Machine Learning for the Generative Design of Mechanical Metamaterials

Dissertation zur Erlangung des Doktorgrades der
Technischen Fakultät der
Albert-Ludwigs-Universität Freiburg im Breisgau

vorgelegt von

Gerrit Felsch

universität freiburg

Dekan
Prof. Dr.-Ing. Frank Balle

Referenten
Erstgutachter: Dr. Viacheslav Slesarenko
Zweitgutachter: Prof. Dr. Lars Pastewka

Datum der mündlichen Prüfung
21.11.2025

Bearbeitungszeit
Juni 2021 bis August 2025

Abstract

Mechanical metamaterials—engineered materials whose behavior is defined by their internal structure rather than their base material—have received great attention for the unusual properties they can exhibit. However, while the forward problem of predicting the mechanical behavior of a given metamaterial can be reliably done through simulation or experiments, the inverse problem—designing metamaterials with specific properties—remains difficult, especially for heterogeneous metamaterials composed of multiple unit cells. This dissertation explores the use of generative machine learning to address this inverse design problem.

The first part of the work investigates how inverse and generative models can be conditioned to produce unit cells with prescribed mechanical properties and how they can be used to generate multiple valid solutions for the same target. This is examined using a curved-beam metamaterial, which allows a compact geometric representation without complex design restrictions. The effect these restrictions have on the generative models is investigated in the second part. For this, a number of generative models—Variational Autoencoders, Generative Adversarial Networks, and Diffusion Models—are evaluated on a kirigami metamaterial where dependencies between the cuts cause intricate design constraints. The final part covers the challenges that arise when generative models are applied to heterogeneous metamaterials, in which multiple unit cells are combined to create spatially varying mechanical properties.

The results show that the applicability of generative models for the design of metamaterials strongly depends on the choice of geometric parameterization. Generative models make assumptions about their input data that generally hold for images, but can be invalid for metamaterials when complex dependencies between the input parameters exist. B-splines can provide a representation without those dependencies and are therefore well suited to be paired up with generative models. Furthermore, it is also important how these geometrical parameters are sampled to create the training data for the generative models, as this sampling also affects the distribution of the properties in the training data.

Overall, while generative machine learning models prove effective for the inverse design of mechanical metamaterials their successful application requires careful parameterization and selection of the training data.

Zusammenfassung

Mechanische Metamaterialien – technische Materialien, deren Eigenschaften durch ihre innere Struktur und nicht durch das Grundmaterial bestimmt werden – haben aufgrund ihrer ungewöhnlichen Eigenschaften große Aufmerksamkeit erregt. Während die Vorhersage des mechanischen Verhaltens eines gegebenen Metamaterials durch Simulationen oder Experimente zuverlässig möglich ist, bleibt das umgekehrte Problem – der Entwurf von Metamaterialien, die bestimmte mechanische Eigenschaften aufweisen – schwierig, insbesondere bei heterogenen Metamaterialien, die aus mehreren Einheitszellen bestehen. In dieser Dissertation wird der Einsatz generativer maschineller Lernmodelle zur Lösung dieses inversen Designproblems untersucht.

Im ersten Teil der Arbeit wird untersucht, wie inverse und generative Modelle konditioniert werden können, um Einheitszellen mit vorgegebenen mechanischen Eigenschaften zu erzeugen, und wie sie verwendet werden können, um mehrere gültige Lösungen für dasselbe Ziel zu generieren. Dies wird anhand eines Metamaterials mit gebogenen Balken untersucht, das eine kompakte geometrische Darstellung ohne komplexe Designeinschränkungen ermöglicht. Die Auswirkungen solcher Einschränkungen auf generative Modelle werden im zweiten Teil gesondert bewertet. Dazu wird eine Reihe generativer Modelle – Variationale Autoencoder, Generative Adversarial Networks und Diffusionsmodelle – auf einem Kirigami-Metamaterial evaluiert, bei dem die Abhängigkeiten zwischen den Schnitten komplizierte Designeinschränkungen verursachen. Der letzte Teil befasst sich mit den Herausforderungen, die sich ergeben, wenn generative Modelle auf heterogene Metamaterialien angewendet werden, bei denen mehrere Einheitszellen kombiniert werden, um räumlich variierende mechanische Eigenschaften zu erzeugen.

Die Ergebnisse zeigen, dass die Anwendbarkeit generativer Modelle für den Entwurf von Metamaterialien stark von der Wahl der geometrischen Parametrisierung abhängt. Generative Modelle machen Annahmen über ihre Eingabedaten, die im Allgemeinen für Bilder gelten, aber für Metamaterialien ungültig sein können, wenn komplexe Abhängigkeiten zwischen den Eingabeparametern bestehen. Es hat sich gezeigt, dass B-Splines eine Darstellung ohne diese Abhängigkeiten liefern

können und daher gut geeignet sind, um mit generativen Modellen gekoppelt zu werden. Darüber hinaus ist es auch wichtig, wie diese geometrischen Parameter abgetastet werden, um die Trainingsdaten für die generativen Modelle zu erstellen, da diese Abtastung auch die Verteilung der Eigenschaften in den Trainingsdaten beeinflusst.

Insgesamt erweisen sich generative maschinelle Lernmodelle zwar als effektiv für das inverse Design mechanischer Metamaterialien, ihre erfolgreiche Anwendung erfordert jedoch eine sorgfältige Konstruktion der Parametrisierungen und Auswahl der Trainingsdaten.

Journal Publications

This thesis comprises parts and figures from the following journal articles:

- Felsch, G., & Slesarenko, V. *Generative models struggle with kirigami metamaterials. Scientific Reports*, **14**, 19397 (2024)
- Felsch, G., & Ghavidelnia, N., & Schwarz, D., & Slesarenko, V. *Controlling auxeticity in curved-beam metamaterials via a deep generative model. Computer Methods in Applied Mechanics and Engineering*, **410**, 0045-7825 (2023)

Further contributions to publications during the PhD:

- Joedicke, I., & Ghavidelnia, N., & Felsch, G., & Slesarenko, V. *Addressing manufacturing defects in architected materials via anisotropy: minimal viable case. Acta Mechanica*, **235**(5), 2715–2724 (2024)
- Schwarz, D., & Felsch, G., & Tauber, F., & Schiller, S., & Slesarenko, V. *Exploiting self-contact in mechanical metamaterials for new discrete functionalities. Materials & Design*, **236**, 112468 (2023)

Conference contributions

Oral presentations on international conferences:

- Generative Design of Metamaterials: Successes and Failures, *17th International Conference on Advanced Computational Engineering and Experimenting*, Barcelona, Spain – July 2024
- Programming Instabilities in Curved-Beam Metamaterials via Deep Generative Models, *9th European Congress on Computational Methods in Applied Sciences and Engineering*, Lisbon, Portugal – June 2024
- Generative Design of Curved Beam Metamaterials, *1st International Conference and Scientific Exhibition on Living Materials Systems*, Freiburg, Germany – March 2023
- Generative Design of Perforated Mechanical Metamaterials, *11th European Solid Mechanics Conference*, Galway, Ireland – July 2022

Poster presentations on international conferences:

- Generate It! When Generative Models Fail to Generate, *1st Conference on Artificial Intelligence in Materials Science and Engineering*, Saarbrücken, Germany – November 2023

Contents

Symbol Definitions

Symbol	Description
$*$	Convolution operator
D_{E}	Euclidean distance
D_{JS}	Jensen–Shannon divergence
D_{KL}	Kullback-Leibler divergence
E	Young's modulus
G_{Θ}	Neural Network graph
N	Number of samples
N_{epochs}	Number of training epochs
Θ	Model parameters
η	Learning rate
$\hat{x}$	Generated element in design space
$\mathcal{G}_{\Theta}$	Function described by G_{Θ}
$\mathcal{L}$	Loss function
$\mathcal{O}$	Objective function
$\mathcal{X}$	Dataset
ν	Poisson's ratio
$\mathbf{A}$	Matrix (bold uppercase)
$\mathbf{A} \odot \mathbf{B}$	Hadamard product between $\mathbf{A}$ and $\mathbf{B}$
$\mathbf{a}$	Vector (bold lowercase)
ε	Strain
x	Element from design space
y	Element from property space
z	Element from latent space

Acronyms

Acronyms	Description
ACGAN	Auxiliary Classifier Generative Adversarial Network
AR	Auxiliary Regressor
BCE	Binary Cross-Entropy
CGAN	Conditional Generative Adversarial Network
CNN	Convolutional Neural Network
CVAE	Conditional Variational Autoencoder
DDPM	Denoising Diffusion Probabilistic Model
ED	Euclidean Distance
FEA	Finite Element Analysis
FGMM	Functionally Graded Metamaterial
FNN	Feedforward Neural Network
GAN	Generative Adversarial Network
GNN	Graph Neural Network
JS	Jensen–Shannon
KL	Kullback-Leibler
MSE	Mean Squared Error
NN	Neural Network
TNN	Tandem Neural Network
TTUR	Two-Timescale Update Rule
VAE	Variational Autoencoder
VAR	Versatile Auxiliary Regressor
WGAN	Wasserstein Generative Adversarial Network

Figure Index

Chapter 1

Introduction

While the mechanical properties of conventional materials are primarily determined by their molecular or atomic composition, the properties of mechanical metamaterials are determined by their internal architecture [1]. These metamaterials are usually built from a periodically repeated building block, known as a unit cell. Based on their internal architecture, metamaterials can exhibit mechanical properties that even surpass the intrinsic properties of the constituent material, particularly in terms of the stiffness-to-weight ratio [2] and the achievement of a negative Poisson's ratio [3].

Designing these metamaterials is often challenging, as the relationship between geometry and mechanical response tends to be complex and highly nonlinear, making it impossible to describe analytically. Finite Element Analysis (FEA) at least allows the forward problem—predicting the mechanical properties of a given metamaterial—to be solved numerically. Unfortunately, from a design perspective, the inverse problem—generating a metamaterial structure that exhibits specific mechanical properties—is much more interesting and is typically ill-posed, as there are often multiple distinct structures that achieve similar properties. While this is certainly true when considering all possible unit cells, it even holds when the design space is restricted to parameterized unit cell variants within a particular metamaterial family.

Still, for the design of a single unit cell, the inverse problem can often be reliably solved using topology optimization or genetic algorithms, although these methods can be computationally expensive and may get stuck in local optima. However, when the design of multiple unit cells is required, it can be more efficient to create a model of the inverse process instead. Although creating such a model might be more costly than running a single topology optimization, it becomes increasingly advantageous as the number of required unit cells grows. This has become particularly important with the recent shift towards combining multiple

unit cells to create heterogeneous metamaterials with locally varying mechanical properties, which is desirable in many applications.

Finding such a model that solves the inverse problem is very similar to the conditional generation tasks at which generative machine learning models have recently been highly successful. Given their ability to generate complex images, they may also offer a way to incorporate more intricate design elements into metamaterials. However, when applying these models to the design of metamaterials, one must not only choose from a variety of machine learning algorithms, but also select a suitable representation for the metamaterial—decisions that may be influenced by the mechanical mechanisms involved and by whether the metamaterial is one based on a single unit cell or a heterogeneous one.

1.1 Goals

This dissertation is concerned with how generative models can be applied to the design of mechanical metamaterials with complex design elements, how different representations influence these models, and what challenges emerge when combining multiple unit cells. The specific objectives are:

- **To investigate how conditional generative models can be applied to the inverse design of mechanical metamaterials.** The focus lies on the use of generative models and their ability to produce multiple distinct unit cells with the same mechanical properties.

- **To evaluate how the geometric representation of metamaterials influences the performance and expressiveness of generative models.** In particular, to study how design constraints affect the model's ability to adequately cover the design space and to map between geometry and mechanical properties.

- **To identify and address the unique challenges associated with designing heterogeneous metamaterials.** Specifically, to explore the difficulties that arise from transitioning between geometric space and property space in the design of structures composed of multiple unit cells.

1.2 Outline

This thesis is divided into four main chapters, each exploring different aspects of applying generative machine learning models to the design of mechanical metamaterials. Chapter 2 provides a review of prior work on mechanical metamaterials and the integration of machine learning in this context. It also introduces the deep learning models that form the foundation for the following chapters. Chapter 3 explores how these models can be applied to generate unit cells for curved-beam metamaterials, designed to exhibit specific Poisson's ratios. These unit cells are parameterized using Bézier splines, allowing for a representation that avoids complex restrictions on the design space. Chapter 4 then moves on to kirigami-based metamaterials, which are subject to intricate design constraints, and discusses how these constraints affect the behavior of the machine learning models. Finally, Chapter 5 examines how machine learning can be used to design shape-morphing heterogeneous metamaterials, and explores the challenges that arise for the machine learning models when combining multiple unit cells.

Chapter 2

Background

2.1 Mechanical Metamaterials

Metamaterials are artificially engineered materials that derive their properties primarily from their internal structure rather than from the intrinsic characteristics of their constituent materials [4]. These properties can even surpass those of the base materials and exhibit behaviors not typically found in conventional materials [5]. Metamaterials are usually constructed from unit cells—fundamental building blocks that are repeated periodically in two or three dimensions. This periodic arrangement ensures that the material's properties are mostly homogeneous on a larger scale and allows the behavior of the entire metamaterial to be described through the mechanics of this unit cell [6]. Metamaterials are often categorized according to the physical phenomena they are designed to control, with mechanical metamaterials primarily concerned with influencing deformation, stress distribution, and the flow of mechanical energy [4].

2.1.1 Historic Development

Compared to other branches of metamaterials, such as electromagnetic or acoustic metamaterials, the branch of mechanical metamaterials is a relatively new addition to the field [4]. Although it is difficult to pinpoint exactly when the field of metamaterials itself began, four influential papers were instrumental in its establishment [15]. In the first of these works—published in 1968—Veselago [16] theorized that a material with a negative refractive index could be created if both its electric permittivity and magnetic permeability were negative. Nearly three decades later, this theory was revisited by Smith et al. [17], who used a lattice of split-ring resonators [18] to create a material fulfilling these conditions. Shortly thereafter, Shelby et al. [19] experimentally demonstrated that this ma-

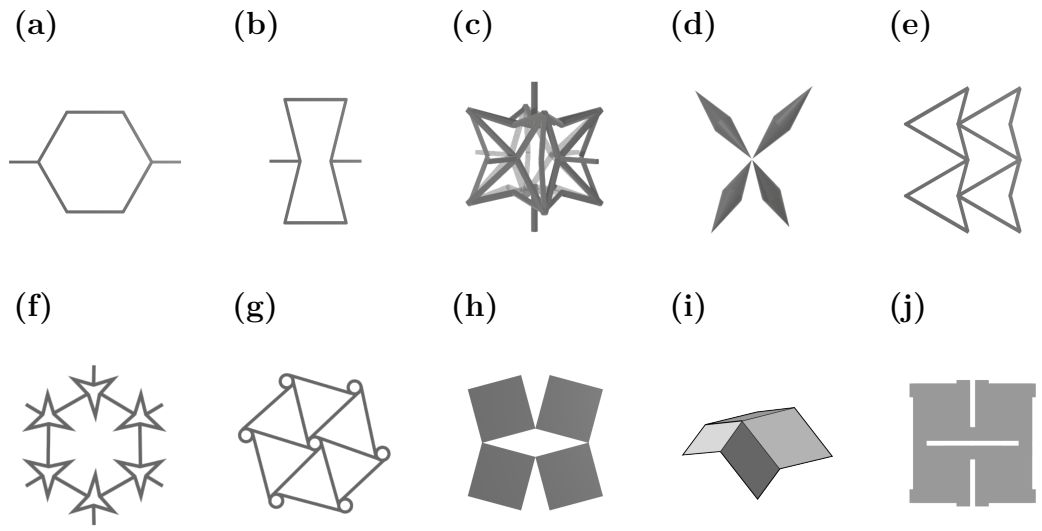

Figure 2.1: Unit cells of selected metamaterials: hexagonal [7] **(a)**, re-entrant [7] **(b)**, 3D re-entrant [8] **(c)**, pentamode [6] **(d)**, STAR-3 [9] **(e)**, arrowhead [10] **(f)**, hexa-chiral [11] **(g)**, rotating squares [12] **(h)**, Miura-ori tessellation [13] **(i)**, and a kirigami metamaterial [14] **(j)**.

terial indeed possessed a negative refractive index, thereby validating Veselago's theory and sparking widespread interest in metamaterials. This interest grew further when Pendry [20], in the last of these four seminal papers, showed that a material with a negative refractive index could be used to construct a perfect lens. Together, these works demonstrated that metamaterials can exhibit unusual properties, including some not found in natural materials [21]. This ability to produce properties beyond the range of conventional materials gave rise to the term "metamaterial," derived from the Greek prefix "meta-" meaning "beyond" or "transcending".

This idea of using internal organization to create materials that could exhibit unusual properties later spread to other areas, giving rise to the branches of acoustic metamaterials [22] and mechanical metamaterials [23]. However, while the concept of mechanical metamaterials is relatively young, specific classes of mechanical metamaterials with unusual properties have been known for a few decades [24]. One of the most well-known classes of these materials—two-dimensional hexagonal materials—was shown by Gibson *et al.* [7] in 1982 to exhibit a negative Poisson's ratio when specific geometrical parameters were used (see Figure 2.1a-b). Building on these observations, Lakes [8] transformed a conventional open-cell foam into a material with a negative Poisson's ratio by creating the 3D version of the re-entrant honeycomb structure shown in Figure 2.1c. Besides this, the discovery by Gibson *et al.* [7] inspired many more works on these structures of metamaterials with negative Poisson's ratio [25–27].

While most of these early works focused on the Poisson's ratio, Milton and Cherkaev [6] investigated in 1995 how the internal architecture influences the stiffness tensor of materials. They found that by combining an ordinary elastic material with regions of empty space, it is possible to design structured materials with any form of stiffness tensor not forbidden by thermodynamics. Using these findings, a new class of metamaterials—called *Extremal materials*—was developed. These materials are very stiff in one deformation mode but highly compliant in others. They are classified as unimode, bimode, trimode, quadramode, or pentamode, depending on whether they exhibit near-zero stiffness in one, two, three, four, or five modes of deformation, respectively. Among these, pentamode materials have received the greatest attention, as they behave similarly to a fluid by primarily resisting volumetric deformations and not shear, due to their high bulk modulus and low shear modulus [6]. Figure 2.1d shows the unit cell of one such pentamode metamaterial developed by Milton *et al.* [6].

While these examples of mechanical metamaterial exhibited promising properties, many were difficult to manufacture, limiting their experimental evaluation. This changed with the development of new additive manufacturing techniques, which enabled the fabrication of free-standing, complex 3D geometries while reducing manufacturing costs and allowing direct fabrication from CAD [28]. For instance, additive manufacturing allowed Kadic *et al.* [29] to create a pentamode material in 2012, which had previously only been theorized and not realized experimentally.

Another key area of mechanical metamaterials—the design of stiff, lightweight structures—has also been advanced by additive manufacturing [30]. For example, Yang *et al.* [31] used electron beam melting to manufacture a 3D re-entrant structure from a titanium alloy, which allowed them to create lightweight structures with high specific strength. Even lighter structures have been created through creating hollow tubular nanolattices, where a structure fabricated using two-photon lithography was coated with aluminum oxide, and the original material was subsequently etched away [32]. Another stiff, ultralight material was created by Zheng *et al.* [2], who used projection microstereolithography to produce a metamaterial that maintained an almost constant stiffness per unit mass density, even when the density was reduced by three orders of magnitude.

Advances in additive manufacturing have also enabled the recent development of structures with locally varied material properties, allowing the functional grading of stiffness [33] and and Poisson's ratio [34] of the metamaterial. This ability can be used to influence the deformation behavior of the metamaterial under load, giving rise to shape-matching metamaterials, which are designed to achieve a predefined shape when subjected to a specific load [35–37].

2.1.2 Auxetic Metamaterials

Auxeticity, which is characterized by a negative Poisson's ratio [3], is one of the most well-studied manifestations of unusual behavior in mechanical metamaterials. Unlike conventional materials, which shrink laterally under uniaxial tension, these metamaterials expand laterally instead [38]. Auxetic metamaterials are usually defined by a single unit cell that repeats itself in two or three dimensions [39], with the auxetic behavior encoded in this unit cell. Since the discovery of auxetic behavior in honeycomb structures by Gibson *et al.* [7] in 1982, numerous mechanisms have been developed to achieve auxeticity within a unit cell. The most prominent of these include re-entrant structures, chiral structures, rotating rigid (semi-rigid) structures, as well as origami and kirigami materials [3, 40, 41].

Re-entrant structures

The most basic form of re-entrant structures is the 2D re-entrant honeycomb investigated by Gibson et al. [7]. These structures differ from classic honeycombs (see Figure 2.1a) only in the angle between some of the beams, causing the vertical ribs of the unit cell to point inward (see Figure 2.1b). Under a vertical tensile load, these ribs straighten, leading the re-entrant unit cell to expand horizontally. Similarly, under compression, the vertical ribs direct the force inward, causing horizontal contraction. This behavior is mostly dominated by hinging at the junctions between the beams of the unit cell, while flexure and stretching of the beams play only a minor role [42]. The strength of this hinging effect depends on the stiffness of the beams, with stiffer beams resulting in a less negative Poisson's ratio and a higher Young's modulus in the vertical direction [43]. However, the dominant factor determining the sign and magnitude of the Poisson's ratio is still the re-entrant angle between the horizontal and vertical beams. When considering metamaterials where only this angle varies, both classic honeycomb and re-entrant structures belong to the same family. As long as this angle is smaller than 90°, the vertical ribs point inward, resulting in a negative Poisson's ratio; for angles larger than 90°, the Poisson's ratio becomes positive [44, 45]. In active metamaterials, it is even possible to switch the sign of the Poisson's ratio dynamically by adjusting this angle [46].

The same deformation mechanism as in the re-entrant honeycomb structures has been used to create a variety of 2D auxetic metamaterials [40]. One notable example of these structures is the double-arrowhead structure, developed by by Larsen *et al.* [10] through structural optimization, which exhibits a negative

Poisson's ratio based on the re-entrant mechanism (see Figure 2.1e). Additionally, a family of star-shaped metamaterials utilizes the same principle to achieve auxetic behavior. In this family, each unit cell is constructed using n arrow-shaped building blocks arranged in a star configuration. These stars are connected to their n neighboring unit cells through n beams, leading to the designations *STAR-n* for these structures [9]. A STAR-3 metamaterial is shown in Figure 2.1f.

The same principles underlying these 2D auxetic structures can also be extended to create 3D auxetic structures. Lakes [8] created auxetic foams using a 3D extension of the re-entrant honeycomb structure, while Lim [47] designed a 3D version of the arrowhead structure and Ras *et al.* [48] introduced a 3D star-shaped metamaterial. Beyond these truss-based designs, the re-entrant mechanism can also be applied to 3D shell structures. These structures rely on the buckling of the shells to create a re-entrant mechanism leading to a negative Poisson's ratio [49].

Chiral structures

In its literal sense, chiral metamaterials refers to all metamaterials that lack a center of symmetry; in other words, structures that are not equivalent to their mirror image [11]. However, over time, this term has come to describe metamaterials whose behavior is dominated by a combination of rotation of the unit cell and flexure of re-entrant beams [40]. The earliest description of chiral structures in relation to mechanics goes back to a molecular model by Wojciechowski [50] with rotational degrees of freedom and inverse interaction with the nearest neighbors. This model showed auxetic behavior when the molecules were given a preferred orientation or tilt. The metamaterial equivalent of this model, introduced by Lakes [11], consists of a central disc with several tangentially attached beams.

These or similar rotational units are typically used as the basic building blocks of chiral metamaterials. Depending on how the beams are attached to the central disc and the resulting rotation direction, the basic units can be classified as either *left-handed* or *right-handed*. Using these basic units as unit cells for a periodic metamaterial is only possible for three, four or six beams per basic unit, otherwise no space filling design is possible [51]. These configurations are known as tri-chiral, tetra-chiral, and hexa-chiral metamaterials, respectively. A hexa-chiral metamaterial is shown in Figure 2.1g. Usually, multiple of these basic units are combined to form the unit cell of a metamaterial. Depending on how *left-handed* or *right-handed* units are combined these are called chiral, anti-chiral or meta-

chiral materials [51]. Chiral metamaterials consist exclusively of one type of unit, while anti-chiral metamaterials are constructed such that each unit is connected only to units of the opposite handedness. Meta-chiral metamaterials cover all remaining combinations. The Poisson's ratio of these structures depends on the thickness of the beams [52] and the ratio between beam length and disc size [53], with negative values reported for hexa-, tetra- and anti-tetra-chiral metamaterials [52].

Rotating rigid (semi-rigid) structures

Rotating rigid structures are usually composed of multiple rigid geometric elements connected by hinges. Under loading, these elements rotate around the hinges, altering the horizontal and vertical dimensions of the metamaterial, which can cause auxetic behavior. The first rotating rigid structure with a negative Poisson's ratio was introduced by Grima and Evans [12] in 2000. Their design features rigid squares, which are connected to one neighboring square at each edge (see Figure 2.1h). Under ideal conditions, this system is isotropic and has a Poisson's ratio of -1, as long as the squares are able to still rotate under the given load. Similar auxetic metamaterials have been developed using other shapes, including triangles [54], rectangles [55] and parallelograms [56]. These other structures can exhibit a positive or negative Poisson's ratio depending on factors such as geometrical parameters or the current angle between the shapes.

While most of these works assume the structures to be rigid, *semi-rigid* structures, where the shapes were allowed to undergo deformation have also been considered [57]. However, while these structures can provide an easy way to facilitate auxeticity, they are prone to stress concentrations, as the shapes are only connected at the hinges [58].

Origami and kirigami metamaterials

Origami materials are named after the Japanese art of paper folding, while kirigami structures also allow cuts in addition to folds. Origami-based structures are typically designed as 2D sheets with predefined fold patterns. These folds are used to create out-of-plane deformations, essentially transforming the 2D sheet into a 3D structure to increase the number of degrees of freedom. Placement of these folds allows great control over how the shape of the sheet is deformed [59–61]. This can also be used to modify in-plane behavior, allowing to create materials with a negative Poisson's ratio [62, 63]. The most well known of these auxetic origami structures is the Miura-ori tessellation [13] (see Figure 2.1i), which

has a negative Poisson's ratio for in-plane deformation and a positive Poisson's ratio for out-of-plane deformation. While the Poisson's ratio of these metamaterials is mostly determined by the fold, the stiffness depends more on the thickness of the material and the ability of the fold to act as a hinge. Fully 3D metamaterial can be created using origami techniques by stacking multiple folded layers [62] or by assembling tubes folded from the flat sheets [64].

By introducing cuts alongside folds, kirigami metamaterials further expand the design possibilities, enabling even better control over the deformation of the sheets [58]. That said, cuts on their own already have the capability to control the Poisson's ratio. So can the most rotating rigid structures be manufactured by placing cuts in flat sheets, only leaving connections between the shapes at the hinges. Additionally, Grima *et al.* [14] demonstrated that Poisson's ratio and stiffness of a flat rubber sheet can be influenced by placing cuts with varying orientations in it. The unit cell of this metamaterial is shown in Figure 2.1j. Du *et al.* [65] showed that auxetic behavior can even be retained under large deformations up to 50% applied strain, by exploiting out of plane buckling.

2.2 Continuum Mechanics and Finite-Element Foundations

This section introduces the physical foundations governing the behavior of deformable bodies and outlines how their mechanical response can be predicted using the Finite Element Method (FEM). The description of the mechanical behavior follows the continuum mechanics approach outlined by Holzapfel [66], while the description of FEM follows the work of Dinkler and Kowalsky [67] and the lecture notes by Bathe [68] and Kelly [69].

2.2.1 Fundamentals of Continuum Mechanics

While physical objects in the real world are composed of molecules, and their properties arise from molecular interactions, predicting the mechanical behavior of a structure directly from these interactions is generally infeasible. Instead, continuum mechanics models the system at the macroscopic level using local averages of microscopic quantities. This continuum description consists of three main components: (1) the kinematics, which describe motion and deformation; (2) the internal stresses and the constitutive equations, which relate them to the deformation; and (3) the balance principles for linear and angular momentum.

Kinematics

Kinematics are concerned with the motion and deformation of the continuum. A movement or deformation can be described by mapping each material point from a reference (initial) position $\mathbf{X}$ to a new position $\mathbf{x}$ in the deformed configuration. This mapping is given by the displacement field $\mathbf{u}$. In the Lagrangian form, this field is a function of the original position:

$$\mathbf{u}(\mathbf{X}) = \mathbf{x}(\mathbf{X}) - \mathbf{X} \tag{2.1}$$

While this generally characterizes how each point of the continuum moves, measuring the deformation requires determining the effect of the displacement field $\mathbf{u}(\mathbf{X})$ on the size and shape of the body. For this purpose, the *deformation gradient* $\mathbf{F}$, given by

$$\mathbf{F} = \frac{\partial \mathbf{x}}{\partial \mathbf{X}} = \mathbf{I} + \nabla \mathbf{u} \tag{2.2}$$

is crucial. The determinant of the deformation gradient, $J = \det(\mathbf{F})$, represents the local volume change ratio. For physically meaningful deformations, it is required that $J > 0$ everywhere.

The deformation gradient $\mathbf{F}$ is also particularly important in the definition of strain tensors, which describe the relative deformation with respect to the reference and current configurations. These strain measures are necessary from an engineering standpoint, as $\mathbf{F}$ is non-zero even for pure rotations. An ideal deformation measure, however, would be zero under pure rotation. Fortunately, polar decomposition allows us to split the deformation gradient into a pure stretch tensor $\mathbf{U}$ and a pure rotation tensor $\mathbf{R}$:

$$\mathbf{F} = \mathbf{R}\mathbf{U} \tag{2.3}$$

Since a rotation followed by its transpose results in the identity, this decomposition allows the rotational component to be eliminated by multiplying $\mathbf{F}$ with its transpose $\mathbf{F}^T$:

$$\mathbf{F}^T\mathbf{F} = \mathbf{U}^T\mathbf{R}^T\mathbf{R}\mathbf{U} = \mathbf{U}^T\mathbf{U} \tag{2.4}$$

The resulting tensor, $\mathbf{C} = \mathbf{F}^T\mathbf{F}$, is called the *right Cauchy-Green deformation tensor*. It contains information about the stretch of the material while excluding any rotational effects. It is the basis for the most common strain measure, the *Green-Lagrangian strain tensor*, which essentially measures how close the right Cauchy-Green deformation tensor $\mathbf{C}$ is to the identity $\mathbf{I}$:

$$\boldsymbol{\varepsilon} = \frac{1}{2}(\mathbf{C} - \mathbf{I}) \tag{2.5}$$

Using Equation 2.2, this can be expressed in terms of $\nabla \mathbf{u}$:

$$\boldsymbol{\varepsilon} = \frac{1}{2}(\mathbf{F}^T\mathbf{F} - \mathbf{I}) \tag{2.6}$$

$$= \frac{1}{2}((\mathbf{I} + \nabla \mathbf{u})^T(\mathbf{I} + \nabla \mathbf{u}) - \mathbf{I}) \tag{2.7}$$

$$= \frac{1}{2}(\nabla \mathbf{u} + \nabla \mathbf{u}^T + \nabla \mathbf{u} \nabla \mathbf{u}^T) \tag{2.8}$$

For small deformations, the second-order term can be neglected, leaving:

$$\boldsymbol{\varepsilon} = \frac{1}{2}(\nabla \mathbf{u} + \nabla \mathbf{u}^T) \tag{2.9}$$

This can alternatively be written in terms of the deformation gradient $\mathbf{F}$ as:

$$\boldsymbol{\varepsilon} = \frac{1}{2}(\mathbf{F} + \mathbf{F}^T) - \mathbf{I} \tag{2.10}$$

Stress

The motion and deformation of a body are deeply tied to interactions between material and neighboring material. One of these interactions is stress. Stress determines how the traction force $\mathbf{t}$ (force per unit area) on a boundary surface $\partial\Omega$ of a continuum body $\mathcal{B}$ occupying a region Ω relates to the unit normal vector $\mathbf{n}$ of the surface:

$$\mathbf{t}(\mathbf{x}, \mathbf{n}) = \boldsymbol{\sigma}\mathbf{n} \tag{2.11}$$

Here, $\boldsymbol{\sigma}$ is a symmetric spatial tensor field called the *Cauchy* (or *true*) *stress tensor*. For continuum mechanics, it is crucial to understand how this stress tensor relates to the deformation of a material. This relation is material-specific and governed through a constitutive model. In the regime of small deformations and small strains, many materials can be accurately described by the theory of linear elasticity. In this setting, the displacement gradients are small, and the linearized (infinitesimal) strain tensor (Eq. 2.10) is used. The stress is then linearly related to the strain by Hooke's law:

$$\boldsymbol{\sigma} = \boldsymbol{C} : \boldsymbol{\varepsilon} \tag{2.12}$$

where $\boldsymbol{C}$ is the fourth-order elasticity tensor, which encodes the material's stiffness. For isotropic materials, this reduces to:

$$\boldsymbol{\sigma} = \lambda \operatorname{tr}(\boldsymbol{\varepsilon})\, \mathbf{I} + 2\mu\, \boldsymbol{\varepsilon} \tag{2.13}$$

where λ and μ are the Lamé parameters, and $\operatorname{tr}(\cdot)$ denotes the trace operator.

Balance Principles

In addition to the kinematical principles, there are universal physical principles that the deformation of a body must obey. The most important of these for continuum mechanics are the balances of linear (**L**) and angular (**J**) momentum. These can be described through the resultant force **P** and the resultant moment **M**:

$$\frac{\partial \mathbf{L}(t)}{\partial t} = \mathbf{P}(t) = \int_{\partial\Omega} \mathbf{t}\,\mathrm{d}s + \int_{\Omega} \mathbf{b}\,\mathrm{d}v \tag{2.14}$$

$$\frac{\partial \mathbf{J}(t)}{\partial t} = \mathbf{M}(t) = \int_{\partial\Omega} \mathbf{r} \times \mathbf{t}\,\mathrm{d}s + \int_{\Omega} \mathbf{r} \times \mathbf{b}\,\mathrm{d}v \tag{2.15}$$

where t is time, $\mathbf{t}$ the traction force, $\mathbf{b}$ the body force, and $\mathbf{r} = \mathbf{x} - \mathbf{x}_0$ the position vector relative to an origin $\mathbf{x}_0$.

To be in equilibrium, both of these derivatives must be zero. Furthermore, in most cases, it can be assumed that the body forces are negligible. First substituting using Eq. 2.11, and then applying the divergence theorem, the expressions can be converted to volume integrals:

$$\mathbf{P}(t) = \int_{\partial\Omega} \mathbf{t}\,\mathrm{d}s = \int_{\partial\Omega} \boldsymbol{\sigma}\mathbf{n}\,\mathrm{d}s = \int_{\Omega} \mathrm{div}\boldsymbol{\sigma}\,\mathrm{d}v \tag{2.16}$$

$$\mathbf{M}(t) = \int_{\partial\Omega} \mathbf{r} \times \mathbf{t}\,\mathrm{d}s = \int_{\partial\Omega} \mathbf{r} \times \boldsymbol{\sigma}\mathbf{n}\,\mathrm{d}s = \int_{\Omega} \mathbf{r} \times \mathrm{div}\boldsymbol{\sigma} + \boldsymbol{\mathcal{E}} : \boldsymbol{\sigma}^T\,\mathrm{d}v \tag{2.17}$$

where $\boldsymbol{\mathcal{E}}$ is the third-order Levi-Civita permutation tensor.

As the region Ω and the volume v are arbitrary, it follows that the integrands must vanish at every point of the continuum:

$$\mathrm{div}\boldsymbol{\sigma} = \mathbf{0} \tag{2.18}$$

$$\boldsymbol{\mathcal{E}} : \boldsymbol{\sigma}^T = \mathbf{0} \tag{2.19}$$

Equation 2.19 holds if and only if $\boldsymbol{\sigma}$ is symmetric; therefore, $\boldsymbol{\sigma} = \boldsymbol{\sigma}^T$.

2.2.2 Finite Element Method (FEM)

While continuum dynamics provide an accurate description of the physical behavior of deformable bodies, solving the resulting equations is often so complex that approximation methods become necessary. The *Finite Element Method (FEM)* is one such method. The general idea behind FEM is to discretize the body into a number of finite elements and then calculate the behavior of the system based

on the behavior of these elements. These elements are called "finite" because they cannot be infinitesimal; otherwise, the problem would revert to the original continuum formulation. These elements connect nodes, which are the points at which the physical behavior is approximated. Inside the elements, it is assumed that the behavior follows preselected shape functions, which use the values at the nodes as parameters. As a result, increasing the number of elements or using more complex shape functions allows for the approximation of more complicated behavior. For this part, we are going to work with vectorized quantities.

To evaluate the values at the nodes, the *Principle of Virtual Work* can be used. Let us consider a body under known body forces $\mathbf{b}$ and traction forces $\mathbf{t}$. To find the corresponding displacement field $\mathbf{u}(\mathbf{X})$ between the current and reference configurations, a small trick can be used. Imagine that the current configuration undergoes an infinitesimal displacement $\delta\mathbf{u}$. As the current configuration is in equilibrium, this is assumed to have no effect on the total potential energy Π. This implies that the virtual internal work δW_{int} must equal the virtual external work δW_{ext}:

$$\delta\Pi = \delta W_{\text{int}} - \delta W_{\text{ext}} = 0 \tag{2.20}$$

The external virtual work is generated by the traction force $\mathbf{t}$ and body force $\mathbf{b}$ acting on a point $\mathbf{x}$ that undergoes a virtual displacement $\delta\mathbf{u}$:

$$\delta W_{\text{ext}} = \int_{\partial\Omega} \mathbf{t} \cdot \delta\mathbf{u} \, \mathrm{d}s + \int_{\Omega} \mathbf{b} \cdot \delta\mathbf{u} \, \mathrm{d}v \tag{2.21}$$

Similarly, internal virtual work is generated by internal forces acting on points that undergo virtual displacements:

$$\delta W_{\text{int}} = \int_{\Omega} \boldsymbol{\sigma}^T \delta\boldsymbol{\varepsilon} \, \mathrm{d}v \tag{2.22}$$

Combining Eq. 2.20, Eq. 2.21, and Eq. 2.22, and neglecting the body forces again:

$$\int_{\Omega} \boldsymbol{\sigma}^T \delta\boldsymbol{\varepsilon} \, \mathrm{d}v - \int_{\partial\Omega} \mathbf{t}^T \delta\mathbf{u} \, \mathrm{d}s = 0 \tag{2.23}$$

This is where the shape functions come in. Through them, a displacement is approximated by the nodal displacements $\hat{\boldsymbol{u}} \in \mathbb{R}^{3n}$, where n is the number of nodes:

$$\mathbf{u}(\mathbf{x}) = \mathbf{N}(\mathbf{x})\hat{\boldsymbol{u}} \tag{2.24}$$

Here, $\mathbf{N}(\mathbf{x}) : \mathbb{R}^3 \to \mathbb{R}^{3\times 3n}$ is an agglomeration of all shape functions that maps a point $\mathbf{x}$ in the body to a weight matrix for the nodes. Since these weights determine

the contribution of each node, they sum up to one for each component:

$$\sum_{j=1}^{3n} \mathbf{N}(\mathbf{x})_{i,j} = 1, \quad \forall\, i \in \{1,2,3\} \tag{2.25}$$

The same also holds for virtual displacements, so Eq. 2.24 becomes:

$$\delta\mathbf{u}(\mathbf{x}) = \mathbf{N}(\mathbf{x})\delta\hat{\boldsymbol{u}} \tag{2.26}$$

This can be used to compute the virtual strain using Eq. 2.10, since the virtual displacement is infinitesimal. We define a function $\mathbf{B}(\mathbf{x}) : \mathbb{R}^3 \to \mathbb{R}^{6\times 3n}$ relating the strain to the displacement:

$$\delta\boldsymbol{\varepsilon} = \frac{1}{2}(\nabla\delta\mathbf{u} + \nabla\delta\mathbf{u}^T) = \frac{1}{2}(\nabla\mathbf{N}(\mathbf{x})\delta\hat{\boldsymbol{u}} + (\nabla\mathbf{N}(\mathbf{x})\delta\hat{\boldsymbol{u}})^T) = \mathbf{B}(\mathbf{x})\delta\hat{\boldsymbol{u}} \tag{2.27}$$

Now Eq. 2.23 can be updated to include the discretization:

$$\int_\Omega \boldsymbol{\sigma}^T\mathbf{B}(\mathbf{x})\delta\hat{\boldsymbol{u}}\,\mathrm{d}v - \int_{\partial\Omega} \mathbf{t}^T\mathbf{N}(\mathbf{x})\delta\hat{\boldsymbol{u}}\,\mathrm{d}s = 0 \tag{2.28}$$

The virtual displacement $\delta\hat{\boldsymbol{u}}$ can now be factored out:

$$\delta\hat{\boldsymbol{u}}^T\left(\int_\Omega \mathbf{B}(\mathbf{x})^T\boldsymbol{\sigma}\,\mathrm{d}v - \int_{\partial\Omega} \mathbf{N}(\mathbf{x})^T\mathbf{t}\,\mathrm{d}s\right) = 0 \tag{2.29}$$

Since $\delta\hat{\boldsymbol{u}}$ is arbitrary, this only holds if:

$$\int_\Omega \mathbf{B}(\mathbf{x})^T\boldsymbol{\sigma}\,\mathrm{d}v = \int_{\partial\Omega} \mathbf{N}(\mathbf{x})^T\mathbf{t}\,\mathrm{d}s \tag{2.30}$$

At this point, displacement and strain have vanished from the equation. We reintroduce them using a constitutive law relating stress to strain. Here, we use Hooke's law (see Eq. 2.12):

$$\int_\Omega \mathbf{B}(\mathbf{x})^T\boldsymbol{C}\boldsymbol{\varepsilon}\,\mathrm{d}v = \int_{\partial\Omega} \mathbf{N}(\mathbf{x})^T\mathbf{t}\,\mathrm{d}s \tag{2.31}$$

Substituting $\boldsymbol{\varepsilon} = \mathbf{B}(\mathbf{x})\hat{\boldsymbol{u}}$, we get:

$$\int_\Omega \mathbf{B}(\mathbf{x})^T\boldsymbol{C}\mathbf{B}(\mathbf{x})\hat{\boldsymbol{u}}\,\mathrm{d}v = \int_{\partial\Omega} \mathbf{N}(\mathbf{x})^T\mathbf{t}\,\mathrm{d}s \tag{2.32}$$

This leads to the final linear system of equations:

$$\underbrace{\int_{\Omega} \mathbf{B}(\mathbf{x})^T \boldsymbol{C} \mathbf{B}(\mathbf{x}) \, \mathrm{d}v}_{\mathbf{K}} \; \hat{\boldsymbol{u}} = \underbrace{\int_{\partial\Omega} \mathbf{N}(\mathbf{x})^T \mathbf{t} \, \mathrm{d}s}_{\mathbf{F}} \tag{2.33}$$

Here, $\mathbf{K} \in \mathbb{R}^{3n \times 3n}$ is the global stiffness matrix, while $\mathbf{F} \in \mathbb{R}^{3n}$ is the nodal force vector. This represents Hooke's law in its classical discretized form:

$$\mathbf{K}\hat{\boldsymbol{u}} = \mathbf{F} \tag{2.34}$$

Solving this system yields the nodal displacements for the applied traction forces.

2.3 Deep Learning

Machine learning is a broad field encompassing all algorithms that can automatically extract valuable information from data [70]. Valuable information is usually identified through correlations between different factors of variation within the data. The difficulty of finding these correlations usually depends on the representation the data is given in, as these factors can be entangled. Deep learning specializes in disentangling this features and discarding those that are irrelevant [71]. The principle behind deep learning is quite simple: it creates a hierarchy of representations in which increasingly complex representations are built from simpler ones. Each of these representations consists of pieces of information usually referred to as features. This hierarchical structure of feature processing is what gives deep learning its "depth"—the more hierarchical layers a model has, the deeper it is. The best example of this principle are deep neural networks, which are essentially nonlinear functions hierarchically constructed from different types of processing layers. This section covers how these deep neural networks operate and can be used to solve different types of machine learning problems. The ideas explained here are drawn from a combination of sources that provide different points of view on deep learning, the classic perspective [71, 72], the stochastic perspective [73, 74] and the mathematical details [75].

2.3.1 Basics of Neural Networks

The most essential deep learning models are deep feedforward networks. Such a feedforward network first and foremost just describes a (mostly nonlinear) function $f : X \to Y$ from an domain X to another domain Y. While this function and the

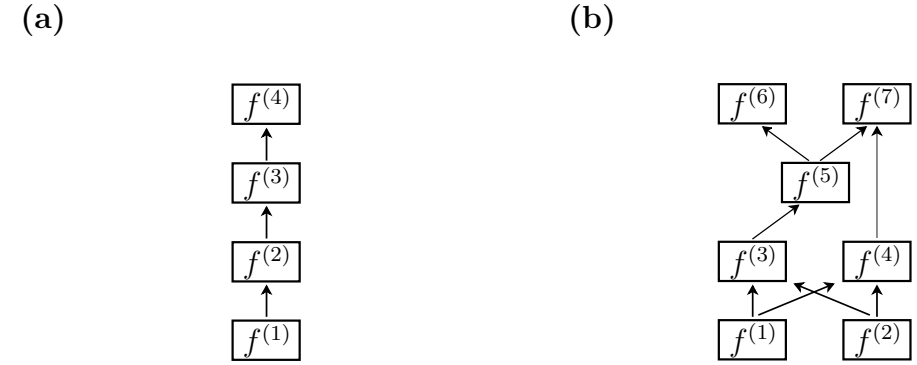

Figure 2.2: Exemplary graphs for two neural networks, one simple linear chain of layers **(a)** and a directed acyclic graph **(b)**.

domains are technically arbitrary, most research focuses on f, that transform one vector into another [76]. Images can be represented as vectors with dimensions determined by the pixel number [76], while words can be converted into a one hot encoding vector over some dictionary [77].

What gives such a function f, defined by a feedforward neural network, its interesting properties is the way it is constructed from a finite number n of other functions $f^{(i)} : X^{(i)} \to Y^{(i)},\ i \in \{1, \ldots, n\}$. Some $f^{(i)}$ serve as inputs to others, giving rise to a hierarchically structure in which the $f^{(i)}$ are called *layers* and the arrangement G the *neural network*. As the layers $f^{(i)}$ are generally arbitrary functions, they can themselves be composed of multiple sub-functions. Such compositions are often used to make the overall structure of the neural network easier to understand. While the simplest form of an neural network architecture is a linear chain of layers (see Figure 2.2a), more complex architectures can form directed acyclic graphs [71] (see Appendix A.1 for the definition), featuring multiple inputs and outputs, as well as skip connections that bypass intermediate layers [78] (see Figure 2.2b):

Definition 2.1 Neural network

A *neural network* is a directed acyclic graph $G_\Theta = (F, C)$ with the layers as nodes $f^{(i)} \in F$ and edges $C \subset F \times F$ which represent the connections between the layers. Where the set of layers is divided into the set of input layers F_{in}, the set of output layers F_{out} and the set of hidden layers F_{hidden}.

$$
\begin{aligned}
&F = F_{\text{in}} \cup F_{\text{out}} \cup F_{\text{hidden}} \\
&F_{\text{in}} \neq \emptyset, F_{\text{out}} \neq \emptyset \\
&F_{\text{hidden}} \cap (F_{\text{in}} \cup F_{\text{out}}) = \emptyset
\end{aligned}
$$

where the depth of the neural network is the maximal length of a path from a input layer to a output layer. Usually, each layer has a parameter vector $\boldsymbol{\theta}^{(i)}$ which together form the set of parameters Θ for the neural network.

If G_Θ has only one input layer $F_{\text{in}} = \{f^{(1)}\}$ and one output layer $F_{\text{out}} = \{f^{(n)}\}$, it describes a function $\mathcal{G}_\Theta : X^{(1)} \to Y^{(n)}$:

$$\begin{aligned} \mathcal{G}_\Theta &= g^{(n)}(\mathbf{x}) \\ g^{(i)}(\mathbf{x}) &= f^{(i)}(\mathcal{K}),\ \mathcal{K} = \{g^{(k)}(\mathbf{x}) | f^{(k)} \in \text{Par}(f^{(i)})\} \\ g^{(1)}(\mathbf{x}) &= f^{(1)}(\mathbf{x}) \end{aligned}$$

where $\text{Par}(f^{(i)})$ are the parent nodes of $f^{(i)}$.

While there are multiple types of layers, the most fundamental one is the fully connected layer. This layer performs an affine transformation (a linear transformation followed by a bias term):

Definition 2.2 Fully connected layer

A *fully connected layer* is a real-valued function $f_L : \mathbb{R}^n \to \mathbb{R}^m$ that is given by a weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ and a bias vector $\mathbf{b} \in \mathbb{R}^m$:

$$f_L(\mathbf{x}; \boldsymbol{\theta}) \coloneqq \mathbf{W}\mathbf{x} + \mathbf{b} \tag{2.35}$$

where $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$ are the parameters of f_L and m is called the width of the layer.

Affine and linear transformations, such as those used in Principal Component Analysis (PCA) [79, 80] and Linear Discriminant Analysis (LDA) [81], can be powerful for processing data; however, certain problems require non-affine functions to solve them. A prominent example of this is learning the XOR function [82].

Since combining two affine transformations results in another affine transformation, approximating more complex functions requires introducing a non-affine element to the layers. To achieve this, the affine transformation is typically paired with a uniform activation function—a one-dimensional, non-affine transformation applied independently to each element of the input vector:

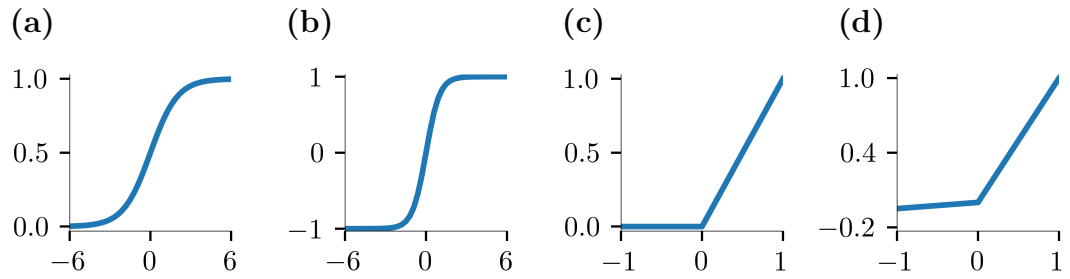

Figure 2.3: Activation functions commonly used in neural networks. Sigmoid: maps inputs to the range (0, 1) **(a)**. Tanh: similar to sigmoid but maps inputs to (–1, 1) **(b)**. ReLU (Rectified Linear Unit): outputs zero for negative inputs and linear for positive inputs **(c)**. Leaky ReLU: introduces a small slope for negative inputs to mitigate the zero gradient problem **(d)**.

Definition 2.3 Uniform activation function

A *uniform activation function* is a real-valued function $g : \mathbb{R} \to \mathbb{R}$, that is applied element-wise to the output vector of the previous layer:

$$f_A(\mathbf{x})_i \coloneqq g(f_L(\mathbf{x}; \boldsymbol{\theta})_i) \tag{2.36}$$

where f_A is the layer resulting from the combination of the fully connected layer with the activation function.

Unlike fully connected layers, these activation functions are fixed and introduce no additional parameters to the combined layer. In this work, the four different activation functions shown in Figure 2.3 will be used: the sigmoid, hyperbolic tangent, ReLU, and Leaky ReLU.

Definition 2.4 Different activation functions

Sigmoid:

$$\text{sig}(x) = \frac{1}{1 + e^{-x}} \tag{2.37}$$

Hyperbolic tangent:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.38}$$

ReLU:

$$\text{ReLU}(x) = \max(0, x) \tag{2.39}$$

Leaky ReLU:

$$\text{LReLU}(x) = \max(0, x) + \alpha \min(0, x) \tag{2.40}$$

where α is a fixed parameter chosen with the activation function.

These four activation functions are typically used in different contexts. The sigmoid function maps all real numbers to the interval $(0, 1)$, making it useful for

output layers in classification or regression tasks where values are constrained to a specific range. Similarly, the hyperbolic tangent activation function maps to $(-1, 1)$, which is useful when working with data that has been normalized to be centered around zero. Such data is often used in generative models, making the hyperbolic tangent popular as activation function of the output layer in generators. The Rectified Linear Unit (ReLU) [83] on the other hand, is more suitable for intermediate layers. However, it has the drawback that its derivative is zero for $x < 0$, which can be an inconvenience that will become evident during the explanation how these neural networks are trained. This is addressed in the Leaky Rectified Linear Unit (LReLU) [84] where the slope of zero has been replaced with one of α.

Depending on the activation functions used, as well as the depth and width of the layers, a neural network can approximate a wide range of functions. This is one of the aspects that make neural networks so powerful, almost every function between two vector spaces can be approximated by a neural network. To quantify how powerful different families of neural networks are, there exist several theorems that describe the types of functions they can approximate. These are known as universal approximation theorems. The two most famous of these theorems are by Cybenko [85] and Hornik *et al.* [86]. They demonstrated that neural networks can approximate all continuous functions from a compact set to the real numbers, and all Borel measurable functions, respectively. This holds for any desired degree of accuracy for neural networks with arbitrary width and bounded depth. As mentioned before, these theorems are valid only for specific activation functions. While Cybenko's theorem [85] is limited to sigmoid activations, the variant by Hornik *et al.* [86] considers a wider range of so-called squashing functions. Similar results have been obtained for a bounded layer width and arbitrary network depth—it is named deep learning, after all—when combined with ReLU activation functions [87].

Neural Network Training

Based on the universal approximation theorem it can be assumed that a neural network G_Θ can approximate a wide range of functions if it used the right activation function and is wide and deep enough. In practice, the important part is to select the parameters $\boldsymbol{\theta}^{(i)}$ of the the layers $f^{(i)}$ so G_Θ approximates a specific function f. In principle this is a optimization problem: finding the set Θ of $\boldsymbol{\theta}^{(i)}$, that minimizes the discrepancy between G_Θ and f. Solving this problem is commonly referred to as "training" G_Θ. However, f is often a black box, and the only way to gather information about it is through sampling, so G_Θ is optimized

to fit a training dataset $\mathcal{X}_{\text{train}} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}$, where $\mathbf{y}^{(i)} = f(\mathbf{x}^{(i)})$ instead. For this, a *loss function* $\mathcal{L}(f, \mathcal{G}_\Theta)$ is used, which quantifies the error between the true values $\mathbf{y}^{(i)}$ and the model predictions $\mathcal{G}_\Theta(\mathbf{x}^{(i)})$. This allows the training process to be formulated as the following minimization problem:

$$\min_{\Theta} \mathcal{O} := \sum_{i} \mathcal{L}(\mathbf{y}^{(i)}, \mathcal{G}_\Theta(\mathbf{x}^{(i)})) \tag{2.41}$$

Solving this problem requires efficient computation of gradients. Along with their expressive power, efficient gradient computation is one of the greatest strengths of neural networks. This efficiency comes from backpropagation, which leverages the layered composition of the neural network. First introduced in the context of optimal control in the 1960s [88, 89], backpropagation only gained recognition in the field of machine learning over a decade later [90]. The key to backpropagation lies in the application of the chain rule of calculus.

Theorem 2.1 Backpropagation

For each layer $f^{(i)} \in F$ of a neural network $G_\Theta = (F, C)$ and $\boldsymbol{z}^{(i)} = g^{(i)}(\mathbf{x})$, the Jacobian $\mathbf{J}_{\boldsymbol{\mathcal{L}}}(\boldsymbol{\theta}^{(i)})$ is given by:

$$\boldsymbol{J}_{\boldsymbol{\mathcal{L}}}(\boldsymbol{\theta}^{(i)}) = \frac{\partial \mathcal{L}(f(\boldsymbol{x}), \mathcal{G}_\Theta(\boldsymbol{x}))}{\partial \boldsymbol{\theta}^{(i)}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^{(n)}} \frac{\partial \boldsymbol{z}^{(n)}}{\partial \boldsymbol{\theta}^{(i)}} \tag{2.42}$$

where (see [74]):

$$\frac{\partial \boldsymbol{z}^{(j)}}{\partial \boldsymbol{\theta}^{(i)}} = \frac{\partial \boldsymbol{z}^{(j)}}{\partial \boldsymbol{z}^{(i)}} \frac{\partial \boldsymbol{z}^{(i)}}{\partial \boldsymbol{\theta}^{(i)}} \quad \text{and} \quad \frac{\partial \boldsymbol{z}^{(j)}}{\partial \boldsymbol{z}^{(i)}} = \sum_{k \in \text{Ch}(f^{(i)})} \frac{\partial \boldsymbol{z}^{(j)}}{\partial \boldsymbol{z}^{(k)}} \frac{\partial \boldsymbol{z}^{(k)}}{\partial \boldsymbol{z}^{(i)}} \tag{2.43}$$

and $\text{Ch}(f^{(i)})$ is the set of indexes of the child nodes of $f^{(i)}$. This is for one input and one output layer, but can be easily extended to multiple ones.

It can be seen that to obtain the gradients with respect to the parameters $\boldsymbol{\theta}^{(i)}$ of a layer $f^{(i)}$, the gradients of the layers along the paths between $f^{(i)}$ and the outputs are used. The error is propagated back through these layers to $f^{(i)}$, which is how backpropagation gets its name.

While backpropagation allows efficient computation of gradients for a given input/label pair, obtaining these gradients is only part of solving the optimization problem stated in Eq. 2.41. Computing the gradients for a single update of Θ with respect to the objective function $\mathcal{O}$ requires calculating the gradients for all ($\mathbf{x}^{(i)}$,$\mathbf{y}^{(i)}$) pairs in the training dataset, which is computationally expensive for large datasets. So instead of performing the update for the gradients computed over the whole dataset, Stochastic gradient descent (SGD) [91, 92] only

uses a single sample to approximate the gradient. This approach can lead to high variance in the approximated gradient, so updates are usually performed based on mini-batches of n samples instead [93]. These mini-batches are created by randomly dividing the dataset. This can be described using the following pseudocode:

Algorithm 2.1 Mini-batch gradient descent

Require: $\boldsymbol{\Theta}$: Start parameters
Require: η: Learning rate
Require: n: Batch size
Require: N_{epochs}: Number of training epochs
Require: $\mathcal{X}_{\text{train}} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) | i \in \{1, \ldots, N\}\}$: Training data
for N_{epochs} **do**
  Shuffle $\mathcal{X}_{\text{train}}$
  $t \leftarrow 1$ (Batch counter)
  while $t \cdot n \leq N$ **do**
    $\hat{\boldsymbol{g}} \leftarrow \frac{1}{n} \nabla_\Theta \sum_{i=(t-1)n+1}^{t \cdot n} \mathcal{L}(\mathbf{y}^{(i)}, \mathcal{G}_\Theta(\mathbf{x}^{(i)}))$ (Approximate gradients)
    $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - \eta \hat{\boldsymbol{g}}$ (Update parameters)
    $t \leftarrow t + 1$
  end while
end for

There are several variants of the Mini-batch gradient descent. One of the most famous ones is Adam [94], which is adapted for large datasets and/or high-dimensional parameter space and can also deal with sparse gradients and on-stationary objectives. For this reason Adam was used throughout this work as the optimizer for neural networks.

While the performance of $\mathcal{G}_\Theta$ on $\mathcal{X}_{\text{train}}$ is a good measure for how good $\mathcal{G}_\Theta$ approximates f, this measure was the target for optimizing $\mathcal{G}_\Theta$. Every measure which becomes a target becomes a bad measure [95]. It is possible that $\mathcal{G}_\Theta$ fits f only for values of $\mathbf{x}^{(i)}$ in $\mathcal{X}_{\text{train}}$, but fails to generalize beyond that. This can be addressed by introducing two more datasets: a validation dataset $\mathcal{X}_{\text{val}}$ and a test dataset $\mathcal{X}_{\text{test}}$. The validation dataset is used to tune model parameters and assure generalization, while the purpose of the test set is to evaluate the final model on data that was not seen during training. Each of these datasets has to be of adequate size to represent f. Although the validation and test datasets can be much smaller than the training dataset, training a neural network typically requires tens of thousands of samples. Obtaining the labels $\mathbf{y}^{(i)} = f(\mathbf{x}^{(i)})$ for all these samples can be challenging. Depending on the problem, manual labeling by humans may be necessary, which is common in computer vision. In

engineering, collecting such a large number of experimental samples is often not feasible. Numerical simulations, on the other hand, are an good alternative for generating labels. The labeling process can be automated, and most simulations are deterministic, resulting in data with a low amount of noise. With numerical simulations, the applicability of the results to real-world scenarios usually depends on the quality of the simulation rather than the machine learning process itself. In this work, machine learning models are generally trained on data from numerical simulations, with experimental data used for validation of the simulations.

For example, for metamaterials, a typical input might be the parameters of the unit cell, with the output being a resulting mechanical property, such as Poisson's ratio. To learn this mapping, a supervised model is provided with example input data and the corresponding predictions, known as labels.

Convolutional Neural Networks

So far, the neural networks discussed here have focused on processing vector-valued data. However, there are cases where a vector is not the most optimal choice to represent a given sample. The best example of data that requires a representation other than vectors is images. Images are generally high-dimensional, such that a neural network made up of fully connected layers would have far in too many parameters to effectively generalize beyond the training data [96]. Fortunately, images also have the convenient properties that their statistics are stationary and that dependencies between pixels are only local [97], which means that an image is hierarchically composed of low-level features, which can appear anywhere in the image and are only correlated with their surrounding features at the same level. This spatial hierarchy aligns well with the hierarchical nature of multiple stacked layers in a neural network, except that the layers which extract the features only need to consider the local portion of the next lower-level features. Such a function, that puts zero weight on features outside of a certain interval is called *a kernel*. Such a kernel can be used on different regions by essentially sliding it over the image, resulting in a significant reduction in parameters compared to fully connected networks. This is essentially a convolution between the kernel and the image, which is the principle behind convolutional layers [98]:

Definition 2.5 2D Convolutional layer

A *2D convolutional layer* is a function $f_C : \mathbb{R}^{n_1 \times n_2 \times n_3} \to \mathbb{R}^{n_1 \times n_2 \times n_C}$, that is given through the kernels $\mathcal{R}^k \in \mathbb{R}^{m_1, m_2, n_3}$ and biases $b_k \in \mathbb{R}$, $k \in \{1, \ldots, n_C\}$:

$$f_C(\mathbf{x}; \boldsymbol{\theta})_{i,j,k} = g((\mathcal{R}^k * \mathbf{x})_{i,j} + b_k) = g(b_k + \sum_{l=0}^{m_1-1} \sum_{m=0}^{m_2-1} \sum_{n=0}^{n_3-1} \mathrm{x}_{i-l,j-m,n} \mathcal{R}^k{}_{l,m,n}) \quad (2.44)$$

where $[m_1, m_2]$ are the dimensions of the kernel, n_3 is the number of channels in the input image, n_C the number of channels in the output image and g the activation function.

The output of a convolutional layer is a 2D map that indicates where the features extracted by the different kernels appear [71]. This map is equivariant to translation, meaning that if the input is shifted in the x or y direction, the same shift will be reflected in the output. When shifting a kernel over an image in this manner, the resulting output is usually smaller in size than the input. To obtain an output with the same first two dimensions as the input, it is necessary to pad the image with zeros for indices where $i - l < 0$ or $i - m < 0$, a process known as *zero padding*. If no padding is applied, the output has smaller dimensions ($\mathbb{R}^{n_1 - m_1 + 1 \times n_2 - m_2 + 1 \times n_C}$), this is called *valid padding*. Any network that uses at least one convolutional layer is called a *Convolutional Neural Network*. In these networks, it is common to first use convolutional layers for feature extraction, than convert the 2D feature map to a vector and continue with fully connected layers.

2.3.2 Inverse and Generative Models

Most datasets in deep learning can be interpreted such that each example in the dataset represents a state, with labels indicating properties inferred from that state. For instance, while a photograph is a recording of photons captured by the camera sensor, we can infer from it that the image depicts a cat. Determining these properties for a given state is called the *forward problem*. The other way around, finding a state that exhibits specified properties is called the *inverse problem*. This is the same as conditional generation, where data is generated that meets specified conditions or properties. Inverse problems have a tendency to be ill-posed, making them difficult to solve [99].

Definition 2.6 Well-posed and ill-posed inverse problems

A inverse problem is called *well-posed* if [100]:

- A solution to the problem exists
- The solution is unique
- The solution depends continuously on the input

Any inverse problem that is not well posed is called *ill-posed*.

Training a neural network to solve a well-posed inverse problem is quite straightforward: it is possible to switch the samples with the labels and treat it as a supervised learning task. Solving ill-posed problems with neural networks is more complex. While it can be safely assumed that there is a range of inputs for which a solution exists (there is a dataset after all), it is more problematic that often multiple solutions exist. This is because labels typically contain only a fraction of the original information. For example, there are countless images that fit the label "cat". To address this, conditional generative models introduce some "noise" as additional input, based on which one of the possible solutions is selected by the model. This section covers several of these models for inverse and generative design, namely: 1) Tandem Neural Networks; 2) generative models—specifically Variational Autoencoders, Generative Adversarial Networks, and Diffusion Models; and 3) approaches for learning constraints within generative models.

There is a sharp distinction in how some of these models approach an inverse design problem. While for the Tandem Neural Network the idea is to start with the properties, find a structure that fulfills them, and then add a mechanism to discover different structures with those properties, for generative models, it is the other way around. Generative models are foremost concerned with creating valid data without considering specific properties and have to be adjusted to include properties for the inverse design.

Generative models view any dataset as a collection of samples drawn from an underlying probability distribution $P(\mathbf{x})$ over the space of all possible inputs—for example, images of cats are drawn from a different distribution than images of dogs within the image domain. The goal of generative models is to o create new, valid data points by learning to sample from this distribution. Because these distributions can be complex and are only indirectly observed through the data sampled from them, it is often easier to map the data—and its distribution—to a simpler one in another space called the latent space. This latent space is usually lower-dimensional than the original space, but it can also be of equal or higher dimensionality. Suitable distributions in latent space include multivariate normal or uniform distributions. This approach allows for the generation of valid data samples, but has to be adjusted to take specific target properties into account. In

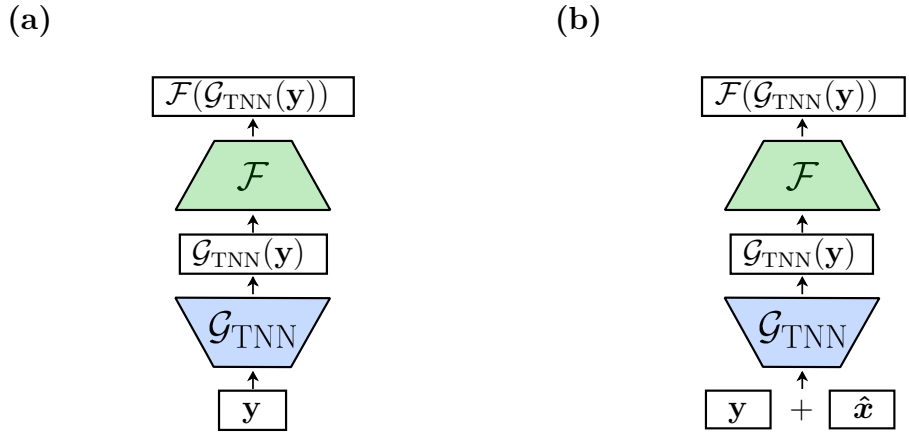

Figure 2.4: Schematic of a Tandem Neural Network (TNN) with a guide **(a)** and without a guide **(b)**. The TNN utilizes a pretrained prediction network $\mathcal{F}$ to establish a mapping between the target properties $\mathbf{y}$ and the properties of the sample generated by $\mathcal{G}_{\text{TNN}}$. In the guided variant, a reference $\hat{\boldsymbol{x}}$ is incorporated as a target for the generated sample.

order to generate samples $\mathbf{x}^{(i)}$ with specific properties $\mathbf{y}^{(i)}$, the sampling is done from the conditional probability distributions $P(\mathbf{x}|\mathbf{y}^{(i)})$.

Tandem Neural Networks

Tandem Neural Networks (TNNs) are a class of neural network architectures specifically devised for inverse design tasks, originally developed in the context of electromagnetics and photonics [101, 102]. As depicted in Figure 2.4a the TNN architecture consists of two parts: the inverse or generative model $\mathcal{G}_{\text{TNN}}$ and the forward model $\mathcal{F}$. The generative model $\mathcal{G}_{\text{TNN}}$ is a neural network that receives specified properties $\mathbf{y}^{(i)}$ as input and outputs a state $\mathbf{x}^{(i)}$, which, in the context of this work, is usually a metamaterial structure. Due to the existence of multiple structures with identical properties, direct training of $\mathcal{G}_{\text{TNN}}$ using $(\mathbf{y}^{(i)}, \mathbf{x}^{(i)})$ pairs is not feasible. The different possible $\mathbf{x}^{(i)}$ values would pull the weights of $\mathcal{G}_{\text{TNN}}$ in different directions, making training ineffective. To circumvent this, $\mathcal{F}$ is used to predict the properties of the generated structure, making it possible to create a one-to-one mapping between target and predicted properties. $\mathcal{F}$ can be pretrained on $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ pairs, and its weights fixed during the training of $\mathcal{G}_{\text{TNN}}$. By using $\mathcal{F}$ for property prediction, $\mathcal{G}_{\text{TNN}}$ can be trained without requiring additional datasets. To ensure valid outputs, $\mathcal{G}_{\text{TNN}}$ is constrained—usually by a sigmoid activation function—such that every possible output corresponds to a valid design. During training, $\mathcal{G}_{\text{TNN}}$ generates samples $\mathbf{x}^{(i)}$ based on $\mathbf{y}^{(i)}$, for which $\mathcal{F}$ is then used to predict their properties. $\mathcal{G}_{\text{TNN}}$ is trained such that the properties of $\mathbf{x}^{(i)}$ (as predicted by $\mathcal{F}$) match the target properties. This allows $\mathbf{y}^{(i)}$ to act both as

input and as the target output for $F(G(\mathbf{y}^{(i)}))$. Therefore, $\mathbf{y}^{(i)}$ values for training $\mathcal{G}_{\text{TNN}}$ can be randomly selected from the range used to train $\mathcal{F}$. Usually, the Mean Squared Error (MSE) is used as the loss function for this:

$$\mathcal{L}(\mathbf{y}, \mathcal{G}_{\text{TNN}}, \mathcal{F}) = \sum_i (y_i - \mathcal{F}(\mathcal{G}_{\text{TNN}}(y_i)))^2 \tag{2.45}$$

It would also be possible to use the FEA simulation, which was used to generate the $\mathbf{y}^{(i)}$ values, directly in place of $\mathcal{F}$. However, using $\mathcal{F}$ instead has two advantages: First, once trained, $\mathcal{F}$ can be more efficient evaluated that running a simulation. Second, obtaining the gradients required for training $\mathcal{G}_{\text{TNN}}$ is less computationally expensive using $\mathcal{F}$, as backpropagation can be used.

Using $\mathcal{F}$ in this manner allows to train $\mathcal{G}_{\text{TNN}}$ to produce structures exhibiting the target properties. However, a one on one mapping between $\mathbf{y}^{(i)}$ and $F(G(\mathbf{y}^{(i)}))$ results in only a single possible solution $\mathbf{x}^{(i)}$ for each $\mathbf{y}^{(i)}$, rather than multiple potential solutions. To address this, one approach is to introduce stochastic sampling in some layers of the model. Bastek *et al.* [103] used this approach to sample from a range of base structures, whose parameters were then adapted to produce the target properties. While this approach produces multiple solutions, its capabilities are limited by the number and complexity of the base structures. We found in the context of this work, that if the $\mathbf{x}^{(i)}$ values are drawn from a compact interval $[0, 1]^n$ with an appropriate distance metric, another approach leads to good results [104]. To select which possible structure is generated by $\mathcal{G}_{\text{TNN}}$, an additional input $\hat{\boldsymbol{x}}^{(i)}$ is provided (see Figure 2.4b). This input is a target structure which the generated structure should resemble. To compare these two structures, usually the distance metric between the structures can be used. In most cases this is also the MSE:

$$\mathcal{L}(\mathbf{x}, \mathbf{y}, \mathcal{G}, \mathcal{F}) = \sum_i (y_i - \mathcal{F}(\mathcal{G}_{\text{TNN}}(y_i, \hat{x}_i)))^2 + \alpha \sum_i (\hat{x}_i - \mathcal{G}_{\text{TNN}}(y_i, \hat{x}_i))^2 \tag{2.46}$$

This creates a trade-off between the two objectives, which is controlled by the parameter α. While the guide can express a preference for a certain generated structure, it can also be selected at random if no such preference exists. Multiple solutions can be generated by choosing different guides. Note that for training, $\mathbf{y}^{(i)}$ and $\hat{\boldsymbol{x}}^{(i)}$ can be chosen independently at random, as the pre-trained model $\mathcal{F}$ is still used to create a mapping between the target properties and the properties of the generated structure.

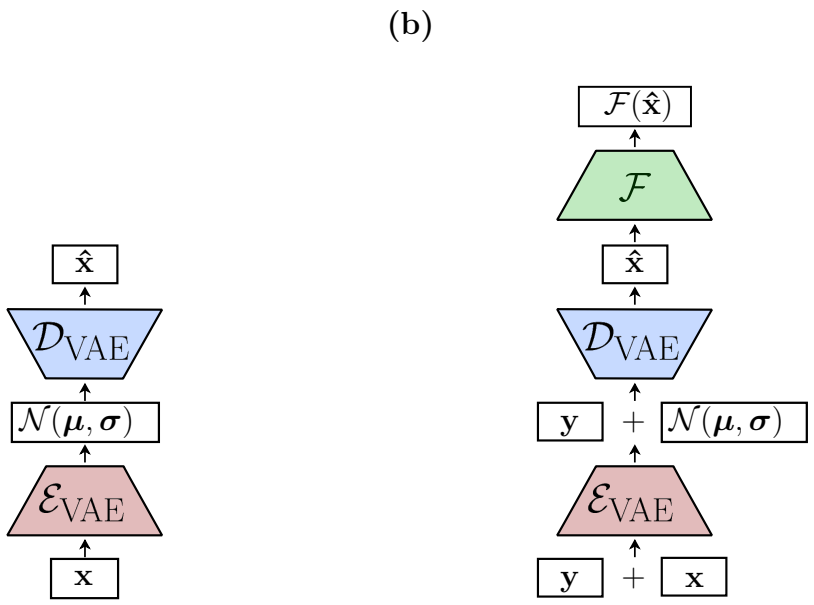

Figure 2.5: Schematic of a Variational Autoencoder (VAE) in its standard form **(a)** and with conditioning on properties $\mathbf{y}$ **(b)**. The encoder $\mathcal{E}_{\text{VAE}}$ maps the input data sample $\mathbf{x}$ to the mean and variance of a latent space distribution. A latent vector is sampled from this distribution and passed to the decoder, which attempts to reconstruct $\mathbf{x}$ as $\hat{\boldsymbol{x}}$. In the conditioned variant, a pretrained prediction network $\mathcal{F}$ is used to enforce a correlation between the reconstructed sample $\hat{\boldsymbol{x}}$ and the target properties $\mathbf{y}$.

Variational Autoencoders

First introduced by Kingma and Welling [105], the Variational Autoencoder (VAE) is a generative extension of the traditional autoencoder architecture. Autoencoders are neural networks designed to compress input data into a lower-dimensional latent space and reconstruct it from this representation [106]. They consist of two main components (see Figure 2.5a): the encoder $\mathcal{E}_{\text{VAE}}$, which maps the input data $\boldsymbol{x}$ to a latent representation, and the decoder $\mathcal{D}_{\text{VAE}}$, which attempts to reconstruct the original data from this representation. This reconstruction is denoted as $\hat{\boldsymbol{x}}$. The goal of this setup is to find a low-dimensional encoding that preserves most of the information in $\boldsymbol{x}$. An autoencoder can be trained on this by minimizing the error between the original data and its reconstruction, which is called the reconstruction loss $\mathcal{L}_R$. In the original paper, Binary Cross-Entropy (BCE) loss was used for the MNIST dataset, where pixels are supposed to be either white or black, while the MSE was used for continuous problems [105]:

$$\mathcal{L}_{\text{BCE}}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \frac{1}{m} \sum_{i=0}^{m} \hat{x}_i \cdot \log x_i + (1-\hat{x}_i) \cdot \log(1-x_i) \tag{2.47}$$

$$\mathcal{L}_{\text{MSE}}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \frac{1}{m} \sum_{i=0}^{m} (x_i - \hat{x}_i)^2 \tag{2.48}$$

This approach allows dimensionality reduction by choosing a lower-dimensional latent space [107]. Besides this, autoencoders can be used for tasks such as denoising or dimensionality reduction, as the latent representation preserves only the most important information [108, 109]. The core idea behind the VAE is straightforward: if every sample can be fully reconstructed from its representation in the latent space, it might be simpler to generate these representations rather than the data itself. To accomplish this, it is necessary to control the probability distribution of the embeddings in the latent space, as data is generated by sampling from its distribution. Essentially, a mapping between the data distribution $P(x)$ and a known distribution in the latent space $P(z)$ is created by learning two conditional distributions: $p_{\Theta_\mathcal{G}}(\boldsymbol{x}|\boldsymbol{z})$ and $p_{\Theta_\mathcal{E}}(\boldsymbol{z}|\boldsymbol{x})$. The VAE achieves this control through a regularization term in the loss function, that constrains the latent distribution to a predefined prior—usually, a multivariate normal distribution with zero mean and unit variance. For this, the regularization term has to measure the difference between difference between the learned latent distribution and the prior, which is difficult as the distribution in the latent space is only known through samples of it. To circumvent this, the encoder of the VAE $\mathcal{E}_{\text{VAE}}$ maps to mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}$ of a normal distribution in the latent space. Using this, the regularization term can then be computed using the Kullback–Leibler (KL) divergence between two normal distributions:

$$\mathcal{L}_{\text{VAE}}(\boldsymbol{x}, \hat{\boldsymbol{x}}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \underbrace{\mathcal{L}_R(\boldsymbol{x}, \hat{\boldsymbol{x}})^2}_{\text{reconstruction loss}} + \beta \underbrace{D_{\text{KL}}\big(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}), \mathcal{N}(\mathbf{0}, \mathbf{I})\big)}_{\text{regularization loss}} \tag{2.49}$$

where β is a hyperparameter introduced later to the VAE formulation by Higgins et al. [110] to control the trade-off between reconstruction loss and regularization loss. The reconstruction loss of the VAE is similar to that of the standard autoencoder, the only difference is that the latent representation is sampled from the distribution given by the encoder rather than being directly computed. Using this combined loss function, VAEs can be trained similar to feedforward neural networks:

Algorithm 2.2 Mini-batch VAE training

Require: $\boldsymbol{\Theta}_{\mathcal{E}}$: Start parameters of $\mathcal{E}_{\text{VAE}}$
Require: $\boldsymbol{\Theta}_{\mathcal{G}}$: Start parameters of $\mathcal{G}_{\text{VAE}}$
Require: η: Learning rate
Require: n: Batch size
Require: N_{epochs}: Number of training epochs
Require: $\mathcal{X}_{\text{train}} = \{\mathbf{x}^{(i)} | i \in \{1, \ldots, N\}\}$: Training data
$\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta}_{\mathcal{E}} \cup \boldsymbol{\Theta}_{\mathcal{G}}$
for N_{epochs} **do**
  Shuffle $\mathcal{X}_{\text{train}}$
  $t \leftarrow 1$ (Batch counter)
  while $t \cdot n \leq N$ **do**
    for i **in** $\{(t-1)n+1, \ldots, t \cdot n\}$ **do**
      $\boldsymbol{\epsilon} \leftarrow$ Random samples from $\mathcal{N}(\mathbf{0}, \mathbf{I})$
      $\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{(i)} \leftarrow \mathcal{E}_{\text{VAE}}(\mathbf{x}^{(i)})$
      $\hat{\boldsymbol{x}}^{(i)} \leftarrow \mathcal{D}_{\text{VAE}}(\boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\epsilon})$
    end for
    $\hat{\boldsymbol{g}} \leftarrow \frac{1}{n} \nabla_{\Theta} \sum_{i=(t-1)n+1}^{t \cdot n} \mathcal{L}_{\text{VAE}}(\mathbf{x}^{(i)}, \hat{\boldsymbol{x}}^{(i)}, \boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{(i)})$ (Approximate gradients)
    $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - \eta \hat{\boldsymbol{g}}$ (Update parameters)
    $t \leftarrow t + 1$
  end while
end for

Reparameterization of $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$ as $\boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, creates a differentiable estimator that allows backpropagation through the stochastic variables. While the latent representations are sampled using this reparameterized formulation during training, it is not needed during generation, as latent representations are sampled from the prior distribution there. In this setup, the decoder of the VAE also doubles as the generative model $\mathcal{G}_{\text{VAE}}$, as it turns random samples from the prior distribution into new data essentially sampled from $P(\boldsymbol{x})$.

While this approach allows to create new samples, it provides no control over the properties of those samples. To enable that the VAE has to be modified such that new data is sampled from $P(\boldsymbol{x}|\boldsymbol{y})$ rather than $P(\boldsymbol{x})$. As a VAE does not sample directly from $P(\boldsymbol{x})$, this has to be broken down into two parts: First, conditioning the decoder to generate by sampling from $p_{\Theta_{\mathcal{G}}}(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{y})$ and second conditioning the encoder on the properties $q_{\Theta_{\mathcal{E}}}(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y})$. This is the core idea behind the Conditional Variational Autoencoder (CVAE) proposed by Sohn *et al.* [111]. Conditioning the encoder is necessary as the distribution of the samples in the latent space $p_{\Theta_{\mathcal{E}}}(\boldsymbol{z}|\boldsymbol{y})$ is modulated by the properties $\boldsymbol{y}$, however this can be relaxed back

to $p_{\Theta_\mathcal{E}}(\boldsymbol{z})$ to force the neural network to make the latent variables statistically independent of input variables [112]. Using this relaxed version changing from the VAE to the CVAE architectures requires only to add the properties $\boldsymbol{y}$ as additional input to both the encoder $\mathcal{E}_{\text{VAE}}$ and the decoder/generator $\mathcal{G}_{\text{VAE}}$. This approach incorporates the properties only implicitly, as the CVAE picks up the dependencies between the properties and the generated inputs during training, as the training data only consists of samples with matching properties. Because they are only incorporated implicitly, the properties might be ignored during training, as other effects on the loss function might be more pronounced. To gain more control over this, it is necessary to explicitly include the properties in the loss function. This approach depends on the ability to compute the properties of the generated samples as well as the gradients regarding the properties. Both can be computed using a forward neural network $\mathcal{F}$ pre-trained to predict the properties for a given sample (see Figure 2.5b), allowing to include the difference between target properties and the properties of the generated samples in the loss function [113]:

$$\mathcal{L}_{\text{CVAE}}(\boldsymbol{x}, \hat{\boldsymbol{x}}, \boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{y}) = \underbrace{\mathcal{L}_R(\boldsymbol{x}, \hat{\boldsymbol{x}})^2}_{\text{reconstruction loss}} + \beta \underbrace{D_{\text{KL}}\big(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}), \mathcal{N}(\mathbf{0}, \mathbf{I})\big)}_{\text{regularization loss}} + \alpha \underbrace{\mathcal{L}_{\text{MSE}}(F(\hat{\boldsymbol{x}}), \boldsymbol{y})}_{\text{property loss}} \tag{2.50}$$

Depending on its task, $\mathcal{F}$ is either called a Auxiliary Classifier (AC) or Auxiliary Regressor (AC). This creates a architecture that is in parts similar to the Tandem Neural Network (TNN) for generation. Instead of receiving the original input, the generator receives its encoding in the latent space. This is necessary since the assumption made for the TNN that data comes from a compact interval may not hold for the data processed by a VAE. Due to these similarities, the training algorithm for the CVAE with a AC/AR is a mix of the algorithms for the TNN and VAE:

Algorithm 2.3 Mini-batch training of the CVAE with AC/AR

Require: $\boldsymbol{\Theta}_{\mathcal{F}}$: Start parameters of $\mathcal{F}$
Require: $\boldsymbol{\Theta}_{\mathcal{E}}$: Start parameters of $\mathcal{E}_{\text{VAE}}$
Require: $\boldsymbol{\Theta}_{\mathcal{G}}$: Start parameters of $\mathcal{G}_{\text{VAE}}$
Require: η: Learning rate
Require: n: Batch size
Require: N_{epochs}: Number of training epochs
Require: $\mathcal{X}_{\text{train}} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) | i \in \{1, \dots, N\}\}$: Training data

for N_{epochs} **do**
  Shuffle $\mathcal{X}_{\text{train}}$
  $t \leftarrow 1$ (Batch counter)
  while $t \cdot n \leq N$ **do**
    $\hat{\boldsymbol{g}} \leftarrow \frac{1}{n} \nabla_{\Theta} \sum_{i=(t-1)n+1}^{t \cdot n} \mathcal{L}(\mathbf{y}^{(i)}, \mathcal{F}(\mathbf{x}^{(i)}))$ (Approximate gradients)
    $\boldsymbol{\Theta}_{\mathcal{F}} \leftarrow \boldsymbol{\Theta}_{\mathcal{F}} - \eta \hat{\boldsymbol{g}}$ (Update parameters)
    $t \leftarrow t + 1$
  end while
end for

$\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta}_{\mathcal{E}} \cup \boldsymbol{\Theta}_{\mathcal{G}}$
for N_{epochs} **do**
  Shuffle $\mathcal{X}_{\text{train}}$
  $t \leftarrow 1$ (Batch counter)
  while $t \cdot n \leq N$ **do**
    for i **in** $\{(t-1)n + 1, \dots, t \cdot n\}$ **do**
      $\boldsymbol{\epsilon} \leftarrow$ Random samples from $\mathcal{N}(\mathbf{0}, \mathbf{I})$
      $\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{(i)} \leftarrow \mathcal{E}_{\text{VAE}}(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$
      $\hat{\boldsymbol{x}}_i \leftarrow \mathcal{G}_{\text{VAE}}(\boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\epsilon}, \mathbf{y}^{(i)})$
    end for
    $\hat{\boldsymbol{g}} \leftarrow \frac{1}{n} \nabla_{\Theta} \sum_{i=(t-1)n+1}^{t \cdot n} \mathcal{L}_{\text{CVAE}}(\mathbf{x}^{(i)}, \hat{\boldsymbol{x}}^{(i)}, \boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{(i)}, \mathbf{y}^{(i)})$ (Approx. gradients)
    $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - \eta \hat{\boldsymbol{g}}$ (Update parameters)
    $t \leftarrow t + 1$
  end while
end for

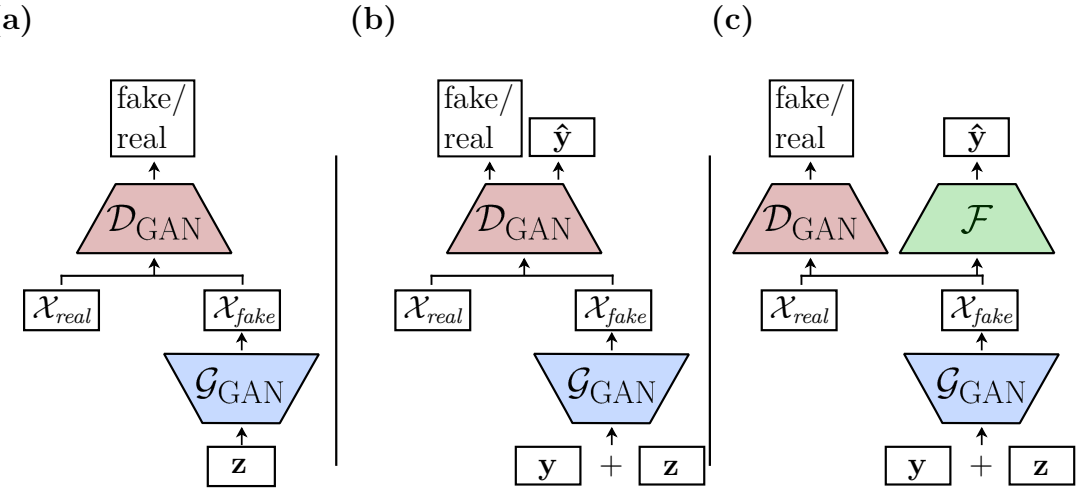

Figure 2.6: Schematics of a Generative Adversarial Network (GAN) in its standard form **(a)**, the Conditional GAN (CGAN) **(b)** and the Auxiliary Classifier GAN (ACGAN) variant **(c)**. In a standard GAN, the discriminator $\mathcal{D}_{\text{GAN}}$ aims to distinguish between real samples from the dataset and synthetic data produced by the generator $\mathcal{G}_{\text{GAN}}$, while $\mathcal{G}_{\text{GAN}}$ attempts to generate samples that can fool $\mathcal{D}_{\text{GAN}}$. In the ACGAN variant, the discriminator additionally predicts the properties of the samples, and the generator is conditioned to produce data that matches the target properties $\mathbf{y}$. In the VAR+GAN setup, property prediction is decoupled and handled by a separate model $\mathcal{F}$.

Generative Adversarial Networks

First introduced in 2014 by Goodfellow *et al.* [114], Generative Adversarial Networks (GANs) provide a framework for training generative machine learning models through an adversarial process. In this framework, shown in Figure 2.6a, a generative model, $\mathcal{G}_{\text{GAN}}$, which converts noise from the latent space into data, is trained alongside a discriminative model, $\mathcal{D}_{\text{GAN}}$, in a two-player game. The goal of this game is to teach $\mathcal{G}_{\text{GAN}}$ to capture the data distribution $P(\boldsymbol{x})$. In this process, $\mathcal{D}_{\text{GAN}}$ is trained to predict the probability that a sample comes from the training data rather than from $\mathcal{G}_{\text{GAN}}$, while $\mathcal{G}_{\text{GAN}}$ is simply trained to fool $\mathcal{D}_{\text{GAN}}$:

$$\mathcal{L}_{\mathcal{D}}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = -\log \mathcal{D}_{\text{GAN}}(\boldsymbol{x}) - \log\left(1 - \mathcal{D}_{\text{GAN}}(\hat{\boldsymbol{x}})\right) \tag{2.51}$$

$$\mathcal{L}_{\mathcal{G}}(\hat{\boldsymbol{x}}) = \log\left(1 - \mathcal{D}_{\text{GAN}}(\hat{\boldsymbol{x}})\right) \tag{2.52}$$

$$\hat{\boldsymbol{x}} = \mathcal{G}_{\text{GAN}}(\boldsymbol{z}) \tag{2.53}$$

where $\boldsymbol{x} \in \mathcal{X}_{\text{train}}$ is a sample from the training data, while $\boldsymbol{z}$ is a sample drawn from the noise distribution. This two-player game reaches an Nash equilibrium

when the distribution of the samples generated by $\mathcal{G}_{\text{GAN}}$ matches the original data distribution. Goodfellow *et al.* [114] have further shown that this process is equivalent to minimizing the Jensen-Shannon (JS) divergence between the two distributions. In practice, this two-player game is implemented by alternating the training of the generator and discriminator on individual batches:

Algorithm 2.4 Mini-batch GAN training

Require: $\boldsymbol{\Theta}_{\mathcal{G}}$: Start parameters of $\mathcal{G}_{\text{GAN}}$
Require: $\boldsymbol{\Theta}_{\mathcal{D}}$: Start parameters of $\mathcal{D}_{\text{GAN}}$
Require: η: Learning rate
Require: n: Batch size
Require: N_{epochs}: Number of training epochs
Require: $\mathcal{X}_{\text{real}} = \{\mathbf{x}^{(i)} | i \in \{1, \ldots, N\}\}$: Training data
for N_{epochs} **do**
  Shuffle $\mathcal{X}_{\text{real}}$
  $t \leftarrow 1$ (Batch counter)
  while $t \cdot n \leq N$ **do**
    $\{\mathbf{z}^{(i)} | i \in \{(t-1)n+1, \ldots, t \cdot n\}\} \leftarrow$ Random samples from $\mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\hat{\boldsymbol{g}}_{\mathcal{D}} \leftarrow \frac{1}{n} \nabla_{\Theta} \sum_{i=(t-1)n+1}^{t \cdot n} \mathcal{L}_{\mathcal{D}}(\mathbf{x}^{(i)}, \mathcal{G}_{\text{GAN}}(\mathbf{z}^{(i)}))$ (Approximate gradients $\mathcal{D}_{\text{GAN}}$)
    $\hat{\boldsymbol{g}}_{\mathcal{G}} \leftarrow \frac{1}{n} \nabla_{\Theta} \sum_{i=(t-1)n+1}^{t \cdot n} \mathcal{L}_{\mathcal{G}}(\mathcal{G}_{\text{GAN}}(\mathbf{z}^{(i)}))$ (Approximate gradients $\mathcal{G}_{\text{GAN}}$)
    $\boldsymbol{\Theta}_{\mathcal{D}} \leftarrow \boldsymbol{\Theta}_{\mathcal{D}} - \eta \hat{\boldsymbol{g}}_{\mathcal{D}}$ (Update parameters $\mathcal{D}_{\text{GAN}}$)
    $\boldsymbol{\Theta}_{\mathcal{G}} \leftarrow \boldsymbol{\Theta}_{\mathcal{G}} - \eta \hat{\boldsymbol{g}}_{\mathcal{G}}$ (Update parameters $\mathcal{G}_{\text{GAN}}$)
    $t \leftarrow t + 1$
  end while
end for

While this approach may seem straightforward, finding Nash equilibria is notoriously challenging [115]. Depending on the problem, it is especially common for gradient descent to enters a stable orbit rather than converging to the equilibrium [71]. Other problems can arise from one of the two neural networks overpowering the other. A common example of this is that in early learning, when $\mathcal{G}_{\text{GAN}}$ is poor, all of the samples generated by it are rejected by $\mathcal{D}_{\text{GAN}}$ causing $\mathcal{L}_{\mathcal{G}}$ to saturate. This can be avoided by modifying $\mathcal{L}_{\mathcal{G}}$, which does not saturate in this case, while preserving the equilibrium point [116]:

$$\mathcal{L}_{\mathcal{G}}(\hat{\boldsymbol{x}}) = -\log\left(\mathcal{D}_{\text{GAN}}(\hat{\boldsymbol{x}})\right) \tag{2.54}$$

Another problem is that, in the long term, $\mathcal{G}_{\text{GAN}}$ can overpower $\mathcal{D}_{\text{GAN}}$, resulting in the generation of only a very narrow range of images that consistently receive

a high rating from $\mathcal{D}_{\text{GAN}}$—a phenomenon known as mode collapse [117]. Consequently, due to the popularity of GANs, numerous solutions to this problem have been proposed. For example, Heusel *et al.* [118] showed that GAN convergence can be improved by using two separate, properly adjusted learning rates for $\mathcal{G}_{\text{GAN}}$ and $\mathcal{D}_{\text{GAN}}$. Other approaches include adding a decoder that reconstructs the noise from a generated image, allowing reconstruction loss to stabilize training [119, 120], as well as training the generator to match the expected value of features at an intermediate layer of the discriminator—a technique known as feature matching [115].

The most well known approach to stabilizing GAN training however is the *Wasserstein Generative Adversarial Network* (WGAN) [121]. The WGAN is a GAN variant that uses the Wasserstein distance instead of the JS divergence to measure the discrepancy between the distributions of the generated and training data. The JS divergence measures similarity between two probability distributions by comparing them locally. This process relies on the ratio between the two probabilities, which can become infinite if one of them is zero, preventing the computation of meaningful gradients. The Wasserstein distance, on the other hand, is based on the principle of optimal transport: a probability distribution is viewed as a distribution of mass, and the difference between distributions is measured as the minimal cost required to transport mass to make one distribution resemble the other. This distance has the benefit of always remaining finite, thereby allowing the computation of meaningful gradients. This adjustment requires reformulation of the objectives of $\mathcal{G}_{\text{GAN}}$ and $\mathcal{D}_{\text{GAN}}$:

$$\mathcal{L}_{\mathcal{D}_{\text{WGAN}}}(\boldsymbol{x}, \boldsymbol{z}) = -\mathcal{D}_{\text{GAN}}(\boldsymbol{x}) + \mathcal{D}_{\text{GAN}}(\mathcal{G}_{\text{GAN}}(\boldsymbol{z})) \tag{2.55}$$

$$\mathcal{L}_{\mathcal{G}_{\text{WGAN}}}(\boldsymbol{z}) = -\mathcal{D}_{\text{GAN}}(\mathcal{G}_{\text{GAN}}(\boldsymbol{z})) \tag{2.56}$$

While this objective allows for reliable gradient computation, it requires the function provided by $\mathcal{D}_{\text{GAN}}$ to be Lipschitz-continuous. In the standard WGAN formulation, this is enforced by clipping the weights to a fixed interval $[-c, c]$. However, this approach can introduce instability into the WGAN training process, which led Gulrajani et al. [122] to propose a WGAN variant that uses a penalty on large gradients in the loss function instead. This gradient penalty, denoted as *gp*, is computed using a randomly weighted average of a sample $\boldsymbol{x}$ from the original data and a sample $\tilde{\boldsymbol{x}}$ generated by $\mathcal{G}_{\text{GAN}}$:

$$\text{gp}(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = (\|\nabla_{\phi(\boldsymbol{x}, \tilde{\boldsymbol{x}})} \mathcal{D}_{\text{GAN}}(\phi(\boldsymbol{x}, \tilde{\boldsymbol{x}}))\|_2 - 1)^2 \tag{2.57}$$

$$\phi(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}} \tag{2.58}$$

$$\epsilon \sim U[0, 1] \tag{2.59}$$

With this gradient penalty, the loss function of $\mathcal{D}_{\text{GAN}}$ can be modified to enforce Lipschitz-continuity:

$$\mathcal{L}_{\mathcal{D}_{\text{WGAN-gp}}}(\boldsymbol{x}, \boldsymbol{z}) = \mathcal{D}_{\text{GAN}}(\boldsymbol{x}) - \mathcal{D}_{\text{GAN}}(\mathcal{G}_{\text{GAN}}(\boldsymbol{z})) + \lambda \text{gp}(\boldsymbol{x}, \mathcal{G}_{\text{GAN}}(\boldsymbol{z})) \tag{2.60}$$

Furthermore, to maintain a strong discriminator during training, $\mathcal{D}_{\text{GAN}}$ is trained on multiple batches (usually five) for each batch that $\mathcal{G}_{\text{GAN}}$ is trained on.

Algorithm 2.5 Mini-batch WAN-gp training

Require: $\boldsymbol{\Theta}_{\mathcal{G}}$: Start parameters of $\mathcal{G}_{\text{GAN}}$
Require: $\boldsymbol{\Theta}_{\mathcal{D}}$: Start parameters of $\mathcal{D}_{\text{GAN}}$
Require: η: Learning rate
Require: n: Batch size
Require: N_{epochs}: Number of training epochs
Require: $\mathcal{X}_{\text{real}} = \{\mathbf{x}^{(i)} | i \in \{1, \ldots, N\}\}$: Training data
Require: $n_{\mathcal{D}}$: Number of iterations of $\mathcal{D}_{\text{GAN}}$ per $\mathcal{G}_{\text{GAN}}$ iteration
Require: λ: The gradient penalty coefficient
for N_{epochs} **do**
  Shuffle $\mathcal{X}_{\text{real}}$
  $t \leftarrow 1$ (Batch counter)
  while $t \cdot n \leq N$ **do**
    for $j = \{1, \ldots, n_{\mathcal{D}}\}$ **do**
      $\{\mathbf{z}^{(i)} | i \in \{(t-1)n + 1, \ldots, t \cdot n\}\} \leftarrow$ Random samples from $\mathcal{N}(\mathbf{0}, \mathbf{I})$
      $\epsilon \leftarrow$ Sample from $U[0, 1]$
      $\tilde{\boldsymbol{x}} \leftarrow \mathcal{G}_{\text{GAN}}(\boldsymbol{z}^{(i)})$
      $\boldsymbol{\phi} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $\text{gp} \leftarrow (\|\nabla_{\boldsymbol{\phi}} \mathcal{D}_{\text{GAN}}(\boldsymbol{\phi})\|_2 - 1)^2$
      $\hat{\boldsymbol{g}}_{\mathcal{D}} \leftarrow \frac{1}{n} \nabla_{\Theta} \sum_{i=(t-1)n+1}^{t \cdot n} \mathcal{D}_{\text{GAN}}(\boldsymbol{x}^{(i)}) - \mathcal{D}_{\text{GAN}}(\tilde{\boldsymbol{x}}) + \lambda \text{gp}$
      $\boldsymbol{\Theta}_{\mathcal{D}} \leftarrow \boldsymbol{\Theta}_{\mathcal{D}} - \eta \hat{\boldsymbol{g}}_{\mathcal{D}}$ (Update parameters $\mathcal{D}_{\text{GAN}}$)
      $t \leftarrow t + 1$
    end for
    $\{\mathbf{z}^{(i)} | i \in \{1, \ldots, n\}\} \leftarrow$ Random samples from $\mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\hat{\boldsymbol{g}}_{\mathcal{G}} \leftarrow \frac{1}{n} \nabla_{\Theta} \sum_{i=1}^{n} -\mathcal{D}_{\text{GAN}}(\mathcal{G}_{\text{GAN}}(\boldsymbol{z}^{(i)}))$ (Approximate gradients $\mathcal{G}_{\text{GAN}}$)
    $\boldsymbol{\Theta}_{\mathcal{G}} \leftarrow \boldsymbol{\Theta}_{\mathcal{G}} - \eta \hat{\boldsymbol{g}}_{\mathcal{G}}$ (Update parameters $\mathcal{G}_{\text{GAN}}$)
  end while
end for

Similar to Variational Autoencoders, GANs can be conditioned to generate samples that exhibit specific properties by sampling from $P(\boldsymbol{x}|\boldsymbol{y})$ instead of $P(\boldsymbol{x})$. The simplest approach to achieve this is to provide both $\mathcal{G}_{\text{GAN}}$ and $\mathcal{D}_{\text{GAN}}$ with properties in addition to the sample. $\mathcal{G}_{\text{GAN}}$ is additionally provided with the target properties, while $\mathcal{D}_{\text{GAN}}$ receives either the corresponding properties for a sample from the original data or the target properties for a sample generated by $\mathcal{G}_{\text{GAN}}$: This is the idea behind Conditional Generative Adversarial Networks (CGANs) [123] (see Figure 2.6b). It allows $\mathcal{D}_{\text{GAN}}$ to capture the dependency of the data distribution on the associated properties, thereby forcing $\mathcal{G}_{\text{GAN}}$ to consider the target properties itself. For this to work, the distribution of the target properties during training must closely resemble the original distribution of the properties. Otherwise, $\mathcal{D}_{\text{GAN}}$ might simply learn to identify samples generated by $\mathcal{G}_{\text{GAN}}$ based on the properties alone. This method incorporates the properties implicitly, which means they might be overlooked by $\mathcal{D}_{\text{GAN}}$ if other effects on the loss function are more pronounced. The *Auxiliary Classifier GAN* (ACGAN) [124], shown in Figure 2.6c, includes the properties explicitly by requiring $\mathcal{D}_{\text{GAN}}$ to predict the properties instead of taking them as additional input. This is done by incorporating a Auxiliary Decoder (or Classifier/Regressor) Network into the discriminator, yielding a discriminator $\mathcal{D}_{\text{ACGAN}}$:

$$\mathcal{D}_{\text{ACGAN}}(\boldsymbol{x}) = \{\mathcal{D}_{\text{GAN}}(\boldsymbol{x}), y_{\mathcal{D}}(\boldsymbol{x})\} \tag{2.61}$$

where $y_{\mathcal{D}}(\boldsymbol{x})$ are the properties predicted by $\mathcal{D}_{\text{ACGAN}}$ for $\boldsymbol{x}$.

The decoding process ensures that $\mathcal{D}_{\text{ACGAN}}$ must preserve information about the properties in all layers. For this, $\mathcal{D}_{\text{ACGAN}}$ is trained simultaneously to predict the probability that a sample was generated by $\mathcal{G}_{\text{GAN}}$ and to predict the corresponding properties of the original data, $\boldsymbol{y}_d$, or the target properties, $\boldsymbol{y}_t$, for the samples generated by $\mathcal{G}_{\text{GAN}}$:

$$\mathcal{L}_{\mathcal{D}_{\text{ACGAN}}}(\boldsymbol{x}, \hat{\boldsymbol{x}}, \boldsymbol{y}_d, \boldsymbol{y}_t) = \mathcal{L}_{\mathcal{D}}(\boldsymbol{x}, \hat{\boldsymbol{x}}) + \alpha \Big[\mathcal{L}_{\text{MSE}}(\boldsymbol{y}_d, y_{\mathcal{D}}(\boldsymbol{x})) + \mathcal{L}_{\text{MSE}}\big(\boldsymbol{y}_t, y_{\mathcal{D}}(\hat{\boldsymbol{x}})\big) \Big] \tag{2.62}$$

where the generator also receives the target properties as input:

$$\hat{\boldsymbol{x}} = \mathcal{G}_{\text{GAN}}(\boldsymbol{z}, \boldsymbol{y}_t) \tag{2.63}$$

Incorporating property prediction in $\mathcal{D}_{\text{GAN}}$ not only forces it to consider the properties but has also been shown to enable more data-efficient training of $\mathcal{D}_{\text{GAN}}$

and a general improvement in sample quality [125]. However, if a pretrained forward network $\mathcal{F}$ is available it can be beneficial to run it in parallel to $\mathcal{D}_{\text{GAN}}$ instead of incorporating the property prediction. When run in parallel to $\mathcal{D}_{\text{GAN}}$, $\mathcal{F}$ is called a Versatile Auxiliary Regressor/Classifier (VAR/VAC) [126] since it can be independently used in different scenarios. This framework is shown in 2.6c. For a VAR/VAC-GAN only the generator loss function changes and if a pretrained $\mathcal{F}$ is used, no labels are needed:

$$\mathcal{L}_{\mathcal{G}_{\text{VAR-GAN}}}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \mathcal{L}_{\mathcal{G}}(\hat{\boldsymbol{x}}) + \alpha \mathcal{L}_{\text{MSE}}\big(\boldsymbol{y}_t, \mathcal{F}(\hat{\boldsymbol{x}})\big) \tag{2.64}$$

Algorithm 2.6 Mini-batch GAN training with pretrained VAR/VAC

Require: $\boldsymbol{\Theta}_{\mathcal{F}}$: Start parameters of $\mathcal{F}$
Require: $\boldsymbol{\Theta}_{\mathcal{G}}$: Start parameters of $\mathcal{G}_{\text{GAN}}$
Require: $\boldsymbol{\Theta}_{\mathcal{D}}$: Start parameters of $\mathcal{D}_{\text{GAN}}$
Require: η: Learning rate
Require: n: Batch size
Require: N_{epochs}: Number of training epochs
Require: α: Trade-off parameter
Require: n_{prop}: Number of properties
Require: $\mathcal{X}_{\text{real}} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) | i \in \{1, \ldots, N\}\}$: Training data

for N_{epochs} **do**
  Shuffle $\mathcal{X}_{\text{real}}$
  $t \leftarrow 1$ (Batch counter)
  while $t \cdot n \leq N$ **do**
    $\hat{\boldsymbol{g}} \leftarrow \frac{1}{n} \nabla_{\Theta} \sum_{i=(t-1)n+1}^{t \cdot n} \mathcal{L}(\mathbf{y}^{(i)}, \mathcal{F}(\mathbf{x}^{(i)}))$ (Approximate gradients)
    $\boldsymbol{\Theta}_{\mathcal{F}} \leftarrow \boldsymbol{\Theta}_{\mathcal{F}} - \eta \hat{\boldsymbol{g}}$ (Update parameters)
    $t \leftarrow t + 1$
  end while
end for

for N_{epochs} **do**
  Shuffle $\mathcal{X}_{\text{real}}$
  $t \leftarrow 1$ (Batch counter)
  while $t \cdot n \leq N$ **do**
    $\{\mathbf{z}^{(i)} | i \in \{(t-1)n + 1, \ldots, t \cdot n\}\} \leftarrow$ Random samples from $\mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\{\mathbf{y}^{(i)} | i \in \{(t-1)n + 1, \ldots, t \cdot n\}\} \leftarrow$ Random samples from $U[0, 1]^{n_{\text{prop}}}$

$\hat{\boldsymbol{g}}_{\mathcal{D}} \leftarrow \frac{1}{n}\nabla_{\Theta}\sum_{i=(t-1)n+1}^{t\cdot n}\mathcal{L}_{\mathcal{D}}(\mathbf{x}^{(i)}, \mathcal{G}_{\text{GAN}}(\mathbf{z}^{(i)}))$ (Approximate gradients $\mathcal{D}_{\text{GAN}}$)
$\hat{\boldsymbol{g}}_{\mathcal{G}} \leftarrow \frac{1}{n}\nabla_{\Theta}\sum_{i=(t-1)n+1}^{t\cdot n}\mathcal{L}_{\mathcal{G}}(\mathcal{G}_{\text{GAN}}(\mathbf{z}^{(i)})) + \alpha\mathcal{L}_{\text{MSE}}\big(\mathbf{y}^{(i)}, \mathcal{F}(\mathcal{G}_{\text{GAN}}(\boldsymbol{z}^{(i)}))\big)$
$\boldsymbol{\Theta}_{\mathcal{D}} \leftarrow \boldsymbol{\Theta}_{\mathcal{D}} - \eta\hat{\boldsymbol{g}}_{\mathcal{D}}$ (Update parameters $\mathcal{D}_{\text{GAN}}$)
$\boldsymbol{\Theta}_{\mathcal{G}} \leftarrow \boldsymbol{\Theta}_{\mathcal{G}} - \eta\hat{\boldsymbol{g}}_{\mathcal{G}}$ (Update parameters $\mathcal{G}_{\text{GAN}}$)
$t \leftarrow t + 1$
end while
end for

Diffusion Models

Denoising Diffusion Models are a class of latent space generative models first introduced by Sohl-Dickstein *et al.* [127] in 2015. These models function quite differently from other latent space models. Instead of learning a direct mapping between the latent space and the data space, they learn to reverse a diffusion process that incrementally transforms data into its representation in the latent space. Each of the T steps in the forward diffusion process adds a small amount of noise to the components of the original data until the values of each component are distributed completely randomly. This is a stochastic process, where the distribution of each step's outcome depends solely on the distribution of the result of the previous step. This makes it possible to model the process as a Markov chain—a stochastic model in which the probability of transitioning to the next state is determined entirely by the current state. For data of the from $\mathbf{x}_0 \sim q(\mathbf{x}_0)$ the forward process $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$ is given as:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \tag{2.65}$$

The variance of the added Gaussian noise depends on the current transition step, t, and is adjusted at each step according to a variance schedule $\beta_1, \ldots, \beta_T$:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) \coloneqq \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{2.66}$$

If the β_i are chosen small enough, for Gaussian noise it is possible to reverse this process through a similar one [128], where $p_{\Theta}(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is also determined by a normal distribution:

$$p_{\Theta}(\mathbf{x}_{1:T}) = p(\mathbf{x}_T)\prod_{t=1}^{T} p_{\Theta}(\mathbf{x}_{t-1}|\mathbf{x}_t) \tag{2.67}$$

$$p_{\Theta}(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_{\Theta}(\mathbf{x}_t, t), \boldsymbol{\Sigma}_{\Theta}(\mathbf{x}_t, t)) \tag{2.68}$$

$$p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}) \tag{2.69}$$

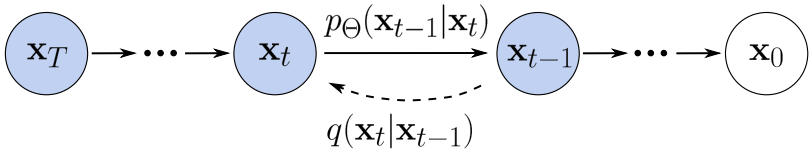

Figure 2.7: Graphical model of the forward and reverse diffusion processes. In both directions, each state depends only on its immediate predecessor, forming a Markov chain.

Both the forward and reverse processes are shown in Figure 2.7. Since the reverse process is defined by a normal distribution, only the functions $\boldsymbol{\mu}_\Theta(\mathbf{x}_t, t)$ and $\boldsymbol{\Sigma}_\Theta(\mathbf{x}_t, t)$—representing the mean and variance—need to be estimated by a neural network during training. This model can be trained by minimizing the variational upper bound on the negative log-likelihood $L(\Theta)$, which is a measure for how well the Markov Chain explains the observed data [129]:

$$L(\Theta) = \mathbb{E}[-\log p_\Theta(\mathbf{x}_0)] \leq \mathbb{E}_q\left[-\log \frac{p_\Theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] \tag{2.70}$$

where with Eq. 2.65 and Eq. 2.67:

$$\mathbb{E}_q\left[-\log \frac{p_\Theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] = \mathbb{E}_q\left[-\log p(\mathbf{x}_T) - \sum_{t\geq 1} \log \frac{p_\Theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})}\right] \tag{2.71}$$

This bound can be rewritten using Kullback-Leibler (KL) divergences between Gaussians:

$$\begin{aligned} L(\Theta) = \mathbb{E}_q\Big[&D_{KL}(q(\mathbf{x}_T|\mathbf{x}_0)\|p(\mathbf{x}_T)) + \sum_{t>1} D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)\|p_\Theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) \\ &- \log p_\Theta(\mathbf{x}_0|\mathbf{x}_1)\Big] \end{aligned} \tag{2.72}$$

A simplified version of this objective was introduced by Ho *et al.* [129]. In this simplified objective, the learned $\boldsymbol{\Sigma}_\Theta(\mathbf{x}_t, t)$ is replaced with untrained, time-dependent constants, $\boldsymbol{\Sigma}_\Theta(t) = \sigma_t^2 \boldsymbol{I}$. The simplification regarding $\boldsymbol{\mu}_\Theta(\mathbf{x}_t, t)$ is more complex. It relies on the property of the forward process that every $\mathbf{x}_t$ can be directly sampled from a modified normal distribution:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\boldsymbol{I}) \tag{2.73}$$

with $\alpha_t \coloneqq 1 - \beta_t$ and $\bar{\alpha}_t \coloneqq \prod_{i=1}^{t} \alpha_i$. This allows reparameterization of $\mathbf{x}_t$ as a

function of $\mathbf{x}_0$ and $\boldsymbol{\epsilon}$ for $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$:

$$\mathbf{x}_t(\mathbf{x}_0, \boldsymbol{\epsilon}) = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon} \tag{2.74}$$

Using this representation, the neural network $\boldsymbol{\epsilon}_\Theta(\mathbf{x}_t, t)$ can be used to approximate $\boldsymbol{\epsilon}$ from $\mathbf{x}_t$ instead of $\boldsymbol{\mu}$. $\boldsymbol{\Theta}$ denotes the parameters of the neural network. This switch from $\boldsymbol{\mu}_\Theta(\mathbf{x}_t, t)$ and $\boldsymbol{\Sigma}_\Theta(\mathbf{x}_t, t)$ to $\boldsymbol{\epsilon}_\Theta$ is what then leads to the simplified version of the objective:

$$L(\Theta) \coloneqq \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\Big[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\Theta(\mathbf{x}_t, t)\|^2\Big] \tag{2.75}$$

Algorithm 2.7 Mini-batch training of Diffusion Models

Require: $\boldsymbol{\Theta}$: Start parameters of $\boldsymbol{\epsilon}_\Theta$
Require: η: Learning rate
Require: n: Batch size
Require: N_{epochs}: Number of training epochs
Require: T: Number of diffusion steps
Require: $\mathcal{X}_{\text{train}} = \{\mathbf{x}_0^{(i)} | i \in \{1, \dots, N\}\}$: Training data
Require: $\{\beta_t | t \in \{1, \dots, T\}\}$: Variance schedule

for t **in** $\{1, \dots, T\}$ **do**
  $\alpha_t \leftarrow 1 - \beta_t$
  $\bar{\alpha}_t \leftarrow \prod_{i=1}^{t} \alpha_i$
end for
for N_{epochs} **do**
  Shuffle $\mathcal{X}_{\text{train}}$
  $j \leftarrow 1$ (Batch counter)
  while $j \cdot n \leq N$ **do**
    $\{t^{(i)} | i \in \{(j-1)n + 1, \dots, j \cdot n\}\} \leftarrow$ Random samples from $U(\{1, \dots, T\})$
    $\{\boldsymbol{\epsilon}^{(i)} | i \in \{(j-1)n + 1, \dots, j \cdot n\}\} \leftarrow$ Random samples from $\mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\hat{\boldsymbol{g}} \leftarrow \frac{1}{n}\nabla_\Theta \sum_{i=(j-1)n+1}^{j \cdot n} \|\boldsymbol{\epsilon}^{(i)} - \boldsymbol{\epsilon}_\Theta(\sqrt{\bar{\alpha}_{t^{(i)}}}\mathbf{x}_0^{(i)} + \sqrt{1 - \bar{\alpha}_{t^{(i)}}}\boldsymbol{\epsilon}^{(i)}, t^{(i)})\|^2$
    $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - \eta\hat{\boldsymbol{g}}$ (Update parameters)
    $j \leftarrow j + 1$
  end while
end for

Conditional Diffusion Models

Like other latent space models, Diffusion Models essentially learn to sample form the distribution given by the training data $P(\mathbf{x})$. To produce samples with specific target properties, several methods have been developed to condition Diffusion Models on these properties, enabling them to sample from $P(\mathbf{x}|\mathbf{y})$ instead. For diffusion models, conditioning the final distribution, also means condition every step of the reverse process $p_\Theta(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{y})$. Conditioning methods for diffusion models are more often referred to as guidance techniques and can be broadly classified into two categories: classifier guidance and classifier-free guidance.

Introduced by Song *et al.* [130], the idea behind classifier guidance is to replace $p_\Theta(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{y})$ with a more easy to compute version derived using Bayes' theorem:

$$p_\Theta(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{y}) = Z p_\Theta(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{y}) p_\Theta(\mathbf{y}|\mathbf{x}_t) \tag{2.76}$$

where Z is a normalization coefficient. Using this modified objective, Dhariwal and Nichol [131] demonstrated that the reverse process can be conditioned by simply replacing $\boldsymbol{\mu}_\Theta$ with a modified mean $\tilde{\boldsymbol{\mu}}_\Theta$:

$$\tilde{\boldsymbol{\mu}}_\Theta(\mathbf{x}_t, t) = \boldsymbol{\mu}_\Theta(\mathbf{x}_t, t) + s\Sigma_\Theta(\mathbf{x}_t)\nabla_{\mathbf{x}_t} \log p_\Theta(\mathbf{y}|\mathbf{x}_t) \tag{2.77}$$

where s is a scaling factor used to control the influence of the classifier guidance on the overall training process. To compute $\nabla_{\mathbf{x}_t} \log p_\Theta(\mathbf{y}|\mathbf{x}_t)$, a separately trained time-dependent classifier, $\mathcal{F}_{Diff}(\mathbf{x}_t, t)$, is required to learn the conditional probability $p_\Theta(\mathbf{y}|\mathbf{x}_t)$. To train the classifier, pairs of training data $(\mathbf{x}_t, \mathbf{y})$ are needed. These pairs are generated by first sampling from $(\mathbf{x}_0, \mathbf{y})$ and then applying the forward process to compute $\mathbf{x}_t$ by sampling from $q(\mathbf{x}_t|\mathbf{x}_0)$. This approach ensures that the classifier has access to time-dependent representations of the data, enabling it to provide accurate gradients for conditioning during the reverse process.

However, as training an additional classifier on noised data introduces additional complexity to the training pipeline, Ho *et al.* [132] introduced an approach they called *classifier-free guidance* to condition diffusion models. This approach discards the classifier and introduces a process, where a single model $\boldsymbol{\epsilon}_\Theta(\mathbf{x}_t, \mathbf{c})$ is trained on both conditional and unconditional generation. For the unconditional training simply zeros are used for the class identifier $\mathbf{c}$ to indicate a unconditional generation $\boldsymbol{\epsilon}_\Theta(\mathbf{x}_t) = \boldsymbol{\epsilon}_\Theta(\mathbf{x}_t, \mathbf{0})$. During training it is randomly determined if a conditional or unconditional class identifier is used for a sample. The rate at which unconditional sampling is used is usually around 10% [133]. Generation

is performed using the linear combination of the conditioned an unconditioned predictions:

$$\tilde{\boldsymbol{\epsilon}}_\Theta(\mathbf{x}_t, \mathbf{c}) = (1 + w)\boldsymbol{\epsilon}_\Theta(\mathbf{x}_t, \mathbf{c}) - w\boldsymbol{\epsilon}_\Theta(\mathbf{x}_t) \tag{2.78}$$

where w is the weight determining the influence of the conditional model. For $w = 0$ sampling only relies on the conditional model, while $w > 0$ also takes the unconditional model into account.

Constraint Learning in Generative Models

While generative models generally aim to approximate the distribution of a training dataset, it is often less crucial to match the exact probability of each sample and more important to identify samples that fit within the original data—i.e., are considered valid—while maintaining a certain degree of diversity. For example, to generate realistic images of cats it is more important to avoid unrealistic colors of fur, than to exactly match the distribution of natural fur colors. The general ability to pick up such design constraints from just a set of exemplary data is one of the core strength of generative models. However, even for these models, finding the exact boundary of the valid space can be difficult based on positive data alone [134], especially when there is a sharp cutoff between valid and invalid data.

One of the reasons it is so difficult to determine the boundary between valid and invalid data is that this boundary is determined using data points from just one side, as training only uses valid data. Introducing invalid—also called negative—data into the training process of generative models adds data on the other side of the decision boundary. Asokan and Seelamantula [135] described this approach as teaching a model what not to learn. They incorporated invalid data into the training of a GAN by providing the discriminator with invalid samples and training it to classify data into three categories: valid, invalid, and generated data. A similar approach was taken by Giannone *et al.* [134] by introducing a second discriminator output that determines if data is valid or invalid. Through an auxiliary classifier they were also able to extend this approach to diffusion models and further showed that adding invalid data is much more valuable than adding additional valid samples.

For these approaches it has to be decided before training which data is valid or invalid. However, in the recent years an approach for the training of diffusion models that handles this more flexible has become more wide spread, especially for text-to-image generation. Negative prompts—descriptions of what should be excluded from the generative images— are used in addition to the normal

prompts to improve the quality of generated images [136]. Adding these negative prompts does not even require any changes to the training process. Liu *et al.* [137] found that prompts can be combined or negated during the generation process by modifying *classifier-free guidance*. Just by altering the score function (see Eq. 2.78), they effectively created the equivalent of a logical conjunction (AND) and negation (NOT) for prompts.

Perhaps even more important than how negative prompts are incorporated into diffusion models is how these prompts take effect. This was the primary focus of Ban *et al.* [136], who analyzed the cross-attention maps of diffusion models. These maps essentially indicate the likelihood of specific tokens appearing in the generated image [138]. By doing so, they found that during the early stages of the diffusion process, negative prompts introduce negative noise that cancels out the positive noise responsible for forming undesired objects in the image. Surprisingly, if the negative prompt is applied only during the early diffusion steps, the cancellation of positive noise results in a greater momentum towards positive noise in the later steps, ultimately increasing the likelihood of the undesired object forming.

2.4 Machine Learning in Mechanical Metamaterials

Over the recent years, machine learning models have been increasingly used to accelerate the design of mechanical metamaterials [1]. These models have been applied primarily in two areas: using predictive models to speed up the computation of material properties, and utilizing inverse or generative models to design metamaterials with desired target properties.

2.4.1 Property Prediction

Assessing the material properties of a new metamaterial is crucial for determining its potential usefulness and applications. To determine these properties, either experiments or simulations can be used. While experiments tend to be more precise, simulations can be automated for high throughput, enabling the assessment of a large number of different structures. Machine learning models can also predict the properties of metamaterials; however, as we know from Section 2.3.1, training these models requires a lot of data, for which the mechanical properties need to be obtained through either experiments or simulations. So, what is the

purpose of assessing the properties of thousands of structures to create a model for assessing similar structures? The answer to this is to create a computationally efficient model to compute the properties of thousands more structures, which is necessary for inverse and generative design [139]. Based on the complexity of the chosen machine learning model and the representation of the metamaterial, this can determine whether it is possible to train an inverse model at all. For instance, Deng *et al.* [140] reported a reduction of computation time for 10000 samples from 23.2 days to 0.2 seconds.

However, not every type of machine learning model is suitable for every type of problem; this depends on what properties shall be predicted, as well as how the metamaterial is represented and what restrictions are imposed on it. Common deep learning models used to predict properties of mechanical metamaterials include Feedforward Neural Networks (FNNs), Convolutional Neural Networks (CNNs), and Graph Neural Networks (GNNs).

Feedforward Neural Networks

As one of the most universal deep learning models, FNNs are not constrained by a specific geometric representation of the input data. The input simply needs to be converted into a feature vector (see 2.3.1). However, when these feature vectors are high-dimensional, it becomes difficult for an FNN to generalize beyond the training data [96]. Consequently, FNNs are best suited for processing metamaterial structures that can be effectively represented through low-dimensional feature vectors.

Due to this limitation of FNNs to low-dimensional representations, the number of parameters used to describe mechanical metamaterials generally remains below triple digits [140–148], with only a few exceptions [149]. As the only requirement for these structures is to be able to be described using a low-dimensional representation, they include a wide range of different mechanisms, such as hinged quadrilaterals [140], beams with varying stiffness [144] or interlocking panels [145].

The properties predicted by FNNs show a large variety. They range from general mechanical properties such as Young's modulus [143, 144], stress-strain curves [140, 145], and effective strain energy density [141, 148, 149], to more structure-specific ones, such as forward and backward snapping force [144] and buckling strength [146] in beam-based metamaterials. To acquire the training data necessary to train a FNN, these properties must be computable through simulation—finite element analysis (FEA) being the method used in all cases mentioned in this

paragraph.

Training datasets for FNNs usually consist of thousands of examples, although exceptions are possible if the design space is kept small. For example, Fatehi *et al.* [141] successfully trained an FNN to predict strain energy density for interlocking panels using only a hundred samples by limiting the forty-nine design parameters to four discrete values each.

Convolutional Neural Networks

While FNNs are well-suited for processing metamaterials with low-dimensional representations, CNNs are designed for grid-like data and can even handle higher-dimensional inputs if they are properly structured [96]. Moreover, the periodic nature of metamaterials can be easily captured using periodic padding in the Convolutional layers. Consequently, CNNs are often used for metamaterials whose unit cells are represented by 2D pixel grids or 3D voxel structures [45, 150–152], even if parameter numbers are low.

Due to this, the sizes of the metamaterial representations processed by CNNs vary significantly—from around a hundred elements [45, 151], which falls within the range manageable by FNNs, to arrays with several thousand elements [153]. For large numbers of elements, these representations become very generic; the designs are no longer tied to a specific class of metamaterials, and the number of parameters only limits the minimum size of the mechanisms that can be included in the unit cells. Furthermore, for this reason, these representations are also employed with genetic algorithms or topology optimization, which also makes convolutional neural networks a good choice as surrogate models in combination with these methods [151, 154].

Similar to the representations used, there is also a large discrepancy in the amount of training data required to train these models. This number depends on the size of the chosen representation, as well as the variance of the structures, and can range from several hundred [150] to over a million [155]. As with FNNs, due to the large number of samples needed for training, FEA is usually used to compute the mechanical properties of the training data.

The mechanical properties predicted by CNNs range from basic mechanical properties such as stiffness [150, 151] and Poisson's ratio [152] to more functional ones, like the shape exhibited under deformation [45]. Bonfanti *et al.* [155] even predicted the efficiency of mechanical pliers built from a truss metamaterial structure.

Graph Neural Networks

As the name suggests, Graph Neural Networks (GNNs) are specifically designed to process data represented in the form of graphs [156]. Graph representations are particularly well-suited for many metamaterial structures, especially mesh and lattice metamaterials, which are essentially composed of interconnected nodes. In addition to mesh and lattice representations [146, 157], GNNs are often applied to predict the properties for porous metamaterials. For example, Xue *et al.* [158] applied GNNs to porous metamaterials by converting them into systems of rigid elements connected by nonlinear springs, while Meyer *et al.* [159] applied them to 3D porous shell metamaterials using primal and crystal graph representations.

Like CNNs, GNNs are well suited to process higher-dimensional inputs. However, graphs offer a more flexible representation with fewer parameters than the grid-like structures required for CNNs, which is why GNNs are more frequently applied in three-dimensional settings [157, 159]. In such cases, grid representations scale cubically with resolution, whereas the adjacency matrix of a graph scales only quadratically with the number of nodes. Generally, for mechanical metamaterials, the number of nodes is relatively small—usually less than a hundred. For example, Zheng *et al.* [157] used up to 27 nodes, while the crystal graph representations employed by Meyer *et al.* [159] required only up to eight.

As with FNNs and CNNs, the number of samples used to train GNNs varies greatly depending on the type of metamaterials they are applied to. Dataset sizes range from a few hundred samples [146] to several hundreds of thousands [157, 159], with mechanical properties usually obtained through FEA. Often, the property in question is the stiffness tensor of the metamaterial [157, 159]. Another common application is predicting the deformation field of truss metamaterials [146, 158]. However, there are also more specialized applications. For instance, Indurkar *et al.* [160] used GNNs to classify whether truss-based metamaterials are bending- or stretching-dominated.

2.4.2 Inverse and Generative Design

After a predictive model for metamaterial properties has been trained, it can serve as a basis for training an inverse or conditional generative model (see Section 2.3.2). There are several possible choices for the architecture of such a model, not to mention all the options for hyperparameter selection. For the design of mechanical metamaterials, the most commonly chosen models belong to two classes: Tandem Neural Networks (TNNs), and latent-space models.

Tandem Neural Networks

Tandem Neural Networks (see Section 2.3.2) provide a convenient way to utilize a pretrained property predictor for training a generative model for mechanical metamaterials. They have been used to generate a variety of different structures, such as spinodoid [161], truss [103], and kirigami metamaterials [162]. In general, TNNs can be applied to any metamaterial structure, as long as it is parametrized in such a way that generated designs cannot lie outside the range of the data used to train the property predictor.

To fulfill these restrictions, the classes of metamaterials for which TNNs are used are carefully constructed with such a parameterization in mind. As the training of TNNs is very straightforward, this construction can become more complicated than training the network itself. For example, to obtain a truss metamaterial with a parameterization that fits these requirements, Bastek *et al.* [103] devised an elaborate construction method. First, they combined one to three out of seven elementary lattices to create an orthotropic unit cell. This unit cell is then stretched and rotated, and subsequently stretched and rotated again to break the orthotropic symmetry. Finally, the relative density of the unit cell is adjusted by varying the thickness of the trusses.

In general, TNNs can only provide a single material structure for given material properties. While this is not an issue for many applications, such as the design of bone-mimetic scaffolds [161], multiple solutions can offer additional flexibility. One approach to obtaining multiple unit cells from a TNN is to introduce stochasticity into some of the TNN's layers [103].

Latent Space Models

As with other data, the most beneficial feature of latent space models for the design of mechanical metamaterials is— as their name suggests— the projection to the latent space. This feature allows these models to easily generate multiple metamaterial structures for the same material properties by sampling from a distribution in the latent space (see Section 2.3.2).

Alternatively, if the optimization of a metamaterial structure is the target, this can be performed in the latent space, which is usually lower-dimensional, thereby reducing computation time. This requires a regressive model that maps representations in the latent space to the properties of the metamaterial. To train this model, it is necessary to obtain the encoding of the training data in the latent space, which is why this approach is usually restricted to VAEs. Zheng *et al.* [157]

used this approach with a VAE to optimize the stiffness tensor of truss-lattice metamaterials. Furthermore, Wang *et al.* [163] observed, that when the regressor is trained together with the encoder and decoder, designs cluster in the latent space not only by shape similarity, but also property, which they exploited to design multiple metamaterial families.

Besides using a VAE with a regressor in the latent space for optimization of metamaterials, standard conditional VAEs can be used for the inverse design of mechanical metamaterials. For instance, Wang *et al.* [164] employed a conditional VAE combined with reduced-order modeling to design unit cells that controlled impact wave transmission in a hexagonal metamaterial. However, GANs are more prominent than VAEs, as they tend to produce higher quality data [165].

Conditioned GANs have been used to create both 2D [166] and 3D metamaterials [1, 167, 168]. This includes different methods of conditioning GANs, with GANs incorporating Versatile Regressors being the most common [1, 167, 168]. However, other approaches, such as the standard conditional GAN, have also been used [166]. Versatile Regressors are particularly common for mechanical metamaterials because they can be pretrained on simulation data, allowing for efficient sampling and gradient computation during the training of the generative model. GANs can also be used to optimize metamaterials. Mao *et al.* [169] used a GAN architecture to train a neural network that generated a variety of different structures with near-maximum normalized stiffness by modifying the discriminator. Instead of using the probability that a structure is from the training data, they used the probability that a structure has high stiffness as the discriminator output.

A newer addition to the latent space models used for metamaterial generation are Denoising Diffusion Models. Diffusion models are known to outperform GANs and VAEs in the quality of generated data for several tasks, including topology optimization in the mechanical domain [170]. However, training and inference with diffusion models are more computationally expensive. Consequently, they are typically applied to more complex problems, such as generating metamaterials that fit specific stress-strain curves [171]. Bastek and Kochmann [172] even employed a video diffusion generative model to predict the nonlinear deformation and stress response of mechanical metamaterials based on 2D Gaussian random fields.

Overall, latent space models can be used with a wide range of metamaterials, such as truss structures [157], chiral metamaterials [167], and metamaterials based on triply periodic minimal surfaces [168]. Still, as they are designed for image processing, these models are most often used with pixel or voxel representations of the design space [163, 166, 169, 171, 172].

Chapter 3

Metamaterials with curved components

3.1 Background

3.1.1 Literature review

One focus of research on metamaterial structures has been on what can essentially be described as an arrangement of straight beams with identical thicknesses connecting shared nodes [173]. Despite using just these beams as simple building blocks, the resulting metamaterial structures are known for their wide range of mechanical properties, especially for their ability to produce auxetic behavior. A large number of metamaterials with a negative Poisson's ratio have been developed, most of which use the re-entrant mechanism. This mechanism relies on hinges formed by the connections between beams within the unit cell. When the unit cell is deformed, these hinges straighten, pushing a third connected beam outward. This mechanism was first introduced in re-entrant honeycomb structures by Lakes *et al.* [174] and later used in other metamaterials such as arrowhead [10] and star form [9] structures. This behavior is mostly bending-dominated, which is typically associated with lower stiffness compared to stretching-dominated behavior [175, 176]. To create more stretching-dominated structures with higher stiffness, approaches have primarily focused on combining different types of auxetic and non-auxetic unit cells with additional connecting beams. Ali *et al.* [177] combined re-entrant honeycomb structures with regular honeycomb, Ebrahimi *et al* [178] combined them with regular honeycomb and star structures, and Etemadi *et al.* [179] combined them with arrow structures. On the other hand efforts to increase compliance in such metamaterial structures have focused on replacing straight beams with curved ones [180].

In general controlling curvature of the beams has been shown to be a mean to control not only the axial and bending stiffness of the beams [181, 182], but also the stiffness of the metamaterial as a whole [183]. Curved beams have also been successfully applied to increase energy absorption [184, 185] and alter the Poisson's ratio [186] of metamaterials. The main difficulty with curved beams is to find efficent numerical representations for them. In literature three main types of parametric curves representing curved beams can be found: Arcs, Sinusodis and Bézier splines [173]. Arcs allow simple parameterization by radius and angle range, but have only a limited range of variability through this. They can be used to cause a rotation of the node when they are deformed. This is mostly used in combination with straight beams to produce metamaterials with high compliance, often in combination with auxetic behavior [11, 187–189]. Sinusoidal beams are parameterized by amplitude and frequency of the sine. They combine high compliance with high energy absorption capability and are often used to extend these properties to the metamaterials they are part of [190, 191]. Bézier splines allow the greatest variance of the parameterized curves, making it possible to also represent the other two types of curves. Their number of parameters can be adjusted to the complexity of the curve. However, as the number of parameters grows it gets more difficult to predict the properties of the resulting curve. B-splines have been used to control complex metamaterials such as "petal-like" structures but also to decouple different mechanical properties such as Poissons's ratio and stiffness [192].

3.1.2 Motivation

To fully harness the benefits provided by the curved beams in the unit cells, a full understanding of the relationship between the geometry and the properties of the structure is needed. This understanding can generally be divided into the forward problem—predicting the properties for a given structure—and the inverse problem—finding a structure with given properties, which is more interesting from a design perspective (see also Section 2.3). While it often is possible to solve the forward problem analytically when only straight beams are involved, incorporating curved beams in the unit cells makes this more difficult [193, 194]. Therefore, numerical methods, such as Finite Element Analysis (FEA), are widely used to determine the mechanical behavior of such structures. The inverse problem however is often even more complicated, as multiple possible structures for the same target properties make it ill-posed. Recent developments in generative machine learning models provide an possible answer to this issue. Here, it was investigated how well Variational Autoencoders (VAEs) [105], Generative Adversarial Networks (GANs)

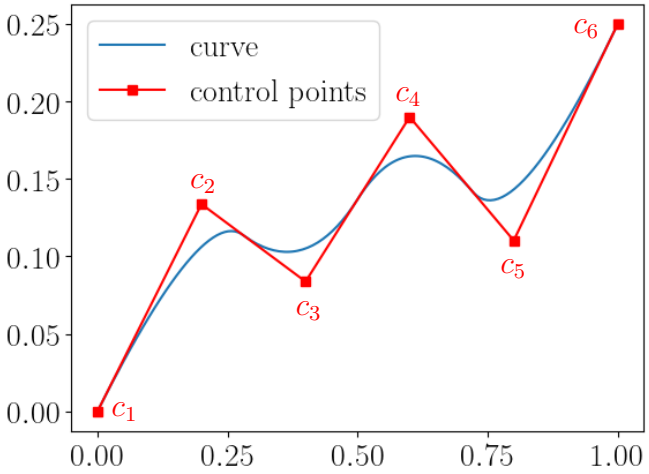

Figure 3.1: A quadratic (p=2) B-spline build from the six control points ($c_i, i = 1, .., 6$) and the knot vector $k = (0, 0, 0, 0.2, 0.5, 0.8, 1, 1, 1)$

[114] and Tandem Neural Networks (TNNs) [161] perform in solving the inverse problem. Parts of this chapter, including most figures, are adapted from: Felsch, G., & Ghavidelnia, N., & Schwarz, D., & Slesarenko, V. *Controlling auxeticity in curved-beam metamaterials via a deep generative model. Computer Methods in Applied Mechanics and Engineering*, **410**, 0045-7825 (2023) [104]. Reproduced in accordance with Elsevier's author rights policy.

3.1.3 Bézier curves

To incorporate curved elements into the unit cells of a metamaterial, a suitable representation of these elements is needed. For machine learning, this representation is ideally low-dimensional with independent components. Such a representation for curved beams can be provided through B-splines. B-spline curves $B(t)$ in $\mathbb{R}^d$ are defined by a combination of n so-called control points $c_i \in \mathbb{R}^d, i = 1, \ldots, n$, a knot vector $k \in \mathbb{R}^{n+p+1}$ and n basis functions $N_p^{(i)} : [0, 1] \to [0, 1]$ of order p [195]. One such B-spline and the corresponding control points are shown in Figure 3.1. Each point $t \in [0, 1]$ along the B-spline curve is given by a weighted sum of the control points, where the weights are defined by the basis functions:

$$B(t) = \sum_{i=1}^{n} N_p^{(i)}(t) c_i \tag{3.1}$$

This method assures that the base functions control how dependent a point on the curve is on each control point. Generally each segment of the curve only depends

on a subset of the control points. To create these segments, the curve is partitioned into smaller intervals, divided by knots $k_i,\ i = 1, \ldots, n + p + 1,\ k_i \leq k_{i+1}$. These knots are then used in the recursive definition of the basis functions. Each zero-order basis ($p = 0$) function is defined as

$$N_0^{(i)}(t) = \begin{cases} 1 & \text{if } k_i < t < k_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

It can be seen in Figure 3.2, that the basis functions of order 0 are not smooth. For higher orders this is changed through the recursive step which performs a smoothing operation on each base functions using one neighboring function, giving continuous curves:

$$N_p^{(i)}(t) = \frac{t - k_i}{k_{i+p} - k_i} N_{p-1}^{(i)}(t) + \frac{k_{i+p+1} - t}{k_{i+p+1} - k_{i+p}} N_{p-1}^{(i+1)}(t) \tag{3.3}$$

Through this operation the number of control points contributing to each point of the curve increases this the order. For quadratic splines ($p = 2$) as the one shown in Figure 3.1 this number is usually three. However, the start and end point of the curve should only depend on the first and last control point to be able to control them better. For this to occur, it is necessary to have $p + 1$ duplicates of these knots, $k_i = k_0$ for $i = 0, \ldots, p$ and $k_i = k_{n+p+1}$ for $i = p + 1, \ldots, n + p + 1$. In this case Eq. 3.1 yields $B(0) = c_1$ and $B(1) = c_n$. To generate the B-splines for the unit cells the package Splipy [196] was used.

3.2 Machine learning for auxeticity prediction

3.2.1 The curved-beam metamaterial

The unit cells of the investigated curved beam metamaterial are based on hexagonal and re-entrant honeycomb structures. As shown in Figure 3.3, the straight beams connecting the horizontal struts have been replaced by a curved beam or its mirrored equivalent. Based on the shape of the curved beam, the metamaterial displays different properties. To ensure that this variation in shape of the curved beams is the only geometric variable influencing the properties, the length of the horizontal struts (AB in Figure 3.3) was kept constant for all unit cells. Due to the periodicity there are several way to pick a unit cell; the unit cell shown in

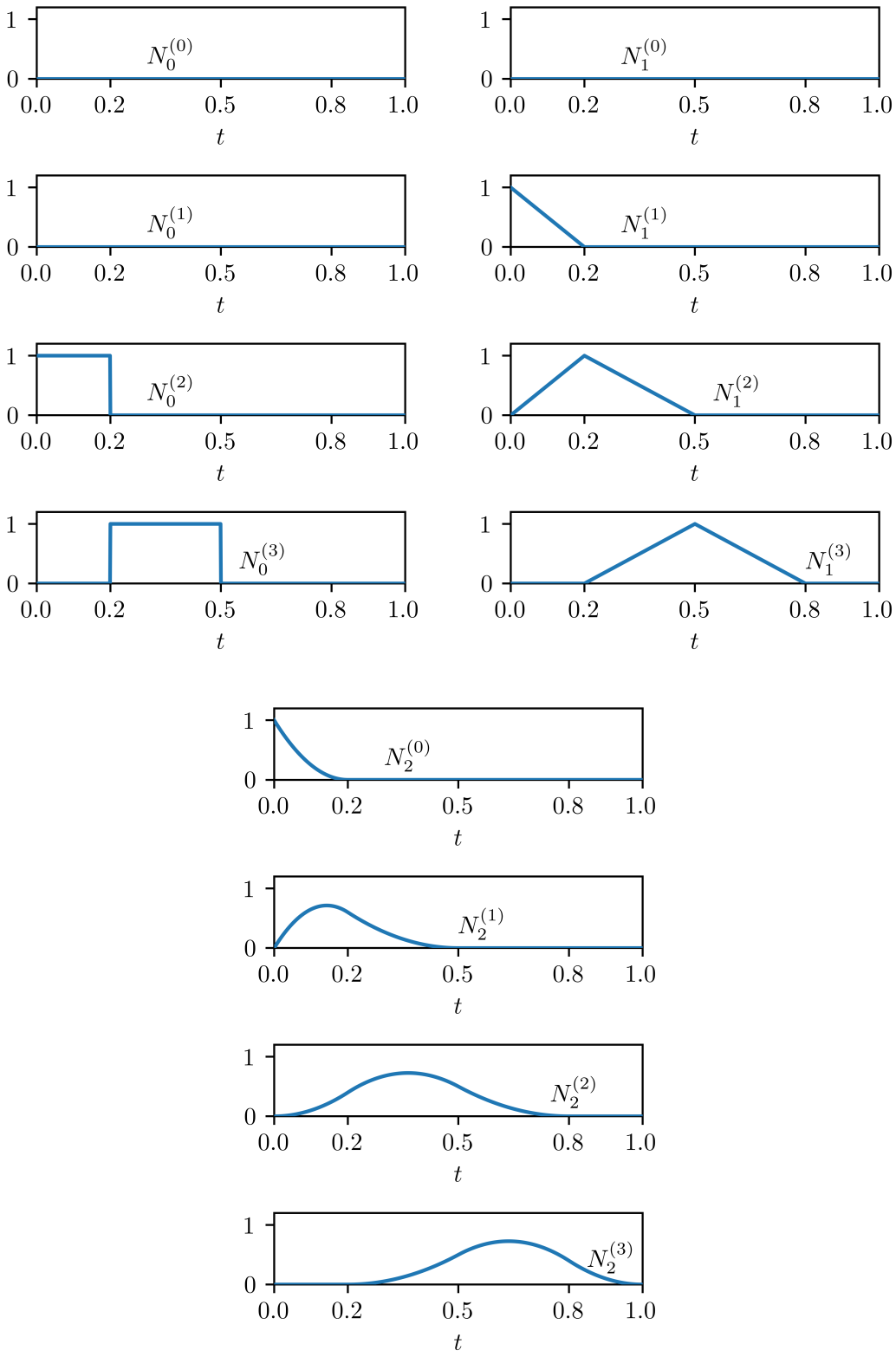

Figure 3.2: Selected B-spline basis functions of order 0,1,2 for the knot vector k = (0, 0, 0, 0.2, 0.5, 0.8, 1, 1, 1)

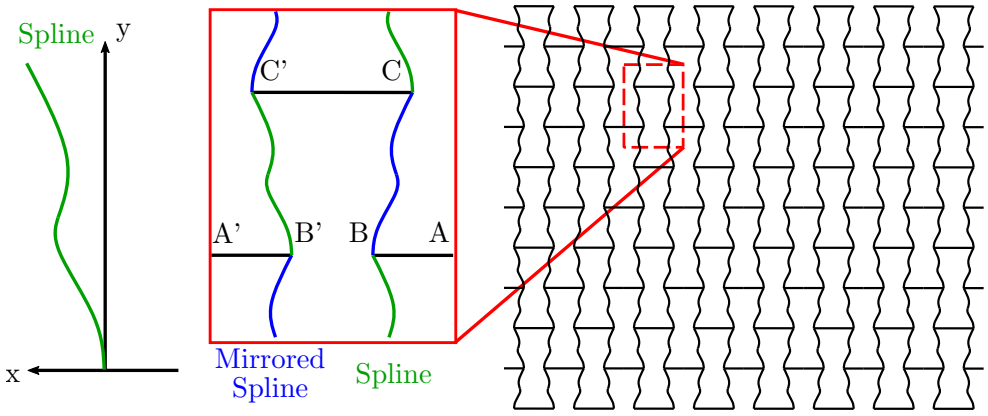

Figure 3.3: A B-spline, its corresponding representative unit cell, and the resulting curved-beam metamaterial composed of 8×5 representative unit cells.

Figure 3.3 is just one of many possibilities.

The curved beams were represented as B-splines, allowing a very efficient representation of the unit cells. Each unit cell can be fully described through the shape of the curved beam, which is then defined by a B-spline. This B-spline, in turn, is fully described by the coordinates of the control points, as the basis functions were fixed. For the unit cell in Figure 3.3, there are six control points with just twelve parameters. This number is further reduced, as control points were placed equidistantly along the y-axis, leaving only the coordinates along the x-axis as design variables. Also the coordinates of the first control point can be fixed as (0,0), allowing to represent each structure only through a set of five independent parameters - the vertical values $c_{i,y}$ of the control points c_i, $i = 2, \ldots, 6$ (see Figure 3.1).

3.2.2 Property Prediction

The Poisson's ratio of the resulting curved-beam metamaterial was predicted through simulation using the Finite Element Modeling software package COMSOL 5.4a. In this model, the curved beams were discretized into 150 Euler–Bernoulli beam elements, while each straight beam was represented by a single element. As described in Section 2.2.2, finite elements are defined by the nodes they connect, the shape functions that interpolate the mechanical response within the element, and the constitutive equations that relate deformation to other relevant physical quantities, such as stress.

Euler–Bernoulli beam elements in COMSOL connect just two nodes, one at each end of the beam. Between these, the displacements $\boldsymbol{u} \in \mathbb{R}^3$ and rotations $\boldsymbol{\theta} \in \mathbb{R}^3$

are interpolated as:

$$\begin{pmatrix} \boldsymbol{u}(\xi) \\ \boldsymbol{\theta}(\xi) \end{pmatrix} = \mathbf{N}(\xi) \begin{pmatrix} \hat{\boldsymbol{u}}_1 \\ \hat{\boldsymbol{\theta}}_1 \\ \hat{\boldsymbol{u}}_2 \\ \hat{\boldsymbol{\theta}}_2 \end{pmatrix} \tag{3.4}$$

where $\hat{\boldsymbol{u}}_i \in \mathbb{R}^3$ and $\hat{\boldsymbol{\theta}}_i \in \mathbb{R}^3$ are the nodal displacements and rotations, and $\xi \in [0, 1]$ is the coordinate running along the beam length. The shape function matrix $\mathbf{N}(\xi)$ is given by:

$$\mathbf{N} = \begin{bmatrix} N_1 & 0 & 0 & 0 & 0 & 0 & N_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & N_3 & 0 & 0 & 0 & N_5 & 0 & N_4 & 0 & 0 & 0 & N_6 \\ 0 & 0 & N_3 & 0 & -N_5 & 0 & 0 & 0 & N_4 & 0 & -N_6 & 0 \\ 0 & 0 & 0 & M_1 & 0 & 0 & 0 & 0 & 0 & M_2 & 0 & 0 \\ 0 & 0 & M_3 & 0 & M_5 & 0 & 0 & 0 & M_4 & 0 & M_6 & 0 \\ 0 & -M_3 & 0 & 0 & 0 & M_5 & 0 & -M_4 & 0 & 0 & 0 & M_6 \end{bmatrix} \tag{3.5}$$

where:

$$N_1(\xi) = 1 - \xi \tag{3.6}$$
$$N_2(\xi) = \xi \tag{3.7}$$
$$N_3(\xi) = 1 - 3\xi^2 + 2\xi^3 \tag{3.8}$$
$$N_4(\xi) = 3\xi^2 - 2\xi^3 \tag{3.9}$$
$$N_5(\xi) = L(\xi - 2\xi^2 + \xi^3) \tag{3.10}$$
$$N_6(\xi) = L(-\xi^2 + \xi^3) \tag{3.11}$$

$$M_1(\xi) = 1 - \xi \tag{3.12}$$
$$M_2(\xi) = \xi \tag{3.13}$$
$$M_3(\xi) = -\frac{6}{L}(\xi - \xi^2) \tag{3.14}$$
$$M_4(\xi) = \frac{6}{L}(\xi - \xi^2) \tag{3.15}$$
$$M_5(\xi) = 1 - 4\xi + 3\xi^2 \tag{3.16}$$
$$M_6(\xi) = -2\xi + 3\xi^2 \tag{3.17}$$

and L is the length of the beam. The constitutive law is given by:

$$\boldsymbol{\sigma} = \boldsymbol{E}\boldsymbol{\varepsilon} \tag{3.18}$$
$$\boldsymbol{\tau} = \boldsymbol{G}\boldsymbol{\gamma} \tag{3.19}$$

where $\boldsymbol{E}$ is the Young's modulus, $\boldsymbol{\tau}$ is the shear stress, $\boldsymbol{\gamma}$ is the shear strain, and $\boldsymbol{G}$ is the shear modulus.

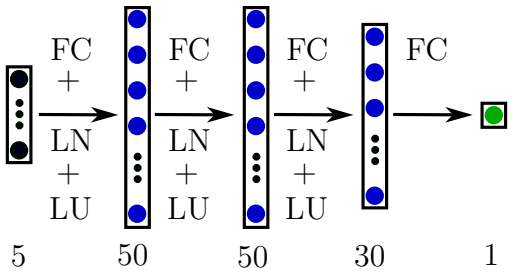

Figure 3.4: The forward NN model $\mathcal{F}$. The different layers are color-coded. Green are Poisson's ratios, black B-splines and blue hidden layers. The hidden layers consist of fully connected (FC) and layer normalization (LN) layers in combination with LeakyReLU (LU) activation functions.

3.2.3 Machine learning approach

Network architecture

To predict the Poisson's ratio of a structure the forward neural network (NN) model $\mathcal{F}$ depicted in Figure 3.4 was chosen. This architecture is quite simple, with just four fully connected layers with a maximum width of fifty. Key to this simplicity is the low dimensional input of the network. As previously discussed, a given structure can be represented by the five constitutive parameters of the corresponding B-spline—the x-coordinates of its control points. Therefore, the input to the neural network is a vector with just five entries. Each of the three hidden layers was followed by a layer normalization layer [197] for regularization and reduced training time, as well as a ReLU activation function.

Training process

Two instances of the forward NN model $\mathcal{F}$ were trained on two separate datasets to predict the Poisson's ratio of a given structure. Both datasets consist of pairs $(\mathbf{x}^{(i)}, \mathrm{y}^{(i)})$ of unit cells based on B-splines $(\mathbf{x}^{(i)})$ and the corresponding Poisson's ratio $(\mathrm{y}^{(i)})$. For *Set 1* the end point c_6 of the B-splines determining the unit cells was fixed, resulting in a dataset where all entries correspond to variation of a single re-entrant unit cell ($B'B = 0.5\ C'C$ see Figure 3.3). All unit cells in this dataset have the same dimensions, also meaning the horizontal struts always maintain the same relative position. For *Set 2* the y-coordinate of the end point of the B-spline was determined randomly, allowing a wider range of geometries and properties. Figure 3.5 shows that this variation can change the underlying geometry of the unit cell from re-entrant-like to more hexagonal. To obtain the Poisson's ratio labels for the structures in both datasets, simulations were performed in the FEA

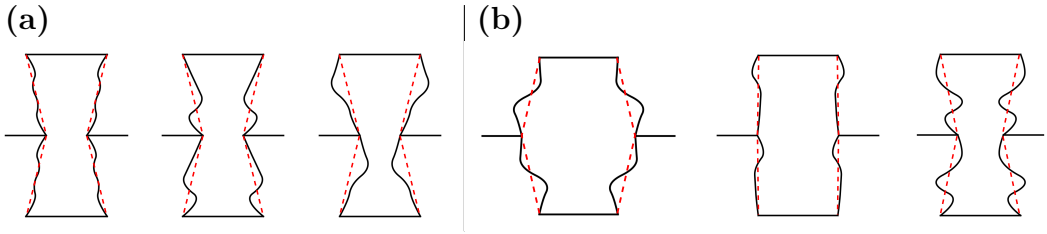

Figure 3.5: Unit cell examples from *Set 1* **(a)** and *Set 2* **(b)**. While the dimensions of the unit cells in *Set 1* are constant, the cells in *Set 2* resemble either re-entrant or hexagonal structures more.

software package COMSOL 5.4a. This means that $\mathcal{F}$ acts as a surrogate model for the FEA simulation. For these simulations, the B-splines were discretized into 150 beam elements. Both datasets comprised 25,000 data points, with 90% used as training data and the remaining 10% used as test set. Both the parameters of the structures and the corresponding Poisson's ratio labels were normalized to the interval $[0, 1]$. The Mean Squared Error (MSE) between the Poisson's ratio predicted by the neural network and the label obtained by the simulation was used as the loss function for training $\mathcal{F}$:

$$\min_{\mathcal{F}} \mathcal{O}(\mathcal{F}) = \frac{1}{N} \sum_{i=1}^{N} \left(\mathcal{F}(\mathbf{x}^{(i)}) - \mathrm{y}^{(i)} \right)^2 \tag{3.20}$$

For both instances of the model, Adam [94] was chosen as the optimizer. Training was performed for 1000 epochs with a learning rate of 1e-4 and a batch size of 32.

3.2.4 Results

Figure 3.6a shows the comparison between the Poisson's ratios obtained by the FEA (ν_s) and the forward model $\mathcal{F}$ (ν_n) for both the train and test parts of *Set 1*. It can be seen that the predictions of the network are very similar to the FEA results ($R^2 = 0.99984$). An ideal fit would yield a 45° line which the results are quite close to. Furthermore, a good generalization beyond the training data can be observed as a similar fit is obtained for the test set ($R^2 = 0.99984$). Similar results were obtained for *Set 2*. Figure 3.6b shows the comparison between the Poisson's ratio calculated through FEA and the value predicted by the network. Again, predictions of $\mathcal{F}$ are very similar to simulation results ($R^2 = 0.99976$) and the model shows good generalization with test set performance equivalent to the training set ($R^2 = 0.99975$).

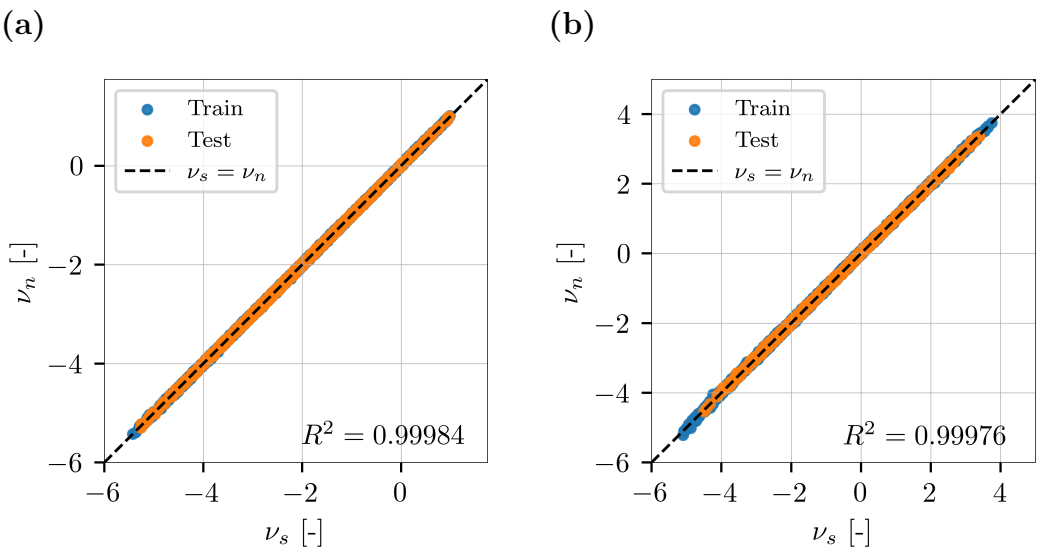

Figure 3.6: Comparison between the Poisson's ratios obtained by FEA (ν_s) and the forward NN model $\mathcal{F}$ (ν_n) for training and testing data from *Set 1* **(a)** and *Set 2* **(b)**.

3.2.5 Conclusion

The presented forward NN model $\mathcal{F}$ shows the ability to reliably predict the mechanical properties of the metamaterial with curved elements ($R^2 > 0.999$) and shows good generalization beyond the training data ($R^2 > 0.999$). Due to the low-dimensional representation of the unit cells, made possible by using B-splines to describe the curved elements, the network has a low number of parameters and a inference time of 0.07s. This combination of precise predictions and computational efficiency allows the model to act as a reliable surrogate for the simulation. Therefore, from here on out, $\mathcal{F}$ will be used to replace simulations for the solution of the inverse problem.

3.3 Inverse Design

3.3.1 Tandem Neural Network

Network architecture

As introduced in Section 2.3.2, a TNN model utilizes a pretrained neural network that solves the forward problem to circumvent the one-to-many mapping problem of inverse problems. Here, the TNN architecture shown in Figure 3.7 was used. It utilizes the forward model $\mathcal{F}$ to establish a one-to-one mapping between the Poisson's ratio of a generated structure and its target value, which is then used

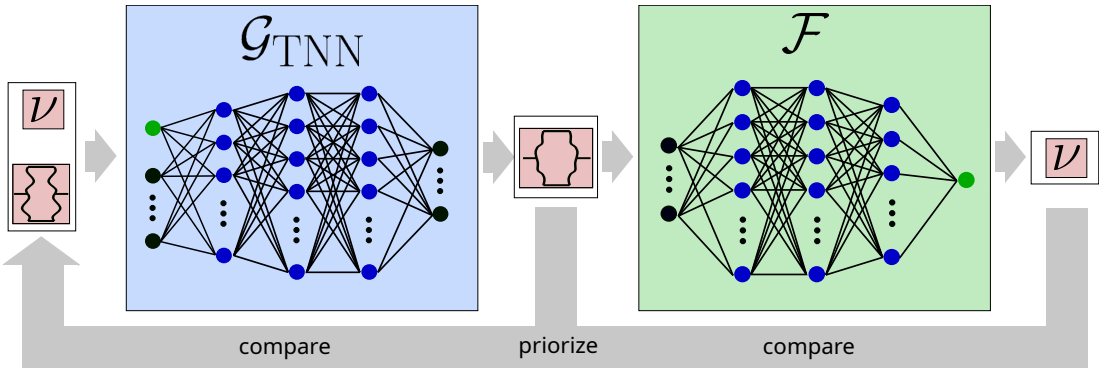

Figure 3.7: Schematic of the inverse design system. The generative model $\mathcal{G}_{\text{TNN}}$ receives a target Poisson's ratio and a guide curve. Based on these inputs, it generates unit cells defined by a B-spline. This B-spline is then passed to the forward model $\mathcal{F}$, which estimates the Poisson's ratio of the generated structure so it can be compared to the target value during training.

to train a generative model $\mathcal{G}_{\text{TNN}}$. To create multiple unit cells with the same properties, the parameters of a target unit cell were supplied as guide.

The generative model $\mathcal{G}_{\text{TNN}}$ shown in Figure 3.8 is a inverted version of $\mathcal{F}$ with the guide as additional input. It has three hidden layers with a width of 50, 50, and 30, respectively, and an output layer with five outputs. Each of the three hidden layers was followed by a layer normalization layer [197] for regularization and reduced training time, as well as a ReLU activation function. The output layer is followed only by a sigmoid activation function to constrain the output to the range of unit cells observed during training.

Training process

Similar to $\mathcal{F}$, two instances of $\mathcal{G}_{\text{TNN}}$ were trained: one to generate unit cells with fixed dimensions and another to generate unit cells with varying dimensions. These two only differ in the instances of $\mathcal{F}$ used during the training process. Since $\mathcal{F}$ requires normalized data, the same normalization as for $\mathcal{F}$ was applied to both the representation of the generated unit cell and the input parameters. This also allowed simple generation of a training dataset for the tandem NN. For the 25,000 Poisson's ratio and guide unit cell combinations $(\mathbf{x}^{(i)}, \mathrm{y}^{(i)})$ the target Poisson's ratios and the parameters of the guide unit cells were sampled randomly from uniform distributions. 90% of these were used as training set and the remaining 10% as test set. For this training no labeling by FEA simulation was required, since the surrogate $\mathcal{F}$ was utilized instead, which can also be seen by the absence of any labels in the loss. The loss function for this TNN is comprised of three parts,

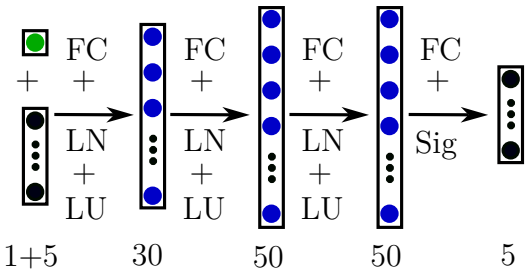

Figure 3.8: The generative NN model $\mathcal{G}_{\text{TNN}}$. The different layers are color-coded. Green are Poisson's ratios, black B-splines and blue hidden layers. The hidden layers consist of fully connected (FC), layer normalization (LN) layers in combination with LeakyReLU (LU) and Sigmoid (Sig) activation functions.

1) the MSE between the target properties and the properties of the generated structure, as predicted by $\mathcal{F}$ 2) the MSE between the parameters of the generated structure and the guide 3) a term that penalizes solutions that deviate too much from a straight line:

$$\begin{aligned}\min_{\mathcal{G}_{\text{TNN}}} \mathcal{O}(\mathcal{F}, \mathcal{G}_{\text{TNN}}) = & \underbrace{\frac{1}{N}\sum_{i=1}^{N}(\mathcal{F}(\mathcal{G}_{\text{TNN}}(\mathbf{x}^{(i)}, \mathrm{y}^{(i)})) - \mathrm{y}^{(i)})^2}_{1)} \\ & + \alpha \underbrace{\frac{1}{n}\sum_{i=1}^{N}(\mathcal{G}_{\text{TNN}}(\mathbf{x}^{(i)}, \mathrm{y}^{(i)}) - \mathbf{x}^{(i)})^2}_{2)} \\ & + \underbrace{\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N_c}\mathbb{1}_{d_j > K}\; d_j}_{3)} \;,\end{aligned} \tag{3.21}$$

Again, for both instances of the model, Adam [94] was chosen as the optimizer. Training was performed for 1000 epochs with a learning rate of 1e-4 and a batch size of 32.

Results

For the validation of the two instances of the generative model $\mathcal{G}_{\text{TNN}}$, the Poisson's ratio of around 2,500 generated structures were predicted by FEA simulation and compared to the target ratio. The result was an agreement with $R^2 > 0.997$ for the version with fixed size unit cells and $R^2 > 0.998$ for the flexible version

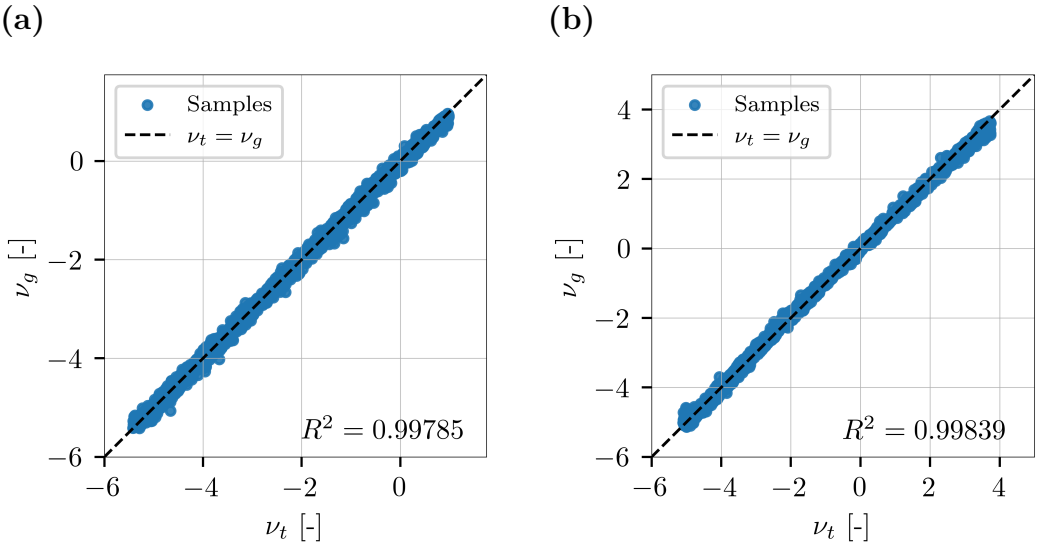

Figure 3.9: Comparison between the target Poisson's ratios (ν_t) and the Poisson's ratios of the corresponding structures by $\mathcal{G}_{\text{TNN}}$ (ν_g) fixed size unit cells **(a)** and unit cells with flexible size **(b)**.

(see Figure 3.9). When assessing these results it is important to remember that $\mathcal{G}_{\text{TNN}}$ was not trained using labels from the simulation, but only the surrogate $\mathcal{F}$. Figures 3.10a-b show the variety of unit cells generated for the same guide with different target Poisson's ratio. It can be seen that the change in target Poisson's ratio leads to a quite smooth transition between the guide and the generated unit cells. This fits the intuition that a continuous change in the beams should lead to a continuous change in the properties. Similarly, Figures 3.10c-d show the variety of curves generated for the same target Poisson's ratio, but with different guides. It can be seen, that the generated curves display great variety, despite having almost the same Poisson's ratio. This becomes especially evident in Figure 3.10d where the model displays the full range of possible unit cell sizes while barely affecting the Poisson's ratio.

3.3.2 Generative Adversarial Network

Network architecture

To solve the inverse problem of generating curved-beam metamaterials with target properties using a GAN, a variant that allows conditioning is needed. For this reason, a GAN with a so called Versatile Auxiliary Regressor (VAR) [126] was chosen (see Section 2.3.2). This architecture is comprised of three models, the generator $\mathcal{G}_{\text{GAN}}$, the discriminator $\mathcal{D}_{\text{GAN}}$ and the VAR (see Figure 3.11). The VAR is an external regression NN that facilitates the conditioning by predicting

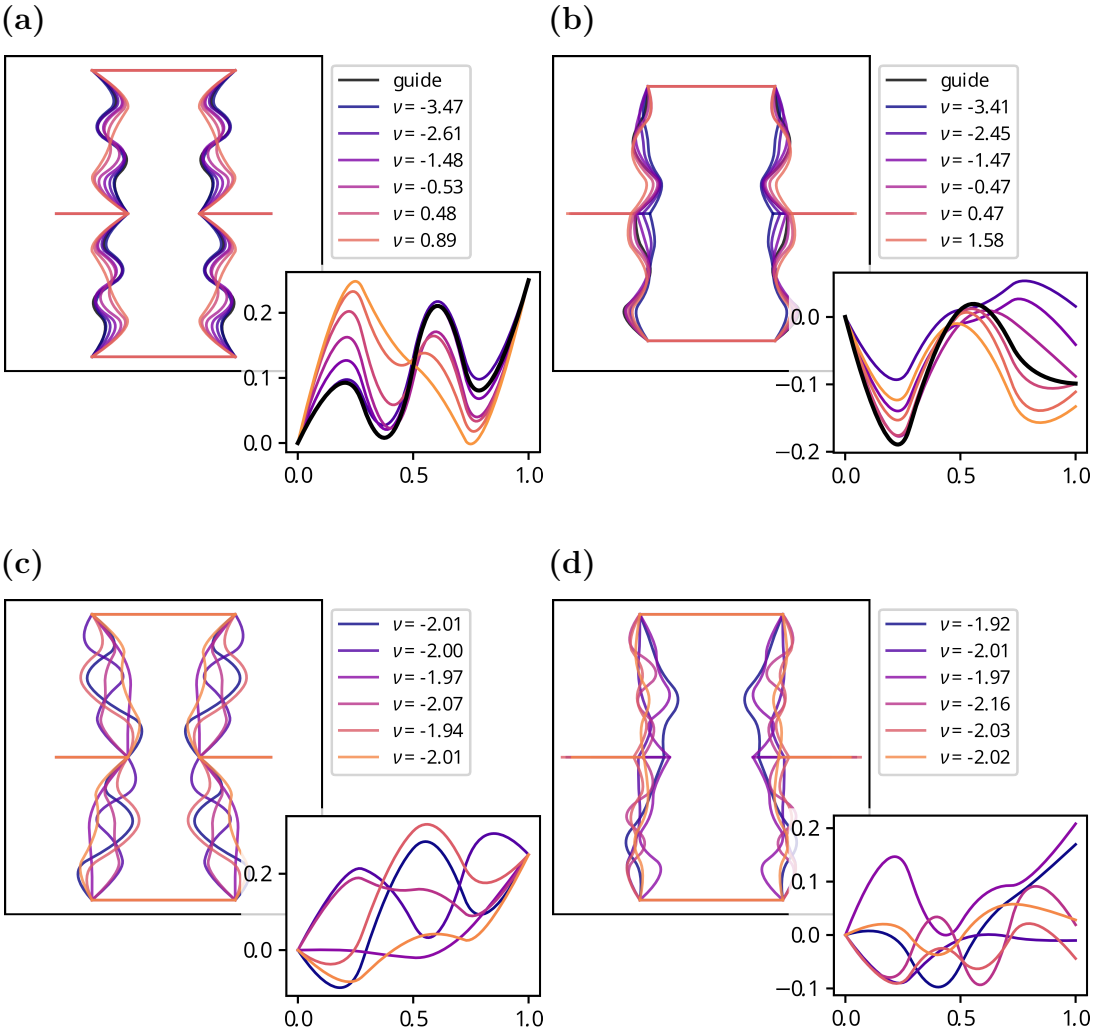

Figure 3.10: Generated unit cells and corresponding B-splines for variation of target Poisson's ratio with the same guide for with constant dimensions **(a)** and flexible dimensions **(b)**. Similarly, the generated unit cells and corresponding B-splines for variation of the guide while keeping the target Poisson's ratio the same for unit cells with fixed size **(c)** and flexible size **(d)**.

the Poisson's ratio of a generated structure, allowing it to be compared with the target Poisson's ratio. Since predicting the Poisson's ratio of a given structure is exactly what the pretrained forward NN $\mathcal{F}$ does, $\mathcal{F}$ was chosen as the VAR.

For the Discriminator $\mathcal{D}_{\text{GAN}}$ almost the same architecture as for $\mathcal{F}$ was used (see Figure 3.12a). $\mathcal{D}_{\text{GAN}}$ only differs from $\mathcal{F}$ through the sigmoid activation function added to the output layer. For the generative model $\mathcal{G}_{\text{GAN}}$ an modified inverted version of $\mathcal{F}$ was chosen (see Figure 3.12b). However, it has another input in addition to the target Poisson's ratio. This input is a noise vector with 16 entries, each sampled from a normal distribution with zero mean and unit variance. It has three hidden layers with a width of 50, 50, and 30, respectively, and an output layer with five outputs. Each of the three hidden layers was followed by a layer

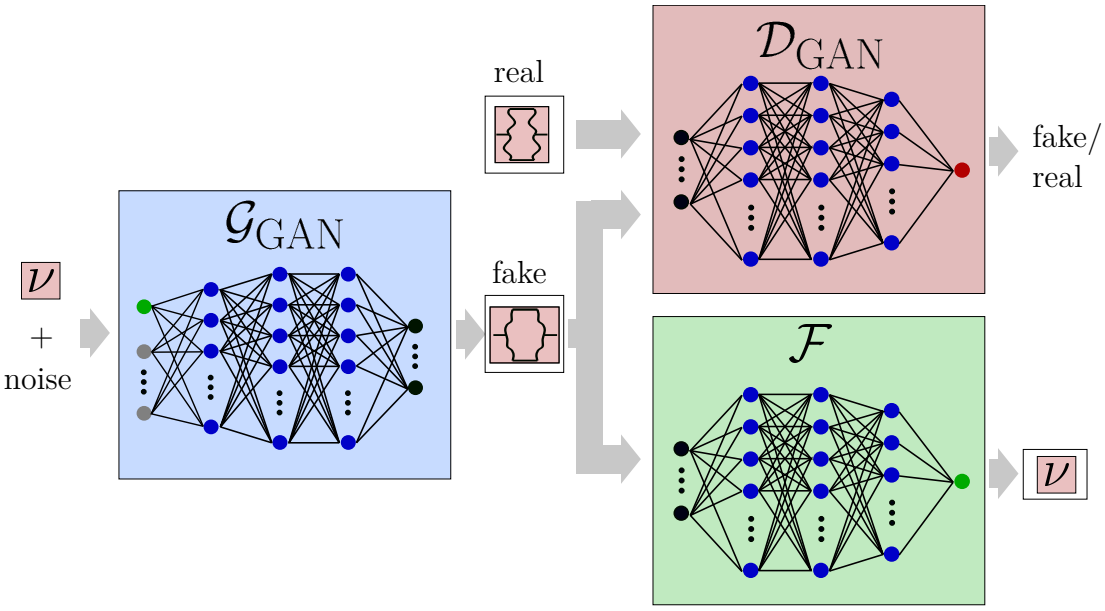

Figure 3.11: Schematic of the Generative Adversarial Network (GAN). The generative model $\mathcal{G}_{\text{GAN}}$ takes as input a target Poisson's ratio and a noise vector. Based on these inputs, it generates unit cells defined by a B-spline representation. The resulting B-spline is passed to both the forward model, $\mathcal{F}$, and the discriminator, $\mathcal{D}_{\text{GAN}}$. The forward model $\mathcal{F}$ estimates the Poisson's ratio of the generated structure to compare it with the target value during training. Meanwhile, the discriminator, $\mathcal{D}_{\text{GAN}}$, receives both the generated unit cells and a training dataset of real unit cells, and determines whether a given unit cell originates from the dataset or was generated by $\mathcal{G}_{\text{GAN}}$.

normalization layer [197] for regularization and reduced training time, as well as a ReLU activation function. The output layer is followed only by a sigmoid activation function to constrain the output to the range of unit cells observed during training.

Training process

Two pairs of instances of $\mathcal{G}_{\text{GAN}}$ and $\mathcal{D}_{\text{GAN}}$ were trained: one pair to generate unit cells with fixed dimensions and another to generate unit cells with varying dimensions. Training of these two pairs differs in three aspects. First, different instances of $\mathcal{F}$ were used as the VAR during training of each pair. These versions of $\mathcal{F}$ were trained either on datasets with fixed or varying dimensions, depending to the desired training outcome. Second, different datasets were supplied as "real" examples to train $\mathcal{D}_{\text{GAN}}$. Here the same datasets as for training the corresponding instance of $\mathcal{F}$ were chosen. Third, different normalizations were applied to the unit cell and Poisson's ratio during the training process. Again, these were chosen to match the versions used to train the different variants of $\mathcal{F}$. Less complicated

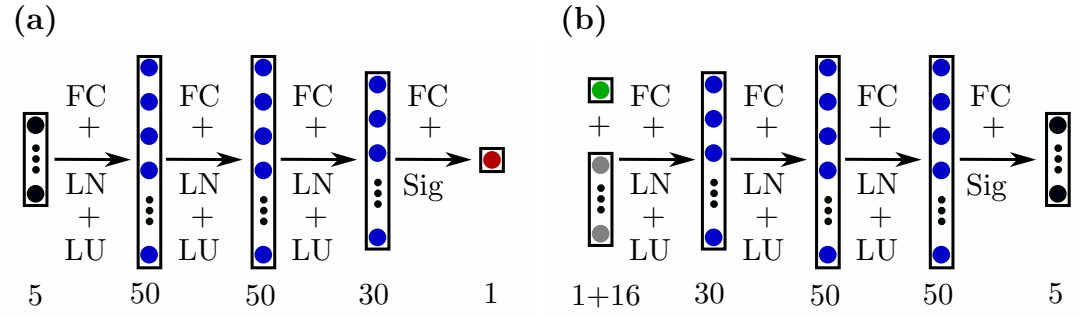

Figure 3.12: The NN models of the discriminator $\mathcal{D}_{\text{GAN}}$ **(a)** and generator $\mathcal{G}_{\text{GAN}}$ **(b)**. The different layers are color-coded: green represents Poisson's ratios, black represents B-splines, gray represents noise, and blue represents hidden layers. The hidden layers consist of fully connected (FC) layers, layer normalization (LN), and a combination of LeakyReLU (LU) and Sigmoid (Sig) activation functions.

was the selection of input batches for $\mathcal{G}_{\text{GAN}}$. Both the noise and Poisson's ratio inputs were samples from a uniform distribution $\mathcal{U}_{[0,1]}$.

Since $\mathcal{F}$ was used as the pretrained VAR, no Poisson's ratio labels were required during training. This can be also seen in the loss functions used to train the generator $\mathcal{G}_{\text{GAN}}$:

$$\begin{aligned}\min_{\mathcal{G}_{\text{GAN}}} \mathcal{O}(\mathcal{F}, \mathcal{G}_{\text{GAN}}, \mathcal{D}_{\text{GAN}}) = &- \frac{1}{N}\sum_{i=1}^{N} \log \mathcal{D}_{\text{GAN}}(\mathcal{G}_{\text{GAN}}(\mathbf{z}^{(i)}, \mathrm{y}^{(i)})) \\ &+ \alpha \frac{1}{N}\sum_{i=1}^{N} (\mathcal{F}(\mathcal{G}_{\text{GAN}}(\mathbf{z}^{(i)}, \mathrm{y}^{(i)})) - \mathrm{y}^{(i)})^2\end{aligned} \tag{3.22}$$

and the discriminator $\mathcal{D}_{\text{GAN}}$:

$$\begin{aligned}\min_{\mathcal{D}_{\text{GAN}}} \mathcal{O}(\mathcal{G}_{\text{GAN}}, \mathcal{D}_{\text{GAN}}) = &- \frac{1}{N}\sum_{i=1}^{N} \log \mathcal{D}_{\text{GAN}}(\mathbf{x}^{(i)}) \\ &+ \frac{1}{N}\sum_{i=1}^{N} \log(1 - \mathcal{D}_{\text{GAN}}(\mathcal{G}_{\text{GAN}}(\mathbf{z}^{(i)}, \mathrm{y}^{(i)})))\end{aligned} \tag{3.23}$$

Adam [94] was chosen as the optimizer for both instances of $\mathcal{G}_{\text{GAN}}$ and $\mathcal{D}_{\text{GAN}}$ each. Training was performed for 3000 epochs with a learning rate of 1e-4 for $\mathcal{G}_{\text{GAN}}$ and 5e-4 for $\mathcal{D}_{\text{GAN}}$. A batch size of 32 and a noise dimension of 16 were used.

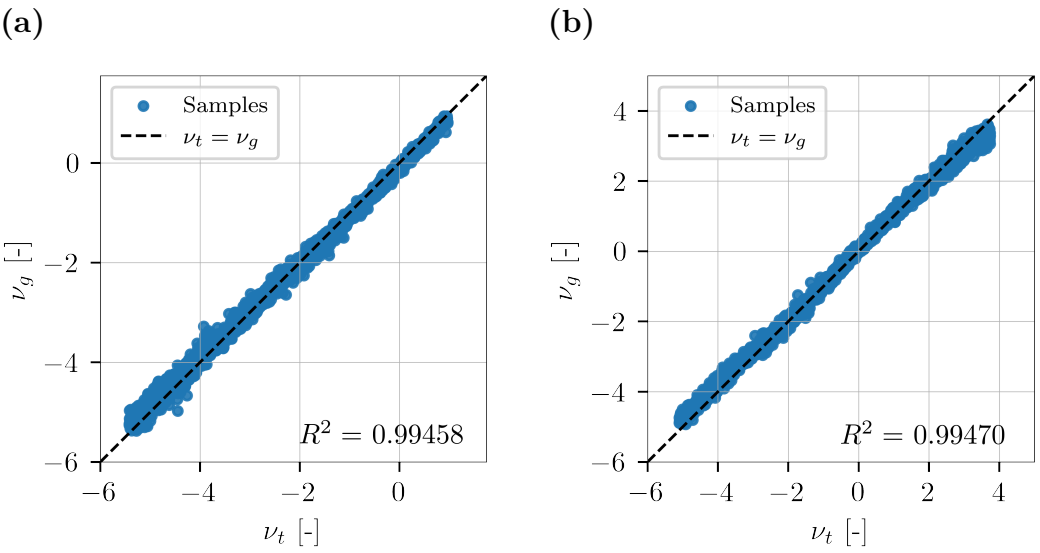

Figure 3.13: Comparison between the target Poisson's ratios (ν_t) and the Poisson's ratios of the corresponding structures by $\mathcal{G}_{\text{GAN}}$ (ν_g) fixed size unit cells **(a)** and unit cells with flexible size **(b)**.

Results

For the validation of the two instances of the generative model $\mathcal{G}_{\text{GAN}}$, the Poisson's ratio of around 2,500 generated structures were predicted by FEA simulation and compared to the target ratio. The results are shown in Figure 3.13. For the version with fixed size unit cells an agreement of $R^2 > 0.994$ was reached and $R^2 > 0.994$ for the flexible version.

3.3.3 Variational Autoencoder

Network architecture

Similar to the GAN, a conditional version of the VAE featuring an Auxiliary Regressor (AR) [113] was used (see Section 2.3.2). This architecture is comprised of three networks, as shown in Figure 3.14: the encoder $\mathcal{E}_{\text{VAE}}$, which maps the structures into the latent space; the decoder $\mathcal{G}_{\text{VAE}}$, which reconstructs the structures from the representation in the latent space; and the aforementioned AR. In VAE architectures, the decoder also functions as generator after training, hence its designation as $\mathcal{G}_{\text{VAE}}$. For generation, the latent representation is randomly drawn from a multivariate normal distribution $\mathcal{N}(0, \mathbf{I})$ and is supplied together with the target properties as input to $\mathcal{G}_{\text{VAE}}$. The encoder and AR on the other hand play only a role during training. $\mathcal{E}_{\text{VAE}}$ is needed to map structures to the latent space so that representation drawn from $\mathcal{N}(0, \mathbf{I})$ are reconstructed as valid

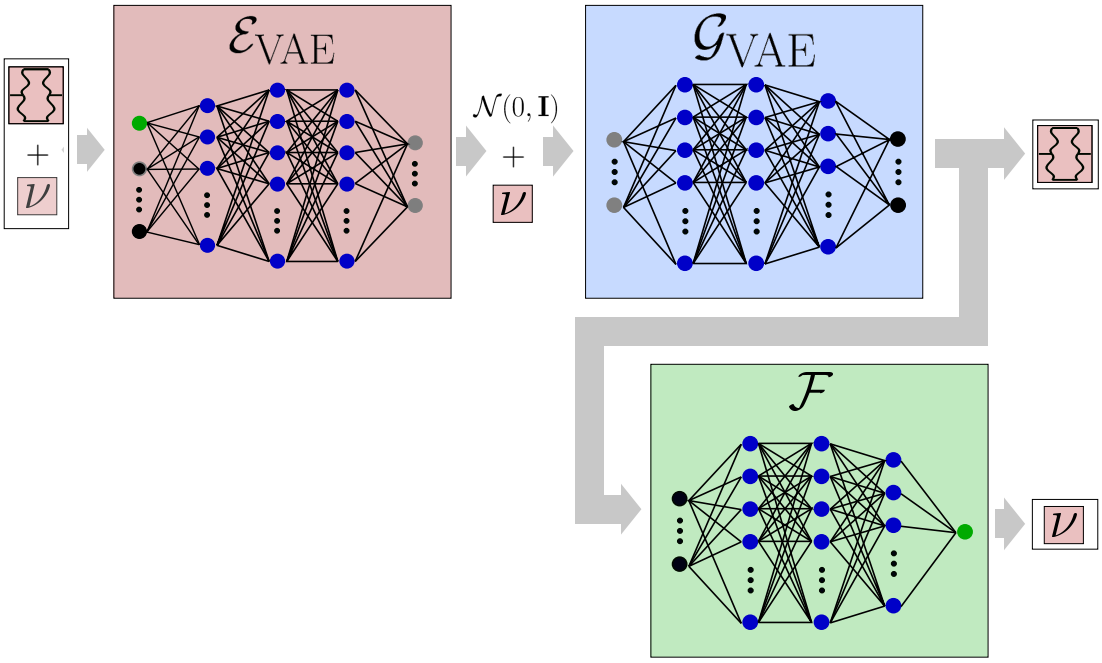

Figure 3.14: Schematic of the Variational Autoencoder (VAE). The encoder, $\mathcal{E}_{\text{VAE}}$, takes a unit cell and its corresponding Poisson's ratio as inputs and maps them to a distribution in the latent space. The generative model, $\mathcal{G}_{\text{VAE}}$, samples noise from this distribution and, together with the Poisson's ratio, generates unit cells represented by a B-spline. For training, the generated unit cells are compared to the original input to the encoder. Additionally, the generated B-spline is passed to the forward model, $\mathcal{F}$, which estimates the Poisson's ratio of the generated structure to evaluate consistency with the target value.

structures. The AR is part of one of two conditioning mechanisms. Its task is to predict the Poisson's ratio $\mathrm{y}^{(i)}$ of the reconstructed unit cell, so it can be compared to the target ratio. This allows to just use the pretrained $\mathcal{F}$ (see Section 3.2.3) as AR. The other conditioning mechanism works by just supplying the target Poisson's ratio to both $\mathcal{E}_{\text{VAE}}$ and $\mathcal{G}_{\text{VAE}}$ this allows $\mathcal{G}_{\text{VAE}}$ to pick up the correlation between $\mathrm{y}^{(i)}$ and the reconstructed image $\hat{\mathbf{x}}^{(i)}$. More details on training VAEs can be found in Section 2.3.2.

For the encoder $\mathcal{E}_{\text{VAE}}$, a similar architecture to $\mathcal{F}$ was used (see Figure 3.15a). This architecture differs only in the output layer. Instead of predicting the Poisson's ratio, $\mathcal{E}_{\text{VAE}}$ produces two output vectors: the mean and the logarithmic variance of the variational distribution in the latent space. Each of these two fully connected layers has a output size of 16, the size that was chosen for the latent space.

For the generative model $\mathcal{G}_{\text{VAE}}$ an modified inverted version of $\mathcal{F}$ was chosen (see Figure 3.15b). As mentioned before, this modified version has two inputs, first the latent vector which contains information on the original structure that should

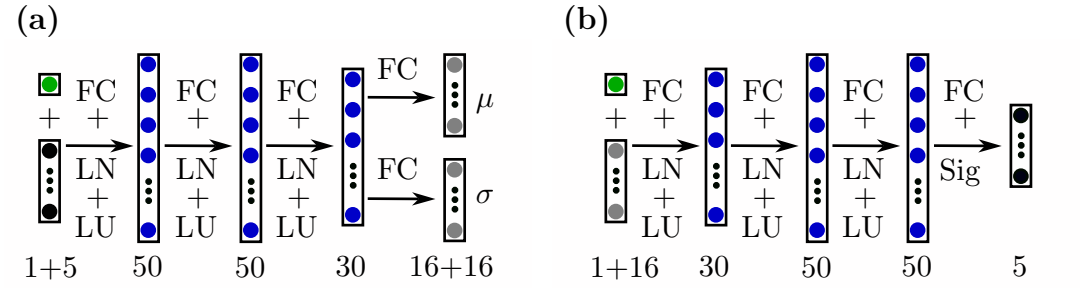

Figure 3.15: The NN models of the encoder $\mathcal{E}_{\text{VAE}}$ **(a)** and generator $\mathcal{G}_{\text{VAE}}$ **(b)**. The different layers are color-coded. Green are Poisson's ratios, black B-splines, gray noise and blue hidden layers. The hidden layers consist of fully connected (FC), layer normalization (LN) layers in combination with LeakyReLU (LU) and Sigmoid (Sig) activation functions.

be reconstructed and second the target Poisson's ratio for the decoded structure. It has three hidden layers with a width of 50, 50, and 30, respectively, and an output layer with five outputs. Each of the three hidden layers was followed by a layer normalization layer [197] for regularization and reduced training time, as well as a ReLU activation function.

Training process

Two pairs of instances of $\mathcal{E}_{\text{VAE}}$ and $\mathcal{G}_{\text{VAE}}$ were trained: one pair to generate unit cells with fixed dimensions and another to generate unit cells with varying dimensions. Since part of the conditioning of the VAE functions through capturing the correlation between unit cell and properties from the training data, a labeled dataset is needed to train $\mathcal{E}_{\text{VAE}}$ and $\mathcal{G}_{\text{VAE}}$. Here it is possible to use the same set as for training $\mathcal{F}$ (see Section 3.2.3). For each pair of $\mathcal{E}_{\text{VAE}}$ and $\mathcal{G}_{\text{VAE}}$ instances, this training dataset and the corresponding instance of $\mathcal{F}$ were selected according to the restrictions on unit cell dimensions. Similar to the training of $\mathcal{F}$, both the parameters of the structures and the corresponding Poisson's ratio labels were normalized to the interval $[0, 1]$. The loss function for this VAE is comprised of three parts, 1) the reconstruction loss, here the MSE between original and reconstructed image 2) the regularization loss, the Kullback-Leibler (KL) divergence between the variational and target distribution 3) the MSE between the target Poisson's ratio and the Poisson's ratio of the reconstructed structure:

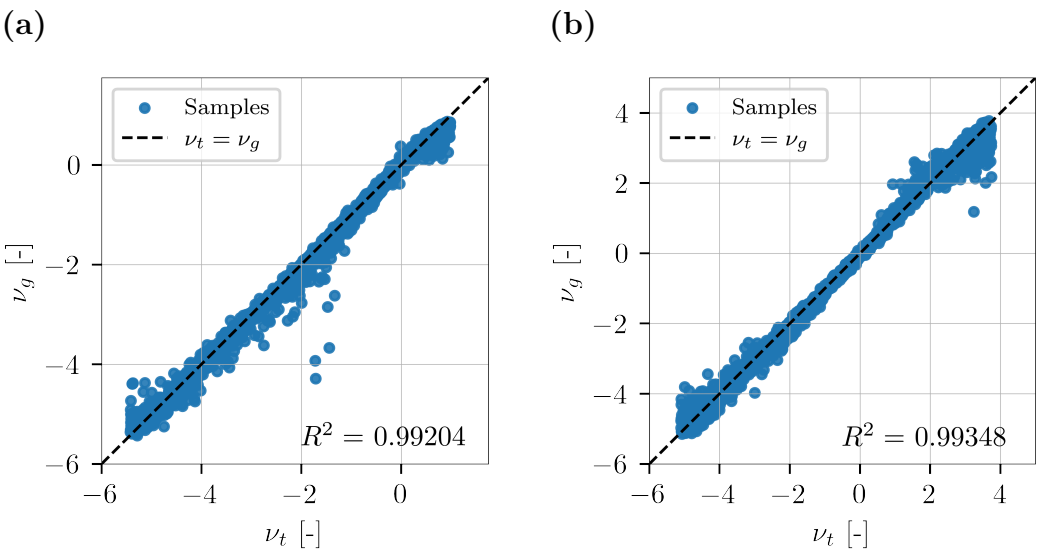

Figure 3.16: Comparison between the target Poisson's ratios (ν_t) and the Poisson's ratios of the corresponding structures by $\mathcal{G}_{\text{VAE}}$ (ν_g) fixed size unit cells **(a)** and unit cells with flexible size **(b)**.

$$\begin{aligned}\min_{\mathcal{G}_{\text{VAE}},\mathcal{E}_{\text{VAE}}} \mathcal{O}(\mathcal{F},\mathcal{G}_{\text{VAE}},\mathcal{E}_{\text{VAE}}) = & \underbrace{\frac{1}{N}\sum_{i=1}^{N}(\hat{\mathbf{x}}^{(i)}-\mathbf{x}^{(i)})^2}_{1)} \\ & \underbrace{-\frac{\alpha}{2N}\sum_{i=1}^{N}1+\log\boldsymbol{\sigma}^{(i)}-(\boldsymbol{\mu}^{(i)})^2-\boldsymbol{\sigma}^{(i)}}_{2)} \\ & \underbrace{+\frac{\beta}{N}\sum_{i=1}^{N}(\mathcal{F}(\hat{\mathbf{x}}^{(i)})-\mathrm{y}^{(i)})^2}_{3)} \end{aligned} \tag{3.24}$$

where $\hat{\mathbf{x}}^{(i)} = \mathcal{G}_{\text{VAE}}(\mathcal{E}_{\text{VAE}}(\mathbf{x}^{(i)},\mathrm{y}^{(i)}),\mathrm{y}^{(i)})$ and $\{\boldsymbol{\mu}^{(i)},\boldsymbol{\sigma}^{(i)}\} = \mathcal{E}_{\text{VAE}}(\mathbf{x}^{(i)},\mathrm{y}^{(i)})$. Adam [94] with a learning rate of 1e-4 was chosen as the optimizer for both instances of $\mathcal{G}_{\text{VAE}}$ and $\mathcal{E}_{\text{VAE}}$ each. Training was performed for 3000 epochs with a batch size of 32 and a noise dimension of 16.

Results

For the validation of the two instances of the generative model $\mathcal{G}_{\text{VAE}}$, the Poisson's ratios of around 2,500 generated structures were predicted by FEA simulation and compared to the target ratio. The results are shown in Figure 3.16. For the

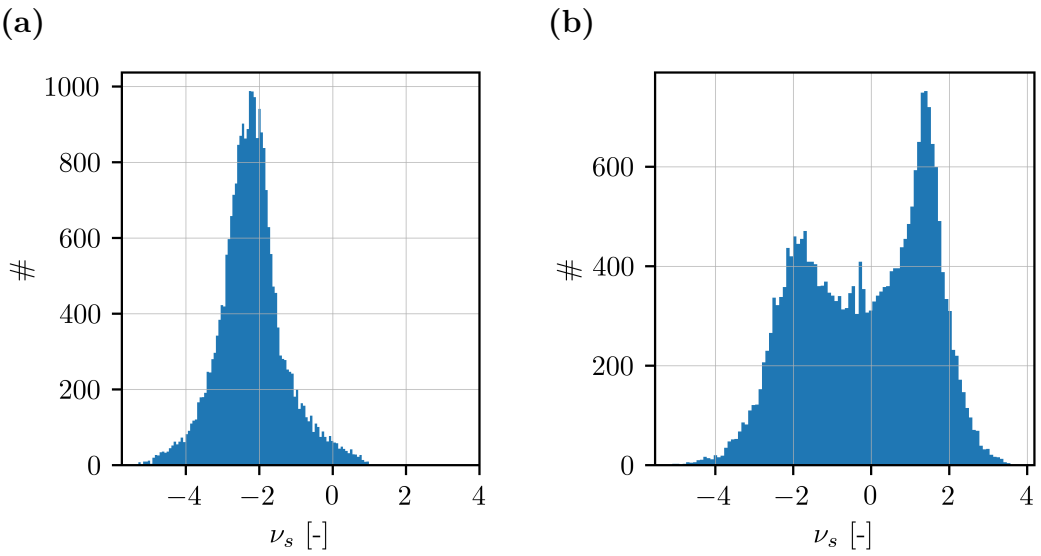

Figure 3.17: Distribution of the Poisson's ratios of the unit cells in the training dataset for $\mathcal{F}$ and $\mathcal{G}_{\text{VAE}}$ for unit cells with fixed size **(a)** and unit cells with flexible size **(b)**.

version with fixed size unit cells an agreement of $R^2 > 0.992$ was reached and $R^2 > 0.993$ for the flexible version. However, it can also be seen in Figure 3.16 that deviations from the target values are higher for extreme cases of ν_t. Very low and high ν_t show more variance than the average ν_t. These regions correspond to lower populations of ν in the training dataset, which can be seen in Figure 3.17. This lower number of training examples makes it more difficult for $\mathcal{G}_{\text{VAE}}$ to learn the correlation between between these Poisson's ratio values and the corresponding structures.

3.3.4 Conclusion

By using curved beams instead of straight ones in re-entrant honeycomb metamaterials, a wide range of Poisson's ratios, including both positive and negative values, can be achieved. NURBS are able to provide a efficient and low dimensional representation for these curved beams, which has been shown to be well suited for machine learning. This representation allowed training of a NN with a maximum layer width of 50, that still could make reliable predictions about the Poisson's ratios of a given unit cell ($R^2 > 0.999$). While training a neural network to solve this forward problem was straightforward, training another network to solve the inverse problem was more complex due to the existence of multiple possible unit cells for the same target properties. For the inverse problem three different machine learning architectures were trained: A modified TNN, a GAN and a VAE. To create a one-on-one mapping to the generated structures, the

generators of all three models received an additional input alongside the target Poisson's ratio. For the GAN and VAE, this additional input was noise, while the TNN received a guide structure that the generated geometry should resemble. All three inverse models were able to accurately generate unit cells that fit target properties ($R^2 > 0.99$), with the VAE performing slightly worse on generating structures with Poisson's ratios at the extremes of the achievable range. Unlike the GAN and TNN, the conditioning mechanism for the VAE prevents training on a dataset with uniformly distributed Poisson's ratios, leading to diminished accuracy in edge cases with lower probability. While the GAN does not have this issue with edge cases, the TNN offers additional advantages over it. The adversarial process used to train the GAN is generally unstable, whereas the TNN can be trained like a forward model. Furthermore, the TNN allows not only to generate unit cells with target properties, but also to express preference for a specific shape.

Chapter 4

Kirigami metamaterials

4.1 Background

4.1.1 Literature review

Kirigami metamaterials—named after the Japanese art of paper cutting—utilize cuts and, in some cases, folds in the material to achieve the desired behavior [198–201]. While adding cuts to a structure is a relatively simple process, it alone can lead to a wide range of behaviors and influence properties such as Young's modulus [14, 202], ultimate stress [203], deformation [162], Poisson's ratio [12, 14, 202] (see also Section 2.1.2), and even bi- and multistability [204]. The effect of a single cut in a plate is relatively simple: it breaks the continuity constraint on the structure's deformation [205]. When multiple cuts are combined, they can be used to design more complex mechanisms.

A common strategy for achieving more complex behavior is to create rotating rigid polygons (see also 2.1.2), which are connected only at their edges. These connections form hinges that allow the polygons to rotate, but they also lead to significant deformation and stress concentrations at these points [205]. To create these hinges, the cuts do not fully stretch the edges of the polygon, which is why this technique is also referred to as a *fractal cut* [206]. The polygons themselves are used to form more complex unit cells. While the cuts and polygons are typically arranged to create ordered structures, even with some degree of randomness in the cuts, the same mechanisms can still result in unusual behaviors, such as auxeticity [14].

Without the added flexibility provided by the hinges in the rotating rigid polygon structures, most kirigami metamaterials tend to undergo out-of-plane deformation [201]. This deformation manifests in buckling behavior that varies significantly depending on whether the kirigami structure is subjected to tension or compres-

sion. The stretch- and buckled-buckled kirigami structures are known to exhibit complex mechanical responses, where behavior varies with the applied strain [207].

4.1.2 Motivation

Kirigami metamaterials are known to exhibit a wide range of mechanical properties based on the arrangement of the cuts [14]. However, the full range of these properties is seldom fully explored, as the possible combinations of cuts are limited to avoid intersections and the possibly resulting stress concentrations—or even isolated pieces of material.

These limitations usually concern the rotations of the cuts, either by limiting them [14] or even aligning them [203]. In other cases, parameters and their ranges are carefully chosen to exclude unwanted designs [162]. Unfortunately, these approaches not only exclude configurations with intersections but also valid designs without them, thus limiting the range of achievable properties. On the other hand, finding parameterizations that include only valid designs is often difficult, if not impossible. This challenge is not unique to kirigami metamaterials; many other metamaterials are carefully constructed so that all structures adhere to certain design restrictions. This can be achieved by limiting designs to a fixed set of base structures that are then deformed [157], by using mirroring during construction to satisfy boundary conditions [172], or—most often—by by choosing a base structure and making small, independent alterations to it, as is done with the curved-beam metamaterial from Chapter 3.

Another approach to the problem of how to deal with these design restrictions is to learn them directly from examples. This strategy leverages a dataset containing only valid designs—those that satisfy the desired constraints—so that the model implicitly learns the underlying restrictions during training, rather than having them hard-coded into the parameterization. This is one of the great strengths of the generative machine learning (ML) models that have revolutionized image generation, such as Variational Autoencoders (VAEs) [105], Generative Adversarial Networks (GANs) [114] and the more recent Denoising Diffusion Models [129].

These techniques have recently been adapted for the inverse design of metamaterials [102, 157, 163, 168]. However, even when these models are used, metamaterials are still most often parameterized so parameters are independent and no meaningful restrictions on the design have to be learned. This poses the question why the use of generative models for generating metamaterials is so different from image generation. This chapter discuses how the non-trivial design space and absence

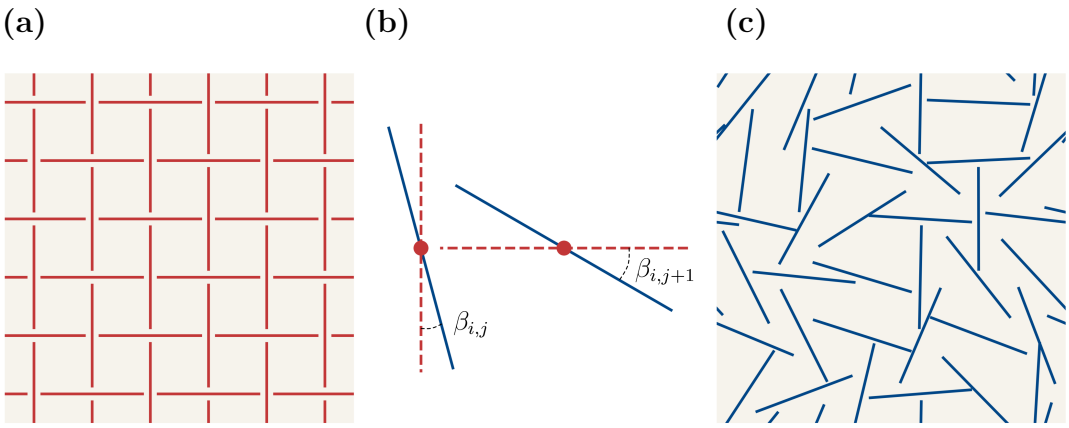

Figure 4.1: Mechanical metamaterials based on straight cuts. **(a)** Initial 6×6 pattern with alternating cuts. This architecture gives rise to auxetic behavior through the rotating squares mechanism. **(b)** Perturbation of the initial architecture is performed by adding rotations $\beta_{i,j}$ to each cut. The absolute value of rotations is capped by parameter $\beta_{\max}$. **(c)** Resulting admissible design without intersections between cuts. The likelihood of obtaining intersection-free sample through random rotations ($\beta_{\max} = 90°$) does not exceed 0.001%.

of a good similarity metric caused by the design restrictions in kirigami metamaterials interacts with the machine learning algorithms and why some of them are better at learning design restrictions than others. Parts of this chapter, including most figures, are adapted from: Felsch, G., & Slesarenko, V. *Generative models struggle with kirigami metamaterials*. *Scientific Reports*, **14**, 19397 (2024) [208]. Reproduced in accordance with Springer Nature's author rights policy.

4.2 The kirigami metamaterial

The unit cells of the kirigami metamaterials investigated here are based on a pattern where cuts with horizontal and vertical orientations are alternated within the sheet (Figure 4.1a). This base structure is known to exhibit auxetic behavior via the rotating squares mechanism [12]. Furthermore, it has been shown that by rotating the cuts (Figure 4.1b), the material properties of the resulting metamaterial can be altered [14], with the range of achievable properties increasing with magnitude of the allowed rotations. However, if the rotations reach a certain magnitude, intersections between the cuts can occur, which may lead to stress concentrations or even isolated regions of material. Due to this, the design space is usually restricted to designs without intersections. The easiest way to do this is by limiting the rotations in general, however, this excludes the majority of intersection-free configurations, such as the one shown in Figure 4.1c.

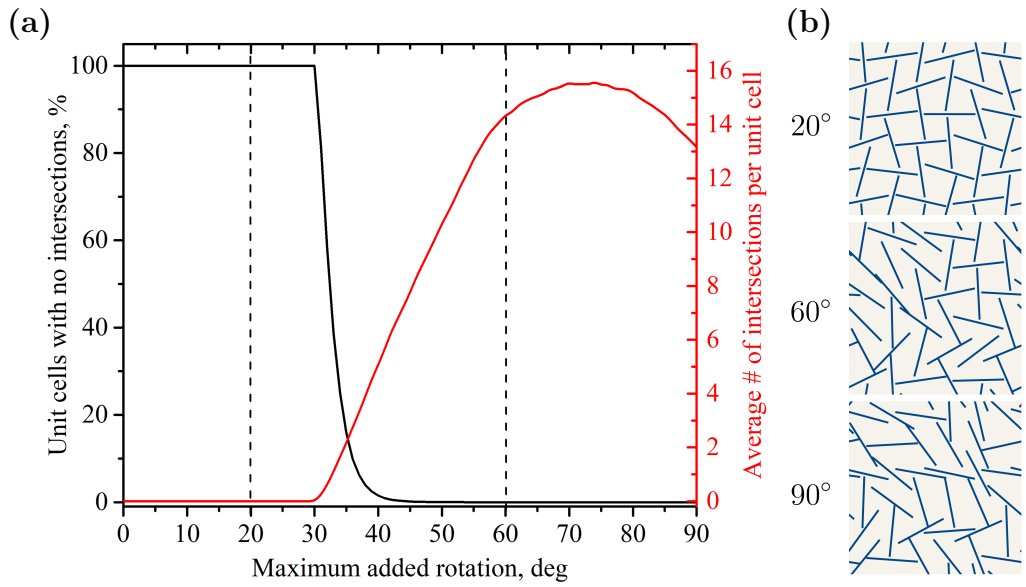

Figure 4.2: Limiting the perturbations enables control over the success rate of generation. **(a)** The average number of intersections in the samples and the likelihood of generating unit cells without intersections vs maximum deviation $\beta_{\max}$ from the base structure. **(b)** Exemplary unit cells for a maximum added rotations $\beta_{\max}$ of 20°, 60° and 90°.

To create the individual unit cells, a 6×6 cut version of the alternating unit cell (Figure 4.1a) is used, with a unit distance between the centers of the cuts and a total cut length of $l = \sqrt{3}$. The rotations, denoted as $\beta_{i,j}$ for a given cut $c_{i,j}$, where $i, j = 0, \ldots, 5$ (Figure 4.1b), are then applied to the base structure. These rotations depend on the position of each cut on the grid, which is challenging for a neural network to process. Therefore, it is necessary to consider the angle of each cut relative to the vertical direction, denoted as $\alpha_{i,j}$. The distance between the cuts and cut length were chosen so no intersections occur as long as $\beta_{\max} < 30°$.

This can be also seen in Figure 4.2a, which shows how the average number of intersections and the percentage of generated unit cells without intersections depend on the maximum for the added rotations, $\beta_{\max}$. For $\beta_{\max} > 30°$, the share of unit cells without intersections drops drastically, almost reaching zero at around 50°, while the average number of intersections rises until it peaks a bit over 70°. This peak likely occurs because cuts have the highest probability of intersecting in regions where they can intersect with both their direct and diagonal neighbors. This means that, depending on $\beta_{\max}$, the restrictions imposed on the unit cells by avoiding intersections vary greatly—from almost no restrictions to making it nearly impossible to fulfill them by randomly selecting cuts.

4.2.1 Design restrictions of the kirigami metamaterial

Before moving on to how the design restrictions imposed on the kirigami metamaterial influence the machine learning models, let us first examine their general effect. From a human perspective, the design restrictions imposed on the kirigami metamaterial are quite simple: the cuts should not intersect, as this prevents stress concentrations and disconnected regions within the metamaterial. However, from a mathematical perspective, this restriction has far more profound implications. Intersections are determined not only by the rotation of a single cut but also by that of its neighbors, meaning that this restriction introduces dependencies between the parameters $\beta_{i,j}$ used to describe the kirigami metamaterial. This makes it more difficult to measure similarity between different instances.

When the normalized parameters $\beta_{i,j} \in [0, 1]$ used to describe a mechanical metamaterial are independent, the design space itself is a compact interval $[0, 1]^n$. On a closed interval the Euclidean distance (ED) between their parameters can be used to measure the similarity between two structures. In a n-dimensional space $\mathbb{R}^n$, the ED between two samples $\mathbf{x}$ and $\hat{\mathbf{x}}$ is calculated as follows:

$$D_E(\mathbf{x}, \hat{\mathbf{x}}) = \sqrt{\sum_{i=1}^{n} (\mathrm{x}_i - \hat{\mathrm{x}}_i)^2} \tag{4.1}$$

Dependencies between parameters, however, cause non-admissible regions in the compact interval that would otherwise represent the design space. This diminishes the ability of the ED to accurately measure distances between structures and furthermore means it can no longer be used for interpolation. Figure 4.3a illustrates this principle by showing three different configurations for two neighboring cuts with the following angles: **A)** [5°,4°], **B)** [−5°,−3°], and **C)** [65°,−45°], with an angle of 0° corresponding to a vertical cut. When using the ED to measure similarity between these structures it yields $D_E(\mathbf{A}, \mathbf{B}) < D_E(\mathbf{A}, \mathbf{C}) < D_E(\mathbf{B}, \mathbf{C})$, since:

$$\begin{aligned} D_E(A, B) &= \sqrt{(5+5)^2 + (4+3)^2} \approx 12.206 \\ D_E(A, C) &= \sqrt{(5-65)^2 + (4+45)^2} \approx 77.466 \\ D_E(B, C) &= \sqrt{(-5-65)^2 + (-3+45)^2} \approx 81.633 \end{aligned} \tag{4.2}$$

Nevertheless, when examining Figure 4.3b, it can be observed that the magenta line, which represents the ED-based linear interpolation between **A** and **B**, crosses

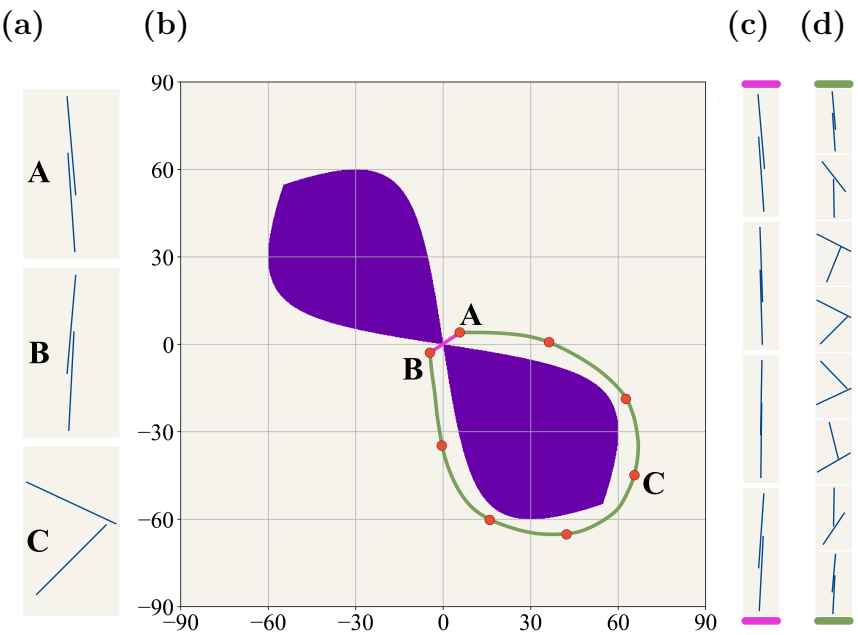

Figure 4.3: Suitability of the Euclidean Distance for two cuts. **(a)** Three different configurations (**A:** [5°,4°], **B:** [−5°,−3°] **C:** [65°,−45°]) of adjacent cuts with unit length between centers and length of $\sqrt{3}$. **(b)** The design space for the considered system with two cuts. Purple zones correspond to the angle pairs of intersecting cuts. Magenta and green lines show two possible routes between **A** and **B**. **(c)** Sequence of cut positions corresponding to direct transition from **A** and **B** (magenta path). **(d)** Sequence of cut positions for detour path shown by green line. Note passing configuration **C** on a route from **A** to **B**.

the dark purple non-admissible zone. This zone indicates pairs of angles where two neighboring cuts intersect. Consequently, despite **A** and **B** being free of intersections, the same does not hold for all intermediate configurations between them. When staying inside the admissible design space, the path between **A** and **B** becomes much longer, as shown by the green line in Figure 4.3b. This path also passes through **C**, meaning that a metric D taking intersections into account should yield $D(\mathbf{A},\mathbf{C}) < D(\mathbf{B},\mathbf{C}) < D(\mathbf{A},\mathbf{B})$, which is quite different from the estimate given by the ED. This matter becomes even more complicated when taking more of the neighboring cuts into account. Therefore, despite seeming initially straightforward, introducing this restriction on intersections between cuts severely influences the design space and prevents measuring the similarity between cuts using the ED.

To investigate how the presence of inadmissible zones in the parameter space and the inadequacy of the ED to measure similarity between unit cells influence the generation of kirigami metamaterials, three different values for $\beta_{\max}$ were chosen. First, 20°, where the design space forms a compact interval and the ED can be

used to measure similarity without limitations. Second, 90°, the fully random case, where large inadmissible zones exist in the parameter space and the ED is not admissible for measuring similarity. And finally, 60°, where the average number of intersections is even slightly higher than for 90° (see Figure 4.2a) and, when considering a cut and its direct neighbor, the fraction of the inadmissible zone within the parameter space increases to 18.7% compared to 16.5% in the 90° case. However, although the inadmissible zone is larger, it is mostly confined to the edges of the design space, making the ED mostly applicable. For the same example as before with just the two neighboring cuts, for $\beta_{\max} = 60°$ only 3.5% of the straight paths connecting two random points within the admissible design space pass through the inadmissible zone, meaning that for all other combinations the ED can be used to measure similarity. For $\beta_{\max} = 60°$, this value is 23.7%.

4.3 Property Prediction

4.3.1 Machine learning approach

Selecting a good predictive neural network model highly depends on how the processed data is structured. In the case of the investigated kirigami metamaterials, the representation in question is a 6×6 matrix containing the $\alpha_{i,j}$ that denote the rotations of the cuts. Furthermore, in terms of the resulting mechanical properties, these representations are translation invariant, as the simulations assume that the metamaterial is infinitely periodic via periodic boundary conditions. Such grid-like, translation-invariant data structures are exactly what Convolutional Neural Networks (CNNs) are designed for.

Network architecture

To predict the Poisson's ratios of the kirigami structures, the neural network model $\mathcal{F}$ depicted in Figure 4.4 was chosen. It consists of four convolutional layers followed by a fully connected layer. This relatively simple architecture was chosen as the 6×6 array is a very small input for a CNN, even compared to older image datasets like MNIST (28×28) [209] and ImageNet (256×256) [210]. All four convolutional layers were followed by a Batch Normalization layer [211] and a LeakyReLU activation function, while a Dropout [212] layer was placed before the fully connected layer to prevent overfitting.

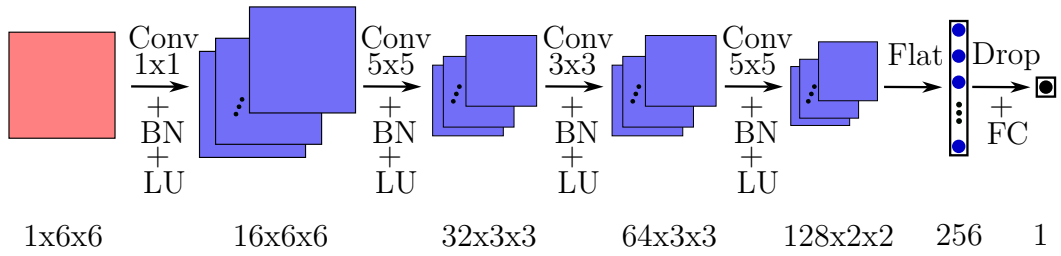

Figure 4.4: The architecture of the forward model $\mathcal{F}$, which predicts the apparent Poisson's ratio for a unit cell of a kirigami metamaterial. The different layers are color-coded: red indicates the input unit cell, black represents the output Poisson's ratio, and blue corresponds to the hidden layers. The network combines fully connected (FC) and convolutional (Conv) layers with dropout (Drop), batch normalization (BN) and LeakyReLU (LU) activation functions.

Training process

To evaluate the influence of the design restrictions on the neural network models, three instances of $\mathcal{F}$ were trained on three separate datasets, each for one of the three $\beta_{\max}$ values chosen in Section 4.2.1. These datasets contain pairs $(\mathbf{x}^{(i)}, \mathrm{y}^{(i)})$ consisting of the unit cell of a kirigami metamaterial $(\mathbf{x}^{(i)})$ and the corresponding Poisson's ratio $(\mathrm{y}^{(i)})$. For the first dataset, $\mathcal{X}_1$, the rotations were limited to $20°$, meaning there are no design restrictions besides limiting each angle value individually. For $\mathcal{X}_2$, the rotations were limited to $60°$, representing an intermediate case between no restrictions and fully random cuts. Finally, for $\mathcal{X}_3$, with $\beta_{\max} = 90°$, the rotations are fully random, imposing complex design restrictions on the kirigami unit cells.

The apparent Poisson's ratio labels for the structures in all three datasets were obtained through simulations in the finite element analysis (FEA) software package COMSOL 5.4a. The metamaterial was modeled as a quadratic 2D sheet featuring rectangular cuts with semicircular fillets at both ends. For the analysis, quadratic serendipity elements were employed under plane strain conditions. To account for the periodic nature of the metamaterial, periodic boundary conditions were applied in the simulations. This was mirrored in the CNN by using circular padding in the convolutional layers. The apparent Poisson's ratio was measured as the ratio between lateral and axial deformations. Since the metamaterial is anisotropic due to the cuts, this measurement corresponds to only one component of the compliance matrix. Nonetheless, as we will see later, even predicting this single value presents a significant challenge for neural networks. Each dataset comprised $3,300$ samples, with 90% used for training and the remaining 10% as the test set. The apparent Poisson's ratio was normalized to the interval $[0, 1]$,

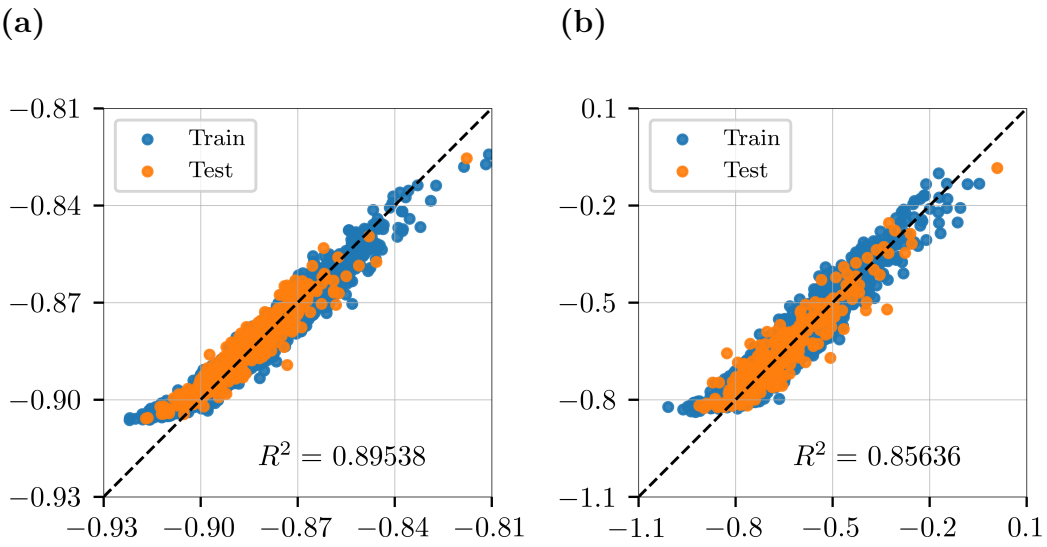

Figure 4.5: Comparison between the apparent Poisson's ratios obtained by FEA (ν_s) and the forward NN model $\mathcal{F}$ (ν_n) for training and testing data for $\mathcal{X}_1$ with $\beta_{\max} = 20°$ **(a)** and $\mathcal{X}_2$ with $\beta_{\max} = 60°$ **(b)**.

while the rotation angles were linearly projected to the interval $[-1, 1]$, where 1 corresponds to $(\beta_{\max}+90°)$ and -1 to its negative counterpart. For training $\mathcal{F}$ the MSE between the apparent Poisson's ratio predicted by the neural network and the label obtained by the simulation was used as the loss function. Furthermore, for all datasets training was performed for 4000 epochs with a learning rate of 1e-4, a batch size of 16 and Adam [94] as the optimizer. Also, L_2 regularization [213] was used to improve generalization.

4.3.2 Results

Figure 4.5a shows the comparison between the apparent Poisson's ratios obtained by the FEA (ν_s) and those predicted by the forward model $\mathcal{F}$ (ν_n) for both the training and test sets of $\mathcal{X}_1$, where $\beta_{\max} = 20°$. It can be seen that the model has learned the correlation between the angle values and the resulting properties ($R^2 = 0.93566$), with the data points loosely stretching along the 45° line that represents the ideal fit. Furthermore, generalization can be observed, as a similar fit is obtained for the test data ($R^2 = 0.89538$).

Figure 4.5b shows similar results for $\mathcal{X}_2$, with $\beta_{\max} = 60°$, although the value range of the apparent Poisson's ratio has changed. Less negative Poisson's ratio can be achieved. At the same time the quality of the fitting on the training has slightly decreased ($R^2 = 0.9017$), with the same for the test data ($R^2 = 0.85636$).

This is very different for $\mathcal{X}_3$, with $\beta_{\max} = 90°$. Figure 4.6a shows a much weaker fit

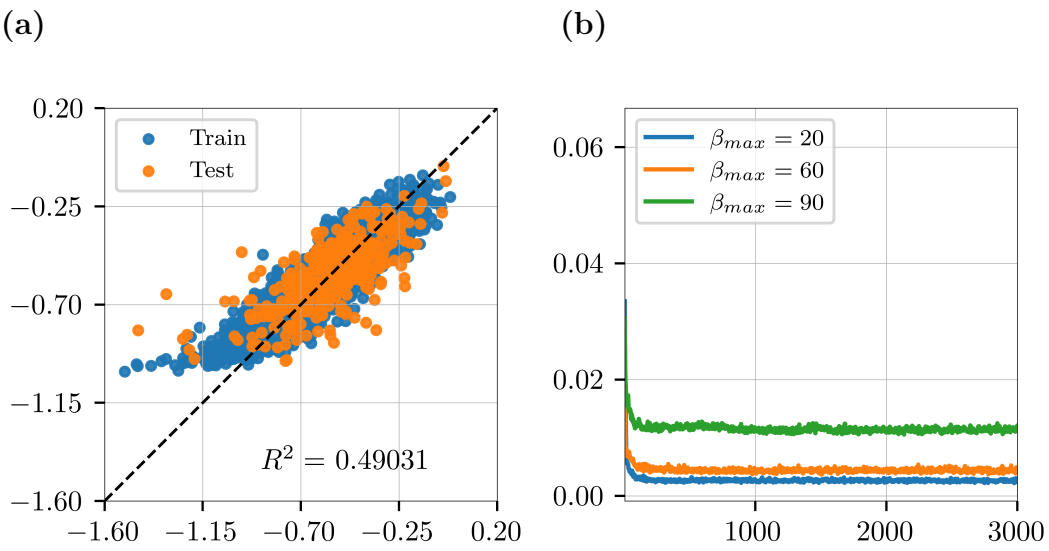

Figure 4.6: Comparison between the apparent Poisson's ratios obtained by FEA (ν_s) and the forward NN model $\mathcal{F}$ (ν_n) for training and testing data for $\mathcal{X}_3$ with $\beta_{\max} = 90°$ **(a)** and the evolution of the MSE-loss during training on the test sets for $\mathcal{X}_1$, $\mathcal{X}_2$ and $\mathcal{X}_3$. An averaging over five epochs was used for curve smoothing **(b)**.

between the apparent Poisson's ratios obtained by the FEA and those predicted by the forward model ($R^2 = 0.79157$). In particular, the generalization of the model to the test data is reduced ($R^2 = 0.49031$), even though Figure 4.6b shows the loss on the test set converging. Furthermore, it can be seen that while there is a small difference in the losses for $\beta_{\max} = 20°$ and $\beta_{\max} = 60°$, there is a much larger difference between these two cases and $\beta_{\max} = 90°$. This suggests a more profound difference between these cases.

4.3.3 Conclusion

While the predictive model $\mathcal{F}$ has shown the ability to reliably predict the apparent Poisson's ratio of the kirigami metamaterial and generalize this beyond the training data when the ED is mostly applicable ($R^2 > 0.85$), it has shown much more trouble to do so when it is not ($R^2 = 0.49031$). At this point, let us remember the theorem by Cybenko [85] (see also Section 2.3.1). This theorem proved that neural networks are capable of approximating continuous functions from a compact set to the real numbers. This is exactly what we are trying to approximate in the case of $\beta_{\max} = 20°$, and it is quite close to the case of $\beta_{\max} = 60°$, with the ability to measure distances via the ED being a core feature of compact intervals. So these two are relatively easy cases for a neural network. For $\beta_{\max} = 90°$, on the other hand, the design space does not form a compact

interval and it can be assumed that the inadmissible zones break the continuity of the property function. This makes it at least a lot more difficult to approximate the function predicting the properties from the unit cell.

While this disadvantage might be overcome through more training data, another approach might be better. More training data also means more computationally expensive FEA simulations for predicting the mechanical properties. So instead of learning to predict the properties first, as was done for the curved-beam metamaterial (see Chapter 3), in this case it might prove useful to first learn the design restrictions and, with them, a mapping to a complex interval in the latent space. Once this is done, a mapping between the embeddings in the latent space and the properties can be learned.

4.4 Learning design restrictions

4.4.1 Generative Models and design restrictions

One of the great strengths of Generative Models (see Section 2.4.2) is often described as having the ability to learn design constraints. By training on a dataset filled with examples that fit these constraints, Generative Models can learn to generate data that also adheres to them. However, more precisely, they possess the ability to generate data similar to the training examples, with ”similar” most often interpreted as also adhering to the design restrictions. For this to work, there needs to be a mathematical understanding of what ”similarity” means in the context of the data. There are two main approaches to this, which can also be combined. The first approach views the training and generated data as probability distributions over the design space, with similarity measured as the difference between these two distributions. In the second approach, similarity is assessed directly through comparisons between two samples, requiring the selection of a sample-to-sample similarity metric. The most commonly used metric for this is the ED, which might impact the generation of kirigami metamaterials, as we have established that it is not suitable for assessing similarity for the examined kirigami metamaterials (see Section 4.2.1). This suggests that generative models relying on the sample-to-sample approach for measuring similarity might not work for kirigami metamaterials, at least when using the ED.

To investigate if this dependency on the ED has an influence on how well generative models handle kirigami metamaterials, it is important to establish first which approaches rely on it and which don’t. We will do so for four of the

models introduced in Section 2.3.2, namely VAEs, GANs, WGANs and Diffusion models.

VAEs

As introduced in Section 2.3.2, the loss function for training the VAE has two parts, the regularization loss and the reconstruction loss. The regularization loss is usually defined by the Kullback–Leibler (KL) divergence [214] between the target and generated sample distributions, thereby clearly falling into the category of distribution-based similarity measures. The reconstruction loss on the other hand uses either the Binary Cross-Entropy (Eq. 2.47) for discrete problems, or the Mean Squared Error (MSE) (Eq. 2.48) for continuous ones [105]. With the MSE, which is actually the square of the ED, the loss of the VAE (see Eq. 2.49) becomes:

$$\mathcal{L}_{\text{VAE}}(\mathbf{x}, \hat{\mathbf{x}}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \underbrace{D_E(\mathbf{x}, \hat{\mathbf{x}})^2}_{\text{reconstruction loss}} + \beta \underbrace{D_{\text{KL}}\big(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}), \mathcal{N}(0, 1)\big)}_{\text{regularization loss}} \tag{4.3}$$

where $\mathbf{x}$ is the original input, $\hat{\mathbf{x}}$ the reconstructed one and $\boldsymbol{\mu}, \boldsymbol{\sigma}$ are the mean and variance of the learned distribution of $\boldsymbol{x}$ in the latent space, while κ is a parameter controlling the trade-off between the two losses.

GANs

Analyzing the influence of the ED on the GAN loss is quite challenging in its two-player game formulation (see Eq. 2.51 and Eq. 2.52). Fortunately, it has been demonstrated that this two-player game is equivalent to minimizing the Jensen–Shannon (JS) divergence between the distribution of the data generated by the generator $\mathcal{G}$ and the distribution of the training data [114]:

$$\mathcal{L}_{\text{GAN}} = D_{\text{JS}}\big(p(\mathbf{x}), p(\hat{\mathbf{x}})\big) = D_{\text{KL}}\big(p(\mathbf{x}), p(\hat{\mathbf{x}})\big) + D_{\text{KL}}\big(p(\hat{\mathbf{x}}), p(\mathbf{x})\big) \tag{4.4}$$

As a symmetric extension of the KL divergence, the JS divergence is a metric between two probability distributions, which means the GAN should be able to learn intersection avoidance.

WGANs

The main difference in training a GAN and WGAN is, that for training the WGAN the JS divergence used for training the GAN has been replaced by the Wasserstein distance (also called Kantorovich–Rubinstein metric or Earth mover distance) to measure the discrepancy between the distributions of the generated and the training data. This metric, first introduced by Kantorovich [215], is based on the principle of optimal transport. The compared probability distributions are seen a distribution of mass in the design space, while the distance between them is measured as the minimal cost of shifting the mass of one distribution, so it resembles the other. While the Wasserstein distance has multiple advantages over the JS divergence, such as always staying finite, it relies on an underlying distance to measure how far the mass has been transported. For this, the WGAN usually used the ED:

$$\mathrm{Wass}_1(p(G(\mathbf{z})), p(\mathbf{x})) = \inf_{\pi \in \Pi(G(\mathbf{z})), p(\mathbf{x}))} \mathbb{E}_{(X_1, X_2) \sim \pi} D_E(X_1, X_2) \tag{4.5}$$

Although the WGAN uses a similarity metric between probability distributions, it still relies on a sample-to-sample similarity metric to learn how to generate data.

DDPMs

Training of Diffusion models is done by learning to reverse a process that adds gradually noise to a sample (see Section 2.3.2). This is usually done by performed by minimizing the variational upper bound on the negative log-likelihood, which can be fully expressed as a difference between the actual and expected noise that was added to the sample (see Eq. 2.75):

$$L(\Theta) \coloneqq \mathbb{E}_{t, \mathbf{x}_0, \boldsymbol{\epsilon}} \left[\left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\Theta(\mathbf{x}_t, t) \right\|^2 \right] \tag{4.6}$$

To measure this difference, the ED is used. However, this formulation is a simplification of the variational upper bound expressed through KL divergences (see Eq. 2.72). Furthermore, the ED is not used to measure similarity between samples, but the added noise, for which is suitable by construction. Therefore, the training of DDPMs only depends on the similarity between the target and generated probability distributions and should not suffer from the inability to use the ED to measure sample-to-sample similarity.

4.4.2 Implementation and network architectures

The PyTorch deep learning framework [216] was chosen for the implementation of all four generative models—the VAE, GAN, WGAN, and DDPM. This followed the same reasoning as choosing PyTorch for the predictive models (see Section 4.3.1). PyTorch allows efficient implementation of CNNs with periodic padding, which were chosen for the kirigami metamaterial given its periodicity and grid-like structure, where intersections primarily depend on the direct neighbors of a cut.

To keep models comparable, the same architecture was used for the generators of both GAN architectures as well as for the decoder of the VAE. This means that the generative models produced by the three frameworks differ only in their weights. Similarly, the discriminator architectures for the GAN and WGAN were kept nearly identical, with the only difference being the addition of a sigmoid activation function for the GAN due to different value range requirements imposed by the objective functions.

To stabilize GAN training, the architectures themselves were mostly based on the DCGAN framework [217], including the use of Batch Normalization in the generators, decoders, and also the VAE's encoder to minimize batch internal dependencies [115]. Furthermore, the two-timescale update rule (TTUR) [118], in which different learning rates are used for the generator and discriminator (10^{-5} and $5 \cdot 10^{-4}$, respectively), was applied for the same purpose.

As DDPMs are limited to latent spaces that share the same dimensions as the data, a different architecture was needed for it [129]. As standard for diffusion models, a modified version of the convolutional U-Net architecture [218] was chosen there, with filter numbers similar to the other generative network to assure comparability.

For each model, three instances corresponding to $\beta_{\max}$ values of 20°, 60°, and 90° were trained. Each instance was trained for 3000 epochs with a batch size of 32 using the Adam optimizer [94]. Detailed descriptions of the network architectures are provided in Appendix A.2.

4.4.3 Results

When examining the capability of generative models to learn the design restrictions imposed on the kirigami metamaterials, we are interested in two factors: First, of course, the general ability of the model to produce metamaterial structures without intersections, for which we will look at the average number of intersections in the generated structures. The second factor is the extent to which the generative

model can reproduce the probability distribution of the cuts and, in particular, learn the dependencies between a cut and its neighbors, as these dependencies determine whether the ED can be used to measure similarities.

For this, we will analyze 2D histograms that represent the number of instances where a random cut and its bottom-side neighbor possess a specific combination of angles $(\beta_{i,j},\ \beta_{i+1,j})$. If the histogram for the generated data matches that of the training data, it can be concluded that the model is capable of capturing the dependencies between neighboring cuts.

One additional detail about these histograms must be noted. Since cuts may have different orientations for the same added rotation depending on their position in the grid (as seen in Figure 4.1b), the first elements of these angle pairs are always taken from cuts at positions corresponding to vertical cuts in the initial undisturbed sample for clarity.

Euclidean Case

Setting the maximum absolute value for the added rotations, $\beta_{\max}$, to 20° automatically ensures that no intersections occur, as there are no angle combinations for which neighboring cuts intersect while still adhering to these restrictions. This also means that the angle values can be chosen independently from each other and that the design space can be represented as a compact interval. Therefore, in this case, the ED can be used to measure the similarity between different kirigami structures.

Figure 4.7a shows that in this case, the VAE, GAN, and DDPM are able to almost completely avoid intersections after a few epochs, while the WGAN, despite poor stability, manages to reduce the average number of intersections to below 0.1 after 500 epochs. To avoid intersections, the model must only learn to restrict the added rotations to the same range as the training data. Figure 4.7b shows this for the DDPM. After 3000 epochs, the DDPM has learned to imitate the distribution of the $\beta_{i,j}$ in the training data, which was drawn from a uniform random distribution. However, there is some deviation around the borders of the admissible zone. Some designs lie outside of the admissible zone, while the number of designs inside the admissible zone close to the border is reduced.

A similar effect can be observed in the histograms in Figure 4.8, which illustrate the relationship between the added rotation of a cut and its bottom-side neighbor $(\beta_{i,j},\ \beta_{i+1,j})$. If the trained model has correctly learned the dependency between neighboring cuts, the distributions in the generated histograms should match those of the training data. All four models learn that angle values should be uniformly

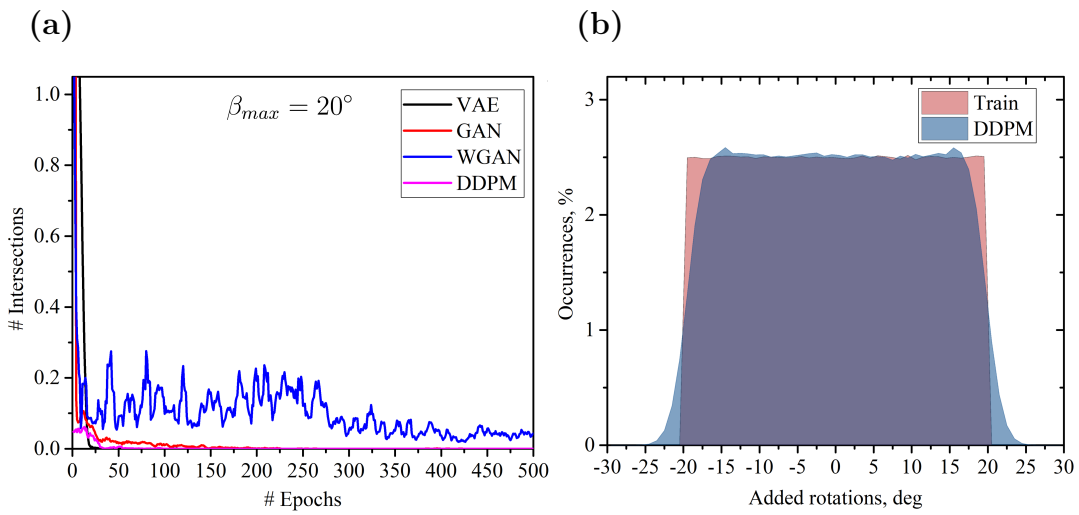

Figure 4.7: Training of models for $\beta_{\max} = 20°$. **(a)** The evolution of the average number of intersections during training for unit cells generated by different machine learning approaches. An averaging over five epochs was used for curve smoothing. **(b)** Distribution of the cuts with the specific added angles $\beta_{i,j}$ in the training dataset for $\beta_{\max} = 20°$ (red) and in the set generated by trained DDPM (blue).

distributed inside admissible zone. However, differences arise in determining the border of this zone and the sharpness of the cutoff. The histograms for the GAN and DDPM show only slight uncertainty regarding the exact placement of the border, while the VAE avoids borderline cases by narrowing the range of generated angles.

This behavior of the VAE is associated with the trade-off between reconstruction and regularization loss in the VAE's loss function (see Eq. 4.3). The presence of the regularization term weakens the reconstruction term, making imperfect reconstructions that are closer to the average value more common, thus leading to a narrowed angle range. This effect persists even after careful hyperparameter tuning. Like the VAE, the WGAN centers designs more toward the middle of the admissible interval. However, there is no clear cutoff between admissible and non-admissible configurations. Instead, there is a smooth transition between these areas, making the distribution resemble a normal distribution more than a uniform one.

Fully Random Case

When the maximum absolute value for the added rotations, $\beta_{\max}$, is set to 90°, intersections can no longer be avoided by looking at every cut individually, the dependencies between them need to be taken into account. That said, compared

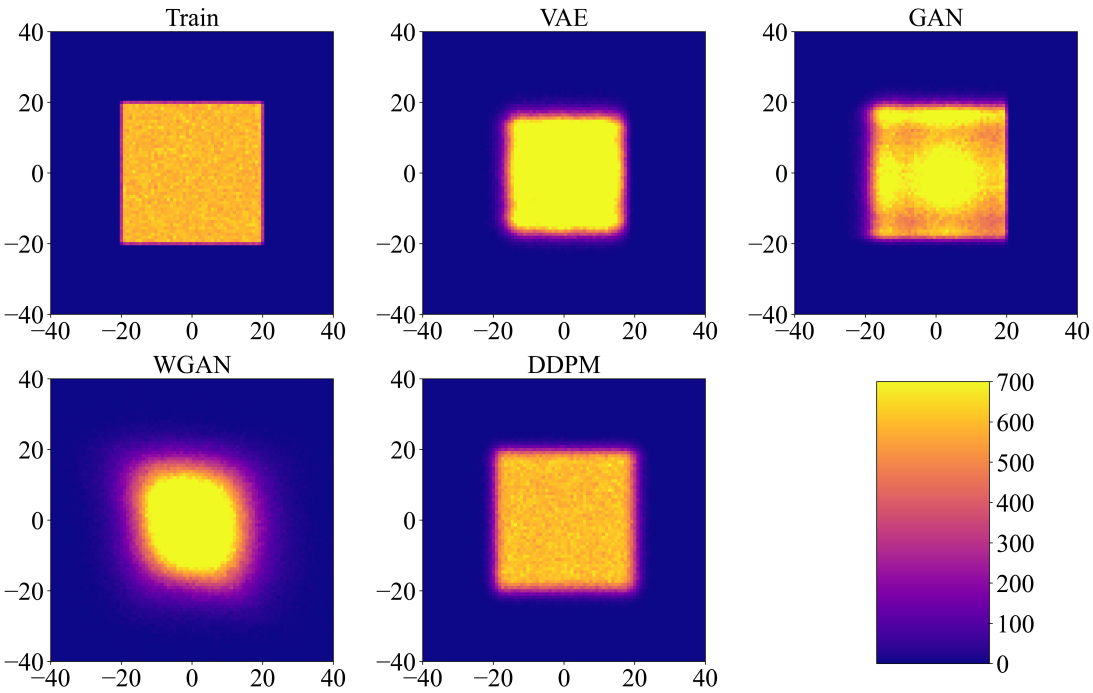

Figure 4.8: Relation between adjacent cuts for $\beta_{\max} = 20°$. 2D histograms show the likelihood of angle combinations for a cut and its bottom neighbor for a maximum absolute value for $\beta = 20°$. Only angles at positions that correspond to vertical cuts in the base structure were chosen as the first elements in pairs..

to drawing the $\beta_{i,j}$ from a uniform random distribution the probability of intersections can still be reduced without taking these dependencies into account.

The reason for this is that the dependencies between the cuts influence the frequency at which certain rotations occur in the dataset, as some rotation values have a lower probability of causing intersections. As a result, angle rotations are not uniformly distributed in the training dataset, and the likelihood of intersections can be reduced by sampling each cut individually from this distribution, even without learning the dependencies between cuts. This difference is illustrated in Figure 4.9a, where the orange dotted line represents the average number of intersections when sampling angles from a uniform distribution, while the black dashed line represents the average number of intersections when sampling from the distribution of angles in the training data. Therefore, to demonstrate that a generative model has learned the dependencies between cuts, it must reduce the average number of intersections below the baseline established by sampling from the distribution of angles in the training data.

Figure 4.9a shows that the VAE does not reduce the average number of intersections below this baseline; instead, it appears to converge toward it. The WGAN manages to reduce the average number of intersections below this baseline but

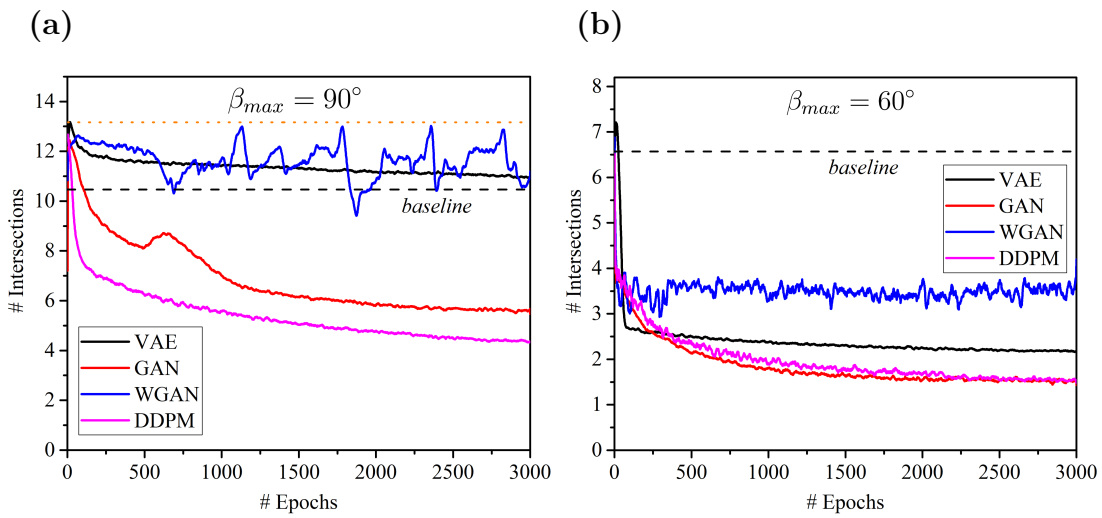

Figure 4.9: Training of models for $\beta_{\max} = 90°$ **(a)** and $\beta_{\max} = 60°$ **(b)**. The evolution of the average number of intersections during training for unit cells generated by different machine learning approaches. An averaging over 20 epochs was used for curve smoothing. The dashed line corresponds to the average number of intersections when sampling all angles of a unit cell randomly and independently from the angle distribution of the training dataset.

fails to converge, making this success short-lived. This failure to converge is a common drawback of WGAN [219]. In contrast to the VAE and WGAN, the GAN and DDPM lower the average number of intersections significantly below the baseline, although both still fail to generate only completely intersection-free samples.

Similar differences can be observed when looking at the 2D histograms (Figure 4.10) showing the dependencies between a cut and its bottom neighbor. The histogram for the training data shows two dark regions with zero occurrence corresponding to angle pairs where the two cuts intersect (compare with Figure 4.3b). For angle combinations that do not cause an intersection between the two examined cuts, the frequency differs greatly, depending on how likely intersections with other neighbors are. It can be seen that the VAE and WGAN struggle to learn these dependencies, with both generating angle pairs inside the non-admissible zones. On the other hand, the GAN and DDPM manage to capture the design space and demonstrate an understanding of the constraints, with the DDPM outperforming the GAN.

Intermediate Case

Setting the maximum absolute value for the added rotations, $\beta_{\max}$, to 60°, creates an intermediate case between the 20° case, where the design space is a compact

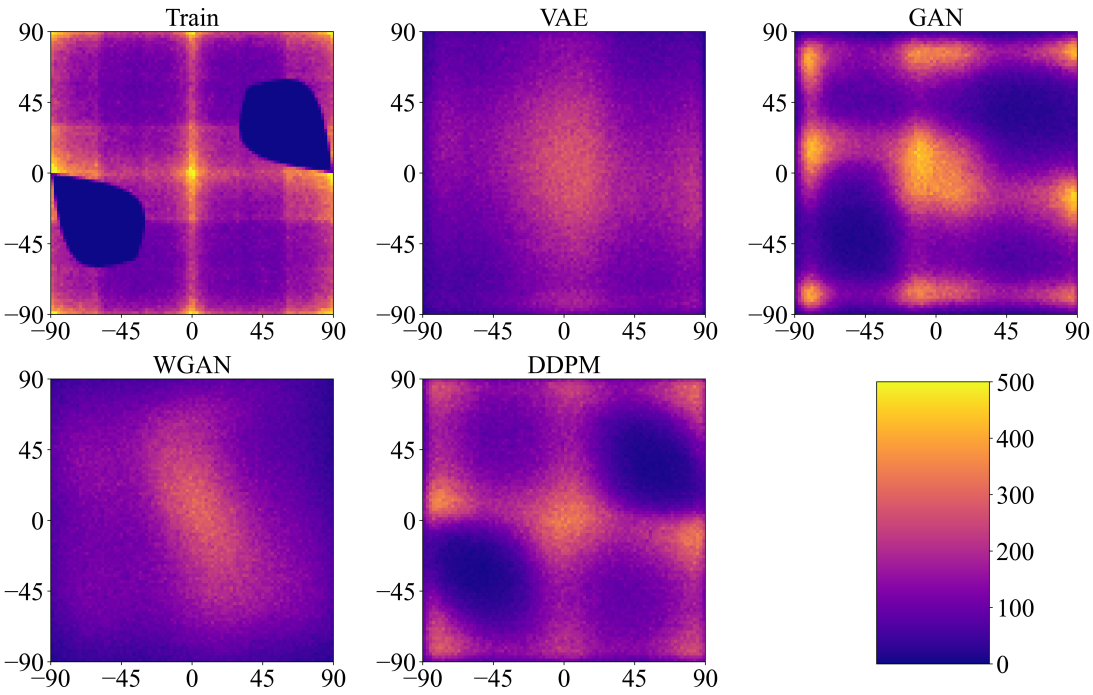

Figure 4.10: Relation between adjacent cuts for $\beta_{\max} = 90°$. 2D histograms show the likelihood of angle combinations for a cut and its bottom neighbor for fully random rotations of the cuts. Only angles at positions that correspond to vertical cuts in the base structure were chosen as the first elements in pairs.

interval, and the 90° case, where the rotations are fully random. For this $\beta_{\max}$, there are non-admissible zones inside the design space, but the ED can roughly be used to measure similarity between samples. Figure 4.9b shows that in this case, while the GAN and DDPM still perform better than the VAE and WGAN, all four models are capable of reducing the average number of intersections significantly below the baseline, for which angles were sampled individually from the angle distribution of the training dataset.

Again, even the DDPM, which performed best was not able to achieve a 100% success rate in generating intersection-free configurations, with an average of 1.7 intersections per sample observed. What is more important though is the increase of the share of generated samples without intersections. For sampling from the uniform distribution only about three structure out of a million have no intersections (success rate less than 0.001%). This increases to around 0.5% of the structures when sampling from the training distribution and approximately 25% success rate for the GAN and DDPM, while it is around 10% for VAE and WGAN.

4.4.4 Conclusion

While all four investigated generative models—the VAE, GAN, WGAN, and DDPM—are able to learn to avoid intersections when the design space of the kirigami metamaterials is a compact interval ($\beta_{max} = 20°$), for kirigami metamaterials where the ED cannot be used to measure similarities between samples, this is not the case. The VAE and WGAN, in particular, depend on this similarity measure for effective generation. In the case where inadmissible zones in the parameter space render the ED inapplicable ($\beta_{max} = 90°$), both the VAE and WGAN failed to outperform the baseline established by randomly sampling from the angle distribution of the training dataset. However, when similar inadmissible zones exist but the ED remains mostly applicable ($\beta_{max} = 60°$), the VAE and WGAN significantly outperform the baseline. This means that it is the reliance on the ED, rather than the presence of inadmissible zones, that impacts their performance.

The GAN and DDPM—neither of which rely on the ED in their objective—demonstrated in both scenarios a greater potential to learn design space restrictions from the training data. They were not only able to achieve a greater reduction of intersection, but show a very distinct behavior from the other two models, further asserting the critical role the admissibility of the ED plays in the design of kirigami metamaterials. Nevertheless, while these two managed to outperform the VAE and WGAN, both still fail to achieve a 100% success rate in generating intersection-free configurations. So even if the generative models do not rely on the ED, there are other effects preventing them from fully capturing the design restrictions and making further investigation necessary.

4.4.5 Possible Solutions

There are several techniques that might help improve the ML-based generation of the kirigami structures. Generally, they fall into one of two categories. The first category comprises methods that help in finding the exact boundary of the valid space. Most of these focus on incorporating negative data into the training process (see Section 2.3.2). Adding negative data allows one to purposefully penalize data outside the admissible space. When only valid data is used for training, it matters more how well the probability distributions of the training and generated data match in general, and less whether generated designs lie outside or inside the admissible space. However, even if negative data is included, edge cases—where two cuts barely intersect—might not be present in the training data, making it impossible for the model to determine whether they are admissible

or not. Active learning is concerned with selecting new samples to be added to the dataset so that the information gain is maximized [220], allowing to add these edge cases to the dataset which is the only way to make it possible to decide these cases. Various active learning approaches have already been successfully applied to improve data collection in engineering applications, such as Gaussian Processes [221] and t-METASET [222].

The other possible solution besides improving the learning of the decision boundaries is to further reduce the assumptions made about the data by the generative models. While they do not assume that the ED can be used to measure similarity between samples, the GAN and DDPM do assume that the data lies on a low-dimensional Euclidean manifold. If this assumption is not fulfilled, GAN and DDPM are not able to completely learn to avoid intersections. Most efforts in this area focus on improving diffusion models so that they can map to more complex manifolds, thus increasing the range of functions they can represent [223–225].

4.5 Conclusion on kirigami metamaterials

The design space limitations imposed on kirigami metamaterials by avoiding intersecting cuts have a profound impact on both property prediction and generative modeling via neural networks. For property prediction, these design restrictions result in a more complex function relating the unit cell to its resulting properties, which makes approximation with a neural network more challenging. While this might be offset by increasing the amount of training data, the additional simulations required to obtain the properties for this data tend to be computationally expensive and should be avoided. An alternative approach is to first learn a mapping to a better representation of the designs in a latent space, and then learn to predict properties for those representations before moving on to conditional generation.

However, the design restrictions also impact the generative models that are candidates for learning them. The main factor is that the restrictions can make the classical Euclidean distance (ED) inapplicable as a metric for assessing the similarity between unit cells. This especially impacts the VAE and WGAN, which rely on this similarity measure for effective generation. This makes these models less suited for learning complex design restrictions that lead to a scattered design space, but at the same time improves stability and lowers computational costs when parameters are independent. on the contrary, the GAN and DDPM-who do not rely on the ED- show better capabilities to learn the design restrictions,

but at the cost of less stability (GAN) or greater computation time (DDPM). Still, even these models fail to fully capture the restrictions, suggesting there are additional factors influencing how they learn them.

These challenges in learning design restrictions have an impact on studies that use machine learning for mechanical metamaterials. While in image generation generative models are often used to derive design constraints from training data, mechanical metamaterials are frequently elaborately constructed to yield ”nice” parameterizations, where the parameters are continuous and independent.

Chapter 5

Heterogeneous metamaterials

.

5.1 Background

5.1.1 Literature Review

While traditional mechanical metamaterials are typically constructed from a single unit cell repeated periodically in two or three dimensions, there has recently been a shift towards the development of heterogeneous metamaterials, which combine different unit cell geometries within a single structure [45]. The variation in unit cells introduces a certain level of disorder into the metamaterial, similar to what is observed in structured materials found in nature [226, 227]. This disorder can enhance mechanical performance compared to fully periodic systems [228].

Naturally, introducing disorder is not the only way to incorporate multiple unit cells within a single structure. Heterogeneous metamaterials with enhanced properties have also been built by mirroring ordered structures, such as architected composites [229] or the microstructures found in natural crystals [230]. Both disordered and ordered heterogeneous metamaterials have been shown to lead to notable improvements in various mechanical properties, including fracture toughness [231], plastic energy absorption [232], stress distribution [233], damage tolerance [230, 234], and the ability to control the shape a material takes under load [37].

However, the most common approach to combining multiple unit cells is through functionally graded metamaterials (FGMMs). Functional grading refers to a smooth spatial transition of material properties [235]. Initially developed independently of metamaterials, with a focus on composition at the microstructural

level [236, 237], the concept of functional grading has since expanded to porous structures [238] and metamaterials [239]. Among other applications, FGMMs have been used to control thermal conductivity [240], reduce fatigue fractures [241], and improve energy absorption [242].

While heterogeneous metamaterials have been shown to exhibit a wide range of useful mechanical properties, their design can be challenging, as it requires a detailed understanding of how unit cell geometries influence the resulting material behavior. For this reason, machine learning has become increasingly common in the design process. Its most typical application is to identify unit cells that match a desired distribution of local properties. To achieve this, machine learning models are used to establish a continuous relationship between geometrical parameters and local material properties [103, 163, 243]. However, it is also possible to use machine learning to determine the local property distribution itself in order to achieve a specific global behavior [45]. For example, Jia *et al.* [233] combined machine learning with topology optimization to design a heterogeneous metamaterial that modulates stress distribution in critical regions of bones.

5.1.2 Motivation

One of the advantages of heterogeneous metamaterials that has recently gained attention is their ability to be designed for specific deformations under load [244]. This *shape-morphing* behavior has proven useful in deployable structures [245], robotics [4, 246], and biomedical applications [247]. However, these shape changes depend on both the individual unit cells and their spatial arrangement within the heterogeneous metamaterial, which can make their design challenging. Designing such materials can be framed as the inverse problem to the forward problem of predicting the shape resulting from a given arrangement of unit cells. While this problem can, in some cases, be solved analytically if the geometrical parameters are chosen accordingly [37], it generally requires more elaborate methods. A common approach is to minimize the difference between the target and exhibited properties [36, 45], often through topology optimization [244]. Nevertheless, this process is typically computationally expensive, which is why machine learning is often used to assist—either by learning the relationship between geometry and local properties [103, 163, 243], or by serving as a surrogate model for shape prediction, enabling efficient gradient computation during topology optimization [233]. Here, however, building on recent successes in training inverse models for mechanical metamaterials [104, 172], we investigate how a Tandem Neural Network (TNN) can be used to directly solve the inverse problem.

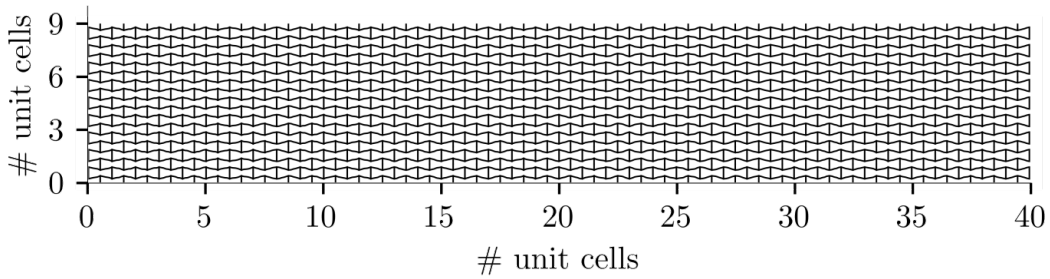

Figure 5.1: A flat, quasi-one-dimensional shape-morphing sheet composed of 40×9 curved-beam unit cells of fixed size.

5.2 The heterogeneous metamaterial

The investigated heterogeneous metamaterial is a 2D structure composed of different unit cells belonging to the class of curved-beam metamaterials introduced in Chapter 3. These unit cells are based on hexagonal re-entrant structures, where the straight, angled horizontal beams have been replaced by curved ones (see Figure 3.3). To ensure that these unit cells can be combined seamlessly into a larger structure, their dimensions were fixed at $2 \times 1.5 \times 0.1$ cm, meaning the properties of the re-entrant unit cells are controlled solely by the curvature added to the vertical beams, while their angles remain fixed.

To form the heterogeneous metamaterial, a 40×9 array of these unit cells was assembled into a flat, shape-morphing sheet (see Figure 5.1), similar to the one presented by Zhang and Krushynska [37]. For simplicity, the unit cells in this sheet are arranged in a quasi-one-dimensional manner, meaning their parameters vary only along the longitudinal direction, while remaining uniform along the vertical axis. The deformation responsible for shape morphing is applied as a fixed displacement along the longitudinal direction.

5.2.1 Shape Prediction

Accurately predicting the morphing of the metamaterial sheets is no simple task, as the boundaries of each layer of unit cells are connected to other layers, and the way these connections are made plays a role. Furthermore, under a displacement-controlled load, the axial deformation of each unit cell depends not only on its own stiffness but also on the stiffness of all other unit cells in the sheet. This means that the deformation of a unit cell is influenced not only by its neighbors but also by the entire arrangement.

Taking this into account, the best way to compute the overall behavior is through Finite Element Analysis (FEA) using the actual unit cells. However, since this is computationally expensive—and the goal here is to demonstrate how machine learning can be used for the design of these quasi-one-dimensional shape-morphing sheets—a simpler approach was used. Instead of using FEA on the full geometry of the sheet to predict the resulting shape, FEA was used only to compute the effective mechanical properties of each unit cell. These properties were then used in a simplified mechanical model to compute the overall shape. This computation was divided into two steps: first, the Young's moduli of the unit cells were used to compute the axial strain; then, the Poisson's ratio was applied to determine the transverse deformation.

For this, the shape-morphing sheet was modeled as a composite material. Since the heterogeneous metamaterial is quasi-one-dimensional in nature, this composite model consists of $N = 40$ stacked layers $l^{(i)}$ that are perpendicular to the loading direction. This allows for the assumption of constant stress between them, $\sigma^{(i)} = \sigma^{(j)} \; \forall \; i, j \in \{1, \ldots, N\}$, which is also equal to the total stress σ_{tot}. Given this, the axial strains of the individual layers $\varepsilon^{(i)}$ can be determined using the total axial strain ε_{tot} and the individual stiffness values $E^{(i)}$ through the Reuss model [248]:

$$\varepsilon^{(i)} = \frac{\varepsilon_{\text{tot}}}{\sum_{j=0}^{N} \frac{E^{(i)}}{E^{(j)}}} \tag{5.1}$$

If it is assumed that each layer deforms independently, then the individual transverse strains $\varepsilon_{\text{trans}}^{(i)}$ can be simply calculated using the Poisson's ratios $\nu^{(i)}$ of the layers. However, unfortunately the transverse strains are not independent here, as the layers are fixed to each other at the connections between the unit cells. To account for this interaction, homogenization was used to compute the mesoscopic strain ε_{mes}. Yvonnet and Bonnet [249] demonstrated that this can be achieved by applying a Gaussian filter to the individual strains via convolution:

$$\varepsilon_{\text{mes}}(x) = \gamma(x) * \varepsilon_{\text{trans}}(x) \tag{5.2}$$

$$\gamma(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-x^2}{2\sigma^2}} \tag{5.3}$$

$$\varepsilon_{\text{trans}}(x) = \sum_i \varepsilon_{\text{trans}}^{(i)} \mathbb{1}_{x \in l_i} \tag{5.4}$$

This approach requires an empirical selection of the parameter σ, which in this case was set to $\sigma = 8$.

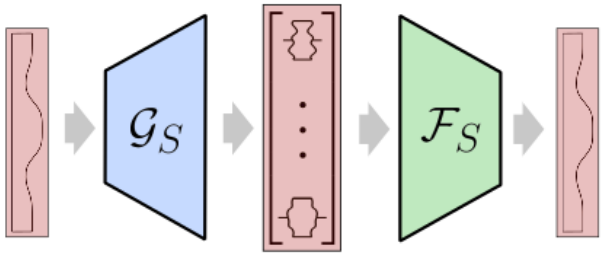

Figure 5.2: The TNN network system used to generate, the quasi-one-dimensional shape morphing sheets. $\mathcal{F}_S$ is a pretrained prediction network, while $\mathcal{G}_S$ generates a sequence of unit cells based on the target shape it has received. For spatial efficiency, the shape-morphing sheets are displayed rotated 90°.

5.3 Shape Matching

Solving the inverse problem of finding a quasi-one-dimensional sequence of unit cells that results in specific shape-morphing behavior requires a good understanding of how the choices of the individual unit cells influence the overall resulting shape of the sheet. As with the shape prediction (see Section 5.2.1), the connections between neighboring unit cells cause dependencies between their deformations, meaning that to achieve a desired shape, it is necessary to design the sequence of unit cells as a whole. To solve this complex inverse problem, a TNN was chosen (see Section 3.3.1).

5.3.1 Network architectures

The TNN, shown in Figure 5.2, consists of a forward neural network, $\mathcal{F}_S$, which predicts the deformation of the sheet, and a generative network, $\mathcal{G}_S$, which generates a sequence of unit cells that produces a target deformation. Since only a single solution is required, no guide was used. Training this TNN requires pairs $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$, where $\mathbf{x}^{(i)}$ is a sequence of unit cells and $\mathbf{y}^{(i)}$ is the corresponding resulting shape. Obtaining the shapes for a set of unit cell sequences is straightforward, as discussed in Section 5.2.1. What is more challenging is determining which unit cell sequences to include in the training dataset. A standard approach would be to uniformly sample the geometric parameters of each unit cell in the sequence. However, this leads to an unbalanced distribution of shapes in the dataset. As shown in Figure 5.3, when uniformly sampling the geometric parameters of the unit cells, the resulting property distribution not only shows a dependency between Poisson's ratio and Young's modulus, but also that some property combinations

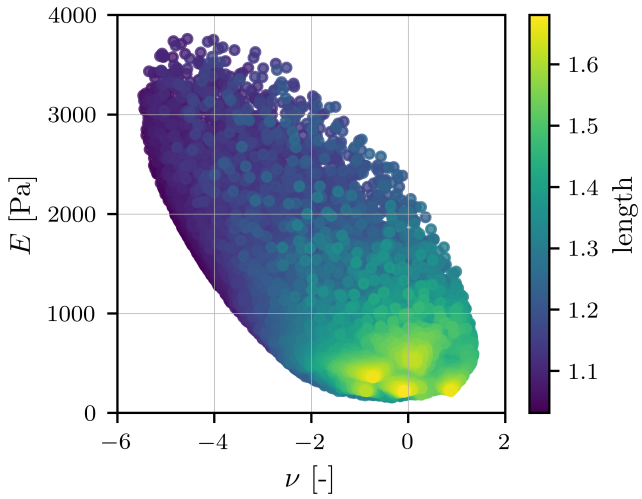

Figure 5.3: Distribution of Poisson's ratios and Young's moduli when sampling the geometric parameters of the unit cells from a uniform random distribution. The color indicates the length of the resulting curve .

are much more likely than others.

This effect is multiplied when randomly selecting 40 unit cells and combining them into a sequence. Since the resulting shape is mainly determined by the mechanical properties of the unit cells, this approach excludes a wide range of possible shapes from the training data—leading to a difference in maximum deformation by a factor of four compared to uniformly sampling in property space. As a result, the generative model is limited to producing only shapes from this reduced distribution. To sample directly from the property space, it is necessary to convert between sequences of unit cells and their associated properties. Fortunately, this can be done using a neural network (see Chapter 3) that processes each unit cell individually. While this switch between geometric parameters and mechanical properties could be done using an inverse model, it is actually simpler to integrate a forward model directly into the TNN architecture.

The forward model

The forward model $\mathcal{F}_S$ of the TNN, which takes a sequence of unit cell geometries as input and predicts the corresponding shape morphing, was built by combining two separate models (see Figure 5.4). First, the model $\mathcal{F}$, which predicts the mechanical properties of an individual unit cell, is used to transform the sequence of geometric parameters—serving as the input to $\mathcal{F}_S$—into a sequence of mechanical properties. This sequence then serves as the input to the second model, $\mathcal{F}_E$ which predicts the shape morphing from it. This has the advantage that $\mathcal{F}$ can be

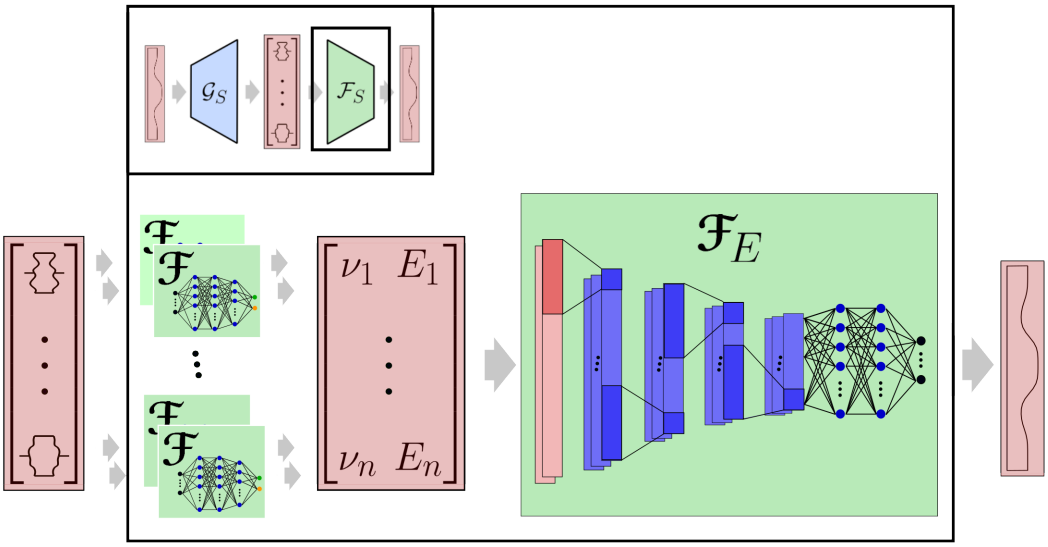

Figure 5.4: The forward model $\mathcal{F}_S$ used to predict the resulting shape under deformation for a sequence of unit cells. This model consists of two separately trained networks: $\mathcal{F}$, which transforms the sequence of geometries into a sequence of material properties, and $\mathcal{F}_E$, which predicts the resulting shape from the sequence of mechanical properties.

trained using FEA data for individual unit cells, while $\mathcal{F}_E$ can be trained directly with data from the property space. A question that may arise here is why go through the geometric design space at all and not stick to the property space. The reason is that this is a simple way to limit the generated property sequences to realizable structures, rather than having to learn these restrictions separately.

For $\mathcal{F}$, a network architecture nearly identical to that in Chapter 3 was used (see Figure 3.4). The only modification was the inclusion of the Young's modulus as an additional output. The network consists of four fully connected layers with a width of 50, 50, 30, and 2, respectively. It receives an five-dimensional vector corresponding to the parameters of the Bézier curve defining the unit cell (see Section 3.2.1), while it has two outputs, one corresponding to the Poisson's ratio, the other to the Young's modulus. Each of the three hidden layers was followed by a layer normalization layer [197] for regularization and reduced training time, as well as a ReLU activation function.

For $\mathcal{F}_E$, the architecture shown in Figure 5.5 was chosen. Since the input consists of a sequence of properties, where the local deformation primarily depends on each unit cell and its neighbors, a 1D Convolutional Neural Network (1D-CNN) was used. The network consists of four 1D convolutional layers with 64, 64, 32, and 8 filters, respectively, followed by two fully connected layers with a width of 256 and 41. The network receives a sequence of 40 unit cells, represented by their Poisson's ratio and Young's modulus, and outputs the transverse displacement of

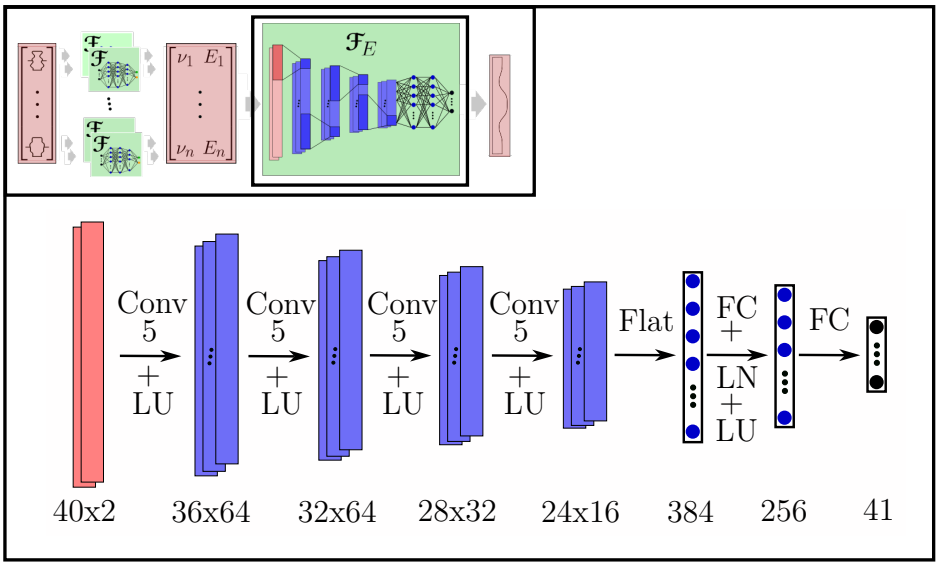

Figure 5.5: The architecture of the forward model $\mathcal{F}_E$, which predicts the shape for a sequence of mechanical properties. The different layers are color-coded: red indicates the input sequence of Poisson's ratios and Young's moduli, black represents the output vector containing the y-coordinates of the deformed shape, and blue corresponds to the hidden layers. The network combines fully connected (FC) and convolutional (Conv) layers with layer normalization (LN) and LeakyReLU (LU) activation functions.

the sheet at 41 positions. Each convolutional layer uses a kernel of size 5 and is followed by a LeakyReLU activation function. The hidden fully connected layer is followed by a layer normalization layer [197], as well as a LeakyReLU activation function.

To combine these two models into $\mathcal{F}_S$, a split layer is first applied to decompose the input sequence into individual unit cells. These unit cells are then processed in parallel by $\mathcal{F}$ and recombined into a sequence using a concatenation layer, before the resulting sequence of properties is fed into $\mathcal{F}_E$.

The inverse model

The inverse model $\mathcal{G}_S$ of the TNN, shown in Figure 5.6, receives a target shape as input and outputs a corresponding sequence of unit cells. The target shape is first processed by a fully connected layer with a width of 40, whose output is then converted into a sequence of features. This combined feature sequence is then fed into a block of five 1D convolutional layers with 16, 32, 32, 64, and 4 filters, respectively. The last of these has four outputs, as the fifth geometric parameter is constant for the unit cells with fixed sizes. Therefore, the final convolutional layer is followed by a concatenation layer, which adds this parameter to the

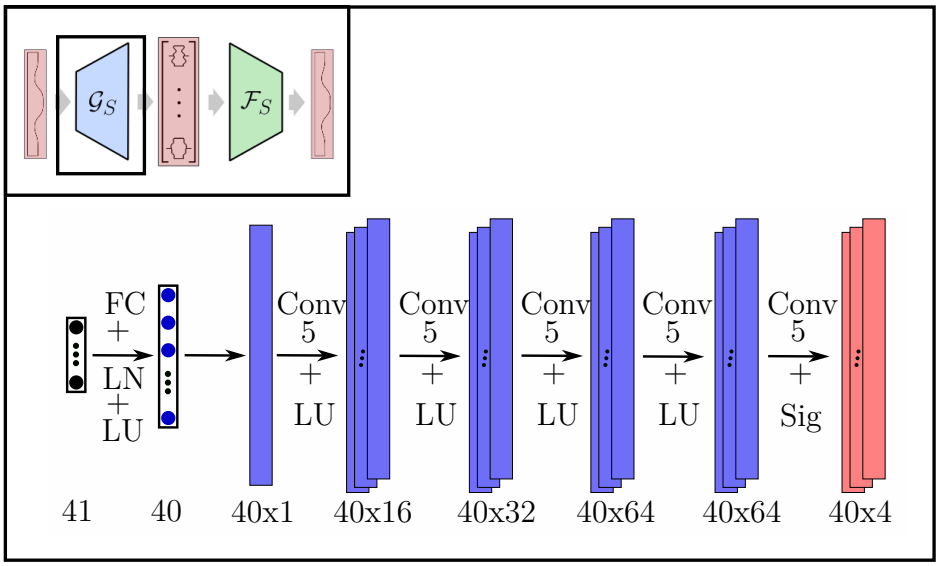

Figure 5.6: The architecture of the inverse model $\mathcal{G}_S$, which predicts the shape for a sequence of mechanical properties. The different layers are color-coded: black indicates the input vector containing the y-coordinates of the deformed shape, red represents the output sequence of unit cell parameters, and blue corresponds to the hidden layers. The network combines fully connected (FC) and convolutional (Conv) layers with layer normalization (LN) and LeakyReLU (LU) and Sigmoid (Sig) activation functions.

others. Each convolutional layer uses a kernel of size 5, with same padding. Regarding activation functions, the first four convolutional layers as well as the fully connected layer use LeakyReLU functions, while the last convolutional layer employs a sigmoid activation function. Furthermore, layer normalization [197] was used in the fully connected layer for regularization.

5.3.2 Training process

As per the TNN architecture, first $\mathcal{F}_S$ was trained to predict the resulting deformation for a given quasi-one-dimensional shape morphing sheet. However, in this case, that meant training its two constituent networks, $\mathcal{F}$ and $\mathcal{F}_E$, independently.

$\mathcal{F}$ was trained on a dataset consisting of pairs of vectors $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$, where $\boldsymbol{x}^{(i)}$ represents the unit cell parameters and $\boldsymbol{y}^{(i)}$ corresponds to the Poisson's ratio and Young's modulus. This dataset is very similar to the one used in Chapter 3 for fixed-dimension unit cells, but with more structures and the addition of labels for the Young's modulus of the metamaterials. Like those for the Poisson's ratio, these labels were obtained through simulations using the finite element (FE) software package COMSOL 5.4a. For these simulations, periodic boundary

conditions were assumed. The dataset comprised $50,000$ data points, with 90% used as training data and the remaining 10% as the test set. Both the parameters of the unit cells and the mechanical properties were normalized to the interval $[0, 1]$. Adam [94] was chosen as the optimizer, with training performed for 1,500 epochs using a learning rate of 1e-4, a batch size of 32, and the MSE as the loss function.

$\mathcal{F}_E$ was trained on $80,000$ pairs consisting of vectors $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$, where $\mathbf{x}^{(i)}$ represents the unit cell sequences and $\mathbf{y}^{(i)}$ corresponds to the resulting shape at 41 points along the curve. Out of these $80,000$ pairs, 90% were used as training data, with the remaining 10% serving as the validation set. All parameters of both the input and output were normalized to the interval $[0, 1]$. Adam [94] was chosen as the optimizer, with training performed for 1,500 epochs using a learning rate of 1e-4, a batch size of 32, and the MSE as the loss function.

The resulting shapes from this dataset were also used to train $\mathcal{G}_S$ with the help of $\mathcal{F}_S$. For this, $\mathcal{G}_S$ was provided with target shapes, which were then compared to the predictions made by $\mathcal{F}_S$ for the corresponding unit cell sequences generated by $\mathcal{G}_S$. As before, Adam [94] was used as the optimizer, with training performed for $3,000$ epochs using a learning rate of 1e-4, a batch size of 32, and the MSE as the loss function.

5.3.3 Results

Since the components $\mathcal{F}$ and $\mathcal{F}_M$ of the forward model $\mathcal{F}_S$ were trained independently, it is important to evaluate both their individual performance as well as their performance in combination. For $\mathcal{F}$, the comparison between the Poisson's ratios (ν_s) predicted through simulation for given unit cells and the predicted values (ν_n) is shown in Figure 5.7a, while Figure 5.7b shows the same comparison for the Young's moduli. It can be observed that for both properties, the predictions made by $\mathcal{F}$ are very close to those computed by simulation ($R^2 = 0.99996$ and $R^2 = 0.99986$), with the fit nearly aligning with the 45° line, which would indicate an ideal match. Furthermore, these predictions generalize well to the test set ($R^2 = 0.99995$ and $R^2 = 0.99985$).

The results for $\mathcal{F}_E$ are shown in Figure 5.8a. It shows the comparison between the components of the resulting shapes (c_i, with $i \in \{1, \ldots, 41\}$) and the corresponding predictions ($\hat{c}_i$) made by $\mathcal{F}_E$ for both the training and testing data. Since there are 41 components, the predictions for all of them were combined into a single graph for convenience. It can be observed that there is good agreement between the labels and predictions ($R^2 = 0.99564$), which also generalizes well to

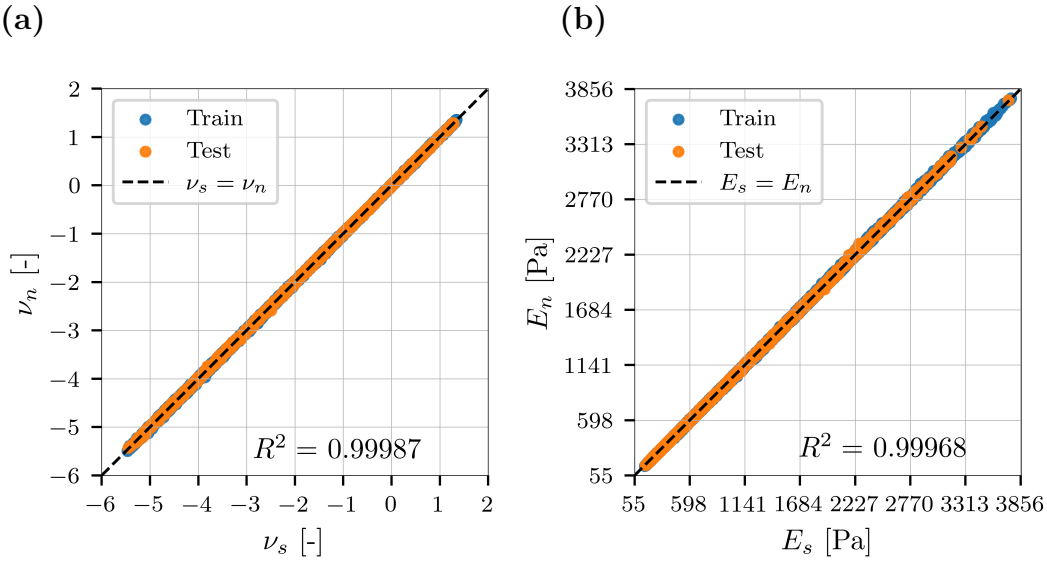

Figure 5.7: Comparison between Poisson's ratios (ν_s) obtained by simulation and the corresponding predictions made by $\mathcal{F}$ (ν_n) **(a)**, between Young's moduli (E_s) obtained by simulation and the corresponding predictions made by $\mathcal{F}$ (E_n) **(b)**.

the test data ($R^2 = 0.99506$). In a similar manner, Figure 5.8b shows the results for $\mathcal{F}_S$ on a dataset where the unit cells of the shape morphing sheet were selected by sampling the geometric parameters from a uniform distribution, instead of sampling the properties. This leads to a very different distribution than the one used to train $\mathcal{F}_E$. Despite this, a good fit was achieved by the combined model $\mathcal{F}_S$ for this data ($R^2 = 0.95301$). The same data was also used to evaluate $\mathcal{G}_S$. Figure 5.8c shows the comparison between the components of the target shape (t_i) and the shapes resulting from the corresponding generated sheets (c_i). It can be observed that there is a good agreement between them ($R^2 = 0.965$).

Furthermore, to validate the entire system, shape morphing was computed via simulation in COMSOL 6.2 for four generated sheets and compared to both the target shapes and the predictions made by $\mathcal{F}_S$ (see Figure 5.9a–d). Of the four target shapes, two—a sine curve and a linear slope—were selected based solely on their geometric form, while the other two were chosen based on their associated distributions of mechanical properties. These correspond to shapes resulting from five discrete sections of material properties and from a randomly assigned property distribution, respectively. All four cases show good agreement between the simulation results and the target shapes, although a closer examination reveals that the fit near the boundaries is less accurate, with values tending to be too small on the left edge and partially too large on the right.

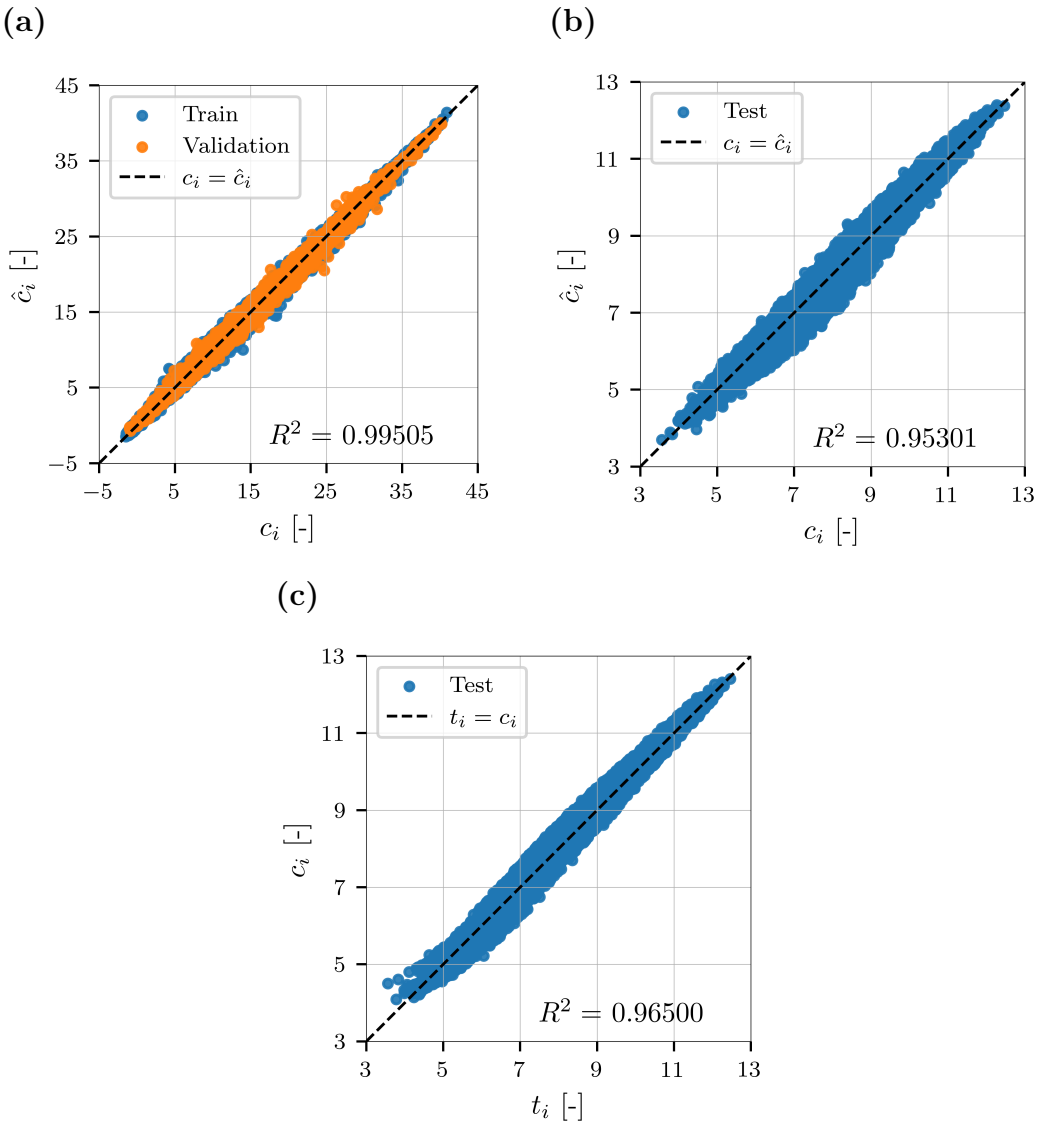

Figure 5.8: Comparison between the components of the shape vector (c_i) and the predictions ($\hat{c}_i$) made by $\mathcal{F}_E$ for the corresponding sequence of properties or training and validation data **(a)**, between the components of the shape vector (c_i) and the corresponding predictions ($\tilde{c}_i$) made by $\mathcal{F}_S$ for the sequence of unit cells **(b)** and between the target components of the shape vector (t_i) and the computed deformation (c_i) for the corresponding sequence of unit cells generated by $\mathcal{G}_S$ **(c)**.

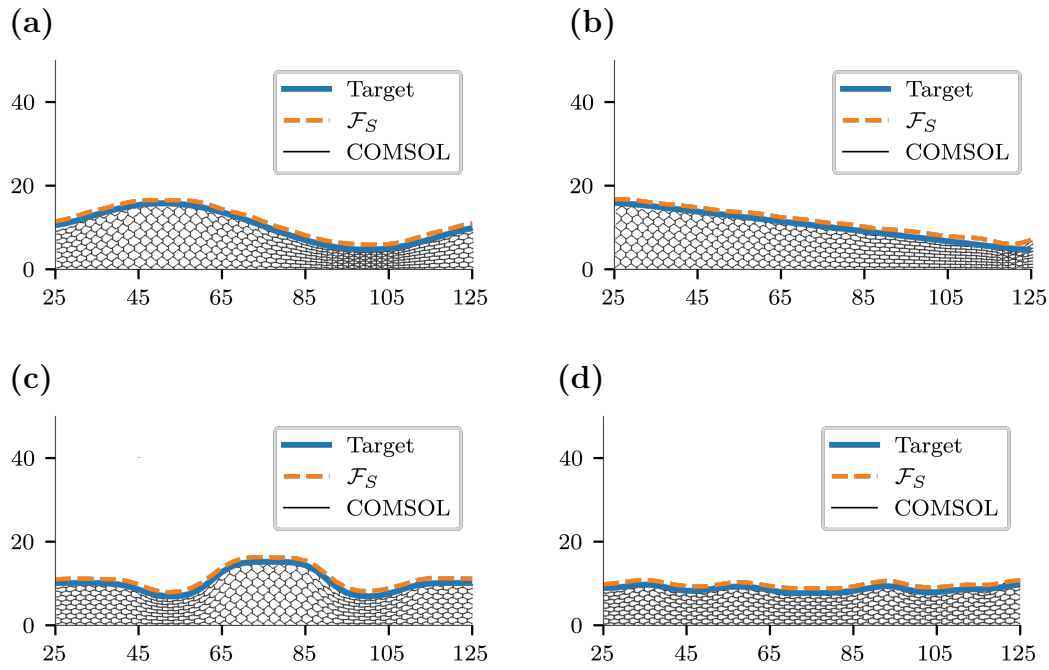

Figure 5.9: Results of the shape morphing for four shape-morphing sheets with the following target shapes: a sine curve **(a)**, a linear slope **(b)**, a shape resulting from five discrete sections of material properties **(c)**, and a shape resulting from randomly assigned material parameters **(d)**. The black unit cells represent the results computed via simulations in COMSOL 6.2, while the blue line indicates the target shape and the orange line represents the prediction made by $\mathcal{F}_S$. The prediction was intentionally shifted by one to avoid overlapping with the other lines.

5.4 Conclusion

By combining different unit cells of curved-beam metamaterials, it is possible to create quasi-one-dimensional shape-morphing sheets. These sheets can be designed to match target deformations using a TNN approach, achieving high accuracy ($R^2 = 0.965$). However, training such a network is not without challenges. A uniform distribution of unit cells in the design space does not necessarily result in a uniform distribution in the property space. While this is generally not problematic, it poses difficulties for data-driven approaches when combining multiple unit cells into a heterogeneous metamaterial. In such cases, combinations involving rare unit cells with extreme properties become increasingly unlikely, leading to training data dominated by examples with average overall properties. Here, the issue was addressed by introducing a second neural network to map between the design and property spaces for individual unit cells. This enabled sampling directly in the property space, allowing for the inclusion of more extreme deformations in the dataset.

Chapter 6

Conclusion

The design of mechanical metamaterials poses a fundamentally challenging inverse problem: how to derive complex internal architectures that produce desired mechanical properties. Traditional computational approaches, such as topology optimization, have achieved great success in solving this problem at the scale of individual unit cells. However, when the design involves multiple unit cells, these methods often struggle with scalability and computational cost—making data-driven, model-based approaches more attractive.

This thesis investigated how generative machine learning models—particularly those that have shown success in image generation—can be applied to solve this inverse problem for mechanical metamaterials, with a focus on how the choice of representation and the presence of design restrictions influence the performance of these generative models.

6.1 Summary

To begin, this thesis first reviewed different types of metamaterials, as well as generative and inverse machine learning models, and their existing applications in the design of mechanical metamaterials. It also discussed how generative models can be used to solve inverse problems. The investigated approaches can be divided into two categories: first, the Tandem Neural Network (TNN), a model designed for inverse problems but limited in its ability to provide multiple solutions for the same target properties; and second, generative machine learning models, which are primarily designed to generate data that satisfy design constraints and require careful conditioning on the target properties to solve inverse problems.

To explore how these models can be used for the design of metamaterials, Chapter 3 focused on their application for the design of a curved-beam metamaterial

based on a single unit cell. By replacing some of the straight beams in the unit cells with curved ones, the Poisson's ratio of the metamaterial could be modified. To parameterize the curves, B-splines were used, enabling the design space to be represented as a compact interval. It was then shown how to apply a TNN, VAE, and GAN to generate unit cells with target properties for a metamaterial defined by such a design space. All three approaches showed good performance; however, each involves a trade-off between the diversity of the generated samples and the accuracy with which the target properties are matched.

The representation chosen for the curved-beam metamaterial had the advantage that no dependencies between the design parameters existed, and therefore no complex design restrictions had to be learned by the generative models. To investigate the effect such constraints have on generative models, Chapter 4 explored the design of a kirigami metamaterial, where design constraints—specifically, the prohibition of intersecting cuts—introduced complex limitations on the design space and caused dependencies among the different design features. This not only made training the forward model difficult, but also impacted the generative models, which are expected to learn design restrictions. As these models are designed for images, they make assumptions about the design space that are generally appropriate for image data but not for metamaterials. For the VAE and WGAN, this assumption is that Euclidean distance can be used to measure similarity between samples. While this leads to greater stability during image generation [121], it prevented the generative models from effectively learning the constraints of the kirigami material. The GAN and diffusion models, which do not rely on this assumption, performed better—yet even they did not fully capture the design restrictions.

Finally, after evaluating how these models can be applied to metamaterials based on a single unit cell, Chapter 5 turned to their application in the design of heterogeneous metamaterials and the challenges that come with combining multiple unit cells. For this purpose, a shape-morphing heterogeneous metamaterial composed of spatially varying unit cells was investigated. Once again, cells from the curved-beam metamaterial were used, allowing a TNN to be employed as the inverse model. Combining multiple unit cells introduced a new challenge: uniformly sampling unit cells in the design space resulted in a non-uniform distribution in the property space. While this is generally acceptable when designing a single unit cell, combinations involving multiple rare unit cells become even less likely. This is often the case for unit cells with extreme properties that lie on the edge of the property space, effectively excluding large deformations of the shape-morphing metamaterial from the training data. This challenge was addressed by splitting

the forward model of the TNN into two parts: first, a model that predicts shape deformation based on the properties of the unit cells used in the metamaterial; and second, a model that predicts these properties from the geometric parameters. This separation made it possible to train the model using a uniform property distribution, including extreme cases in the training data, thereby enabling the training of a generative model capable of controlling shape deformation.

6.2 Limitations and Outlook

The central objective of this thesis was to investigate the use of generative machine learning models for the design of various classes of mechanical metamaterials. A key finding was that the parameterization of the metamaterial has a significant influence on how well these models can be applied to its design. In Chapter 4, it was observed that generative models struggle when design constraints introduce complex dependencies between parameters. Since most generative models have been originally developed for image data, they often rely on assumptions—such as the use of Euclidean distance to quantify similarity—that do not necessarily hold for the representations of metamaterials. As a result, these models may fail to learn valid design spaces and can generate geometries that violate design constraints, even when not conditioned on specific target properties.

This limitation can be addressed in two ways: through careful selection of parameterizations or through more advanced machine learning models. Regarding parameterization, B-splines—used in Chapter 3 and Chapter 5—proved to be especially effective. Not only do they work well with generative models, but they also open the door to further integration with simulation techniques, such as Isogeometric Analysis (IGA), which could improve the accuracy of property predictions [195]. On the model side, advances such as the Diffusion Probabilistic Field [225], which are specifically designed for complex and structured design spaces in 3D model generation, may also offer the capability to learn the design constraints on metamaterial design spaces. Furthermore, the choice of parameterization also affects how the design space is sampled—ultimately shaping the training dataset. This, in turn, influences critical factors such as the distribution of mechanical properties across the dataset, which can have a strong impact on the training performance and generalization ability of the models. As demonstrated in Chapter 5, it is essential to consider which distributions—in geometry space or property space—should be achieved, and to sample the training data accordingly. This is especially important for the design of heterogeneous metamaterials, where the combination of multiple unit cells amplifies these effects.

Overall, while generative machine learning models have proven useful for the design of mechanical metamaterials—both for those based on a single unit cell and for heterogeneous ones—their application requires the selection of intelligent parameterizations and careful consideration of the training data.

Chapter 7

Acknowledgments

I first and foremost want to thank my supervisor, Viacheslav Slesarenko. The PhD experience depends greatly on the supervisor, and I was fortunate to have an exceptional one. Thank you for the support, the guidance, and most of all the direct, honest feedback — it makes training much easier when one has proper labels.

I would also like to thank Lars Pastewka for his calm and constructive feedback, and the members of the committee for taking on these duties.

I am grateful to my coauthors for sharing their expertise in mechanics and for their help with the experiments. Thanks to the members of the LivMatS cluster for their ongoing support, and to the members of the Micro- and Material Mechanics Group, the Soft Machines Group, and the Plant Biomechanics Group for the friendly atmosphere and interesting discussions that encouraged me to look beyond my own field.

Further thanks go to Angelika Gedsun, Naeim Ghavidelnia, and Nadira Grübel for helping me navigate academia.

I was very lucky to share my office with amazing people. I'm grateful to Esther Johnson, David Schwarz, and Michael Somr for all the fun, support, and discussions — scientific and otherwise.

Thanks to Lorenz Diener, Tim Laue, Kevin Moerman, William Ronan, Thomas Röfer, and Tanja Schultz for setting me on this path.

And last but not least, thank you to my family and friends for their support, their welcome distractions, and for patiently listening to complaints about things they sometimes didn't even understand.

Bibliography

[1] X. Zheng, X. Zhang, T.-T. Chen, and I. Watanabe, "Deep learning in mechanical metamaterials: From prediction and generation to inverse design," *Advanced Materials*, vol. 35, no. 45, p. 2 302 530, 2023. DOI: 10.1002/adma.202302530.

[2] X. Zheng *et al.*, "Ultralight, ultrastiff mechanical metamaterials," *Science*, vol. 344, no. 6190, pp. 1373–1377, 2014. DOI: 10.1126/science.1252291.

[3] H. M. A. Kolken and A. A. Zadpoor, "Auxetic mechanical metamaterials," *RSC Advances*, vol. 7, no. 9, pp. 5111–5129, 2017. DOI: 10.1039/C6RA27333E.

[4] K. Bertoldi, V. Vitelli, J. Christensen, and M. van Hecke, "Flexible mechanical metamaterials," *Nature Reviews Materials*, vol. 2, no. 11, p. 17 066, Oct. 2017. DOI: 10.1038/natrevmats.2017.66.

[5] F. Libonati and M. J. Buehler, "Advanced structural materials by bioinspiration," *Advanced Engineering Materials*, vol. 19, no. 5, p. 1 600 787, 2017. DOI: 10.1002/adem.201600787.

[6] G. W. Milton and A. V. Cherkaev, "Which elasticity tensors are realizable?" *Journal of Engineering Materials and Technology*, vol. 117, no. 4, pp. 483–493, Oct. 1995. DOI: 10.1115/1.2804743.

[7] L. J. Gibson, M. F. Ashby, G. S. Schajer, and C. I. Robertson, "The mechanics of two-dimensional cellular materials," *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, vol. 382, no. 1782, pp. 25–42, 1982. DOI: 10.1098/rspa.1982.0087.

[8] R. Lakes, "Foam structures with a negative poisson's ratio," *Science*, vol. 235, no. 4792, pp. 1038–1040, 1987. DOI: 10.1126/science.235.4792.1038.

[9] J. N. Grima, R. Gatt, A. Alderson, and K. E. Evans, "On the potential of connected stars as auxetic systems," *Molecular Simulation*, vol. 31, no. 13, pp. 925–935, 2005. DOI: 10.1080/08927020500401139.

[10] U. Larsen, O. Signund, and S. Bouwsta, "Design and fabrication of compliant micromechanisms and structures with negative poisson's ratio," *Journal of Microelectromechanical Systems*, vol. 6, no. 2, pp. 99–106, 1997. DOI: 10.1109/84.585787.

[11] R. Lakes, "Deformation mechanisms in negative poisson's ratio materials: Structural aspects," *Journal of Materials Science*, vol. 26, no. 9, pp. 2287–2292, May 1991. DOI: 10.1007/BF01130170.

[12] J. N. Grima and K. E. Evans, "Auxetic behavior from rotating squares," *Journal of Materials Science Letters*, vol. 19, no. 17, pp. 1563–1565, Sep. 2000. DOI: 10.1023/A:1006781224002.

[13] K. Miura, "Method of packaging and deployment of large membranes in space," *The Institute of Space and Astronautical Science report*, no. 618, pp. 1–9, 1985.

[14] J. N. Grima, L. Mizzi, K. M. Azzopardi, and R. Gatt, "Auxetic perforated mechanical metamaterials with randomly oriented cuts," *Advanced Materials*, vol. 28, no. 2, pp. 385–389, 2016. DOI: 10.1002/adma.201503653.

[15] E. Shamonina and L. Solymar, "Metamaterials: How the subject started," *Metamaterials*, vol. 1, no. 1, pp. 12–18, 2007. DOI: 10.1016/j.metmat.2007.02.001.

[16] V. G. Veselago, "The electrodynamics of substances with simultaneously negative values of ϵ and μ," *Soviet Physics Uspekhi*, vol. 10, no. 4, p. 509, Apr. 1968. DOI: 10.1070/PU1968v010n04ABEH003699.

[17] D. R. Smith, W. J. Padilla, D. C. Vier, S. C. Nemat-Nasser, and S. Schultz, "Composite medium with simultaneously negative permeability and permittivity," *Phys. Rev. Lett.*, vol. 84, pp. 4184–4187, 18 May 2000. DOI: 10.1103/PhysRevLett.84.4184.

[18] J. Pendry, A. Holden, D. Robbins, and W. Stewart, "Magnetism from conductors and enhanced nonlinear phenomena," *IEEE Transactions on Microwave Theory and Techniques*, vol. 47, no. 11, pp. 2075–2084, 1999. DOI: 10.1109/22.798002.

[19] R. A. Shelby, D. R. Smith, and S. Schultz, "Experimental verification of a negative index of refraction," *Science*, vol. 292, no. 5514, pp. 77–79, 2001. DOI: 10.1126/science.1058847.

[20] J. B. Pendry, "Negative refraction makes a perfect lens," *Phys. Rev. Lett.*, vol. 85, pp. 3966–3969, 18 Oct. 2000. DOI: 10.1103/PhysRevLett.85.3966.

[21] Y. Liu and X. Zhang, "Metamaterials: A new frontier of science and technology," *Chemical Society reviews*, vol. 40, pp. 2494–2507, 5 2011. DOI: 10.1039/C0CS00184H.

[22] G. Ma and P. Sheng, "Acoustic metamaterials: From local resonances to broad horizons," *Science Advances*, vol. 2, no. 2, e1501595, 2016. DOI: 10.1126/sciadv.1501595.

[23] M. Kadic, T. Bückmann, R. Schittny, and M. Wegener, "Metamaterials beyond electromagnetism," *Reports on Progress in Physics*, vol. 76, no. 12, p. 126 501, Nov. 2013. DOI: 10.1088/0034-4885/76/12/126501.

[24] A. A. Zadpoor, "Mechanical meta-materials," *Materials horizons*, vol. 3, pp. 371–381, 5 2016. DOI: 10.1039/C6MH00065G.

[25] R. Almgren, "An isotropic three-dimensional structure with poisson's ratio =-1," *Journal of Elasticity*, vol. 15, pp. 427–430, 1985. DOI: 10.1007/BF00042531.

[26] A. Kolpakov, "Determination of the average characteristics of elastic frameworks," *Journal of Applied Mathematics and Mechanics*, vol. 49, no. 6, pp. 739–745, 1985. DOI: https://doi.org/10.1016/0021-8928(85)90011-5.

[27] G. W. Milton, "Composite materials with poisson's ratios close to - 1," *Journal of the Mechanics and Physics of Solids*, vol. 40, no. 5, pp. 1105–1137, 1992, ISSN: 0022-5096. DOI: 10.1016/0022-5096(92)90063-8.

[28] M. Askari *et al.*, "Additive manufacturing of metamaterials: A review," *Additive Manufacturing*, vol. 36, p. 101 562, 2020. DOI: 10.1016/j.addma.2020.101562.

[29] M. Kadic, T. Bückmann, N. Stenger, M. Thiel, and M. Wegener, "On the practicability of pentamode mechanical metamaterials," *Applied Physics Letters*, vol. 100, no. 19, p. 191 901, May 2012. DOI: 10.1063/1.4709436.

[30] N. A. Fleck, V. S. Deshpande, and M. F. Ashby, "Micro-architectured materials: Past, present and future," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 466, no. 2121, pp. 2495–2516, 2010. DOI: 10.1098/rspa.2010.0215.

[31] L. Yang, O. Harrysson, H. West, and D. Cormier, "Compressive properties of ti-6al-4v auxetic mesh structures made by electron beam melting," *Acta Materialia*, vol. 60, no. 8, pp. 3370–3379, 2012. DOI: 10.1016/j.actamat.2012.03.015.

[32] L. R. Meza, S. Das, and J. R. Greer, "Strong, lightweight, and recoverable three-dimensional ceramic nanolattices," *Science*, vol. 345, no. 6202, pp. 1322–1326, 2014. DOI: 10.1126/science.1255908.

[33] S. Daynes, S. Feih, W. F. Lu, and J. Wei, "Optimisation of functionally graded lattice structures using isostatic lines," *Materials & Design*, vol. 127, pp. 215–223, 2017. DOI: 10.1016/j.matdes.2017.04.082.

[34] Y. Han and W. Lu, "Evolutionary design of nonuniform cellular structures with optimized poisson's ratio distribution," *Materials & Design*, vol. 141, pp. 384–394, 2018. DOI: 10.1016/j.matdes.2017.12.047.

[35] M. J. Mirzaali, S. Janbaz, M. Strano, L. Vergani, and A. Zadpoor, "Shape-matching soft mechanical metamaterials," *Scientific Reports*, vol. 8, Jan. 2018. DOI: 10.1038/s41598-018-19381-3.

[36] T. Lichti *et al.*, "Optimal design of shape changing mechanical metamaterials at finite strains," *International Journal of Solids and Structures*, vol. 252, p. 111 769, 2022. DOI: 10.1016/j.ijsolstr.2022.111769.

[37] Z. Zhang and A. O. Krushynska, "Programmable shape-morphing of rose-shaped mechanical metamaterials," *APL Materials*, vol. 10, no. 8, p. 080 701, Aug. 2022. DOI: 10.1063/5.0099323.

[38] X. Ren, R. Das, P. Tran, T. D. Ngo, and Y. M. Xie, "Auxetic metamaterials and structures: A review," en, *Smart Materials and Structures*, vol. 27, no. 2, p. 023 001, 2018. DOI: 10.1088/1361-665X/aaa61c.

[39] E. Barchiesi, M. Spagnuolo, and L. Placidi, "Mechanical metamaterials: A state of the art," *Mathematics and Mechanics of Solids*, vol. 24, p. 108 128 651 773 569, Feb. 2018. DOI: 10.1177/1081286517735695.

[40] X. Hou and V. V. Silberschmidt, "Metamaterials with negative poisson's ratio: A review of mechanical properties and deformation mechanisms," in *Mechanics of Advanced Materials: Analysis of Properties and Performance*, V. V. Silberschmidt and V. P. Matveenko, Eds. Cham: Springer International Publishing, 2015, pp. 155–179. DOI: 10.1007/978-3-319-17118-0_7.

[41] M. Balan P, J. Mertens A, and M. V. A. R. Bahubalendruni, "Auxetic mechanical metamaterials and their futuristic developments: A state-of-art review," *Materials Today Communications*, vol. 34, p. 105 285, 2023. DOI: 10.1016/j.mtcomm.2022.105285.

[42] I. Masters and K. Evans, "Models for the elastic deformation of honeycombs," *Composite Structures*, vol. 35, no. 4, pp. 403–422, 1996. DOI: 10.1016/S0263-8223(96)00054-2.

[43] J. P. M. Whitty, F. Nazaré, and A. Alderson, "Modelling the effects of density variations on the in-plane poisson's ratios and young's moduli of periodic conventional and re-entrant honeycombs - part 1: Rib thickness variations," *Cellular Polymers*, vol. 21, pp. 69–98, 2002.

[44] D. Yang, S. Lee, and F. Huang, "Geometric effects on micropolar elastic honeycomb structure with negative poisson's ratio using the finite element method," *Finite Elements in Analysis and Design*, vol. 39, no. 3, pp. 187–205, 2003. DOI: 10.1016/S0168-874X(02)00066-5.

[45] J. K. Wilt, C. Yang, and G. X. Gu, "Accelerating auxetic metamaterial design with deep learning," *Advanced Engineering Materials*, vol. 22, no. 5, p. 1 901 266, 2020. DOI: 10.1002/adem.201901266.

[46] O. Skarsetz, V. Slesarenko, and A. Walther, "Programmable auxeticity in hydrogel metamaterials via shape-morphing unit cells," *Advanced Science*, vol. 9, no. 23, p. 2 201 867, 2022. DOI: 10.1002/advs.202201867.

[47] T.-C. Lim, "A 3d auxetic material based on intersecting double arrowheads," *physica status solidi (b)*, vol. 253, no. 7, pp. 1252–1260, 2016. DOI: 10.1002/pssb.201600015.

[48] M. Rad, Z. Ahmad, and A. Alias, "Computational approach in formulating mechanical characteristics of 3d star honeycomb auxetic structure," *Advances in Materials Science and Engineering*, vol. 2015, pp. 1–11, Jan. 2015. DOI: 10.1155/2015/650769.

[49] S. Babaee, J. Shim, J. C. Weaver, E. R. Chen, N. Patel, and K. Bertoldi, "3d soft metamaterials with negative poisson's ratio," *Advanced Materials*, vol. 25, no. 36, pp. 5044–5049, 2013. DOI: 10.1002/adma.201301986.

[50] K. Wojciechowski, "Two-dimensional isotropic system with a negative poisson ratio," *Physics Letters A*, vol. 137, no. 1, pp. 60–64, 1989. DOI: 10.1016/0375-9601(89)90971-7.

[51] J. N. Grima, R. Gatt, and P.-S. Farrugia, "On the properties of auxetic meta-tetrachiral structures," *physica status solidi (b)*, vol. 245, no. 3, pp. 511–520, 2008. DOI: 10.1002/pssb.200777704.

[52] A. Alderson *et al.*, "Elastic constants of 3-, 4- and 6-connected chiral and anti-chiral honeycombs subject to uniaxial in-plane loading," *Composites Science and Technology*, vol. 70, no. 7, pp. 1042–1048, 2010, Special issue on Chiral Smart Honeycombs. DOI: 10.1016/j.compscitech.2009.07.009.

[53] D. Mousanezhad, B. Haghpanah, R. Ghosh, A. M. Hamouda, H. Nayeb-Hashemi, and A. Vaziri, "Elastic properties of chiral, anti-chiral, and hierarchical honeycombs: A simple energy-based approach," *Theoretical and Applied Mechanics Letters*, vol. 6, no. 2, pp. 81–96, 2016. DOI: 10.1016/j.taml.2016.02.004.

[54] J. N. Grima *et al.*, "On the auxetic properties of generic rotating rigid triangles," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 468, no. 2139, pp. 810–830, 2012. DOI: 10.1098/rspa.2011.0273.

[55] J. N. Grima, R. Gatt, A. Alderson, and K. E. Evans, "On the auxetic properties of 'rotating rectangles' with different connectivity," *Journal of the Physical Society of Japan*, vol. 74, no. 10, pp. 2866–2867, 2005. DOI: 10.1143/JPSJ.74.2866.

[56] J. N. Grima, P.-S. Farrugia, R. Gatt, and D. Attard, "On the auxetic properties of rotating rhombi and parallelograms: A preliminary investigation," *physica status solidi (b)*, vol. 245, no. 3, pp. 521–529, 2008. DOI: 10.1002/pssb.200777705.

[57] E. Chetcuti *et al.*, "Modeling auxetic foams through semi-rigid rotating triangles," *physica status solidi (b)*, vol. 251, no. 2, pp. 297–306, 2014. DOI: 10.1002/pssb.201384252.

[58] C. Jiang, F. Rist, H. Wang, J. Wallner, and H. Pottmann, "Shape-morphing mechanical metamaterials," *Computer-Aided Design*, vol. 143, p. 103 146, 2022, ISSN: 0010-4485. DOI: 10.1016/j.cad.2021.103146.

[59] L. H. Dudte, E. Vouga, T. Tachi, and L. Mahadevan, "Programming curvature using origami tessellations," *Nature Materials*, vol. 15, no. 5, pp. 583–588, May 2016. DOI: 10.1038/nmat4540.

[60] C. D. Santangelo, "Extreme mechanics: Self-folding origami," *Annual Review of Condensed Matter Physics*, vol. 8, no. Volume 8, 2017, pp. 165–183, 2017. DOI: 10.1146/annurev-conmatphys-031016-025316.

[61] T. Mukhopadhyay *et al.*, "Programmable stiffness and shape modulation in origami materials: Emergence of a distant actuation feature," *Applied Materials Today*, vol. 19, p. 100 537, 2020. DOI: 10.1016/j.apmt.2019.100537.

[62] M. Schenk and S. D. Guest, "Geometry of miura-folded metamaterials," *Proceedings of the National Academy of Sciences*, vol. 110, no. 9, pp. 3276–3281, 2013. DOI: 10.1073/pnas.1217998110.

[63] S. Kamrava, D. Mousanezhad, H. Ebrahimi, R. Ghosh, and A. Vaziri, "Origami-based cellular metamaterial with auxetic, bistable, and self-locking properties," *Scientific Reports*, vol. 7, no. 1, p. 46 046, Apr. 2017. DOI: 10.1038/srep46046.

[64] E. T. Filipov, G. H. Paulino, and T. Tachi, "Origami tubes with reconfigurable polygonal cross-sections," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 472, no. 2185, p. 20 150 607, 2016. DOI: 10.1098/rspa.2015.0607.

[65] C. Du, Y. Wang, and Z. Kang, "Auxetic kirigami metamaterials upon large stretching," *ACS Applied Materials & Interfaces*, vol. 15, no. 15, pp. 19 190–19 198, 2023. DOI: 10.1021/acsami.3c00946.

[66] G. A. Holzapfel, *Nonlinear solid mechanics : a continuum approach for engineering*, eng, Corrected reprint. Chichester: J. Wiley, 2010.

[67] D. Dinkler and U. Kowalsky, *Introduction to Finite Element Methods*, 1st ed. Springer Vieweg Wiesbaden, 2023. DOI: 10.1007/978-3-658-42742-9.

[68] K.-J. Bathe, *Finite element analysis of solids and fluids i*, https://ocw.mit.edu/courses/2-092-finite-element-analysis-of-solids-and-fluids-i-fall-2009/, MIT OpenCourseWare, Accessed: 2025-06-28, 2009.

[69] P. A. Kelly, *Mechanics lecture notes: Foundations of continuum mechanics*, http://homepages.engineering.auckland.ac.nz/~pkel015/SolidMechanicsBooks/index.html, Accessed: 2025-06-28.

[70] M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for Machine Learning*. Cambridge University Press, 2020. DOI: 10.1017/9781108679930.

[71] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[72] R. Kruse *et al.*, *Computational intelligence*. Springer, 2011. DOI: 10.1007/978-3-658-10904-2.

[73] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009, vol. 2. DOI: 10.1007/978-0-387-84858-7.

[74] K. P. Murphy, *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. [Online]. Available: probml.ai.

[75] P. Petersen and J. Zech, "Mathematical theory of deep learning," *arXiv e-prints*, arXiv:2407.18384, Jul. 2024. DOI: 10.48550/arXiv.2407.18384.

[76] V. Santhanam, V. I. Morariu, and L. S. Davis, " Generalized Deep Image to Image Regression," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Los Alamitos, CA, USA: IEEE Computer Society, Jul. 2017, pp. 5395–5405. DOI: 10.1109/CVPR.2017.573.

[77] A. Vaswani *et al.*, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. DOI: 10.5555/3295222.3295349.

[78] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[79] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901. DOI: 10.1080/14786440109462720.

[80] H. Hotelling, "Analysis of a complex of statistical variables into principal components.," *Journal of Educational Psychology*, vol. 24, pp. 498–520, 1933. DOI: 10.1037/h0071325.

[81] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[82] M. Minsky and S. Papert, *Perceptrons; an Introduction to Computational Geometry*. MIT Press, 1969, ISBN: 9780262630221.

[83] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" In *2009 IEEE 12th International Conference on Computer Vision*, 2009, pp. 2146–2153. DOI: 10.1109/ICCV.2009.5459469.

[84] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.*, "Rectifier nonlinearities improve neural network acoustic models," in *Proceedings of the International Conference on Machine Learning*, Atlanta, GA, vol. 30, 2013, p. 3.

[85] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, Dec. 1989. DOI: 10.1007/BF02551274.

[86] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. DOI: 10.1016/0893-6080(89)90020-8.

[87] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The expressive power of neural networks: A view from the width," ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6232–6240. DOI: 10.5555/3295222.3295371.

[88] K. H. J., "Gradient theory of optimal flight paths," *ARS Journal*, vol. 30, no. 10, pp. 947–954, 1960. DOI: 10.2514/8.5282.

[89] A. Bryson and Y. Ho, *Applied Optimal Control: Optimization, Estimation, and Control* (Blaisdell book in the pure and applied sciences). Blaisdell Publishing Company, 1969.

[90] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition, ed. de rumelhart and j. mcclelland. vol. 1. 1986," *Biometrika*, vol. 71, no. 599-607, p. 6, 1986.

[91] H. Robbins and S. Monro, "A Stochastic Approximation Method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951. DOI: 10.1214/aoms/1177729586.

[92] J. Kiefer and J. Wolfowitz, "Stochastic estimation of the maximum of a regression function," *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 1952. DOI: 10.1214/aoms/1177729392.

[93] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv e-prints*, arXiv:1609.04747, Sep. 2016. DOI: 10.48550/arXiv.1609.04747.

[94] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[95] C. Goodhart, *Problems of monetary management : The u.k. experience*, [Sydney], 1975.

[96] Y. Le Cun *et al.*, "Handwritten digit recognition with a back-propagation network," in *Proceedings of the 3rd International Conference on Neural Information Processing Systems*, ser. NIPS'89, Cambridge, MA, USA: MIT Press, 1989, pp. 396–404. DOI: 10.5555/2969830.2969879.

[97] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012.

[98] Y. LeCun, "Generalization and network design strategies," *Connections in Perspective*, 1989.

[99] L. Ardizzone *et al.*, "Analyzing Inverse Problems with Invertible Neural Networks," *arXiv e-prints*, arXiv:1808.04730, Aug. 2018. DOI: 10.48550/arXiv.1808.04730.

[100] J. Hadamard, "Sur les problèmes aux dérivés partielles et leur signification physique," *Princeton University Bulletin*, vol. 13, pp. 49–52, 1902.

[101] H. Kabir, Y. Wang, M. Yu, and Q.-J. Zhang, "Neural network inverse modeling and applications to microwave filter design," *IEEE Transactions on Microwave Theory and Techniques*, vol. 56, no. 4, pp. 867–879, 2008. DOI: 10.1109/TMTT.2008.919078.

[102] D. Liu, Y. Tan, E. Khoram, and Z. Yu, "Training deep neural networks for the inverse design of nanophotonic structures," *ACS Photonics*, vol. 5, no. 4, pp. 1365–1369, 2018. DOI: 10.1021/acsphotonics.7b01377.

[103] J.-H. Bastek, S. Kumar, B. Telgen, R. N. Glaesener, and D. M. Kochmann, "Inverting the structure-property map of truss metamaterials by deep learning," *Proceedings of the National Academy of Sciences*, vol. 119, no. 1, e2111505119, 2022. DOI: 10.1073/pnas.2111505119.

[104] G. Felsch, N. Ghavidelnia, D. Schwarz, and V. Slesarenko, "Controlling auxeticity in curved-beam metamaterials via a deep generative model," *Computer Methods in Applied Mechanics and Engineering*, vol. 410, p. 116 032, 2023. DOI: 10.1016/j.cma.2023.116032.

[105] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2014.

[106] D. H. Ballard, "Modular learning in neural networks," in *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1*, ser. AAAI'87, Washington, DC, USA: AAAI Press, 1987, pp. 279–284. DOI: 10.5555/1863696.1863746.

[107] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006. DOI: 10.1126/science.1127647.

[108] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, Gold Coast, Australia QLD, Australia: Association for Computing Machinery, 2014, pp. 4–11. DOI: 10.1145/2689746.2689747.

[109] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08, Helsinki, Finland: Association for Computing Machinery, 2008, pp. 1096–1103. DOI: 10.1145/1390156.1390294.

[110] I. Higgins *et al.*, "Beta-vae: Learning basic visual concepts with a constrained variational framework," in *International Conference on Learning Representations*, 2016.

[111] K. Sohn, X. Yan, and H. Lee, "Learning structured output representation using deep conditional generative models," in *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15, Montreal, Canada: MIT Press, 2015, pp. 3483–3491. DOI: 10.5555/2969442.2969628.

[112] D. P. Kingma, D. J. Rezende, S. Mohamed, and M. Welling, "Semi-supervised learning with deep generative models," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14, Montreal, Canada: MIT Press, 2014, pp. 3581–3589. DOI: 10.5555/2969033.2969226.

[113] H. Kameoka, T. Kaneko, K. Tanaka, and N. Hojo, "Acvae-vc: Non-parallel voice conversion with auxiliary classifier variational autoencoder," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 9, pp. 1432–1443, 2019. DOI: 10.1109/TASLP.2019.2917232.

[114] I. Goodfellow *et al.*, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[115] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16, Barcelona, Spain: Curran Associates Inc., 2016, pp. 2234–2242. DOI: 10.5555/3157096.3157346.

[116] S. Nowozin, B. Cseke, and R. Tomioka, "F-gan: Training generative neural samplers using variational divergence minimization," *Advances in neural information processing systems*, vol. 29, 2016.

[117] K. Roth, A. Lucchi, S. Nowozin, and T. Hofmann, "Stabilizing training of generative adversarial networks through regularization," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 2015–2025. DOI: 10.5555/3294771.3294963.

[118] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6629–6640. DOI: 10.5555/3295222.3295408.

[119] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16, Barcelona, Spain: Curran Associates Inc., 2016, pp. 2180–2188. DOI: 10.5555/3157096.3157340.

[120] A. Srivastava, L. Valkov, C. Russell, M. U. Gutmann, and C. Sutton, "Veegan: Reducing mode collapse in gans using implicit variational learning," ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 3310–3320. DOI: 10.5555/3294996.3295090.

[121] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17, Sydney, NSW, Australia: JMLR.org, 2017, pp. 214–223. DOI: 10.5555/3305381.3305404.

[122] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," *Advances in neural information processing systems*, vol. 30, 2017.

[123] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," *arXiv e-prints*, arXiv:1411.1784, Nov. 2014. DOI: 10.48550/arXiv.1411.1784.

[124] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier gans," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17, Sydney, NSW, Australia: JMLR.org, 2017, pp. 2642–2651. DOI: 10.5555/3305890.3305954.

[125] A. Odena, "Semi-Supervised Learning with Generative Adversarial Networks," *arXiv e-prints*, arXiv:1606.01583, Jun. 2016. DOI: 10.48550/arXiv.1606.01583.

[126] S. Bazrafkan, V. Varkarakis, J. Lemley, H. Javidnia, and P. Corcoran, "Versatile auxiliary classification and regression with generative adversarial networks," *IEEE Access*, vol. 9, pp. 38 810–38 825, 2021. DOI: 10.1109/ACCESS.2021.3063793.

[127] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, Lille, France: JMLR.org, 2015, pp. 2256–2265. DOI: 10.5555/3045118.3045358.

[128] W. Feller, "On the Theory of Stochastic Processes, with Particular Reference to Applications," in *First Berkeley Symposium on Mathematical Statistics and Probability*, J. Neyman, Ed., Jan. 1949, pp. 403–432.

[129] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20, Vancouver, BC, Canada: Curran Associates Inc., 2020. DOI: 10.5555/3495724.3496298.

[130] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-Based Generative Modeling through Stochastic Differential Equations," *arXiv e-prints*, arXiv:2011.13456, Nov. 2020. DOI: 10.48550/arXiv.2011.13456.

[131] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," *Advances in neural information processing systems*, vol. 34, pp. 8780–8794, 2021.

[132] J. Ho and T. Salimans, "Classifier-free diffusion guidance," in *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021. [Online]. Available: https://openreview.net/forum?id=qw8AKxfYbI.

[133] C. Saharia *et al.*, "Photorealistic text-to-image diffusion models with deep language understanding," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS '22, New Orleans, LA, USA: Curran Associates Inc., 2022. DOI: 10.5555/3600270.3602913.

[134] G. Giannone, L. Regenwetter, A. Srivastava, D. Gutfreund, and F. Ahmed, "Learning from Invalid Data: On Constraint Satisfaction in Generative Models," *arXiv e-prints*, arXiv:2306.15166, Jun. 2023. DOI: 10.48550/arXiv.2306.15166.

[135] S. Asokan and C. S. Seelamantula, "Teaching a gan what not to learn," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20, Vancouver, BC, Canada: Curran Associates Inc., 2020. DOI: 10.5555/3495724.3496058.

[136] Y. Ban, R. Wang, T. Zhou, M. Cheng, B. Gong, and C.-J. Hsieh, "Understanding the impact of negative prompts: When and how do they take effect?" In *Computer Vision – ECCV 2024: 18th European Conference, Milan, Italy, September 29–October 4, 2024, Proceedings, Part LXXXIX*, Berlin, Heidelberg: Springer-Verlag, 2024, pp. 190–206. DOI: 10.1007/978-3-031-73024-5_12.

[137] N. Liu, S. Li, Y. Du, A. Torralba, and J. B. Tenenbaum, "Compositional visual generation with composable diffusion models," in *Computer Vision – ECCV 2022*, S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, Eds., Cham: Springer Nature Switzerland, 2022, pp. 423–439, ISBN: 978-3-031-19790-1. DOI: 10.1007/978-3-031-19790-1_26.

[138] R. Tang *et al.*, "What the DAAM: Interpreting stable diffusion using cross attention," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds., Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 5644–5659. DOI: 10.18653/v1/2023.acl-long.310.

[139] A. Zunger, "Inverse design in search of materials with target functionalities," *Nature Reviews Chemistry*, vol. 2, no. 4, p. 0121, Mar. 2018. DOI: 10.1038/s41570-018-0121.

[140] B. Deng, A. Zareei, X. Ding, J. C. Weaver, C. H. Rycroft, and K. Bertoldi, "Inverse design of mechanical metamaterials with target nonlinear response via a neural accelerated evolution strategy," *Advanced Materials*, vol. 34, no. 41, p. 2 206 238, 2022. DOI: 10.1002/adma.202206238.

[141] E. Fatehi, H. Yazdani Sarvestani, B. Ashrafi, and A. Akbarzadeh, "Accelerated design of architectured ceramics with tunable thermal resistance via a hybrid machine learning and finite element approach," *Materials & Design*, vol. 210, p. 110 056, 2021. DOI: 10.1016/j.matdes.2021.110056.

[142] T. Gärtner, M. Fernández, and O. Weeger, "Nonlinear multiscale simulation of elastic beam lattices with anisotropic homogenized constitutive models based on artificial neural networks," *Computational Mechanics*, vol. 68, no. 5, pp. 1111–1130, Nov. 2021. DOI: 10.1007/s00466-021-02061-x.

[143] S. Lee, Z. Zhang, and G. X. Gu, "Generative machine learning algorithm for lattice structures with superior mechanical properties," *Mater. Horiz.*, vol. 9, pp. 952–960, 3 2022. DOI: 10.1039/D1MH01792F.

[144] F. Liu, X. Jiang, X. Wang, and L. Wang, "Machine learning-based design and optimization of curved beams for multistable structures and metamaterials," *Extreme Mechanics Letters*, vol. 41, p. 101 002, 2020, ISSN: 2352-4316. DOI: 10.1016/j.eml.2020.101002.

[145] C. Ma *et al.*, "Accelerated design and characterization of non-uniform cellular materials via a machine-learning based framework," *npj Computational Materials*, vol. 6, no. 1, p. 40, Apr. 2020. DOI: 10.1038/s41524-020-0309-6.

[146] M. Maurizi, C. Gao, and F. Berto, "Inverse design of truss lattice materials with superior buckling resistance," *npj Computational Materials*, vol. 8, no. 1, p. 247, Nov. 2022. DOI: 10.1038/s41524-022-00938-w.

[147] V. Slesarenko, "Bandgap structure in elastic metamaterials with curvy bezier beams," *Applied Physics Letters*, vol. 123, no. 8, p. 081 702, Aug. 2023. DOI: 10.1063/5.0156529.

[148] T. Xue *et al.*, "A data-driven computational scheme for the nonlinear mechanical properties of cellular mechanical metamaterials under large deformation," *Soft Matter*, vol. 16, pp. 7524–7534, 32 2020. DOI: 10.1039/D0SM00488J.

[149] L. Zheng, D. M. Kochmann, and S. Kumar, "Hypercan: Hypernetwork-driven deep parameterized constitutive models for metamaterials," *Extreme Mechanics Letters*, vol. 72, p. 102 243, 2024. DOI: 10.1016/j.eml.2024.102243.

[150] M. D. Bermejillo Barrera, F. Franco-Martínez, and A. Díaz Lantada, "Artificial intelligence aided design of tissue engineering scaffolds employing virtual tomography and 3d convolutional neural networks," *Materials*, vol. 14, no. 18, 2021. DOI: 10.3390/ma14185278.

[151] A. P. Garland, B. C. White, S. C. Jensen, and B. L. Boyce, "Pragmatic generative optimization of novel structural lattice metamaterials with machine learning," *Materials & Design*, vol. 203, p. 109 632, 2021. DOI: https://doi.org/10.1016/j.matdes.2021.109632.

[152] X. Jiang, F. Liu, and L. Wang, "Machine learning-based stiffness optimization of digital composite metamaterials with desired positive or negative poisson's ratio," *Theoretical and Applied Mechanics Letters*, vol. 13, no. 6, p. 100 485, 2023. DOI: 10.1016/j.taml.2023.100485.

[153] M. Zhao *et al.*, "Machine learning accelerated design of lattice metamaterials for customizable energy absorption," *Thin-Walled Structures*, vol. 208, p. 112 845, 2025. DOI: 10.1016/j.tws.2024.112845.

[154] H. T. Kollmann, D. W. Abueidda, S. Koric, E. Guleryuz, and N. A. Sobh, "Deep learning for topology optimization of 2d metamaterials," *Materials & Design*, vol. 196, p. 109 098, 2020. DOI: https://doi.org/10.1016/j.matdes.2020.109098.

[155] S. Bonfanti, R. Guerra, F. Font-Clos, D. Rayneau-Kirkhope, and S. Zapperi, "Automatic design of mechanical metamaterial actuators," *Nature Communications*, vol. 11, no. 1, p. 4162, Aug. 2020. DOI: 10.1038/s41467-020-17947-2.

[156] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges," *arXiv e-prints*, arXiv:2104.13478, Apr. 2021. DOI: 10.48550/arXiv.2104.13478.

[157] L. Zheng, K. Karapiperis, S. Kumar, and D. M. Kochmann, "Unifying the design space and optimizing linear and nonlinear truss metamaterials by generative modeling," *Nature Communications*, vol. 14, no. 1, p. 7563, Nov. 2023. DOI: 10.1038/s41467-023-42068-x.

[158] T. Xue, S. Adriaenssens, and S. Mao, "Learning the nonlinear dynamics of mechanical metamaterials with graph networks," *International Journal of Mechanical Sciences*, vol. 238, p. 107 835, 2023. DOI: 10.1016/j.ijmecsci.2022.107835.

[159] P. P. Meyer, C. Bonatti, T. Tancogne-Dejean, and D. Mohr, "Graph-based metamaterials: Deep learning of structure-property relations," *Materials & Design*, vol. 223, p. 111 175, 2022. DOI: 10.1016/j.matdes.2022.111175.

[160] P. Prashant Indurkar, S. Karlapati, A. Jyoti Dipanka Shaikeea, and V. S. Deshpande, "Predicting deformation mechanisms in architected metamaterials using GNN," *arXiv e-prints*, arXiv:2202.09427, Feb. 2022. DOI: 10.48550/arXiv.2202.09427.

[161] S. Kumar, S. Tan, L. Zheng, and D. M. Kochmann, "Inverse-designed spinodoid metamaterials," *npj Computational Mathematics*, vol. 6, 73, p. 73, 2020. DOI: 10.1038/s41524-020-0341-6.

[162] N. A. Alderete, N. Pathak, and H. D. Espinosa, "Machine learning assisted design of shape-programmable 3d kirigami metamaterials," *npj Computational Materials*, vol. 8, no. 1, p. 191, Sep. 2022. DOI: 10.1038/s41524-022-00873-w.

[163] L. Wang, Y.-C. Chan, F. Ahmed, Z. Liu, P. Zhu, and W. Chen, "Deep generative modeling for mechanistic-based learning and design of metamaterial systems," *Computer Methods in Applied Mechanics and Engineering*, vol. 372, p. 113 377, 2020. DOI: 10.1016/j.cma.2020.113377.

[164] W. Wang, W. Cheney, and A. V. Amirkhizi, "Generative design of graded metamaterial arrays for dynamic response modulation," *Materials & Design*, vol. 237, p. 112 550, 2024. DOI: 10.1016/j.matdes.2023.112550.

[165] A. Plumerault, H. Le Borgne, and C. Hudelot, "Avae: Adversarial variational auto encoder," in *2020 25th International Conference on Pattern Recognition (ICPR)*, 2021, pp. 8687–8694. DOI: 10.1109/ICPR48806.2021.9412727.

[166] Z. Yang, C.-H. Yu, and M. J. Buehler, "Deep learning model to predict complex stress and strain fields in hierarchical composites," *Science Advances*, vol. 7, no. 15, eabd7416, 2021. DOI: 10.1126/sciadv.abd7416.

[167] J. Park, J. Noh, J. Shin, G. X. Gu, and J. Rho, "Investigating static and dynamic behaviors in 3d chiral mechanical metamaterials by disentangled generative models," *Advanced Functional Materials*, vol. 35, no. 2, p. 2 412 901, 2025. DOI: 10.1002/adfm.202412901.

[168] J. Wang, W. (Chen, D. Da, M. Fuge, and R. Rai, "Ih-gan: A conditional generative model for implicit surface-based inverse design of cellular structures," *Computer Methods in Applied Mechanics and Engineering*, vol. 396, 2022. DOI: 10.1016/j.cma.2022.115060.

[169] Y. Mao, Q. He, and X. Zhao, "Designing complex architectured materials with generative adversarial networks," *Science Advances*, vol. 6, no. 17, eaaz4169, 2020. DOI: 10.1126/sciadv.aaz4169.

[170] F. Mazé and F. Ahmed, "Diffusion models beat gans on topology optimization," in *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in*

Artificial Intelligence, ser. AAAI'23/IAAI'23/EAAI'23, AAAI Press, 2023. DOI: 10.1609/aaai.v37i8.26093.

[171] H. Wang, Z. Du, F. Feng, Z. Kang, S. Tang, and X. Guo, "Diffmat: Data-driven inverse design of energy-absorbing metamaterials using diffusion model," *Computer Methods in Applied Mechanics and Engineering*, vol. 432, p. 117 440, 2024. DOI: 10.1016/j.cma.2024.117440.

[172] J.-H. Bastek and D. M. Kochmann, "Inverse design of nonlinear mechanical metamaterials via video denoising diffusion models," *Nature Machine Intelligence*, vol. 5, no. 12, pp. 1466–1475, Dec. 2023. DOI: 10.1038/s42256-023-00762-x.

[173] A. Alvarez-Trejo, E. Cuan-Urquizo, D. Bhate, and A. Roman-Flores, "Mechanical metamaterials with topologies based on curved elements: An overview of design, additive manufacturing and mechanical properties," *Materials & Design*, vol. 233, p. 112 190, 2023. DOI: 10.1016/j.matdes.2023.112190.

[174] C. Lu *et al.*, "Architectural design and additive manufacturing of mechanical metamaterials: A review," *Engineering*, vol. 17, pp. 44–63, 2022. DOI: 10.1016/j.eng.2021.12.023.

[175] V. Deshpande, M. Ashby, and N. Fleck, "Foam topology: Bending versus stretching dominated architectures," *Acta Materialia*, vol. 49, no. 6, pp. 1035–1040, 2001. DOI: 10.1016/S1359-6454(00)00379-7.

[176] J. U. Surjadi *et al.*, "Mechanical metamaterials and their engineering applications," *Advanced Engineering Materials*, vol. 21, no. 3, p. 1 800 864, 2019. DOI: 10.1002/adem.201800864.

[177] S. Ali, M. N. Sheikh, and M. N. S. Hadi, "Behavior of axially loaded plain and fiber-reinforced geopolymer concrete columns with glass fiber-reinforced polymer cages," *Structural Concrete*, vol. 22, no. 3, pp. 1800–1816, 2021. DOI: 10.1002/suco.202000231.

[178] M. Ebrahimi, R. Hashemi, and E. Etemadi, "In-plane energy absorption characteristics and mechanical properties of 3d printed novel hybrid cellular structures," *Journal of Materials Research and Technology*, vol. 20, pp. 3616–3632, 2022. DOI: 10.1016/j.jmrt.2022.08.064.

[179] "Improved mechanical characteristics of new auxetic structures based on stretch-dominated-mechanism deformation under compressive and tensile loadings," *Thin-Walled Structures*, vol. 184, p. 110 491, 2023. DOI: 10.1016/j.tws.2022.110491.

[180] Y. Bai *et al.*, "Mechanical properties of a chiral cellular structure with semicircular beams," *Materials*, vol. 14, no. 11, 2021. DOI: 10.3390/ma14112887.

[181] B. Balakrisnan, A. Nacev, J. M. Burke, A. Dasgupta, and E. Smela, "Design of compliant meanders for applications in mems, actuators, and flexible electronics," *Smart Materials and Structures*, vol. 21, no. 7, p. 075 033, Jun. 2012. DOI: 10.1088/0964-1726/21/7/075033.

[182] C. Gonzalez and J. Lorca, "Stiffness of a curved beam subjected to axial load and large displacements," *International Journal of Solids and Structures*, vol. 42, no. 5, pp. 1537–1545, 2005. DOI: 10.1016/j.ijsolstr.2004.08.018.

[183] Y. Fu and W. Liu, "Design of mechanical metamaterial with controllable stiffness using curved beam unit cells," *Composite Structures*, vol. 258, p. 113 195, 2021. DOI: 10.1016/j.compstruct.2020.113195.

[184] H. Yin, D. Guo, G. Wen, and Z. Wu, "On bending crashworthiness of smooth-shell lattice-filled structures," *Thin-Walled Structures*, vol. 171, p. 108 800, 2022. DOI: 10.1016/j.tws.2021.108800.

[185] Y. Wu, L. Sun, P. Yang, J. Fang, and W. Li, "Energy absorption of additively manufactured functionally bi-graded thickness honeycombs subjected to axial loads," *Thin-Walled Structures*, vol. 164, p. 107 810, 2021. DOI: 10.1016/j.tws.2021.107810.

[186] Y. Chen, T. Li, F. Scarpa, and L. Wang, "Lattice metamaterials with mechanically tunable poisson's ratio for vibration control," *Phys. Rev. Appl.*, vol. 7, p. 024 012, 2 Feb. 2017. DOI: 10.1103/PhysRevApplied.7.024012.

[187] M. Lei *et al.*, "3d printing of auxetic metamaterials with digitally reprogrammable shape," *ACS Applied Materials & Interfaces*, vol. 11, no. 25, pp. 22 768–22 776, 2019. DOI: 10.1021/acsami.9b06081.

[188] K. Meena and S. Singamneni, "A new auxetic structure with significantly reduced stress concentration effects," *Materials & Design*, vol. 173, p. 107 779, 2019. DOI: 10.1016/j.matdes.2019.107779.

[189] Y. Yuan *et al.*, "3d printed auxetic metamaterials with tunable mechanical properties and morphological fitting abilities," *Materials & Design*, vol. 244, p. 113 119, 2024. DOI: 10.1016/j.matdes.2024.113119.

[190] M. Kucewicz, P. Baranowski, M. Stankiewicz, M. Konarzewski, P. Platek, and J. Malachowski, "Modelling and testing of 3d printed cellular structures under quasi-static and dynamic conditions," *Thin-Walled Structures*, vol. 145, p. 106 385, 2019. DOI: 10.1016/j.tws.2019.106385.

[191] X. Yang, Y. Sun, J. Yang, and Q. Pan, "Out-of-plane crashworthiness analysis of bio-inspired aluminum honeycomb patterned with horseshoe mesostructure," *Thin-Walled Structures*, vol. 125, pp. 1–11, 2018. DOI: 10.1016/j.tws.2018.01.014.

[192] Z.-P. Wang and L. H. Poh, "Optimal form and size characterization of planar isotropic petal-shaped auxetics with tunable effective properties using iga," *Composite Structures*, vol. 201, pp. 486–502, 2018. DOI: 10.1016/j.compstruct.2018.06.042.

[193] S. Mukherjee and S. Adhikari, "The in-plane mechanics of a family of curved 2D lattices," en, *Composite Structures*, vol. 280, p. 114 859, 2022. DOI: 10.1016/j.compstruct.2021.114859.

[194] A. Harkati, E. H. Harkati, A. Bezazi, F. Scarpa, and M. Ouisse, "Out-of-plane elastic constants of curved cell walls honeycombs," en, *Composite Structures*, vol. 268, p. 113 959, 2021. DOI: 10.1016/j.compstruct.2021.113959.

[195] T. Hughes, J. Cottrell, and Y. Bazilevs, "Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement," *Computer Methods in Applied Mechanics and Engineering*, vol. 194, no. 39, pp. 4135–4195, 2005. DOI: 10.1016/j.cma.2004.10.008.

[196] K. A. Johannessen and E. Fonn, "Splipy: B-spline and nurbs modelling in python," *Journal of Physics: Conference Series*, vol. 1669, no. 1, p. 012 032, Oct. 2020. DOI: 10.1088/1742-6596/1669/1/012032.

[197] J. Lei Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," *arXiv e-prints*, arXiv:1607.06450, Jul. 2016. DOI: 10.48550/arXiv.1607.06450.

[198] L. Jin and S. Yang, "Engineering kirigami frameworks toward real-world applications," en, *Advanced Materials*, p. 2 308 560, 2023. DOI: 10.1002/adma.202308560.

[199] D. Misseroni, P. P. Pratapa, K. Liu, and G. H. Paulino, "Experimental realization of tunable Poisson's ratio in deployable origami metamaterials," *Extreme Mechanics Letters*, vol. 53, p. 101 685, 2022. DOI: 10.1016/j.eml.2022.101685.

[200] N. Yang, M. Zhang, and R. Zhu, "3d kirigami metamaterials with coded thermal expansion properties," *Extreme Mechanics Letters*, vol. 40, p. 100 912, 2020. DOI: 10.1016/j.eml.2020.100912.

[201] Z. Zhai, L. Wu, and H. Jiang, "Mechanical metamaterials based on origami and kirigami," *Applied Physics Reviews*, vol. 8, no. 4, p. 041 319, 2021. DOI: 10.1063/5.0051088.

[202] T. Liu, S. Sun, H. Liu, N. An, and J. Zhou, "A predictive deep-learning approach for homogenization of auxetic kirigami metamaterials with randomly oriented cuts," *Modern Physics Letters B*, vol. 35, no. 01, p. 2 150 033, 2021. DOI: 10.1142/S0217984921500330.

[203] P. Z. Hanakata, E. D. Cubuk, D. K. Campbell, and H. S. Park, "Forward and inverse design of kirigami via supervised autoencoder," *Phys. Rev. Res.*, vol. 2, p. 042 006, 4 Oct. 2020. DOI: 10.1103/PhysRevResearch.2.042006.

[204] A. Rafsanjani and D. Pasini, "Bistable auxetic mechanical metamaterials inspired by ancient geometric motifs," *Extreme Mechanics Letters*, vol. 9, pp. 291–296, 2016. DOI: 10.1016/j.eml.2016.09.001.

[205] J. Tao, H. Khosravi, V. Deshpande, and S. Li, "Engineering by cuts: How kirigami principle enables unique mechanical properties and functionalities," *Advanced Science*, vol. 10, no. 1, p. 2 204 733, 2023. DOI: 10.1002/advs.202204733.

[206] Y. Cho *et al.*, "Engineering the shape and structure of materials by fractal cut," *Proceedings of the National Academy of Sciences*, vol. 111, no. 49, pp. 17 390–17 395, 2014. DOI: 10.1073/pnas.1417276111.

[207] M. Isobe and K. Okumura, "Initial rigid response and softening transition of highly stretchable kirigami sheet materials," *Scientific Reports*, vol. 6, no. 1, p. 24 758, Apr. 2016. DOI: 10.1038/srep24758.

[208] G. Felsch and V. Slesarenko, "Generative models struggle with kirigami metamaterials," *Scientific Reports*, vol. 14, no. 1, p. 19 397, Aug. 2024. DOI: 10.1038/s41598-024-70364-z.

[209] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012. DOI: 10.1109/MSP.2012.2211477.

[210] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[211] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, Lille, France: JMLR.org, 2015, pp. 448–456. DOI: 10.5555/3045118.3045167.

[212] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014. DOI: 10.5555/2627435.2670313.

[213] G. E. Hinton, "Learning distributed representations of concepts.," in *Proceedings of the Annual Meeting of the Cognitive Science Society, 1986*, 1986.

[214] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951. DOI: 10.1214/aoms/1177729694.

[215] L. V. Kantorovich, "Mathematical methods of organizing and planning production," *Management Science*, vol. 6, no. 4, pp. 366–422, 1960. DOI: 10.1287/mnsc.6.4.366.

[216] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA: Curran Associates Inc., 2019. DOI: 10.5555/3454287.3455008.

[217] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," *arXiv e-prints*, arXiv:1511.06434, Nov. 2015. DOI: 10.48550/arXiv.1511.06434.

[218] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, Springer, 2015, pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28.

[219] L. Mescheder, A. Geiger, and S. Nowozin, "Which training methods for GANs do actually converge?" In *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, PMLR, Oct. 2018, pp. 3481–3490.

[220] P. Ren *et al.*, "A survey of deep active learning," *ACM Comput. Surv.*, vol. 54, no. 9, Oct. 2021. DOI: 10.1145/3472291.

[221] D. Khatamsaz, B. Vela, P. Singh, D. D. Johnson, D. Allaire, and R. Arróyave, "Bayesian optimization with active learning of design constraints using an entropy-based approach," *npj Computational Materials*, vol. 9, no. 1, p. 49, Apr. 2023. DOI: 10.1038/s41524-023-01006-7.

[222] D. Lee, Y.-C. Chan, W. (Chen, L. Wang, A. van Beek, and W. Chen, "T-metaset: Task-aware acquisition of metamaterial datasets through diversity-based active learning," *Journal of Mechanical Design*, vol. 145, no. 3, p. 031 704, Nov. 2022. DOI: 10.1115/1.4055925.

[223] A. A. A. Elhag, Y. Wang, J. M. Susskind, and M. Á. Bautista, "Manifold diffusion fields," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=BZtEthuXRF.

[224] C.-W. Huang, M. Aghajohari, A. J. Bose, P. Panangaden, and A. Courville, "Riemannian diffusion models," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS '22, New Orleans, LA, USA: Curran Associates Inc., 2022. DOI: 10.5555/3600270.3600469.

[225] P. Zhuang, S. Abnar, J. Gu, A. Schwing, J. M. Susskind, and M. Á. Bautista, "Diffusion probabilistic fields," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=ik91mY-2GN.

[226] A. S. Bhuwal, Y. Pang, I. Ashcroft, W. Sun, and T. Liu, "Discovery of quasi-disordered truss metamaterials inspired by natural cellular materials," *Journal of the Mechanics and Physics of Solids*, vol. 175, p. 105 294, 2023. DOI: 10.1016/j.jmps.2023.105294.

[227] K. Liu, R. Sun, and C. Daraio, "Growth rules for irregular architected materials with programmable properties," *Science*, vol. 377, no. 6609, pp. 975–981, 2022. DOI: 10.1126/science.abn1459.

[228] M. Zaiser and S. Zapperi, "Disordered mechanical metamaterials," *Nature Reviews Physics*, vol. 5, no. 11, pp. 679–688, Nov. 2023. DOI: 10.1038/s42254-023-00639-3.

[229] Z. Jia, F. Liu, X. Jiang, and L. Wang, "Engineering lattice metamaterials for extreme property, programmability, and multifunctionality," *Journal of Applied Physics*, vol. 127, no. 15, p. 150 901, Apr. 2020. DOI: 10.1063/5.0004724.

[230] M. Pham, C. Liu, I. Todd, and J. Lertthanasarn, "Damage-tolerant architected materials inspired by crystal microstructure," *Nature*, vol. 565, p. 305 311, Jan. 2019. DOI: 10.1038/s41586-018-0850-3.

[231] X. Shu, Y. Mao, M. Lei, D. Da, S. Hou, and P. Zhang, "Toughness enhancement of honeycomb lattice structures through heterogeneous design," *Materials & Design*, vol. 217, p. 110 604, 2022, ISSN: 0264-1275. DOI: 10.1016/j.matdes.2022.110604.

[232] R. Alberdi *et al.*, "Multi-morphology lattices lead to improved plastic energy absorption," *Materials & Design*, vol. 194, p. 108 883, 2020. DOI: 10.1016/j.matdes.2020.108883.

[233] Y. Jia, K. Liu, and X. Zhang, "Modulate stress distribution with bio-inspired irregular architected materials towards optimal tissue support," *Nature Communications*, vol. 15, p. 4072, May 2024. DOI: 10.1038/s41467-024-47831-2.

[234] D. Aranguren van Egmond *et al.*, "The benefits of structural disorder in natural cellular solids," *arXiv e-prints*, arXiv:2110.04607, Oct. 2021. DOI: 10.48550/arXiv.2110.04607.

[235] S. Yadav, S. Liu, R. K. Singh, A. K. Sharma, and P. Rawat, "A state-of-art review on functionally graded materials (fgms) manufactured by 3d printing techniques: Advantages, existing challenges, and future scope," *Journal of Manufacturing Processes*, vol. 131, pp. 2051–2072, 2024. DOI: 10.1016/j.jmapro.2024.10.026.

[236] A. Kawasaki and R. Watanabe, "Concept and p/m fabrication of functionally gradient materials," *Ceramics International*, vol. 23, no. 1, pp. 73–83, 1997. DOI: 10.1016/0272-8842(95)00143-3.

[237] A. Mortensen and S. Suresh, "Functionally graded metals and metal-ceramic composites: Part 1 processing," *International Materials Reviews*, vol. 40, no. 6, pp. 239–265, 1995. DOI: 10.1179/imr.1995.40.6.239.

[238] D. Chen, K. Gao, J. Yang, and L. Zhang, "Functionally graded porous structures: Analyses, performances, and applications - a review," *Thin-Walled Structures*, vol. 191, p. 111 046, 2023. DOI: 10.1016/j.tws.2023.111046.

[239] S. Alkunte *et al.*, "Functionally graded metamaterials: Fabrication techniques, modeling, and applications—a review," *Processes*, vol. 12, no. 10, 2024, ISSN: 2227-9717. DOI: 10.3390/pr12102252.

[240] C. Jansari, S. P. Bordas, M. Montemurro, and E. Atroshchenko, "Design of thermal meta-structures made of functionally graded materials using isogeometric density-based topology optimization," *Composite Structures*, vol. 364, p. 119 114, 2025. DOI: 10.1016/j.compstruct.2025.119114.

[241] Hirshikesh, S. Natarajan, and E. T. Ooi, "Fatigue crack growth in functionally graded materials using an adaptive phase field method with cycle jump scheme," *Engineering Fracture Mechanics*, vol. 312, p. 110 573, 2024. DOI: 10.1016/j.engfracmech.2024.110573.

[242] A. Coluccia, G. Meyer, S. Liseni, C. Mittelstedt, and G. De Pasquale, "Functionally graded lattice structures for energy absorption: Numerical analysis and experimental validation," *Composite Structures*, vol. 360, p. 119 013, 2025. DOI: 10.1016/j.compstruct.2025.119013.

[243] C. S. Ha *et al.*, "Rapid inverse design of metamaterials based on prescribed mechanical behavior through machine learning," *Nature Communications*, vol. 14, no. 1, p. 5765, Sep. 2023. DOI: 10.1038/s41467-023-40854-1.

[244] K. K. Dudek, M. Kadic, C. Coulais, and K. Bertoldi, "Shape morphing metamaterials," *arXiv e-prints*, arXiv:2501.14804, Jan. 2025. DOI: 10.48550/arXiv.2501.14804.

[245] Q. Guo *et al.*, "Programmable 3d self-folding structures with strain engineering," *Advanced Intelligent Systems*, vol. 2, no. 12, p. 2 000 101, 2020. DOI: 10.1002/aisy.202000101.

[246] J. A. Faber, A. F. Arrieta, and A. R. Studart, "Bioinspired spring origami," *Science*, vol. 359, no. 6382, pp. 1386–1391, 2018. DOI: 10.1126/science.aap7753.

[247] A. A. Zadpoor and J. Malda, "Additive manufacturing of biomaterials, tissues, and organs," *Annals of Biomedical Engineering*, vol. 45, no. 1, pp. 1–11, Jan. 2017. DOI: 10.1007/s10439-016-1719-y.

[248] A. Reuss, "Berechnung der fließgrenze von mischkristallen aufgrund der plastizitätsbedingung für einkristalle," *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 9, no. 1, pp. 49–58, 1929. DOI: 10.1002/zamm.19290090104.

[249] J. Yvonnet and G. Bonnet, "A consistent nonlocal scheme based on filters for the homogenization of heterogeneous linear materials with non-separated scales," *International Journal of Solids and Structures*, vol. 51, no. 1, pp. 196–209, 2014. DOI: 10.1016/j.ijsolstr.2013.09.023.

Appendix A

A.1 Graphs

A directed graph G is a tuple $G = (V, A)$, where V is a set of nodes and $A \subseteq V \times V$ is a set of directed edges—connections between two nodes that are represented as ordered pairs. If $(u, v) \in A$, then node u is called the *parent* of node v, and v is the *child* of u. A directed graph is called a *Directed Acyclic Graph (DAG)* if it contains no directed cycles—that is, there is no non-empty sequence of directed edges $(v^{(1)}, v^{(2)}), (v^{(2)}, v^{(3)}), \ldots, (v^{(k-1)}, v^{(k)})$ such that $v^{(1)} = v^{(k)}$.

A.2 Network Architectures (Chapter 4)

The tables below show the neural network architectures used in Chapter 4.

VAE Encoder

Layer	Kernel	Strides	Features	BN	Activation	Padding	Previous
Input I1 6×6	-	-	1	-	-	-	-
Convolution C1	3×3	1×1	8	Yes	LeakyReLU	circular	I1
Convolution C2	5×5	2×2	16	Yes	LeakyReLU	circular	C1
Convolution C3	3×3	1×1	32	Yes	LeakyReLU	circular	C2
Convolution C4	5×5	2×2	64	Yes	LeakyReLU	circular	C3
Flatten F1	-	-	256	No	-	-	C4
Linear L1	-	-	64	No	-	-	F1
Linear L2	-	-	64	No	-	-	F1
Optimizer	Adam lr=1e-4						
Batch size	32						
Epochs	3000						
LeakyReLU slope	0.2						

Table A.1: Dimensions and training hyperparameters of the Encoder of the VAE

VAE Decoder + GAN Generator + WGAN Generator

Layer	Kernel	Strides	Features	BN	Activation	Padding	Previous
Input I1	-	-	64	-	-	-	-
Linear L1	-	-	256	No	ReLU	-	I1
Reshape R1 2×2	-	-	64	-	-	-	L1
Transposed Conv T1	3×3	1×1	32	No	ReLU	zeros	R1
Convolution C1	3×3	1×1	32	Yes	ReLU	circular	T1
Transposed ConvT2	3×3	1×1	32	No	ReLU	zeros	C1
Convolution C2	3×3	1×1	1	No	Tanh / None (WGAN)	circular	T2
Optimizer	Adam lr=1e-4						
Batch size	32						
Epochs	3000						

Table A.2: Dimensions and training hyperparameters of the generative parts of the VAE, GAN and WGAN

DDPM

Layer	Kernel	Strides	Features	BN	Activation	Padding	Previous
Input I1 6×6	-	-	1	-	-	-	-
Convolution C1	3×3	1×1	256	Yes	GELU	circular (1)	I1
Convolution C2	3×3	1×1	256	Yes	GELU	circular (1)	C1
Convolution C3	3×3	1×1	256	Yes	GELU	circular (2)	C2
Convolution C4	3×3	1×1	256	Yes	GELU	circular (2)	C3
MaxPool M1	2×2	-	256	-	-	-	C4
Convolution C5	3×3	1×1	256	Yes	GELU	circular (1)	M1
Convolution C6	3×3	1×1	512	Yes	GELU	circular (1)	C5
MaxPool M2	2×2	-	512	-	-	-	C6
AvgPool A1	2×2	-	512	-	GELU	-	M2
Transposed Conv T1	2×2	2×2	512	-	-	-	A1
GroupNorm G1	8×8	-	512	-	ReLU	-	T1
Input I2 (t)	-	-	1	-	-	-	-
Linear L1	-	-	512	No	GELU	-	I2
Linear L2	-	-	512	No	GELU	-	L1
Linear L3	-	-	512	No	GELU	-	I2
Linear L4	-	-	512	No	GELU	-	L3
Transposed Conv T2	2×2	2×2	256	No	-	-	G1 + L2
Convolution C7	3×3	1×1	256	No	GELU	circular (1)	T2 + C6
Convolution C8	3×3	1×1	256	No	GELU	circular (1)	C7 + C6
MaxPool M3	2×2	-	256	-	-	-	C8
Transposed Conv T3	3×3	1×1	256	No	-	-	M3 + L4
Convolution C9	3×3	1×1	256	No	GELU	None	T3 + C4
Convolution C10	3×3	1×1	256	No	GELU	None	C9
MaxPool M4	2×2	-	256	-	-	-	C10
Convolution C11	3×3	1×1	256	No	LeakyReLU	circular (1)	M4 + C1
GroupNorm G2	8×8	-	256	-	ReLU	-	C11
Convolution C12	3×3	1×1	1	No	LeakyReLU	circular (1)	G2
Optimizer	Adam lr=1e-4						
Batch size	32						
Epochs	3000						
Number of time steps	400						
Noise schedule	$\beta_1 = 1e-4$, $\beta_2 = 0.02$						

Table A.3: Dimensions and training hyperparameters of the DDPM

GAN + WGAN Discriminator

Layer	Kernel	Strides	Features	BN	Activation	Padding	Previous
Input I1 6×6	-	-	1	-	-	-	-
Convolution C1	3×3	1×1	16	Yes	LeakyReLU	circular	I1
Convolution C2	5×5	2×2	32	Yes	LeakyReLU	circular	C1
Convolution C3	3×3	1×1	64	Yes	LeakyReLU	circular	C2
Convolution C4	5×5	2×2	128	Yes	LeakyReLU	circular	C3
Flatten F1	-	-	512	No	-	-	C4
Linear L1	-	-	1	No	Sigmoid / None (WGAN)	-	F1
Optimizer	Adam lr=1e-4						
Batch size	32						
Epochs	3000						
LeakyReLU slope	0.2						

Table A.4: Dimensions and training hyperparameters of the discriminators of GAN and WGAN