Proceedings of the ASME 2023
International Design Engineering Technical Conferences and
Computers and Information in Engineering Conference
IDETC-CIE2023
August 20-23, 2023, Boston, Massachusetts

DETC2023-116429

HEAT SINK DESIGN OPTIMIZATION VIA GAN-CNN COMBINED DEEP-LEARNING

Nathan Flynn, Xiaoping Qian*
n8flynn@outlook.com

Department of Mechanical Engineering
University of Wisconsin-Madison, Madison, WI

ABSTRACT

This work proposes a combined deep learning based approach to improve thermal component heat sinks involving turbulent fluid flow. A Generative Adversarial Network (GAN) is trained to learn and recreate the new ellipse based heat sinks. Simulation data for new designs is efficiently generated using OpenFOAM 7 (Open Source Computational Fluid Dynamics software) along with high throughput computing. To improve the speed of design evaluation, a Convolutional Neural Network (CNN) is trained to predict the entire temperature field for a given design. The trained CNN is able to predict the entire temperature field for the design with a mean average error of 1.140 degrees kelvin in 0.04 seconds (22,500 times faster than the simulation). A combined model is formed using the trained CNN and GAN networks to create and simulate new designs. The combined model optimizes the latent representation of 64 random designs on a Graphical Processing Unit (GPU) in ten minutes. The optimized designs perform fourteen degrees kelvin better on average than the non-optimized designs. The highest preforming design outperforms any design in the training data by 1.83 *degrees kelvin.*

1. INTRODUCTION

Improving the performance of heat sinks via design optimization is important to be able to more effectively pull heat away from demanding electrical components such as processors. Typical approaches to this problem involve using an optimization scheme such as shape [1] or topology optimization [2] to improve the performance of a given design. Shape and topology optimization involves optimizing a design using an objective, such as minimizing pressure drop in a fluid channel. Shape optimization optimizes the shape of an initial design, while topology optimization creates large topological changes from the initial design. To evaluate the objective function the design must be simulated whenever the design is updated, which can be computationally expensive over many iterations.

*Corresponding author: Email: qian@engr.wisc.edu

Another approach to optimizing fluid related problems involves the use of deep-learning methods such as regression based surrogate models which allow the prediction of an output based on an input such as the channel length in a fluid problem [3]. Surrogate models use machine learning models to learn how to relate the input of a simulation to an output to replace the simulation. The surrogate models are then paired with a genetic algorithm or optimization method to improve the performance of the design by changing some portion of the geometry, such as optimizing the length of the baffles in a micro-channel [4]. The drawback of this approach is that full simulations are required to be able to predict only a few of the desired outputs (temperature of the outlet, pressure drop, etc) but not other desired outputs (entire solution field for temperature or pressure) and the geometries have to be relatively simple.

Recent advances in deep learning have created models such as the Generative Adversarial Networks (GAN) which are able to create new designs by learning the distribution of a given set of input designs [5]. With GAN's ability to generate new data and designs, they have been recently paired with topology optimization-based approaches to produce new optimal structures. In another paper, a GAN is used in combination with transfer learning to create topologically-optimized designs to seen and un-seen boundary conditions while testing the model [6]. In another paper, previously generated topological designs are used alongside new designs generated by a GAN to generate aesthetically optimal and high performance designs [7]. Lastly, a Variational Auto-Encoder GAN (VEGAN) is used alongside a database of optimized airfoil designs to be able to create new designs that have equivalent or slightly better performance than the training data [8]. The issues with these TO-based models is that rely on high performance data, either from a data base or data created through topology optimization.

Recently, a GAN and Convolutional Neural Network (CNN) combined based approach is able to produce optimal microstructure based designs by using inverse design based method to achieve a desired compliance [9]. The method developed by

Tan et al. [9] is successfully used to generate training micro structure based images from scratch, generate new realistic data with a GAN and then predict the performance of a given design using a Le-Net CNN architecture. This method is applied to a relatively simple problem and inverse design produces desired structures.

In this paper, a similar combined deep learning approach is applied to a conjugate heat-transfer design optimization problem. The problem involves optimizing the design of a heat sink using both steady state and turbulent air flow. The GAN is used to create new designs from a set of randomly generated fin structures with elliptical cross sections using a Python library called "ellipse packing" [9]. The CNN evaluates the performance of the design by predicting the entire temperature field for a given design. The two models are separately trained and then combined into a single model. The combined model is used along side an optimizer to directly improve the performance of a batch of 64 randomly generated heat sink designs.

2. DESIGN PROBLEM

The design problem is a turbulent ($Re = 4520$), steady state, conjugate heat-transfer problem with conduction and convection. The fluid domain is a 74 x 66 mm rectangle and the design domain is a 64 x 64 mm square that is centered within the fluid domain. A graphical representation of the problem, initial and boundary conditions is shown in Fig. 1.

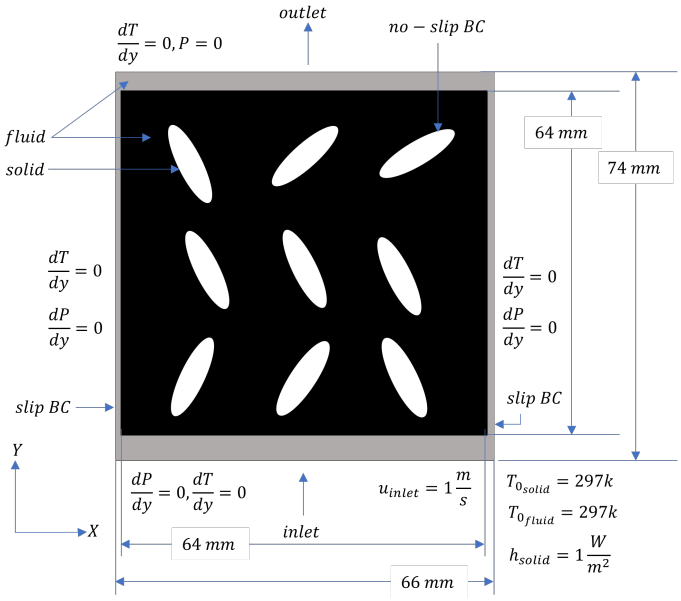

FIGURE 1: DESIGN PROBLEM PARAMETERS

Fluid flows from the inlet at the bottom to the outlet at the top. The white ellipses within the design domain represent the solid fins of the heat sink. The goal of this project is to generate optimal structures that dissipate a 1 W/m^2 load on the heat sink. The fluid is air and the solid is 6061 Aluminum Alloy (Al). The material properties for the solid and fluid are shown in Table 1. Performance of a given design is measured by calculating the average solid temperature. The best design has the lowest average solid temperature.

TABLE 1: DESIGN PROBLEM MATERIAL PROPERTIES

Parameters	Solid	Fluid
Material	6061 Al	Air
Molecular Weight (g/mol)	63.5	28.966
$\rho\ (kg/m^3)$	2719	1.225
$c_p\ (J/kg/K)$	871	1006.43
$\mu\ (kg/m/s)$	-	$1.7884e-05$
Pr	-	0.71

3. PROPOSED METHOD

The proposed method to solve this problem involves using a combined deep-learning optimization framework composed of a design generator and a design evaluator. The design generator (g) creates new designs (X) from a given input latent space vector (a compressed representation of an image) z shown in Eq. (1).

$$X = g(z) \tag{1}$$

The design evaluator, f, uses the design (X) as the input and outputs the temperature field (Y) for that design shown in Eq. (2).

$$Y = f(X) \tag{2}$$

The combined model consists of both the design generator (g) and the design evaluator (f) used in combination shown in Eq. (3).

$$Y = f(g(z)) \tag{3}$$

The loss function for the combined model is the average temperature shown in Eq. (4).

$$Loss = \frac{\sum (Y \cdot X)}{\sum X} \tag{4}$$

The Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP) [10] creates the generator model in Eq. (1) and the U-NET CNN model [11] creates the evaluator model in Eq. (2). Training of both models is done separately. The WGAN-GP is trained on images of randomly generated fin designs with elliptical cross sections. After training, the WGAN-GP is used to create new designs by passing a random latent space vector (z) into the GAN. Simulation is preformed on the new designs using OpenFOAM 7 (an open-source computational fluid dynamics package) to produce the temperature field for a given design. The U-NET maps the designs to their corresponding temperature field. Once training for both models is complete, a combined model is formed using the generator of the GAN and the U-NET CNN. Optimization is performed on the input to the generator (z) using Average Stochastic Gradient Decent (ASGD) within a Python library called PyTorch to minimize Eq. (4)

3.1 Design generation

To generate heat sink designs, an open source Python library called "ellipse packing" from GitHub creates random elliptical structures [12]. The number of mesh points in the x and y direction and the scale of the major and minor axes is controlled with the library. Examples shown in Fig. 3. The placement of the

ellipses is random and is created by producing an evenly space grid of points. Next, Delaunay Triangulation is used to form a mesh of triangles. Finally, formation of Steiner In-ellipses is accomplished by inscribing the ellipses within the triangles shown in Fig. 2.

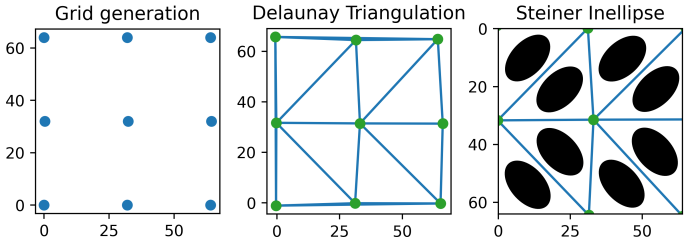

FIGURE 2: DESIGN GENERATION METHOD

Each image is gray-scale and has a resolution of 64 x 64 pixels. To generate 100,000 unique designs, randomization is done using the NumPy random function between the following settings; number of grid points in the x and y direction between 2 and 10, the scale of the major axis between 0.5 and 2.75, and the minor axis between 0.5 and 1.25.

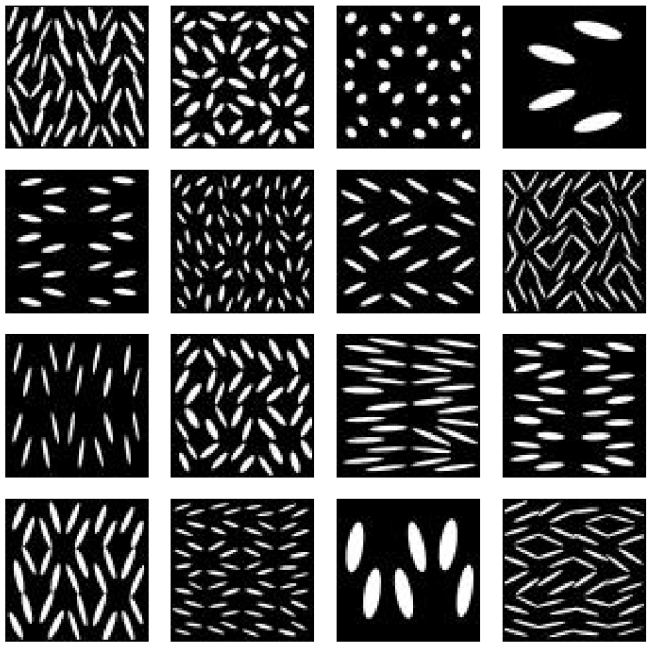

FIGURE 3: EXAMPLE ELLIPSE FIN DESIGNS

3.2 Design simulation

Temperature field data for the design evaluator is generated using simulations from OpenFOAM 7 to produce the temperature field for each design. The kEpsilon model is used to simulate turbulent fluid flow. Conversion of the designs into a mesh friendly format for OpenFOAM is accomplished by filtering the gray-scale images to a solid (1) and fluid (0) representation. The gray-scale images have pixel value ranges from 0 to 255, where 255 and 0 represent a solid and fluid pixel respectively. Any pixel value

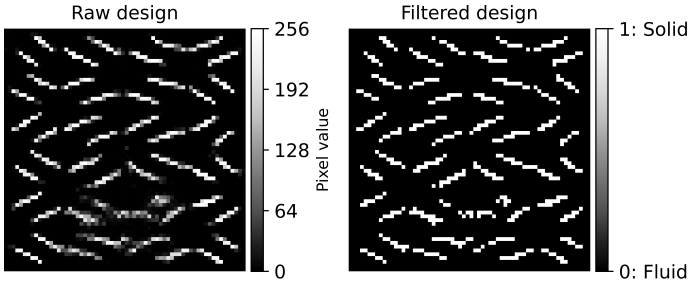

FIGURE 4: IMAGE FILTERING

greater than or equal to 90 is changed to 1 and any pixel value less than 90 is changed to 0. An example is shown in Fig. 4. The solid/fluid threshold of 90 is selected by trial and error. The purpose of this threshold is to ensure that no fluid pixels ended up inside of the solid domain. If this happens, the simulation fails to run. Image processing is applied on-top of filtering to fill in potential voids using the SciPy Python library. The binary fill

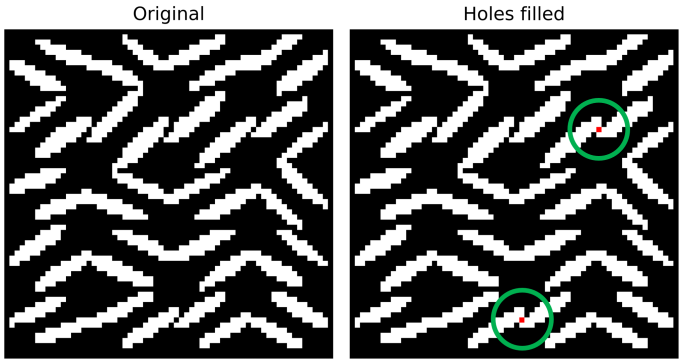

FIGURE 5: FILLING VOID EXAMPLE

holes function within SciPy invades the complement of the image from the boundary of the image using binary dilation. Once the invasion step is complete, only the holes remain in the image because there is no connection from them to the boundary. A final step is done to fill in the holes to create a void free design [13] shown in Fig. 5 where the red pixels denote filled in holes. The design (64 x 64) is then added to the center of the fluid domain (74 x 66) and then is meshed in OpenFOAM 7. To improve the

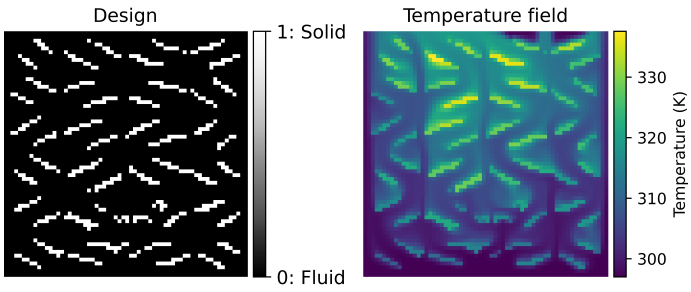

FIGURE 6: DESIGN AND TEMPERATURE FIELD

accuracy of the simulation, the mesh is refined to increase the number of cells from 4884 (74 x 66) to 19,536 (148 x 132). The output temperature field of the simulation is then mapped back

into the 74 x 66 domain and the fluid region is trimmed off to leave the 64 x 64 temperature field. An example of the input and output of the simulation is shown in Fig. 6.

3.3 Design performance metric and benchmark

The average solid temperature is used to record performance, EQ. (5). In this equation, Y represents the temperature field prediction, and X_{solid} is the filtered version of X only containing solid pixels, example is shown in Fig. 4.

$$T_{avg_{solid}} = \frac{\sum (Y \cdot X_{solid})}{\sum X_{solid}}, \quad X_{solid} = \begin{cases} 1 & X \geq \frac{90}{255} \\ 0 & X < \frac{90}{255} \end{cases} \tag{5}$$

A simple parallel fin design is constructed and simulated as a benchmark to compare to the results from the combined model. The parallel fin design and performance is shown in Fig. 7.

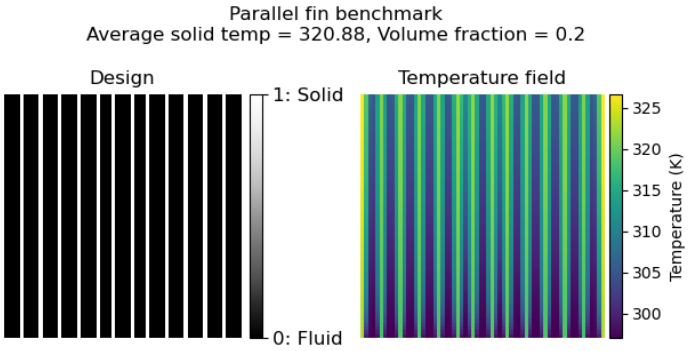

FIGURE 7: PARALLEL-FIN BENCHMARK

3.4 Scaling simulation runs

Each simulation takes about 2 minutes to run on a single core (Intel 6700K at 4.00 GHz). Assuming 2 minutes per run, conducting 500,000 simulations using a single core would take around 2 years. The Center for High Throughput Computing (CHTC) at the University of Wisconsin-Madison is used to generate simulation data. The Center has over 20,000 cores available and is managed using HTCondor software. Using this resource for data generation made it possible to complete this project in a reasonable amount of time [14]. An OpenFOAM 7 Docker image is used to run the simulation at the CHTC [15]. To balance out the workload on all of the servers nodes, the simulations need to be broken up into smaller amounts of work. For a batch of 500,000 simulations, creation of 5,000 jobs is required, with each job containing 100 simulations worth of work. Each job runs on a single node of the server, allowing results for 500,000 simulations to be generated in less than a day.

3.5 Design generator

A GAN is used to create new designs from a set of existing designs. A traditional GAN is made up of two components, a generator and a discriminator. The discriminator is trained using existing images and the generator produces images that look similar to the existing images. The discriminator scores the generated images on the probability that the image came from the existing image data set [5]. Training of the two models is accomplished by playing a min-max game. If the generator creates an image that fools the discriminator, the discriminator learns from that image and improves the discriminator model. Similarly, the generator model improves when the discriminator rejects images created by the generator due to the generated images being too different from the training set of images. GANs take a random uniform distribution vector between 0 and 1 called the latent space z. GANs learn how to decompress this latent space into the full design space (latent space vector 100D to 64x64 image). Designs can be changed by moving within the latent space and certain areas of this space are attributed to certain features in the images. In this paper the latent space is a 100D vector similar to the work done by Lipton et al. [16].

The Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP) [10] is used to generate new designs (implementation by [17] in Pytorch). The WGAN-GP is the successor to the Deep Convolutional GAN (DCGAN) which aims at eliminating common problems with the network; model collapse and vanishing gradients. Model collapse is a common issue within GAN's and causes the model to produce the same group of images. The WGAN-GP reduces the chance of model collapse by using a different loss formulation. Wasserstein loss uses Earth Mover Distance (the minimum cost to transport the mass of one distribution into another to make them the same). The gradient penalty (GP) is added to the loss formulation to add more stability than the original WGAN alone [10].

Use of the conventional loss function for the generator and the critic is unreliable to gauge model convergence due to both networks playing a min-max game during training causing the loss function for both the generator and the discriminator to fluctuate. The other way to gauge model performance is through visual inspection of the images. The challenge with visual inspection is that it is a qualitative approach. A new set of quantitative metrics emerged to remedy this issue called Inception Score (IS) [18] and Fréchet Inception Distance (FID) [19]. The Inception Score is used to evaluate the quality of the images being produced. Fréchet Inception Distance is used to measure the diversity of the images. A Python library called pytorch-gan-metrics is used to calculate both the FID and IS scores [20].

To train the GAN, simulation is preformed on 100,000 designs and evaluated using Eq. 5. From the pool of designs, the GAN training set of designs is made up of 50K randomly selected designs. Examples of training data is shown in Fig. 3. The original implementation from [17] is used along side a learning rate value of $1e-4$ for both the critic and generator. The decreased learning rate along with the other original settings is found to produce the best diversity in the designs.

Once the WGAN-GP is trained the generator from the model is separated and the last layer of the model is changed from the hyperbolic tangent function (tanh) to the sigmoid function. The tanh function is used during training to restrict the output of the GAN between -1 and 1. The sigmoid function replaces the tanh function to restrict the output between 0 and 1. A comparison is done for the same image using the Sigmoid function and scaling the tanh functions output between 0 and 1 show in Fig. 9. The Sigmoid function produces similar images to the scaled tanh function and reduces the number of computational steps to scale

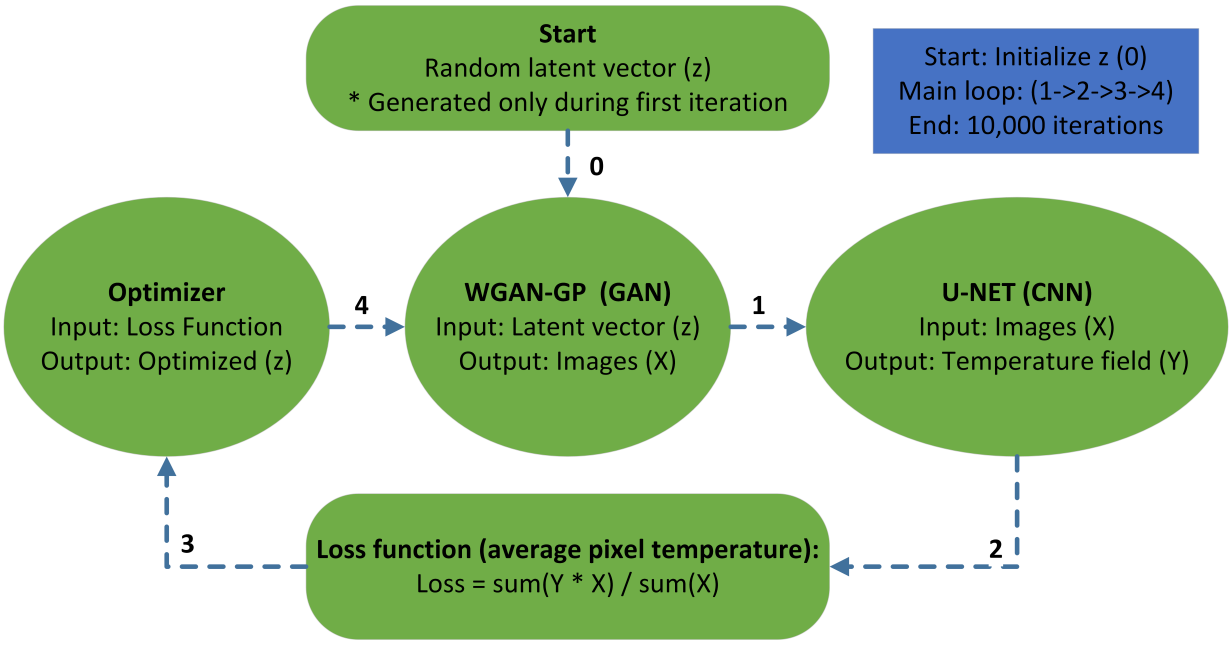

FIGURE 8: COMBINED MODEL FLOWCHART

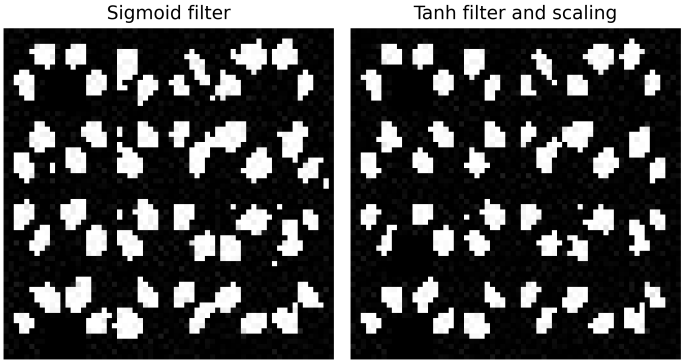

FIGURE 9: IMAGE FILTERING COMPARISON

the image between 0 and 1. A random latent space vector is generated between 0 and 1 using the PyTorch random function and then is fed into the generator to create new designs. In total, the generator created 750,000 new designs to train the evaluator with the designs corresponding simulation data.

3.6 Design evaluator

To evaluate all of the WGAN-GP's generated designs a CNN is used to predict the temperature field of the design. The U-NET CNN architecture is selected for its ability to execute pixel by pixel based regression. It was originally designed for fast biomedical pixel based segmentation tasks [11]. Since its inception, the U-NET architecture has been used for a variety of engineering based tasks. Recently, the U-NET is used to predict a temperature field for various heat source intensities, sizes, and layouts within a 200 x 200 grid problem [21]. Later, the same model is used along side an optimization framework to create the best heat-source layout for various electronics [22]. In this work, the U-NET CNN takes the 64 x 64 input design in the form of an image and outputs a 64 x 64 temperature field prediction. The implementation from [21] is used to create the model within PyTorch.

Mean absolute error (MAE) is used to keep track of the U-NET's performance during training, shown in Eq. (6) where N represents the mesh size. In this case is 64 from the 64 x 64 design. MAE represents the average prediction error per pixel in terms of Kelvin (K). In Chen et al. [21] they were able to achieve a MAE between 0.02 and 0.5 K depending on the complexity of problem.

$$MAE = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\left|\hat{Y}_{ij} - Y_{ij}\right| \tag{6}$$

Before training the network, the temperature field data is normalized to aid in training. The temperature field data has values that range from 298K to 1100K. Using equation Eq. (7) with a $X_m = 100$ and $X_{std} = 297\ K$, the data is normalized between 0 and 1. A similar approach is done by Chen eta. [21] to train the U-NET on a heat source layout problem.

$$X_0 = \frac{X - X_m}{X_{std}} \tag{7}$$

To train the network, simulation of the 750K GAN generated designs is done using HTC to produce temperature field data. Of the 750K sets of data, only 550K sets is used to train the U-NET CNN. With 50K of the data being used to test the model after training. Of the 500K sets of data, 80% (400K) is used to directly train the model and the remaining 20% (100K) is used to evaluate the MAE during training (validation data). A surplus of 200K designs is used as a margin of safety. Occasionally, the GAN produces designs with fluid inclusions in solid regions, causing the simulation to fail without image processing to fill in the voids.

3.7 Combined Model GAN + CNN

Once training of both WGAN-GP and U-NET is completed, the generator from the WGAN-GP and the U-NET model is combined to form the model shown in Fig. 8. The combined model functions as follows: a random latent space variable is generated with 64 designs and then the latent space variable z is passed into generator of the GAN. The output of the GAN (a set of images) is passed directly into the U-NET as an input as shown in Eq. (3). To optimize the latent space variable (z) the loss function below in Eq. (4) is used. This loss function approximates the average temperature of each pixel. In this formulation, the design X is not filtered to decrease computational time. The ASGD optimizer from PyTorch is used with a learning rate of $9e^{-3}$ to optimize the latent representation of the designs. The loss is back propagated into z and z is optimized to minimize the loss function. Stochastic gradient clipping is employed on the latent space variable after each optimization step to improve convergence [23]. Post processing is preformed on the designs using the binary fill holes function to fill in any voids in the designs.

4. RESULTS

4.1 GAN

Training the GAN with 50K images took about 8 hours on an RTX 4080. To monitor convergence of the GAN, Inception Score (IS) and Frechet Inception Distance (FID) values are computed every epoch as shown in Fig. 10. Inception Score measures the diversity of the images, while Fréchet Inception Distance measures the quality of the images. The convergence of training the GAN is monitored using both metrics (IS and FID). Training of the GAN is stopped when both metrics plateau. The GAN is trained for a total of 200 epochs with an image batch size of 64. A version of the model parameters is saved for every iteration and the final model version is selected based on the model having the smallest IS and highest FID score possible. This happens to be the model at iteration 155, which is denoted by the blue star in Fig. 10.

Once training the WGAN-GP is completed, the generator

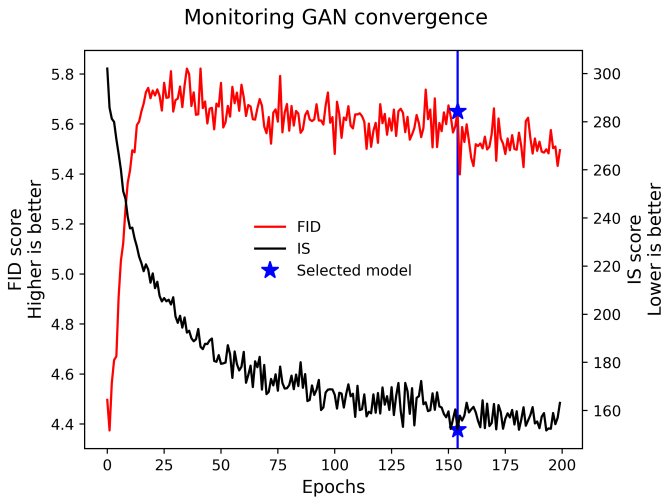

FIGURE 10: GAN TRAINING EVALUATION

model is separated from the Network and generates 750K new designs by feeding random noise into the generator via the latent space z, examples shown in Fig. 11. The images appear to be

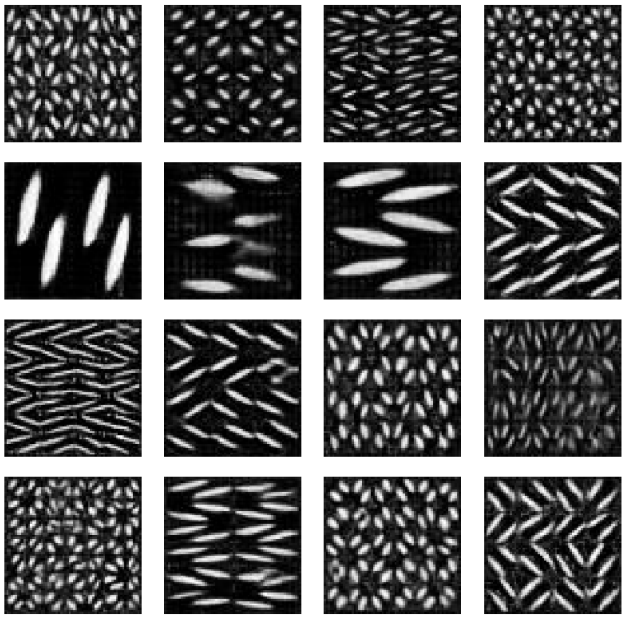

FIGURE 11: GAN GENERATED DESIGNS

nosier that the original training data because GANs approximate the original training data. Image filtering on the images still produces meaningful and high performance designs.

The volume fractions of the designs produced by the GAN and the training data is compared to training data in Fig. 12, computed using EQ (8). A large portion of the

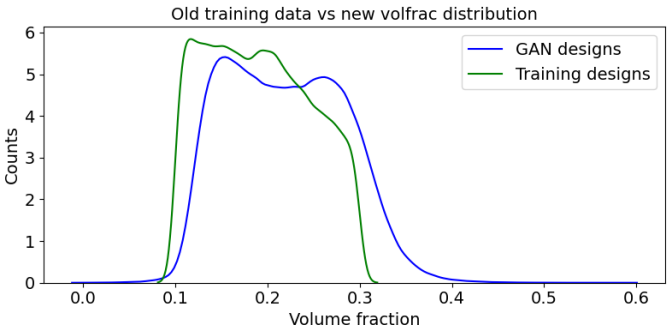

FIGURE 12: VOLUME FRACTION DISTRIBUTION

designs in the training data have a volume fraction between 0.1 and 0.3. While the GAN's designs seem to have a larger emphasis on volume fractions falling in the range of 0.15 to 0.35. This is most likely due to the GAN's images being more grainy than the original images, causing the volume fraction to increase. The GAN produced a small amount of designs outside of the training data volume fraction range (0 to 0.1 and 0.3 to 0.6).

$$Vol_{frac} = \frac{\sum X_{solid}}{64 * 64}, \quad X_{solid} = \begin{cases} 1 & X \geq \frac{90}{255} \\ 0 & X < \frac{90}{255} \end{cases} \tag{8}$$

The average solid temperatures of the training data and the output of the GAN is shown in Table 2. The GAN is able to create

TABLE 2: GAN DESIGN PERFORMANCE

Average solid temperature	Average [K]	Best [K]	Worst [K]
Training data	333.62	312.56	499.87
GAN	335.27	312.23	1117.75

designs that preformed two degrees worse on average compared to training data and produced a design that is slightly better than the best design in the training data. The volume fraction of the design versus its temperature is shown in Fig. 13 (the data is truncated below 500 K to zoom in on the majority of the data). The scatter plot shows that volume fraction isn't a good indicator of design performance and that placement, spacing, and orientation of the ellipses is important.

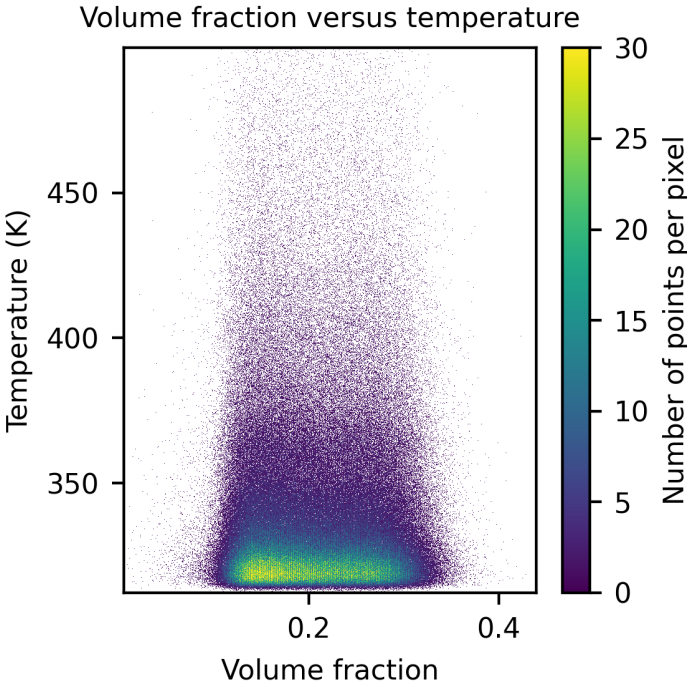

FIGURE 13: DESIGN VOLUME FRACTION VS TEMPERATURE

Velocity plots for the worst (avg temp of 1053.69 K), middle (avg temp of 325.79 K), and top 3 designs (avg temp of 312.39 K) from each category shown in Fig. 14. The top row is the best designs, the middle row is the mediocre designs, and the bottom row represents the worst designs. The red coloring indicates high fluid velocity and blue represents low fluid velocity. The worst designs comprise of very few fins and did not take advantage of the fluid flow, as most of the fluid went past the fins. The middle preforming designs made better use of the fluid flow and had smaller fins staggered evenly. The best preforming designs had small evenly spaced fins with even flow throughout the heat sink.

4.2 U-NET

Training the CNN took around 8 hours on an RTX 4080. Mean average error (MAE) is employed on the validation data to

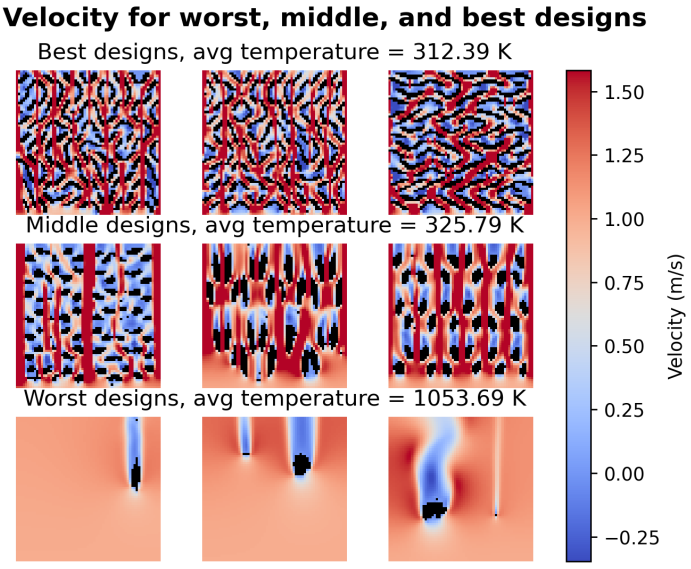

FIGURE 14: VELOCITY FOR THE GAN'S BEST, MEDIOCRE, AND WORST DESIGNS

keep track of how the model is doing. The loss for the training model is shown in Fig. 15. The model is able to achieve an

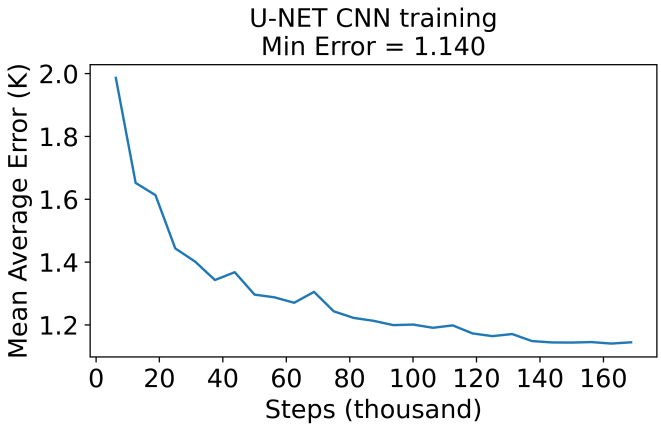

FIGURE 15: CNN TRAINING RESULTS

MAE of 1.140 K, about double the results for the complicated heat source layout case from [21]. Once training the model is complete, it can produce accurate predictions shown in Fig. 16. The left most figure is the design itself, the middle figure is the true temperature field, and the right most image is the predicted

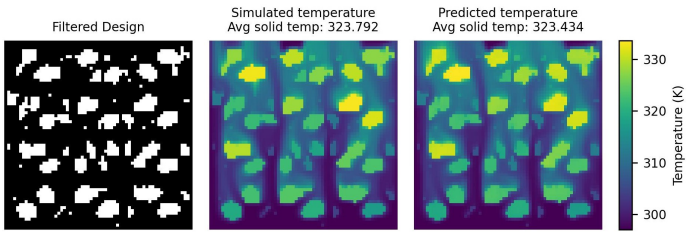

FIGURE 16: EXAMPLE OUTPUT OF U-NET CNN

temperature field. Averaged over the 10,000 iterations, the CNN took about 0.004 seconds to make a single temperature field prediction. On average, the simulation took about 90 seconds to run. The CNN ran about 22,500X faster than the simulation, making it an excellent tool to run optimizations (optimization requires many solutions) or explore the performance of many designs quickly.

4.3 Optimization results

Using the combined architecture shown in Fig. 8, the WGAN-GP generates 64 designs by passing a randomly generated latent variable z from the PyTorch rand function between 0 and 1 (latent space has a shape of 64x100x1x1) into the GAN. The GAN then decompresses the latent variable z into full scale designs (from 64x100x1x1 to 64x1x64x64). Optimization is preformed on the latent space representations of the designs using ASGD within PyTorch. Optimization takes 10 minutes on the GPU over the course of 10,000 iterations. Simulation is preformed on the original and optimized designs to get the before and after temperatures shown in Table 3.

TABLE 3: DESIGN OPTIMIZATION RESULTS

Average solid temperature	Average [K]	Best [K]	Worst [K]
Before optimization	336.05	315.35	441.88
After optimization	321.96	310.73	453.11

On average, the optimized designs improved by fourteen degrees kelvin. Optimization is also able to produce a design better than the training data with an average temperature of 310.73K. The best design from optimization out-preformed the best design in the 550K designs generated by the GAN to train the CNN or the GANs training data by 1.83 degrees K. The worst design ended up getting worse after optimization going from 441.88K to 453.11 K. A visual representation for the optimization improvement can be seen in Fig. 17 where optimization improves the performance of the majority of the designs.

Extraction is preformed on highest preforming design from the 64 optimized designs. The before and after design is shown in Fig. 18. The left side of the figure is the design before optimization and the right side is after optimization. The design in the top changes to minimize the hot-zone in the original design. The hot zone size after optimization is decreased and the overall scale of the temperatures is decreased as well.

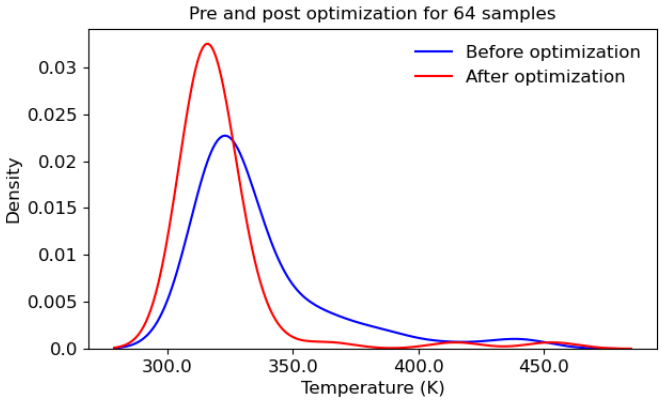

FIGURE 17: TEMPERATURE DISTRIBUTION BEFORE AND AFTER OPTIMIZATION

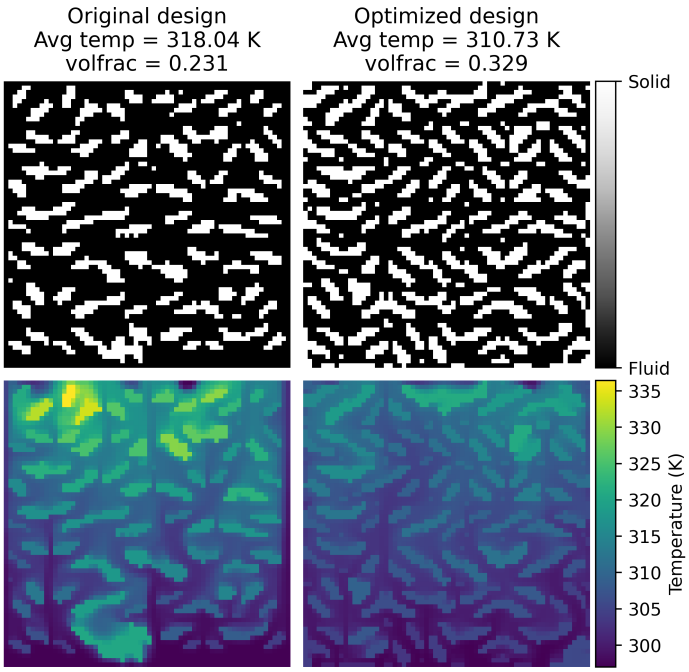

FIGURE 18: BEST DESIGN OPTIMIZATION BEFORE AND AFTER

5. CONCLUSION

In this work, we propose a combined GAN and CNN approach based on a deep learning model that is capable of producing improved designs from poor-performance training data on a conjugate heat transfer optimization problem. The WGAN-GP model learns from a wide distribution of images from the training data with acceptable quality without experiencing model collapse. The U-NET CNN model is able to predict the corresponding temperature field with an MAE of 1.140K with a 22,500x speed up compared to the simulation model. Combining the generator from the GAN and CNN form the combined model and optimization is preformed on the latent space representation of the designs with an average improvement of fourteen degrees over 64 non-optimized designs. The combined model is able to produce designs that out-preformed the training data. The best design had an average temperature of 310.73 K, 1.83 K better than any of the designs in the training data.

In future work, the combined model could be applied to a higher resolution images 512 x 512, or even 3D designs by changing the layers in the GAN and CNN. Finally, this combined model could be applied to a different set of fluid or engineering related problems.

ACKNOWLEDGMENTS

The authors want to acknowledge the support from NSF grants #1941206 and #2219931.

REPRODUCIBILITY

Code can be found at https://github.com/FlynnDesigns/Research

REFERENCES

[1] Zhang, Ruochun and Qian, Xiaoping. "Parameter-free Shape Optimization of Heat Sinks." *2020 19th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*: pp. 756–765. 2020. DOI 10.1109/ITherm45881.2020.9190501.

[2] Sun, Sicheng, Liebersbach, Piotr and Qian, Xiaoping. "3D topology optimization of heat sinks for liquid cooling." *Applied Thermal Engineering* Vol. 178 (2020): p. 115540. DOI https://doi.org/10.1016/j.applthermaleng.2020.115540. URL https://www.sciencedirect.com/science/article/pii/S1359431120330222.

[3] Jiang, Ping, Zhou, Qi and Shao, Xinyu. *Surrogate model-based engineering design and optimization*. Springer (2020).

[4] Shi, Haoning, Ma, Ting, Chu, Wenxiao and Wang, Qiuwang. "Optimization of inlet part of a microchannel ceramic heat exchanger using surrogate model coupled with genetic algorithm." *Energy Conversion and Management* Vol. 149 (2017): pp. 988–996. DOI https://doi.org/10.1016/j.enconman.2017.04.035. URL https://www.sciencedirect.com/science/article/pii/S019689041730345X.

[5] Goodfellow, Ian J., Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron and Bengio, Yoshua. "Generative Adversarial Networks." (2014). DOI 10.48550/ARXIV.1406.2661. URL https://arxiv.org/abs/1406.2661.

[6] Behzadi, Mohammad Mahdi and Ilieş, Horea T. "GANTL: Toward Practical and Real-Time Topology Optimization With Conditional Generative Adversarial Networks and Transfer Learning." *Journal of Mechanical Design* Vol. 144 No. 2 (2021). DOI 10.1115/1.4052757. URL https://asmedigitalcollection.asme.org/mechanicaldesign/article-pdf/144/2/021711/6806350/md_144_2_021711.pdf, URL https://doi.org/10.1115/1.4052757. 021711.

[7] Oh, Sangeun, Jung, Yongsu, Kim, Seongsin, Lee, Ikjin and Kang, Namwoo. "Deep Generative Design: Integration of Topology Optimization and Generative Models." *Journal of Mechanical Design* Vol. 141 No. 11 (2019). DOI 10.1115/1.4044229. URL https://asmedigitalcollection.asme.org/mechanicaldesign/article-pdf/141/11/111405/6578473/md_141_11_111405.pdf, URL https://doi.org/10.1115/1.4044229. 111405.

[8] Wang, Yuyang, Shimada, Kenji and Farimani, Amir Barati. "Airfoil GAN: Encoding and Synthesizing Airfoils forAerodynamic-aware Shape Optimization." (2021). DOI 10.48550/ARXIV.2101.04757. URL https://arxiv.org/abs/2101.04757.

[9] Tan, Ren Kai, Zhang, Nevin L. and Ye, Wenjing. "A deep learning-based method for the design of microstructural materials." *STRUCTURAL AND MULTIDISCIPLINARY OPTIMIZATION* Vol. 61 No. 4 (2020): pp. 1417–1438. DOI 10.1007/s00158-019-02424-2.

[10] Gulrajani, Ishaan, Ahmed, Faruk, Arjovsky, Martin, Dumoulin, Vincent and Courville, Aaron. "Improved Training of Wasserstein GANs." (2017). DOI 10.48550/ARXIV.1704.00028. URL https://arxiv.org/abs/1704.00028.

[11] Ronneberger, Olaf, Fischer, Philipp and Brox, Thomas. "U-Net: Convolutional Networks for Biomedical Image Segmentation." (2015). DOI 10.48550/ARXIV.1505.04597. URL https://arxiv.org/abs/1505.04597.

[12] https://github.com/nicoguaro/ellipse_packing.

[13] https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.binary_fill_holes.html.

[14] "Center for High Throughput Computing (CHTC)." DOI https://doi.org/10.21231/GNT1-HW21. URL https://chtc.cs.wisc.edu.

[15] https://hub.docker.com/r/natsumizu/myopenfoam7.

[16] Lipton, Zachary C. and Tripathi, Subarna. "Precise Recovery of Latent Vectors from Generative Adversarial Networks." DOI 10.48550/ARXIV.1702.04782. URL https://arxiv.org/abs/1702.04782.

[17] https://github.com/aladdinpersson/Machine-Learning-Collection.

[18] Barratt, Shane and Sharma, Rishi. "A Note on the Inception Score." (2018). DOI 10.48550/ARXIV.1801.01973. URL https://arxiv.org/abs/1801.01973.

[19] Heusel, Martin, Ramsauer, Hubert, Unterthiner, Thomas, Nessler, Bernhard and Hochreiter, Sepp. "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium." (2017)DOI 10.48550/ARXIV.1706.08500. URL https://arxiv.org/abs/1706.08500.

[20] https://pypi.org/project/pytorch-gan-metrics/.

[21] Chen, Xianqi, Zhao, Xiaoyu, Gong, Zhiqiang, Zhang, Jun, Zhou, Weien, Chen, Xiaoqian and Yao, Wen. "A Deep Neural Network Surrogate Modeling Benchmark for Temperature Field Prediction of Heat Source Layout." (2021). DOI 10.48550/ARXIV.2103.11177. URL https://arxiv.org/abs/2103.11177.

[22] Sun, Jialiang, Zheng, Xiaohu, Yao, Wen, Zhang, Xiaoya, Zhou, Weien and Chen, Xiaoqian. "Heat Source Layout Optimization Using Automatic Deep Learning Surrogate and Multimodal Neighborhood Search Algorithm." (2022). DOI 10.48550/ARXIV.2205.07812. URL https://arxiv.org/abs/2205.07812.

[23] Mai, Vien V. and Johansson, Mikael. "Stability and Convergence of Stochastic Gradient Clipping: Beyond Lipschitz Continuity and Smoothness." (2021). URL 2102.06489.