CS6850 Final Project
Generative Neural Network Approach to Designing Dynamic Inductive Power Transfer Systems

1st Andrew Curtis
Computer Science Department
Utah State University
Logan, UT, United States
a02043584@aggies.usu.edu

Abstract

Generative Neural Networks (GNN) have demonstrated remarkable power in creating novel graphic design images from text-to-image training. This work applies GNNs to design dynamic inductive power transfer systems to make charging electrical vehicles (EVs) more convenient and cheaper. Discovering optimal and safe coil implementations (for EV and road) is challenging because of the combinatorial explosion of possible configurations, along with multiple conflicting objective functions, such as maximizing the output power, while minimizing stray magnetic fields and the volume of windings and magnetic cores. To solve the problem, a differentiable simulator and evaluator define loss functions that train a generative neural network to only produce configurations that satisfy eight given design criteria. Before training, the rate of finding successful designs is 0.005%, but within 500 training epochs, the rate becomes 98% (about 30 seconds run time). Improving solution diversity and quality is investigated by applying a variety of novel loss functions. Other investigations include validation of GNN reproducibility, inference from viewing the results over the course of training and the effect of adjusting hyper-parameters on the performance of the GNN.

I. Introduction

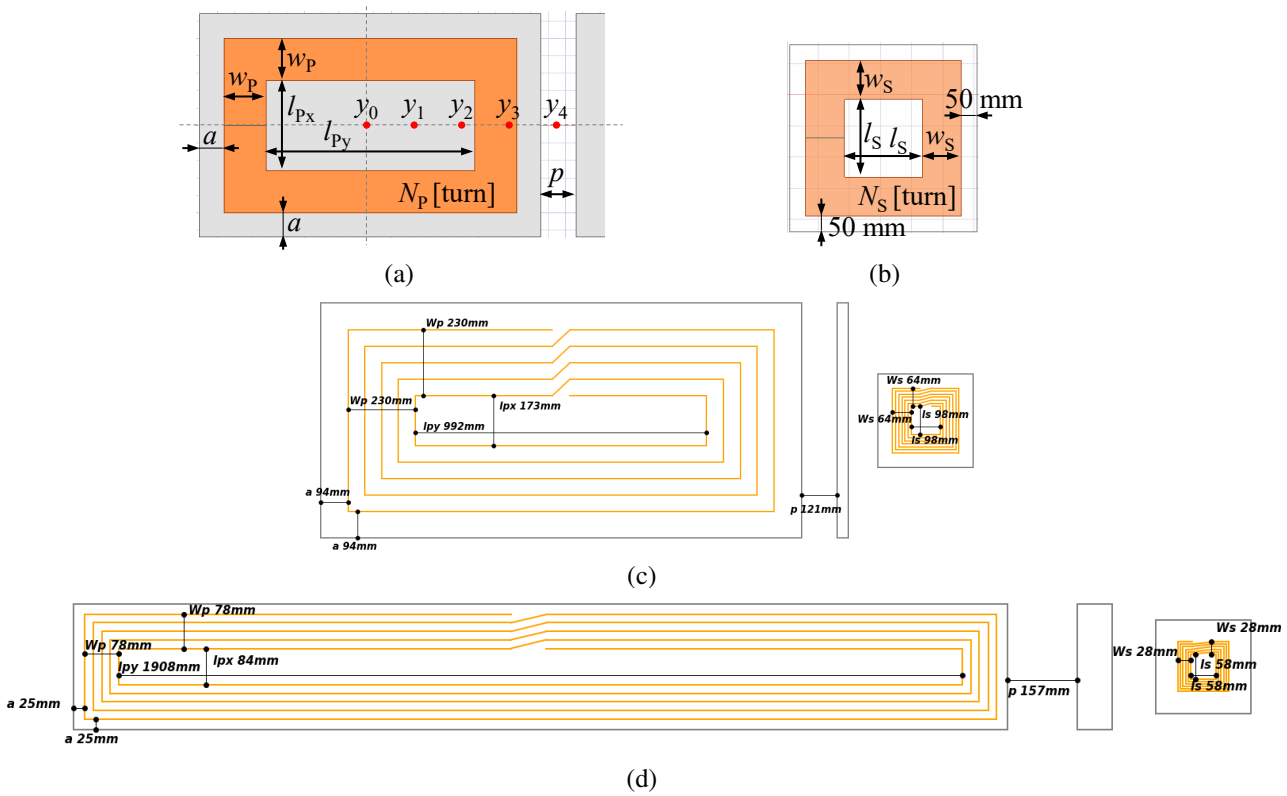

Fig. 1. **Coil design parameters**. (a) The primary coil, described by the parameters l_{Px}, l_{Py}, w_{P}, a, p, which are dimensions in mm and N_p the number of wire turns. (b) The secondary coil, described by the parameters, $l_{\mathrm{S}}, w_{\mathrm{S}}$, dimensions in mm and N_{S}, the number of wire turns. (c) A specific near-optimal design was discovered by the GNN trained by the Product-of-Means loss illustrated in Figure 3.1. (d) A near-optimal design was discovered by the GNN trained by the Minimum-of-products loss illustrated in Figure 3.4.

An urgent need is to electrify transportation to lower carbon emissions into the atmosphere. Wireless charging makes electrical vehicles (EVs) more convenient and cheaper because energy is transferred to the vehicle without the need to plug it in. Dynamic wireless charging is particularly interesting, where the vehicle does not need to stop to receive the energy. This technology requires the EV and the roadway to include coils of wire, where the roadway coil is energized as the vehicle passes over it to induce an electrical current in the EV coil through electromagnetic induction. However, the problem of designing the two coils (EV and road) is complex due to the many configurations possible, the need to maximize power transfer, and the need to minimize stray, possibly dangerous, electromagnetic fields during operation.

A dynamic inductive power transfer system (DIPT) system is comprised of two coils of wire, the primary coil, illustrated in Fig. 1(a) and the secondary coil within the EV, illustrated in Fig. 1(b). The physics of operation is well understood, and computational models exist [1] that enable simulation and evaluation of a specific coil design using conventional numerical analytic methods, such as the finite-element method (FEM). Given a means to simulate the behavior of a potential design, engineers can define specific objective functions that quantify an implementation precisely. The design problem is to find those coil configurations that are optimal and safe, out of a large combinatorial space of possible solutions.

This work presents a machine-learning solution to this design problem where an initial random design generator is trained to produce only designs that pass specific objectives. A generative neural network is applied to shift a Gaussian distribution, initially independent of design objectives, to one in which most designs generated are good. The method works in a similar way that a generative adversarial network learns to create images that look like paintings by Monet [2].

II. Related Work

Pareto optimization or multi-objective optimization is an area of decision making involving multiple conflicting objectives. These objectives are numerical and can be mathematically computed simultaneously. When this issue is present for a set of objectives, points will appear on a non-dominated Pareto front. This means for a pair of objectives, no improvements can be made for that design in that objective without some trade-off in another. The points that fit into this criteria are called non-dominated or Pareto points. An example of this can be seen in Figure 2. The blue points represent design solutions that are dominated, the colored points are Pareto points lying along the Pareto front, visualized by the red line.

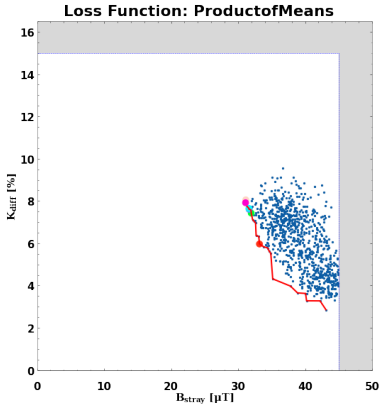

Fig. 2. Example pareto optimal front seen in red, passing through dominant pareto optimal points for the minimization objectives.

The appearance of Pareto optimization problems are common in many areas such as economics, engineering, science and logistics. The field of electrical power systems often deals with these optimization problems and have used a variety of algorithms and viewpoints to address it. The most successful of which primarily derive from evolutionary algorithms. These algorithms imitate the biological evolution of living things or the social interactions among species [3] to perform multi-objective optimization.

Evolutionary-based approaches to multi-objective optimization problems are clever and can work in the case of non-differentiable solutions. However, they are relatively slow algorithms and often will not come close to finding global minimums for optimization. Powerful new gradient-based approaches with deep-learning back-propagation allow for greatly increased speed and performance in the case that the solution can be determined in a fully-differentiable manner.

Successful generative models in the text-to-image domain [4] rely on fully differentiable loss functions [5], that enable back-propagation of error calculated from image and text data samples. The work presented here utilizes the simulation and evaluation model given in [1] as a loss function because it is fully differentiable. Inoue [1] initially implemented a traditional FEM simulator, which was then used to train a simple feed-forward neural network forming a surrogate model. This surrogate model was validated experimentally, as described in [1]. Since the surrogate model is an ordinary neural network and the objective functions are written within Pytorch, the whole code is fully differentiable, enabling the application of generative neural network approaches. The work presented here utilizes the simulation and evaluation model given in [1] in conjunction with a new generative neural network to perform back-propagation of novel loss functions utilizing the objectives.

III. Problem Definition: Generating Near-Optimal DIPTs

Find:

A generator of DIPT designs that quickly learns to create a diversity of coil implementations that satisfy and optimize all eight criteria given in Table I.

Given:

1) A trainable, but initially random design generator that returns a vector of numbers that define a specific DIPT design configuration, $DS_i = \langle I_p,\ N_p,\ N_s,\ l_{\mathrm{Px}},\ l_{\mathrm{Py}},\ w_{\mathrm{P}},\ a,\ p,\ l_{\mathrm{S}},\ w_{\mathrm{S}} \rangle$, the ranges for which are given in Table II. I_p is the electrical current provided to the primary coil, while N_p and N_s define the number of coil turns for the primary and secondary coil, respectively. The remaining seven parameters define physical characteristics measured in mm of the two coils, illustrated in Fig. 1. Additional fixed design parameters are given in Table III.
2) A simulation model that takes a specific design DS_i as input and calculates the expected behavior of the DIPT system under operation. Here the magnetic fields generated and electrical power induced in the secondary coil is determined as the secondary coil passes over the primary coil.
3) Eight objective functions that take the simulation result and calculate specific design objectives, given in Table I.

TABLE I
EIGHT DESIGN OBJECTIVE FUNCTIONS $f_i \in O$ DEFINITION AND CONSTRAINTS

Design Criteria	Objective function	Threshold
Coupling coefficient difference	K_{diff}	$\leq 15\%$
Stray magnetic fields	B_{stray}	$\leq 45\ \mu T$
Coil loss	Q_{loss}	≤ 2000 W
Number of inverters	N_{inv}	≤ 1 1/ m
Power average	P_{ave}	≥ 30 KW/ m
Secondary core volume	$V_{SecCore}$	$\leq 1500\ cm^3$
Primary core volume	$V_{PriCore}$	$\leq 6000\ cm^3$
Secondary side core winding volume	$V_{SecWind}$	$\leq 1000\ cm^3$

TABLE II
THE DESIGN PARAMETER DEFINITIONS AND RANGES FOR THE DIPT SYSTEM.

Parameter	Variable	Value
Primary Coil RMS current	I_P	50∼200 A
Turn number of the primary side	N_p	4-10
Turn number of the secondary side	N_s	4-10
X-direction length of the primary winding	l_{Px}	50∼650 mm
Y-direction length of the primary winding	l_{Py}	50∼2050 mm
Width of the primary winding	ω_P	25∼325 mm
Length of the edge of the primary core	a	0∼200 mm
Pitch of the adjacent cores	p	0∼200 mm
Length of the secondary winding	l_S	50∼450 mm
Width of the secondary winding	ω_S	25∼225 mm

TABLE III
DESIGN SPECIFICATIONS FOR THE FIXED PARAMETERS OF THE DIPT SYSTEM.

Parameter	Variable	Value
Input DC voltage	V_{dc}	400 V
Output DC voltage	V_{bat}	400 V
Output power at the center	$P_{out,x0,y0}$	50 kW
Switching frequency	f_s	85 kHz
Air Gap	g	200 mm
Length of the edge of the secondary core	b	50 mm

IV. Method

A. Generative Neural Network Approach

Generative neural networks can solve engineering design problems since they demonstrate the ability to be creative yet constrained within a regular domain [6]. In this work, we use a similar back-propagation generative neural network approach to [6]. However, here the loss function is calculated directly on the objectives over a self-generated data set, rather than an error over an external data set.

Fig. 3 depicts the learner's architecture. A vector of Gaussian noise ζ_i is generated and input to a generator network, $\zeta_i \circ GNN(\theta)$, where θ are the trainable parameters (tensor size 68,618). The output is scaled into physical units d_k to represent

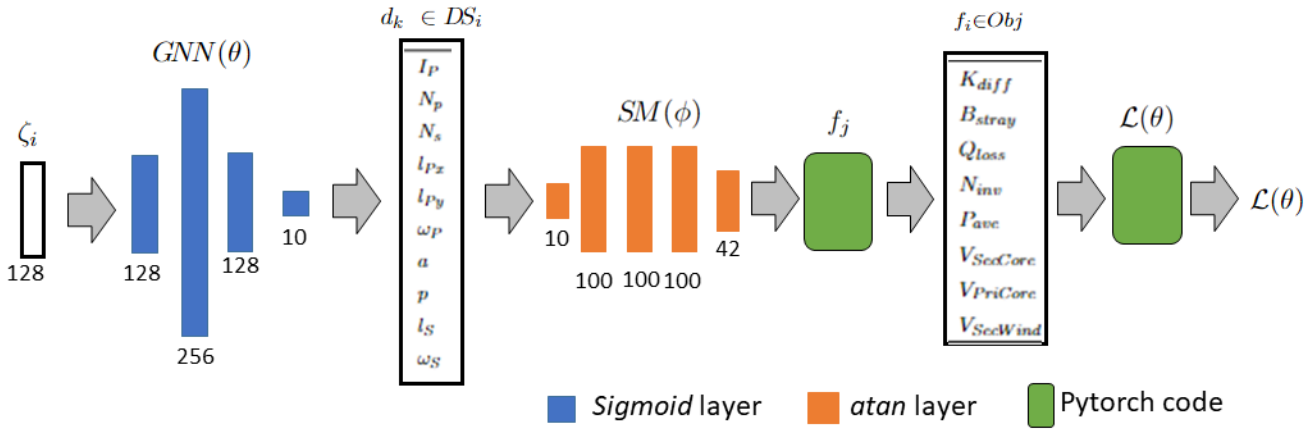

Fig. 3. **Generative Model Architecture**. A design $d_k \in DS_i$ (see Figure 1) is generated from Gaussian noise ζ_i passed through a four-layer neural network $GNN(\theta)$. The design is evaluated using a surrogate simulation model $SM(\phi)$ followed by eight objective functions $f_j \in O$. Learning is accomplished by back-propagating gradients of a loss function $\mathcal{L}(\theta)$ formulated to improve all the objectives.

a design solution, $d_k \in DS_i$ as illustrated in Fig. 1. The design DS_i is input into the surrogate model $SM(\phi)$, (ϕ fixed tensor size 25,742) producing the spatiotemporal behavior of the design under operation. The surrogate model is a fixed neural network, trained from FEM simulations and verified in [1].

The eight objective functions defined in [1] and listed in Table I are then calculated. The objective functions have minimization or maximization goals and are combined into a single loss function $\mathcal{L}(\theta)$ calculation, described in Section IV-B. Finally, to learn, both loss $\mathcal{L}(\theta)$ and its gradient $\partial\mathcal{L}(\theta)/\partial\theta$ are calculated and back-propagated into the generator network $GNN(\theta)$ to update weights θ using the Adam optimizer.

B. Optimization and Loss Functions

Objective functions, $K_{diff}, B_{stray}, Q_{loss}, N_{inv}, V_{SecCore}, V_{PriCore}, V_{SecWind}$ are all to be minimized and have upper bounds that must not be exceeded (see Table I). The remaining objective function, P_{ave}, has both a lower bound, given in Table I, and an upper bound of 50 kW. For simplicity, the objective of maximization of P_{ave} was flipped to minimization by subtracting it from 10^4 kW. Various novel loss functions are explored in below in the results section.

V. Results

A. Product-of-Means Loss

To avoid the introduction of arbitrary hyper-parameters, no scaling is performed on any of the objective functions. The Adam optimizer was used with a learning rate of $3 \cdot 10^{-4}$. A simple product-of-means loss was first explored, where $\mathcal{L}(\theta)$ is the product of the means of all individual objective functions over a solution population generated during an epoch, which is set at $n = 10^3$. Let O be the set of objective functions = $\{K_{diff}, B_{stray}, Q_{loss}, N_{inv}, V_{SecCore}, V_{PriCore}, V_{SecWind}, 10^4 - P_{ave}\}$. Then the product-of-means loss function is defined:

$$\mathcal{L}(\theta) = \prod_{f_j \in O} \frac{1}{n} \sum_{i=1}^{i=n} \zeta_i \circ GNN(\theta) \circ SM(\phi) \circ f_j$$

Where θ is the vector of 68,618 trainable weights in the generative NN and ϕ is the vector of 25,742 non-trainable weights in the surrogate model NN.

In all training, 500 epochs were run, each generating 10^3 ζ_i Gaussian noise vectors. Initially, the network (whose weights were initialized randomly) produced designs with only 0.005 % of solutions that passed all eight objectives listed in Table I. After 500 epochs, training θ on $\mathcal{L}(\theta)$ found 991 satisfactory solutions out of 1000, demonstrating successful learning.

Fig. 4 shows the actual designs found (left), scaled to the percentage of their ranges given in Table II. Fig. 4 (right) represents the quality of the solutions defined by the objective functions. For this plot, the blue points are solutions produced by the generator falling within the threshold values, the limits of which are indicated by the grey regions. The red line in the objectives plot is a visualization of the Pareto front for each pair of objectives. A single Pareto optimal solution for the pair of objectives, K_{diff} and B_{stray}, is shown in red and its actual design is illustrated in Fig. 1(c).

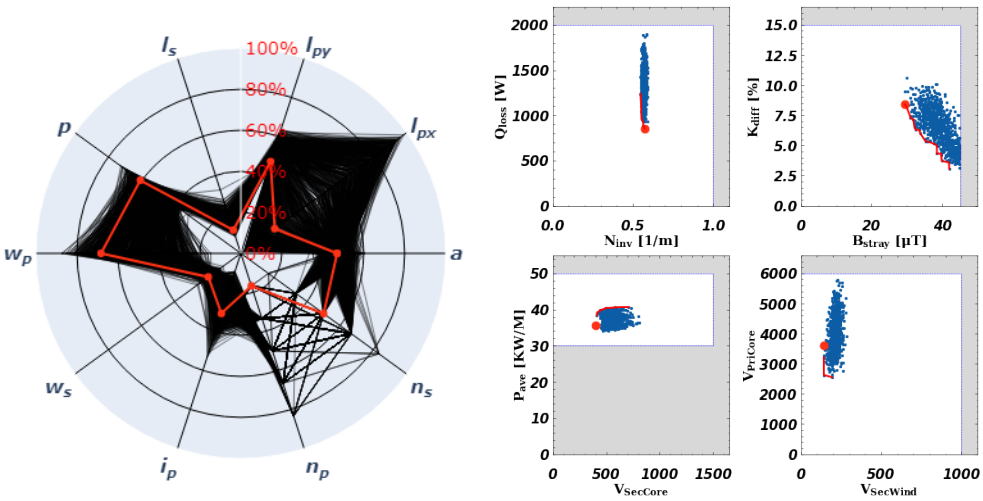

Fig. 4. **Product-of-means loss.** Design solution parameters (left) and performance in objective functions(right) produced by generator after training.

B. Pairwise Pareto Optimal Loss

To achieve further improvements in design objectives and diversity, two other loss functions were explored. The first considers design solutions that are Pareto optimal w.r.t. specific objective function pairs. Pareto optimal designs are non-dominated solutions for which no improvements can be made for an objective function without losing some optimization in another. Pairwise sets of the objective functions can be analyzed during training to determine the Pareto solutions for the generated set. These Pareto solutions are then utilized in the loss function directly to have the learning focus on the best designs discovered rather than using all the created designs.

For this loss function, a specific pair of optimization criteria (from Table I) was chosen, and the Pareto optimal set of solutions generated with respect to that pair is used in the loss function. Let O_o be the pair of objective functions to be improved, $\hat{O}$ be $O - O_o$ and PO be the set of random vectors ζ_i that produce designs that are non-dominated with respect to O_o, then the specialized loss function may be defined as:

$$\mathcal{L}_{PO}(\theta) = \prod_{f_j \in \hat{O}} \frac{1}{n} \sum_{i=1}^{i=n} \zeta_i \circ GNN(\theta) \circ SM(\phi) \circ f_j \cdot$$
$$\prod_{f_j \in O_o} \frac{1}{\|PO\|} \sum_{s_i \in PO} s_i \circ GNN(\theta) \circ SM(\phi) \circ f_j$$

Setting N_{inv} and Q_{loss} in O_o produces results shown in Fig. 5, which finds 484 solutions that pass all eight objectives out of 1000 after training for 500 epochs. All five Pareto Optimal design solutions of N_{inv} and Q_{loss} objectives are shown in different colors. The distribution of solutions in the design space is similar to the generator trained by product-of-means loss but significant improvements are found for the objective functions of Q_{loss} and B_{stray}.

Setting $V_{SecCore}$ and P_{ave}in O_o produces results shown in Fig. 6, which finds 481 solutions that pass all eight objectives out of 1000 after training for 500 epochs. All seven Pareto Optimal design solutions of $V_{SecCore}$ and P_{ave} objectives are shown in different colors. Training on this set of objectives produces a distinct distribution from the previous two loss functions explored. Interestingly, the Pareto fronts are expanded and potentially produce many more designs of interest.

C. Minimum-of-Products Loss

The next loss function explored has an increased trade-off for increasing performance in objective functions while producing fewer accepted solutions. For each epoch, the minimum-of-products loss function is calculated by taking the minimum over all element-wise products of the objective functions applied to each random design.

The minimum-of-products loss function may be defined as:

$$\mathcal{L}_{min}(\theta) = \min_{i=1}^{i=n} \left(\prod_{f_j \in O} \zeta_i \circ GNN(\theta) \circ SM(\phi) \circ f_j \right)$$

The results are shown in Fig. 7, finding 172 solutions out of 1000. Significant improvements can be seen for the objectives of B_{stray}, Q_{loss}, N_{inv}, $V_{SecCore}$, $V_{SecWind}$ and $V_{PriCore}$. The distribution of the solutions in the objective and the design space are discernibly different from generators trained with the other loss functions. A single solution is illustrated in red and

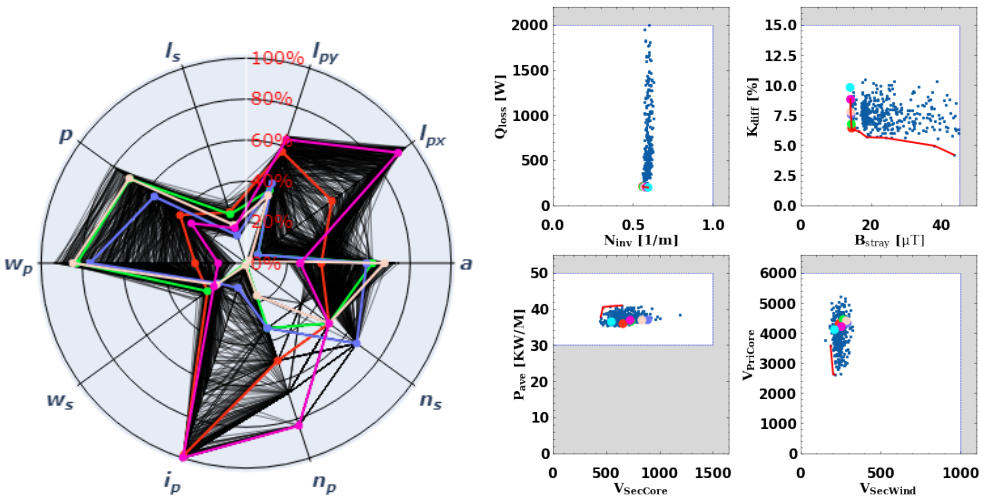

Fig. 5. **Pairwise Pareto Loss N_{inv} and Q_{loss}.** Design solution parameters (left) and performance in objective functions(right) produced by generator after training.

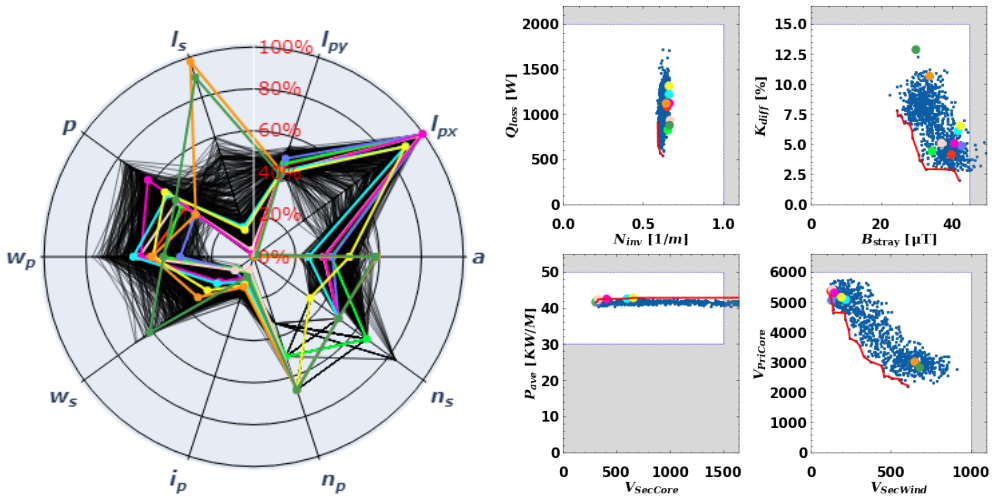

Fig. 6. **Pairwise Pareto Loss $V_{SecCore}$ and P_{ave}.** Design solution parameters (left) and performance in objective functions(right) produced by generator after training.

its actual design is illustrated in Fig. 1(d). This solution is especially interesting since several of the design parameters lie near the end ranges and produce the best results for many of the optimization criteria.

D. Mean-of-PCA Loss

In order to see if a subset of features would be effective at producing interesting designs, the final two loss functions were explored. The mean-of-PCA loss computes the first 1000 principal components of the objective functions and then takes the mean of the result as the loss. It can be defined as follows:

$$\mathcal{L}_{meanPCA}(\theta) = \frac{1}{n}\sum_{i=1}^{i=n}(\underset{1000}{PCA}\prod_{f_j \in O}\zeta_i \circ GNN(\theta) \circ SM(\phi) \circ f_j) \tag{1}$$

The generator trained with the mean-of-products loss found only 4 accepted solutions out of 1000. However it does produce the lowest resulting design for N_inv, seen in Figure 8. The PCA loss functions were interesting in that the generator was able to be trained to improve on the objective functions and produced different distributions than just the product-of-means and minimum-of-products loss functions.

E. Minimum-of-PCA Loss

Similar to the mean-of-PCA loss, this loss function is calculated by computing the first 1000 prinicipal components of the objective functions but then taking the minimum of the result as the loss. This loss function learned to produce 56 accepted

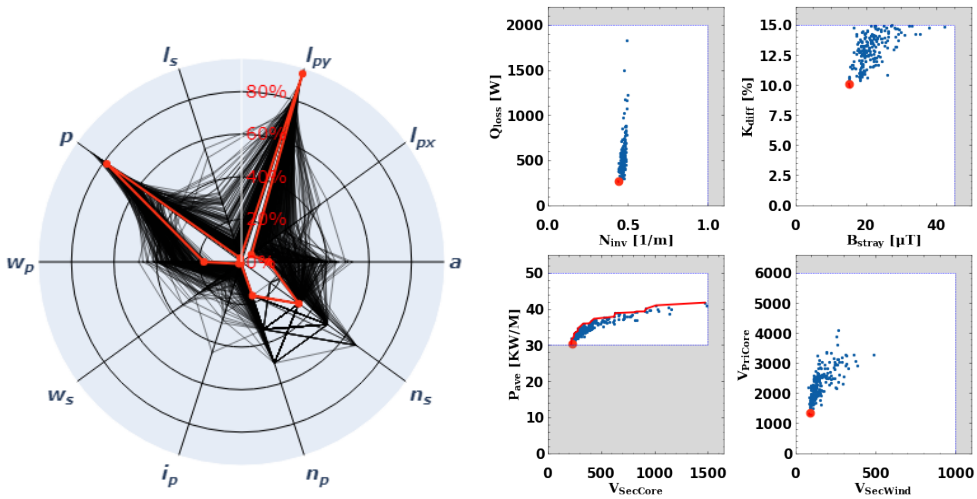

Fig. 7. **Minimum-of-products loss.** Design solution parameters (left) and performance in objective functions(right) produced by generator after training.

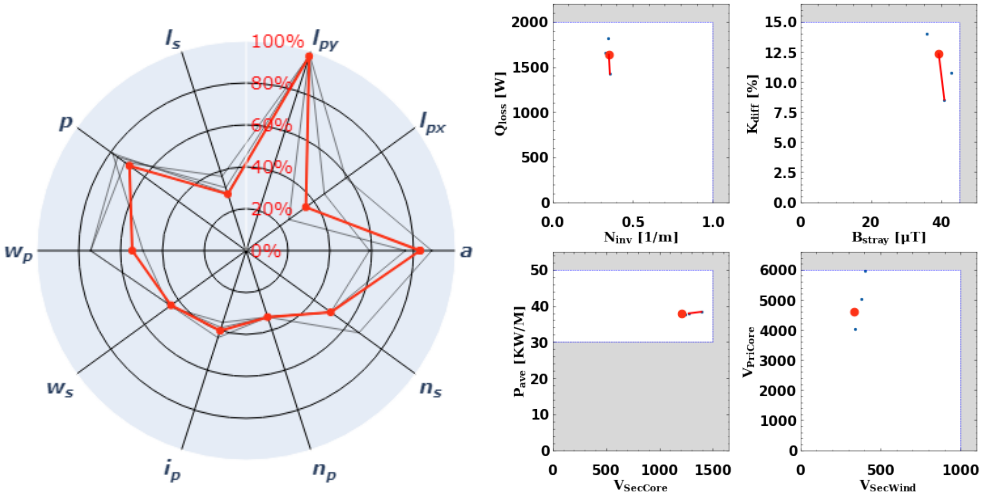

Fig. 8. **Mean-of-PCA loss.** Design solution parameters (left) and performance in objective functions(right) produced by generator after training.

solutions out of 1000. It also produced interesting results in the design space and low values for N_inv, as seen in Figure 9. The minimum-of-PCA loss is defined as follows:

$$\mathcal{L}_{minPCA}(\theta) = \min_{i=1}^{i=n}(\underset{1000}{PCA} \prod_{f_j \in O} \zeta_i \circ GNN(\theta) \circ SM(\phi) \circ f_j) \tag{2}$$

F. Additional investigations

1) Trained generator network reproducibility: In order to check the reproducibility of the trained network, it was fed random sets of gaussian noise after training in order to see if the distribution of produced design solutions would be similar to each other as well as their performance in the objective functions. As expected, the distributions are similar since the parameters of the trained network are held constant. This was a good validation of the trained generator's reproducibility.

2) Retraining of the generator network: In this investigation, the generator was retrained and then fed noise to see if it would produce similar distributions of the designs. This can be thought of as an investigation of whether the generator is reaching a similar local minima as it is trained by stochastic gradient descent. The results indicate that that the distribution of the designs is similar although it may vary slightly in performance of the objectives. Retraining for a better result might be worth investigation when trying to reach a near-optimal design.

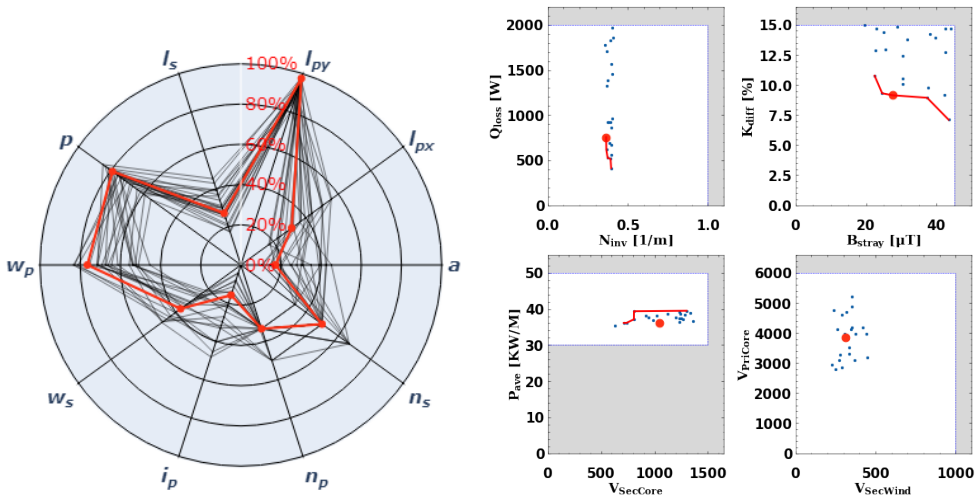

Fig. 9. **Minimum-of-PCA loss.** Design solution parameters (left) and performance in objective functions(right) produced by generator after training.

3) Learning of the network over training: Visualizing the learning of the network over training perhaps provides some of the best inference of additional information for domain experts to see what designs are being focused on by the generator and why. Unraveling the black box of the neural networks is partially enabled by doing this step. The product-of-means loss produces more accepted solutions and a greater diversity of designs over the course of training. However, the minimum-of-products loss produces fewer and fewer accepted solutions, but appears to hone in on a singular near optimal design. It works towards and favors particular extreme values for the design parameters, seen in Figure 10. Training with the minimum-of-products loss also produces a few designs that lie on the pareto optimal fronts for all or nearly all of the objective functions at the same time, seen in Figure 11.

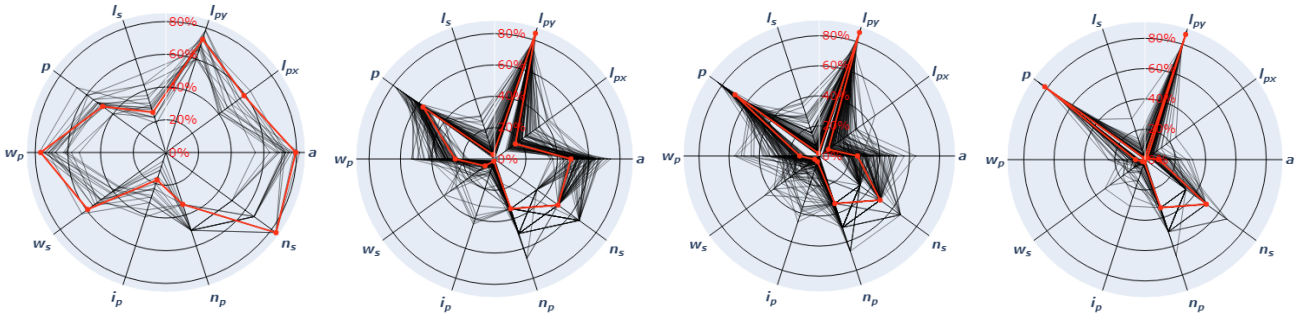

Fig. 10. Design solutions over the course of training with the minimum-of-products loss. Epochs shown: 0, 150, 300, 500.

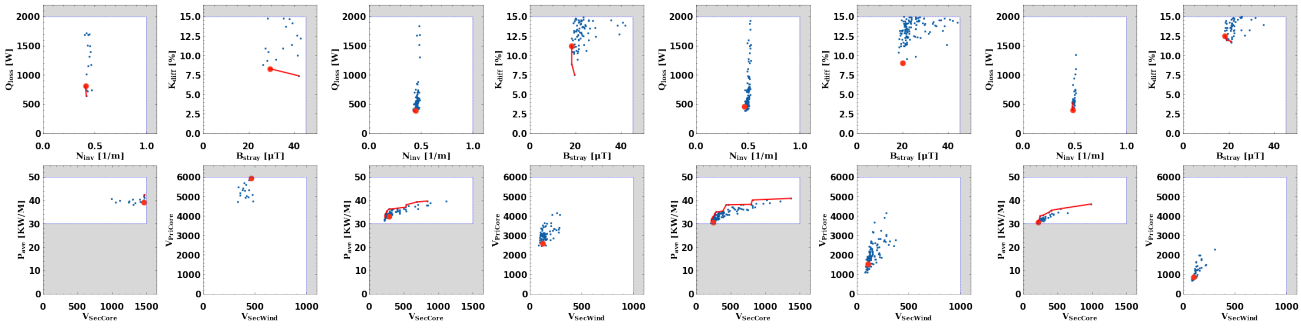

Fig. 11. Performance of objectives over the course of training with the minimum-of-products loss. Epochs shown: 0, 150, 300, 500.

4) Effect of generator network depth: The effect of network depth was tested to see the consequence it would have on this multi-objective optimization problem. Three generator networks with differing amounts of hidden layers were tested. The first network which was the standard for the rest of the research was 3 hidden layers with a total of 68,096 trainable parameters, seen in Figure 12.

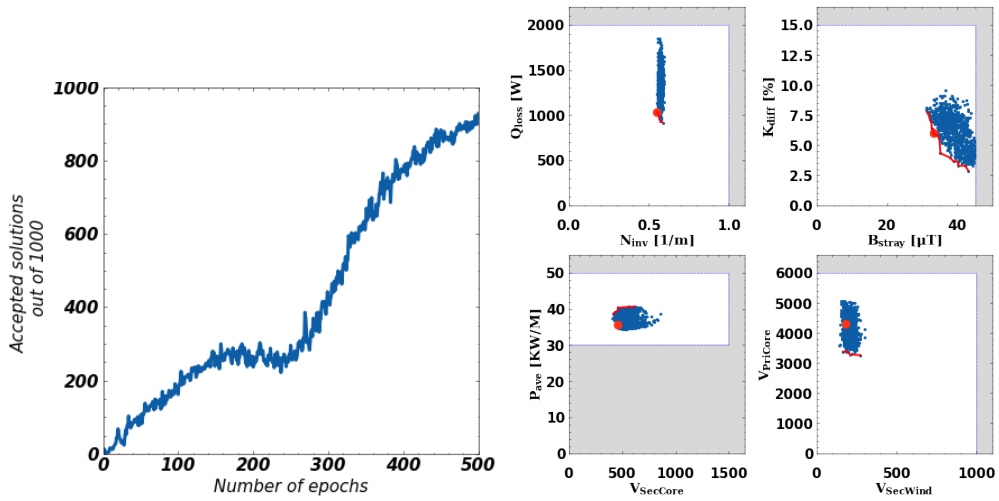

Fig. 12. Number of accepted solutions produced per training epoch (left) and performance in objective functions(right) produced by 3-hidden layer generator after training.

The second network had 5 hidden layers with a total of 330,240 trainable parameters, Figure 13. This network produced good results with some different distributions than the 3-layer network.

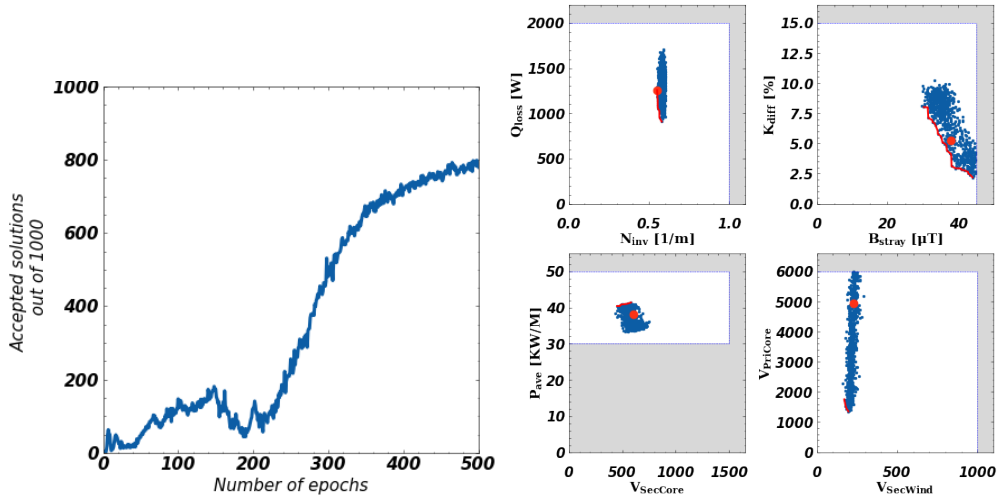

Fig. 13. Number of accepted solutions produced per training epoch (left) and performance in objective functions(right) produced by 5-hidden layer generator after training.

A third network had 7 layers with 1,378,816 trainable parameters. This larger network produced about half as many accepted solutions, but had especially good performance for the objective of k_{diff}, the best seen during any of the research tests, seen in Figure 14. Deeper and larger networks do not necessarily produce better results for the multi-objective optimization problem but may be an interesting method to reach different local minimum and find an additional diversity of near-optimal designs for certain objectives.

5) Effect of learning rate on generator network: The effect of learning rate on the generator network was tested to see if different local minima could be reached for the multi-objective optimization problem. All testing included the standard of 500 training epochs using the Adam optimizer which performs stochastic gradient descent. The standard used for the rest of the research was $3 \cdot 10^{-}4$, additionally the values of $3 \cdot 10^{-}3$, $3 \cdot 10^{-}5$ and $3 \cdot 10^{-}6$ are tested here. Seen in Figure 15, the learning rate of $3 \cdot 10^{-}3$ had too great of rate of change and threw off the parameters after the loss became negative. Learning of the generator stopped after this was the case and accepted solutions did not increase.

The learning rate of $3 \cdot 10^{-}5$ had good performance and appears to nearly converge to a local minimum quickly, Figure 16.

The learning rate of $3 \cdot 10^{-}6$ appears to approach a similar local minimum but did not converge by the end of the training epochs, seen in Figure 17. It does show a more smooth increase in number of accepted solutions found but appears to require more epochs and training time. The conclusion from this investigation is that learning rate hyper-parameters should be

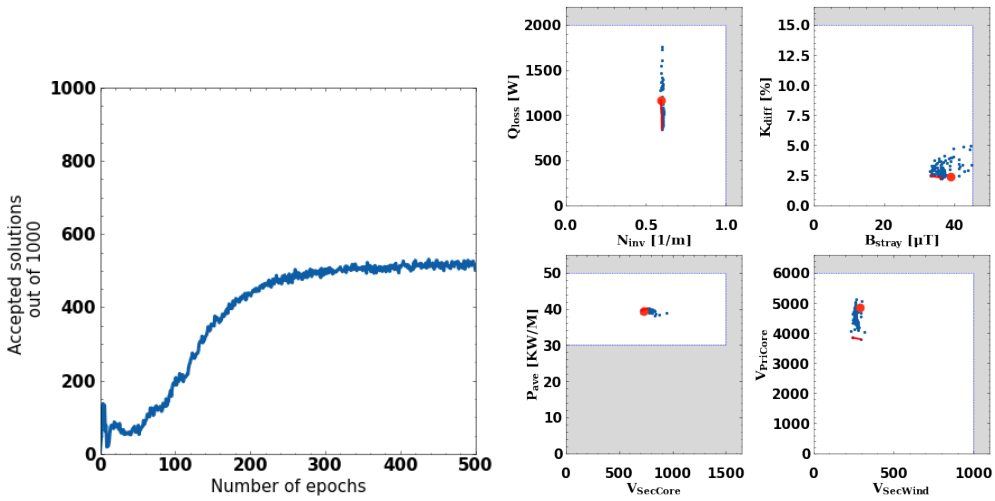

Fig. 14. Number of accepted solutions produced per training epoch (left) and performance in objective functions (right) produced by 7-hidden layer generator after training.

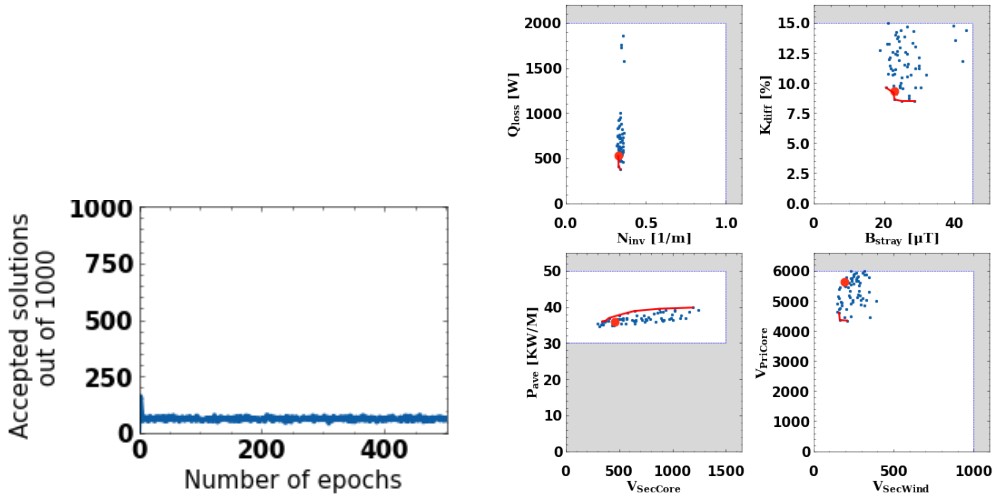

Fig. 15. Number of accepted solutions produced per training epoch (left) and performance in objective functions (right) produced by generator after training with learning rate of $3 \cdot 10^{-}3$.

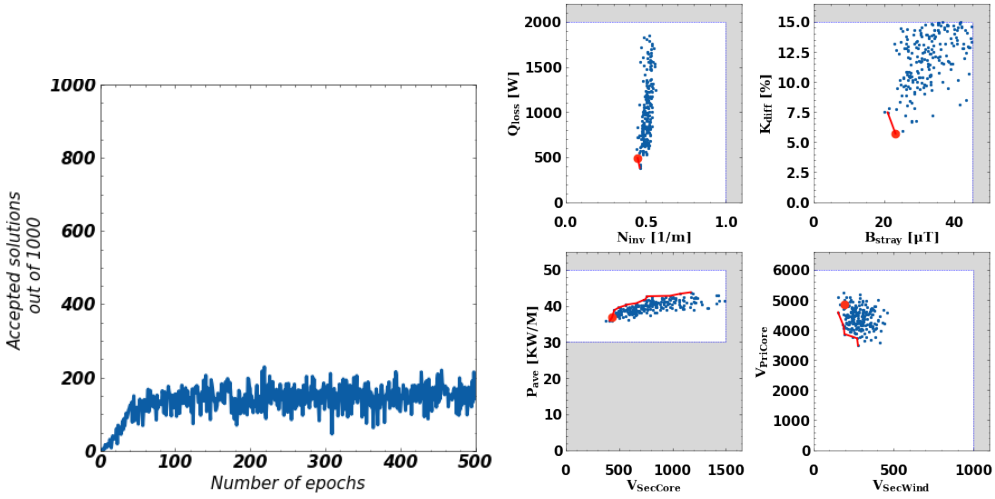

Fig. 16. Number of accepted solutions produced per training epoch (left) and performance in objective functions (right) produced by generator after training with learning rate of $3 \cdot 10^{-}5$.

investigated with number of epochs to reach an efficient balance for training time, but it does not seem to change the local minimum reached or change distributions of performance in objective functions.

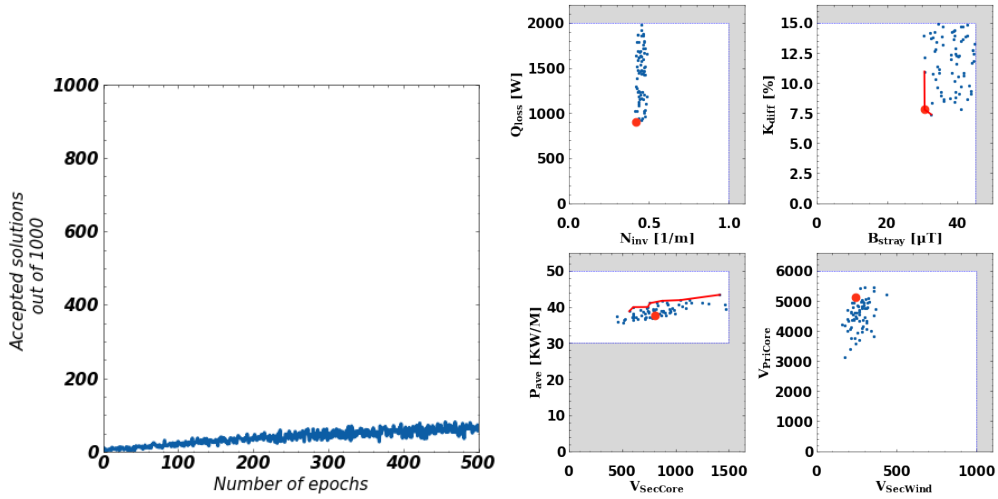

Fig. 17. Number of accepted solutions produced per training epoch (left) and performance in objective functions(right) produced by generator after training with learning rate of $3 \cdot 10^{-}6$.

VI. Conclusion

Generative neural networks can quickly learn to create design configurations that both satisfy minimization requirements and optimize for specific objectives. Compared to a conventional Genetic Algorithm approach [1], the proposed GNN can find a set of satisfying solutions within seconds compared to many hours of run time. Applying this method to other engineering design problems requires that the simulator and the evaluator be differentiable, which may be achieved by first learning a neural network surrogate model from FEM samples [1] or utilization of differentiable simulators [5]. Introducing novel loss functions that directly incorporate the objectives can create unique design configuration distributions that optimize for objective functions. Viewing the how the generator learns over time provides additional inference for domain experts on what designs configurations may be interesting to pursue. Testing different hyper-parameters of the network such as depth and size may also lead to different local minima and design outcomes.

VII. Roles

This project was an extension of my research. I was able to work on it jointly with the extra time. Specific investigations enabled by the CS6850 project time include:

- Testing of new loss functions based on PCA analysis
- Trained generator network reproducibility
- Retraining of the generator network
- Learning of the network over training epochs
- Effect of generator network depth
- Effect of learning rate

References

[1] S. Inoue, R. Nimri, A. Kamineni, and R. Zane, "A new design optimization method for dynamic inductive power transfer systems utilizing a neural network," in *2021 IEEE Energy Conversion Congress and Exposition (ECCE)*. IEEE, 2021, pp. 1496–1501.

[2] I. Santos, L. Castro, N. Rodriguez-Fernandez, Torrente-Patiño, and A. Carballal, "Artificial Neural Networks and Deep Learning in the Visual Arts: a review," *Neural Computing and Applications*, vol. 33, no. 1, pp. 121–157, Jan. 2021. [Online]. Available: https://doi.org/10.1007/s00521-020-05565-4

[3] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[4] A. Oussidi and A. Elhassouny, "Deep generative models: Survey," in *2018 International Conference on Intelligent Systems and Computer Vision (ISCV)*, Apr. 2018, pp. 1–8.

[5] Y. Hu, L. Anderson, T.-M. Li, Q. Sun, N. Carr, J. Ragan-Kelley, and F. Durand, "DiffTaichi: Differentiable Programming for Physical Simulation," Feb. 2020, arXiv:1910.00935 [physics, stat]. [Online]. Available: http://arxiv.org/abs/1910.00935

[6] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.