NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

LUIS FILIPE MARTINS TRIPA

BSc in Computer Science and Engineering

COMBINING DEEP LEARNING AND OPTIMIZATION FOR MEMS DEVICES DESIGN

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon

February, 2024

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

COMBINING DEEP LEARNING AND OPTIMIZATION FOR MEMS DEVICES DESIGN

LUIS FILIPE MARTINS TRIPA

BSc in Computer Science and Engineering

Adviser: David Semedo
Assistant Professor, NOVA University Lisbon

Co-adviser: Chen Wang
Post-doctorate Researcher, ESAT-MNS KU Leuven

Examination Committee

Chair: Susana Maria dos Santos Nascimento Martins de Almeida
Assistant Professor, FCT-NOVA

Rapporteur: Carlos Jorge Andrade Mariz Santiago
Assistant Researcher, Instituto Superior Técnico da Universidade de Lisboa

Adviser: David Fernandes Semedo
Assistant Professor, FCT-NOVA

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
February, 2024

Combining Deep Learning and Optimization for MEMS Devices Design

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.1.5) [1].

To my grandfather, who would have loved to see me graduate.

Acknowledgements

I would like to express my deepest gratitude to the following people, whose support and encouragement were essential throughout the development of this thesis:

First and foremost, my professor and instructor, David Semedo, for his invaluable guidance, support, and motivation. His provision of technical resources, including access to the cluster where our AI models were trained, was instrumental to the success of this work.

I would also like to thank my fellow researcher, Catarina Bento, for her contributions in the development and validation of multiple components of this thesis, and the KU Leuven team for their domain expertise and valuable feedback.

Furthermore, I am grateful to my university and the department of informatics for the knowledge and skills I have gained over the past five years, which have culminated in the completion of this thesis.

Last but not least, I want to express my heartfelt appreciation to my friends and family, who have been a constant source of support and patience throughout these years. Their understanding and encouragement have made this journey much more bearable.

”

“I would like to die on Mars. Just not on impact.”

— Elon Musk, Speaking at an SXSW conference
(Businessman and investor)

Abstract

Micro-electromechanical systems (MEMS) devices play a crucial role in today's electronics revolutionizing various industries through their miniaturization and increased performance capabilities. In particular, MEMS resonators are essential for sensing, timing, and filtering applications, where demand for smaller, better-performing devices is paramount.

Recent advancements in resonator design, use machine-learning algorithms to streamline the generation process, by significantly increasing design speed. Moreover, these approaches allow operators to generate designs based on target properties, customizing the generated designs for specific use cases. However, these advancements do not particularly focus on generating designs with the best possible performances, meaning the best results are not always obtained.

In this thesis, done in collaboration with the ESAT-MNS from KU Leuven, we bridged this gap by developing an optimization pipeline capable of mixing optimization strategies with generative models to generate resonator geometries with improved performances.

The approach, which uses a process similar to Active Learning, has achieved a mean generated designs' performance improvement of around 5.6%, compared to designs generated by an unoptimized generative model. This improvement ensures design geometry solutions that are higher performing, while still generating them faster than human designers ever could, streamlining the design process.

Keywords: MEMS design, Artificial intelligence, Generative AI, Optimization, Performance optimization

Resumo

Dispositivos MEMS desempenham um papel crucial na eletrónica hoje-em-dia, revolucionando várias indústrias por meio de sua miniaturização e capacidades acrescidas de desempenho. Em particular, os osciladores MEMS são essenciais para aplicações de detecção, temporização e filtragem, onde a demanda por dispositivos menores e de melhor desempenho é de extrema importância.

Avanços recentes no design de osciladores utilizam algoritmos de aprendizagem automática para otimizar o processo de geração, aumentando significativamente a velocidade de design. Além disso, essas abordagens permitem que os operadores gerem designs com base em propriedades alvo, personalizando os designs gerados para casos de uso específicos. No entanto, esses avanços não se concentram particularmente em gerar designs com os melhores desempenhos possíveis, ou seja, os melhores resultados nem sempre são obtidos.

Nesta tese, realizada em colaboração com o ESAT-MNS da KU Leuven, preenchemos esta lacuna ao desenvolver uma pipeline de otimização capaz de combinar estratégias de otimização com modelos generativos, para gerar geometrias de osciladores com desempenhos melhorados.

Esta abordagem, que utiliza um processo semelhante ao Active Learning, obteve uma melhoria média de performance dos designs gerados de cerca de 5.6%, em comparação com designs gerados por um modelo generativo não otimizado. Esta melhoria garante geometrias de design com melhor desempenho, gerando os designs mais rapidamente do que os deseigners humanos poderiam, simplificando o processo de design.

Palavras-chave: Design de MEMS, Inteligência artificial, AI generativa, Otimização, Otimização de performance

Contents

Annexes

List of Figures

List of Tables

Acronyms

AL Active Learning *(pp. 19, 21)*

CGAN Conditional Generative Adversarial Network *(pp. 16, 17, 30, 83)*

FEA Finite Element Analysis *(p. 11)*
FEM Finite Element Method *(pp. 11, 36, 37)*

GAAL Generative Adversarial Active Learning *(p. 20)*
GAN Generative Adversarial Network *(pp. 14, 16, 17, 20, 30, 31, 46, 58, 82, 83)*

KNN K-Nearest Neighbors *(p. 82)*

MEMS Micro-electromechanical systems *(pp. v–vii, x, xi, 1–5, 7–9, 11, 14, 18, 24–26, 30–32, 35, 54, 57, 58, 60, 61, 75, 82, 83)*
MSE Mean Squared Error *(pp. 40, 44)*

Q Quality Factor *(pp. x, 1, 2, 9–11, 20, 26–28, 36, 37, 53, 59–68, 71–74, 76–80, 82, 83)*

SVM Support Vector Machine *(p. 20)*

1 Introduction

1.1 Context and Motivation

Micro-electromechanical systems (MEMS) play a crucial role in modern technology, revolutionizing various industries through their miniaturization and integration of mechanical and electrical components on a microscopic scale [2]. These devices have transformed everyday gadgets and systems, ranging from consumer electronics and automotive sensors [3] to medical devices [4]. In particular, MEMS resonators and oscillators stand out as essential components in sensing applications [5], where changes in a resonant element are used to monitor a given quantity, timing applications [6], where a resonant element is used to generate a high-quality clock signal, or in filtering applications [7], where resonant structures implement filters that can be of use in radiofrequency wireless transceivers.

Despite their significance, current MEMS resonator design processes are often slow and yield devices with suboptimal performance values. Traditional design methods struggle to efficiently explore the vast design space, leading to time-consuming iterations and limited optimization capabilities. In recent years, there has been growing interest in leveraging generative models to streamline the design process by automatically generating MEMS device layouts [8, 9]. However, existing approaches primarily focus on generating designs automatically rather than optimizing performance metrics such as their Q value (described in detail in Section 2.2).

The primary motivation behind optimizing MEMS resonators stems from the observation that designs with higher quality factors often exhibit superior energy efficiency, increased design stability, and heightened sensitivity compared to their less-performant counterparts. These enhanced characteristics not only improve the overall performance of MEMS resonators but also unlock new opportunities for their utilization in novel applications and innovative techniques. By optimizing the quality factor of MEMS resonators, we can harness their full potential and explore previously untapped avenues for MEMS technology in a multitude of research fields.

1.2 Thesis Scope and Objective

This thesis aims to address the existent gap between the design of MEMS resonators and their optimization, specifically by improving their performance. By combining generative modeling techniques with optimization algorithms, this research seeks to create MEMS resonator designs with enhanced efficiencies that compete with the current state-of-the-art designs, while tapping into the potential and ease of use of a generative model.

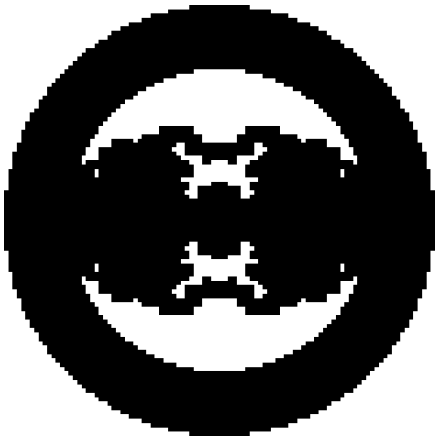

Figure 1.1: A 2D representation of a MEMS resonator design. The black pixels represent regions of the design with material, while the white pixels represent void regions (nonexistence of material).

The primary objective of this thesis is to develop an optimization framework dedicated to improving the performance of MEMS resonators, with a central focus on optimizing their Q value. The Q value is a critical metric that directly influences the efficiency of MEMS resonators in diverse applications. By systematically enhancing the quality factor of generated resonator designs, this framework aims to elevate the overall performance and reliability of MEMS devices, thereby addressing key challenges and limitations in current MEMS resonator optimization practices. Further details of Q-factor and other properties of resonators are explored in Section 2.2.

To achieve this objective, the optimization framework will leverage generative models to automatically generate MEMS resonator designs, which will then be optimized using our framework to generate improved designs. Furthermore, the framework is designed to be flexible and adaptable, allowing for the integration of new generative models and optimization strategies as they emerge. This adaptability ensures that the optimization framework remains at the forefront of research and development in MEMS resonator design, enabling continuous improvements and advancements in the field.

This thesis was done in collaboration with several other researchers from both the KU Leuven University and NOVA School of Science and Technology, all working on different components of the MEMS resonators' problem. Figure 1.2 presents the several work divisions created to tackle several areas of interest.

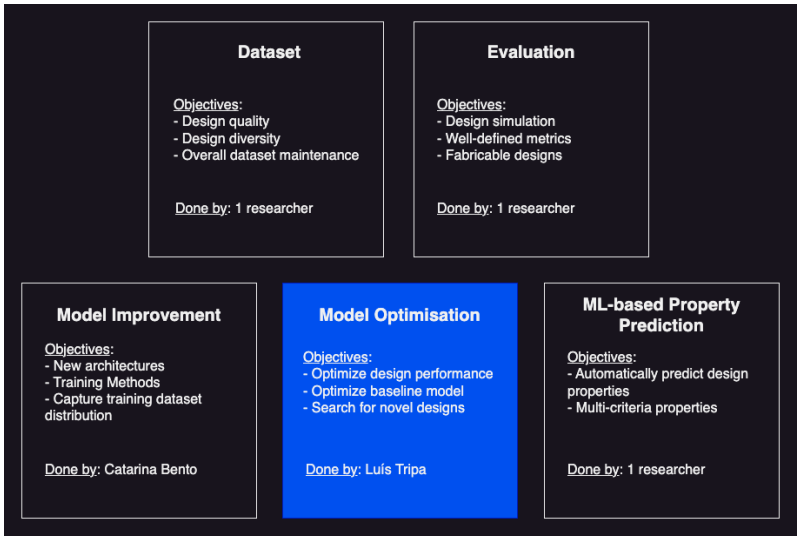

Figure 1.2: The various fields of research being done in parallel with this thesis. Highlighted in blue is the core scope of our thesis focused primarily on the optimization of resonator devices, however, constant collaboration will occur between all components. Other components include the maintenance of the dataset, evaluation of designs through both physics simulation and ML/DL models, and the generation of designs using state-of-the-art generative architectures.

One of the key aspects of this collaboration is that our optimization component will be used as a complement to the generation component in the way that our optimization pipeline should be "generative model agnostic", meaning that whichever breakthroughs the "Model Improvement" component achieves, whether it is tweaking the baseline pre-trained generative model, discussed in subsequent sections, or replacing it with more advanced generation models, can and will be integrated into our optimization pipeline and carefully evaluated. The same thing will happen with the "ML-based Property Prediction" component, responsible for creating a model capable of predicting the properties of a device faster than the physics simulation software.

1.3 Challenges and Research Hypothesis

By delving into this relatively unexplored domain, the research aims to introduce innovative methodologies to enhance the performance and efficiency of MEMS resonator designs. The exploration of optimization techniques tailored specifically to MEMS resonators signifies a significant step forward in advancing the capabilities and reliability of MEMS devices across various applications.

To achieve this goal, the usage of an Active Learning-based approach, discussed in detail in subsequent sections is proposed as a possible optimization route. This approach leverages generative approaches, using data augmentation in order to achieve the desired optimized designs. The hypothesis is that by doing this, the quality of the designs generated by the baseline generative model can be improved. However, there are several

challenges that must be addressed to validate this hypothesis:

- A good baseline generative model is needed to kick-start the optimization research, meaning one will have to be trained from scratch;
- Mixing generative models and optimization algorithms can be challenging. The optimization frameworks, if used incorrectly, can negatively impact the quality of the generated designs, asserting the need for careful tuning and evaluation;
- The search space is enormous. Designs have an intricate geometry with many variations, which are very hard to explore efficiently. The optimization algorithm has to be good at exploring the large design space;
- The objective function, whether explicit or implicit, is most likely non-convex due to the complexity of the resonator designs. This means that finding the global optimum will be a challenge, as discussed in Section 2.4;
- The evaluation of designs can be hard. Physics simulations are too slow to incorporate into training loops, meaning other methods will have to be devised, as discussed in Section 2.3;

We propose to address these challenges by leveraging optimization methodologies that have been trial-and-tested in other similar domains and bring them to the MEMS resonator research field, where they can be implemented and carefully evaluated. By using state-of-the-art algorithms and frameworks, problems such as the convergence to local optima and the inefficiency of traversing large design spaces have the potential to be mitigated enough to yield significant improvements in the optimization process of MEMS resonators, paving the way for innovative and high-performance devices across various applications.

1.4 Contributions and Achievements

As a result of the work performed in this thesis, we highlight the following contributions:

1. **Generation and evaluation of a new dataset**: A new dataset, containing diverse and high-quality MEMS resonator designs, was generated and simulated in collaboration with the KU Leuven team. (Chapter 4);
2. **Generative model training specification**: A baseline generative model was trained from scratch to generate MEMS resonator designs, providing a starting point for the optimization research. The key contribution here is the clear specification of the training process and the model's architecture, which was not present in the literature before (Chapter 5);

3. **Novel design evaluation metrics**: New metrics were developed to evaluate the quality of the generated designs, providing insights into the performance of the generative model that generated them. As far as we know, most of these metrics have not been used in the literature before (Chapter 6);

4. **Active-learning based optimization pipeline**: A fully customizable optimization pipeline was developed, leveraging techniques similar to the active-learning approach to optimize the quality factor of MEMS resonators. Several internal optimization strategies were also developed and tested to improve the optimization process (Chapter 8);

5. **Abstract submission under review for IEEE MEMS conference**: "*Generative and Generalizable Optimization Framework For MEMS Devices Through Active Learning*" (Available in Annex I);

1.5 Document Structure

Chapter 1 Introduction: This chapter describes the context and motivation as well as introduces the thesis' objective and possible challenges that will arise.

Chapter 2 Background: This chapter provides a comprehensive overview of MEMS resonators, covering their fundamental principles, key performance metrics, and applications. Additionally, it discusses the datasets that will be used in the research and does a light introduction to optimization tasks.

Chapter 3 Related Work: This chapter introduces literature that covers areas such as the generation of MEMS resonators and possible optimization approaches that could be adapted to our problem.

Chapter 4 Dataset and Data: This chapter presents the dataset used in this work, as well as analysis of the data contained in it.

Chapter 5 Generation Methodology: This chapter presents the generative model used in this work, as well as the training process and data preprocessing steps.

Chapter 6 Design Evaluation: This chapter presents the evaluation of the generated designs, as well as the models used to predict the properties of the designs.

Chapter 7 Generation Results: This chapter presents the results of the generative model, as well as the evaluation of the generated designs.

Chapter 8 Optimization Methodology: This chapter presents the optimization strategies used in this work.

Chapter 9 Optimization Results: This chapter presents the results of the optimization strategies used in this work and evaluates their overall performance.

2

Background

This chapter introduces the background necessary to understand both the MEMS resonator domain and optimization algorithms. First, we discuss the most prominent features of these devices, including which of these features are responsible for defining the properties of a design. Next, we discuss two main ways on how to obtain the properties of a design weighting the pros and cons of each approach. Finally, we also discuss the datasets that we will be using throughout this thesis to achieve our goals.

2.1 MEMS resonators

Before we delve into the more advanced topics of optimization, it is crucial to establish a foundational understanding of MEMS resonator devices and, more specifically, grasp their structural intricacies. Figure 2.1 provides a visual representation of a typical MEMS resonator structure. The designs are depicted through 2D images composed of 100 by 100 pixels. Within this pixel grid, each element is assigned a binary value of either 0 or 1, denoting the presence of a 'void' or a material element (non-void) at the respective position. This binary representation serves as a fundamental language for characterizing the resonator's composition.

It's noteworthy that each pixel within these representations corresponds to a discrete section of the MEMS resonator device, portraying minute dimensions. To put this into perspective, Guo et al. [10] established a specific definition for a pixel, considering it as an area measuring $0.88 \times 0.88\ \mu m$ within the resulting resonator. This level of precision underscores the intricate nature of MEMS resonators, where even the smallest details play a significant role in their overall functionality. As we proceed, this understanding of the basic structural elements will provide a solid foundation for exploring more intricate aspects such as design optimization in the realm of MEMS technology.

Even though we're utilizing 2D images to represent designs, it's important to note that the resulting MEMS resonators inherently manifest as 3D structures in real applications. However, the choice to employ 2D representations is a practical simplification, as demonstrated by studies such as Guo et al. [10] and Sui et al. [8], where the complexity of the

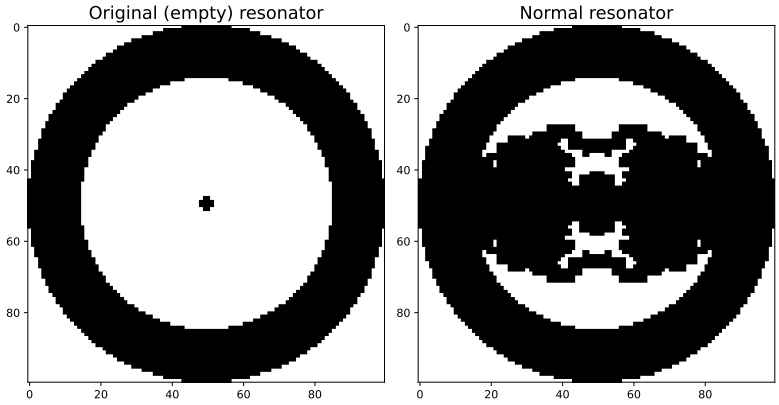

Figure 2.1: A resonator design image (right) and an empty resonator ring (left). Resonators are represented by 2D images of 100×100 binary pixels. A pixel with a value of 0 (white) represents a 'void' element and a pixel with a value of 1 (black) represents a material element (non-void). The leftmost image is not a valid resonator, but instead the base structure that all resonators are built upon.

resonator design can be adequately captured in two dimensions. This third dimension primarily pertains to a thickness value, a parameter influenced by external factors rather than being determined by the design itself. These external factors, while significant in shaping the resonator's physicality, fall beyond the direct scope of the thesis.

In Figure 2.2, a compelling 3D visualization provides insight into the physical embodiment of a real resonator. This representation serves to bridge the conceptual gap between the 2D depictions used in design illustrations and the actual three-dimensional nature of the fabricated MEMS resonator.

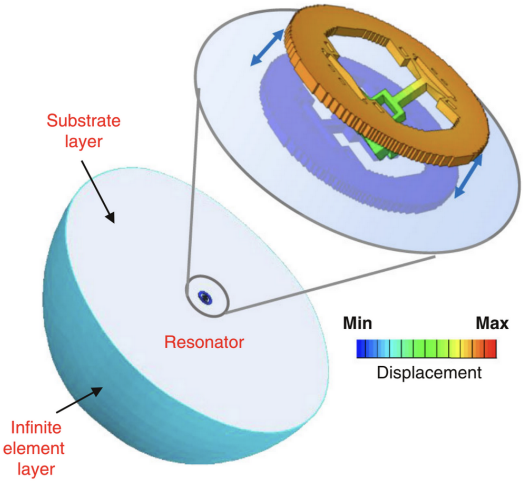

Figure 2.2: A 3D representation of a MEMS resonator. Resonators are placed in a silicon substrate and are fixated using their anchor. Figure adapted from Guo et al. [10]

Upon closer inspection of Figure 2.1, three distinct features emerge as defining the visual characteristics of a resonator:

The Ring This is a ring-shaped structure located in the outermost parts of the resonator. This is consistently present in every design, meaning it remains unchanged in terms of both shape and position.

The Anchor Positioned in the center of the resonator, the anchor secures the resonator to the substrate as illustrated in Figure 2.2. This is the only component that is physically connected to the substrate

The Internal Structure Appearing between the ring and the anchor, this component forms a complex structure that resembles a myriad of paths connecting the ring to the resonator. This structure is unique in every device and defines their properties, like the frequency modes and the Q value. Our work will focus on generating and optimizing this component. Additionally, a few design constraints are present in this component, which guarantee that a design can be fabricated. We will discuss these constraints in detail in Section 6.1.

It is worth mentioning, however, that even though the anchor and the ring can have different configurations for different problem settings, for the purposes of this work, we will consider them as fixed unchanged components, focusing our efforts on the internal structure described above.

Understanding these three key components and their roles in the resonator design is crucial to make sure that the subsequent generative model training and optimization tasks are well-informed and aligned with the underlying principles of MEMS resonators.

2.2 Resonant frequencies and Q Factor

MEMS resonators operate by vibrating at a specific frequency determined by their internal structure, however, given their circular shape, each design encompasses a multitude of different vibration directions, called frequency modes. For example, Guo et al. [10] used 4 resonator frequency modes in their work: torsional over axis X, torsional over axis Y, rotational and flexural modes. These directions are illustrated in Figure 2.3.

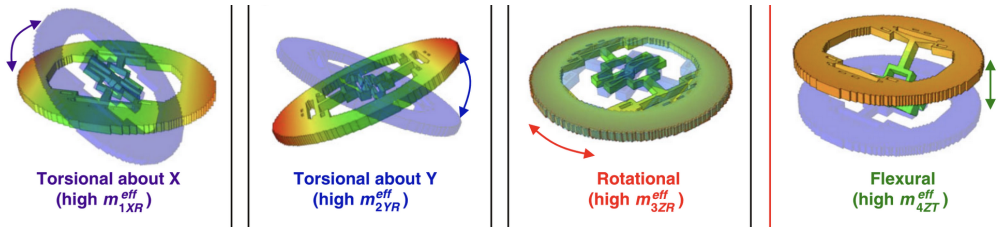

Figure 2.3: A 3D representation of the MEMS resonators' 4 frequency modes used in [10]. Figure adapted from Guo et al. [10]

Besides the frequency modes, each resonator design also has a performance metric associated with it, called the Q value (sometimes also referred to as the quality factor).

This unitless metric measures the ratio between the energy stored in the resonator and the energy lost in each cycle due to a damping effect. This damping effect, also known as the anchor loss, is inversely proportional to the Q value, meaning that higher Q values correspond to lower anchor losses and vice-versa [2].

A lower anchor loss (higher Q value) corresponds to systems whose vibration is sustained for longer periods and is often associated with more efficient designs or designs that are more sensitive, while higher anchor losses (lower Q value) are associated with systems that lose energy more quickly and are less efficient or less sensitive.

Since the anchor loss/Q value duality limits how a resonator vibrates, there is a direct correlation between the Q value and the frequency of a resonator, as observed in Figure 2.4.

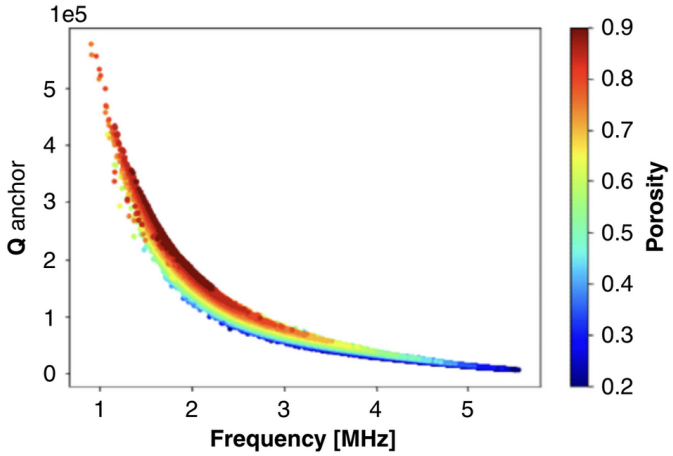

Figure 2.4: A plot of designs' flexural mode frequency (mode 4) and respective Q value. This figure also contains the concept of porosity, which will be explained later. Figure adapted from Guo et al. [10]

Lower frequency values tend to have higher Q values, thus lower anchor losses and vibrate for longer periods of time, while higher frequency values tend to have lower Q values, thus higher anchor losses and vibrate for shorter periods of time. This observed correlation informs our expectations for the models we end up training, suggesting that they will likely exhibit similar behavior.

Although a multitude of frequency modes are present in every resonator device, irrespective of the internal structure's shape, typically only one frequency mode is considered the most important. The choice of this 'main' frequency is, as usual, dependent on the specific problem being addressed. In the work of Guo et al. [10], the authors focused on the flexural mode as their frequency mode of interest, and, due to its convenience for actuation and sensing, we will also adopt a similar approach in our investigation.

2.3 Resonator evaluation

In the domain of design optimization, the availability of robust performance metrics is indispensable. These metrics serve as the pivot for comparing the performance of various

designs. Without them, the task of evaluating and selecting the superior design choice becomes notably difficult.

As outlined in Section 2.2, MEMS resonators encapsulate a frequency of interest and an accompanying Q value, serving as a metric for design quality. However, these have to be obtained somehow.

The most used approach to evaluate these properties is through physics-based simulations, where the resonator's frequency and Q value are calculated by simulating its vibration behavior. Physics software, such as the *COMSOL* software [11] used by the KU Leuven team, is often used to computationally model the resonator's physical behavior and infer its properties. These physics-based simulations use a technique called the Finite Element Method (FEM) to model the resonator's physical behavior by dividing it into smaller chunks, called finite elements, and numerically solving the equations governing the resonators' vibration. The output of these simulations is a precise frequency and Q value for each resonator design, which can then be used in further analysis.

One particular advantage of these simulation-based evaluations is the ability to precisely determine the device's properties with great accuracy. Oftentimes, these simulations are considered the ground truth for the resonator's performance, with the frequency and Q value error margins being mostly negligible and attributed to the simulation software's numerical precision.

Our dataset, which will be further discussed in Chapter 4, was obtained using this method, ensuring that the frequency and Q values are as similar as possible to the real-world performance.

However, these simulations are particularly computationally expensive, with each resonator design taking several minutes to simulate completely. According to Guo et al. [10], the authors experienced approximately 41 seconds for computing the frequency value and 235 seconds for calculating the Q value using their own Finite Element Analysis (FEA) simulation software, for a single resonator. The KU Leuven team also reported similar computational costs for their simulations. This computational cost is a significant drawback when evaluating numerous designs, which will be the case when training generative models or optimizing them.

On the flip side, as a solution to this challenge, rather than simulating the intricate physics governing MEMS resonators, an alternative approach involves using a deep learning model to predict the frequency and Q values based exclusively on the resonator's image. The key idea, as proposed by Guo et al. [10], is to leverage the computational efficiency of such a model compared to traditional physics simulations, while still maintaining a high level of accuracy.

In their work, the authors advocated for the adoption of a pre-trained deep learning model and fine-tuning it for predicting the resonators' frequency and Q value. Impressively, the results obtained closely paralleled those reported by their FEA simulation software, with an overall mean accuracy of 98.6% for frequency prediction and 96.5% for Q value prediction. Additionally, this alternative approach exhibited significant improvement

in computational efficiency, being capable of swiftly predicting performance values in approximately 9.3 ms. This makes it highly suitable for our specific use case, as it mitigates the computational challenges associated with the traditional simulation approach and offers expedited predictions, aligning well with our requirements.

We will return to this topic in Section 6.2, where we will discuss in detail the solution proposed by Guo et al. [10] and how we will use it in our work.

2.4 The concept of optimization

Optimization, the process of finding the optimal solution from a set of possible alternatives, is a fundamental concept in not only virtually all scientific and engineering fields, but also in economics and others [12].

At its core, optimization is quite a simple process: given a set of input variables that define our current solution, an objective function that tells us how good a specific solution is, and a set of constraints that define the boundaries of the problem, we want to find the best possible solution that maximizes or minimizes the objective function, while still respecting the constraints. The idea is to create a systematic way to adjust the input variables and monitor changes in the objective function to help us find the optimal solution.

In the realm of mathematical optimization, various methodologies exist to tackle this task, each offering unique strengths and trade-offs. One prominent approach is gradient-based optimization [13], which leverages the concept of gradients to iteratively refine solutions towards an optimal point in our objective function. Figure 2.5 shows a comprehensive visualization of what this approach would look like when minimizing a convex objective function.

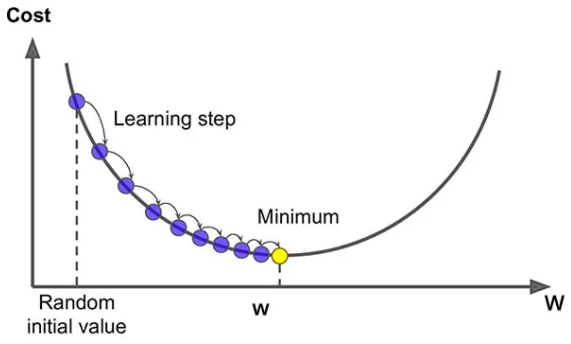

Figure 2.5: Visualization of a gradient descent optimization of a convex cost function. This approach involves initializing a random weight w (the topmost blue dot) and gradually modifying its value, according to the gradients (the direction of optimization), to achieve an optimal solution (the yellow dot). A learning rate hyperparameter is necessary to tune how big of a step the algorithm should take when changing the weight w. Figure adapted from [14].

This is perhaps the simplest form of optimization and is widely used in many fields,

including machine learning, where it is used to train models by adjusting their weights to minimize a loss function.

However, not all optimization problems are alike, and many real-world problems are non-convex, meaning multiple optimal solutions exist and the objective function is not a simple bowl- or arch-shaped curve. Unlike convex functions, which feature a single global optimum, non-convex function landscapes can possess multiple local optima, making it challenging for gradient-based optimization algorithms to converge to the global optimum. Figure 2.6 illustrates this challenge visually.

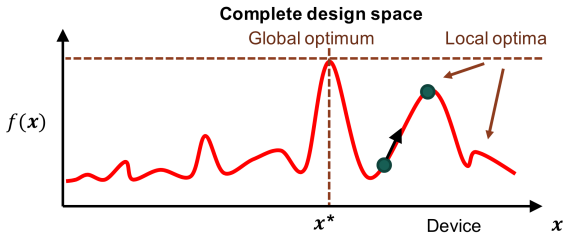

Figure 2.6: Visualization of a gradient descent optimization of a non-convex design space. With the standard hill-climb algorithm, the left-most green dot will converge to a suboptimal solution, instead of converging to the global optimum. Figure adapted from [15].

Multiple strategies exist to address this challenge, such as using special algorithms that temporarily accept worse solutions to escape local optima [16, 17], using particle-swarm optimization [18] approaches, where multiple points are initialized and optimized instead of a single one at a given time, or using mathematical tricks that help emphasize the global optimum. However, reaching the best possible solution in every single case is often not guaranteed, and the optimization process can be quite complex and computationally expensive.

3

Related Work

In this chapter, we discuss previous research works to better understand already established methodologies to generate MEMS resonators and optimization algorithms that are currently being employed in various domains, including in applications similar to ours. Firstly, we introduce the state-of-the-art in the generation of MEMS resonator's designs including an explanation of how these models work, including the conditioned design generation. Finally, we delve into optimization schemes and how they are implemented in different domains that can be transposed to MEMS resonators, including approaches such as Active Learning-based optimization and Global optimization networks, towards creating a pipeline to optimize generated designs and thus achieving the objective of this thesis.

3.1 Deep Generative Models

A cornerstone of this thesis involves assisting in the development of a baseline model capable of generating designs without undergoing any optimization, a crucial initial step that lays the groundwork for subsequent optimization endeavors. Recent advancements in MEMS resonator generation, as evidenced in works like [8, 9], highlight the efficacy of deep generative models in capturing the complexity of these devices' designs.

A model that lays the foundation for [8], the Generative Adversarial Network (GAN) [19], has been used for a while for other domains [20–22] and has shown a robust ability to discern and replicate the distribution of their training sets. Figure 3.1 provides a visual representation of the fundamental structure of a simple GAN model.

The underlying mechanism of such a model revolves around two primary components:

The Generator This component is responsible for generating the image, taking as input a latent vector of fixed dimension with randomized values. It is noteworthy to mention that in the initial iterations, the generated images tend to exhibit high levels of noise and may not even look anything like the designs from the training set.

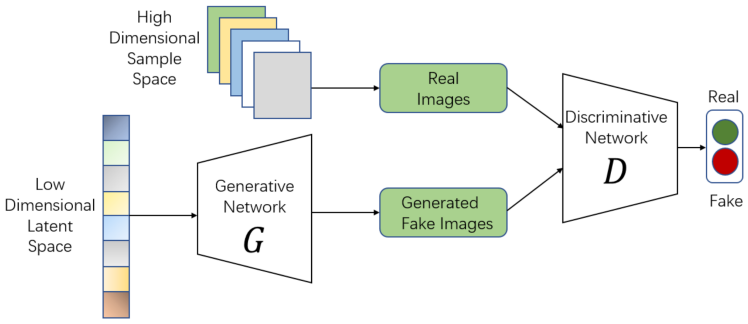

Figure 3.1: The architecture of a vanilla GAN model. G represents the Generator model, which generates synthetic images and D represents the Discriminator model, which tries to evaluate the images as being real (in the same distribution of the dataset) or fake (not in the same distribution of the dataset.), depending on whether we provide an image from the dataset or an image from the generator. Figure adapted from Cai et al. [23]

The Discriminator Designed to take an image as input, the discriminator's primary objective is to determine whether the specified image is real (i.e., falls within the dataset distribution) or fake (i.e., lies outside the dataset distribution, was generated), assigning the values 0 or 1 corresponding to fake or real images respectively. The discriminator's loss function combines the losses incurred from discriminating between these two types of images, as shown in Equation 3.1

$$\min_G \max_D V(D,G) = \mathop{\mathbb{E}}_{x \sim p_{data}(x)} [\log D(x)] + \mathop{\mathbb{E}}_{z \sim p_z(z)} [1 - \log D(G(z))] \tag{3.1}$$

Where G corresponds to the Generator component, D to the Discriminator, x to an image from the distribution p_{data} and z to a latent vector from the distribution p_z, which usually is the standard normal distribution.

The dynamics between these two components create an adversarial relationship. The discriminator aims to maximize the value of Equation 3.1, signifying its proficiency in distinguishing real from fake images, while the generator strives to minimize this value. This adversarial interplay establishes a potent mechanism for capturing and replicating the intricate distribution of the training dataset that has been trial-and-tested in multiple image generation domains [24].

This approach is particularly relevant to our thesis, as it allows the creation of diverse and representative designs without the need for optimization. The resulting baseline model will serve as a foundation upon which we can iteratively enhance and fine-tune designs for improved performance. However, while the vanilla GAN model serves as a robust candidate for our baseline optimization pipeline, it lacks a critical feature essential for our objectives. Specifically, it falls short in terms of conditioning the generated results with specific input parameters, essentially generating designs based on an initial frequency value. To address this crucial aspect, we delve into a model architecture that addresses and mitigates this limitation in the subsequent section.

3.2 Conditional deep generative models

Models like GAN models [19] and Diffusion Models [25] exhibit remarkable performance in capturing the distribution of the training dataset. However, their conventional versions often lack a crucial feature, particularly significant in various domains: the ability to generate image representations based on specific input parameters that define the desired properties of the generated output. In practical terms, while these models excel at capturing intricate dataset patterns, they may fall short when it comes to providing control over the generation process.

In specific problem domains, such as ours focused on resonator designs, operators face challenges when relying on vanilla versions of these models. For example, when generating resonator designs, operators need a mechanism to create designs based on pre-defined parameters, like a target frequency. The limitation arises from the inherent randomness in the generation process of vanilla GAN, where operators may find themselves at the mercy of chance in obtaining the desired characteristics. This issue underscores the importance of enhancing these models to incorporate parameter-controlled generation, allowing operators greater precision and control in achieving specific design objectives.

In the study presented by Sui et al. [8], the authors addressed the limitation outlined earlier by employing a Conditional Generative Adversarial Network (CGAN). A CGAN is a specialized variant of the GAN model that introduces a conditioning functionality, providing a solution to the challenge of parameter-controlled generation. The conditioning mechanism allows the model to take specific input parameters into account during the generation process, enabling a more targeted and controlled output.

Figure 3.2 provides a comprehensive visual representation of the CGAN architecture employed by the authors, offering insights into its key components and the seamless integration of conditioning functionality. A noteworthy observation is an architectural resemblance to the conventional GAN illustrated in Figure 3.1, featuring essential components such as the Generator and Discriminator. However, the authors introduce two crucial modifications that imbue the model with conditioning capabilities: the Fully Connected Decoder (FCD), and the Predictor component.

The Fully Connected Decoder (FCD) assumes a pivotal role in the conditioning process. It encodes the desired design properties, represented as ϕ_c into a higher dimensional vector, mirroring the structure of the randomized latent vector z. This transformation lays the foundation for controlled generation, allowing the model to incorporate specific design parameters.

Subsequently, the augmented higher dimensional vector, resulting from the FCD's encoding process, is concatenated with the randomized latent vector already familiar to the standard GAN model. This combined vector is then fed into the Generator model, with the expectation that the model not only adheres to the dataset distribution but also generates designs conforming to the specified properties.

Following a pattern reminiscent of a standard GAN, the generated image undergoes

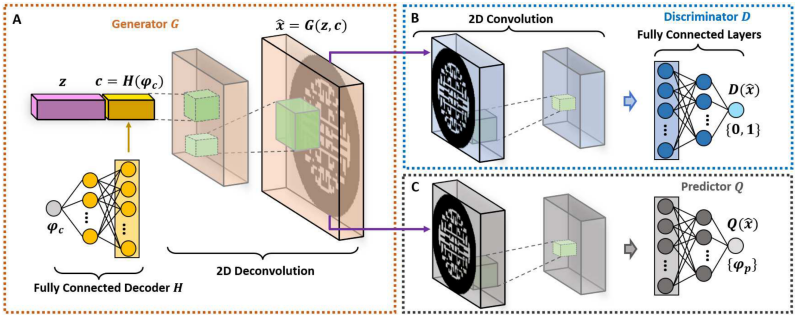

Figure 3.2: The CGAN architecture used by the authors in [8]. Common components with the vanilla GAN include the Generator and the Discriminator, but two new structures are present: the Fully Connected Decoder and the Predictor component. These components are fundamental to condition the generation. ϕ_c represents the desired property and ϕ_p the predicted property, which are then both used in a distance loss function to update the weights of the FCD. Figure adapted from [8]

scrutiny by the discriminator, which distinguishes the images as being real or artificially generated. However, a noteworthy modification sets this architecture apart, allowing it to harness the conditioning functionality: the incorporation of the Predictor model. This component plays a pivotal role in predicting the properties of the generated image, denoted as ϕ_p, which can diverge greatly from the desired properties, especially in the first iterations of the training loop.

The predicted properties, specifically the frequency-related information, serve as a key driver in the conditioning process. They are harnessed to train the FCD, a crucial element in the architecture. Through this training process, the FCD progressively adapts its encoding of the desired design property, leading to refined inputs provided to the Generator model. This iterative adjustment mechanism ensures that the Generator is guided to produce images with the desired frequency value, introducing a nuanced level of control over the generated outputs.

It is worth noting that the Predictor model employed in this context remains consistent with the description provided in Section 2.3 and was introduced by Guo et al. [10]. Its role in predicting image properties, particularly those related to frequency, underscores its significance in shaping the conditioning process and contributing to the model's ability to generate designs aligned with specified criteria. This interplay between the Predictor and the FCD highlights a thoughtful integration of predictive capabilities, enhancing the overall versatility and adaptability of the CGAN architecture.

Additionally, it is critical to understand that whatever generative architecture we use in our optimization pipeline is easily detachable from it, meaning that the pipeline remains model-agnostic. For example, if the Generator-Discriminator components from a GAN is replaced by a diffusion model, the pipeline would still be capable of optimizing the performance of the generated designs. However, implementing and evaluating these new model architectures is outside this thesis' scope.

3.3 Optimization schemes

Optimization algorithms come in various forms, each suited to specific situations where they should or should not be applied. Broadly speaking, optimization can be divided into two main types:

Explicit optimization : Involves systematically searching for the optimal solution to a problem by clearly defining an objective function, decision variables, and any constraints. This approach is distinguished by its precise mathematical formulation, which guides the search for optimal parameter values through algorithms like gradient descent [13]. In our case, the lack of a differentiable objective function makes this approach generally more challenging. However, the GLOnet framework [15, 26], which will be discussed further, offers a promising and adaptable solution that aligns with this type of optimization.

Implicit optimization : Refers to the process of optimizing a model without explicitly specifying an objective function. Instead, the optimization is driven by an implicit objective that is learned or inferred from the data. This means the search space must be explored semi-randomly to approximate the global optimum. This method is particularly interesting because it can handle black-box functions or simulations, which aligns well with our evaluation methods and research interests. However, this technique often relies on chance to achieve optimal results, making it inefficient at times.

3.3.1 Optimization in MEMS accelerometers

The optimization of MEMS devices has witnessed significant progress in various domains over the past decades, with notable achievements in areas such as photonics devices [26] and mechanical devices, like MEMS accelerometers [27]. However, much of the work in this field predominantly focuses on parametric optimization, where the optimal set of parameters defining a specific device's shape is sought. In tandem with this technique, condensing the shape of a structure in a set of parameters with specific ranges is a method that has been used to tackle several problems in various domains [27, 28]. This allows the parameterization of a shape that didn't have any parameters *a priori*, which can then be optimized using a plethora of algorithms. However, such a technique is only possible to apply in specific cases when researchers have access to mathematical models capable of representing the device's structure using a set of parameters.

For instance, Wang et al. [27] proposed the usage of a genetic algorithm to try and optimize parameters that define the shape of a free-form structure of a MEMS accelerometer device. Variations in these parameters will result in devices with vastly different visual features and consequently, variations in the sensitivity and other performance metrics. The algorithm works by introducing random modifications to the parameters of each candidate design which will then be evaluated and compared with the current pool of candidate designs. Then, new designs are included in the pool if they have better

characteristics than those previously seen, and the worst designs are excluded, following a pattern similar to the natural selection present in nature. The loop is then repeated until the designs' performance values converge to a value.

The algorithm used by the authors is particularly interesting since it doesn't need an objective function to be explicitly defined, and its intrinsic exploration capabilities can help it escape the local optimum problem [16] discussed in Section 2.4. However, this type of optimization often fails to account for the full design space, which may result in not finding designs with optimal characteristics.

Additionally, as can be observed in Figure 2.1, our designs are represented by 2D images defining their topology, meaning there are no parameters that define how a device is to be constructed, meaning there are no input parameters that can be optimized. Moreover, even if we tried to use a genetic algorithm to directly create a design by exploring the various possible combinations of pixels in the internal structure, the process would probably be inefficient due to the sheer size of the design space; there are simply too many possible combinations of pixels. For this reason, other methods of optimization, that are capable of exploring a design space efficiently will have to be considered instead.

3.3.2 Active Learning

In supervised learning, data points are paired with corresponding labels, enabling the model to learn patterns and make predictions based on labeled examples. This approach forms the foundation for tasks like classification [29, 30] and regression [31]. However, in real-world applications, while unlabeled data is abundant, labeling can be prohibitively difficult or expensive [32, 33].

To address this challenge, the concept of Active Learning (AL) was introduced [32]. This approach effectively combines labeled and unlabeled datasets for training models. Rather than relying solely on pre-labeled data, it employs an iterative process where the model actively selects the most informative instances from the unlabeled set for manual labeling. By strategically choosing data points that contribute the most to model improvement, the learning process is optimized. This dynamic strategy is especially valuable when labeling resources are limited or costly, enhancing the efficiency of supervised learning by focusing on the most impactful samples.

Perhaps, one of the most common types of AL is pool-based active learning [29, 32], where the unlabeled samples are kept in a pool that is queried for samples to be annotated in every AL cycle. Figure 3.3 shows a representation of what a pool-based AL pipeline looks like.

In essence, AL emerges as a pivotal tool within supervised learning, providing an adaptive approach to data annotation that effectively reduces the number of labeled data points required for model training. Beyond its primary goal of efficient label acquisition, AL reveals a key takeaway: the capacity to influence and bias the output of a model by selectively modifying the composition of the training set. This can prove very useful for

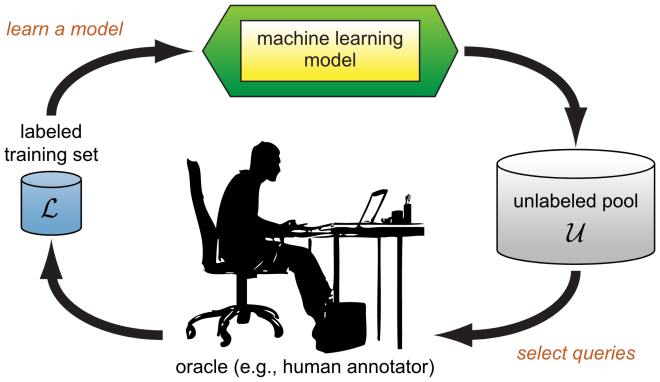

Figure 3.3: Typical architecture of pool-based AL pipeline. A small labeled dataset is used to train an initial version of the predictive model. Then, the model is used to choose samples from the unlabeled set, by measuring the uncertainty or "usefulness" of each sample according to some criteria. Chosen samples are then labeled by an oracle, an entity that can accurately annotate them. The annotated samples are integrated into a new training set and the process repeats until some stop criteria is met, usually a labeling budget.

us, especially if we try to selectively bias the training set towards more performant devices or to augment the training set with samples with a larger design variety.

Zhu and Bento [29] proposed the Generative Adversarial Active Learning (GAAL) technique that leverages GAN models to generate samples that are more valuable for the trained model to learn. In their case, a Support Vector Machine (SVM) classifier was used to predict the classes of images from the *MNIST* dataset, and the GAN model, which was trained on an unlabeled subset of that dataset, was used to generate examples which the predictive model was more uncertain about, meaning samples that where closer to the boundary of the hyperplane defined by a SVM model.

This work has two primary takeaways that might be useful in our case:

1. The synthetic samples from a generative model are used to enrich the training set with data that is deemed as most useful to train a predictive model, meaning possible optimization tasks could leverage the usage of synthetic data to guide a model into generating designs with better performance values.

2. The latent space can be explored using gradient descent techniques to find designs that minimize or maximize an objective function such as the distance to a certain boundary, which is a technique that may be interesting to explore for optimization tasks.

Focusing primarily on the first point, this idea could be taken into consideration with the particular objective of guiding a model towards generating designs with certain properties, say with better Q values, exclusively by tailoring the data on which it is trained. This could act as the foundational concept of a possible naive baseline optimization pipeline to be implemented and evaluated during the execution of this thesis.

Another interesting approach where AL may prove useful is the maximization of designs' diversity which allows finding novel designs with possible superior characteristics using informed exploration approaches. This technique of using AL to traverse the search space to generate novel information has been used, for example, in materials sciences to search for new materials for semiconductors. Xin et al. [34] adapted an AL pipeline to work in tandem with *MATGAN*, a variation of a GAN designed and trained for the generation of hypothetical new inorganic materials with various compositions [35], to find new materials with properties inside a given range. The authors' pipeline managed to find several potential new materials, showing that AL can successfully be used to search the whole design space and identify informative samples to increase the variability of generated data.

3.3.3 Global optimization networks

Global Topology Optimization Networks, or GLOnets for short, is a framework that allows the integration of optimization in generative models by modifying their parameters using gradient descent-based methods to maximize or minimize a specified objective function [15]. It is a non-convex optimization algorithm, meaning there is no guarantee that a global optimum is found [36]. However, the framework employs various techniques that help avoid local optimums. A representation of the framework's high-level architecture applied to general optimizations can be observed in Figure 3.4a.

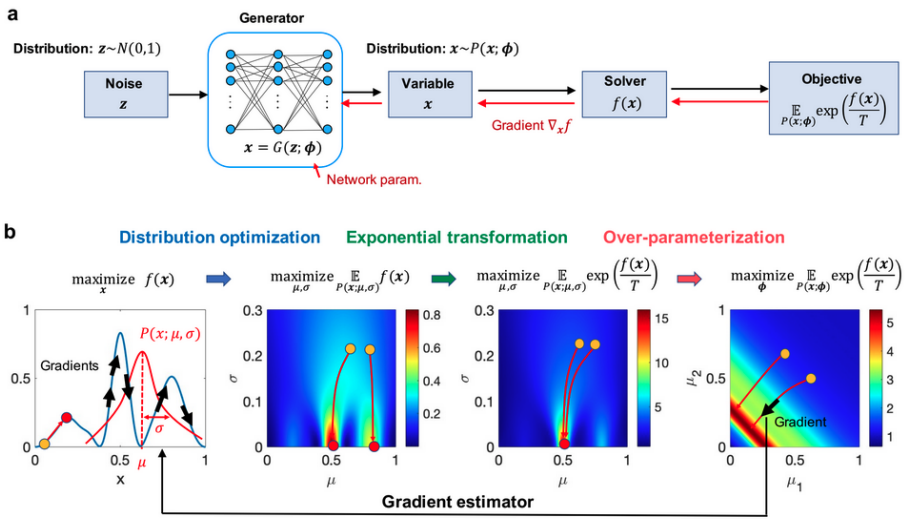

Figure 3.4: (a) Framework of the GLOnet algorithm. A deep generative network produces a distribution of design variables x and the distribution is narrowed around high-performing optima by backpropagation. (b) Visualization of key concepts that enable effective gradient-based optimization within a non-convex landscape, including 1) transforming the optimization problem to the optimization of parameters within a distribution; 2) exponential weighing of the objective function; 3) over-parameterization of the distribution function; and 4) Effective gradient estimation during the network training procedure. Figure adapted from [15].

The key idea of the GLOnet framework is to find the global optimum of an objective function given an input, for instance an image, meaning our problem can be formulated as the following optimization expression:

$$\max_{x} f(x) \tag{3.2}$$

Where x is a multi-dimensional variable, such as the geometrical or physical parameters of a device, and $f(x)$ is the non-convex objective function that characterizes the performance of the device. This can be obtained using a plethora of ways, being the most common a numerical simulator. This function aims to find the x that maximizes the $f(x)$ score, however, since x is something obtained from the generation of a model that has its own parameters, the expression can be adapted to represent a different optimization task:

$$\max_{\phi} \mathop{\mathbb{E}}_{x \sim P(x;\phi)} f(x) \tag{3.3}$$

Where ϕ denotes the parameters of the generator model and x is now in the distribution $P(x;\phi)$. This is an important step because it lets us reframe the optimization problem $w.r.t.$ the parameters of the generator model, making the problem now a matter of finding the right set of generator's parameters that maximize $f(x)$, instead of finding the x directly. Such a modification makes this expression capable of being used in gradient-based optimization algorithms that update the generator's parameters.

Even though the current expression is already fit for optimization, the author suggested including both an exponential function and a temperature hyperparameter in the final objective function, which when combined, can help shape the search space in a way that the global optimum is more easily accessible. Figure 3.4b shows a visualization where the consequences of this improvement can be observed, however, a good definition of the temperature parameter is required for this to work properly. The final objective function then becomes:

$$\max_{\phi} \mathop{\mathbb{E}}_{x \sim P(x;\phi)} \exp \frac{f(x)}{T} \tag{3.4}$$

In practice, this function is approximated using the Monte Carlo method, where a batch of M samples is obtained to progressively refine the approximation. Moreover, the distribution $P(x;\phi)$ is usually implicit, meaning we cannot do direct sampling from it. Therefore, we firstly need to draw the M samples $\{z^{(m)}\}_{m=1}^{M}$ from standard normal distribution and transform them to $\{x^{(m)}\}_{m=1}^{M}$, by passing them through the generator. The approximated objective function L and its gradient $\nabla_{\phi} L$ $w.r.t.$ the network parameters ϕ is then defined as:

$$L \approx \frac{1}{M} \sum_{m=1}^{M} \exp \frac{f(x^{(m)})}{T} \tag{3.5}$$

$$\nabla_\phi L \approx \frac{1}{M}\sum_{m=1}^{M}\frac{1}{T}\exp\left(\frac{f(x^{(m)})}{T}\right)\nabla_x f\ D_\phi x^{(m)} \tag{3.6}$$

Where $\nabla_x f = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_d}]$ are the gradients of $f(x)$ and $D_\phi x^{(m)} = \frac{\partial(x_1, x_2, \ldots)}{\phi_1, \phi_2, \ldots}$ is the Jacobian matrix. As highlighted by the author, the overall performance of the GLOnet is going to be impacted by both the batch size M and by the temperature hyperparameter T. Their careful selection and tuning are imperative for ensuring optimal outcomes and maximizing the effectiveness of the GLOnet in generating high-quality and high-performant designs.

However, Equation 3.6, which shows a method of estimating the gradients known as pathwise gradient estimation, can only be applied if the derivative of f is known, meaning that for problems where f is non-differentiable or a mathematical method to calculate its gradients $w.r.t.\ x$ doesn't exist, this technique won't work.

To circumvent this problem, the authors suggested another way of calculating the gradients without explicitly knowing the derivative of f, by using the derivative of the log probability distribution $P(x;\phi)$ instead. Notably, this strategy, known as the score function gradient estimator or REINFORCE estimator, allows for f to remain unknown, effectively treating it as a black-box function, meaning it could even be discontinuous, and is often employed in Reinforcement Learning due to the non-differentiable nature of most score functions. The expression for computing gradients in this scenario is depicted in Equation 3.7.

$$\nabla_\phi L \approx \frac{1}{M}\sum_{m=1}^{M}\frac{1}{T}\exp\left(\frac{f(x^{(m)})}{T}\right)\nabla_\phi \log P(x^{(m)};\phi) \tag{3.7}$$

Ideally, this function would be enough to calculate the gradient, however, often the distribution of the data $P(x;\phi)$ is unknown and cannot be derived easily from the operations of the neural network that define the generator. However, we can assume that the data follows a specific parametric distribution, such as the Gaussian distribution, that can be learned using sampled data and approximates the distribution of $P(x;\phi)$. However, same as with any approximation, this can introduce some error that affects the optimization step, however, it is the most viable way of obtaining the distribution we need.

3.3.4 GLOnets in practice

A variation of this framework has been successfully applied in the photonics domain. In [26], this framework was adapted to optimize the topology of metragrating structures to improve their efficiency. These structures are capable of diffracting light when a beam passes through them and their topology significantly dictates the efficiency at which they do it, called the deflection efficiency.

In their specific problem, the authors defined the function $f(x)$ as the deflection efficiency for a given design Eff(x). Here, x represents an array of values ranging from

−1 to 1, where −1 corresponds to the void element and 1 represents the material element, showcasing similarities with our case. To obtain this efficiency metric, they employed an electromagnetic solver, which required the use of the Adjoint Variable Method (AVM) to compute the gradients of the function Eff(x) with respect to x. This was necessary because the solver output is not directly differentiable. Armed with the gradients of Eff(x), the authors employed an equation akin to Equation 3.6 to compute the gradients of the generator parameters and subsequently update its weights.

The experiments conducted by the authors reveal that their framework is proficient in optimizing the generation of these devices, achieving comparable or superior performance in most cases compared to standard gradient-based topology optimization methods, which usually optimize a single device at a time. Notably, the GLOnets exhibited inferior performance only in very specific instances, however, the authors argued that further fine-tuning of the batch size M and the temperature T hyperparameters could potentially enhance the performance of the GLOnet algorithm.

Another interesting aspect of this work that may be relevant for our optimization tasks during our thesis, is the integration of a penalty factor in the objective function, which aids in the binarization of the results' vector ensuring that the optimization has less tendency to yield values far from the domain extremes. The penalty function can be observed in Equation 3.8.

$$\textsc{BinPenalty} = \frac{1}{M}\sum_{m=1}^{M} |x^{(m)}| \cdot (2 - |x^{(m)}|) \tag{3.8}$$

This penalty function could also be useful to us, albeit with slight modifications, to ensure the pixels of the image representing the optimized devices are as close as possible to 0 or 1, improving the quality of the optimized results. Additionally, other penalty functions could be introduced, for example, to guarantee that the optimized designs don't deviate from their target frequency values, however, the need for such regularizations will have to be evaluated through experimentation.

Overall, the GLOnet framework stands out as a formidable candidate to serve as the backbone of our MEMS resonator optimization pipeline, offering a stark contrast to the Active Learning-based approach. Its robust capabilities and innovative methodologies position it as a prime contender for revolutionizing the optimization process of MEMS resonators and paving the way for advancements in MEMS resonator technology.

3.3.5 Optimization in MEMS resonators

The field of optimization has been thoroughly applied in several areas of MEMS devices, however, the work done for the particular case of MEMS resonators is extremely scarce, being the work done by Sui et al. [8] the only resonator-related work that shows a glimpse of anything looking even remotely like optimization.

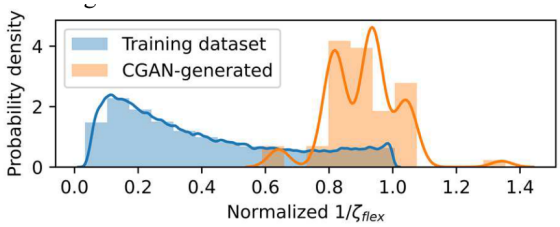

Figure 3.5: Distribution comparison between the training dataset (blue) and CGAN outputs (orange) for normalized $1/\zeta_{flex}$, where ζ_{flex} is the anchor loss of the MEMS resonator in the flexural mode. The CGAN-generated designs are not in the distribution of the training dataset and can achieve $\sim$ 34% improved performance than the best value from the training dataset. Figure adapted from [8].

Even though the authors' work focused mostly on the conditioned generation of MEMS resonator designs using their customized CGAN architecture, they were able to make an interesting discovery that allowed the generation of high-performant designs without having to explicitly optimize the model.

In their architecture, explained in section 3.2, it is possible to pass conditioning parameters to the generator component of a generative model. These parameters act as the target properties of the generated designs and can be many things: the frequency modes' values, the frequency deltas between designs, and even the target Q-factors or anchor losses.

The authors discovered that, by using the normalized $1/\zeta_{flex}$ as the input φ_c, where ζ_{flex} represents the anchor loss for the flexural frequency mode, the CGAN can learn the dataset's distribution is such a way that when using out-of-distribution $1/\zeta_{flex}$ values as the input φ_c, the distribution of the generated designs is shifted towards the desired direction and achieve a $\sim$ 34% higher performance than the best value from the training set. Figure 3.5 shows the results obtained by the authors when using this technique.

However, the optimization capability described appears to be predominantly driven by chance, as the values utilized as input to generate designs with higher performance metrics lie outside the distribution learned by the model. Also, this method proposed by the authors only conditions designs exclusively based on the normalized $1/\zeta_{flex}$ values. This means that no conditioning based on the frequency characteristics is being done, which is not suitable for operators wanting to generate designs with specific properties.

Nevertheless, it remains intriguing to explore this approach, although using the frequency as one of the conditioning factors as well. By incorporating these additional constraints there is potential to generate designs with higher performance values than those in the training dataset. However, it is crucial to acknowledge that working outside the training distribution poses several challenges, and special care must be taken to ensure that the model continues to generate meaningful designs.

4

Dataset and Preliminary Data Analysis

To train valuable machine learning models, it is essential to have a high-quality dataset comprising relevant data for the task at hand. The better the quality the dataset has, the more likely it is that models will be able to learn the desired features, achieving better results.

In this chapter, we will present the dataset used in this work, along some analysis to better understand its structure and data distribution.

4.1 Dataset

To create a model robust enough to generate MEMS resonator designs consistently, we need a strong dataset that contains a wide variety of design topologies and good label diversity. Throughout this thesis, we will make use of a single dataset containing all the information we need, that will be useful for both training the baseline generative model and to delve into the optimization process.

The dataset used in this work was generated and simulated by the KU Leuven team and contains two components:

- **Features**: Contains a set of design topologies of 100×100 pixels, where each pixel represents a material or void in the design. The dataset contains a total of 10,480 samples.
- **Labels**: Contains two labels for each design sample:
  - **Flexural Mode Frequency**: The frequency at which the flexural mode of the resonator vibrates. This label is continuous and represents the frequency in Hz.
  - **Q value**: The quality of the resonator, which is a measure of the damping of the resonator. This label is also continuous and is unitless.

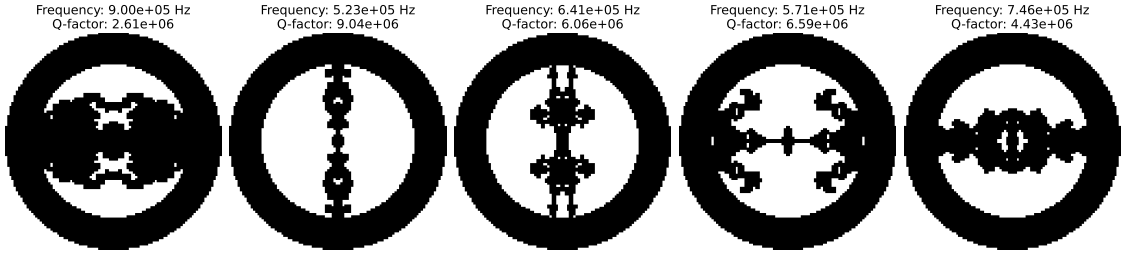

Figure 4.1: Random design samples from the dataset.

The dataset was generated using a Brownian motion algorithm, which is a stochastic process that generates random walks by simulating the random movement of particles in a fluid. The algorithm was used to generate the design topologies, which were then simulated using the COMSOL simulation software to obtain the frequency and Q value properties. In Figure 4.1, we show some random design samples from the dataset, while in Figure 4.2 we show the distribution of the flexural mode frequency and Q value labels.

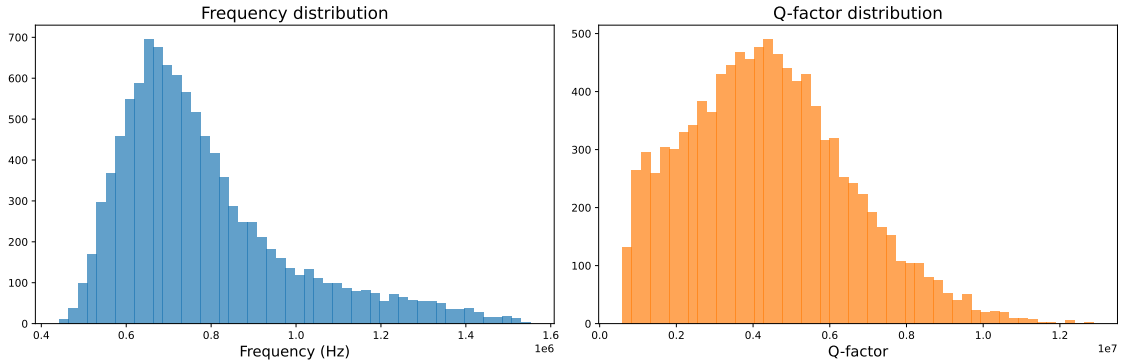

Figure 4.2: Flexural mode frequency and Q value distribution.

One interesting aspect about the flexural mode frequency and the Q value is that they are related to each other in a way that lower frequencies tend to exhibit higher Q values, and higher frequencies tend to have lower Q values. This effect can be observed in Figure 4.3

The presence of this effect in our dataset, informs us that our generative models will likely follow the same trend and will generate designs along the observed region.

Another important aspect required to understand our dataset better, is the concept of **porosity**. The porosity is a metric that represents the ratio between the 'void' pixels and the material pixels, which helps understand how much material is inside each design, without accounting the ring's pixels. Figure 4.4, showcases 3 design samples with different porosity values.

As observed, a low-porosity design contains a lot more material than a high-porosity one, making them very structurally dissimilar. Additionally, this porosity metric also

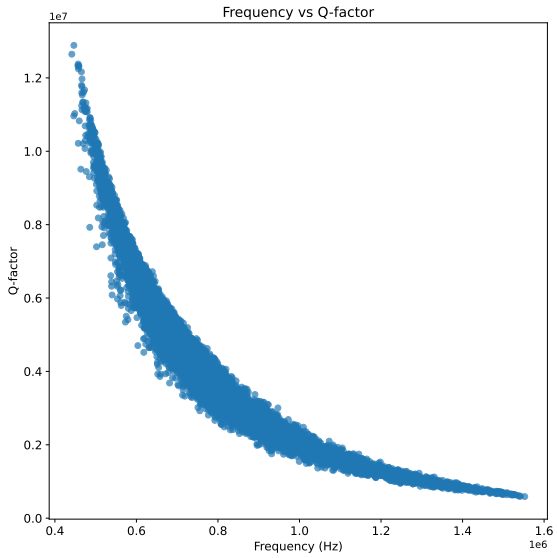

Figure 4.3: Relationship between frequency and Q value. We can observe that lower frequencies exhibit higher Q values, while the opposite happens for higher frequencies.

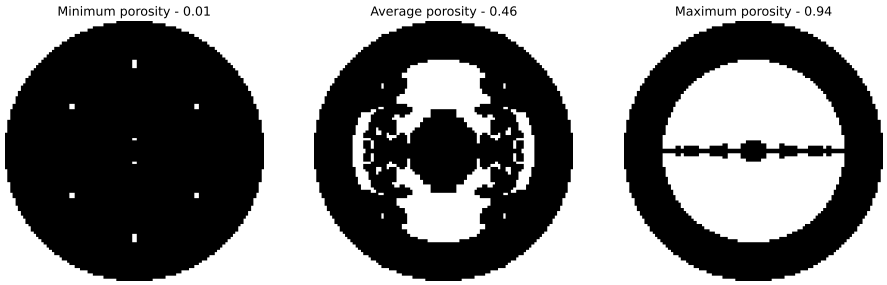

Figure 4.4:

shares a relationship between the flexural mode frequency and the Q value. In Figure 4.5 shows this, where three things are observable:

1. Designs with lower frequencies tend to exhibit higher porosity values, which means that designs with lower frequencies tend to have less material in them;

2. Designs with higher frequencies tend to exhibit lower porosity values, meaning that they typically contain more material in them;

3. For designs in the same frequency range, designs with higher porosity values have higher Q values;

To conclude, in this section we have delved into analyzing the dataset that we will be using to train our generative models. We have obtained important information about the dataset's structure, including the design topologies, labels, and their relationships. This

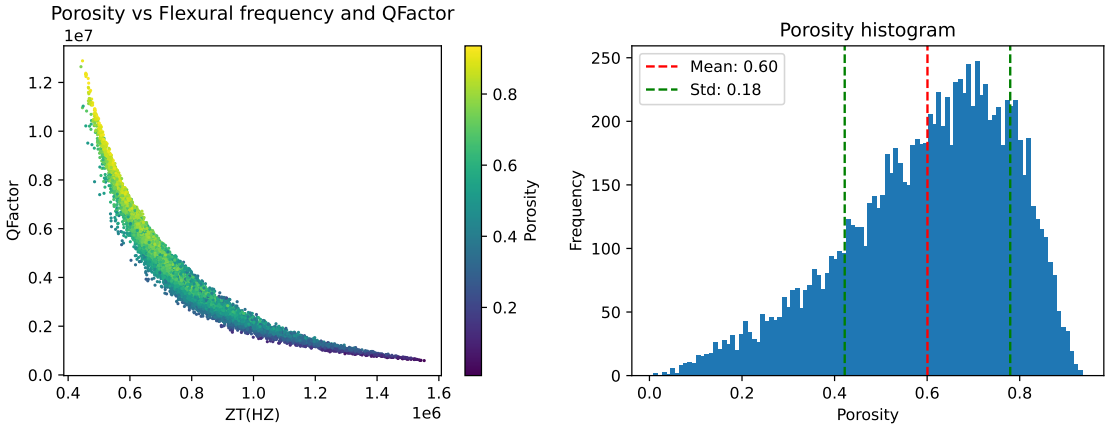

Figure 4.5: On the left, a plot showing the Q value with respect to the frequency of the designs. The color code represents the porosity, where a brighter color means higher porosity. This plot is similar to the one obtained by Guo et al. [10] and shown in Figure 2.4. On the right, a histogram displaying the design frequency for multiple porosity ranges.

analysis will provide valuable insights for training our models and understanding the behavior of the generated designs.

5

Design generation methodology

In this chapter, we will present the methodology used to generate new MEMS resonator designs using generative models. Here, we will describe all the steps involved in the generation process, including the data preprocessing, the generative architecture used, the training procedure, and the hyperparameters used.

5.1 The generative model

Although in Sui et al. [8] the authors used a CGAN model to generate new MEMS resonator designs with great success, our problem domain is slightly different. Here, we are not really interested in optimizing designs of a specific frequency, but rather in optimizing designs generated on the whole frequency range. Therefore, we decided to stick with a simpler generative model, the GAN model, which was straightforward enough to train, while being capable of generating new and diverse designs after just a few epochs.

Our model, whose architecture is described in Figure 5.1, follows the conventional architecture for every GAN model, starting with a generator component, which generates the MEMS resonator designs from a random noise vector, and a discriminator component, which tries to distinguish between real and fake designs.

As for the training itself, we trained the model for a total of 200 epochs, with a batch size of 64 designs. In GAN models, the generator and discriminator are trained in an alternating fashion, as mentioned before, meaning that different optimizers and hyperparameters are used for each component. For the generator in particular, we used the Adam optimizer [37] with a learning rate of 0.001 and betas of $(0.5, 0.9999)$. For the discriminator, we used the stochastic gradient descent (SGD) optimizer with a learning rate of 0.001 and a layer dropout of 0.1. Gradient clipping was also applied to the parameters' gradients of the whole model, clipping at a norm of value 10.

The loss function used to train the model was the binary cross-entropy loss, which is the standard loss function used in GAN models.

Another important hyperparameter in GAN models is the size of the random noise vector, which might affect the model's performance. In this case, we found that a random

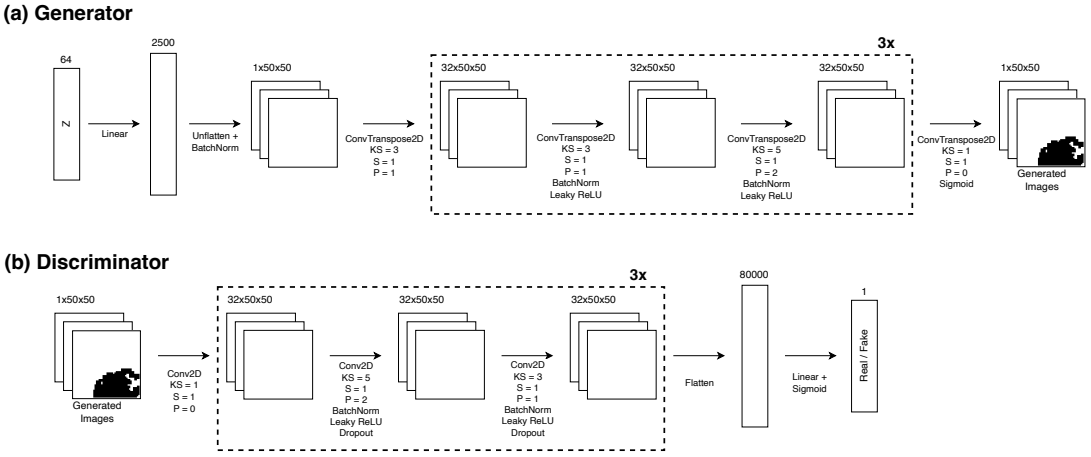

Figure 5.1: The architecture of our GAN model. The figure also shows the parameters associated with each layer/operation: kernel size (KS), stride (S), and padding (P). The architecture contains a total of ≈ 453k parameters, where the Generator contains ≈ 268k parameters, and the Discriminator contains ≈ 185k parameters.

noise vector of size 64 was enough to generate set of designs, with a good balance between design feasibility and diversity.

5.1.1 Data preprocessing

The data contained in the dataset that we used to train the GAN model contains MEMS resonator design figures of 100×100 pixels. However, given that the resonators are symmetric on the x and y-axis, meaning that a quarter of the design is enough to represent the whole design, we decided to crop the images to 50×50 pixels. This reduction in size helps to reduce the total number of parameters in the model and speeds up the training process. Additionally, since all designs have a ring and anchor, and those structures are always in the same position, we decided to remove them from the images. This allows the model to focus on the internal structure itself, instead of the ring and anchor, which can then be added back to the generated designs in a post-processing step.

6

Design Evaluation

Before delving into the results of the generative models and delving into the performance optimization, it is important to understand how we evaluate the designs generated by the models. This is crucial to interpret the results and assert if a batch of generated designs has any meaning at all.

In this chapter, we will discuss several design evaluation methods, which includes several novel metrics to evaluate the quality of generated designs. To the extent of our knowledge, these metrics have not been used before in the context of MEMS resonator design generation, which makes them unique and innovative.

6.1 Design validity and the validity algorithm

One of the most important metrics to evaluate generated designs is their validity, a binary metric that helps us understand if a design satisfies the constraints imposed by the definition of a MEMS resonator. By evaluating the validity of a batch of generated designs, we can understand if the associated generative model has a good performance by being capable of generating meaningful designs, which helps to quickly discard models that are not performing well.

To compute a design's validity, a special algorithm was developed, which takes the design's topology as input and outputs a binary value representing a valid or invalid design. However, before jumping into the specifics of the algorithm itself, it is important to grasp the different ways a design can be invalid. Figure 6.1 shows illustrative examples of a valid design and all the possible ways it can be invalid.

6.1.1 Topological Validity Checking

In the context of our problem, some of the invalid designs can be easily fixed by modifying a few pixels. For example, since we know what an *empty* (ring and anchor only) design should look like, designs missing part of the ring or anchor can be fixed by adding the missing pixels, while designs with extra pixels outside the ring can be fixed by removing them. However, designs with disconnected components or designs that don't have a path

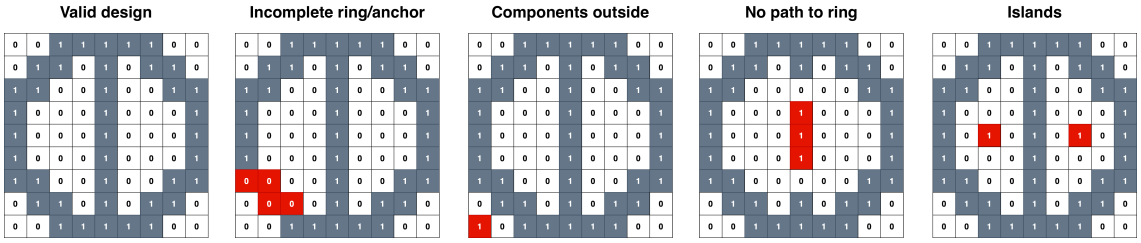

Figure 6.1: Examples of a valid design (left) and some invalid designs (right). The red pixels represent areas that make the design invalid. In the case of the design with no path to the ring, it would also be considered as having islands.

connecting the anchor and the ring have no way of being fixed, meaning they will always be considered invalid throughout the whole process.

As a first method to compute the validity of a design, we used a pathfinding algorithm that traverses the designs' pixels from the anchor until it reaches a pixel from the ring. If the pathfinding algorithm is successful, the design is considered valid; otherwise, it is considered invalid. This process is quite straightforward and easy to implement, but it fails to account for designs with disconnected components, which are also invalid.

As an alternative, following a similar approach, would be to do the same pathfinding process, but starting from every pixel in the design. This way, pixels in disconnected components would never find a path to the ring, making the design invalid. But, as we quickly realized, this method poses a significant computational cost, especially when we try to scale the problem by computing the validity of a large batch of designs, which we do on every training iteration of our generative models. Moreover, this method could traverse the same pixels multiple times, which is inefficient and unnecessary.

To address these issues, a *flood-fill*-based algorithm was developed. The key idea is to start from a known position in the design, and then traverse all the connected pixels until no more pixels are left to explore. This way, we can efficiently compute the validity of a design, by comparing the visited pixels with the original design's pixels. If all pixels are visited, the designs are considered valid; otherwise, which could happen when disconnected components are present, for instance, the design is considered invalid. Figure 6.2 helps illustrate this process. This algorithm resembles the fill algorithm used by image-processing tools like *Microsoft Paint* to fill areas with color.

Comparing to the other alternatives, this algorithm is much more efficient and can detect designs with disconnected components, which is crucial for our problem. However, it can still take a long time to compute for a large batch of designs, taking around 115 seconds to compute the validity of a batch of 512 designs. If we assume the training process to be around 200 epochs, and we compute the validity at the end of each epoch, the total time spent computing the validity would be around 6 hours, which is quite significant.

To address this, there is a simple optimization that can be applied to the algorithm, which can greatly reduce the compute time required: cropping the designs. Since the

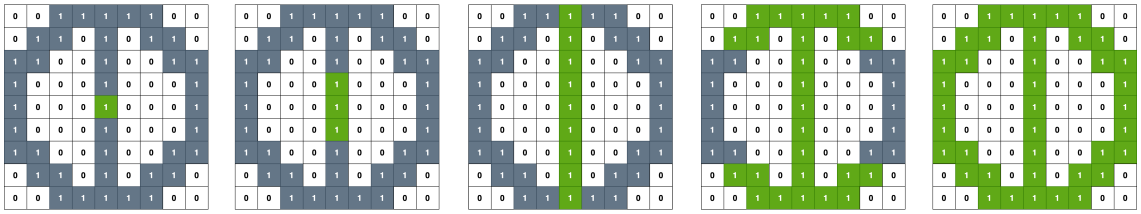

Figure 6.2: Representation of the validity's algorithm process from left to right. The algorithm starts from one of the anchors' pixels and traverses the whole design until all connected pixels are visited. If all pixels are marked as visited, the design is considered valid.

designs are symmetric, meaning that if we assume the design being composed of 4 quarters of 50×50 size and that all quarters can be obtained by rotating and mirroring a single one, we can crop the designs to a single quarter and use it as input to the algorithm. This way, we can reduce the number of pixels traversed by 4 times, which should reduce the computation time by a similar factor. Figure 6.3 shows the runtime performance comparison between using the original and the cropped designs as input to the algorithm.

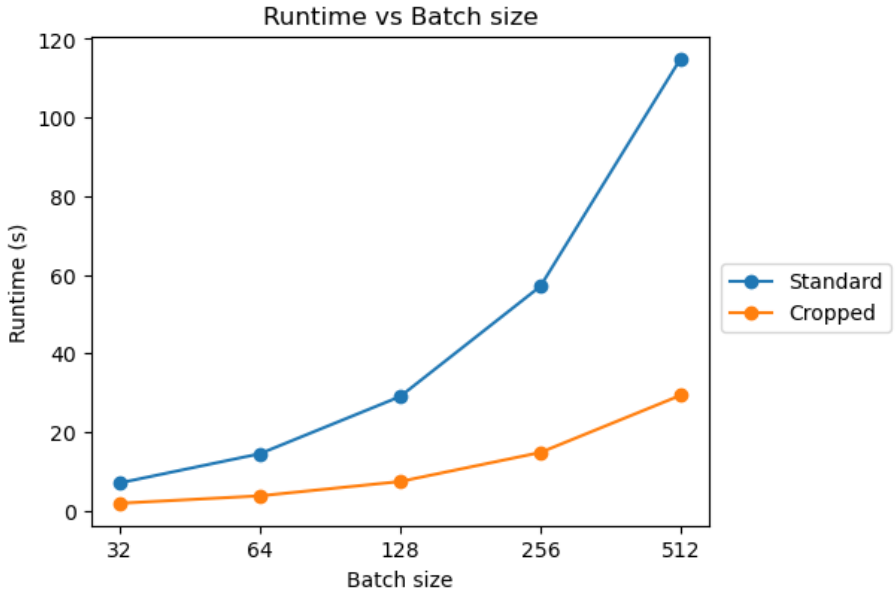

Figure 6.3: Runtime performance comparison between the original and the cropped designs.

As observed, by reducing the designs' size, we can significantly reduce the runtime required to compute the validity of a batch of designs. Whereas using the original designs takes around 115 seconds to compute the validity for a batch of 512 designs, by using the cropped designs, the compute time drops to around 30 seconds, which is 3.8× faster than the original. Assuming the same training process as before, the total time spent computing the validity would now be around 1.5 hours, which is a significant improvement over the

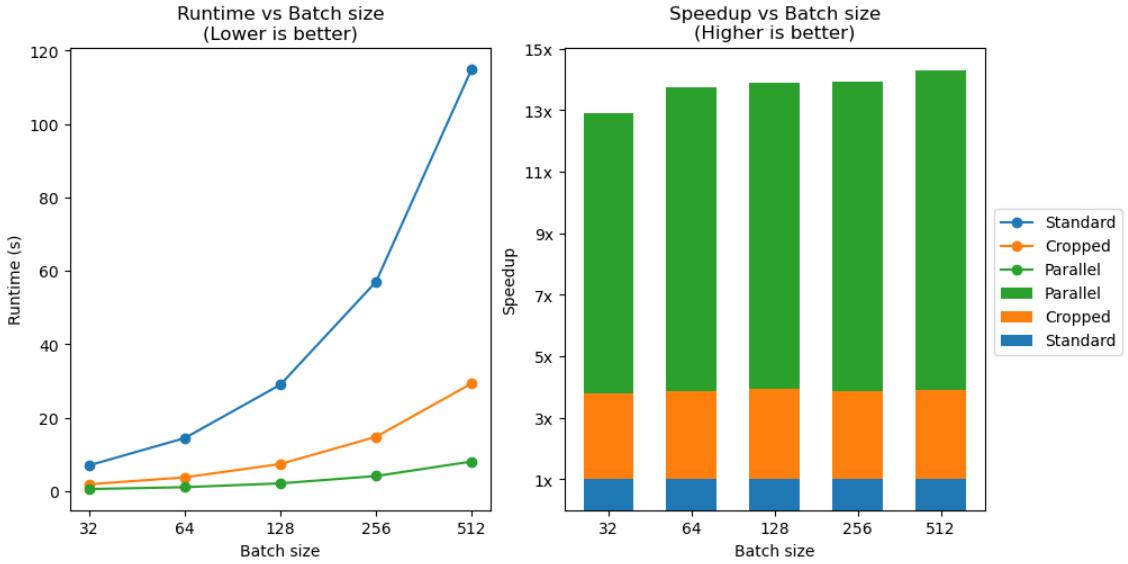

Figure 6.4: Runtime performance and speedup comparisons between the sequential, cropped and parallel versions of the algorithm. The parallel version is run on 4 CPU cores and uses the cropped designs.

original runtime of 6 hours.

However, this is not the only optimization that can be applied to the algorithm. In fact, the algorithm can clearly be parallelized, as the computation of a single design's validity is independent of the other designs in the batch. This means that, to further improve the algorithm's performance, we can leverage multicore computation to check the validity of multiple designs at the same time. Figure 6.4 shows the runtime performance and speedup comparisons between the sequential, cropped and parallel versions of the algorithm.

As we can see, the computation time is once again reduced dramatically, with the parallel version of the algorithm taking around 8 seconds to compute the validity of a batch of 512 designs, which is around 14× faster than the sequential version and 3.8× faster than the cropped version. Again, assuming the same training process as before, the total time spent computing the validity would now be around 26 minutes, which is a significant improvement over the previous runtime of 1.5 hours.

In summary, the validity's algorithm is a fundamental metric used in the evaluation of our generated designs (and in consequence our generative models), that measures if a design satisfies the constraints imposed by the definition of a MEMS resonator. The algorithm we developed is capable of efficiently computing the validity of a large batch of designs, after applying designs' cropping and parallelization techniques. From this point on, we will be using the algorithm's fully optimized version to compute the validity of the generated designs.

6.2 Label prediction

Another important metric to evaluate generated designs is to try and predict their characteristics or labels. This is not a metric per se, as it doesn't evaluate the quality of the designs directly, but it can help us understand the properties of a design without having to simulate it in a FEM software.

One of the methods used to predict the labels of a design using exclusively its topology is to train a Deep Learning model on a dataset of designs with known labels. This was the foundation for the work of Guo et al. [10], which used a ResNet50 model to predict the properties of input designs with high accuracy.

Following this approach, we trained a similar model that predicts both the flexural mode frequency and the corresponding designs' Q values. To ensure consistency with the generative model, the input designs were cropped to a 50×50 size and the ring and anchor structures were removed.

To train the model, we used 80% of the total dataset, with the remaining 20% used as a test set. Additionally, 20% of the train set was used as a validation set to monitor the model's performance during training. The model was trained using the SGD optimizer with an initial learning rate of 0.1 and a momentum of 0.9. The learning rate was scheduled to be multiplied by 0.1 after every 70 epochs, and the training process was repeated for a total of 230 epochs. The loss curves and respective test-set performance is observed in Figure 6.5 and Table 6.1, respectively.

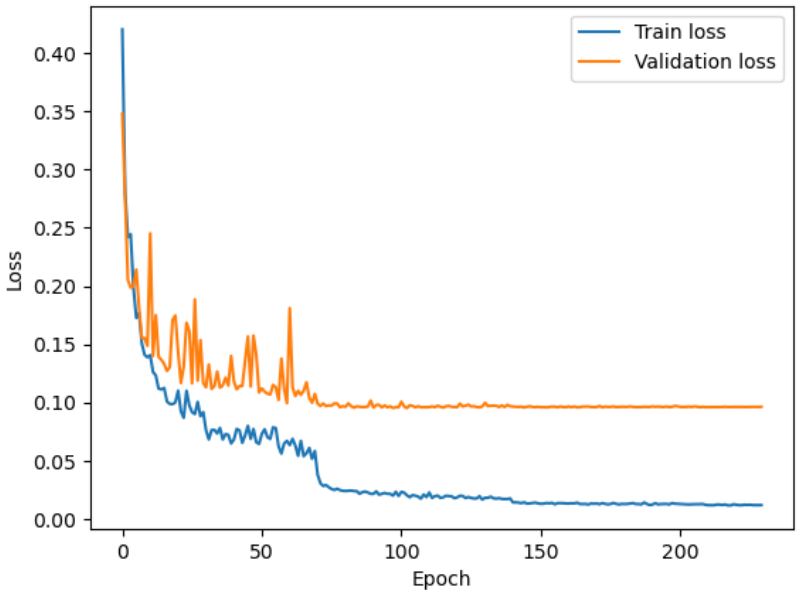

Figure 6.5: Training and validation loss curves for the ResNet50 predictor model.

We observe that the model converges very well, and it is capable of predicting both the designs' flexural mode frequency and its Q value with high accuracy, similar to the

Label	Mean Relative Error	Mean Absolute Error
Flexural mode	2.29%	17525.26 Hz
Q Factor	4.99%	213479.64

Table 6.1: ResNet50 predictor's performance for the flexural mode frequency and corresponding Q factor on the test set, and the mean relative and absolute errors.

one from Guo et al. [10]. However, according to the train-validation curves, the model doesn't appear to be benefiting from the learning rate scheduling, as the validation loss doesn't decrease after the first learning rate change. This is not necessarily a bad thing, as the model is still capable of achieving high accuracy, but it is something to keep in mind for future experiments.

Alas, we will use this model as our baseline to predict the labels of generated designs, thus allowing use to avoid simulating them in a FEM software and not having too much of an accuracy loss.

6.3 Design Matching

Even though a method to predict the labels of designs is crucial to help evaluate the generative models, it is not the only way of measuring their performance. In fact, another evaluation method that is very useful to understand certain group characteristics of designs, such as their diversity, is to compare them with each other directly, by using some sort of method that computes the similarity between them. With this ability, we can perform a multitude of tasks that can help us understand both the generated designs and the generative models better, such as:

- **Proxy label prediction**, which involves using the most similar designs from a ground truth dataset to predict the characteristics of a generated design. Although we have the predictor model to accurately predict design labels, it becomes unreliable when predicting outside the training-set distribution, hence why this method can be useful;
- **Diversity analysis**, which involves using the similarity between designs to try and understand how diverse a batch of generated designs is. This is the most relevant use case for this method;

It is crucial to note that several methods can be employed to approach this problem. However, most of them are based on the fact that visually similar designs should have similar characteristics, which is a good assumption for our problem. Very similar designs, such as those that differ by a few pixels or have features that still resemble each other, should have similar characteristics, such as the flexural mode frequency and the Q values.

In the following sections, we will discuss some of the methods used to compare designs and how they can be used to evaluate the generative models' performance.

Design Label	Mean Relative Error	Mean Absolute Error
Flexural mode	0.16 %	1179.72 Hz
Q Factor	0.4 %	12174.04

Table 6.2: Mean relative and absolute errors between the test-set designs and their closest train-set designs in the label space. To measure the distances, the labels were scaled using the *StandardScaler* from the *scikit-learn* library, however, the errors were computed with the labels in their original scale.

6.3.1 Closest label comparison

Before moving on to more complex methods, it is important to establish a baseline for comparison, which will help us in two ways:

1. Establish a maximum performance threshold, which sets a lower bound for the error of other design comparison methods.

2. Understand how the closest designs in the pixel space compare to those in the label space, which is a key assumption for the following methods;

To compute the similarities for this method, we used a train-test split, where the 'train' set was used to establish the closest designs for each design in the 'test' set. In total, we used 80% of the dataset for the train set and the remaining 20% for the test set. For each test-set design, we compared its label vector against all train-set designs' label vectors to find the closest design and compute the relative and absolute errors between them. To measure the distance, we used the Euclidean distance, which is both straightforward and easy to compute. The results are shown in Table 6.2, and in Figure 6.6 we observe an example of a design from the test set and its closest counterpart from the train set.

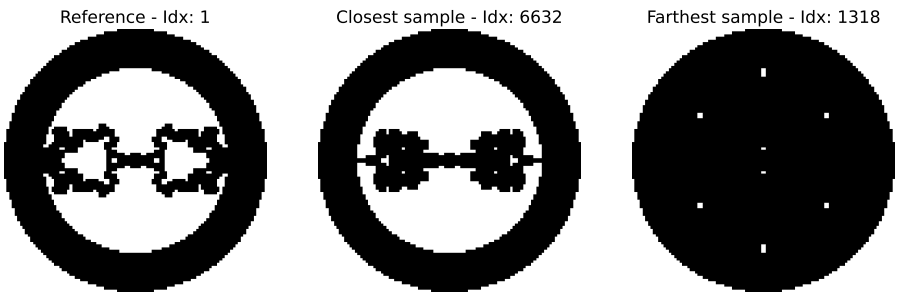

Figure 6.6: Example of a test-set reference design and its closest and farthest train-set designs based on the flexural mode frequency and the Q value labels. This is just an illustration of a random sample from the dataset. No generalizations should be extracted from this figure.

As observed, the label error between the test-set designs and their closest train-set counterparts is very low, which is expected since we are using the label vectors themselves to establish the distance value. Additionally, observed random samples from the dataset

showed that most designs tend to have similar topology features to the closest design, while differing from the farthest designs, which solidifies our initial assumptions.

Although a seemingly superfluous form of comparing designs, this method establishes a lower bound for further design comparison techniques. Also, the ability to establish that the closest designs in the labels space are indeed similar to each other in the pixel space is a key assumption for the methods that we will discuss in the following sections.

6.3.2 Pixel-space design comparison

Moving on to more complex design comparison methods, a straightforward way to compare designs is to compare them pixel by pixel. This method is very intuitive and easy to implement, as several libraries provide functions to compute the distance between two images, and it requires no prior knowledge of the designs' labels. We will use this approach as our baseline that will be compared against other methods.

Following the same train-test split and the same data transformation process as before, we flattened the train and test designs into a 1D array and used the Euclidean pairwise distances [1] to compute the distance between the designs. The result of this operation is a matrix, where each element i, j represents the distance between designs i and j from the train and test sets, respectively.

Next, by sorting the matrix by row, where each row represents each design from the test set, we are able to find the most similar designs from the train set for each design in the test set, and thus compute the relative and absolute errors between their labels. Table 6.3 shows the results obtained using this method, when using both the Euclidean and Cosine distances to compute the pairwise distances.

Design Label	MRE (Euclidean Dist.)	MRE (Cosine Dist.)
Flexural mode	**10.26%**	15.62%
Q Factor	**19.92%**	24.53%

Table 6.3: Mean relative and absolute errors between the test-set designs and their most similar train-set designs using the Euclidean distance.

As observed, the Euclidean distance is capable of reaching much better results than the Cosine distance for all designs' labels. This is expected, as the Euclidean distance is using the pixels' values directly, while the Cosine distance is trying to establish a sense of *vector direction*, which is not as relevant in this context. However, these results are suboptimal, as they can still be improved using more sophisticated methods.

6.3.3 Autoencoder-based design comparison

Another method of computing the similarities between two given designs is to somehow encode them to a lower-dimensional feature space, which captures the most important

[1] Using the pairwise_distances function from the scikit-learn Python's module

information about the designs. This new feature space, often called the embedding-space or latent-space, if capable of non-linear correlations correctly, can help us in two ways:

1. This new feature-space may be capable of capturing the most important features of a design, which can help us better compare/cluster them;

2. Since this feature-space is more compact than the original pixel-space, we can expect the comparison/clustering to be more efficient;

However, to obtain this new feature-space, where each design has a corresponding vector representation, we need some kind of model capable of encoding the designs into this space, often-times called an *Encoder*. One of the most popular models used for this purpose is the Autoencoder, a two-part neural network capable of encoding a design into the embedding-space that we want. This model's operation involves getting the original pixel-space design as input, encoding it to the embedding-space, and then attempting to decode it back to the original pixel-space, as shown in Figure 6.7. At the end of the encoding-decoding process, we expect the reconstructed design to be as close as possible to the original input design, by computing the Mean Squared Error (MSE) loss between them. If the loss is low, it means that the autoencoder is capable of capturing the designs' features correctly, Additionally, given that the embedding vector has a lower size than the original pixel-space design, it compresses the designs' information, often capturing the most important features.

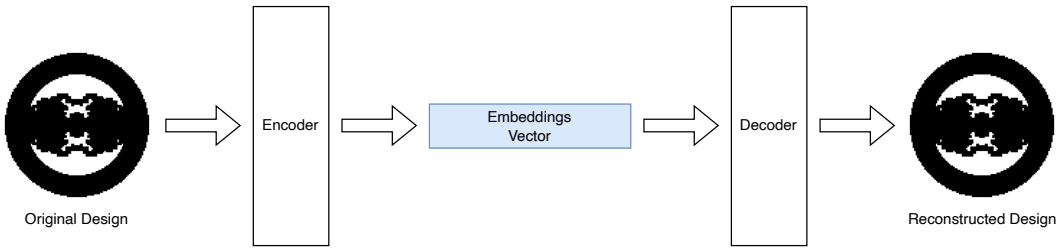

Figure 6.7: Overview of the operation of an autoencoder model.

Moving to the practical side, given that we are using images as input, we will use a convolutional autoencoder, which uses convolutional layers to extract features from the images. Figure 6.8 shows the high-level architecture of our autoencoder.

Then, following the same exact train-test split as the one used in the ResNet Predictor described in Section 6.2, we trained our autoencoder model to learn to encode and decode the designs. Figure 6.9 shows the validation-set loss curves, while Figure 6.10 shows the original and reconstructed designs' examples from the test set.

As observed in Figure 6.9, the losses converge quite well, with the train and validation losses closely following each other, which indicates that the autoencoder is capturing the designs' features correctly. This is further confirmed by the reconstructed designs, which preserve the original designs' features very well, with only a few minor differences that

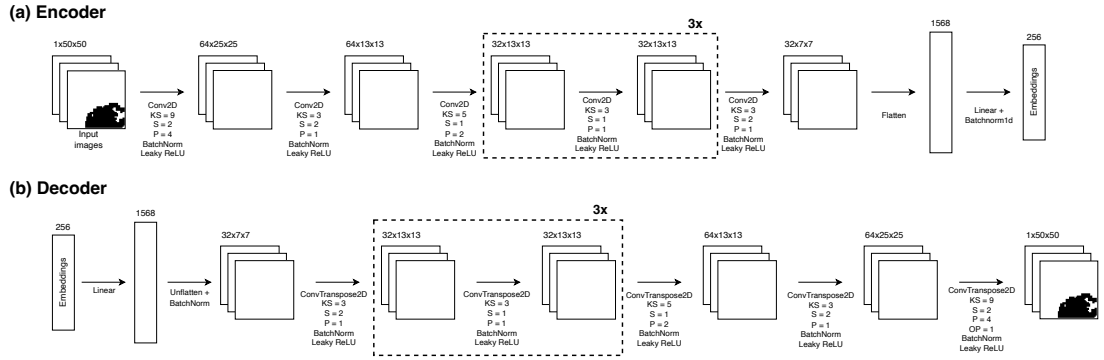

Figure 6.8: Architecture of the autoencoder used in our work to compress resonator design topologies to the embeddings space. The figure also shows the parameters associated with each layer/operation: kernel size (KS), stride (S), padding (P), and output padding (OP).

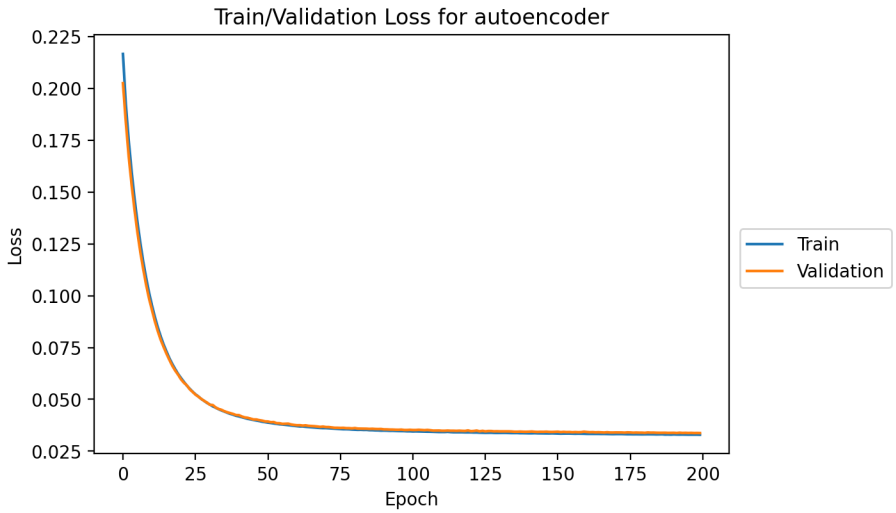

Figure 6.9: Autoencoder's train and validation loss curves.

are expected in this type of model. Even very small details, such as the small holes in the designs, are captured by the autoencoder, which is a good sign that the model is working as expected, while demonstrating its ability to compress the designs' features.

The correct way to evaluate the autoencoder's performance would be to use a dataset of designs with their known designs to try and observe if the autoencoder is capable of preserving that closeness in the embedding space. However, since we don't have this information, we will have to use the label information instead, which although not ideal, will still give us a good idea of the model's performance, since we know that similar designs should have similar labels.

After training the model, we encoded all the designs from the train- and test-sets to the embedding space and computed the pairwise distances between the resulting embedding vectors. Then, we selected the closest train-set designs for each test-set design and computed the labels' mean relative errors between them. This way, we can understand

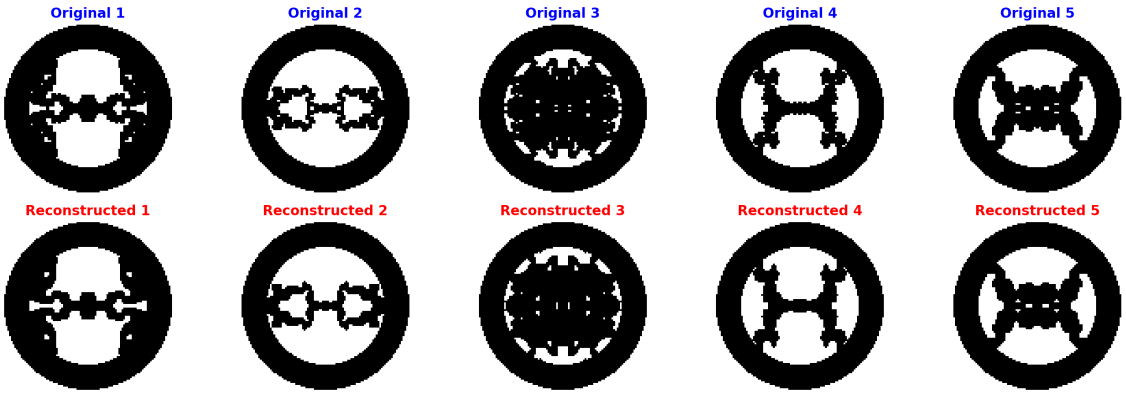

Figure 6.10: Original and reconstructed designs from the test set using the autoencoder. The designs were reconstructed using the decoder component of the autoencoder and then converted back to the 100x100 representation by replicating across the 4 quarters. Additionally, designs were passed through a threshold process with reference point value of 0.5 to obtain the final binary representation.

Design Label	Pixel-Space	Autoencoder
Flexural mode	**10.26%**	11.72%
Q Factor	**19.92%**	28.88%

Table 6.4: Mean relative error comparison between a test-set design and their most similar train-set designs using the autoencoder-based design comparison method. The pixel-space design comparison method's results are also shown for comparison.

how good the autoencoder's embedding space is by using the label closeness as a proxy for the designs' closeness. The results are shown in Table 6.4.

As we can observe, the autoencoder-based design comparison method is incapable of achieving better results than the pixel-space design comparison method. For the Q factor especially, the differences can reach a whopping 17%, which is quite significant.

This is rather surprising, as we expected the autoencoder's embedding space to at least be similar to the pixel-space method, if not better. Although for the flexural mode frequency the difference is not as high ($\approx 1.5\%$), the pixel-space comparison method still outperforms the autoencoder-based method for both labels, which indicates that similar embedding-space vectors surprisingly do not represent designs that are close in the label space, which is one of our key assumptions.

Despite that, we can still force the autoencoder's embedding space to capture the designs' labels information and thus align them with the label space, which should help us achieve better results. We will discuss this in the following section.

6.3.4 Label-Conditioned Autoencoder

As discussed in the previous section, the autoencoder-based design comparison method is incapable of beating the pixel-space design comparison method, when comparing the closest designs' labels, most likely due to the fact that the autoencoder's embedding space is not aligned with the label space. In this section, we will propose a new method that tries to improve the autoencoder's performance by forcing it to learn a representation that is capable of capturing the designs' labels information, which should help us achieve better results when comparing designs.

As detailed before, we want to force the autoencoder to learn a representation that is follows the following criteria:

1. The designs' visual features are successfully captured in the embedding space, meaning that from the embedding vector a successful reconstruction of the design's topology should be achieved;

2. The designs' labels are also captured in the embedding space, which, according to the assumption that similar designs should have similar labels, should help us achieve better results when comparing designs;

So, we developed a new method that combines the previous autoencoder method with the designs' labels information. The key idea is to use the labels as an additional prediction target for the autoencoder, which will force the autoencoder to learn a representation that is capable of capturing both the designs' visual features and their labels' information. This way, when we encode the designs to the embedding's space and use it to compare designs, we will also be including the labels' information in the comparison process, which should help us achieve better relative error results.

To achieve this in practice, there are two alternatives we can consider:

1. **Adapting the decoder**: Modifying the decoder to both reconstruct the images and predict the labels;

2. **New regressor structure**: Adding a new structure to the autoencoder that is capable of predicting the designs' labels after they have been encoded to the embedding space;

Even though the first method is more straightforward and does not require any additional components, it is not ideal, as the decoder is not capable of reconstructing the designs as well as the autoencoder with the additional structure. This was expected since the decoder, which has a structure similar to our encoder, is likely incapable of handling the additional stress of doing both the reconstruction and label prediction task at the same time.

This prompted us to use the second method, which uses an additional regressor component that predicts the designs' labels after they have been encoded to the embedding

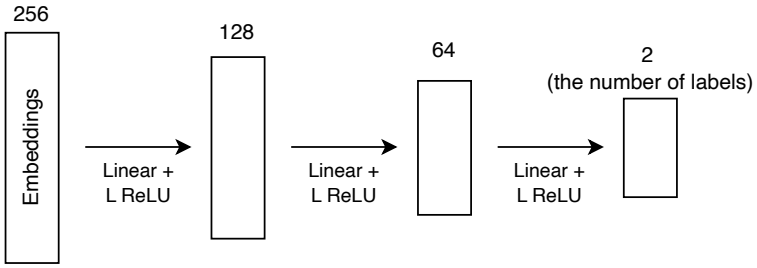

Figure 6.11: Regressor component of the autoencoder with label biasing.

space. The regressor component is a straightforward neural network with fully connected layers and two outputs, one for each label, as shown in Figure 6.11.

This newly added component works in tandem with the decoder component and just like the decoder, is no longer required after training has completed, as its only purpose is to help the encoder part bias the embeddings to capture the designs' labels information. Additionally, this component, like the decoder, is only used during the training process, which means that the autoencoder can still be used to encode designs in cases where labels are not available, as the regressor component is not required for encoding. Cases like these will happen when we use the autoencoder to encode the generated designs, which will then be used in further analysis.

In practice, the training process is quite similar to the previous autoencoder. However, we now have an additional loss term that forces the regressor to predict the designs' labels correctly, which we will call the **labels' loss** and is computed using the MSE loss function. This loss term is then added to the reconstruction loss and the total loss is minimized via backpropagation. The training process's results are shown in Figure 6.12, which shows the loss curves and the reconstruction and regression losses for the train- and validation-sets or an autoencoder with embedding size 256.

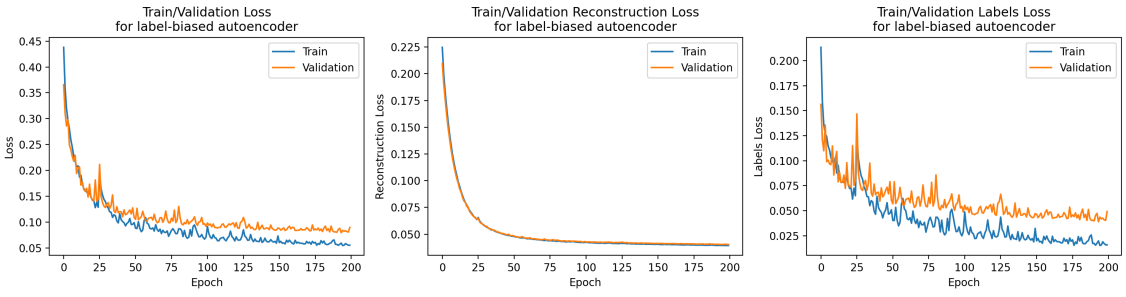

Figure 6.12: Train and validation loss curves for the label-conditioned autoencoder model with an embedding size of 256. The reconstruction and regression losses are also shown.

On one hand, the labels' loss is quite low, indicating that our model is capable of predicting the designs' labels with high-accuracy after the designs have been encoded to the embedding space. This proves that the autoencoder successfully captures the label

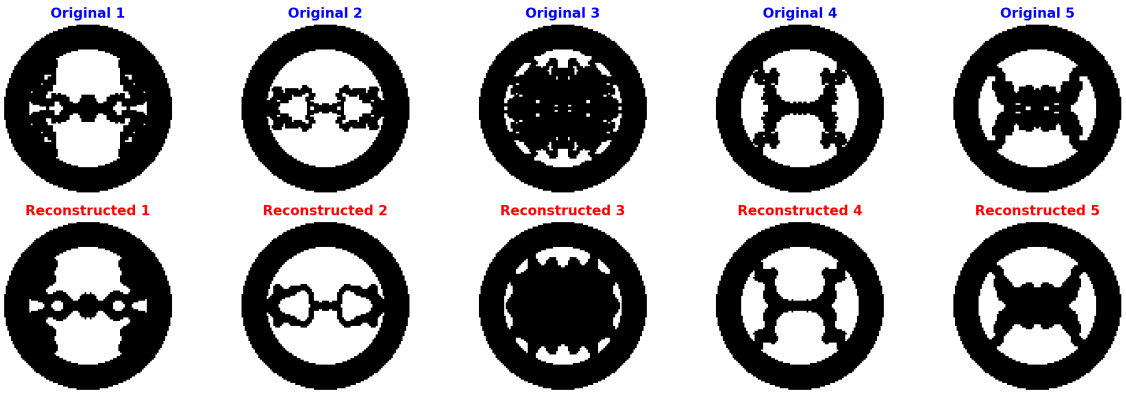

Figure 6.13: Original and reconstructed designs obtained by encoding and decoding designs from the test set using the label-biased autoencoder. The designs were reconstructed using the decoder component and then converted back to the 100x100 representation through a simple quarter replication process. Additionally, designs were passed through a threshold process with reference point value of 0.5 to obtain the final binary representation. The designs shown were selected randomly and used exclusively for visualization purposes. No generalization should be inferred from these.

information in the embedding space.

On the other hand, the reconstruction loss is a little higher than the standard autoencoder, which can be explained by the fact that the autoencoder must now learn to encode additional information in the embedding space, which might be hindering the reconstruction process. Figure 6.13 shows the original and reconstructed designs from the test set, which, if we observe closely, are a little less accurate than the standard autoencoder, but still manage to capture the overall design topology correctly.

To evaluate the label-conditioned autoencoder designs comparison method, just like the previous methods, we encoded all the train- and test-set designs to the embedding space and computed the pairwise distances between the obtained embedding vectors. Since in the previous method it was unclear whether the Euclidean distance was the best choice, we still used both distance metrics in this method to compute the similarities between the designs. The mean relative errors obtained for this particular method are shown in Table 6.5.

As observed in the results, the relative error is considerably better than that observed in the standard autoencoder comparison and the pixel-space comparison methods, for both labels. This indicates that including the label information in the encoding process is beneficial for the autoencoder's design comparison performance and that it will be the best method to use in further experiments.

In summary, the label-conditioned design comparison method is capable of achieving very good results, which will be the foundation for our variability and diversity analysis processes in the following sections. Per the results, this method is capable of outperforming the standard autoencoder and the pixel-space design comparison methods by quite a

Design Label	Pixel-Space	Autoencoder	Label-Conditioned AE
Flexural mode	10.26%	11.72%	**5.33%**
Q Factor	19.92%	28.88%	**11.21%**

Table 6.5: Mean relative error percentage comparison between a test-set design and their most similar train-set designs using the pixel-space, the standard autoencoder and the label-conditioned autoencoder design comparison methods.

margin, which further solidifies its position as our current best model for comparing designs.

6.4 Batch diversity

Following the creation of a method to compare designs, we can now use it to analyze certain aspects of our generated designs, which directly relates to the generative models' performance that generated them.

The first thing we will analyze is the design diversity of a single generated batch. This metric measures how diverse the output of a generative model is, which is crucial to understand if a wide range of designs are being generated or if the model is restricting itself to only a few design choices. On the same note, this metric can also be used for earlier detection of mode collapse, where a model starts generating the same designs over and over again, which is a common problem in GAN models [38].

To compute the batch diversity, we used the label-biased autoencoder model to encode generated designs in a batch to the embedding space, ensuring that designs' are represented in this lower-dimensional space. Then, we computed the pairwise distances between the resulting vectors, just like in the previous methods, and obtained a matrix where each element i, j represents the distance between designs i and j from the batch. However, because we don't want to compare the same design with itself nor compare the same pair of designs twice, we ignore both the diagonal and the upper triangle of the matrix and compute the mean distance between the remaining elements. This mean distance is then used as a metric to evaluate the batch diversity, where a lower mean distance indicates a more diverse batch.

The distance metric used when comparing the embedding vectors is the cosine similarity, which outputs values between -1 and 1, where 1 indicates that the designs are identical and -1 indicates that they are completely different. By computing the average of the distances between the designs' vectors that maybe be similar or dissimilar, we obtain batch similarity values close to 0, which is indicative of a more diverse batch. On the other hand, if designs are very similar, the average distance will drift towards 1, which will indicate that the batch is not very diverse.

To quickly and naively verify this method, we obtained from our testing set k designs and then replicated them n times to create a batch of 500 designs, where only k designs

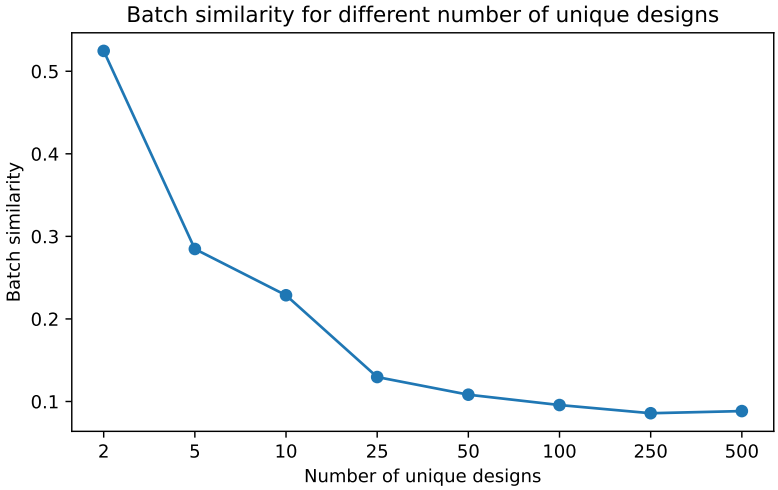

Figure 6.14: Batch similarity values for different unique designs counts. The batch similarity represents the inverse of the diversity, where higher similarities means lower diversities. The values presented are a mean of 3 different runs to minimize instability.

are unique. For example, if we use $k = 2$, and $n = 250$ we will have a batch of 500 designs, where each design is repeated 250 times. This method, which models the worst case scenario for batch diversity (some designs are exactly the same), allows us to control the design diversity inside a batch and observe, by changing the number of unique designs k, how the batch diversity metric changes. For this experiment, we used $k \in \{2, 5, 10, 25, 50, 100, 250, 500\}$ and $n = 500/k$ to ensure that the batch size is always 500, and measured the batch diversity metric for each case. The results are shown in Figure 6.14.

As we can observe, the batch similarity metric decreases as the number of unique designs k increases, which is expected, as the more unique designs we have in a batch, the more diverse it will be, hence the less similar the designs are to each other. Also, as expected, the metric converges to a value close to 0 as k approaches 500, proving our hypothesis that more diverse batches will have a lower batch similarity metric.

Empirically, however, we will observe that most generative models trained will converge to a batch similarity value in the $[0.1, 0.2]$ interval, which will be used as our reference for a good batch diversity. If a model is capable of achieving a batch diversity value close to this baseline, we can consider it to be generating a diverse set of designs.

6.5 Design variability analysis

Having established an initial method to measure the diversity of designs generated in a single batch, we can now use some of the methods discussed to extract additional information from the generated designs. An interesting aspect to analyze, besides the batch diversity, is the similarity of the generated designs to some design groups that are present in our dataset.

In fact, our dataset contains designs that have many different characteristics, such as different holes shapes and sizes, different anchor-to-ring bridge shapes, etc., which can be used to group designs into several similar-looking groups. For example, we can group designs that have a specific anchor-to-ring bridge shape, by using techniques similar to our similarity comparison methods, in order to measure two extremely important things:

- **Dataset balance**, which will help us understand if the dataset that we are using to train our generative models is balanced, meaning that it has a good representation of all the possible design groups;

- **Model coverage and balance**, which will help us understand if the generative models in question are capable of generating designs that cover all the design groups present in the dataset, and if they do so in a balanced way;

Firstly, to split the dataset into groups, we used the K-Means clustering algorithm and our label-biased autoencoder embedding vectors to group the training-set designs into several clusters. Because the number of clusters is a hyperparameter that must be defined beforehand, we used the Elbow method to determine the optimal number of clusters. This method involves training the K-Means algorithm with a wide range of cluster numbers and then plotting the sum of squared distances between the designs and their respective cluster centers. Then, we observe the plot and select the number of clusters where the sum of squared distances starts to decrease at a slower rate, which is called the *elbow point*. This point is the optimal number of clusters, as it indicates that the clusters are well-defined and that adding more clusters will likely not improve the separation between design groups.

In our particular case, we trained the K-Means algorithm with a range of cluster numbers from 2 to 100, which is shown in Figure 6.15, using the *KElbowVisualizer* from the *yellowbrick* library.

After determining the optimal number of clusters and training the final K-Means algorithm with 21 clusters, we obtained the cluster assignments for all the designs in the training set and then computed the distribution of designs in each cluster. Figure 6.16 shows the distribution of designs in each cluster.

As observed, some clusters have a higher number of designs than others, which indicates that the dataset is not balanced and that some designs have more representation than others. This might influence the generative model to generate designs that are similar to the designs in the clusters with more designs, which is not ideal. However, even the clusters that have fewer designs still have a considerable number of them, which might be enough for the models to learn to generate these designs. Though, this is something that we will have to keep in mind when analyzing the generative models' performance.

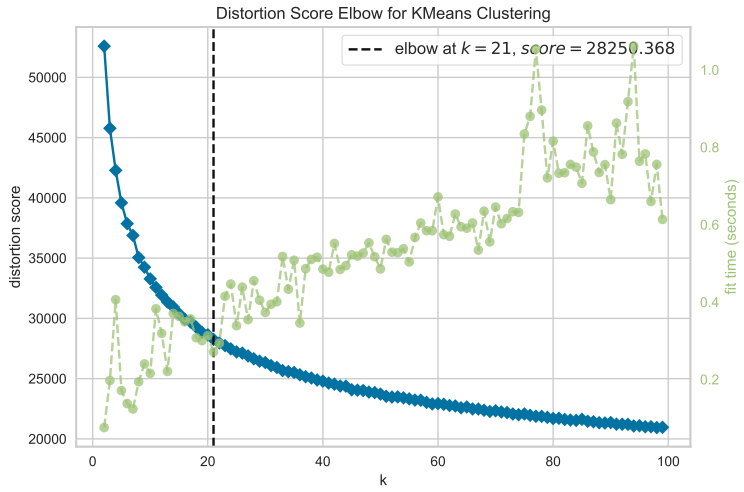

Figure 6.15: K-Means distance results for all models trained with 2 to 100 clusters. The dashed line represents our elbow point, which represents the model trained with 21 clusters.

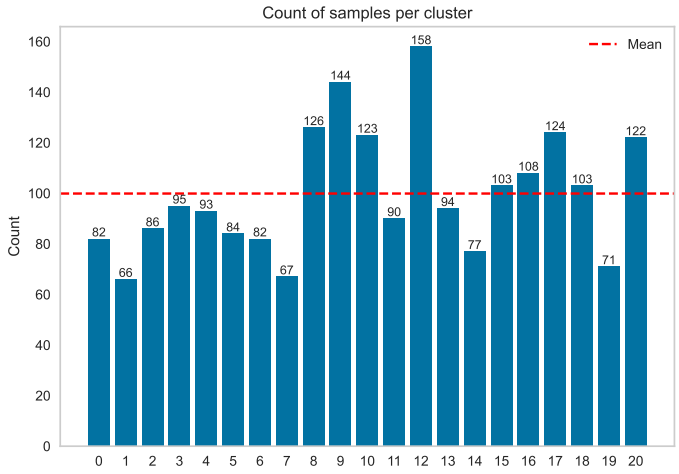

Figure 6.16: Test-set designs distribution across all clusters. The dashed line represents the mean number of designs in all clusters.

6.5.1 Group Design Prototypes

When clusters are computed, what we are essentially doing is grouping designs that are similar to centroids, which are points in the embedding space that represent the center of the cluster.

Since these points are in the embedding space, meaning they are embedding vectors after all, we can decode them back to either the pixel-space using our decoder component or to the label space using our regressor component in order to better understand what each of these centroids represent. This is useful when a generative model is having trouble generating designs for a given cluster, as we can use the centroid designs as a reference

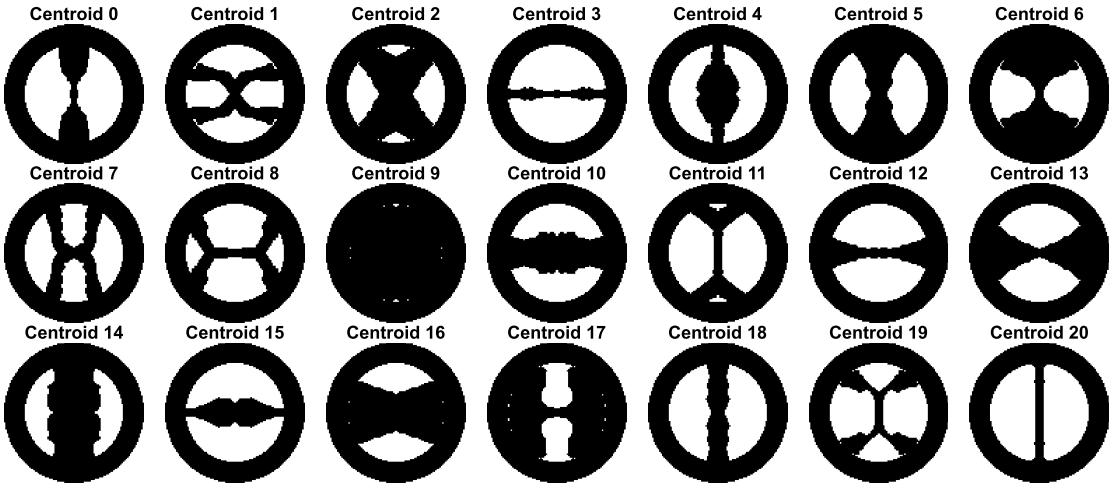

Figure 6.17: Designs obtained from decoding the cluster centroids embedding vectors. Each cluster represents a specific design topology.

point to understand what the model is missing.

To decode the centroids, we used the decoder part of the label-biased autoencoder model to decode the centroids back to original pixel-space designs. Figure 6.17 shows the decoded centroids for each one of the 21 clusters.

As we can see, the decoded centroids are quite different from each other, as expected, which indicates that each centroid is capturing a different design group. Also, designs that are inside the same cluster tend to be similar to the cluster centroid's topology, which is a good sign that the clustering process was successful. Figure 6.18 shows a few examples of designs from each cluster, which share some similar traits with the centroids decoded topology.

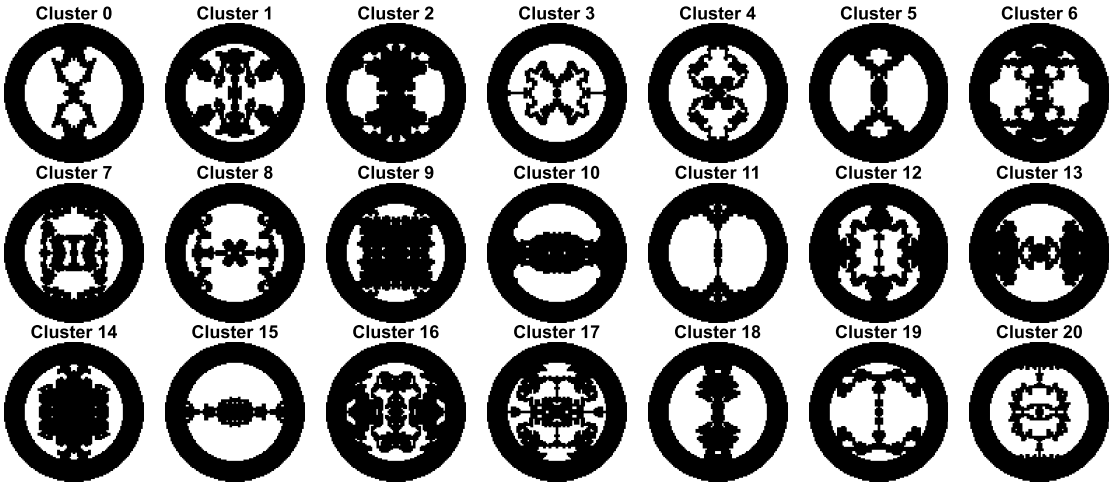

Figure 6.18: Random designs from each cluster obtained using our K-Means model. Each design was obtained from random sampling the 5 closest designs to their specific cluster's centroid.

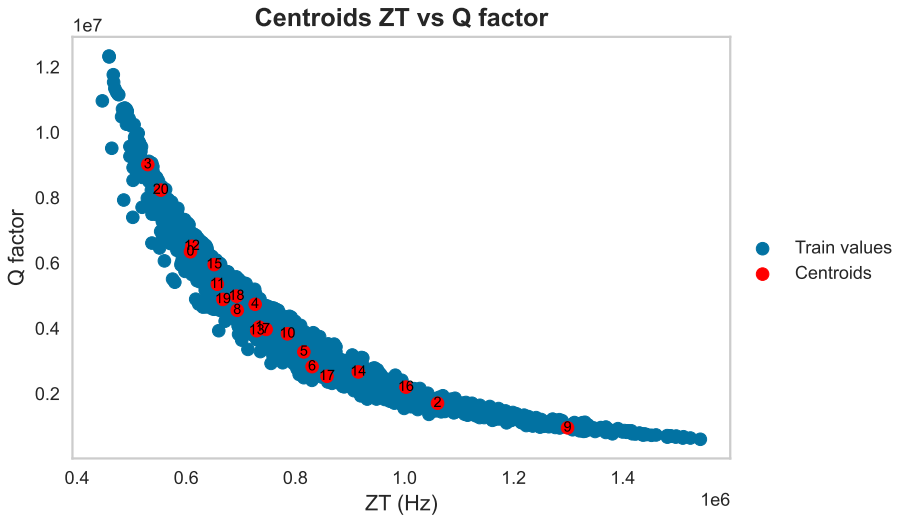

Figure 6.19: Labels obtained from decoding the centroids' embedding vectors, compared to the dataset's test-set labels' distribution.

Additionally, centroids can also have their labels decoded using the regressor component, which will help us understand the labels' distribution for each cluster and if it covers the entire dataset. Figure 6.19 shows the decoded labels for each one of the 21 clusters.

As we can observe, the centroids' labels cover almost all parts of the dataset, with the notable exception being designs with higher frequencies. This could be due to the fact that these designs are similar between each other, and minimal differences can create quite large frequency differences. Most clusters are centered between the range [0.6, 1.0] MHz, which contains not only the most designs in the dataset, but also the range where designs have more variability due to the higher Q-factor span.

6.5.1.1 Coverage and Balance Metrics

With this, we introduce two new metrics that will be used during generative model evaluation:

1. **The coverage**, which measures the percentage of clusters that have designs generated by the model;

2. **The balance**, which measures the overall balance of the number of designs in each cluster;

The coverage metric is quite straightforward, as all we need to do is count the number of clusters that have at least one design generated by the model and divide it by the number of clusters. This metric is useful to understand if the model is capable of generating designs that cover all the design groups present in the dataset, which is crucial for a generative model to be considered successful.

The balance metric is a little more complex, as it requires counting the number of designs in each cluster and then calculating the mean and standard deviation of all values.

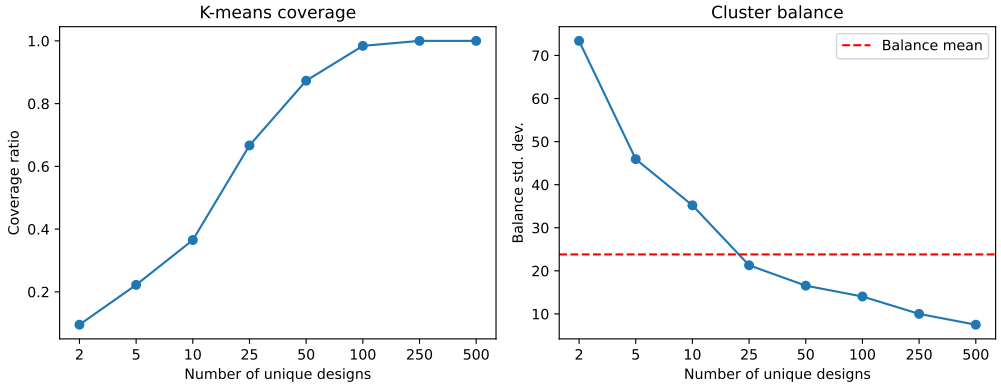

Figure 6.20: Coverage and balance values for the artificial batch created using the method described in Section 6.4, for validating the batch diversity method. The balance metric shown is the standard deviation. The balance mean is fixed at 23.8 as the number of designs never changes.

This metric helps us verify if the model is generating designs in a balanced way, by checking if the standard deviation is too large. The closest the standard deviation is to 0, the more balanced the model is, as it indicates that the model is generating designs for all clusters in a similar proportion. On the other hand, a higher standard deviation indicates that the model is generating a disproportionate number of designs for some clusters, which is not ideal and could indicate that the model is not learning to generate designs for all design groups.

If we use the same validation method used in the batch diversity method, where we replicate designs from the test set to create an artificial batch with 500 designs, we can use the cluster assignments to compute the coverage and balance metrics for the generative model. Figure 6.20 shows the coverage and balance (standard deviation) metrics for multiple variations of the unique designs k.

As we can observe, the coverage and balance metrics expectedly converge quite differently. The coverage increases towards 1.0, which indicates 100% cluster coverage. However, this happens only when there are 250 and 500 unique designs.

On the other hand, the balance standard deviation decreases from a value around 70, and converges around 10 in the more diverse batches. The balance mean is used as an anchor point, where batches with balance standard deviations greater than it have much less diversity/coverage, and vice-versa. The value 10 likely reflects our own dataset's balance, which itself is not balanced.

6.6 Label-Conditioned Autoencoder's Label Prediction

As detailed in Section 6.3.4, we developed a new method that combines the autoencoder-based design comparison method with the labels' information in order to improve the autoencoder's design comparison performance.

Design Label	Label-Conditioned AE	ResNet50 Predictor
Flexural mode	3.34%	**2.29%**
Q Factor	7.79%	**4.99%**

Table 6.6: Mean relative error comparison between the autoencoder and the ResNet50 predictor model for the flexural mode frequency and Q factor labels.

This method is particularly interesting because it allows us to use the autoencoder's design encoding capabilities to compare designs in a lower-dimensional space, while also including the labels' information in the encoding process thus improving the comparison results. However, since our model has a regressor structure capable of predicting the labels of a resonator design, it could also be used for that purpose, if it yields better results than the ResNet50 Predictor model we used as a baseline in Section 6.2.

To evaluate the autoencoder's label prediction capabilities, we used the test-set designs and encoded them to the embedding space using the autoencoder. Then, we used the regressor component to predict the designs' labels and compute the differences between the predicted and true labels. Table 6.6 shows the mean relative errors obtained for the flexural mode frequency and Q value labels for both the autoencoder and the ResNet50 predictor model.

As observed, the label-conditioned autoencoder model is incapable of achieving better results than the ResNet50 predictor model for both the flexural mode frequency and Q factor labels. However, the error values are not far behind, which could indicate that our predictor model is overkill for the label prediction task and that a simpler model could be used instead to achieve similar results.

Either way, the autoencoder's label prediction capabilities are not on-par with the ResNet50 model, which indicates that for further experiments we will continue to use the ResNet50 predictor for the designs' label prediction task.

7

Baseline generative models evaluation

In this chapter, we will present the results obtained from the training of the baseline generative model. We will evaluate the model's performance based on the metrics described in Chapter 6, which give us an in-depth understanding of the model's capabilities to generate new MEMS resonator designs.

7.1 GAN model evaluation

As our baseline model described in Section 5.1, the GAN model was quite straightforward to train and was able to start generating new MEMS resonators designs after just a few epochs for nearly every hyperparameter configuration we tried.

For GAN models specifically, the expectation is that the generator will learn to generate designs that are indistinguishable from the real designs, while the discriminator will learn to distinguish between real and fake designs, meaning that the generator's loss will decrease overtime and the discriminator's loss will increase. Realistically, both models will tend to enter a state of equilibrium, where both losses are more or less aligned.

Our GAN model was able to achieve this equilibrium after around 75 epochs, as shown in Figure 7.1. However, after this point, the generator's loss started to consistently increase, while the discriminator's loss started to decrease, which indicates that the generator is not 'fooling' the discriminator well enough. Interestingly, both the validity and batch diversity metrics remained stable, which indicates that the generator is still generating valid and diverse designs, even though the discriminator is not being fooled by them. This behavior was present in all hyperparameter configurations we tried.

Even though this situation is not ideal, we have metrics capable of evaluating the generated designs directly, which are far more informative than the standard loss functions used in GAN models. This means that we can still evaluate the model's performance based on these metrics, instead of relying solely on the loss curves.

As observed in Figure 7.1, the validity score of the generated designs was quite high, having reached a maximum of 73% at epoch 197. Even though, this is still far from the 100% validity score we would like to achieve, most models trained were unable to surpass

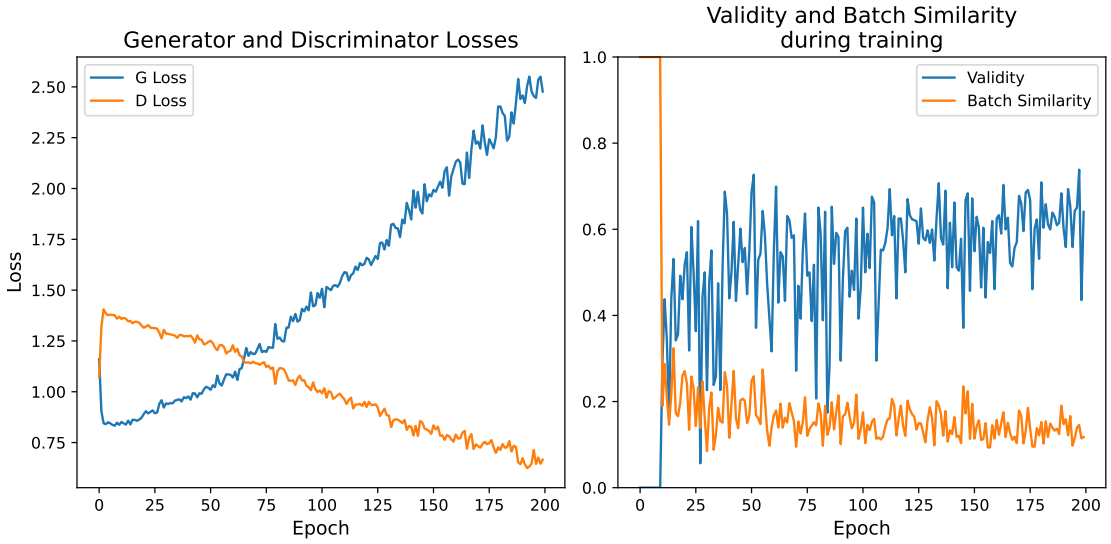

Figure 7.1: Generator and Discriminator training losses and associated validity and batch diversity.

the 60% threshold, meaning that the GAN model was able to generate valid designs with a high degree of success.

The high validity score is paired with a low batch similarity score. As mentioned previously, the batch similarity score is defined as measuring the similarity between the generated designs in a batch, in our case of 512 designs. A low batch similarity score indicates that the generated designs are less similar to each other, which indicates that the model is generating diverse designs.

In terms of clustering diversity, which measures the diversity of the generated designs in terms of the clusters they belong to, the GAN model handled the task quite well, easily achieving a coverage of 100% in most epochs, as shown in Figure 7.2. This means that all the clusters present in the dataset were represented in the generated designs by at least one design.

Additionally, the number of generated designs in each cluster was quite balanced as well. This can be observed in Figure 7.2 and Figure 7.3, where most clusters have a similar number of generated designs. This cannot be directly associated with the lower representativity of some clusters, as some clusters that are over-represented in the dataset (e.g.: cluster 20) are under-represented in the generated designs, while some clusters that are under-represented in the dataset (e.g.: cluster 14) are slightly over-represented in the generated designs. Therefore, the likely cause for this imbalance is due to the structural complexity of some of the designs, which might make them harder to generate. For example, cluster 9, the cluster with the highest number of generated designs which tends to contain designs with lower frequencies, is over-represented in the generated designs. As we have seen before, lower frequency designs tend to have simpler structures with low porosity, which might make them easier to generate from the generator's perspective.

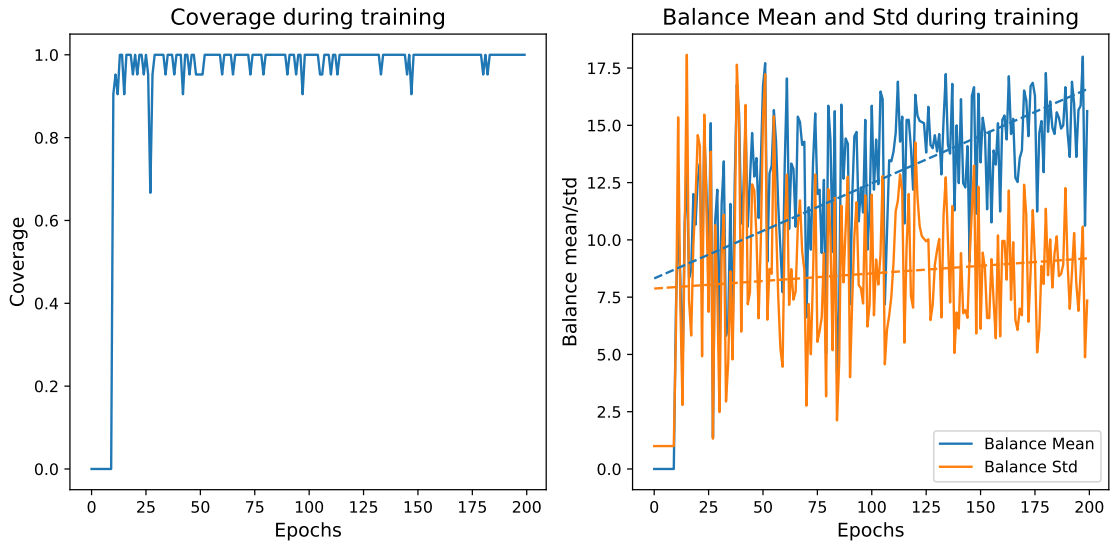

Figure 7.2: Coverage and balance metrics obtained during the generative model training.

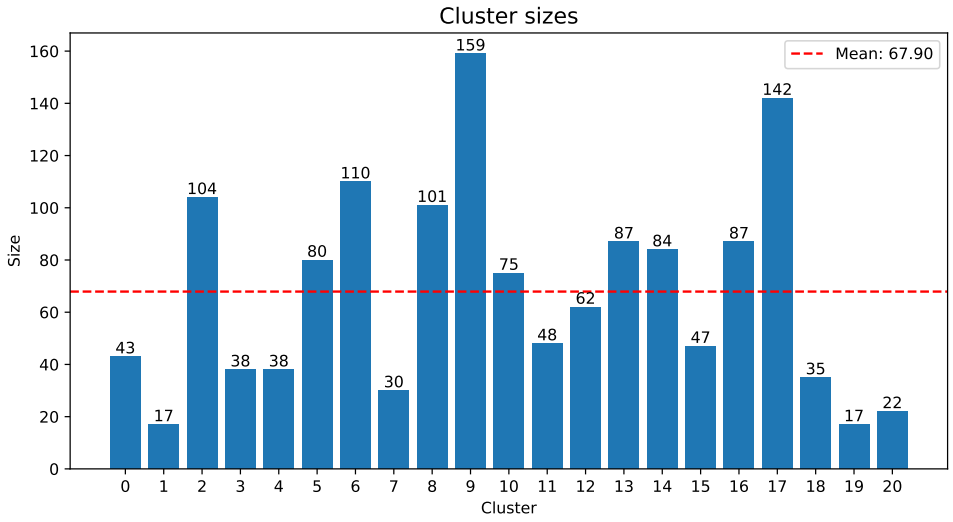

Figure 7.3: Distribution of generated designs across all clusters.

Since many metrics come in play when evaluating the generated designs, choosing the best checkpoint is not a trivial task. To do that, we followed the following criteria:

1. Coverage must be 100%, which ensures that all clusters are represented in the generated designs;

2. The Balance Mean must be higher than the Balance Standard Deviation, which ensures that clusters are somewhat balanced;

3. If both 1 and 2 are met, we choose the checkpoint with the highest validity score;

Metric	Value
Validity	73.8%
Batch Similarity	0.145
Coverage	100%
Balance Mean	18.0
Balance Standard Deviation	10.6

Table 7.1: Best checkpoint metrics for the GAN model. The model was trained following the hyperparameters described in Section 5.1.

4. If neither condition 1 nor condition 2 are met, we choose the checkpoint with the highest validity score;

This checkpoint selection strategy ensures that we select a checkpoint which generates diverse, balanced, and valid designs, which will be incredibly valuable when using it in the optimization stage.

Following this strategy, we selected the checkpoint from epoch 197 as the best checkpoint for the GAN model. Table 7.1 shows the metrics for this checkpoint and Figure 7.4 shows some of the designs generated by the model.

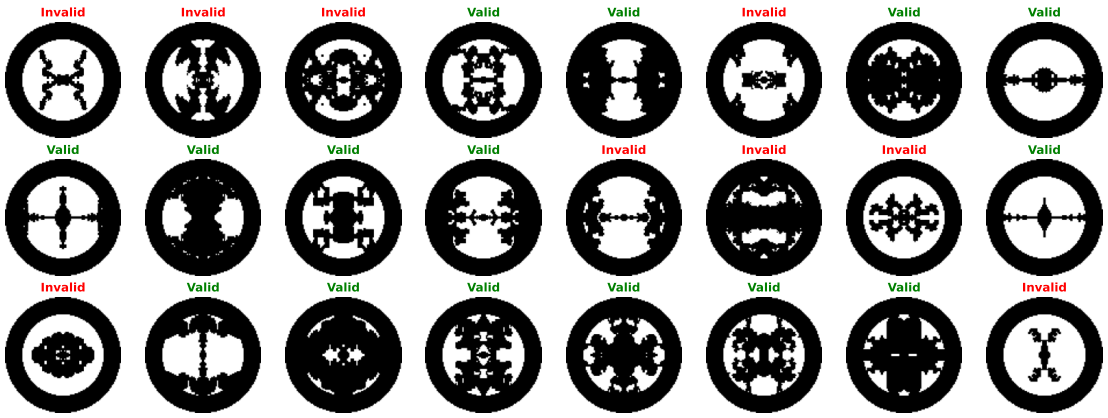

Figure 7.4: Sample of 24 generated designs using our baseline generative model and information on whether they are valid or not.

This concludes the evaluation of the GAN model. In the next chapter, we will be delving into the optimization of MEMS resonators using this generative model as the baseline for the optimization process.

8

Optimization methodology

In this chapter, we present the methodology used to optimize the generation of new MEMS resonators. We start by presenting the novel framework that was developed to automate the optimization process, followed by a few experiments that were conducted to validate its performance.

8.1 Active Learning-based MEMS Optimization Framework

As stated before, the optimization of the generation of MEMS resonators is a complex task that involves the tuning of many parameters inside the generative model, which is impossible to be done manually.

To this effect, we developed a novel optimization framework that automates the optimization process by tuning the data on which the generative model is trained in order to achieve optimized designs. This framework is composed of 4 main components:

1. **A Generative Model** that is used to generate new designs of MEMS resonators. In our case, we will use a pre-trained GAN model that was trained on a dataset of MEMS resonators;

2. **A Predictor**, which is a component that is able to predict the performance of a given design. In our case, we will use the ResNet Predictor described in Section 6.2, which is able to predict both the resonant frequency and the quality factor of a given design;

3. **An Optimization Strategy**, which is responsible for filtering the designs considered suboptimal and selecting the best designs to be used in the next iteration. Different Q factor optimization strategies will be used, which will be discussed in Section 8.2;

4. **An Optimization Buffer**, which is a component that stores the best designs found so far, according to the specified optimization strategy;

A high-level overview of the entire optimization framework setup is shown in Figure 8.1.

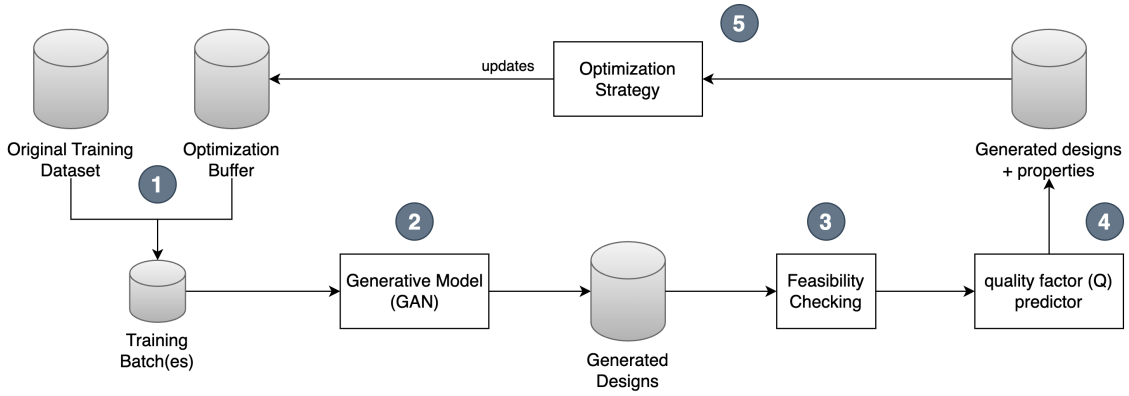

Figure 8.1: Optimization framework pipeline architecture: (1) training mini-batches are constructed from both the original training dataset and the optimization buffer. These will be used to fine-tune a pre-trained generative adversarial network (2) to generate batches of new designs. These new designs will then pass through a feasibility/validity checker (3), which validates the structure and filters out any non-feasible designs. Furthermore, the remaining designs' Q value is computed using a predictor (4) and these are passed to the optimization strategy component (5), which contains the optimization buffer update logic. This process is repeated a configurable number of times.

The process initiates with the generation of a batch of suboptimal designs utilizing the generator, which are subsequently assessed through the Predictor to obtain their resonant frequency and Q value. Following this evaluation, the selected optimization strategy determines which designs will be integrated into the buffer, effectively replacing those with lower Q values. Ultimately, the GAN is trained in an interleaved manner, with designs from the original training dataset and the optimization buffer. This biases the model's generation towards designs that optimize the target criteria, in this case, the Q value.

One interesting aspect of this framework is that it is agnostic in almost all of its components, meaning that it can be used with a different generative model, a different predictor, and even a different optimization strategy, as long as they are compatible with each other. If an operator wants to use a different generative model, for example, they can simply replace the GAN model with the intended model, and the framework will work as expected, provided the right hyperparameters are set.

The available hyperparameters for this optimization framework apart from the ones already present in the GAN model are:

1. **Buffer size**, which controls the number of designs that are stored in the optimization buffer.

2. **Buffer ratio**, which controls the ratio of designs which come from the optimization buffer in the training batches. A higher ratio will have a more impactful effect on the model generation, as it will bias the model more;

3. **Buffer update size**, which controls the number of designs that are replaced in the optimization buffer in each epoch. A higher update size will have a more impactful effect on the diversity of the designs in the buffer, as it will replace more designs in a single epoch;

4. **Iterations**, which controls the number of iterations that will be run in each epoch. Every iteration will have a different batch of designs. The more iterations, the more times the model will be trained with the same optimization buffer;

8.2 Optimization strategies

In this section, we present the optimization strategies that were used to optimize the generation of new MEMS resonators, as well as the experiments that were conducted to validate their performance.

8.2.1 Greedy Q optimization

The first optimization strategy used in this work, which we will refer to as the *Greedy Q optimization*, is a simple concept-validation strategy that aims to optimize the Q value of the generated designs by always selecting the designs with the highest Q value.

Even though this is what we would like to achieve in practice, this strategy won't achieve a full resonant frequency range optimization, as it only optimizes for highest Q values and thus biases towards lower frequency designs. However, this strategy is useful to validate the optimization framework and to understand if the framework is capable of biasing the model generation towards specific target criteria.

For this specific optimization strategy, the optimization buffer is initialized with a batch of designs from the original training dataset, and the optimization process is run for 50 epochs, with a total of 180 iterations per epoch. The buffer size is set to 2000 designs, and the buffer update size is set to 1000. The buffer ratio is set to 0.8, meaning that 80% of the designs in the training batches come from the optimization buffer.

8.2.2 Polynomial-based Q optimization

Given the nature of the previous optimization strategy, which biases the models towards lower frequency designs, we need a more complex optimization strategy that can optimize the Q value across the entire frequency range, offering a more realistic optimization scenario.

The second optimization strategy used in this work, which we will refer to as the *Polynomial-based Q optimization*, aims to optimize the Q value of the generated designs across the entire frequency range by selecting the designs that are closest to a polynomial function that represents the best possible Q values for a given frequency.

As observed in Section 4.1, the frequency and the Q value of the MEMS resonators in the dataset used in this work are highly correlated, meaning that we can create a function that represents the best possible Q values for a given frequency, by having its output somewhere close the upper bound of the Q values in the dataset.

To achieve this, we first sliced the dataset into different frequency ranges and obtained the largest Q values for each, which can be observed in Figure 8.2. The frequency range was divided into 107 slices, each representing a frequency range of around 10 kHz, which is the maximum number of slices that we can have without having a slice with no data points.

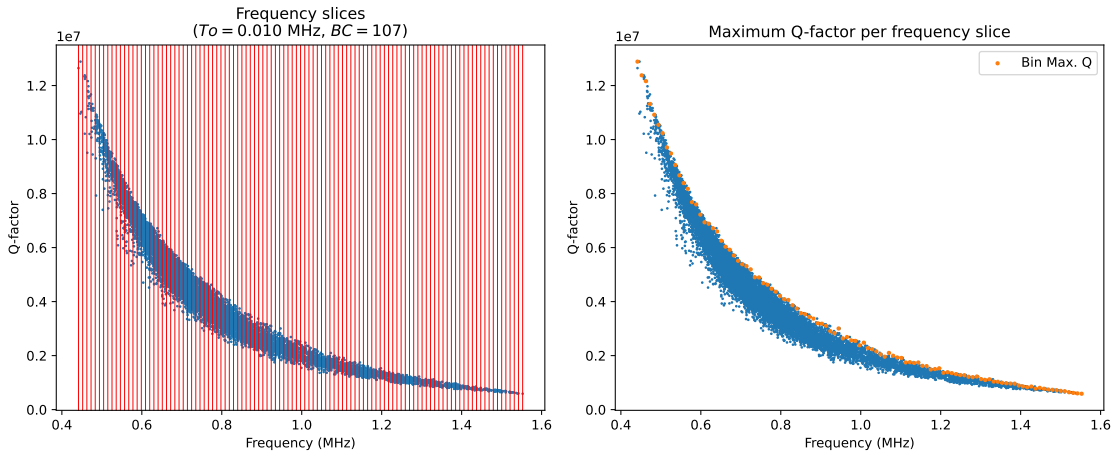

Figure 8.2: Frequency slices (or bins) used to obtain the maximum Q value for each slice. Each vertical red line represents the start of a new slice. Each covers a frequency range tolerance $To \approx 0.01$ MHz, and the number of bins $BC = 107$.

Then, we fitted a 5-th degree polynomial function to the data points selected for each slice, which can be observed in Figure 8.3 and whose coefficients are shown in Table 8.1. This polynomial function represents the approximate best Q values for a given resonant frequency, which can be used as our optimization target.

α_5	α_4	α_3	α_2	α_1	α_0
-3.918×10^7	2.166×10^8	-4.766×10^8	5.289×10^8	-3.053×10^8	7.802×10^7

Table 8.1: Coefficients of the polynomial fitted to the maximum Q value obtained from each bin.

The optimization strategy then selects the designs that are closest to the polynomial function for each frequency slice, by measuring the distance between the predicted Q value of the design and the value presented by the polynomial function.

$$\text{Distance} = Q_{\text{poly}} - Q_{\text{pred}}$$

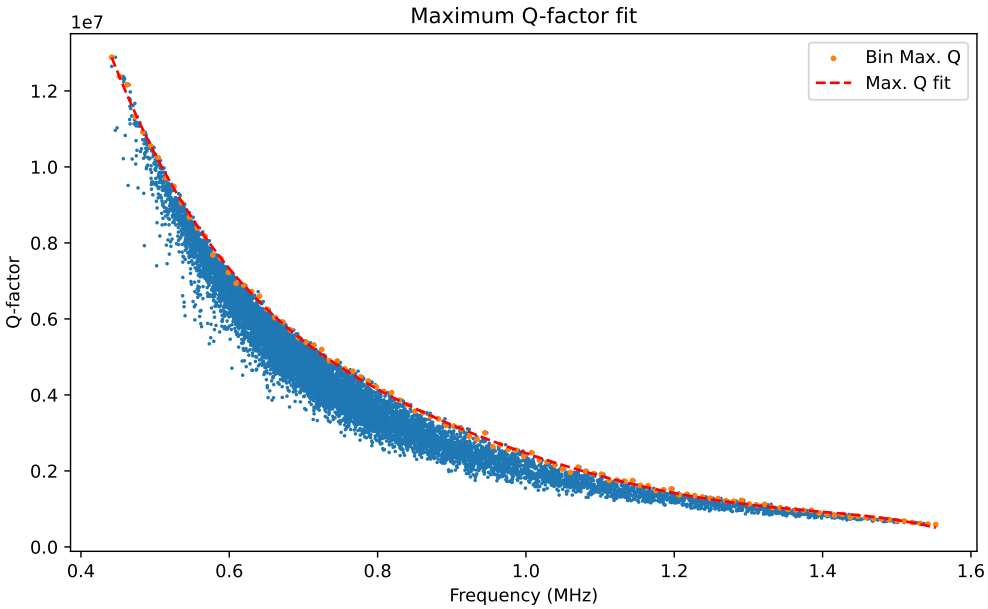

Figure 8.3: Polynomial fit of the maximum Q values registered in each bin.

Where Q_{poly} is the Q value predicted by the polynomial function and Q_{pred} is the Q value predicted by the ResNet Predictor.

Note that distances can be negative, meaning that the design is above the polynomial function. This is a design choice, as the polynomial function represents just an approximation of the best Q values for a given frequency, and the model might be able to generate designs with even higher Q values.

Similar to the previous method, this method is run for 50 epochs, with a total of 180 iterations per epoch. Both the buffer size and update size are set to 1000 designs, and the buffer ratio is set to 0.8. The optimization strategy then ensures that the closest designs to the polynomial function are stored in the optimization buffer, effectively replacing those with lower Q values.

8.2.3 Binned Q optimization

The third and final Q value optimization strategy used in this work is the *Binned Q optimization*, which combines the core ideas of both the *Greedy* Q optimization and Polynomial-based Q optimization strategies to improve the Q value of the generated designs across the entire frequency range.

Rather than optimizing the Q value across the full spectrum at once, as in previous methods, the Binned Q optimization strategy divides the frequency range into discrete slices (or bins) and optimizes the Q value for each bin individually. This approach mitigates the issue of biasing towards lower frequency designs, as seen in the *Greedy* Q optimization strategy, while retaining the ability to optimize Q across the entire spectrum, similar to the Polynomial-based Q strategy.

For this strategy, we use 20 bins as a default setting, each representing a frequency range of approximately 0.1 MHz. This choice balances the number of bins and the resolution of the frequency slices, providing enough granularity to capture frequency-dependent Q variations without overcomplicating the optimization process. Note that the more bins we have, the more precise the optimization process will be. However, this will also require more data points to be generated to guarantee that we have enough data points for each bin, increasing the computational cost of the optimization process.

The buffer initialization, buffer size, buffer update size, buffer ratio, and number of epochs are the same as in the *Polynomial-based Q optimization* strategy.

In the next chapter, we will be delving into the experiments that were conducted to validate the performance of these optimization strategies, and do an in-depth analysis of the results obtained.

9

Optimization results

This chapter presents the results of the three Q value optimization strategies explored in this work: Greedy Q optimization, Polynomial-based Q optimization, and Binned Q optimization. We evaluate the performance of each strategy in terms of the Q value improvement across the entire frequency range and analyze the trade-offs between design diversity and Q value optimization.

9.1 Greedy Q optimization results

The Greedy Q optimization strategy aims to optimize the Q value of the generated designs by selecting the designs with the highest Q values, without considering any frequency information whatsoever. In this method, the optimization strategy chooses the designs with the best Q values from a batch of 1000 designs at the end of each optimization epoch, and updates the buffer with these, replacing the designs with the lowest Q values.

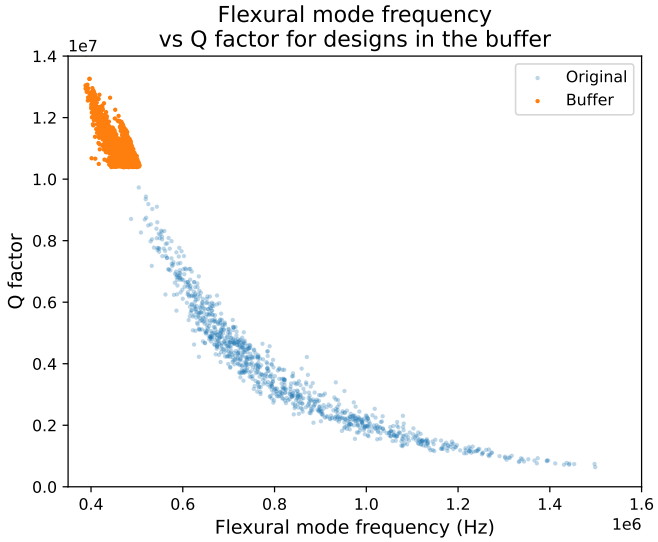

Figure 9.1: Flexural mode frequency and Q value of designs contained in the optimization buffer in the last optimization epoch of our *Greedy* strategy, compared to the ones generated by the non-optimized model.

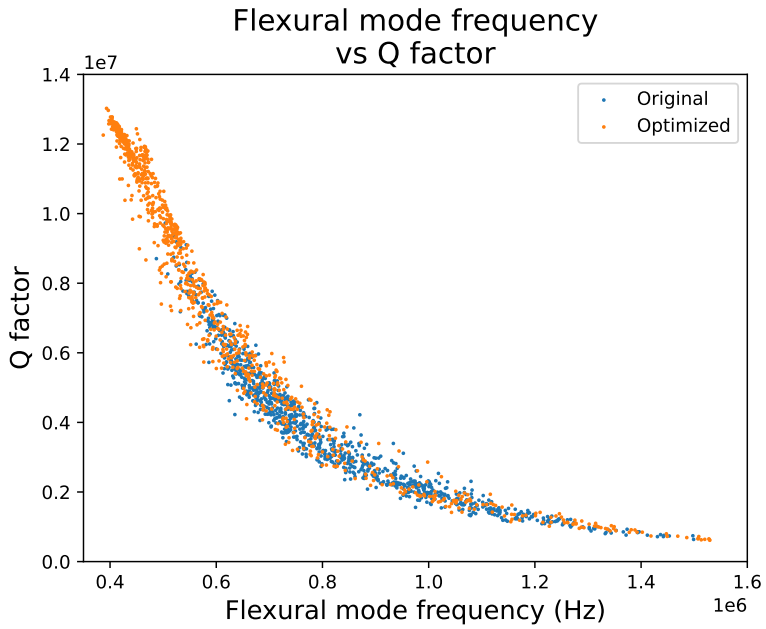

Figure 9.2: Flexural and Q value distribution of generated samples generated by our optimized model comparing to the original non-optimized model.

As observed in Figure 9.1, the designs contained in the optimization buffer at the end of the optimization process are all concentrated around the $4-4.5$ MHz frequency range and between the $1.2-1.3\times10^7$ Q value, meaning that the optimization strategy is working as intended, selecting the designs with the highest Q values.

As for the designs generated by the model, their distribution is expectedly biased towards the lower frequencies, with a significant number of additional designs generated there, as observed in Figure 9.2 and 9.3.

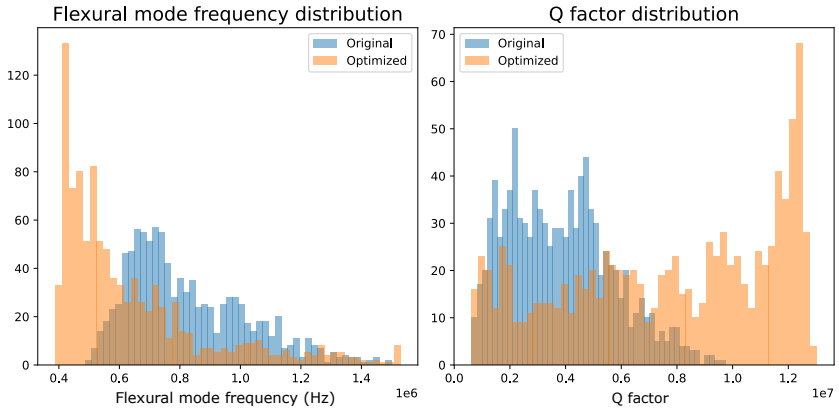

Figure 9.3: Flexural and Q value distribution of generated samples generated by our optimized model comparing to the original non-optimized model.

This observed bias in the generated designs essentially proves that the optimization framework works as expected, and is capable of optimizing the generated designs towards

Metric	Value
Generator loss	2.517
Discriminator loss	0.673
Validity	70.7%
Batch similarity	0.197
Coverage	100%
Balance Mean	17.2
Balance Std	23.7

Table 9.1: Metrics obtained in the last checkpoint of the optimization process using the greedy strategy.

a specific goal, making it a good baseline for further optimization strategies. Additionally, we observe that the optimized model is generating designs in a frequency range that our original non-optimized model was not capable of generating due to low data representativity in that region. This is a good indicator that this framework, with the right strategy, can be used to search for designs in specific frequency ranges, or with certain properties, that are not well represented in the dataset.

In terms of Q value improvement, a significant improvement can be observed in the lower frequencies, as shown in Figure 9.4, where it is also suggested that by just optimizing the Q value, the model is capable of generating designs with a higher Q value across some other regions of the frequency range, with one frequency range showing an improvement of over 10% in higher frequencies.

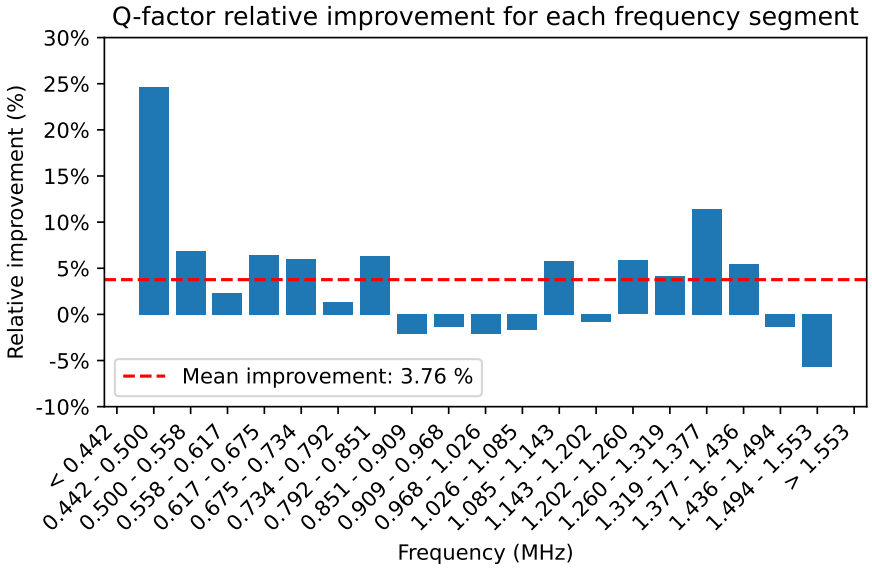

Figure 9.4: Mean Q value improvement for designs generated by the Greedy Q Optimized model, compared to the original non-optimized model.

In Figure 9.5, we can observe samples of random designs generated by our optimized model. There, a bias towards these "cross-shaped" designs, which belong to groups with

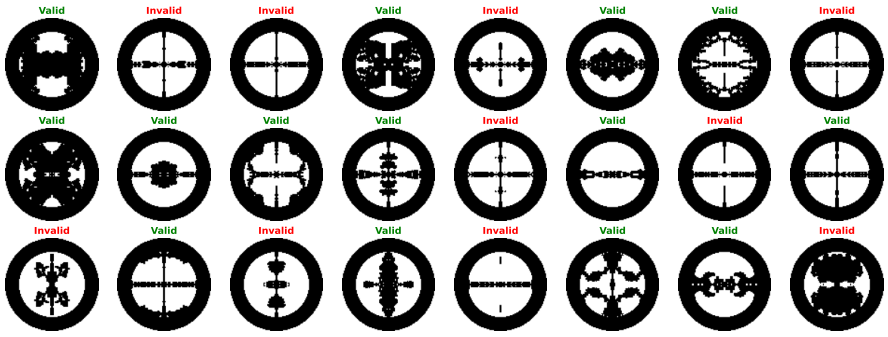

Figure 9.5: Samples of valid and invalid generated designs using our Greedy Q optimized model.

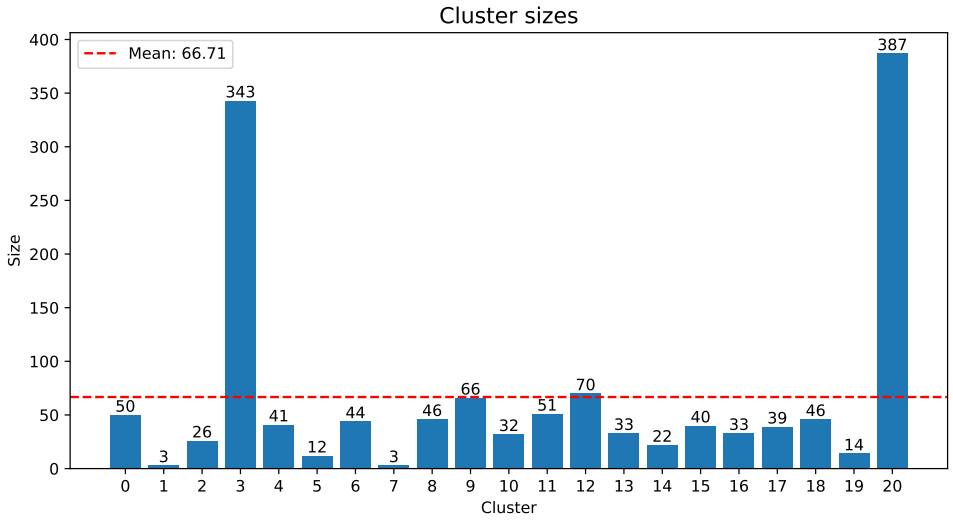

Figure 9.6: Cluster sizes of 1400 generated designs using our Greedy Q optimized model. These designs represent the valid ones from a batch of 2000 designs.

lower frequencies, is evident, providing more evidence that our optimization strategy worked as intended. Additionally, this bias also reflects in the cluster sizes, visible in Figure 9.6, where **cluster 3** and **cluster 20**, which represents the lower frequency designs, have way more designs than other clusters, further confirming the existence and properties of this induced bias.

With this, we can conclude that the *Greedy* Q optimization strategy, although simply a concept-validation strategy, is capable of adjusting the generated designs towards a specific goal, which paves the way towards more complex optimization strategies.

9.2 Polynomial-based Q optimization results

The Polynomial-based Q optimization strategy aims to optimize the Q value of the generated designs by using a polynomial function to predict the approximate maximum Q value for a given frequency and then measuring the distance between the predicted

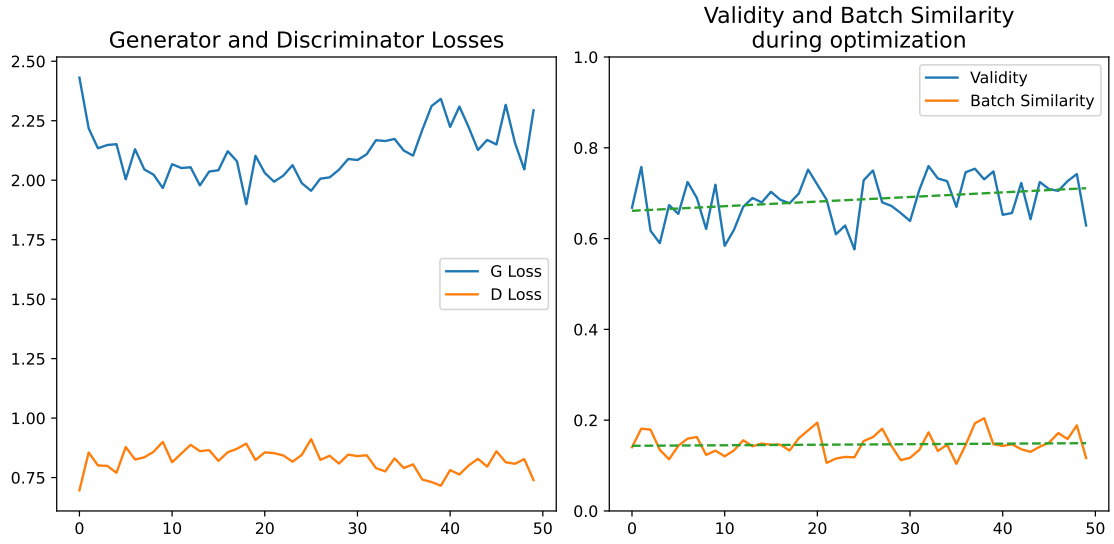

Figure 9.7: Evolution of losses, validity and batch similarity during the optimization process.

Metric	Value
Generator loss	2.293
Discriminator loss	0.739
Validity	62.8%
Batch similarity	0.116
Coverage	100%
Balance Mean	15.3
Balance Std	10.0

Table 9.2: Metrics obtained in the last checkpoint of the optimization process using the polynomial-based strategy.

and actual Q values. Unlike the previous method, this strategy is expected to optimize the Q value across the entire frequency range, as the polynomial function covers the entire frequency domain.

As observed in Figure 9.7, the optimization process evolved with no issues, with the generator and discriminator losses being reasonably stable. Similarly, the validity, which measures the number of valid/feasible designs, and batch similarity, which measures how similar the designs in the batch are to each other, remained stable throughout the whole process, meaning that no design feasibility or diversity penalty was introduced when optimizing them.

In terms of **coverage** and **balance**, which can be observed in Figure 9.8, the model remains quite stable, measuring a coverage of 100% in every single epoch, and always maintaining a **balance mean** greater than the **balance standard deviation**, indicating that the model is generating designs in a balanced way across all design groups. Table 9.2 shows the metrics obtained in the last checkpoint of the optimization process.

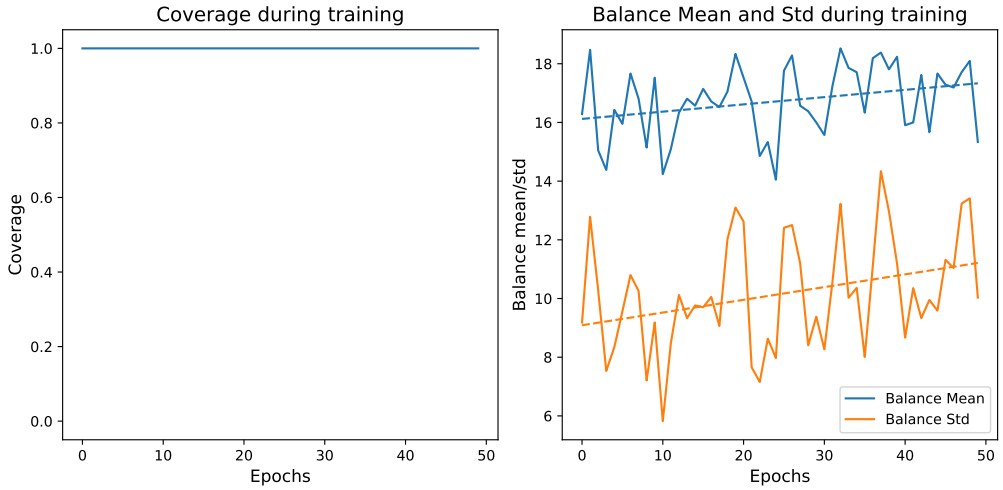

Figure 9.8: Coverage and balance obtained throughout the optimization epochs for the polynomial-based optimization strategy

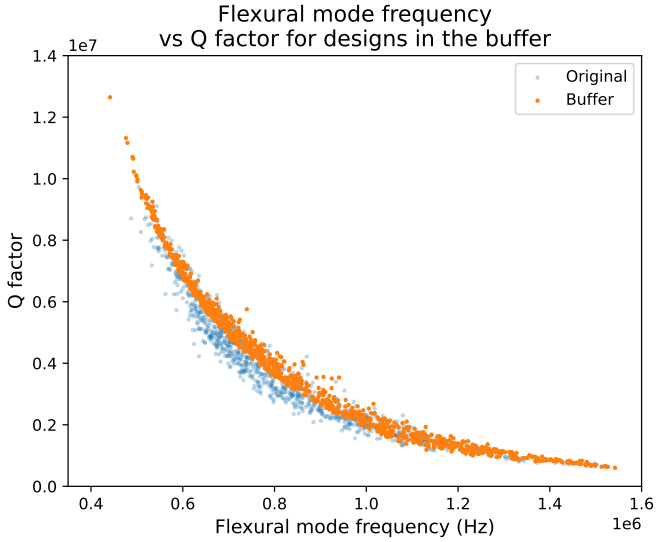

Figure 9.9: Flexural mode frequency and Q value of designs contained in the optimization buffer in the last optimization epoch of our polynomial-based strategy, compared to the ones generated by the non-optimized model.

In Figure 9.9, the distribution of the designs in the optimization buffer is shown, which evidences the designs' closeness to the polynomial line. This is expected and indicates that our optimization strategy is working as expected. However, there appears to be a bias towards the lower frequencies in the designs in the buffer, while higher frequencies are not as well represented. This, in turn, affects the final distribution of the generated designs, where more designs seem to be generated between the 0.6 Hz and 0.8 Hz frequency range, as observed in Figure 9.11.

Increasing the buffer size to 2000 designs, and maintaining the 1000 designs for the update size, appears to not have any effect on both the distribution of the designs in the

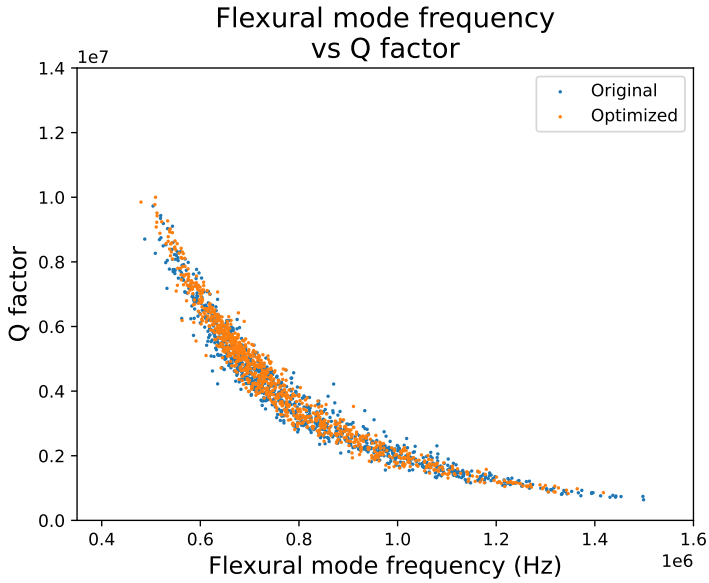

Figure 9.10: Distribution of generated samples generated by our optimized model using the polynomial-based strategy, compared to the original non-optimized model's generated designs.

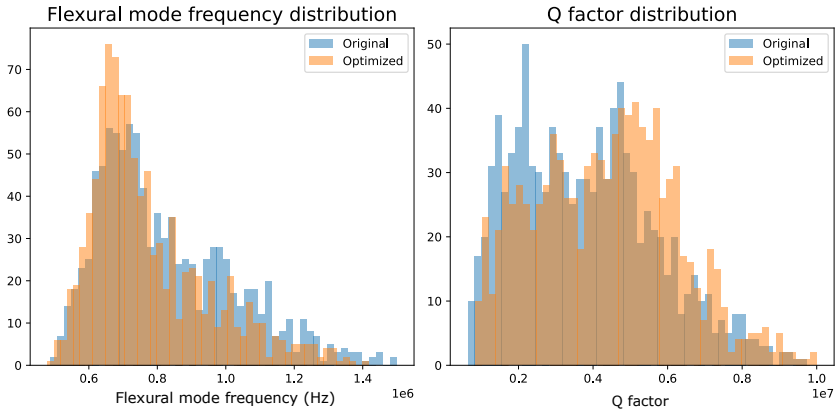

Figure 9.11: Distribution of generated samples generated by our optimized model using the polynomial-based strategy, compared to the original non-optimized model's generated designs.

buffer and the generated designs. Additionally, using a buffer ratio of 0.5 instead of the default 0.8, achieves a similar distribution of designs in the buffer, but the bias in the final generated designs is less pronounced yet still present, which is expected since we are using less optimized designs to generate the final designs. A similar situation happens if the update size is decreased to 512 designs.

This bias towards lower frequencies, observed in multiple runs of the optimization strategy for different hyperparameters, is likely due to two main factors. First, the original model is already biased towards frequencies in the $(0.6, 0.8)$ MHz range, as observed in Figure 9.11, meaning that the optimization strategy is not able to fully correct this bias.

Secondly, the polynomial function used to approximate the best Q values for a given frequency is not perfect, which can introduce larger distances between the predicted and actual Q values for higher frequencies, making the optimization strategy less reliable for these cases.

Either way, this method still showed an improvement in the Q values of generated designs for certain frequency ranges, albeit achieving results similar to the *Greedy* optimization strategy, which indicates that this method is not as effective as expected. These results can be observed in Figure 9.12.

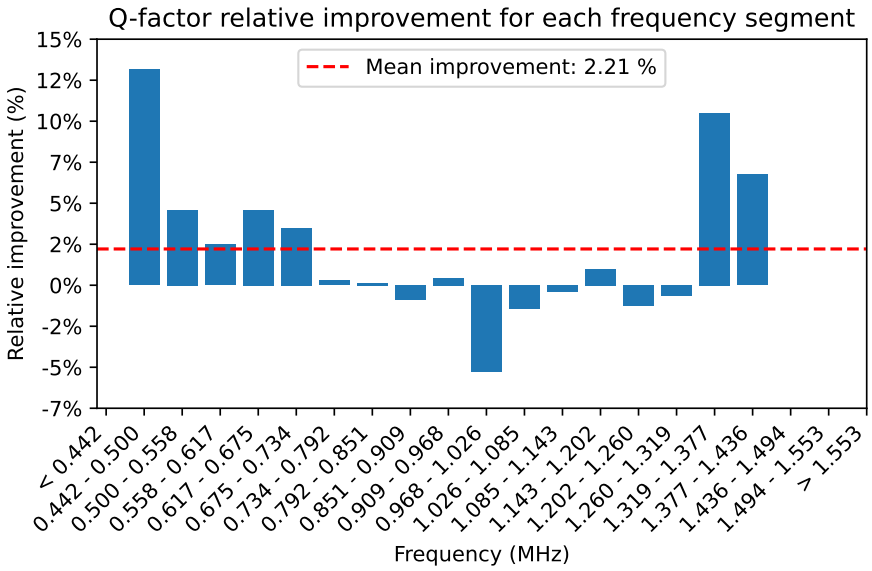

Figure 9.12: Mean Q value improvement for each frequency range for designs generated by our polynomial-based optimized model.

As we can observe, frequency ranges near the extremities showed greater improvements, with a particular emphasis on lower frequencies, where these improvements were larger. However, similar to the *Greedy* optimization strategy, intermediate frequencies observed no improvement at all, with one range even obtaining a worse mean Q value. This means that the polynomial optimization is potentially ineffective at optimizing the Q value in a general way, suggesting that a more capable optimization strategy should be used instead to guarantee a more broad Q optimization.

9.3 Binned Q optimization results

The Binned Q optimization aims to solve the problem of the previous optimization attempt, by considering "frequency slices" that are optimized individually, thus solving the design distribution problem.

As observed in Figure 9.13, we can observe that the optimization process evolved nicely, with no major instability in any of the primary metrics. Same as with the previous optimization strategy, the batch similarity remained stable with the validity increasing

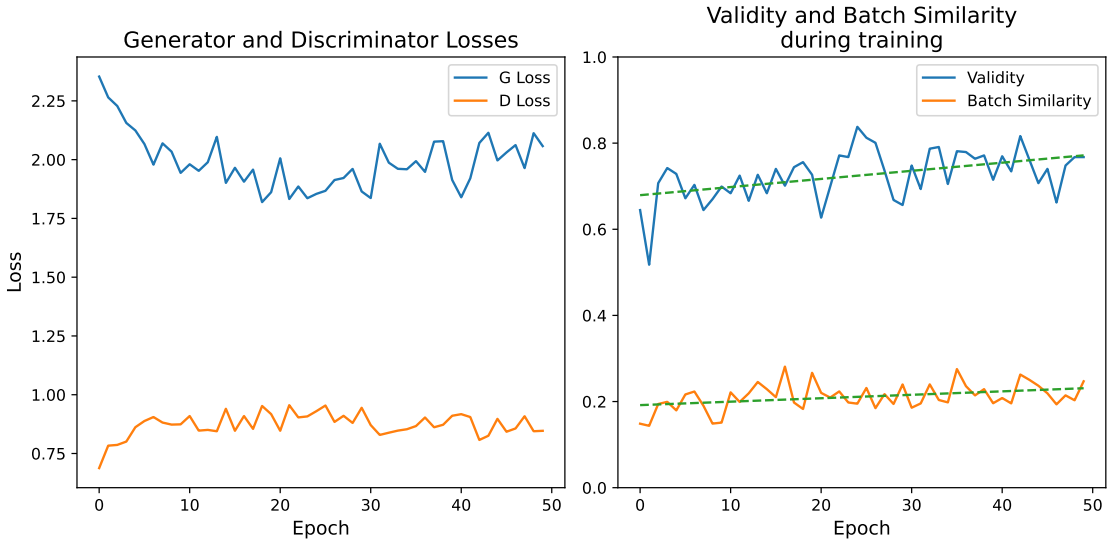

Figure 9.13: Evolution of losses, validity and batch similarity during the binned optimization process.

a little bit overall. This means that our optimization process didn't incur in any penalty towards the generated designs' feasibility or diversity.

In Figure 9.14, the content of the optimization buffer is shown, and we can observe that the Q value "spread", which existed primarily in intermediate frequencies as now collapsed to a single file. This is expected, as the optimization strategy is now optimizing the Q value of designs in a frequency range, and not the entire frequency range, which is a significant improvement over the previous strategy. Table 9.3, shows the metrics obtained in the last checkpoint of the optimization process.

Metric	Value
Generator loss	2.057
Discriminator loss	0.846
Validity	76.7%
Batch similarity	0.24
Coverage	100%
Balance Mean	18.7
Balance Std	20.8

Table 9.3: Metrics obtained in the last checkpoint of the optimization process using the binned approach.

In terms of *coverage* and *balance*, this optimization strategy is quite different from the previous one. As observed in Figure 9.16, although the final *coverage* is 100%, the *balance mean* and *balance std* are starting to converge, occasionally the mean being surpassed by the standard deviation. This is a strong indicator that the balance between design clusters, observed in Figure 9.15 is not as good as in the previous optimization strategy.

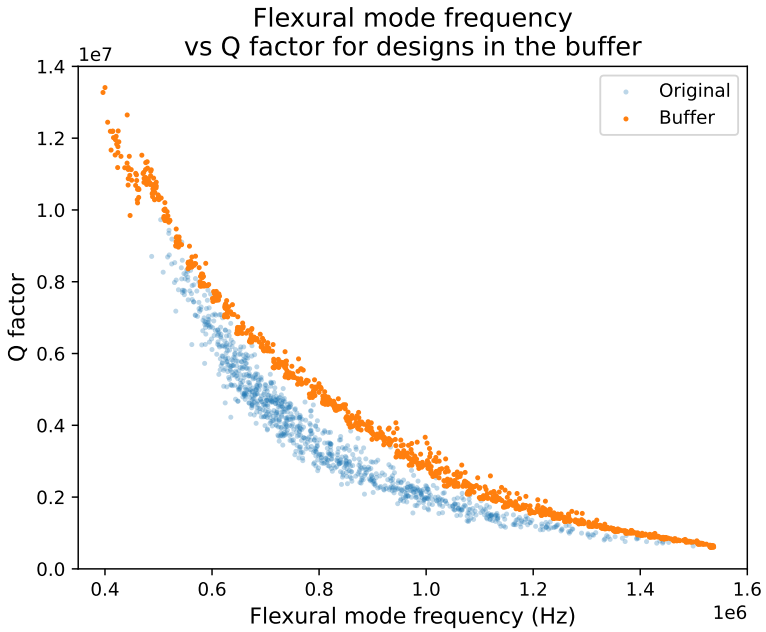

Figure 9.14: Flexural mode frequency and Q value of designs contained in the optimization buffer in the last optimization epoch of our binned strategy, compared to the ones generated by the non-optimized model.

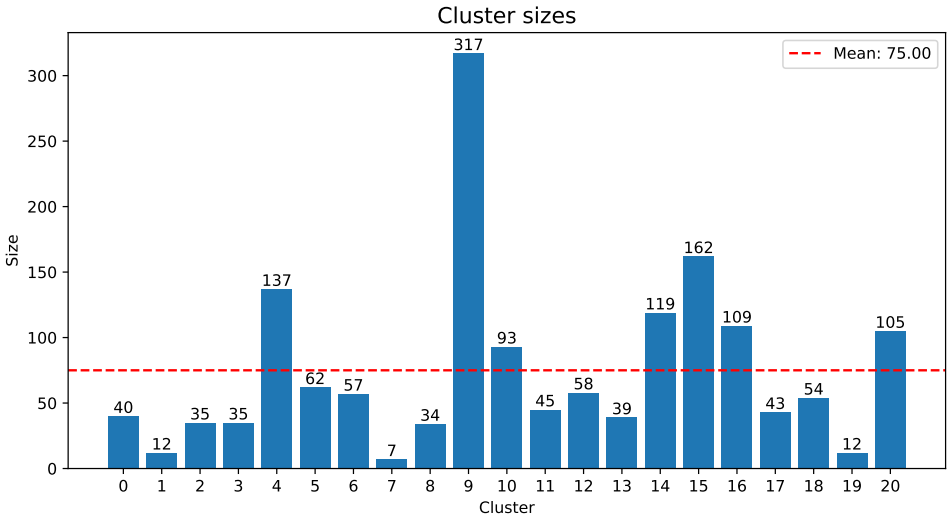

Figure 9.15: Cluster sizes for the final checkpoint of the binned optimization strategy.

This is expected, however, as the more we optimize the designs towards a specific goal, the more likely they are to become less diverse.

As to the generated designs, their distribution is shown in Figures 9.17 and 9.18, where we can observe that, although similar to the original model at first, there are some new improved designs in most frequency ranges. This is a strong indicator that this strategy has been successful at improving designs' Q value, without compromising the original

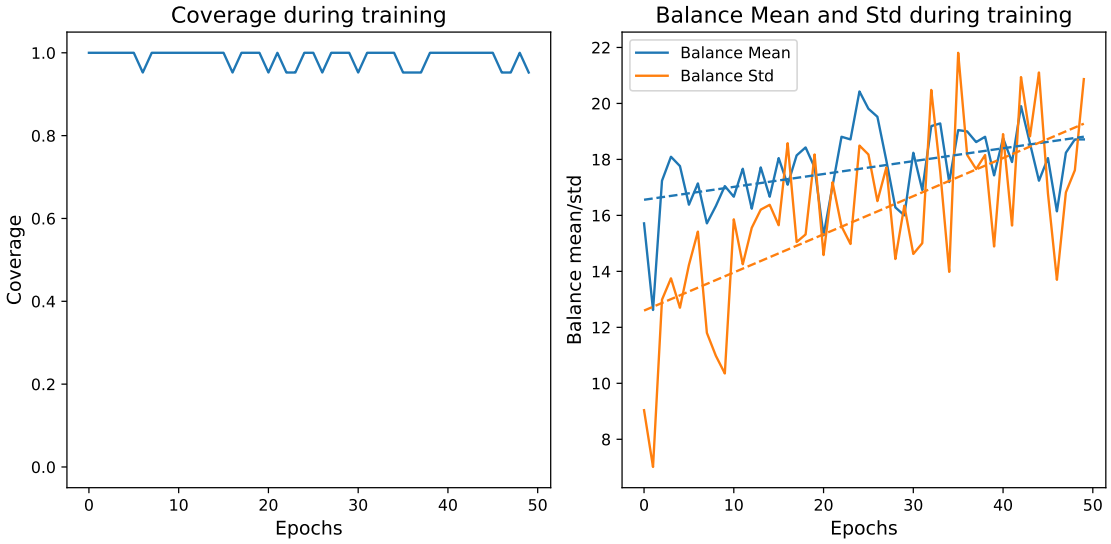

Figure 9.16: Coverage and balance metrics across the binned optimization process epochs.

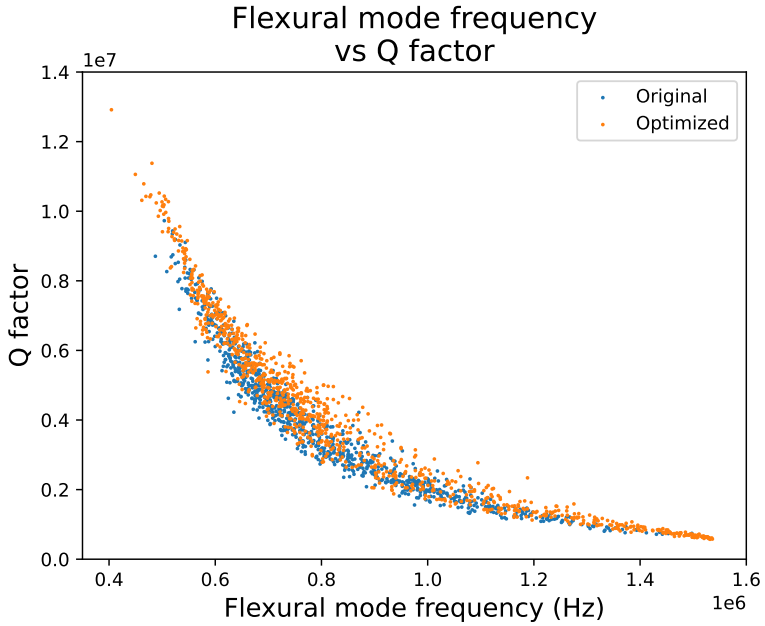

Figure 9.17: Distribution of generated samples generated by our optimized model using the binned strategy, compared to the original non-optimized model's generated designs.

frequency distribution.

From what we can tell, this method shows improved optimization capabilities, which are better than the previous optimization attempt. Figure 9.19 shows the mean Q improvement registered in each frequency range.

This shows that this optimization strategy is capable of achieving > 10% mean Q improvement, with some ranges having peaks of up to 20%. The notable exception being the 1.39 – 1.55 MHz, which registered a decrease of around 5%. This is likely explained

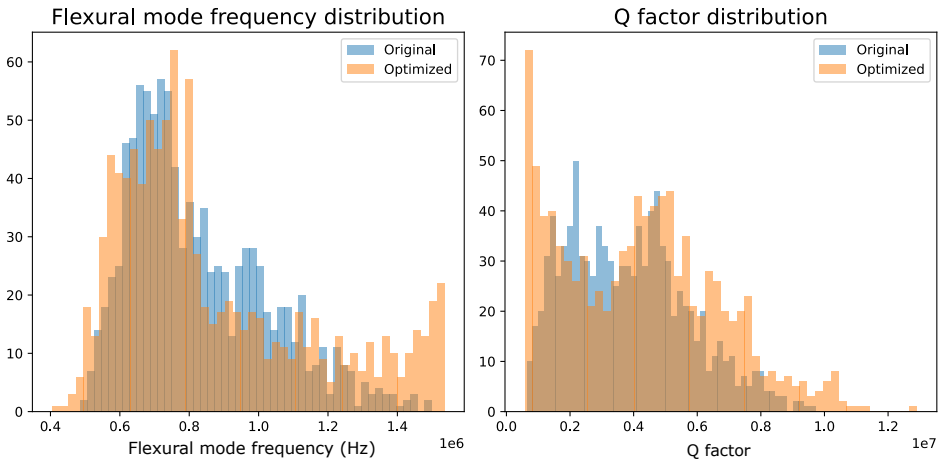

Figure 9.18: Distribution of generated samples generated by our optimized model using the binned strategy, compared to the original non-optimized model's generated designs.

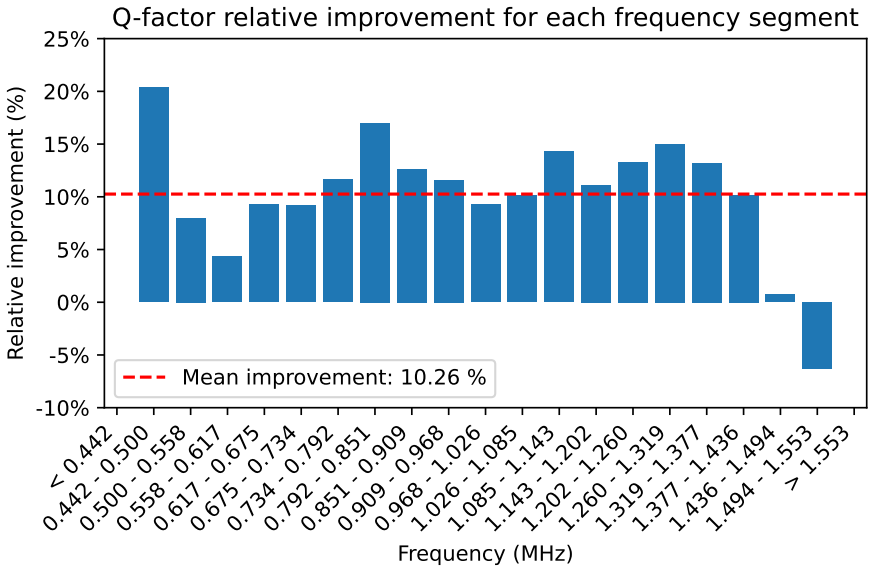

Figure 9.19: Mean Q value improvement for each frequency range of designs generated by our binned optimized model.

by the fact that extreme frequencies (very high or very low) have less representativity in the dataset and generative models are more unstable when generating in these regions.

Anyhow, this method is still considerably better than the previous one, and should be considered as the flagship strategy when optimizing MEMS resonators' generative models. Table 9.4 shows the comparison between all the optimization strategies attempted in this thesis, and their respective metrics.

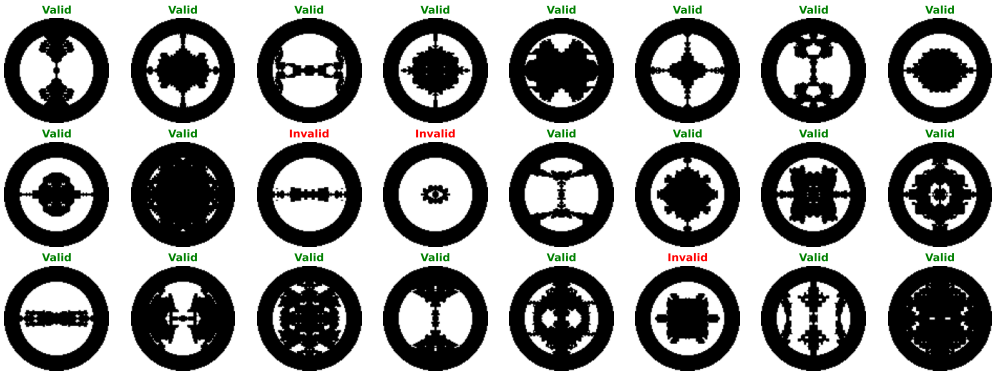

Figure 9.20: Samples of generated designs using our binned optimized model.

Strategy	Validity	Similarity	Coverage	Balance (μ/σ)	Q Improvement
Greedy	70.7%	0.197	100%	17.2 / 23.7	3.76%
Polynomial	62.8%	**0.126**	100%	**15.3 / 10.0**	2.21%
Binned	**76.7%**	0.24	100%	18.7 / 20.8	**10.26%**

Table 9.4: Results aggregation of all the strategies used to optimize the baseline generative model. Several metrics are shown including (from left to right) the validity, the batch similarity (similarity), the coverage, the balance mean (μ) and balance standard deviation (σ), and the mean Q value improvement. In bold, the best results for each metric.

9.4 Binned strategy - COMSOL simulation

To validate the effectiveness of the Binned Q optimization strategy, we simulate designs from both the baseline generative model and the optimized model, then compare their results. This comparison allows us to assess the improvements from the optimization and confirm whether the new designs outperform the original ones.

We assess this in two ways:

- **Simulate k random designs:** We randomly select and simulate k designs from each frequency bin for both the baseline and optimized models. This method provides a broader perspective and ensures that individual designs do not bias the results.
- **Simulate the k best designs:** We simulate the top k designs with the highest Q values from each frequency bin for both models. This targeted approach highlights the most promising designs, which is more relevant for practical applications.

9.4.1 Random designs simulation

Here, we simulate $k = 10$ random designs from each frequency bin, resulting in around 170 designs per model, although bins at extreme frequencies are less represented.

For the baseline model, Figure 9.21 shows the predicted frequency and Q values, revealing a broad distribution across the frequency range, with no apparent bias. Additionally, the predicted values closely match the true values, suggesting that the ResNet Predictor

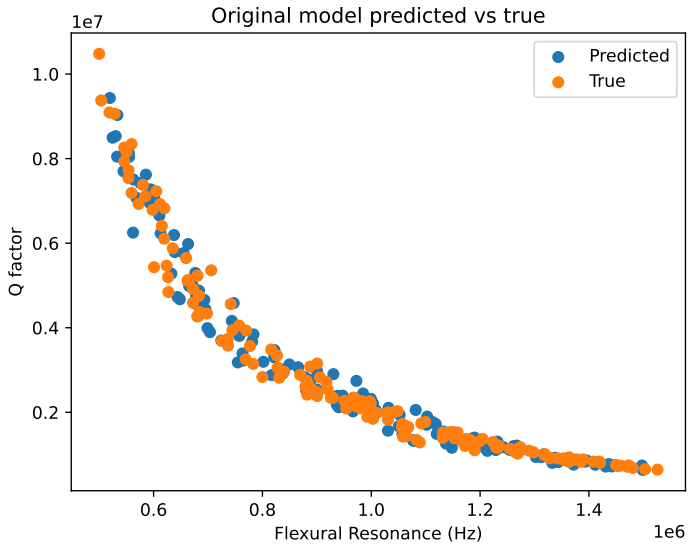

Figure 9.21: Predicted vs. true frequency and Q values for generated designs using the baseline model. Designs chosen at random from each frequency bin.

model is performing well. The mean relative error between predicted and true values is 2.8% for frequency and 7.1% for Q, aligning with the Predictor's overall performance.

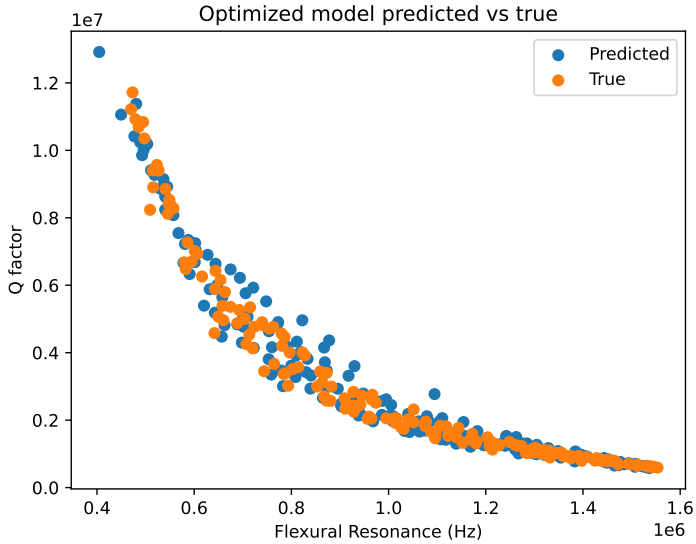

Figure 9.22: Predicted vs. true frequency and Q values for generated designs using the optimized model. Designs chosen at random from each frequency bin.

For the optimized model, the distribution of designs remains similar, as shown in Figure 9.22. However, the mean relative error increases to 5.2% for frequency and 20.1% for Q. This significant increase in Q prediction error suggests that the Predictor is overestimating Q values for the optimized designs. Since the optimization strategy depends on accurate predictions from the Predictor, this limitation becomes more evident

for the optimized designs, which are less represented in the training data.

Despite this, the optimized model generally produces designs with higher Q values across most frequency ranges, as shown in Figure 9.23. The average improvement in Q is approximately 5.66%, with the greatest improvement – nearly 20% – occurring in the 0.6 – 1.0 MHz range.

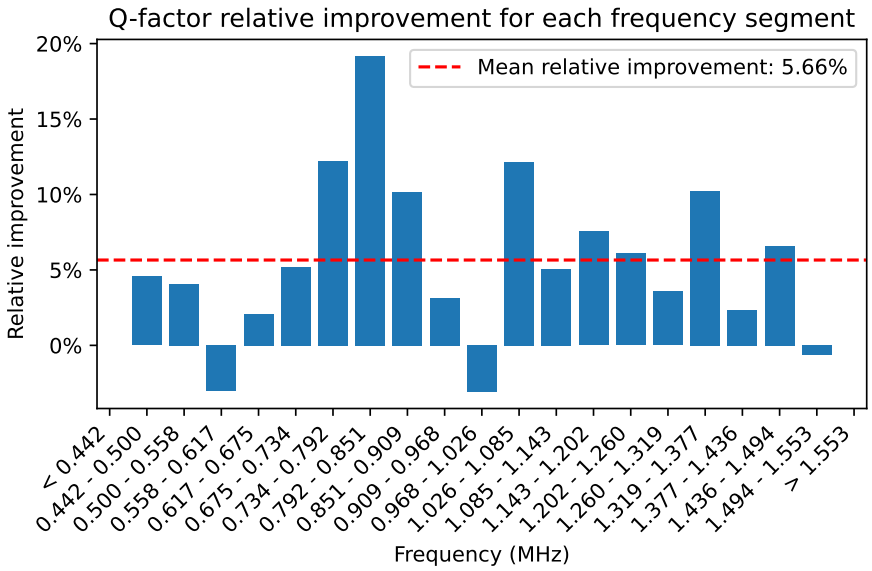

Figure 9.23: Mean Q value improvement between the baseline and optimized designs, when selecting 10 designs randomly from each bin.

However, certain frequency ranges, such as 0.558 – 0.610 MHz, show a slight decrease in Q values. This decrease is likely due to the inherent randomness in the design selection process, which may lead to suboptimal designs being chosen for simulation. The decrease is minimal, less than 1%, and does not significantly affect the overall improvement in Q values.

9.4.2 Best designs simulation

To identify the best designs for each frequency bin, we selected the $k = 10$ designs with the highest Q values from each bin, resulting in roughly 170 designs per model.

Similar to random sampling, the baseline model's designs show a broad distribution across the frequency range and closely match the predicted and true values, as shown in Figure 9.24.

The mean relative error between predicted and true values is 3.8% for frequency and 12.4% for Q, which is slightly higher than for random designs. This increase may be because top-performing designs are more challenging to predict due to their out-of-distribution nature.

For the optimized model, the distribution of the designs differs significantly, as shown in Figure 9.25. Here, the predicted Q values are notably higher than the actual values. The

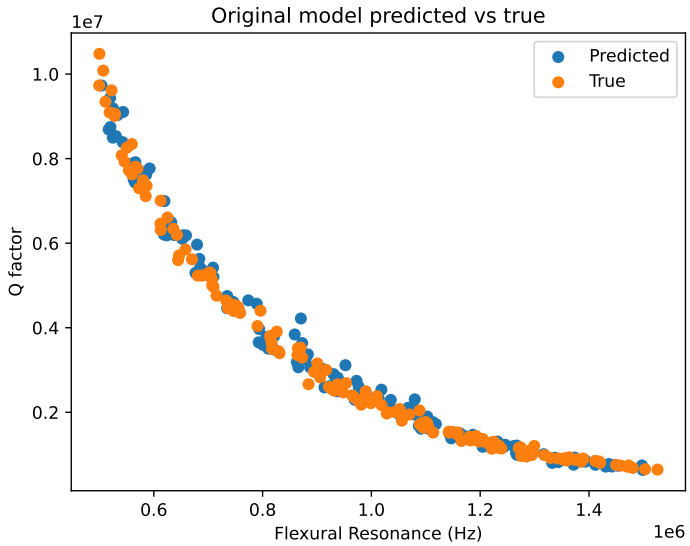

Figure 9.24: Predicted vs. true frequency and Q values for the best Q value designs generated using the baseline model.

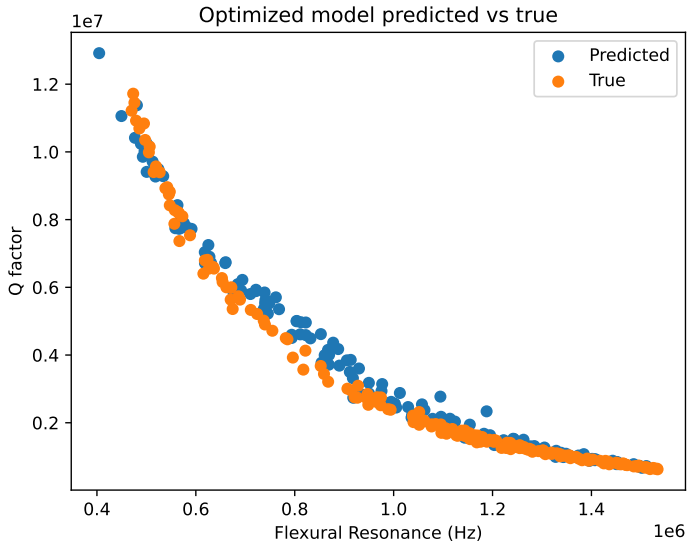

Figure 9.25: Predicted vs. true frequency and Q values for the best Q value designs generated using the optimized model.

mean relative error is 9.4% for frequency and 49.6% for Q, which represents a substantial increase compared to the random designs.

Nevertheless, the optimized model still demonstrates a significant improvement in Q values, as shown in Figure 9.26, with an average improvement of 6.52%. In contrast to the random designs, no frequency range shows a decrease in Q, indicating that the optimization strategy effectively enhances the designs.

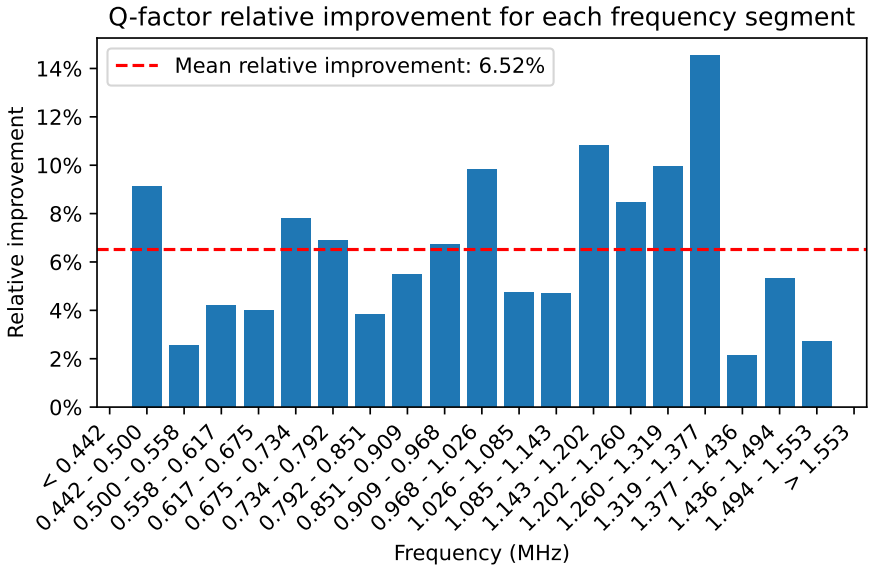

Figure 9.26: Mean Q value improvement between the baseline and optimized designs, when selecting the top 10 designs of each bin.

9.4.2.1 Comparison to dataset designs

One important experiment we should conduct is to check if these optimized designs are better than the ones we currently have in the dataset. This is useful for understanding if new designs have been discovered, or if only the generative model capabilities have been improved.

To do this, we used a similar experiment described in Section 9.4.2, where we binned the frequency range of the dataset in multiple 21 bins and obtained the top $k = 10$ best Q value designs from each bin, and then compared them with the top $p = 10$ designs of each bin of the optimized model.

The results of this experiment can be seen in Figure 9.27, where we can observe that nearly all frequency ranges have a significant decrease in Q value, except for the > 1.553 MHz range, which shows a negligible improvement.

This strongly indicates that the optimization strategy is unable to generate designs better than those in the dataset, which can be explained by two factors:

- The baseline generative model approximates the overall distribution of the dataset. This approximation is insufficient to effectively reach the highest-performing designs;

- The optimization process does not sufficiently explore the design space to reach the highest performing designs;

These limitations prevent our optimized model from achieving the highest-performing designs of our dataset, however, proposed solutions could address these issues:

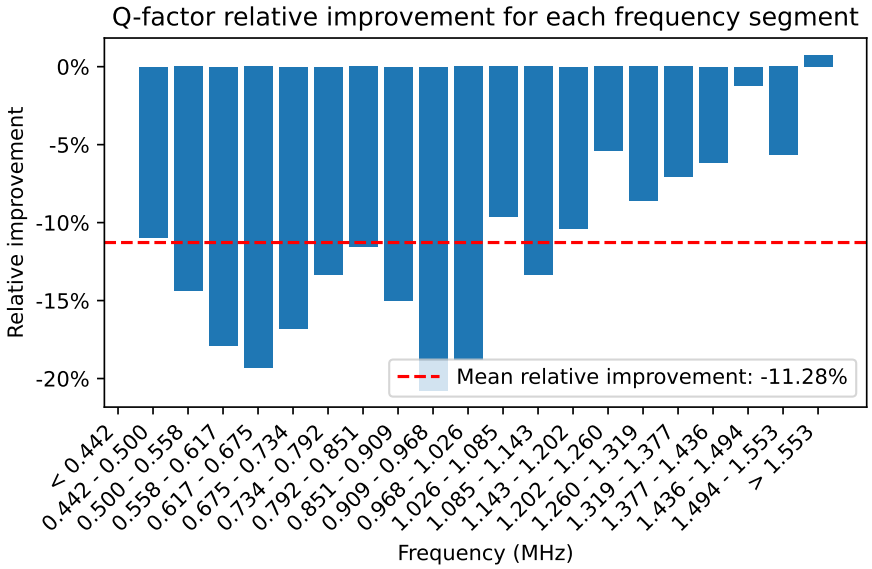

Figure 9.27: Mean Q value improvement of our generated design from our optimized model, when compared to the best designs from our dataset

- **Improve the baseline model's architecture** to broaden its reach within the design space, when generating new designs, which will have a direct impact on the optimization process's ability to explore the design space effectively;
- **Improve the optimization process**, such as using other optimization strategies, more hyperparameter tuning, or exploring new buffer initialization strategies. Improvements in these areas might help the optimization process in exploring the design space;

However, these are merely suggestions, and further research is needed to fully understand the limitations of our optimization process, and to propose effective solutions to these limitations.

10

Conclusions

10.1 Overview

For this thesis, we investigated and studied the problem of generating MEMS resonators using generative models, and how they can be optimized using data augmentation techniques. Our final goal was achieved, which was to **create an optimization framework capable of optimizing the designs' Q value**.

We started by collecting a dataset of MEMS resonators, which was generated and simulated in collaboration with the team at KU Leuven. This dataset was used to train a GAN model, which was capable of generating new designs, and a frequency and Q value prediction model, which was used to evaluate the generated designs.

Then, we designed a **novel set of metrics to evaluate the performance of the generative model and its generated designs**, which helped understand models' behavior and fine-tune hyperparameters. These metrics included the **validity/feasibility**, the **batch similarity**, the **cluster coverage**, and the **cluster balance**, which involved the use of an autoencoder to compress the designs' features into a lower-dimensional space, and a K-Nearest Neighbors (KNN) algorithm to cluster the designs in several design groups.

Next, we implemented an optimization framework, which was used to improve the Q value of the generated designs. This framework used an active learning approach, which was capable of selecting the best designs according to defined criteria. Moreover, the framework is flexible, as it can be used with different optimization strategies, and different generative and prediction models. This prompted us to try three different optimization strategies, which were the **Greedy Q value optimization**, the **Polynomial Q optimization**, and the **Binned Q optimization**, with the last one achieving the best results.

Finally, joining all the pieces together, we were able to optimize the generated designs, and achieve a **mean Q value improvement of** $\approx$ 6.5%, with some frequency ranges achieving an improvement of nearly 15%.

10.2 Impact on the MEMS resonator design field

In the modern world, the demand for smaller, more efficient, and more reliable MEMS devices is increasing. This is especially true for MEMS resonators, which are used in a wide range of applications, such as filters, oscillators, and sensors. Their efficiency, driven by the Q value, is a critical factor to consider when designing and improving these devices.

The work presented in this thesis has the potential to significantly impact the field of MEMS resonator design, by introducing the use of optimization frameworks that can improve the designs' performance, hence obtaining better resonators.

Additionally, given that this process is automated, it can save time and resources for designers, designing new structures in a fraction of a second, and optimizing them in a fraction of the time it would take for a human designer. Also, the data-driven nature of this approach also allows the framework to be adapted for other MEMS devices types, and even other devices altogether, as long as a dataset is available, and a property can be optimized.

Finally, the use of generative models to generate new designs, and the use of data augmentation techniques to optimize them, can lead to the discovery of new designs that might not have been considered before, which can lead to new discoveries and breakthroughs in the field of MEMS resonator design.

10.3 Future Work

Despite the promising results obtained in this thesis, there are several areas that can be improved and further explored in the future, such as:

- **Optimization strategies**: The optimization strategies used in this thesis represent a small subset of the infinite possibilities that can be used to optimize the designs. Future work can explore other optimization strategies, which might yield better results. Optimization strategies that try to optimize other properties other than the Q value can also be explored, such as the design diversity.

- **Buffer initialization strategies**: This thesis used a simple buffer initialization strategy, which randomly selected designs from the original dataset. Future work can explore other buffer initialization strategies, such as the use of the best designs from the original dataset to help guide towards improved designs.

- **Generative models**: The GAN model used in this thesis was a simple model, which was capable of generating designs well enough for the optimization process. However, this model does not use any conditional information, which could be used to generate more specific designs. Future work can explore the use of CGANs, or other generative models, to generate more specific designs.

- **Feedback loop**: The optimization process used in this thesis was a one-shot process, where the designs were generated and evaluated in a single step. Future work can explore the use of a feedback loop, where the designs are generated, evaluated, and then used to retrain key components of the optimization process, such as the generative model and the prediction model, which tend to degrade over time.
- **Explicit optimization**: The optimization process used in this thesis was implicit, as it used an active learning approach to select the best designs. Future work can explore the use of explicit optimization algorithms, such as the GloNet framework described in Section 3.3.3. This framework uses gradient-based optimization to optimize the designs, which is a different approach to this problem.

Bibliography

[1] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/main/template.pdf (cit. on p. i).

[2] S. M. Heinrich and I. Dufour. "Fundamental Theory of Resonant MEMS Devices". en. In: *Resonant MEMS*. John Wiley & Sons, Ltd, 2015, pp. 1–28. ISBN: 978-3-527-67633-0. DOI: 10.1002/9783527676330.ch1. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9783527676330.ch1 (visited on 2024-01-04) (cit. on pp. 1, 10).

[3] S. Finkbeiner. "MEMS for automotive and consumer electronics". In: *2013 Proceedings of the European Solid-State Device Research Conference (ESSDERC)*. ISSN: 2378-6558. 2013-09, pp. 9–14. DOI: 10.1109/ESSDERC.2013.6818809. URL: https://ieeexplore.ieee.org/document/6818809 (visited on 2024-01-04) (cit. on p. 1).

[4] C. Pramanik and H. Saha. "Low Pressure Piezoresistive Sensors for Medical Electronics Applications". In: *Materials and Manufacturing Processes* 21.3 (2006-05). Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/10426910500464446, pp. 233–238. ISSN: 1042-6914. DOI: 10.1080/10426910500464446. URL: https://doi.org/10.1080/10426910500464446 (visited on 2024-01-04) (cit. on p. 1).

[5] P. Hauptmann. "Resonant sensors and applications". en. In: *Sensors and Actuators A: Physical* 26.1-3 (1991-03), pp. 371–377. ISSN: 09244247. DOI: 10.1016/0924-4247(91)87018-X. URL: https://linkinghub.elsevier.com/retrieve/pii/092442479187018X (visited on 2024-01-31) (cit. on p. 1).

[6] E. Vittoz, M. Degrauwe, and S. Bitz. "High-performance crystal oscillator circuits: theory and application". In: *IEEE Journal of Solid-State Circuits* 23.3 (1988-06), pp. 774–783. ISSN: 0018-9200, 1558-173X. DOI: 10.1109/4.318. URL: http://ieeexplore.ieee.org/document/318/ (visited on 2024-01-31) (cit. on p. 1).

[7] B. Razavi. "Architectures and circuits for RF CMOS receivers". In: *Proceedings of the IEEE 1998 Custom Integrated Circuits Conference (Cat. No.98CH36143)* (1998). Conference Name: IEEE 1998 Custom Integrated Circuits Conference ISBN: 9780780342927

Place: Santa Clara, CA, USA Publisher: IEEE, pp. 393–400. DOI: 10.1109/CICC.1998.695005. URL: http://ieeexplore.ieee.org/document/695005/ (visited on 2024-01-31) (cit. on p. 1).

[8] F. Sui et al. "Customizing Mems Designs via Conditional Generative Adversarial Networks". In: *2022 IEEE 35th International Conference on Micro Electro Mechanical Systems Conference (MEMS)* (2022-01). Conference Name: 2022 IEEE 35th International Conference on Micro Electro Mechanical Systems Conference (MEMS) ISBN: 9781665409117 Place: Tokyo, Japan Publisher: IEEE, pp. 450–453. DOI: 10.1109/MEMS51670.2022.9699476. URL: https://ieeexplore.ieee.org/document/9699476/ (visited on 2024-01-11) (cit. on pp. 1, 7, 14, 16, 17, 24, 25, 30).

[9] F. Sui et al. "Trial-and-Error Learning for MEMS Structural Design Enabled by Deep Reinforcement Learning". In: *2023 IEEE 36th International Conference on Micro Electro Mechanical Systems (MEMS)*. ISSN: 2160-1968. 2023-01, pp. 503–506. DOI: 10.1109/MEMS49605.2023.10052277. URL: https://ieeexplore.ieee.org/document/10052277 (visited on 2024-01-04) (cit. on pp. 1, 14).

[10] R. Guo et al. "Deep learning for non-parameterized MEMS structural design". en. In: *Microsystems & Nanoengineering* 8.1 (2022-08). Number: 1 Publisher: Nature Publishing Group, pp. 1–10. ISSN: 2055-7434. DOI: 10.1038/s41378-022-00432-9. URL: https://www.nature.com/articles/s41378-022-00432-9 (visited on 2024-01-04) (cit. on pp. 7–12, 17, 29, 36, 37).

[11] Comsol, Inc. *COMSOL: Multiphysics Software for Optimizing Designs*. en. URL: https://www.comsol.com/ (visited on 2024-01-04) (cit. on p. 11).

[12] J. R. R. A. Martins and A. Ning. *Engineering Design Optimization*. en. 1st ed. Cambridge University Press, 2021-11. ISBN: 978-1-108-98064-7 978-1-108-83341-7. DOI: 10.1017/9781108980647. URL: https://www.cambridge.org/core/product/identifier/9781108980647/type/book (visited on 2024-02-09) (cit. on p. 12).

[13] J. Zhang. *Gradient Descent based Optimization Algorithms for Deep Learning Models Training*. arXiv:1903.03614 [cs, stat]. 2019-03. DOI: 10.48550/arXiv.1903.03614. URL: http://arxiv.org/abs/1903.03614 (visited on 2024-01-31) (cit. on pp. 12, 18).

[14] Rishi. *Understanding of Gradient Descent: Intuition and Implementation*. en. 2023-05. URL: https://blog.gopenai.com/understanding-of-gradient-descent-intuition-and-implementation-b1f98b3645ea (visited on 2024-01-29) (cit. on p. 12).

[15] J. Jiang. "DEEP LEARNING FOR INVERSE DESIGN OF PHOTONIC DEVICES". PhD thesis. Stanford University, 2022-08. URL: https://purl.stanford.edu/qv500br5170 (cit. on pp. 13, 18, 21).

[16] A. Hassanat et al. "Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach". en. In: *Information* 10.12 (2019-12), p. 390. ISSN: 2078-2489. DOI: 10.3390/info10120390. URL: https://www.mdpi.com/2078-2489/10/12/390 (visited on 2024-02-09) (cit. on pp. 13, 19).

[17] P. Ladosz et al. "Exploration in Deep Reinforcement Learning: A Survey". en. In: *Information Fusion* 85 (2022-09). arXiv:2205.00824 [cs], pp. 1–22. ISSN: 15662535. DOI: 10.1016/j.inffus.2022.03.003. URL: http://arxiv.org/abs/2205.00824 (visited on 2024-02-09) (cit. on p. 13).

[18] A. G. Gad. "Particle Swarm Optimization Algorithm and Its Applications: A Systematic Review". en. In: *Archives of Computational Methods in Engineering* 29.5 (2022-08), pp. 2531–2561. ISSN: 1134-3060, 1886-1784. DOI: 10.1007/s11831-021-09694-4. URL: https://link.springer.com/10.1007/s11831-021-09694-4 (visited on 2024-02-09) (cit. on p. 13).

[19] I. Goodfellow et al. "Generative adversarial networks". In: *Communications of the ACM* 63.11 (2020-10), pp. 139–144. ISSN: 0001-0782. DOI: 10.1145/3422622. URL: https://dl.acm.org/doi/10.1145/3422622 (visited on 2024-01-04) (cit. on pp. 14, 16).

[20] N. Torres-Reyes and S. Latifi. "Audio Enhancement and Synthesis using Generative Adversarial Networks: A Survey". en. In: *International Journal of Computer Applications* 182.35 (2019-01), pp. 27–31. ISSN: 09758887. DOI: 10.5120/ijca2019918334. URL: http://www.ijcaonline.org/archives/volume182/number35/torresreyes-2019-ijca-918334.pdf (visited on 2024-02-09) (cit. on p. 14).

[21] X. Wu, K. Xu, and P. Hall. "A survey of image synthesis and editing with generative adversarial networks". In: *Tsinghua Science and Technology* 22.6 (2017-12). Conference Name: Tsinghua Science and Technology, pp. 660–674. ISSN: 1007-0214. DOI: 10.23919/TST.2017.8195348. URL: https://ieeexplore.ieee.org/document/8195348 (visited on 2024-02-09) (cit. on p. 14).

[22] R. Zhou, C. Jiang, and Q. Xu. "A survey on generative adversarial network-based text-to-image synthesis". In: *Neurocomputing* 451 (2021-09), pp. 316–336. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2021.04.069. URL: https://www.sciencedirect.com/science/article/pii/S0925231221006111 (visited on 2024-02-09) (cit. on p. 14).

[23] L. Cai et al. "Utilizing Amari-Alpha Divergence to Stabilize the Training of Generative Adversarial Networks". en. In: *Entropy* 22.4 (2020-04), p. 410. ISSN: 1099-4300. DOI: 10.3390/e22040410. URL: https://www.mdpi.com/1099-4300/22/4/410 (visited on 2024-01-11) (cit. on p. 15).

[24] J. Brownlee. *18 Impressive Applications of Generative Adversarial Networks (GANs)*. en-US. 2019-06. URL: https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/ (visited on 2024-01-31) (cit. on p. 15).

[25] J. Ho, A. Jain, and P. Abbeel. "Denoising Diffusion Probabilistic Models". In: *ArXiv* (2020-06). URL: https://www.semanticscholar.org/paper/Denoising-Diffusion-Probabilistic-Models-Ho-Jain/5c126ae3421f05768d8edd97ecd44b1364e2c99a (visited on 2024-01-04) (cit. on p. 16).

[26] J. Jiang and J. A. Fan. "Simulator-based training of generative neural networks for the inverse design of metasurfaces". en. In: *Nanophotonics* 9.5 (2020-05), pp. 1059–1069. ISSN: 2192-8614, 2192-8606. DOI: 10.1515/nanoph-2019-0330. URL: https://www.degruyter.com/document/doi/10.1515/nanoph-2019-0330/html (visited on 2024-01-31) (cit. on pp. 18, 23).

[27] C. Wang et al. "Design of freeform geometries in a MEMS accelerometer with a mechanical motion preamplifier based on a genetic algorithm". en. In: *Microsystems & Nanoengineering* 6.1 (2020-11). Number: 1 Publisher: Nature Publishing Group, pp. 1–15. ISSN: 2055-7434. DOI: 10.1038/s41378-020-00214-1. URL: https://www.nature.com/articles/s41378-020-00214-1 (visited on 2024-01-04) (cit. on p. 18).

[28] E. Tyflopoulos and M. Steinert. "Topology and Parametric Optimization-Based Design Processes for Lightweight Structures". en. In: *Applied Sciences* 10.13 (2020-06), p. 4496. ISSN: 2076-3417. DOI: 10.3390/app10134496. URL: https://www.mdpi.com/2076-3417/10/13/4496 (visited on 2024-02-09) (cit. on p. 18).

[29] J.-J. Zhu and J. Bento. "Generative Adversarial Active Learning". In: *ArXiv* (2017-02). URL: https://www.semanticscholar.org/paper/e9ff047489490e505d44e573c4240b4dd8137f33 (visited on 2024-01-04) (cit. on pp. 19, 20).

[30] C. Mayer and R. Timofte. "Adversarial Sampling for Active Learning". In: *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*. ISSN: 2642-9381. 2020-03, pp. 3060–3068. DOI: 10.1109/WACV45572.2020.9093556. URL: https://ieeexplore.ieee.org/document/9093556 (visited on 2024-01-04) (cit. on p. 19).

[31] D. Wu, C.-T. Lin, and J. Huang. *Active Learning for Regression Using Greedy Sampling*. arXiv:1808.04245 [cs, stat]. 2018-08. DOI: 10.48550/arXiv.1808.04245. URL: http://arxiv.org/abs/1808.04245 (visited on 2024-01-15) (cit. on p. 19).

[32] B. Settles. "Active Learning Literature Survey". In: 2009. URL: https://www.semanticscholar.org/paper/Active-Learning-Literature-Survey-Settles/818826f356444f3daa3447755bf63f171f39ec47 (visited on 2024-01-04) (cit. on p. 19).

[33] A. Tharwat and W. Schenck. "A Survey on Active Learning: State-of-the-Art, Practical Challenges and Research Directions". en. In: *Mathematics* 11.4 (2023-02), p. 820. ISSN: 2227-7390. DOI: 10.3390/math11040820. URL: https://www.mdpi.com/2227-7390/11/4/820 (visited on 2024-02-09) (cit. on p. 19).

[34] R. Xin et al. "Active-Learning-Based Generative Design for the Discovery of Wide-Band-Gap Materials". en. In: *The Journal of Physical Chemistry C* 125.29 (2021-07), pp. 16118–16128. ISSN: 1932-7447, 1932-7455. DOI: 10.1021/acs.jpcc.1c02438. URL: https://pubs.acs.org/doi/10.1021/acs.jpcc.1c02438 (visited on 2024-01-18) (cit. on p. 21).

[35] Y. Dan et al. "Generative adversarial networks (GAN) based efficient sampling of chemical composition space for inverse design of inorganic materials". en. In: *npj Computational Materials* 6.1 (2020-06). Number: 1 Publisher: Nature Publishing Group, pp. 1–7. ISSN: 2057-3960. DOI: 10.1038/s41524-020-00352-0. URL: https://www.nature.com/articles/s41524-020-00352-0 (visited on 2024-01-29) (cit. on p. 21).

[36] R. Agnihotri. *Problems with Gradient Descent*. en. URL: https://www.encora.com/insights/problems-with-gradient-descent (visited on 2024-01-31) (cit. on p. 21).

[37] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980 (cit. on p. 30).

[38] D. Pfau and O. Vinyals. *Connecting Generative Adversarial Networks and Actor-Critic Methods*. 2017. arXiv: 1610.01945 [cs.LG]. URL: https://arxiv.org/abs/1610.01945 (cit. on p. 46).

I

Annex

This annex reveals the abstract submitted to the conference IEEE MEMS 2024.

8.05 Machine Learning (ML) & Artificial Intelligence (AI)-Enhanced MEMS/NEMS Design, Manufacturing, and Applications 0796

GENERATIVE AND GENERALIZABLE OPTIMIZATION FRAMEWORK FOR MEMS DEVICES THROUGH ACTIVE LEARNING

Novelty / Progress Claim(s)

This paper presents a novel and highly flexible active learning framework for the optimization of micro-electro-mechanical systems (MEMS) devices. At its core, it leverages a deep convolutional generative adversarial network (GAN), regularized by a MEMS performance predictor, to selectively and iteratively shift the model design geometry learned distribution towards optimizing a target criterion. Given its principled design, the framework trivially generalizes to distinct optimization criteria. To illustrate the efficacy of the framework, this study introduces a MEMS resonator featuring an unparameterized arbitrary suspension, where we explicitly optimize the quality factor (Q) while ensuring fabrication feasibility. We showcase a diverse array of fabrication-feasible MEMS resonators exhibiting improved Q-factors, achieving up to 10% Q relative improvement. To the best of our knowledge, this is the first successful implementation of a MEMS devices' design generative machine learning optimization process.

Background / State of the Art

Recently, machine learning algorithms have emerged as a novel methodology in the design of MEMS devices. For instance, Siu *et al.* conducted a study on the conditional generation of MEMS devices utilizing deep convolutional generative adversarial networks (GANs) [1]. Despite promising, it often produces designs that exhibit suboptimal Q values. That suggests its limitation in generating designs higher-performing designs than those in the training dataset, due to the lack of explicit optimization. Active learning optimization techniques [2,3], have been applied across various scientific domains and show considerable potential. This is further evidenced by a recent study on the design of freeform diffractive meta gratings based on generative adversarial networks [4]. Nevertheless, these optimization strategies have yet to be implemented in the design of MEMS devices, thereby presenting a significant opportunity to advance device performance to a new level.

Description of the New Method or System

Our proposed methodology is structured around a four-part pipeline that includes (Figure 1): (1) a generator pre-trained on a designated training dataset, (2) a Q-value evaluation process, (3) a buffer designed to temporarily store generated designs along with their evaluated properties, and (4) a buffer update strategy that iteratively incorporates new and improved designs based on a predefined criteria. The process initiates with the generation of a batch of sub-optimal designs utilizing the generator, which are subsequently assessed through the Q value evaluation process. Following this evaluation, the selected update strategy determines which designs will be integrated into the buffer, effectively replacing those with lower Q values. Ultimately, the GAN is trained in an interleaved manner, with designs from the original training dataset the optimization buffer. This biases the model's generation toward designs that optimize the target criteria. The principled active learning mechanism can be tuned to different target criteria of the operator, by simply defining a corresponding buffer update strategy.

Experimental Results

Our proposed methodology has been successful at improving the quality factor (Q) of generated designs by an average of 10% as observed in Figure 4. Our process involved optimizing the generative model for 50 epochs with 180 steps each and with 80% of designs in batches coming from the optimization buffer. Some flexural frequency ranges have achieved higher improvements, up to around 20%. Higher frequencies, on the other hand, noticed little improvement, sometimes even decreasing in Q value by a small margin. Additionally, the framework could optimize the designs with no penalty on feasibility or design diversity, as observed in Figure 3.

Word count: 536

References

[1] Sui, Fanping, et al. 2022 IEEE 35th MEMS, pp. 450-453.
[2] Rui Xin, et al. The Journal of Physical Chemistry C 2021 125 (29),
[3] Bailey, Michael, et al. bioRxiv (2023): 2023-07.
[4] Wen, Fufang, et al. Acs Photonics 7.8 (2020): 2098-2104.

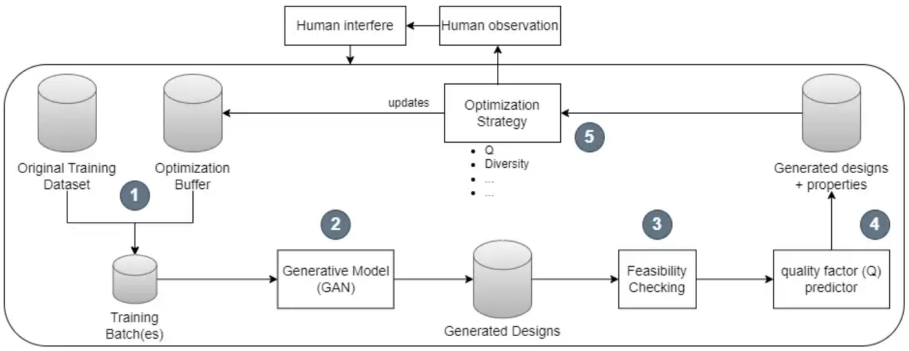

Figure 1 - Optimization framework pipeline architecture: (1) training mini-batches are constructed from both the original training dataset and the optimization buffer. These will be used to fine-tune a pre-trained generative adversarial network (2) to generate batches of new designs. These new designs will then pass through a feasibility checker (3), which validates the structure and filters out any non-feasible designs. Furthermore, the remaining designs' quality factor (Q) is computed using a predictor (4) and these are passed to the optimization strategy component (5), which contains the optimization buffer update logic. This process is repeated a configurable number of times.

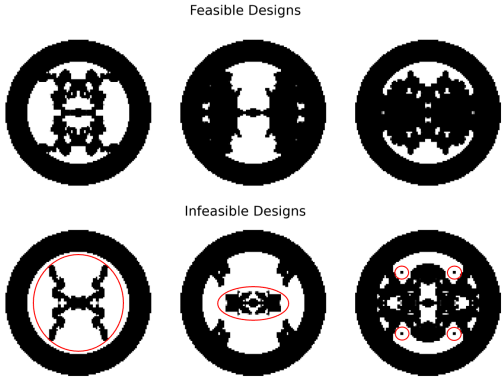

Figure 2 - Examples of feasible and infeasible designs, with reasons for infeasibility marked in red.

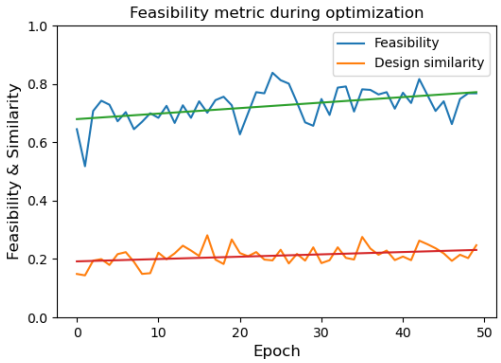

Figure 3 - Feasibility and similarity observed during the optimization epochs. Both the feasibility and the design similarity remain stable throughout the whole process meaning our optimization framework can optimize the generated designs with no feasibility or diversity penalty.

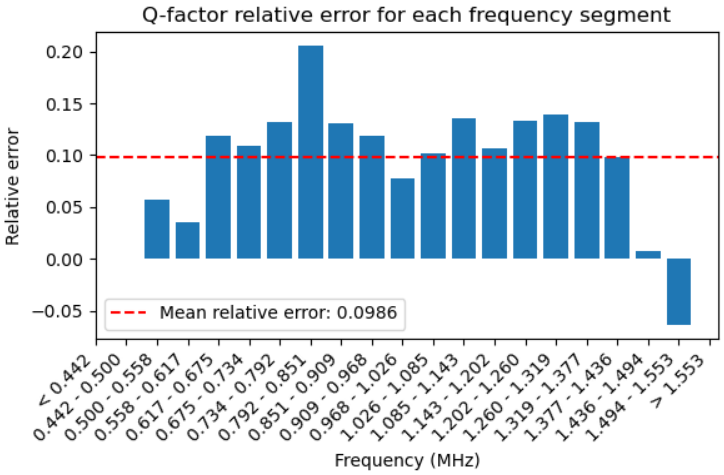

Figure 4 - Q-factor mean relative error when comparing with the non-optimized model. Most segments have improved Q factors suggesting our optimization framework improved the original model to the point of generating improved Q factor designs.

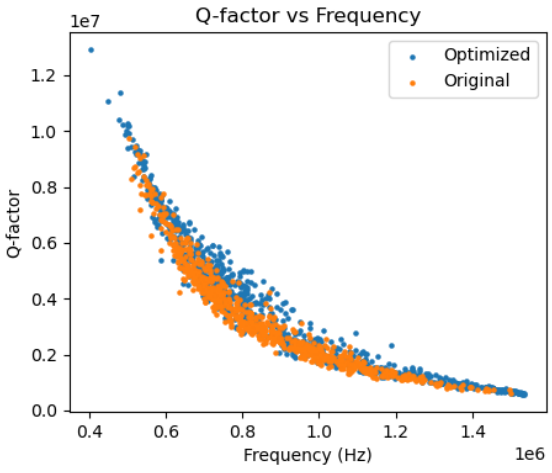

Figure 5 - Distribution of the frequencies and Q factors of designs generated by the original non-optimized and the optimized checkpoint.