







## Accuracy Assessment and Reinforcement Training of Deep Neural Networks in the Context of Accelerating a Simulation-Based Toolpath Generation Method

Tathagata Chakraborty<sup>1</sup> , Christine Zuzart<sup>2</sup> , Chinmaya Panda<sup>3</sup> , Nitin Umap<sup>4</sup> 

<sup>1</sup>HCL Technologies, [tathagata.chakr@hcl-software.com](mailto:tathagata.chakr@hcl-software.com)

<sup>2</sup>HCL Technologies, [christine.zuzart@hcl-software.com](mailto:christine.zuzart@hcl-software.com)

<sup>3</sup>HCL Technologies, [chinmaya.panda@hcl-software.com](mailto:chinmaya.panda@hcl-software.com)

<sup>4</sup>HCL Technologies, [nitin.umap@hcl-software.com](mailto:nitin.umap@hcl-software.com)

Corresponding author: Tathagata Chakraborty, [tathagata.chakr@hcl-software.com](mailto:tathagata.chakr@hcl-software.com)

**Abstract.** Despite the success of deep neural networks (DNNs) in many fields, their use in scientific domains remains challenging. DNNs give over-confident results and do not directly report a confidence measure alongside their output. The use of DNNs is thus often restricted to more error-tolerant domains. In scientific modeling, we usually understand the system's structure and the process mechanism in-depth. It is, however, difficult to incorporate this knowledge into a DNN. Nevertheless, there is a growing interest in applying DNNs to scientific computing. This paper presents a novel method for integrating the knowledge from simulation-driven systems into DNNs. In particular, we illustrate the use of the method to accelerate a simulation-driven toolpath generation method. The proposed method has a regularization effect and can add the right inductive bias to the model. Where backtracking is possible, one can also use the method to assess the accuracy of the predictions and take corrective action.

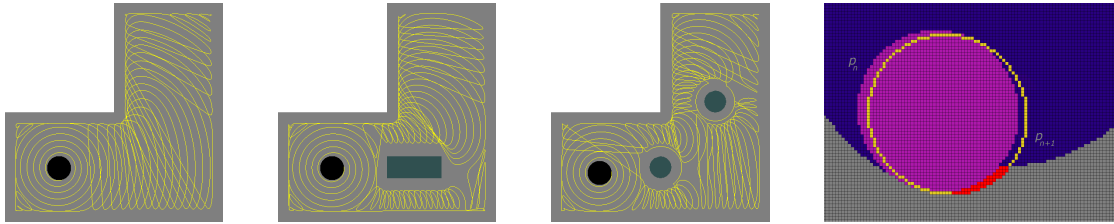
**Keywords:** deep learning, regularization, accuracy assessment, simulation, toolpaths

**DOI:** <https://doi.org/10.14733/cadaps.2025.81-90>

## 1 INTRODUCTION

Simulation-driven techniques are now becoming viable with the increasing availability and affordability of computing power [21]. Simulation-based techniques can replace complex algorithms, trading algorithmic complexity for computational cost. Over the long term, the cost of additional computing power is more than offset by the cost savings from not having to maintain complex software [3]. However, the constant demand for higher fidelity and accuracy in simulations keeps the cost of these techniques high.

In the context of simulation-driven techniques, modern deep learning methods can significantly accelerate the computations involved [4]. However, this is far from trivial to do [19]. DNNs today accelerate fluid flow



**Figure 1:** Post-processed toolpaths for an L-shaped pocket with islands using the method described in [2]. Extreme Right: Pixel-based tool engagement computation.

simulations [16], seismic wave simulations [15], material and micro-structure simulation [18, 13], and molecular system dynamics [22] among other things. While the particulars of the methods vary widely, they all fall under the umbrella of scientific machine learning (SciML). We can classify these SciML techniques based on how they incorporate scientific knowledge [14]. The three broad approaches are - architectural changes, loss functions augmentation, and hybrid methods. Our method adds constraints to a DNN in the form of additional outputs and leans closer to the techniques that modify the loss function.

### 1.1 Application in Computer-Aided Manufacturing

In computer-aided manufacturing, simulation-based methods can be thought of as alternatives to computational geometry-based approaches for generating toolpaths. However, simulation-based techniques are still too compute-intensive for interactive use. In [2], we described a simulation-based method for generating high-speed toolpaths. This paper shows how DNNs can accelerate the method described in [2]. Unlike values computed using analytic methods, a DNN's predictions are not always accurate. We can overcome this inaccuracy by using a fallback technique to calculate the required value when the DNN results are inaccurate. The problem is that there is no straightforward way to know *when* the values predicted by a DNN are incorrect [5, 19].

In toolpath generation, the margin of error must be very low, and using DNNs requires robust uncertainty estimation. By constraining the network to output additional values related to the state of the simulation environment, our method adds a physics-based inductive bias to the DNN and acts as a regularizer. It can also be used to assess the accuracy of predictions. Uncertainty estimation enables us to identify model failures and retrain the model for improved accuracy.

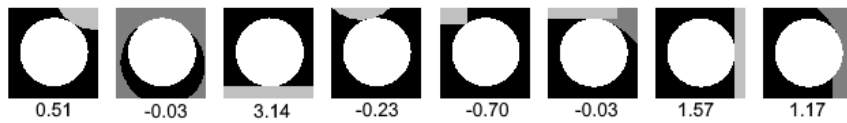
### 1.2 Applications in Other Domains

Applying our technique to similar simulation-driven methods in mechanical engineering and related domains should be relatively easy. In the latter half of the paper, we also theoretically analyze the method. We hope this theoretical understanding can inspire the discovery of similar methods in other domains.

The rest of the paper is organized as follows. In the next section, we describe our method as applied to accelerating a simulation-based method for toolpath generation. This is followed by a theoretical analysis of the method and a comparison to traditional DNN regularizers, skip connections, and physics-inspired neural networks (PINNs). We end the paper with conclusions and suggest directions for extending the current work.

## 2 SIMULATION-BASED TOOLPATH GENERATION

In [2], we described a simulation-driven algorithm for generating high-speed toolpaths for 2.5D pockets (see Fig. 1 for some results). However, the method was computationally too intensive for interactive applications. This paper extends the work done in [2] by accelerating the simulation using a convolutional neural network (CNN).



**Figure 2:** A random sample of input training images (66x66 pixels) with the target tool movement angles.

In [2], we compute the toolpath incrementally as a series of small, fixed-length segments. The orientation of a particular segment is calculated based on the previous segment's direction and the current tool engagement. We compute the next move direction by exploring multiple directions around the previous move direction. In each direction, we calculate the tool engagement value and choose the direction where the engagement is closest to a desired value. Computing the tool engagement value involves capturing small regions of the frame buffer in the neighborhood of the tool and counting the difference in the number of pocket pixels (see Fig. 1 (extreme right)).

We can speed up the simulation process by reducing the number of orientations evaluated at each step. One can do this by estimating the most likely direction for tool movement. However, implementing an analytic algorithm for this can be tedious and error-prone. Today, such problems are best solved using a DNN.

## 2.1 Over-confidence and Accuracy Assessment of DNNs

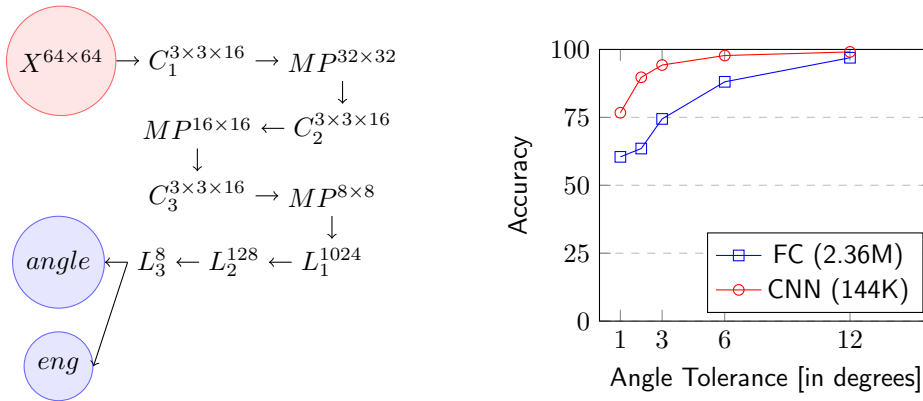
Modern DNNs are often over-parameterized, which increases prediction accuracy but makes the network prone to input memorization. Regularization is therefore required to generalize the network to unseen data. Regularization penalizes complex models and reduces overconfident predictions. However, getting the amount of regularization just right can be challenging. It is also impractical to train a DNN on all possible inputs. A large amount of data is also challenging to obtain and adversely affects the training time. Due to all this, DNNs are rarely calibrated accurately [7], making it difficult to assess their accuracy.

Accuracy assessment of the predictions from a DNN is a long-standing open problem, with much ongoing research [7, 8]. For classification problems, post-processing the penultimate layers of the network can be useful [7, 12]. One can also train the network to recognize outliers and report them during inference [6]. A more generic technique is to use an ensemble of models, where predictions from multiple networks are averaged and analyzed [9, 11].

Our technique is related to similar ideas from physics-guided machine learning [17, 20]. However, instead of explicitly adding constraints to the network that encodes an understanding of the underlying physical process, we take a more data-driven approach that is easier to incorporate into traditional DNN architectures. Our method adds constraints to the network by making the network predict additional parameters derived from the system's current state. These constraints have a regularization effect on the network and can be used for accuracy assessment.

## 2.2 Generating the Training Data

The data for training the DNN is generated from the brute-force method described in [2]. In [2], we capture a small region in the neighborhood of the tool to determine the next move direction. We generate the input data by saving this neighborhood region as an image. For the target data, we compute the angle the chosen tool direction makes with the x-axis. A set of captured input images is shown in Fig. 2 along with the target angles. In these images, the tool is rendered in white, the area outside the pocket is in light gray, the remaining material is in a darker shade of gray, and the already cut material is colored black.



**Figure 3:** Left: CNN model architecture (where the convolutional layers are represented by  $C$ , the  $MP$  represents max-pool layers, and  $L$  is used to denote linear layers). Right: Accuracy of the models at different angular tolerances.

### 2.3 Models and Performance

Since our training data consists of images, it immediately suggests using CNNs. However, the input data occupies a much lower-dimensional space, so even a fully connected network performs equally well, as seen from Fig. 3. Vision transformers can also be used, but they are both slow and difficult to train. Fig. 3 (right) shows the performance of a fully connected (FC) model and a CNN model (the architecture of which is shown in Fig. 3 (left)). At  $\pm 10^\circ$  tolerance, both the FC and CNN models have nearly the same accuracy. However, at 2.36 million, the FC model requires more than an order of magnitude of parameters to reach the same level of performance as the CNN model with only 144K parameters.

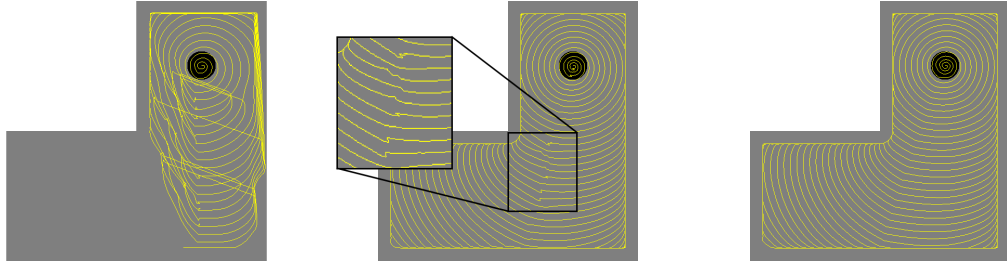
### 2.4 Accuracy Assessment

All existing techniques for computing a confidence score for neural networks are generic methods applicable irrespective of the problem domain. In this paper, we show that for a particular class of problems, there are potentially more robust accuracy assessment methods. Our method trains the network to predict one or more additional target parameters derived from the system's current state. This is equivalent to adding additional terms to the loss function. During training, the sum of the loss across all the targets is minimized. During inference, the system takes an action based on the primary target value predicted by the model. It then compares the computed additional values with the predicted ones to determine the accuracy.

The novelty of our method lies in counterintuitively training the neural network to predict these additional system states. For toolpath generation, the primary target is the tool movement angle, and the additional target is the engagement value. Apart from helping us determine the accuracy of the prediction, an extra value, like the potential tool engagement, adds an inductive bias to the model. A physics-based inductive bias can guide the optimization of the network in the right direction, ensuring that the network learns the right abstractions. Other state parameters, such as the tool's previous move direction, can also be added.

### 2.5 Results

Fig. 4 shows the computed toolpaths when using the CNN model without any fallback and tolerance (left), with fallback and  $\pm 10^\circ$  tolerance (middle), and with accuracy assessment and fallback (right). If we only use the angle predicted by the CNN model, the model fails in many cases (the model's  $\pm 1^\circ$  tolerance accuracy



**Figure 4:** Toolpaths generated using predictions from a DNN - without any tolerance (left), with tolerance (middle), with accuracy assessment and fallback (right).

is only around 77%). The model errors get magnified later when the tool moves over an inaccurately cut section and encounters regions very different from what it has seen in training. With fallback and tolerance (see Fig. 4 (middle)), the algorithm evaluates a  $\pm 10^\circ$  range centered on the direction predicted by the CNN model and chooses the best direction from among these. This approach usually results in an almost usable toolpath requiring smoothing and gouge checks. However, sometimes, such a toolpath can have some sharp corners and kinks where the correct direction may fall just outside the tolerance range or is wrong (see inset Fig. 4 (middle)).

## 2.6 Overall Algorithm

Identifying model errors requires some form of accuracy assessment. To estimate accuracy, we compare the engagement value predicted by the network with that computed when we move in the predicted direction. Suppose there is a significant difference in the predicted and computed engagement values. In that case, we reject the prediction and fallback on the brute-force method where a much larger range of angles  $\pm 90^\circ$  is evaluated. Algorithm 1 shows an outline of the method for predicting the next toolpath step, where the function *PredictDNN* evaluates the CNN model and the *FindBestMove* function is the brute-force function that evaluates all the angles in a given range centered around a given direction; both functions returning an angle and tool engagement pair of values. We can store the identified failure cases and use them to retrain the model in a reinforcement learning loop.

---

**Algorithm 1:** Assessing the accuracy of the move direction predicted by the CNN

---

**Input:**  $\sigma$  // standard deviation of the engagement value difference  
 $ctx$  // the context neighborhood image input  
 $P_{tp} = [p_1, p_2, \dots, p_n]$  // points comprising the toolpath

**Function** FindNextMove( $ctx, P_{tp}$ ):

```

 $(\theta_{pred}, \varepsilon_{pred}) \leftarrow PredictDNN(ctx)$ 
 $\vec{d} \leftarrow (\sin \theta_{pred}, \cos \theta_{pred})$  // direction predicted by the CNN model
 $(\theta_{best}, \varepsilon_{best}) \leftarrow FindBestMove(P_{tp}, \vec{d}, \pm 10^\circ)$ 
if  $|\varepsilon_{pred} - \varepsilon_{best}| > 2\sigma$  then
     $\vec{d} \leftarrow p_n - p_{n-1}$  // direction of the last toolpath segment
     $(\theta_{best}, \varepsilon_{best}) \leftarrow FindBestMove(P_{tp}, \vec{d}, \pm 90^\circ)$ 
end
return  $\theta_{best}$ 

```

---

## 2.7 Further Notes

At first glance, our system may seem similar to multi-target DNNs (MT-DNNs) (see [24]). However, while MT-DNNs try to club multiple tasks into the same model, our approach trains the model for a single task but adds multiple redundant targets. Also, in MT-DNNs, the additional targets are not actionable, and there is no attempt to measure the accuracy of such systems using these additional target values.

In the brute-force version of the simulation-based method as described in [2], each step in the process required, on average, the evaluation of around 180 different angles (from  $-90^\circ$  to  $90^\circ$ , in  $1^\circ$  increments). With a DNN, we can reduce the number of evaluated angles by 9 fold to the range  $-10^\circ$  to  $10^\circ$  centered around the direction predicted by the DNN.

## 3 THEORETICAL ANALYSIS

Most common DNNs (if we ignore details like skip connections) can be thought of as a system composed of a series of non-linear functions like  $f \circ g \circ h \cdots s \circ t(x) = \hat{y}$ . Our method can then be conceptualized as adding one or more additional constraints to the system in the form of similar functions like  $f \circ g \circ h \cdots s' \circ t'(x) = \hat{y}'$ , where the first few functions  $f$ ,  $g$ ,  $h$ , and so on, representing the first few layers of the DNN remain the same, and only the last few layers  $s'$  and  $t'$  here are different (see Fig. 5). Note that each function is parameterized by a set of parameters  $\theta$ , and so the function  $f$  is really  $f(\theta_f)$ . Here, we omit the adjustable parameters  $\theta$  for simplicity.

Another way to look at our method is to see it as adding additional terms to the average loss function. If  $J(\theta, x) = \frac{1}{m} \sum_{i=0}^m L(y, \hat{y})$ , is the average loss over  $m$  training examples, then in our case the total loss function  $J$  is composed of multiple loss terms corresponding to each constraint,  $J = J(\theta_1, x) + J(\theta_2, x) + \cdots = \frac{1}{m} \sum_{i=0}^m L(y_1, \hat{y}_1) + \frac{1}{m} \sum_{i=0}^m L(y_2, \hat{y}_2) + \cdots$ , where there is ideally significant overlap between the set of network parameters represented by  $\theta_1$ ,  $\theta_2$ , etc.. Note that if there is no overlap between the network parameters  $\theta_1$ ,  $\theta_2$ , etc., the constraint branches are then as good as separate networks and provide no inductive bias.



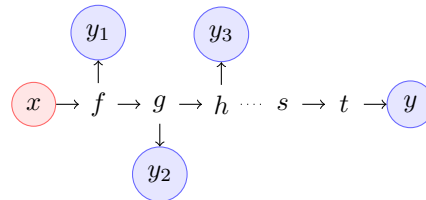
**Figure 5:** (Right) Schematic diagram of a simple neural network interpreted as a composition of non-linear functions. (Left) Adding additional constraints to the network in the form of branches.

When adding the constraints, we have the freedom to add them at one or more layers, nearer to the input or further away from it. Adding a constraint nearer to the input has a regularizing effect only on the layers sandwiched between the input and the constraint (e.g. the layers  $f$ ,  $g$ , and  $h$  in Fig. 5 (right)). One also has the freedom to add additional layers (e.g.  $s'$  and  $t'$  in Fig. 5 (right)) in the branch of the constraint. Where and how to add these constraints will depend on the problem and some trial and error testing to evaluate the performance.

In general, it makes more sense to add constraints based on more abstract state parameters nearer to the input. In the context of our toolpath acceleration example, we would prefer to add the tool engagement constraint (which is closely related to the main required output of the next tool movement direction) nearer to the output of the network. On the other hand, we would rather add a less important constraint, like the tool's previous move direction, nearer to the input. That said, further experiments and analysis are required to come up with a more solid theoretical foundation based on which one can take such decisions.

### 3.1 Regularization Effects

DNNs are often over-parameterized. Over-parameterization is necessary because a DNN needs to have sufficient depth so that it can learn a complex function as a composition of simpler functions. An alternate theory is that over-parameterization helps an optimization method, like the stochastic gradient descent (SGD) method, converge to a global minima [1]. While there is some implicit regularization that is inherent in over-parameterized neural networks [10], it is also important to have some regularization in the network so that the network can be trained without requiring a very large number of training samples [23].



$$Loss = Error(y, \hat{y}) + Error(y_1, \hat{y}_1) + Error(y_2, \hat{y}_2) + Error(y_3, \hat{y}_3) + \dots \quad (1)$$

**Figure 6:** (Right) Schematic diagram of a simple neural network with a skip connection. (Left) The skip connection can be equivalently represented like this.

Our method provides an alternative form of regularization for DNNs. The most common forms of regularization, like the  $L2$  regularization, batch normalization, and dropouts, all add some constraint on the network weights. A similar regularization effect can be achieved by placing various constraints directly as branches at different layers on the network. Such branching will have the same effect as adding multiple additional terms to the loss function. The difference, however, in our case is that these additional loss terms are also dependent on the input to the neural network, whereas the other typical forms of regularization are independent of the input. The main problem with regularizing like this is determining the values of  $y_1$ ,  $y_2$ , etc., that make sense for a particular data set and network.

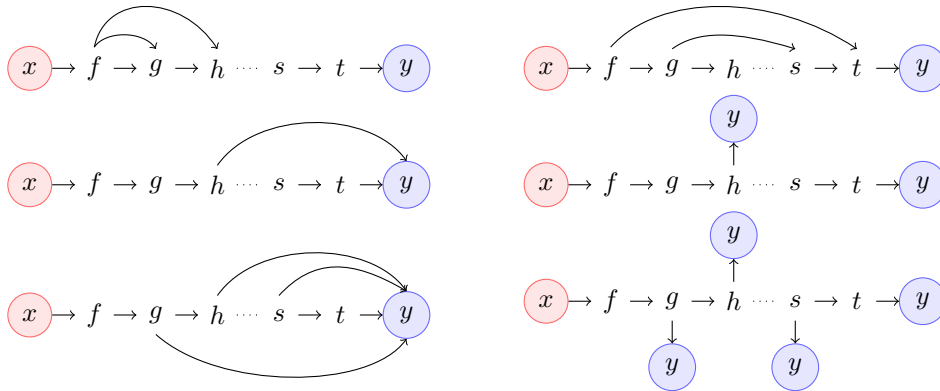
### 3.2 Inductive Bias

In problems where we can use meaningful values of  $y_1$ ,  $y_2$ , etc. we can add an inductive bias to the system. Adding such an inductive bias may not be practical in the case of many problems (especially in non-engineering related domains), where one has little or no idea about the system under investigation, and a DNN is being explicitly used because it is good at discovering the properties of the underlying system. On the other hand, in the case of scientific problems, we often have a deep understanding of the methods and models used to study scientific phenomena. An action performed during the simulation of a scientific process often has some side effects that can potentially be measured, even if they are not always measured and not fed back into the system to simplify calculations.

Almost every physics-based system involves using and solving differential equations. While there is a lot of work going on related to physics-informed neural networks (PINNs), our method suggests a much simpler way of integrating the physics of the system into existing neural network architectures. In the first section of this paper, we have described an application of this type of neural network architecture to the problem of computing CNC toolpaths using a simulation-based method. One can experiment with similar techniques in other domains where one is trying to model some physical phenomena.

### 3.3 Full Skip Connections

When the values of  $y_1, y_2$ , etc. are replaced by the value of the output parameter  $y$ , then the network becomes equivalent to a network with full skip connections, where, by 'full skip connections,' we mean connections from the layers containing the additional constraint branches to the output layer (see Fig. 7). This specialization of the skip connections retains many of the benefits of the traditional skip connections. They can help avoid vanishing gradients, thus making the network learn faster. However, with our generalization of the full skip connection, the layers can be connected to different outputs, thus adding an inductive bias to the system as described in the previous section.



**Figure 7:** (Top) Schematic diagram of a simple neural network illustrating short and long skip connections. (Middle and Bottom Right) Simple neural network with a full skip connection. (Middle and Bottom Left) The full skip connection can be equivalently represented like this.

### 3.4 Accuracy Assessment

When the additional output constraints  $y_1, y_2$ , etc. are different from the primary DNN output  $y$ , then we can sometimes use these output values to assess the accuracy of the predictions made by the DNN. The hypothesis behind this is that the DNN learns (hopefully) different aspect of the complex physical system under investigation. An additional requirement is that we are able to take the measurements for  $y_1, y_2$ , etc. from the simulation of the system. When the DNN predicts a value for  $y$ , we can then take an action based on this prediction in the simulator and measure the values of  $y'_1, y'_2$ , etc. from the system. These values can then be compared with the values of these parameters predicted by the DNN. The difference between the predicted and measured values can be used to estimate the accuracy of the prediction.

Note that the method works only when the DNN has been sufficiently trained and accurate. Assuming a well trained DNN we can further assume that the differences between the predicted and measured additional parameters are normally distributed. For example, if the DNN is 95% accurate then we can mark those predictions as inaccurate which differ, in one or more additional parameters, by more than two standard deviations from the mean (i.e. if  $||y_i - y'_i| - \mu_i| > 2\sigma_i$ , for some additional parameter, where  $\mu_i$  and  $\sigma_i$  are the mean and the standard deviation for that particular parameter). The distribution mean and variance must be computed once after training for each parameter on some test data.



## 4 CONCLUSIONS

In this paper, we have proposed a novel method for the regularization of DNNs by designing the DNNs to predict additional parameters of a physical system in the context of which the DNN is applied. Such a regularization method, when applied to physical simulations, could potentially perform much better than generic regularization methods. Because these additional parameters indirectly encode the physics of the system, such a regularization method also adds a physics-grounded inductive bias to the system enabling the learning of the right abstractions by the DNN. Lastly, in the context of simulations where backtracking is possible, such a method of regularization can also help us estimate the accuracy of the predictions made by comparing the predicted and measured values of the additional parameters.

This regularization method can be extended to other problems where there isn't necessarily a physical simulation environment in place. In such cases an artificial environment can be created and the DNN designed to predict the additional parameters under these artificial constraints. The benefit of such a system is that these additional parameters can then be used for accuracy assessment without having to use an ensemble of models. Further research is required to determine how effective such a system of accuracy assessment will be in other domains. It will also be interesting to explore and investigate the effects of various such artificial constraints on generic problems.

## ACKNOWLEDGEMENTS

This research was conducted as a part of the CAMWorks project. CAMWorks is a popular CAM software that is used by small and medium-sized workshops globally. We are thankful to the team and management of CAMWorks and HCL Software who continue to believe in the importance of sponsoring fundamental research in a variety of domains.

*Tathagata Chakraborty*, <https://orcid.org/0000-0002-2752-2533>

*Christine Zuzart*, <https://orcid.org/0009-0005-3128-4418>

*Chinmaya Panda*, <https://orcid.org/0009-0004-6096-9582>

*Nitin Umap*, <https://orcid.org/0000-0002-9063-1230>

## REFERENCES

- [1] Allen-Zhu, Z.; Li, Y.; Song, Z.: A convergence theory for deep learning via over-parameterization. In International conference on machine learning, 242–252. PMLR, 2019.
- [2] Chakraborty, T.; Panda, C.; Umap, N.: Simulation-driven computation of high-speed pocket machining toolpaths. *Computer-Aided Design*, 21(1), 88–103, 2024. <http://doi.org/10.14733/cadaps.2024.88-103>.
- [3] Dehaghani, S.M.H.; Hajrahimi, N.: Which factors affect software projects maintenance cost more? *Acta Informatica Medica*, 21(1), 63, 2013. <http://doi.org/10.5455/AIM.2012.21.63-66>.
- [4] El-Shamouty, M.; Kleeberger, K.; Lämmle, A.; Huber, M.: Simulation-driven machine learning for robotics and automation. *tm-Technisches Messen*, 86(11), 673–684, 2019. <http://doi.org/10.1515/teme-2019-0072>.
- [5] Gawlikowski, J.; Tassi, C.R.N.; Ali, M.; Lee, J.; Humt, M.; Feng, J.; Kruspe, A.; Triebel, R.; Jung, P.; Roscher, R.; et al.: A survey of uncertainty in deep neural networks. *Artificial Intelligence Review*, 56(Suppl 1), 1513–1589, 2023. <http://doi.org/10.1007/s10462-023-10562-9>.
- [6] Grabinski, J.; Gavrikov, P.; Keuper, J.; Keuper, M.: Robust models are less over-confident. *Advances in Neural Information Processing Systems*, 35, 39059–39075, 2022.

- [7] Guo, C.; Pleiss, G.; Sun, Y.; Weinberger, K.Q.: On calibration of modern neural networks. In International conference on machine learning, 1321–1330. PMLR, 2017.
- [8] Hendrickx, K.; Perini, L.; Van der Plas, D.; Meert, W.; Davis, J.: Machine learning with a reject option: A survey. arXiv preprint arXiv:2107.11277, 2021.
- [9] Ju, C.; Bibaut, A.; van der Laan, M.: The relative performance of ensemble methods with deep convolutional neural networks for image classification. *Journal of applied statistics*, 45(15), 2800–2818, 2018. <http://doi.org/10.1080/02664763.2018.1441383>.
- [10] Kubo, M.; Banno, R.; Manabe, H.; Minoji, M.: Implicit regularization in over-parameterized neural networks. arXiv preprint arXiv:1903.01997, 2019.
- [11] Lakshminarayanan, B.; Pritzel, A.; Blundell, C.: Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.
- [12] Mandelbaum, A.; Weinshall, D.: Distance-based confidence score for neural network classifiers. arXiv preprint arXiv:1709.09844, 2017.
- [13] Mendizabal, A.; Márquez-Neila, P.; Cotin, S.: Simulation of hyperelastic materials in real-time using deep learning. *Medical image analysis*, 59, 101569, 2020. <http://doi.org/10.1016/j.media.2019.101569>.
- [14] Moseley, B.: Physics-informed machine learning: from concepts to real-world applications. Ph.D. thesis, University of Oxford, 2022.
- [15] Moseley, B.; Nissen-Meyer, T.; Markham, A.: Deep learning for fast simulation of seismic waves in complex media. *Solid Earth Discussions*, 2019, 1–23, 2019. <http://doi.org/10.5194/se-2019-157>.
- [16] Obiols-Sales, O.; Vishnu, A.; Malaya, N.; Chandramowlishwaran, A.: Cfdnet: A deep learning-based accelerator for fluid simulations. In *Proceedings of the 34th ACM international conference on supercomputing*, 1–12, 2020. <http://doi.org/10.1145/3392717.3392772>.
- [17] Pawar, S.; San, O.; Aksoylu, B.; Rasheed, A.; Kvamsdal, T.: Physics guided machine learning using simplified theories. *Physics of Fluids*, 33(1), 2021. <http://doi.org/10.1063/5.0038929>.
- [18] Peivaste, I.; Siboni, N.H.; Alahyarizadeh, G.; Ghaderi, R.; Svendsen, B.; Raabe, D.; Mianroodi, J.R.: Accelerating phase-field-based simulation via machine learning. arXiv preprint arXiv:2205.02121, 2022.
- [19] Rackauckas, C.V.; Abdelrehim, A.: Scientific machine learning (sciml) surrogates for industry, part 1: The guiding questions. <http://doi.org/10.31219/osf.io/p95zn>.
- [20] Raissi, M.; Perdikaris, P.; Karniadakis, G.E.: Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378, 686–707, 2019. <http://doi.org/10.1016/j.jcp.2018.10.045>.
- [21] Shalf, J.: The future of computing beyond moores law. *Philosophical Transactions of the Royal Society A*, 378(2166), 20190061, 2020. <http://doi.org/10.1098/rsta.2019.0061>.
- [22] Vlachas, P.R.; Zavavlav, J.; Praprotnik, M.; Koumoutsakos, P.: Accelerated simulations of molecular systems through learning of effective dynamics. *Journal of Chemical Theory and Computation*, 18(1), 538–549, 2021. <http://doi.org/10.1021/acs.jctc.1c00809>.
- [23] Wei, C.; Lee, J.D.; Liu, Q.; Ma, T.: Regularization matters: Generalization and optimization of neural nets vs their induced kernel. *Advances in Neural Information Processing Systems*, 32, 2019.
- [24] Zeng, Z.; Liang, N.; Yang, X.; Hoi, S.: Multi-target deep neural networks: Theoretical analysis and implementation. *Neurocomputing*, 273, 634–642, 2018. <http://doi.org/10.1016/j.neucom.2017.08.044>.