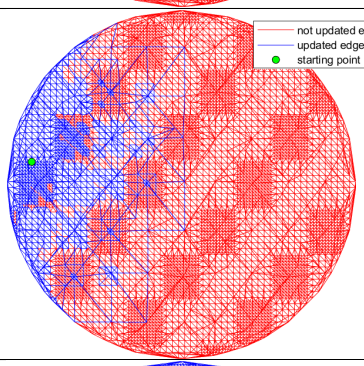
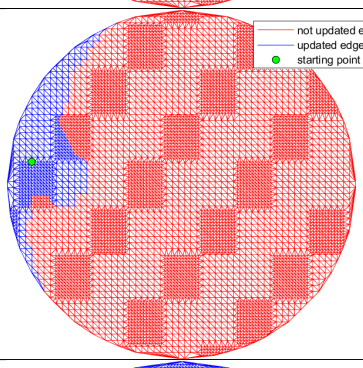
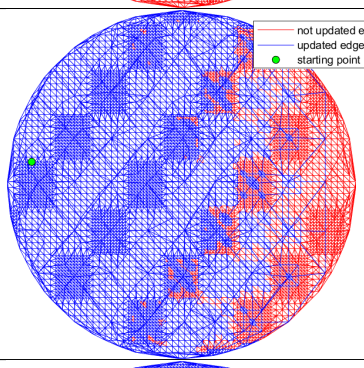
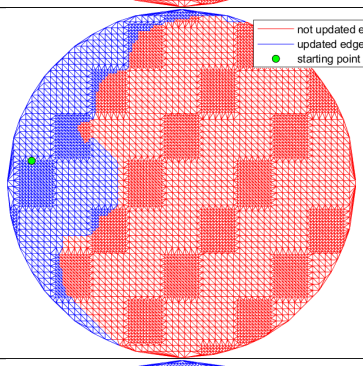
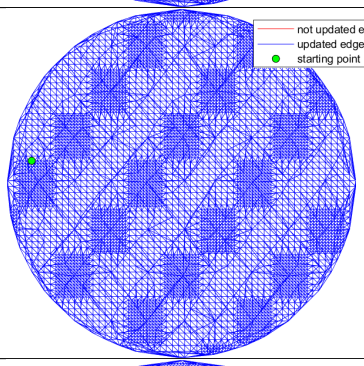
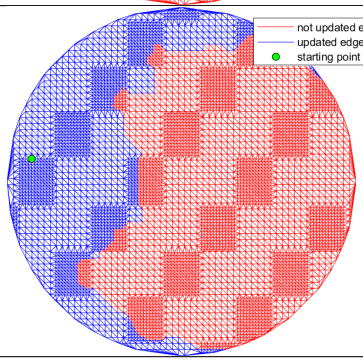
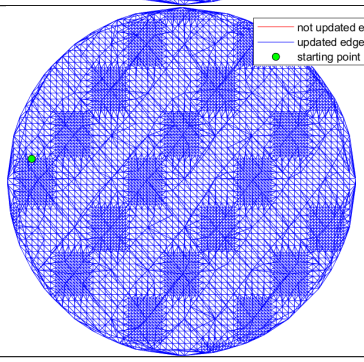
Figure 5.4: The same graphs as of figure 5.3 but now with a synthetic overlay dataset with a uniform evenly spaced measurement layout and fitted to this data a model that uses just the edges of E_i^{fine} for its computation graph

The effects of our multiscale graph

The MeshgraphNet model [37], which inspired our model, used as its computation graph the mesh used for the finite element method that created the target physics simulation. Because during a single time step, information can only flow from a node to its neighbors, simple meshes with ”short” edges such as the one used in the MeshGraphNet model can limit information spread and, in turn, decrease a model’s accuracy on a task where wider context is needed [17]. This problem is especially relevant for meshes with a high node count such as ours. We will test if our multiscale computation graph, which is a combination of three different Delauney meshes with different average edge lengths as defined by equations 4.8-4.14, does indeed increase the model accuracy compared to the model with just the single Delauney mesh with edges E^{fine} as defined in equation 4.10. Because our multiscale graph should reduce the problem of limited information spread, we will compare the two graphs with different numbers of message-passing steps N^{mps}.

We will first investigate the diameters of the graphs. If we want every node to be theoretically able to exchange information with every other node in the graph, the number of message-passing steps should be at least equal to the graph diameter. The diameter of the graph constructed with just the edges E_i^{fine} is 54 and the diameter of our multiscale graph is 18, a significant reduction. The difference in information transport can also be seen in Table 5.1. These illustrations show how many nodes the green starting node can theoretically exchange information with, using the two different graphs for, a given number of message-passing steps. We see that information can travel way faster along our multiscale graph. When using 15 message-passing steps, the information from the starting vertex feature vector can spread to all nodes of the multiscale graph, where this happened to less than a third of the nodes of the graph constructed from the edges E_i^{fine}. On the left of the table, we further see that the speed of information travel is faster along the edges of the wafer and slower in the densely sampled fields because of differing edge lengths in the mesh of the graph constructed from E_i^{fine}. Information can travel more uniformly on the multiscale graph.

Table 5.1: An illustration of how fast information can transport along the graph for the graph using just E_i for its edges, and along our multiscale graph defined by equations 4.8-4.14. The spread of information is shown for the same starting vertex and a different number of message-passing steps N^{mps}.

N^{mps}	graph using only E_i^{fine}	full graph
3		
6		
10		
15		
20		

Because our multiscale graph considerably increases the speed of information travel, we expect this to lead to an increased accuracy of our model. To test if our multiscale graph does indeed increase the accuracy of the model, we trained our model with the configurations of Table 5.1 for which the smoothed ground truth validation loss can be seen in Figure 5.5. We can see that the model with the multiscale graph always outperformed the version that used the graph with edges E_i^{fine} with the same number of message-passing steps and thus the same parameter count. The proportion of nodes reached in Table 5.1 seems to give a good indication of the model's accuracy. For example, 15 message-passing steps with the fine graph reach about the same number of nodes as 6 message-passing steps with the multiscale graph, and both model configurations give similar accuracy.

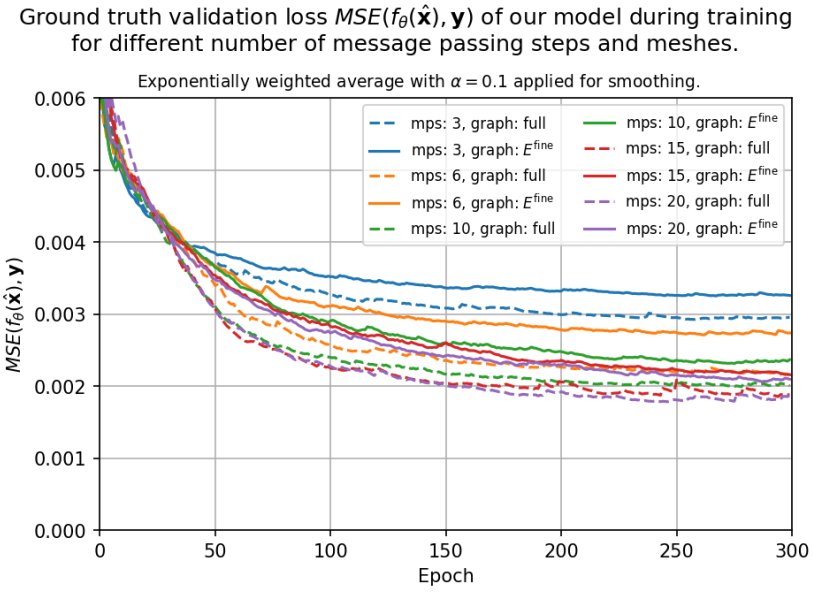


Figure 5.5: The ground truth validation loss curves during training of our model configurations. The model configurations are the same as the number of messages passing steps and the two graph types used in Table 5.1. Smoothing was applied to the loss curves, making the results easier to interpret.

When we investigated the error for the model using 6 message passing steps trained using the graph with edges E_i^{fine} and the multiscale graph with edges E_i, we noticed that the biggest error reduction was at the locations of the simulated continuations. This difference can be seen in Figure 5.6, which shows the error for both models on a single representative sample. Since the number of parameters is identical between the models, the ability to better remove simulated contaminations must be the results of a larger context window made possible by the multiscale graph.

We conclude that our multiscale graph increases the speed of information transport and thereby allows the use only 20 message-passing steps for a sufficiently accurate denoiser, thus, in turn, reducing the necessary parameter count and increasing training and inference speed. While the extra edges in the multiscale graph lead to more calculations performed during training and inference and thereby a 10% speed reduction, the lower message-passing number more than compensates for this speed reduction. Using the multiscale graph does not increase the parameter count.

Studying measures to prevent overfitting

When training our model in the early stages of the project on the synthetic dataset, after about 60 epochs, the training- and validation loss started to diverge, leading to reduced performance on the validation set, as can be seen in the blue line of Figure 5.7. A clear example of overfitting. One feature of our model that should prevent overfitting is that the model is translation invariant. If we translate the model input in space, the model's output will not change. Our model is not invariant or equivariant for rotations. When the input wafer is rotated in space, the relative locations $u_{i,u} - u_{i,v}$ in the edge feature vectors $\mathbf{e}_{u,v}$ change. Since the vectors that form the overlay measurements represent shifts in the x and y directions on the wafer, we also rotate the input and target overlay vectors when we rotate the wafer. For the rotated measurement locations $\mathbf{u}_i^*$ and the rotated measurements $\hat{\mathbf{x}}_i^*$ where the values

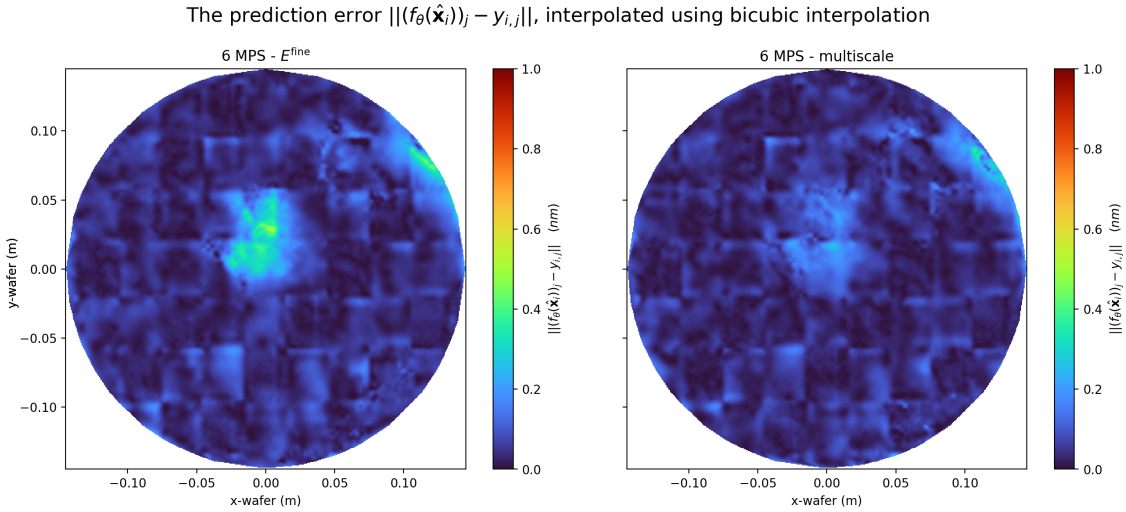


Figure 5.6: The prediction error $||(f_\theta(\hat{\mathbf{x}}_i))_j - y_{i,j}||$ on a single sample $\hat{\mathbf{x}}_i$, for our model trained using the graphs with the edges E_i^{fine} and our multiscale graph with edges E_i. This plot was generated using the same sample $\hat{\mathbf{x}}_i$ as was used for Figure 5.1.

have been transformed by some orthogonal rotation matrix R s.t. $u^*_{i,j} = Ru_{i,j}$ and $\hat{x}^*_{i,j} = R\hat{x}_{i,j}$, we, in general, do not have rotation equivariance

$$\left(f_\theta(\hat{\mathbf{x}}_i^*, \mathbf{u}_i^*)\right)_j = R\left(f_\theta(\hat{\mathbf{x}}_i, \mathbf{u}_i)\right)_j, \tag{5.2}$$

where $(\cdot)_j$ represent the jth measurement and the fact that the measurement locations $\mathbf{u}_i^*$ and $\mathbf{u}_i$ are also an input to our model is emphasized by writing them as inputs to f_θ.

Thus, we saw an opportunity to reduce our overfitting problem by using this observation. One option was to make our model rotation invariant for the measurement locations $\hat{\mathbf{u}}_i$, which we can achieve by only keeping the distance between measurement point $||u_{i,u} - u_{i,v}||$ in the encoding of the edge feature vectors $\mathbf{e}_{u,v}$ and discarding the relative location vector $u_{i,u} - u_{i,v}$. If we make this change, we guarantee

$$\left(f_\theta(\hat{\mathbf{x}}_i, \mathbf{u}_i^*)\right)_j = \left(f_\theta(\hat{\mathbf{x}}_i, \mathbf{u}_i)\right)_j, \tag{5.3}$$

but not equality of equation 5.2 for rotation equivariance. This could still prevent some overfitting as the amount of input data is reduced.

The second option we explored is to use data augmentation during training as is described in section 4.2, where we rotate both the locations and the measurements of the input and the target during training. Since the fit to our augmented data will not be perfect, we can still not guarantee equality of equation 5.2. We will, however, increase the number of examples of our training set, which could lead to improved denoising performance.

To test if these two approaches reduced the overfitting of our model, we implemented them and trained them on the synthetic dataset. Figure 5.7 shows the resulting validation loss from these experiments. The first thing we can see is that adding the data augmentation step to our model significantly improved the performance of the resulting model. Our second observation is that only using the distance between nodes as input to the model gave almost the same performance as the full model. Thus, it seems that the model already ignored the values $u_{i,u} - u_{i,v}$ in the edge feature vectors, and removing them did not change the model performance. This could result from how the synthetic dataset was made and may not generalize to the full dataset, but this has not yet been tested. If we use both the data augmentation step and remove $u_{i,u} - u_{i,v}$ from $\mathbf{e}_{u,v}$, the measurement location information given to the model is not augmented as $||u_{i,u} - u_{i,v}|| = ||Ru_{i,u} - Ru_{i,v}||$, and only the measurements are rotated.

This left us with a choice of which model we would like to use. One could argue that removing $u_{i,u} - u_{i,v}$ from the edge feature vectors slightly reduces the parameter count, reduces the input information, and does not decrease the model's performance, so it should reduce overfitting. We, however, suspect that on the real dataset, this location information is more important as the patterns in this dataset are

more complex. Thus, we left it in the model. The data augmentation step clearly increased model performance, so it was included in the model.

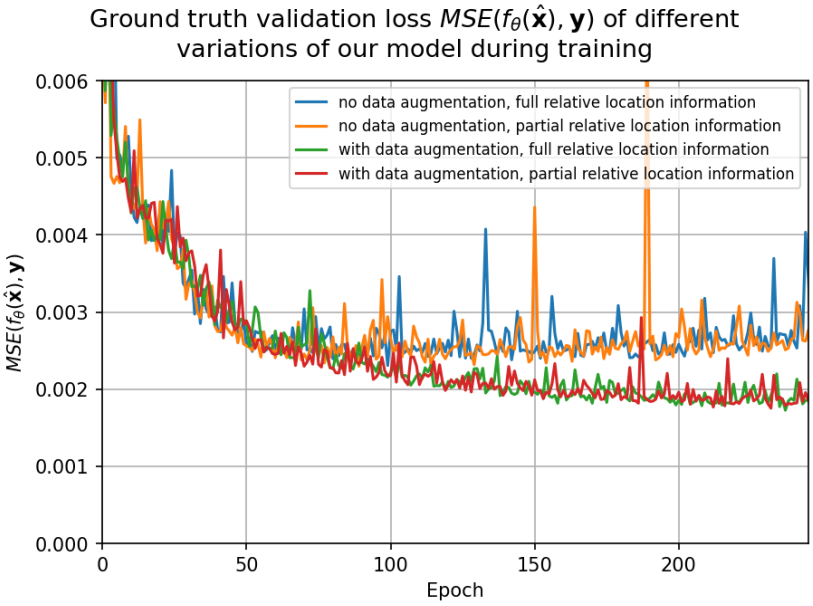


Figure 5.7: The validation L^2 loss during training of four different variations of our model using two strategies aimed at reducing overfitting in our model. The first strategy is to introduce data augmentation as defined in section 4.2. The second strategy is to only include the distance between measurement locations and not the relative location vectors in the model input.

Predicting noise instead of signal

During training with the Noise2Noise method the loss is calculated between $f_\theta(\hat{\mathbf{x}}_i)$ and $\tilde{\mathbf{x}}_i$, where $\hat{\mathbf{x}}_i$ and $\tilde{\mathbf{x}}_i$ are from the same distribution with mean $\mathbf{y}_i$. Since $\hat{\mathbf{x}}_i$ and $\tilde{\mathbf{x}}_i$ are from the same distribution we would expect $\hat{\mathbf{x}}_i$ to be more similar to $\hat{\mathbf{x}}_i$ than the the random initialization of $f_\theta(\hat{\mathbf{x}}_i)$ is to $\tilde{\mathbf{x}}_i$. This observation led us to investigate an alternative model we will denote as $g_\theta(\cdot)$, where

$$g_\theta(\hat{\mathbf{x}}_i) := \hat{\mathbf{x}}_i + f_\theta(\hat{\mathbf{x}}_i), \tag{5.4}$$

and $f_\theta(\cdot)$ is the original model defined in chapter 4. When fitting $g_\theta(\cdot)$ we achieve a low loss if $f_\theta(\hat{\mathbf{x}}_i) \approx \mathbf{y}_i - \hat{\mathbf{x}}_i$ instead of when $f_\theta(\hat{\mathbf{x}}_i) \approx \mathbf{y}_i$. Thus, our model is then trained to predict the overlay noise $\mathbf{y}_i - \hat{\mathbf{x}}_i$ instead of the overlay signal $\mathbf{y}_i$.

During initial training, our model using $g_\theta(\cdot)$ failed to converge, unlike $f_\theta(\cdot)$. This was due to a single wafer measurement $\hat{\mathbf{x}}_i$. An error in the synthetic data generation led to an overlay measurement 1000 times larger than others, affecting. This skewed loss dominated the training, causing $g_\theta(\cdot)$ to focus on this error rather than generalizing well to the validation set. Thus, $g_\theta(\cdot)$ is more sensitive to large input overlay values compared to $f_\theta(\cdot)$. After we removed the wafer measurement pair with the extreme value in the synthetic dataset, the alternative function $g_\theta(\cdot)$ did converge during training as can be seen in the loss curve of Figure 5.8. We can see that the performance of $g_\theta(\cdot)$ and $f_\theta(\cdot)$ is virtually identical on the synthetic overlay dataset. On the real overlay dataset, the results of this experiment are quite different as there $g_\theta(\cdot)$ is considerably faster to train as illustrated in section 5.2.

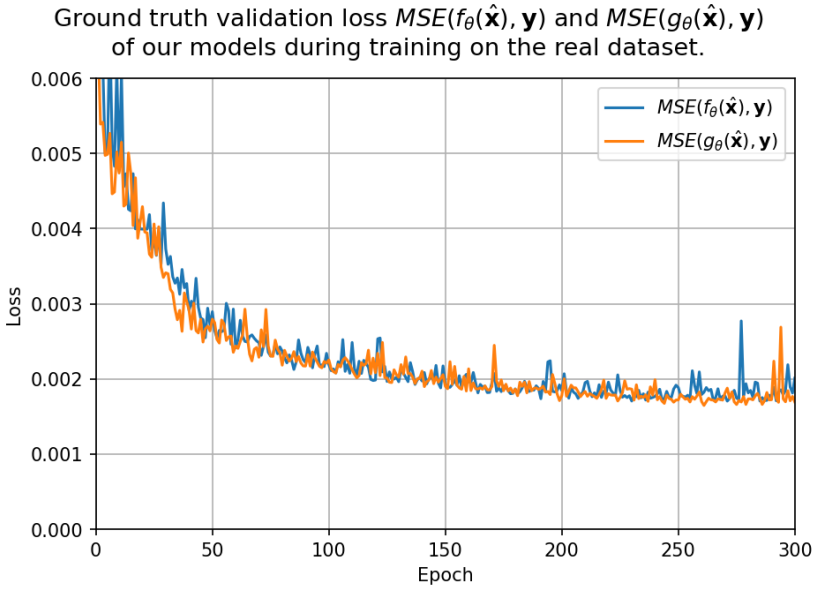


Figure 5.8: The validation L^2 loss during training of our model based on $f_\theta(\cdot)$ as defined in chapter 4 and the alternative function $g_\theta(\cdot)$ defined by equation 5.4. Where these models are trained on our synthetic overlay dataset

The orthogonality of the Noise2Noise loss

When we trained our models we noticed that the Noise2Noise loss $MSE(f_\theta(\hat{\mathbf{x}}), \tilde{\mathbf{x}})$ always seemed to be equal to the ground truth loss $MSE(f_\theta(\hat{\mathbf{x}}), \tilde{\mathbf{y}})$ plus some constant, as can be seen on the model validation losses of Figure 5.9. To mathematically investigate this phenomenon, we wondered if we could rewrite the expectation of the Noise2Noise L^2 loss to some constant plus the ground truth L^2 loss. This leads to the following calculation where we use the vectors in $\mathbb{R}^{2M_i}$ instead of the tensors in $\mathbb{R}^{2\times M_i}$ for the overlay values.

$$\begin{aligned}
\mathbb{E}_{\hat{\mathbf{x}},\tilde{\mathbf{x}}}||f_\theta(\hat{\mathbf{x}}) - \tilde{\mathbf{x}}||^2 &= \mathbb{E}_{\hat{\mathbf{x}},\tilde{\mathbf{x}},\mathbf{y}}||(f_\theta(\hat{\mathbf{x}}) - \mathbf{y} - (\tilde{\mathbf{x}} - \mathbf{y})||^2 && (5.5)\\
&= \mathbb{E}_{\hat{\mathbf{x}},\mathbf{y}}||f_\theta(\hat{\mathbf{x}}) - \mathbf{y}||^2 + \mathbb{E}_{\tilde{\mathbf{x}},\mathbf{y}}||\tilde{\mathbf{x}} - \mathbf{y}||^2 - 2\mathbb{E}_{\hat{\mathbf{x}},\tilde{\mathbf{x}},\mathbf{y}}\langle f_\theta(\hat{\mathbf{x}}) - \mathbf{y}, \tilde{\mathbf{x}} - \mathbf{y}\rangle && (5.6)\\
&= \mathbb{E}_{\hat{\mathbf{x}},\mathbf{y}}||f_\theta(\hat{\mathbf{x}}) - \mathbf{y}||^2 + \mathbb{E}_{\tilde{\mathbf{x}},\mathbf{y}}||\tilde{\mathbf{x}} - \mathbf{y}||^2 - 2\mathbb{E}_{\hat{\mathbf{x}},\mathbf{y}}\left[\langle f_\theta(\hat{\mathbf{x}}) - \mathbf{y}, \mathbb{E}_{\tilde{\mathbf{x}}|\mathbf{y}}[\tilde{\mathbf{x}}] - \mathbf{y}\rangle\right] && (5.7)\\
&= \mathbb{E}_{\hat{\mathbf{x}},\mathbf{y}}||f_\theta(\hat{\mathbf{x}}) - \mathbf{y}||^2 + \mathbb{E}_{\tilde{\mathbf{x}},\mathbf{y}}||\tilde{\mathbf{x}} - \mathbf{y}||^2 - 2\mathbb{E}_{\hat{\mathbf{x}},\mathbf{y}}\left[\langle f_\theta(\hat{\mathbf{x}}) - \mathbf{y}, \mathbf{y} - \mathbf{y}\rangle\right] && (5.8)\\
&= \mathbb{E}_{\hat{\mathbf{x}},\mathbf{y}}||f_\theta(\hat{\mathbf{x}}) - \mathbf{y}||^2 + \mathbb{E}_{\tilde{\mathbf{x}},\mathbf{y}}||\tilde{\mathbf{x}} - \mathbf{y}||^2 && (5.9)
\end{aligned}$$

Here we used the law of total expectation and the assumptions on the sampling process made in section 1.2, namely that conditioning $\tilde{\mathbf{x}}$ on $\mathbf{y}$ and $\hat{\mathbf{x}}$ is the same as conditioning $\tilde{\mathbf{x}}$ on just $\mathbf{y}$, and that $\mathbb{E}_{\tilde{\mathbf{x}}|\mathbf{y}}\tilde{\mathbf{x}} = \mathbf{y}$. The resulting equation shows that the Noise2Noise loss L^2 is indeed separated from the ground truth L^2 loss by a constant. It is also an alternative proof for the fact that minimizing the Noise2Noise L^2 loss with respect to θ is identical to minimizing the ground truth L^2 loss for infinite data given that $\mathbb{E}_{\tilde{\mathbf{x}}|\mathbf{y}}\tilde{\mathbf{x}} = \mathbf{y}$. We proved this without the need for M-estimators [22] as is used in the proof of the original Noise2Noise paper [34]. This proof, however, only works for the L^2 loss, whereas the proof in the Noise2Noise paper works for all losses with an M-estimator. The above calculation further shows the theoretical minimum of the Noise2Noise L^2 loss. When our sampling assumptions are met, we have for any parameter set θ:

$$\mathbb{E}_{\hat{\mathbf{x}},\tilde{\mathbf{x}}}||f_\theta(\hat{\mathbf{x}}) - \tilde{\mathbf{x}}||^2 \geq \mathbb{E}_{\tilde{\mathbf{x}},\mathbf{y}}||\tilde{\mathbf{x}} - \mathbf{y}||^2. \tag{5.10}$$

This inequality and the equality above still use the variable $\mathbf{y}$, which is not observable in the real overlay dataset. Reusing the above calculations but now for $\hat{\mathbf{x}}$ instead of $f_\theta(\hat{\mathbf{x}})$ and using the fact that $\hat{\mathbf{x}}$ and $\tilde{\mathbf{x}}$ are I.I.D. conditional on $\mathbf{y}$ we get:

$$\mathbb{E}_{\hat{\mathbf{x}},\tilde{\mathbf{x}}}||\hat{\mathbf{x}}-\tilde{\mathbf{x}}||^2 = \mathbb{E}_{\hat{\mathbf{x}},\tilde{\mathbf{x}},\mathbf{y}}||(\hat{\mathbf{x}}-\mathbf{y}-(\tilde{\mathbf{x}}-\mathbf{y})||^2 \tag{5.11}$$

$$= \mathbb{E}_{\hat{\mathbf{x}},\mathbf{y}}||\hat{\mathbf{x}}-\mathbf{y}||^2 + \mathbb{E}_{\tilde{\mathbf{x}},\mathbf{y}}||\tilde{\mathbf{x}}-\mathbf{y}||^2 - 2\mathbb{E}_{\hat{\mathbf{x}},\tilde{\mathbf{x}},\mathbf{y}}\langle\hat{\mathbf{x}}-\mathbf{y},\tilde{\mathbf{x}}-\mathbf{y}\rangle \tag{5.12}$$

$$= \mathbb{E}_{\mathbf{y}}\left[\mathbb{E}_{\hat{\mathbf{x}}|\mathbf{y}}||\hat{\mathbf{x}}-\mathbf{y}||^2 + \mathbb{E}_{\tilde{\mathbf{x}}|\mathbf{y}}||\tilde{\mathbf{x}}-\mathbf{y}||^2\right] - 2\mathbb{E}_{\hat{\mathbf{x}},\mathbf{y}}\left[\langle\hat{\mathbf{x}}-\mathbf{y},\mathbb{E}_{\tilde{\mathbf{x}}|\mathbf{y}}[\tilde{\mathbf{x}}]-\mathbf{y}\rangle\right] \tag{5.13}$$

$$= \mathbb{E}_{\mathbf{y}}\left[2\mathbb{E}_{\tilde{\mathbf{x}}|\mathbf{y}}||\tilde{\mathbf{x}}-\mathbf{y}||^2\right] - 2\mathbb{E}_{\hat{\mathbf{x}},\mathbf{y}}\left[\langle\tilde{\mathbf{x}}-\mathbf{y},\mathbf{y}-\mathbf{y}\rangle\right] \tag{5.14}$$

$$= 2\mathbb{E}_{\tilde{\mathbf{x}},\mathbf{y}}||\tilde{\mathbf{x}}-\mathbf{y}||^2 \tag{5.15}$$

Thus, this allows us to estimate the ground truth L^2 loss without any access to the ground truth random variable, namely as

$$\mathbb{E}_{\hat{\mathbf{x}},\mathbf{y}}||f_\theta(\hat{\mathbf{x}})-\mathbf{y}||^2 = \mathbb{E}_{\hat{\mathbf{x}},\tilde{\mathbf{x}}}||f_\theta(\hat{\mathbf{x}})-\tilde{\mathbf{x}}||^2 - \frac{1}{2}\mathbb{E}_{\hat{\mathbf{x}},\tilde{\mathbf{x}}}||\hat{\mathbf{x}}-\tilde{\mathbf{x}}||^2. \tag{5.16}$$

And we can now also rewrite our inequality for the Noise2Noise L^2 loss without the random variable $\mathbf{y}$ as

$$\mathbb{E}_{\hat{\mathbf{x}},\tilde{\mathbf{x}}}||f_\theta(\hat{\mathbf{x}})-\tilde{\mathbf{x}}||^2 \geq \frac{1}{2}\mathbb{E}_{\hat{\mathbf{x}},\tilde{\mathbf{x}}}||\hat{\mathbf{x}}-\tilde{\mathbf{x}}||^2. \tag{5.17}$$

As the mean squared error is an unbiased estimator for the expectation of the L^2 norm between two random variables, we can conclude from equation 5.16 that $MSE(f_\theta(\hat{\mathbf{x}}),\tilde{\mathbf{x}}) - \frac{1}{2}MSE(\hat{\mathbf{x}},\tilde{\mathbf{x}})$ is an unbiased estimator the ground truth L^2 loss $MSE(f_\theta(\hat{\mathbf{x}}),\mathbf{y})$. In figure 5.9, we can see that our estimator follows the ground truth L^2 loss very closely while training our model on the synthetic dataset. This estimator will thus allow us to estimate the ground truth L^2 loss for our real overlay dataset with the assumptions of section 1.2, which we made for both the synthetic and real dataset, without any ground truth overlay for this dataset.

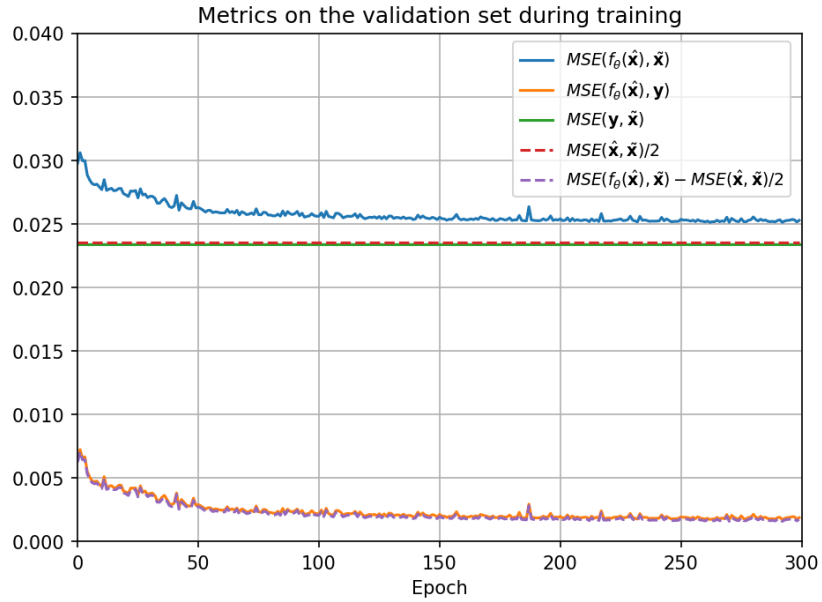


Figure 5.9: To verify our calculations above we trained our model and kept track of the Noise2Noise L^2 loss $MSE(f_\theta(\hat{\mathbf{x}}),\tilde{\mathbf{x}})$, the ground truth L^2 loss $MSE(f_\theta(\hat{\mathbf{x}}),\mathbf{y})$, the mean squared error between the ground truth overlay and the noisy overlay $MSE(\mathbf{y},\tilde{\mathbf{x}})$, our ground truth free estimate for this value $\frac{1}{2}MSE(\hat{\mathbf{x}},\tilde{\mathbf{x}})$, and our ground truth free estimate for the ground truth L^2 loss $MSE(f_\theta(\hat{\mathbf{x}}),\tilde{\mathbf{x}}) - \frac{1}{2}MSE(\hat{\mathbf{x}},\tilde{\mathbf{x}})$.

5.2. Results on the real overlay dataset

Interpreting results on the real overlay dataset

Unlike the synthetic overlay dataset, for the real overlay dataset, we do not have any ground truth overlay states $\mathbf{y}_i$ in the dataset. We do still make the sampling assumptions of section 1.2, and instead of trying to approximate the measurement $\mathbf{y}_i$ which we previously construct ourselves, we now

approximate the conditional expectation $\mathbb{E}\,\hat{\mathbf{x}}_i|\tilde{\mathbf{x}}_i$ and make this our target $\mathbf{y}_i := \mathbb{E}\,\hat{\mathbf{x}}_i|\tilde{\mathbf{x}}_i$. Using these assumptions in section 5.9, we found a lower bound for our L^2 Noise2Noise expected risk, namely

$$\mathbb{E}_{\hat{\mathbf{x}},\tilde{\mathbf{x}}}||f_\theta(\hat{\mathbf{x}}) - \tilde{\mathbf{x}}||^2 \geq \frac{1}{2}\mathbb{E}_{\hat{\mathbf{x}},\tilde{\mathbf{x}}}||\hat{\mathbf{x}} - \tilde{\mathbf{x}}||^2. \tag{5.18}$$

If our model $f_\theta(\cdot)$ perfectly predicts $\mathbf{y}_i$ s.t. for any i $f_\theta(\hat{\mathbf{x}}_i) = \mathbf{y}_i$ we achieve equality. If our model, on the other hand, does not remove any noise and is just the identity function, we get

$$\mathbb{E}_{\hat{\mathbf{x}},\tilde{\mathbf{x}}}||f_\theta(\hat{\mathbf{x}}) - \tilde{\mathbf{x}}||^2 = \mathbb{E}_{\hat{\mathbf{x}},\tilde{\mathbf{x}}}||\hat{\mathbf{x}} - \tilde{\mathbf{x}}||^2. \tag{5.19}$$

We can thus judge how well our denoising function $f_\theta(\cdot)$ is performing by how close $MSE(f_\theta(\hat{\mathbf{x}}), \tilde{\mathbf{x}})$ is to $\frac{1}{2}MSE(\hat{\mathbf{x}}, \tilde{\mathbf{x}})$ and whether it is lower than $MSE(\hat{\mathbf{x}}, \tilde{\mathbf{x}})$. To make this easier, we will, for the real dataset, normalize the losses $MSE(f_\theta(\hat{\mathbf{x}}), \tilde{\mathbf{x}})$ by $MSE(\hat{\mathbf{x}}, \tilde{\mathbf{x}})$, where both measures are calculated either on the training or validation set. We found that $MSE(\hat{\mathbf{x}}, \tilde{\mathbf{x}})$ differs considerably between these sets, and by normalizing $MSE(f_\theta(\hat{\mathbf{x}}), \tilde{\mathbf{x}})$ by $MSE(\hat{\mathbf{x}}, \tilde{\mathbf{x}})$, we are left with proportion on the means squared error removed, which can be compared between the validation and training set. Because the overlay measurements of the monitor wafers are sensitive information we will not show these measurements or the output of our denoising function on these measurements.

Predicting noise instead of signal

When we first trained our model $f_\theta(\cdot)$ with the masked loss of equation 4.34, we found that while the validation loss did decrease over time, it took almost a week of training to consistently achieve a loss lower than $MSE(\hat{\mathbf{x}}, \tilde{\mathbf{x}})$ and thus achieve a better denoising function than the identity function. This decrease in loss can be seen in the blue line of Figure 5.10. When we then inspected the output of the resulting denoising function, we found that the function made little changes for many monitor wafer measurements, as the output was very close to the input. This gave us the idea to use instead of $f_\theta(\cdot)$, the function $g_\theta(\cdot)$ as defined by equation 5.4, so

$$g_\theta(\hat{\mathbf{x}}_i) := \hat{\mathbf{x}}_i + f_\theta(\hat{\mathbf{x}}_i). \tag{5.20}$$

If for this function we take $f_\theta(\cdot) = \mathbf{0}$, we end up with the identity function, meaning that the identity function no longer needs to be approximated.

When we trained our model based on $g_\theta(\cdot)$, we found very different convergence behavior compared to our baseline function $f_\theta(\cdot)$, unlike our previous experiment on the synthetic dataset. After a single epoch the validation loss $MSE(g_\theta(\hat{\mathbf{x}}), \tilde{\mathbf{x}})$ was already lower than $MSE(\hat{\mathbf{x}}, \tilde{\mathbf{x}})$ and $g_\theta(\cdot)$ thus a better denoiser than the identity function. After tens of hours the validation loss $MSE(g_\theta(\hat{\mathbf{x}}), \tilde{\mathbf{x}})$ reached its minimum instead of more than two weeks for $MSE(f_\theta(\hat{\mathbf{x}}), \tilde{\mathbf{x}})$. The minimum achieved was also lower than the minimum achieved by $f_\theta(\cdot)$, indicating better denoising performance. The reduction in validation loss was initially fast but, after a few epochs, relatively noisy compared to the loss reduction. Because $g_\theta(\cdot)$ was considerably faster to train and gave better performance compared to $f_\theta(\cdot)$, we continued with $g_\theta(\cdot)$ as the denoising function on the real dataset.

Comparing the denoising performance of our model on the real and synthetic overlay dataset

To compare the performance of our best performing denoising models on both the synthetic- and real overlay datasets, we created the two plots of Figure 5.11. Both plots show the Noise2Noise L^2 training and validation loss during training of the models on their respective datasets. Here the model on the real data used the model $g_\theta(\cdot)$ of equation 5.4, does not used the random rotation augmentation method, and has the measurements with highest loss filtered out during training. The latter to design decisions are treated in appendix A. Both losses are normalized by $MSE(\hat{\mathbf{x}}, \tilde{\mathbf{x}})$ calculated on their respective validation set such that we can compare them. Both models denoise better than the identity functions as their loss is lower than $MSE(\hat{\mathbf{x}}, \tilde{\mathbf{x}})$. The model trained on the synthetic dataset is relatively closer to the minimal possible loss $\frac{1}{2}MSE(\hat{\mathbf{x}}, \tilde{\mathbf{x}})$ and thus removes a larger part of the noise in the input. The model trained on the real overlay dataset seems to start overfitting after 15 epochs, while the model trained on the synthetic dataset does not seem to overfit significantly, even after 300 epochs.

We can use equation 5.16 to convert our Noise2Noise mean squared error $MSE(g_\theta(\hat{\mathbf{x}}), \tilde{\mathbf{x}})$ to the ground truth mean squared error $MSE(g_\theta(\hat{\mathbf{x}}), \mathbf{y})$, where $\mathbf{y}$ then equals the mean $\mathbb{E}\,\hat{\mathbf{x}}_i|\tilde{\mathbf{x}}_i$; the noise free

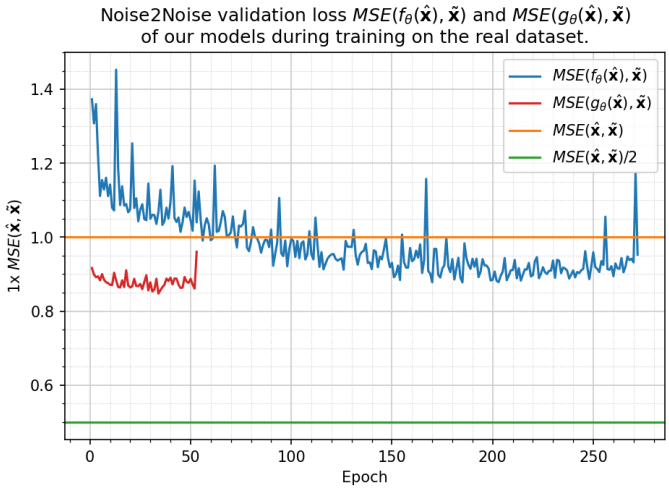
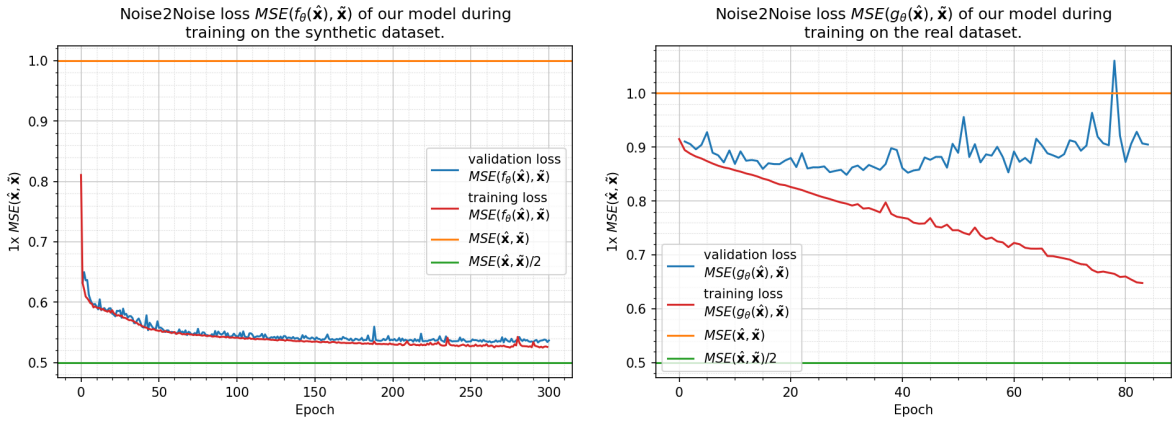


Figure 5.10: The masked mean squared error validation loss during training of our model based on $f_\theta(\cdot)$ as defined in chapter 4 and with the alternative function $g_\theta(\cdot)$ defined by equation 5.4. Both models are trained and validated on our real overlay dataset without the data augmentation procedure of section 4.2.

overlay state for each measurement pair. The mean squared error on the real dataset is calculated using the masking of equation 4.34, meaning that the most extreme overlay measurements are already removed from the error calculation and such that these few extreme overlay measurements do not dominate the error. Using this, we can conclude that for the real overlay data $g_\theta(\hat{\mathbf{x}}_i)$ is on average a 30% lower mean squared error estimator for $\mathbf{y}_i$ than $\hat{\mathbf{x}}_i$ is. On our synthetic overlay dataset $f_\theta(\hat{\mathbf{x}}_i)$ is on average a 97% lower mean squared error estimator for $\mathbf{y}_i$ than $\hat{\mathbf{x}}_i$ is. We think this difference is largely caused by how the synthetic dataset is created, which makes denoising the synthetic dataset a relatively simpler task. This gap in denoising performance could possibly be made smaller with a better training strategy and/or model $g_\theta(\cdot)$.



(a) (b)

Figure 5.11: The Noise2Noise L^2 validation loss during training of our model $f_\theta(\cdot)$ on the synthetic overlay dataset (a), and of $g_\theta(\cdot)$ trained on the real overlay dataset (b). Both loss plots are normalized by $MSE(\hat{\mathbf{x}}, \tilde{\mathbf{x}})$ calculated on their respective validation set. The model $g_\theta(\cdot)$ does not use the data augmentation procedure of section 4.2, but does skip large loss iterations as explained in appendix A.

6

Conclusions and recommendations

This thesis developed a machine-learning model for outlier removal in overlay measurements. As outliers for overlay measurements are not clearly defined, we instead made a model that denoised the overlay measurements. Because no noisy and noise-free overlay measurement pairs exist, our model was trained using the Noise2Noise method, which instead used noisy measurement pairs to learn to denoise overlay measurements. The model is based on a message-passing neural network with an Encoder-Processor-Decoder architecture. Because our overlay dataset does not have reference noise-free overlay measurements, we constructed a synthetic overlay dataset. This synthetic overlay dataset was designed to have some of the core properties of the real overlay dataset such that we could validate the features added to our model on this dataset. The research was guided by several key questions regarding the model's effectiveness as posed in section 3.4. These questions are split up into questions for the model trained on the synthetic overlay dataset and the model trained on the real overlay dataset.

6.1. Conclusions on the model trained on the synthetic overlay dataset

Our first question was if our model $f_\theta(\cdot)$ could, for the synthetic overlay dataset, learn to properly denoise noisy overlay measurements pairs $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{x}}_i$ without access to the target noise-free overlay measurements $\mathbf{y}_i$. We can answer this question with a resounding "yes", as our model scores an average R^2 score of 0.976 for predicting the noise-free overlay measurement from the noisy overlay measurements. From samples of the model's output, we can see that predictions $f_\theta(\hat{\mathbf{x}}_i)$ closely approximate their noise-free overlay target $\mathbf{y}_i$ and that both the added normal noise and simulated contaminations are accurately removed from the input overlay measurements. While our model trained without clean overlay targets performed slightly worse than the same model trained with clean overlay targets, which achieved an average R^2 score of 0.981, we think this difference stems from the fact that our dataset is of finite size and thus slightly worse performance is expected. This would mean that the difference in performance is not caused by the wrong norm during Noise2Noise training, and thus, as we proved using our overlay sampling assumptions, the L^2 norm is the proper norm to use for the Nois2Noise training on our data.

Because the synthetic overlay dataset includes the clean overlay targets $\mathbf{y}_i$, we were able to test multiple model features of our model using ablations studies. We first performed an ablation study on the physics-based encoding in our model, namely the marking of vertexes close to the border of the wafer and the marking of graph edges that cross exposure field borders. We found that the model's performance reduced slightly at the locations of the encodings when when these encodings where removed from the model. However, these differences in performance were small compared to the noise of the loss when the model was trained. We hypothesized that this small difference was because the model used the spatial information in the mesh to fit the field and wafer borders instead. This hypothesis was confirmed by the fact that the accuracy improvement of the field border encoding increased significantly when all further information on the field borders was removed in the mesh encoding. The physics-based encodings should make the model robust to different measurement layouts in the real

dataset where some overlay measurements on the wafer fail resulting in a mesh which is no longer constant.

To determine if our multiscale graph, which incorporates longer edges to enhance the speed of information travel, improves accuracy compared to a graph with only shorter edges, we trained models with various numbers of message-passing steps on synthetic data and these two different computation graphs. Our multiscale graph notably decreased the maximal path length between measurement points within the network, enabling accurate denoising with fewer message-passing steps. This reduction leads to fewer parameters and faster training and inference times. Despite the additional calculations required by the extra edges in the multiscale graph, the overall model speed improves due to the reduced number of message-passing steps. This demonstrates that the multiscale graph structure significantly enhances computational efficiency and reduces the number of necessary parameters.

We further found that our random rotation augmentation during training on the synthetic overlay dataset prevented our model from overfitting, leading to significantly better denoising performance on the validation set. Surprisingly, our alternative approach to decreasing overfitting, which removed the relative location vectors in the spatial encoding of the mesh, did not seem to change the model's training behavior and thus also did not decrease overfitting. We decided not to remove this information from the model's inputs as the information may be necessary on the real dataset, for which we expect more complicated behavior. This has, however, not been tested.

When we made a slight modification to our model, which we define as $g_\theta(\hat{\mathbf{x}}_i) := \hat{\mathbf{x}}_i + f_\theta(\hat{\mathbf{x}}_i)$ and interoperate as predicting the noise instead of signal, we found on the synthetic dataset no significant performance difference. We did find, through an error in our dataset, that the function $g_\theta(\cdot)$ is more sensitive to extreme values in the training set.

When investigating the relation between the ground truth L^2 loss and our Noise2Noise L^2 loss, which we use to train our model and is the only loss available on the real dataset, we found an estimator for the ground truth loss that could be calculated without access to the ground truth data. This estimator could also be used as an alternative proof for the Noise2Noise method with the L^2 loss.

6.2. Conclusions on the model trained on the real overlay dataset

Since we validated most of our model features on the synthetic overlay data, we assumed their effectiveness would generalize to the real overlay dataset. We did, during training, find that the real overlay dataset contained very large overlay values which dominated the training loss and led to overfitting on these values. After masking the measurement locations where these extreme values occurred in the loss, we found that the validation loss decreased when training our model.

An interesting observation was that, if we learned to predict the overlay noise with $g_\theta(\cdot)$ instead of the overlay signal with $f_\theta(\cdot)$, we got, unlike for our model applied to the synthetic dataset, very different validation loss curves. The minimum on the validation set for $g_\theta(\cdot)$ during training was reached in days instead of weeks when training $f_\theta(\cdot)$, and this minimum loss is also slightly lower for $g_\theta(\cdot)$ than for $f_\theta(\cdot)$. We can thus conclude that the prior of $g_\theta(\cdot)$, which can be interpreted as that the noise added to the overlay measurement is easier to learn than the denoised signal, better fits the real overlay data than the synthetic overlay. We found that after this minimum validation loss was achieved, both models started to overfit on the real overlay data. For our fastest training model, $g_\theta(\cdot)$, this significant overfitting already began after just 15 epochs of training.

Using our estimator for the ground truth loss, we can compare our model's performance on both datasets. Here, the target of our synthetic dataset is the noise-free measurement $\mathbf{y}_i$ the ground truth overlay state to which noise was added, for our real dataset this target is defined as the mean $\mathbb{E}\,\hat{\mathbf{x}}_i|\tilde{\mathbf{x}}_i$; the noise-free overlay state for each measurement pair. For the real overlay data $g_\theta(\hat{\mathbf{x}}_i)$ is on average a 30% lower mean square error estimator for $\mathbb{E}\,\hat{\mathbf{x}}_i|\tilde{\mathbf{x}}_i$ than $\hat{\mathbf{x}}_i$ is. On our synthetic overlay dataset $f_\theta(\hat{\mathbf{x}}_i)$ is on average a 97% lower mean square error estimator for $\mathbf{y}_i$ than $\hat{\mathbf{x}}_i$ is. There is quite a significant gap between these two scores. We see two possible explanations for this gap. The first could be that our model was not complex enough to approximate the noise-free overlay measurements or was hindered by overfitting. The second explanation could be that when sampling $\hat{\mathbf{x}}_i$ from the noisy overlay distribution $p(\mathbf{x}_i|\mathbf{y}_i)$ a significant part of the information on $\mathbf{y}_i$ is lost. We think both explanations for the difference in scores play a role, but it is hard to distinguish to what proportion both explanations are responsible for the performance gap.

If we compare the outputs of our denoising models on their respective datasets, we can make some

interesting observations. On the synthetic dataset, our model properly removed both the normal noise and the simulated outliers we defined. For the real dataset, our model removes what appears to be local contaminations and more global shifts in the overlay caused by the stochastic state of the scanner.

6.3. Recommendations

Our model learned to accurately predict the target ground overlay state on the synthetic dataset overlay dataset. The main unanswered question about the model's features is the effect of the relative location vector encoding. This encoding was taken from the MeshGraphNet model on which we based our model, but when removed, it did not seem to change the model's performance on the synthetic dataset. It would be interesting to study further if this is just the case for our synthetic dataset or would also be the case for other datasets and applications.

Our model removes a significantly lower proportion of the wafer-to-wafer noise from the real overlay dataset than from the synthetic overlay dataset. How big of a proportion of the wafer-to-wafer noise can theoretically be removed is unknown. This means how much better a more complex or better-trained model could perform is also unknown.

One major indication of room for improvement is that our current model for the real dataset suffers from overfitting. This was most significant when we did not mask out the largest losses, but even masking overfitting remains a major concern as a relatively small number of overlay measurements make a relatively big contribution to the loss. When we minimize the loss, we thus are mainly minimizing the loss on this small subset of wafer measurement leading to overfitting. Normally, this problem could be solved by using some sort of relative loss that normalizes the losses such that the contributions are more equal. In the Noise2Noise paper, the authors mentioned that they had this exact problem when denoising images that used the unbounded HDR value for the pixel values. They warn that replacing the mean square error with the relative mean square error changes the expectation the Noise2Noise method will converge to. Instead, they develop an alternative loss that does not alter the converge point of the method and is in the limit equal to the relative mean square error for positive numbers. This new loss gives their model a great performance improvement. Their loss function is, however, only convex for positive numbers and thus does not work as a loss function for our data. We think that a similar loss function that works for our data could bring major improvements by reducing overfitting. Some sort of normalization of the measurements, which limits the contribution of the largest loss values to the total loss could also be investigated.

An alternative way to reduce overfitting would be adding data augmentation methods during training. This approach worked very well for our synthetic data and also prevented overfitting on the real dataset, but it led to worse overall performance. We hypothesize that rotating a single overlay measurement conceals that the dx and dy values of the measurement come from separate measurements, which is lost if the vector is rotated. A possible solution would be only to rotate the wafers by multiples of 90 degrees and use mirroring. A different data augmentation technique could be to remove a random proportion of the input measurement points and thus create multiple different input graphs and feature vectors from a single monitor wafer batch.

We see two possible methods to use our model and reduce the output variance of the overlay calibration model. The first would be to classify outliers in $\hat{\mathbf{x}}_i$ by the length of the vectors of the predicted noise $\hat{\mathbf{x}}_i - g_\theta(\hat{\mathbf{x}}_i)$. We would expect this method to generate an accurate classifier for outliers, and we perceive the chance that this method adds significant bias to the calibration model to be very small. The second way to integrate the model would be to use the outputs of our model $g_\theta(\hat{\mathbf{x}}_i)$ and $g_\theta(\tilde{\mathbf{x}}_i)$ as input to the overlay calibration model. This method has the possibility of significantly reducing the output variance of the model but comes with the risk of adding some bias to the calibration. Both methods have not been tested and should be rigorously validated before implementation. These implementations would come with the usual risks of a black box model.

Bibliography

[1] ASML. *2018 Annual Report*. https://www.asml.com/en/investors/annual-report/2018. Feb. 2019.

[2] ASML. *Inside NXE3400 - Metrology*. 2018. URL: https://asml.picturepark.com/s/AB09FG2k.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer Normalization". In: (2016). arXiv: 1607.06450 [stat.ML].

[4] Joshua Batson and Loïc Royer. "Noise2Self: Blind Denoising by Self-Supervision". In: *CoRR* abs/1901.11365 (2019). arXiv: 1901.11365. URL: http://arxiv.org/abs/1901.11365.

[5] Peter W. Battaglia et al. "Relational inductive biases, deep learning, and graph networks". In: *CoRR* abs/1806.01261 (2018). arXiv: 1806.01261. URL: http://arxiv.org/abs/1806.01261.

[6] Twan Bearda et al. "The Effect of Backside Particles on Substrate Topography". In: *Japanese Journal of Applied Physics* 44 (Oct. 2005), pp. 7409–7413. DOI: 10.1143/JJAP.44.7409.

[7] Indranil Bose and Radha Mahapatra. "Business data mining - A machine learning perspective". In: *Information Management* 39 (Dec. 2001), pp. 211–225. DOI: 10.1016/S0378-7206(01)00091-X.

[8] Michael M. Bronstein et al. "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges". In: *CoRR* abs/2104.13478 (2021). arXiv: 2104.13478. URL: https://arxiv.org/abs/2104.13478.

[9] Jens Busch et al. "Improving lithographic performance for 32 nm". In: 7638 (2010), pp. 58–73.

[10] Adria Font Calvarons. "Improved Noise2Noise Denoising with Limited Data". In: (2021), pp. 796–805. DOI: 10.1109/CVPRW53098.2021.00089.

[11] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey". In: *ACM Comput. Surv.* 41.3 (July 2009). ISSN: 0360-0300. DOI: 10.1145/1541880.1541882. URL: https://doi.org/10.1145/1541880.1541882.

[12] Priyam Chatterjee and Peyman Milanfar. "Is Denoising Dead?" In: *IEEE Transactions on Image Processing* 19.4 (2010), pp. 895–911. DOI: 10.1109/TIP.2009.2037087.

[13] Nick DiCicco. *Node representations update in a Message Passing Neural Network (MPNN) layer. Node receives messages from all of his immediate neighbours to . Messages are computing via the message function , which accounts for the features of both endpoints of an adjacency.* 2022. URL: https://en.wikipedia.org/wiki/Graph_neural_network#/media/File:Message_Passing_Neural_Network.png.

[14] Michael Elad, Bahjat Kawar, and Gregory Vaksman. "Image Denoising: The Deep Learning Revolution and Beyond – A Survey Paper –". In: (2023). arXiv: 2301.03362 [eess.IV].

[15] Matthias Fey and Jan Eric Lenssen. "Fast Graph Representation Learning with PyTorch Geometric". In: *CoRR* abs/1903.02428 (2019). arXiv: 1903.02428. URL: http://arxiv.org/abs/1903.02428.

[16] Ugo Fiore et al. "Network anomaly detection with the restricted Boltzmann machine". In: *Neurocomputing* 122 (Dec. 2013), pp. 13–23. DOI: 10.1016/j.neucom.2012.11.050.

[17] Meire Fortunato et al. "MultiScale MeshGraphNets". In: (2022). arXiv: 2210.00612 [cs.LG].

[18] Glosser.ca. *Artificial neural network with layer coloring*. 2013. URL: https://en.wikipedia.org/wiki/Neural_network_%28machine_learning%29#/media/File:Colored_neural_network.svg.

[19] Yulan Guo et al. “Deep Learning for 3D Point Clouds: A Survey”. In: *CoRR* abs/1912.12033 (2019). arXiv: 1912.12033. URL: http://arxiv.org/abs/1912.12033.

[20] Elad Hazan. *Introduction to Online Convex Optimization*. 2023. arXiv: 1909.05207 [cs.LG].

[21] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[22] Peter J. Huber. “Robust Estimation of a Location Parameter”. In: *The Annals of Mathematical Statistics* 35.1 (1964), pp. 73–101. DOI: 10.1214/aoms/1177703732. URL: https://doi.org/10.1214/aoms/1177703732.

[23] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[24] Yasushi Ito. *Delaunay Triangulation*. Ed. by Björn Engquist. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 332–334. ISBN: 978-3-540-70529-1. DOI: 10.1007/978-3-540-70529-1_314. URL: https://doi.org/10.1007/978-3-540-70529-1_314.

[25] M.J. Jansen. “Development of a wafer geometry measuring system : a double sided stitching interferometer”. English. Phd Thesis 1 (Research TU/e / Graduation TU/e). Mechanical Engineering, 2006. ISBN: 90-386-2758-0. DOI: 10.6100/IR611508.

[26] Longxiang Jiang and Junjie Wang. *Learning Mesh-Based Simulation with Graph Networks*. https://github.com/echowve/meshGraphNets_pytorch. 2022.

[27] Ryan Keisler. “Forecasting Global Weather with Graph Neural Networks”. In: (2022). arXiv: 2202.07575 [physics.ao-ph].

[28] Henry J Kelley. “Gradient theory of optimal flight paths”. In: *Ars Journal* 30.10 (1960), pp. 947–954.

[29] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations* (Dec. 2014).

[30] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *CoRR* abs/1609.02907 (2016). arXiv: 1609.02907. URL: http://arxiv.org/abs/1609.02907.

[31] Chiew-seng Koay et al. “Automated optimized overlay sampling for high-order processing in double patterning lithography”. In: 7638 (2010). Ed. by Christopher J. Raymond, 76381R. DOI: 10.1117/12.846371. URL: https://doi.org/10.1117/12.846371.

[32] Alexander Krull, Tim-Oliver Buchholz, and Florian Jug. “Noise2Void - Learning Denoising from Single Noisy Images”. In: *CoRR* abs/1811.10980 (2018). arXiv: 1811.10980. URL: http://arxiv.org/abs/1811.10980.

[33] Remi Lam et al. “Learning skillful medium-range global weather forecasting”. In: *Science* 382.6677 (2023), pp. 1416–1421. DOI: 10.1126/science.adi2336. eprint: https://www.science.org/doi/pdf/10.1126/science.adi2336. URL: https://www.science.org/doi/abs/10.1126/science.adi2336.

[34] Jaakko Lehtinen et al. “Noise2Noise: Learning Image Restoration without Clean Data”. In: (2018). arXiv: 1803.04189 [cs.CV].

[35] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. Cambridge, Mass. [u.a.]: MIT Press, 2013. ISBN: 9780262018029 0262018020. URL: https://www.amazon.com/Machine-Learning-Probabilistic-Perspective-Computation/dp/0262018020/ref=sr_1_2?ie=UTF8&qid=1336857747&sr=8-2.

[36] Ali Bou Nassif et al. “Machine Learning for Anomaly Detection: A Systematic Review”. In: *IEEE Access* 9 (2021), pp. 78658–78700. DOI: 10.1109/ACCESS.2021.3083060.

[37] Tobias Pfaff et al. “Learning Mesh-Based Simulation with Graph Networks”. In: *CoRR* abs/2010.03409 (2020). arXiv: 2010.03409. URL: https://arxiv.org/abs/2010.03409.

[38] Charles Ruizhongtai Qi et al. “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”. In: *CoRR* abs/1612.00593 (2016). arXiv: 1612.00593. URL: http://arxiv.org/abs/1612.00593.

[39] Charles Ruizhongtai Qi et al. “PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space”. In: *CoRR* abs/1706.02413 (2017). arXiv: 1706.02413. URL: http://arxiv.org/abs/1706.02413.

[40] John C. Robinson et al. “Improved overlay control using robust outlier removal methods”. In: 7971 (2011). Ed. by Christopher J. Raymond, 79711G. DOI: 10.1117/12.879494. URL: https://doi.org/10.1117/12.879494.

[41] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[42] Emil Schmitt-Weaver and Kaustuve Bhattacharyya. “Pairing wafer leveling metrology from a lithographic apparatus with deep learning to enable cost effective dense wafer alignment metrology”. In: 10961 (2019). Ed. by Jongwook Kye and Soichi Owa, p. 1096109. DOI: 10.1117/12.2514455. URL: https://doi.org/10.1117/12.2514455.

[43] Peter Vanoppen et al. “Lithographic scanner stability improvements through advanced metrology and control”. In: 7640 (2010). Ed. by Mircea V. Dusa and Will Conley, p. 764010. DOI: 10.1117/12.848200. URL: https://doi.org/10.1117/12.848200.

[44] Chris Wellons. *Noise Fractals and Clouds*. 2007. URL: https://nullprogram.com/blog/2007/11/20/ (visited on 03/29/2024).

[45] Dongxian Wu et al. “Skip Connections Matter: On the Transferability of Adversarial Examples Generated with ResNets”. In: *CoRR* abs/2002.05990 (2020). arXiv: 2002.05990. URL: https://arxiv.org/abs/2002.05990.

A

Additional results on the real overlay dataset

Random rotation data augmentation on the real data

In section 5.1 we saw large improvements when we used our random rotation data augmentation procedure during training on the synthetic dataset. To see if this is also the case for the real dataset we trained two models $g_\theta(\cdot)$ with and without data augmentation on the real dataset. The resulting validation loss curves can be seen in Figure A.1. The data augmentation technique seems to prevent overfitting as the training loss and validation loss now remain similar for 70 epochs instead of 30, but the technique also seems to introduce some bias as the model trained without the data augmentation achieves a lower validation loss at its respective minimum.

We hypothesize that the source of this bias is the fact that on the real dataset, each measurement vector is made up of two separate dx and dy measurements. This means that if only the dx marker is contaminated the dy marker does not also need to be contaminated and thus need not give a wrong large measurement. If you now rotate the measurement vector by a random angle we no longer get the separate values from the measurements but some combination, and information is lost. More tests should be performed to see if this is the case.

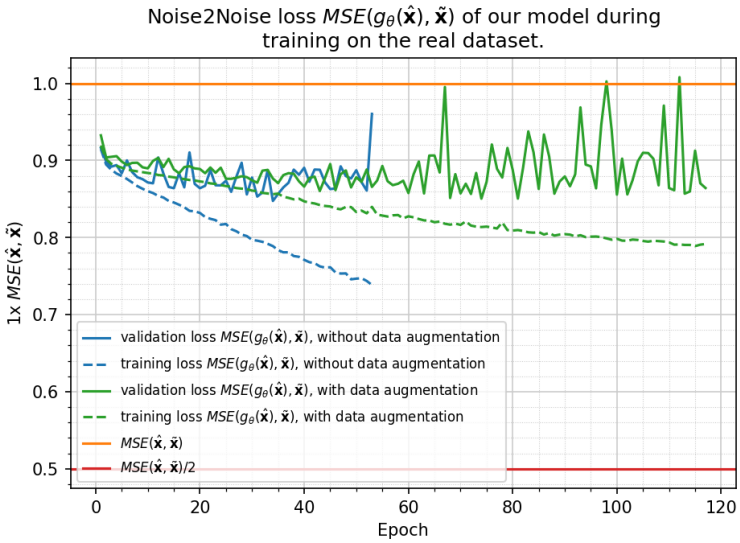


Figure A.1: The masked mean square error validation loss during training of our model $g_\theta(\cdot)$ for our model trained with and without the random rotation technique of section . Both models are trained and validated on our real overlay dataset.

Ignoring large loss values during training

In section 5.1, we saw that large values in the training set of the synthetic dataset led to very high loss values which in turn led to unstable or non-convergence of the validation loss. When training our model on the real dataset we saw similar high loss values for some batches. This gave us the idea to skip the optimization step of batches with extremely high loss values, which we hoped would lead to more stable training like when we removed the faulty measurement from the synthetic dataset. This could lead to some bias as the measurements causing large losses would no longer be trained on but we hoped the extra training stability would make up for this fact. We decided to skip the optimization step if a batch loss was larger than $3 \cdot MSE(\hat{\mathbf{x}}, \tilde{\mathbf{x}})$ as with this number most batches were still included but not the most extreme values.

We can see the result of this experiment in Figure A.2, that while our technique does seem to have slightly stabilized our validation loss during training, it did not increase the model's performance.

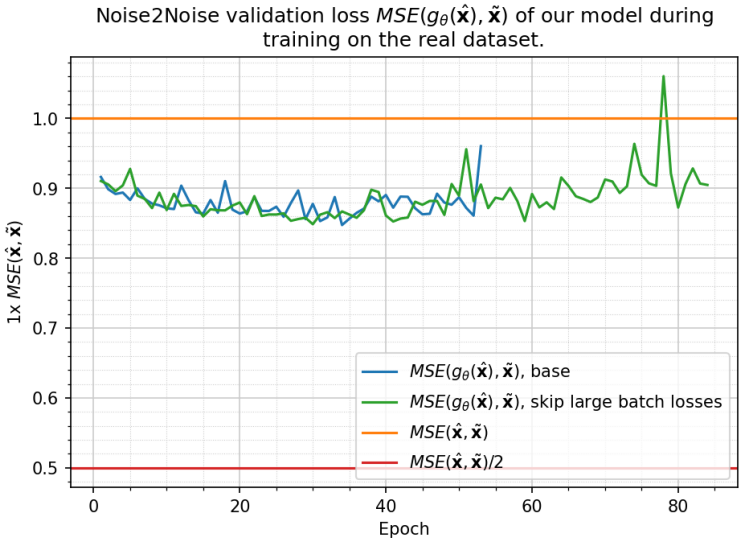
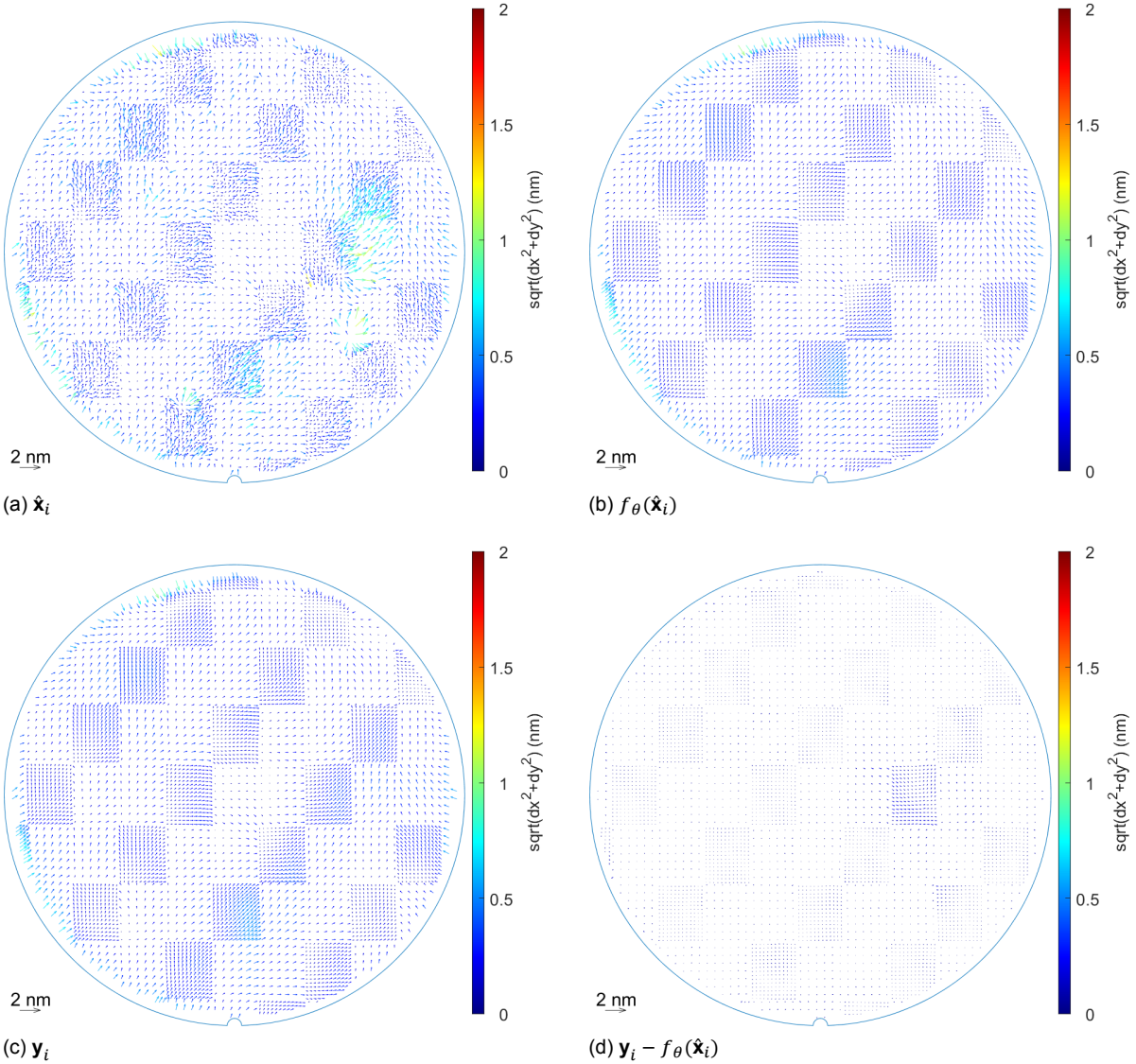


Figure A.2: The masked mean square error validation loss during training of our model $g_\theta(\cdot)$ where we compare the base model with a model where the optimization step was skipped if the batch loss was bigger than $3 \cdot MSE(\hat{\mathbf{x}}, \tilde{\mathbf{x}})$. Both models are trained and validated on our real overlay dataset without the data augmentation procedure of section 4.2.

B

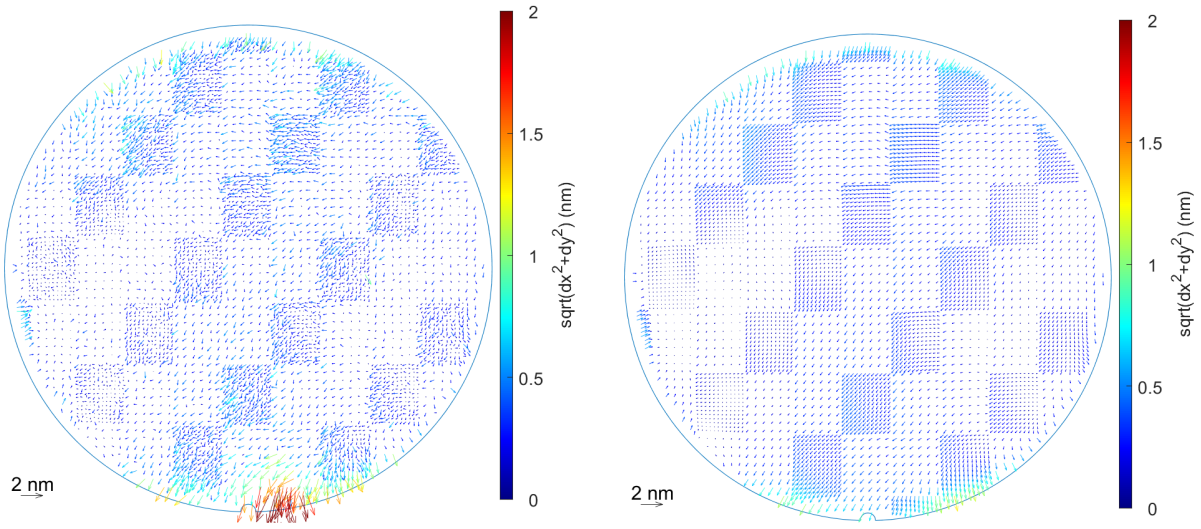
Example model outputs on the validation set of the synthetic data



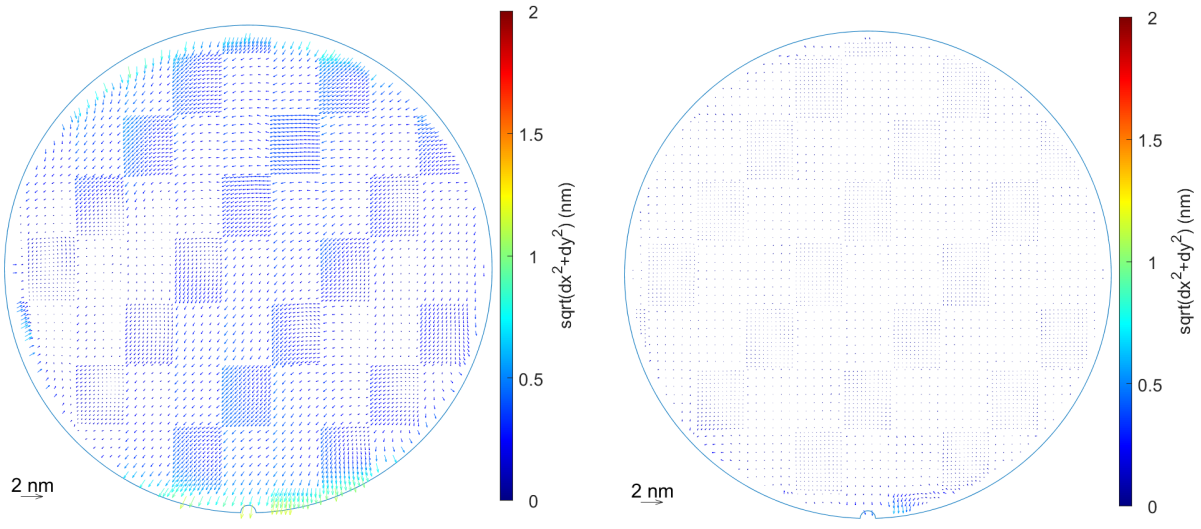
(a) $\hat{\mathbf{x}}_i$ (b) $f_\theta(\hat{\mathbf{x}}_i)$

(c) $\mathbf{y}_i$ (d) $\mathbf{y}_i - f_\theta(\hat{\mathbf{x}}_i)$

Figure B.1: Our denoising model applied to a noisy overlay sample. Figure (a) shows the noisy input sample $\hat{\mathbf{x}}_i$, (b) shows the output of our model $f_\theta(\hat{\mathbf{x}}_i)$ on this sample, (c) shows the ground truth overlay $\mathbf{y}_i$ from which the sample $\hat{\mathbf{x}}_i$ was created, and (d) shows the error between the prediction and ground truth overlay defined as $\mathbf{y}_i - f_\theta(\hat{\mathbf{x}}_i)$.

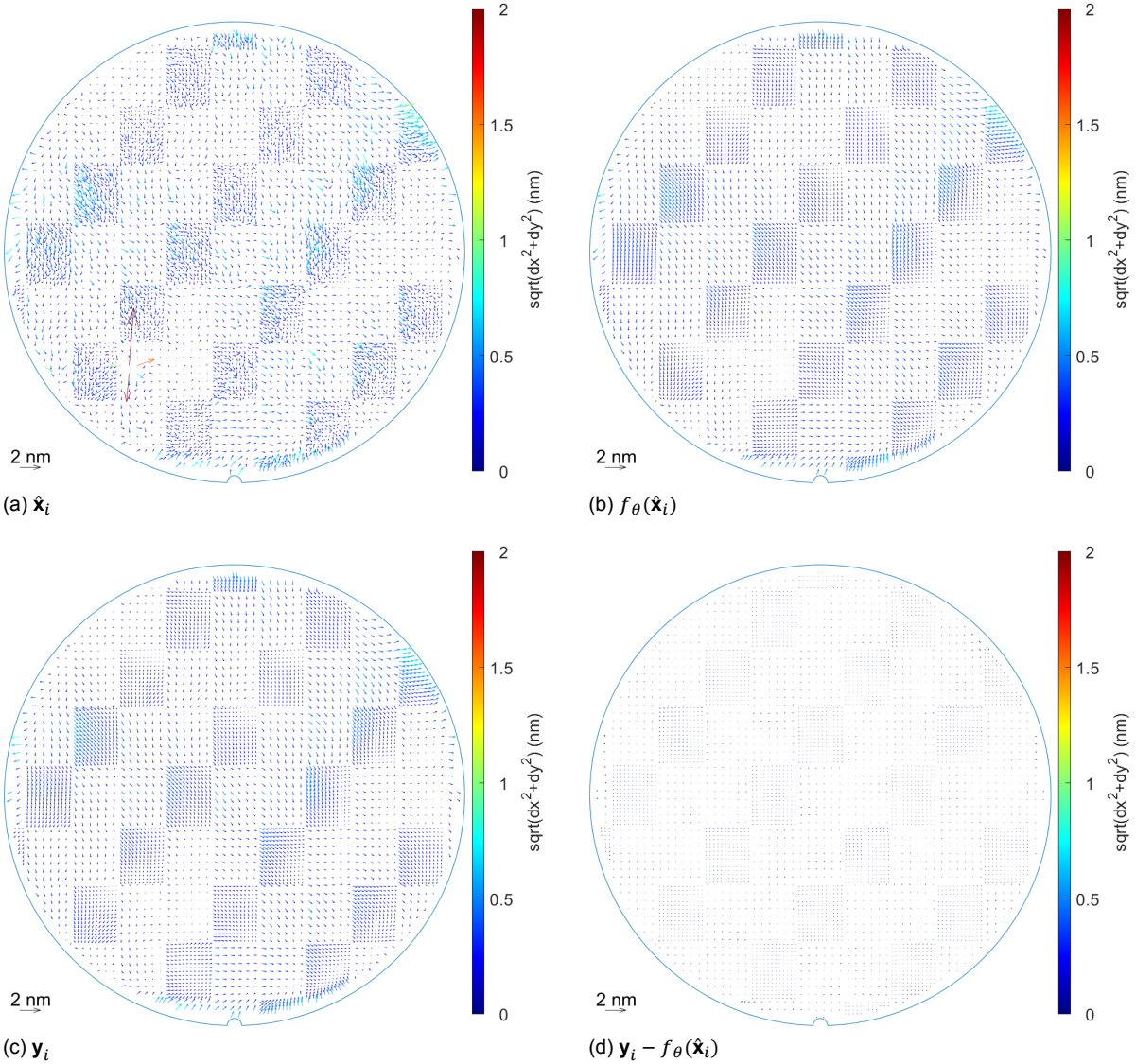


(a) $\hat{\mathbf{x}}_i$ (b) $f_\theta(\hat{\mathbf{x}}_i)$



(c) $\mathbf{y}_i$ (d) $\mathbf{y}_i - f_\theta(\hat{\mathbf{x}}_i)$

Figure B.2: Our denoising model applied to a noisy overlay sample. Figure (a) shows the noisy input sample $\hat{\mathbf{x}}_i$, (b) shows the output of our model $f_\theta(\hat{\mathbf{x}}_i)$ on this sample, (c) shows the ground truth overlay $\mathbf{y}_i$ from which the sample $\hat{\mathbf{x}}_i$ was created, and (d) shows the error between the prediction and ground truth overlay defined as $\mathbf{y}_i - f_\theta(\hat{\mathbf{x}}_i)$.



(a) $\hat{\mathbf{x}}_i$ (b) $f_\theta(\hat{\mathbf{x}}_i)$

(c) $\mathbf{y}_i$ (d) $\mathbf{y}_i - f_\theta(\hat{\mathbf{x}}_i)$

Figure B.3: Our denoising model applied to a noisy overlay sample. Figure (a) shows the noisy input sample $\hat{\mathbf{x}}_i$, (b) shows the output of our model $f_\theta(\hat{\mathbf{x}}_i)$ on this sample, (c) shows the ground truth overlay $\mathbf{y}_i$ from which the sample $\hat{\mathbf{x}}_i$ was created, and (d) shows the error between the prediction and ground truth overlay defined as $\mathbf{y}_i - f_\theta(\hat{\mathbf{x}}_i)$.

C

Synthetic overlay data creation

Fractal noise pattern

All the synthetic overlay data has been implemented using a fractal-like noise pattern created by [44]. The fractal-like noise pattern is created by the following Matlab function where `m,n` are the dimensions of the resulting matrix.

```
function im = generate_fractal_noise(n, m)
    im = zeros(n,m);
    i = 0;
    w = sqrt(n*m);

    while w > 3
        i = i + 1;
        d = interp2(randn(n, m), i-1,'cubic');
        im = im + i * d(1:n, 1:m);
        w = w - ceil(w/2 - 1);
    end
end
```

Interfield overlay pattern

The Matlab function used to generate the interfield overlay pattern of figure 4.4, where `X_wafer, Y_wafer` are the coordinates of the measurements on the wafer.

```
function overlay_interfield = noise_interfield(X_wafer, Y_wafer)

    % create the fractal like noise arrays
    n = 16;
    m = 16;
    noise_X = linspace(-0.15, 0.15, n);
    noise_Y = linspace(-0.15, 0.15, m);
    noise_array_U = generate_fractal_noise(n, m);
    noise_array_V = generate_fractal_noise(n, m);

    % Interpolate at the wafer coordinates and standerdize
    U = interp2(noise_X,noise_Y,noise_array_U,X_wafer,Y_wafer);
    U = (U/std(U));
    V = interp2(noise_X,noise_Y,noise_array_V,X_wafer,Y_wafer);
    V = (V/std(V));

    overlay_interfield = [U V];
end
```

Intrafield overlay pattern

The Matlab function used to generate the intrafield overlay pattern of figure 4.5, where `X_field`, `Y_field` are the coordinates of the measurements relative to the center of their respective fields.

```
function overlay_intrafield = noise_intrafield(X_field, Y_field)

    % create the fractal like noise arrays
    n = 8;
    m = 8;
    noise_X = linspace(-0.15, 0.15, n);
    noise_Y = linspace(-0.15, 0.15, m);
    noise_array_U = generate_fractal_noise(n, m);
    noise_array_V = generate_fractal_noise(n, m);
    noise_array_U = (noise_array_U/std(noise_array_U(:)));
    noise_array_V = (noise_array_V/std(noise_array_V(:)));

    % Interpolate at the field coordinates and standerdize
    U = interp2(noise_X,noise_Y,noise_array_U,X_field,Y_field);
    U = U / max(abs(U(:)));
    V = interp2(noise_X,noise_Y,noise_array_V,X_field,Y_field);
    V = V / max(abs(V(:)));

    overlay_intrafield = [U V];
end
```

Radial overlay pattern

The Matlab function used to generate the radial overlay pattern of figure 4.6, where `X_wafer, Y_wafer` are the coordinates of the measurements on the wafer.

```
function overlay_radial = noise_radial(X_wafer, Y_wafer)

    % generate radial pattern and normalize radii to (0,1]
    R = sqrt(X_wafer.^2 + Y_wafer.^2);
    U = R.^16 .* X_wafer;
    V = R.^16 .* Y_wafer;
    norm_factor = max(sqrt(U.^2 + V.^2));
    U = U / norm_factor;
    V = V / norm_factor;

    % generate fractal noise and interpolate to measurement points
    n = 8;
    m = 8;
    zoom = 2;
    noise_X = linspace(-0.15 * zoom, 0.15 * zoom, n);
    noise_Y = linspace(-0.15 * zoom, 0.15 * zoom, m);
    noise_array_U = generate_fractal_noise(n, m);
    noise_array_U = (noise_array_U/std(noise_array_U(:)));
    noise_R = interp2(noise_X, noise_Y, noise_array_U, X_wafer, Y_wafer);

    % multiply the noise pattern with the radial pattern pairwise
    U = U .* noise_R;
    V = V .* noise_R;
    overlay_radial = [U V];
end
```

Global overlay noise pattern

The Matlab function used to generate the global noise or defects of figure 4.8, where `X_wafer`, `Y_wafer` are the coordinates of the measurements on the wafer.

```
function overlay_global_noise = global_noise(X_wafer, Y_wafer)

    % sample the number of global defects
    number_of_defects = geornd(0.3);
    U = zeros(size(X_wafer));
    V = zeros(size(Y_wafer));

    if number_of_defects > 0
        for i = 1:number_of_defects
            % Sample derfect location, sigma (radius), amplitude (size of
               vectors), and split of radial and noise in the defect.
            X_loc = -0.15 + 0.3 * rand(1);
            Y_loc = -0.15 + 0.3 * rand(1);
            sigma  = 0.0025 + exprnd(0.01);
            amplitude = 0.4 + exprnd(0.5);
            radial_noise_split = 0.4 * exprnd(1);

            % construct Gaussian surface from pdf
            X_gauss = linspace(-0.15,0.15,200);
            Y_gauss = linspace(-0.15,0.15,200);
            [X_gauss, Y_gauss] = meshgrid(X_gauss, Y_gauss);
            Z_func =  exp(-1/(sigma^2)*((Y_gauss-Y_loc).^2 + (X_gauss-
               X_loc).^2));

            % create radial defect pattern from gradient of the pdf
            [X_grad, Y_grad] = gradient(Z_func);
            U_radial = - interp2(X_gauss, Y_gauss, X_grad, X_wafer,
               Y_wafer);
            V_radial = - interp2(X_gauss, Y_gauss, Y_grad, X_wafer,
               Y_wafer);
            R_radial = sqrt(U_radial.^2+V_radial.^2);
            U_radial = U_radial / max(R_radial);
            V_radial = V_radial / max(R_radial);

            % Generate noise of defect and multiply with pdf at
               measurement locations. Then normalize the radii to [0,1].
            n = 8;
            m = 8;
            noise_X = linspace(-0.15, 0.15, n);
            noise_Y = linspace(-0.15, 0.15, m);
            noise_array_U = generate_fractal_noise(n, m);
            noise_array_V = generate_fractal_noise(n, m);
            noise_array_U = (noise_array_U/std(noise_array_U(:)));
            noise_array_V = (noise_array_V/std(noise_array_V(:)));
            U_noise = interp2(noise_X,noise_Y,noise_array_U,X_wafer,
               Y_wafer).* interp2(X_gauss,Y_gauss,Z_func,X_wafer,Y_wafer);
            V_noise = interp2(noise_X,noise_Y,noise_array_V,X_wafer,
               Y_wafer).* interp2(X_gauss,Y_gauss,Z_func,X_wafer,Y_wafer);
            R_noise = sqrt(U_noise.^2+V_noise.^2);
            U_noise = U_noise / max(R_noise);
            V_noise = V_noise / max(R_noise);
```

```
            % Add the defect to the overlay using the radial and noise
                split sampled before, and multiply by the amplitude.
            U = U + amplitude * (radial_noise_split * U_radial + (1-
                radial_noise_split) * U_noise);
            V = V + amplitude * (radial_noise_split * V_radial + (1-
                radial_noise_split) * V_noise);
        end
    end

    overlay_global_noise = [U V];
end
```

Local overlay noise pattern

The Matlab function used to generate the local noise of figure 4.9, where `X_wafer, Y_wafer` are the coordinates of the measurements on the wafer and `overlay` is the ground truth overlay together with the global noise and defects.

```
function overlay_local_noise = local_noise(X_wafer, Y_wafer, overlay)

    % Sample 2d multivariate noise for every measurement
    base_noise = mvtrnd(eye(2),7,length(X_wafer));

    % Sample 2d multivariate noise for every measuremennt and multiply it
       with the radius of the base overlay to get radius dependent noise.
    radius_dependent_noise = sqrt(overlay(:,1).^2 + overlay(:,2).^2) .*
       mvnrnd(zeros(2,1), eye(2),length(X_wafer));

    % Add the two noises where the radius dependent noise is normalized to
        about 1
    overlay_local_noise = base_noise + 4e9 * radius_dependent_noise;
end
```

Combing the overlay patterns

The Matlab code with all the scaling parameters chosen that combine all the different synthetic noise patterns in to a ground truth overlay with two samples of added noise. All the functions used have been described above, `X_wafer, Y_wafer` are the coordinates of the measurements on the wafer, and `X_field, Y_field` are the coordinates of the points relative to their field center.

```
% ground truth overlay
overlay_interfield  = 0.08e-9 * noise_interfield(X_wafer, Y_wafer);
overlay_intrafield  = 0.35e-9 * noise_intrafield(X_field, Y_field);
overlay_radial      = 0.50e-9 * noise_radial(X_wafer, Y_wafer);
overlay_no_noise    = overlay_interfield + overlay_intrafield +
   overlay_radial;

% sample 1
global_noise_1      = 0.70e-9 * global_noise(X_wafer, Y_wafer);
overlay_1           = overlay_no_noise + global_noise_1;

local_noise_1       = 0.06e-9 * local_noise(X_wafer, Y_wafer, overlay_1);
overlay_1           = overlay_1 + local_noise_1;

% sample 2
global_noise_2      = 0.70e-9 * global_noise(X_wafer, Y_wafer);
overlay_2           = overlay_no_noise + global_noise_2;
```

```
local_noise_2       = 0.06e-9 * local_noise(X_wafer, Y_wafer, overlay_2);
overlay_2           = overlay_2 + local_noise_2;
```