



Neural networks vs. splines: advances in numerical extruder design

Jaewook Lee¹ · Sebastian Hube² · Stefanie Elgeti^{1,2}

Received: 26 January 2023 / Accepted: 23 April 2023
© The Author(s) 2023

Abstract

In this paper, we present a novel approach to geometry parameterization that we apply to the design of mixing elements for single-screw extruders. The approach uses neural networks of a specific architecture to automatically learn an appropriate parameterization. This stands in contrast to the so far common user-defined parameterizations. Geometry parameterization is crucial in enabling efficient shape optimization as it allows for optimizing complex shapes using only a few design variables. Recent approaches often utilize computer-aided design (CAD) data in conjunction with spline-based methods where the spline's control points serve as design variables. Consequently, these approaches rely on the design variables specified by the human designer. This approach results in a significant amount of manual tuning to define a suitable parameterization. In addition, despite this effort, many times the optimization space is often limited to shapes in close proximity to the initial shape. In particular, topological changes are usually not feasible. In this work, we propose a method that circumvents this dilemma by providing low-dimensional, yet flexible shape parametrization using a neural network, which is independent of any computational mesh or analysis methods. Using the neural network for the geometry parameterization extends state-of-the-art methods in that the resulting design space is not restricted to user-prescribed modifications of certain basis shapes. Instead, within the same optimization space, we can interpolate between and explore seemingly unrelated designs. To show the performance of this new approach, we integrate the developed shape parameterization into our numerical design framework for dynamic mixing elements in plastics' extrusion. Finally, we challenge the novel method in a competitive setting against current free-form deformation-based approaches and demonstrate the method's performance even at this early stage.

Keywords Shape optimization · Single-screw extruder · Neural networks · Mixing · Filter · Geometry parameterization

1 Introduction

Modern numerical design is boosted by high-performance computers and the advent of neural networks. While neural networks are well-established in fields such as image recognition, their power to further polymer processing is yet to be

fully discovered. This work attempts to contribute towards this goal. We combine deep neural networks with established shape-optimization methods to enhance mixing in single-screw extruders via a novel numerical design.

In many polymer processing steps, screw-based machines play a crucial role. Screws are, e.g., used as plasticators to prepare polymer melts for injection molding or in extruders in profile extrusion. For simplicity, we will, in the remainder, summarize all such screw-based machines as *extruders*. Single-screw extruders (SSEs) are especially widespread among the many variants of extruders for their economic advantages and simple operation. Economics also drives current attempts to further increase the throughput. This increase is achieved using fast-rotating extruders. However, the current SSE's poor mixing ability has limited the advances and, therefore, improving the mixing ability is a topic of research [1–6].

Special focus is put on improved mixing elements that alleviate this limitation. Approaches to improve mixing

✉ Jaewook Lee
jaewook.lee@tuwien.ac.at

Sebastian Hube
hube@cats.rwth-aachen.de

Stefanie Elgeti
stefanie.elgeti@tuwien.ac.at

¹ Institute of Lightweight Design and Structural Biomechanics (E317), TU Wien, Gumpendorfer Str. 7, 1060 Vienna, Austria

² Chair for Computational Analysis of Technical Systems (CATS), RWTH Aachen University, Schinkelstr. 2, 52062 Aachen, Germany

elements have been proposed based on analytical derivations, experimental, and simulation-based works. In the following, we review recent developments in these three areas. Subsequently, we outline relevant developments in the field of neural networks and, finally, motivate the use of neural nets in the numerical design of mixing elements.

Due to the high pressures and temperatures, analyzing the flow inside extruders is a difficult task. Early studies thus focus on analytical models and geometrically simpler screw sections, e.g., the metering section [7]. Experiments complement these theoretical derivations and allow extending the analysis to more complex screw sections. As reported by Gale, typical configurations rely on photomicrographs of the solidified melt [2] that allow either investigating cross sections of the flow channel or the extrudate. One example of such flow channel photomicrographs is Kim and Kwon's pioneering work on barrier screws via cold-screw extrusion [8]. Apart from investigating solidified melt streams, attempts to analyze the melt flow during the actual operation of extruders are occasionally reported, e.g., by Wong et al. [9]. Despite the great success of such experiments, a standard limitation is their focus on a single operating condition. In contrast, numerical analysis allows studying different designs and operating points at significantly reduced costs and, therefore, proliferates. In the following, we give an overview of such numerical analyses.

One early example is Kim and Kwon's quasi-three-dimensional finite-element (FE) simulation of the striation formation, studying the influence of the barrier flight [10]. Another example is the work by Domingues et al., who obtain global mixing indices for dispersive and distributive mixing in both liquid–liquid and solid–liquid systems [11]. Utilizing a two-dimensional simplification, their simulation domain extends from the hopper to the metering section, and their framework even allows for design optimization.

While these early works typically neglect mixing sections, studying the influence of mixers has recently become a vital research topic. Celik et al. use three-dimensional flow simulation coupled with a particle-tracking approach to determine the degree of mixing based on a deformation-based index [1]. Another example is Marschik et al.'s study comparing different Block-Head mixing screws in distributive and dispersive mixing [6]. A comparable study—focused on the mixing capabilities of different pineapple mixers—is reported by Roland et al. [3]. Both works rely on three-dimensional non-Newtonian flow simulations. Besides such works towards the numerical assessment of *given* screw designs, numerical *design* is also reported, however, partially in other fields of polymer processing. For example, Elgeti et al. aim for balanced dies and reduced die swell by applying shape optimization [12, 13]. Design by optimization is also reported by Gaspar-Cunha and Covas, who alter the length of the feed

and compression zones, the internal screw diameters of the feed and metering zone, the screw pitch, and the flight clearance [14]. Potente and Többen report another recent study devoted to mixing elements that develops empirical models for shearing sections' pressure-throughput and power consumption for numerical design [15]. Finally, a first approach combining the shape-optimization methods inspired by [12] with a mixing-quantifying objective function to design mixing sections is reported in [16].

However, the shape optimizations above share one commonality: they essentially only modify predefined geometry features. This is accepted in many cases like die or mold design, where the final product's shape is close to the initial one (i.e., the shape variation is small). However, topologically flexible shape parameterizations offer far greater optimization gains for mixing element design, because the optimal geometry might differ significantly from the initial shape. The achievable improvements motivate research on geometry parametrization.

Established shape-parameterization approaches include radial basis functions (RBF) [17], surface parameterizations using Bezier surfaces [18], and surface splines [19]. All these methods may be understood as *filters* that parameterize a geometry by a few variables at the price of a lack of local control. The use of surface splines in shape optimizations can also be found in [12, 13]. A similar concept to surface splines is free-form deformation (FFD) [20] that encapsulates the body-to-deform in a volumetric spline, which allows tailoring the spline further towards an efficient optimization. An alternative approach that does, however, not parameterize the geometry as a filter is given using the computational grid's mesh nodes as shape parameters [21]. Fortunately, with the advent of neural networks, novel means of shape parameterizations offering outstanding flexibility emerged. Finalizing the introduction, we will summarize the most relevant works in this field.

Many neural networks are essentially classifiers. These neural networks are non-linear algorithms that are optimized, (i.e., trained), to determine—possibly counterintuitive—similarities and dissimilarities to discriminate between objects. One typical use case is image recognition using red–green–blue (RGB) pixel data. Neural networks can, however, be trained to classify features far beyond RGB-pixel values. One example is style transfer or texture synthesis [22]: instead of aiming at reproducing *pixel* data, output images are generated in combination with *perceptual* data. This allows image transformations, where one image's style is transferred to the motive of another. An extension of these ideas to three-dimensional shapes is first reported by Friedrich et al. [23]. Comparing different shape representations, the authors find that style transfer is applicable to shapes as well.

Our work is especially inspired by Liu et al. [24], who utilize a so-called *Variational Shape Learner*, that learns a voxel representation of three-dimensional shapes. *Learning* here refers to creating a so-called *latent space*, a low-dimensional, feature-rich embedding space to represent and morph between various shapes. Even beyond simple shape interpolation, it is shown that—using the latent representation—geometry features can be transferred from one to another shape. Successful learning of voxel-based shapes can also be found in [25, 26]. In terms of shape representations, pointcloud-based approaches [27–29], which utilize coordinates of three-dimensional point sets, as well as polygonal mesh-based approaches with either template meshes [30, 31] or multiple mesh planes [32] are widely adopted.

While previously mentioned representations show that learning an embedding space of three-dimensional shapes is possible, each work lacks at least one of the following properties: water-tight surfaces, flexible output resolution, and smooth and continuous surface details. Recent works satisfy the aforementioned properties by learning shapes represented by continuous implicit functions, such as signed-distance functions (SDFs) [33] and binary occupancies [34, 35], from which the shapes are extracted as isosurfaces. This work investigates the shape-parameterization capabilities of the DeepSDF auto-decoder [33].

We exploit the feature richness of this latent space as an aid to reduce the optimization space’s dimension for the given mixing-element shape optimization. The important novelty compared to recent spline-based filters is that the neural network finds—possibly counterintuitive—ways to commonly parameterize a set of significantly different shapes irrespective of user-defined design features. This abstraction from the human designer yields low-dimensional yet far more flexible shape parameterizations, which sets the motivation for the work presented here.

This paper is structured as follows: We start in Sect. 2 by summarizing numerical shape optimization and splines, which leads to the concept of geometric filters. Based on that, we explain in Sect. 3 how neural networks can be utilized to create suitable geometry parameterizations for shape optimization. In Sect. 4, we review the utilized software components, summarize the proposed framework’s building blocks, and detail the specific differences to spline-based shape optimization setups. The results obtained from the new approach are presented in Sect. 5, including comparisons to current spline-based designs. Finally, we discuss the results and outline further developments in Sect. 6.

2 Geometric filters as a component of shape-optimization frameworks

The following section discusses shape parameterizations as one building block of numerical shape-optimization frameworks. Therefore, we first introduce the general shape optimization problem. After that, we recall spline-based shape parameterizations. Based on this general introduction of shape-optimization frameworks, we will continue by discussing the specific changes needed to adapt neural nets in Sect. 3.

2.1 Building blocks of numerical shape-optimization frameworks

The general optimization problem is formulated as the minimization of a cost function J that relates the design variables σ to some output—here, the degree of mixing ability obtained with a specific mixing element, (i.e., a particular design). In shape optimization, this minimization problem is typically solved subject to two sets of constraints: (1) inequality and equality conditions, as well as bound constraints on the design variables and (2) partial differential equations (PDEs) that need to be fulfilled by each design to qualify as a feasible solution. This results in the following formulation:

$$J : \mathbb{R}^{n_\sigma} \mapsto \mathbb{R}, \quad (1a)$$

$$\arg \min_{\sigma \in \Sigma \subset \mathbb{R}^n} J(\sigma), \quad (1b)$$

$$\text{s.t. } \mathbf{F}(\sigma) = \mathbf{0} \quad \text{in } \Omega(\sigma), \quad (1c)$$

$$\sigma_i \geq \sigma_{\min,i}, \quad i = 1, \dots, n_\sigma, \quad (1d)$$

$$\sigma_i \leq \sigma_{\max,i}, \quad i = 1, \dots, n_\sigma. \quad (1e)$$

Here, (1d) and (1e) describe bound constraints on the optimization variables σ , whereas (1c) denotes the set of governing PDEs. One approach to numerically solve such a *PDE-constraint* design problem is to alternately compute (1) shape updates and (2) the cost function value. For the studied use case of mixing-element design, this results in the computational steps depicted in Fig. 1.

First, we update the shape (i.e., the simulation domain covering the mixing element). We use this modified computational domain to compute the flow field from which we afterwards infer the objective (i.e., the cost function). The design loop is closed by feeding back the cost function value to the optimization algorithm that now computes an updated shape. This loop continues until any termination criterion,

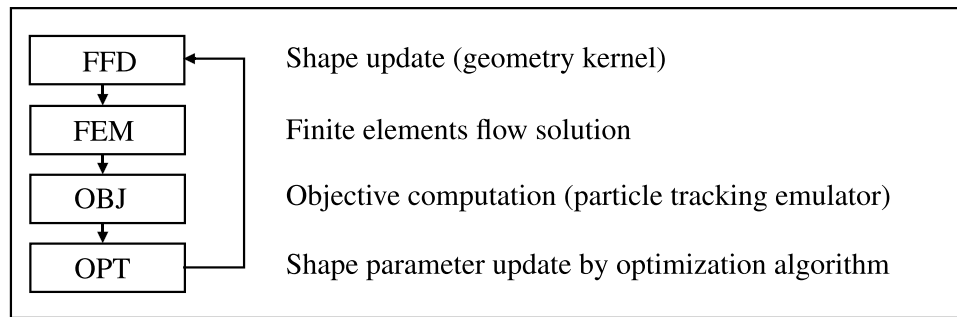


Fig. 1 Building blocks of a shape-optimization framework. The shape is updated by a geometry kernel, such as FFD. Subsequently, the flow field is computed using this updated shape and given as input to the objective calculator. Based on the current design variables and the

design's objective value, the optimization algorithm computes optimized shape parameters and restarts the design loop until at least one termination criterion for the design loop is met

such as a minimal objective decrease, a maximum number of iterations, or another condition, is met.

2.2 Spline-based shape parameterizations

In classical shape-optimization frameworks, the actual shape parameterization, or geometry filtering, is often achieved using splines. The following paragraph, therefore, first provides a summary of splines illustrating how one achieves the filtering. For a detailed description of B-splines, we refer the reader to the book of Piegl and Tiller [19]. After that, we detail on *boundary splines* and FFD as two particular use cases of spline parameterizations.

Splines belong to the group of parametric shape representations. Therefore, each coordinate in the parametric space is connected to one point in physical space. This mapping is best understood using a simple *B-spline* surface that is written as

$$S(\xi, \eta) = \sum_{j=1}^m \sum_{i=1}^n N_{i,r} N_{j,p}(\xi, \eta) \mathbf{B}_{i,j}, \quad (2)$$

where ξ and η denote the parametric coordinates (two for the surface), $N_{i,r}$ denote the interpolation or *basis functions* of order r in the first parametric direction, $N_{j,p}$ denote the basis functions of order p in the second parametric direction, and finally, \mathbf{B} denotes the support or *control points*. Figure 2 illustrates the concept and visualizes how single control points affect the geometry.

The control grid (i.e., the polygon spanned by the control point) aligns with the ξ and η directions, and any parametric coordinate (within the spline's parametric bounds) maps to one point of the blue shape. Consequently, the spline mapping allows controlling an arbitrary number of parametric points by a constant, typically low, number of control points. Being able to control a high number of points with few control points will be the basic idea of filtering using splines.

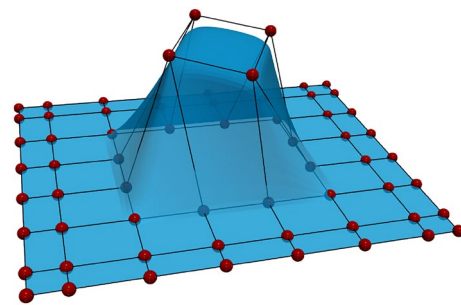


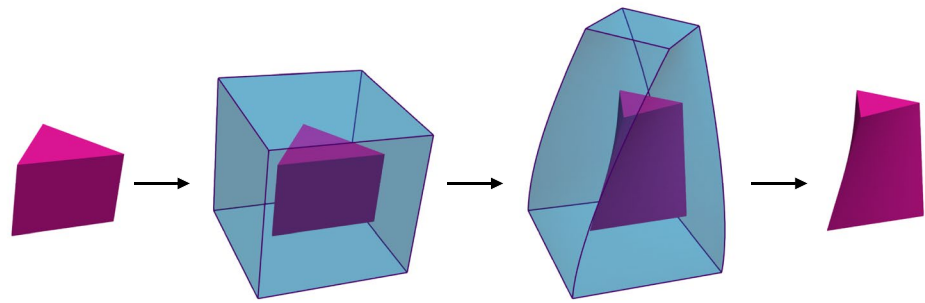
Fig. 2 B-spline representation (blue) obtained from control points (red) for a bi-quadratic B-spline. The upper four control points are rotated, illustrating a possible deformation (colour figure online)

One can obtain geometry parameterizations from splines in multiple ways. As shown in Fig. 2, one way uses the B-splines as a boundary representation. Such spline-based boundary representations are common in CAD. Using these CAD representations, their control points (i.e., the red points in Fig. 2) can be directly used as design variables in shape optimization. However, this use of the CAD's geometry parameterization limits the design process, because a given spline may not be able to represent shapes substantially different from the initial design. Consequently, if modifications of the spline's parameterization, such as inserting additional control point lines, are to be avoided, this limitation restricts the use of the CAD spline to use cases that deal with small shape updates such as *die or mold design* [12].

An alternative to using boundary B-splines is FFD [20]. In FFD, first, an—often volumetric—spline is constructed around the body to be deformed. Second, this volumetric spline is deformed, and finally, the resulting deformation field is imposed on the enclosed body. Figure 3 visualizes this process.

The advantage of FFD is that the spline is constructed irrespective of the enclosed shape, which gives complete

Fig. 3 Free-form deformation using a volumetric spline (light blue) applied to a mixing element (pink). The control points are omitted in this figure. The embedded shape deforms correspondingly to the embedding, simple, volumetric spline (colour figure online)



freedom in choosing degree and resolution. This freedom allows tailoring the spline to the designer's needs (rather than using a given parameterization optimized for CAD usage). Therefore, FFD is widely applied, with just one example being the recent works by Lassila and Rozza combining FFD and reduced order modeling [36]. A combination of both methods, boundary B-splines and FFD, will be compared against the novel shape parameterization based on neural networks that use FFD as a generic interface to modify any given CAD spline, which in turn is used to update the boundary of the simulation domain [16].

3 Shape parametrization using neural networks

As explained in Sect. 2, the prime objective of this work is to investigate how neural networks can be used to encode different shapes in a single set of a few continuous variables. To train the network, thereby determining such a condensed representation, it has to be provided with suitable data. *Suitable* here means that the input data (i.e., shapes) are provided in such a way that the network can learn from these data. In addition—using the same data format—we need to be able to produce high-quality computational meshes from the neural network's output.

In the following, we first introduce deep generative models and then describe a shape representation meeting these two requirements. Finally, we discuss the training data generation and utilization of neural networks as shape generators.

3.1 Deep generative models

With the advent of *generative models*, an alternative approach to shape parameterization emerged. In this subsection, we review two of the most common approaches of generative models, explain their basic concepts and use, and detail how they can be employed for geometric filtering.

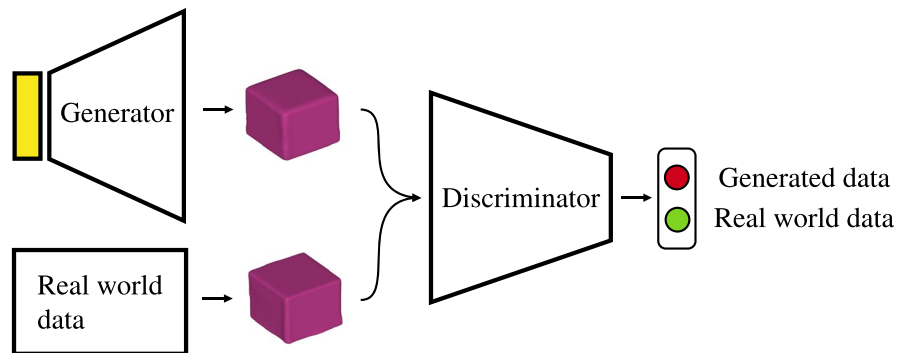
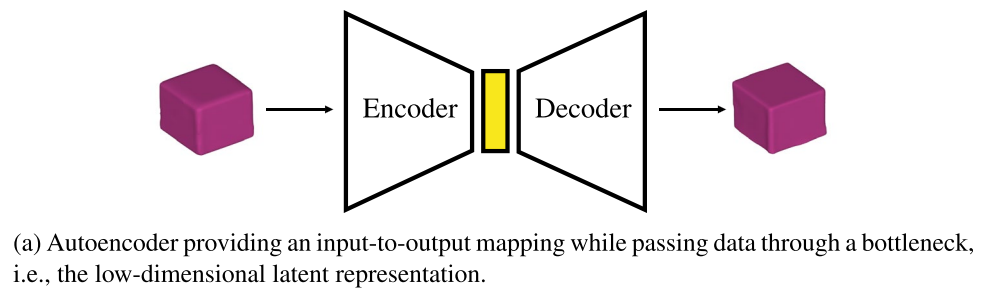
Generative models are an application of neural networks and, thus, in essence, classification algorithms.

Classification here means the ability to determine whether a certain object is in some measure *close* to a specified input. Conversely to just classifying input, such models can also be used to generate an output that resembles an input. *Resemble*, however, needs to be explained. In most applications, the user is not interested in reproducing a given input exactly. Instead, the output should only be *like* the input (i.e., the output should feature a slight variation). Generative models attempt to achieve this goal via statistical modeling. An excellent guide to generative models is found in [37], with special focus on the *Variational Autoencoder* (VAE).

The VAE, like the traditional autoencoder, consists of an *encoder* and a *decoder* and aims to reproduce any given data while passing the input through a bottleneck. However, its probabilistic formulation using the so-called “reparametrization trick” provides an exceptional advantage over the traditional autoencoder in practice [38]. The roles of the encoder and the decoder can be interpreted as two separate processes. The encoder learns relations in the given data and encodes them in the so-called *latent variables*, \mathbf{z} . Given these latent variables, the decoder, in turn, learns to produce data that are *likely* to match the input. Once trained, the user can omit the encoder and directly generate new data from sampling the latent space. For details, we refer to [37, 38], and for applications, we refer to [24] and [39].

The difference between the spline-based approach and generative models is the choice of latent variables. When the human designer creates a spline parameterization that allows modifying geometry in the desired way, the optimization variables are the control points, which are *intuitively* placed in \mathbb{R}^3 by the designer. Generative models, in contrast, *learn* a latent space and explicitly assume that the single latent variables do not have an intuitive interpretation. As a result, data are compressed from a high-dimensional intuitive design space, in our case, $\chi \subset \mathbb{R}^{3 \times n}$, onto a hardly interpretable, feature-dense, low-dimensional latent space \mathbf{Z} . In short, generative models use the computational power of neural networks to find a dense classification space that one can sample to produce new data. For the VAE, this process is depicted in Fig. 4a.

Fig. 4 a Autoencoder providing an input-to-output mapping while passing data through a bottleneck, i.e., the low-dimensional latent representation. **b** Generative adversarial model learning latent space by inferring representations that enable generating output indistinguishable from the input. Two main concepts of deep generative networks: variational autoencoders and generative adversarial networks



A competing concept to VAEs are *Generative Adversarial Networks (GANs)*. Their basic structure is shown in Fig. 4b. GANs, first introduced by Goodfellow et al. [40], follow a different concept and train two adversarial nets, the *generator* and the *discriminator*. In GANs, the generator is trained to create data that mimics real-world data, while the discriminator tries to determine whether or not a dataset was artificially created. In a minimax fashion, the generator's learning goal is to maximize the probability of the discriminator making a wrong decision.

GANs have proven to be an excellent tool for shape modeling. Wu et al., for example, apply a GAN for 3D shape generation and demonstrate their superior performance compared to three-dimensional VAEs. They even use a GAN to reconstruct three-dimensional models from two-dimensional images based on the a VAE output that is used to infer a latent representation for these images [41]. As in [24], Wu et al. also demonstrate the ability to apply shape interpolation and shape arithmetic to the learned latent representation. More recently, Ramasinghe et al. [28] utilize a GAN to model high-resolution three-dimensional shapes using point clouds.

3.2 Implicit shape representation

The neural network learns a mapping between the low-dimensional latent space and a three-dimensional body. To construct such a mapping, we first need to define how to represent our shapes (i.e., define what data the neural network

actually has to learn). Before presenting the approach chosen in this work, we review standard methods and their limitation.

Three ways of shape representation are common in machine learning: (1) voxels, (2) point clouds, and (3) meshes [33]. The problem with meshes is that the mesh topology also prescribes the possible shape topologies. Point clouds, in contrast, can represent arbitrary topologies, but prescribe a given resolution. Finally, voxels can represent arbitrary topologies and vary in resolution, but, unfortunately, the memory consumption scales cubically with the resolution. Because of these drawbacks, the network utilized in this work learns SDFs following a network configuration originally proposed by Park et al. [33].

SDFs provide the distance to the closest point on the to-be-encoded surface for every point in space. Furthermore, encoded in the sign, information on whether the point lies inside or outside the surface is available. Using such continuous SDF data, a shape is then extracted—at an arbitrary resolution suitable for meshing—as its zero-valued isosurface.

3.3 Training set generation

As mentioned in Sect. 3.1, training a neural network requires a set of source shapes. However, to the authors' knowledge, no shape library exists for mixing elements in single screw extruders. Thus, we explain an approach to building custom training sets.

To generate a suitable training set, we first select categories of basis shapes that should be considered—pin and pineapple mixers in our case. From this choice, we arbitrarily infer a total of four basis shapes (i.e., triangle, square, hexagon, and cylinder—cf. Figure 5). At the same time, we define a set of deformations, which should be considered within the design space. Examples of applied deformations are given in Fig. 5.

We start by creating basis shapes represented as triangular meshes. For each basis shape, we apply the aforementioned deformations and their combinations in varying magnitudes using FFD to gather a rich set of shapes. To obtain SDF-training data from these shapes, we follow the approach by Park et al. [33]: first normalize each shape to fit into a unit sphere, and then sample 500,000 pairs of spatial coordinates and their corresponding SDF values using the trimesh library [42] from each shape. In total, 2659 training shapes are generated, which constitute the accessible deformations within the design space.

3.4 Shape generator

As explained, the shape generator’s task is to provide a mixing element given a set of optimization variables. The shape generator—in this work—is thus built around the neural network, which is presented in the following.

The utilized neural network is based on DeepSDF auto-decoder [33]: a feed-forward network with ten fully connected layers, with each of the eight hidden (i.e., internal) layers having 256 neurons and ReLU activation functions. In contrast to autoencoders, the auto-decoder only trains the decoder using a simultaneous optimization of the network parameters and the latent code during training. We investigate 4, 8, and 16 as latent dimensions, l . The input layer consists of these l neurons concatenated with a three-dimensional query location. The output layer has only one neuron with a tanh activation function. For details on the chosen SDF network, we, again, refer to [33]. To train the network, we use the ADAM optimization algorithm [43]. To utilize improved learning rates, we follow a progressive approach

with the initial rates $\epsilon_0 = 5e - 4$ for θ , and $\epsilon_0 = 1e - 3$ for z , and a decay as

$$\epsilon = \epsilon_0 \cdot (0.5^{e\%500}), \tag{3}$$

where e denotes the current training iteration (i.e., *epochs*)—and $\%$ denotes integer division. The network’s training can be seen as the parametrization of the shapes.

To extract isosurfaces (i.e., to generate new mixing elements) from the trained network’s SDF output, we sample a discrete SDF field and apply a marching cube algorithm [44] in the implementation of [45]. Finally, we apply automated meshing using TetGen [46] to obtain a simulation domain as depicted in Fig. 6, including the new mixing element.

4 The developed shape-optimization framework

In general, our framework consists of three building blocks: (1) *shape generator*, (2) *flow solver*, and (3) *optimizer*, which will be described in the following.

Starting with an initial set of optimization variables, σ_0 , the shape generator creates a new mixing element $\Omega(\sigma_0)$. The flow solver then computes the flow field around this

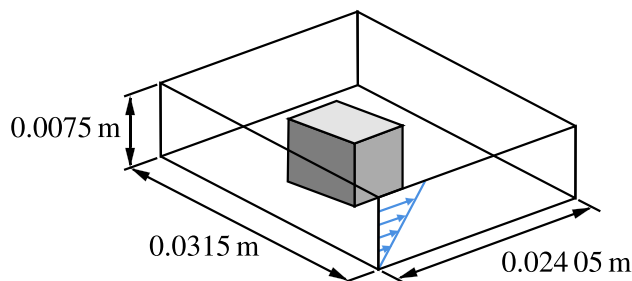
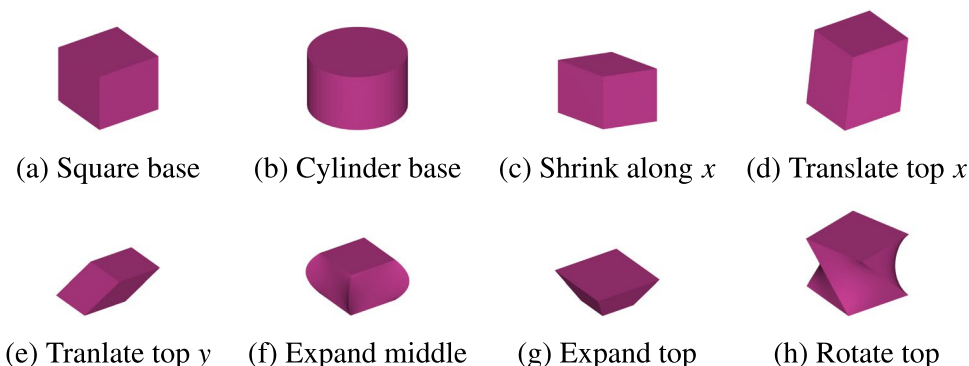


Fig. 6 Simulation domain with single mixing element resembling the flow around a single mixing element in the unwound screw channel. Flow conditions are shown in blue using a barrel rotation setup. For a detailed description of the objective function and governing equations, we refer the reader to [16]

Fig. 5 a Square base, b cylinder base, c shrink along x, d translate top x, e translate top y, f expand middle, g expand top, and h rotate top. Examples of basic shapes and applied deformations. In total, a triangle, a square, a cylinder, and a hexagon are used as basis shapes



mixing element, which the optimizer evaluates to determine the flow's degree of mixing. Based on the obtained mixing value and by comparison to previous iterations, an optimization algorithm determines a new set of optimization variables. This sequence is iteratively re-run until either a maximum number of iterations is reached or any other termination criterion—typically a good objective value or insignificant objective decrease—is met.

4.1 Flow solver and simulation model

The flow solver and simulation model is identical to the one introduced in [16] and therefore only summarized in the following. The flow field induced by the various mixing elements is obtained from solving the steady, incompressible non-isothermal Navier–Stokes equations using a Carreau model and WLF temperature correction. The governing equations are discretized with linear stabilized finite elements and solved using a Newton linearization and a GMRES iterative solver. Subsequently, we solve a set of advection equations using the identical configuration to mimic particle tracking, which we use as an input to our objective function. All methods are implemented in an in-house flow solver.

We make two simplifications to our simulation model (i.e., the single-screw-extruder flow channel): first, we simulate the flow around only a single mixing element instead of simulating the entire mixing section. Second, we assume barrel rotation in an unwound flow channel section. Both assumptions yield significantly reduced computational costs while allowing a qualitative mixing improvement. To assess mixing, we mimic particle tracking by solving a series of advection equations yielding an inflow–outflow mapping for particles advected by the melt flow. We process this advection information by subdividing a portion of the inflow domain into smaller rectangular subdomains. In each of these rectangles, we select a set of particles, such that the particle set's bounding box coincides with the rectangular subdomain. Then, we follow each particle as they are conveyed through the domain, store each particle's position at the outflow domain, and finally construct a convex hull at the outflow around the same sets of points. Averaging the convex hull's length increments between inflow and outflow yields a simple yet robust objective function inspired by interfacial area measurements. Using this objective function, we found that such a simulation model provides a good balance between accuracy and computational efficiency [16]. Figure 6 depicts the chosen simulation domain.

4.2 Optimizer

We utilize the open-source optimization library Dakota [47] to drive the design process. Two different algorithms are

selected and described in the following. The first algorithm is the Dividing RECTangle (DIRECT) algorithm, first introduced in [48]. DIRECT belongs to the category of *branch-and-bound* methods and uses n -dimensional trisection to iteratively partition the design space. To find minima, it follows the approach of Lipschitzian optimization, which identifies the design space partition that should be further sampled by evaluating a lower bound to the objective value in each partition. The partition with the lowest lower bound is chosen and further sampled. DIRECT modifies that concept and computes multiple lower bounds that weight the current sampling value (i.e., the objective value in the partition center). This promotes to further sample partitions with good objective values against the partition size, which permits to effectively sample large areas of unexplored design space. Thereby, DIRECT identifies *multiple* partitions that are *possibly optimal* and allows for global convergence.

The second algorithm utilized in this work is the single-objective genetic algorithm (SOGA) introduced (as its multi-objective variant) in the JEGA package [49]. As it belongs to the class of *genetic* algorithms, it solves optimization problems by recreating biological evolution. Therefore, each optimization run consists of numerous samples referred to as the *population*. Members of the population are paired and recombined in such ways that the *fitness* (i.e., the objective value) is successively improved. Regarding its application in this work, it is especially noteworthy that the recreation of evolution includes a *mutation* step, which modifies or re-initializes design variables randomly. The added randomness allows the algorithm to escape locally convex regions of the design space. Such evolutionary optimization approaches generally converge slower yielding higher computational costs. However, they are often able to find better results than non-evolutionary algorithms. For both DIRECT and SOGA, we rely on the default convergence criterion and a maximum of 1000 iterations as a termination criterion. Additionally, for SOGA, we choose a population size of ten times the number of design dimensions. The complete computational framework is depicted in Fig. 7.

5 Numerical results

This section presents the results obtained using shape parameterizations from neural networks.

Thereby, Sect. 5.1 focuses on the results of the offline phase, i.e., the training of the shape-representing neural network. In particular, we will discuss the differences in the constructed latent space based on its dimension using the widely used data reduction technique *t-Distributed Stochastic Neighbor Embedding (t-SNE)* to visualize the learned, n -dimensional shape parameterization. In Sect. 5.2, we then

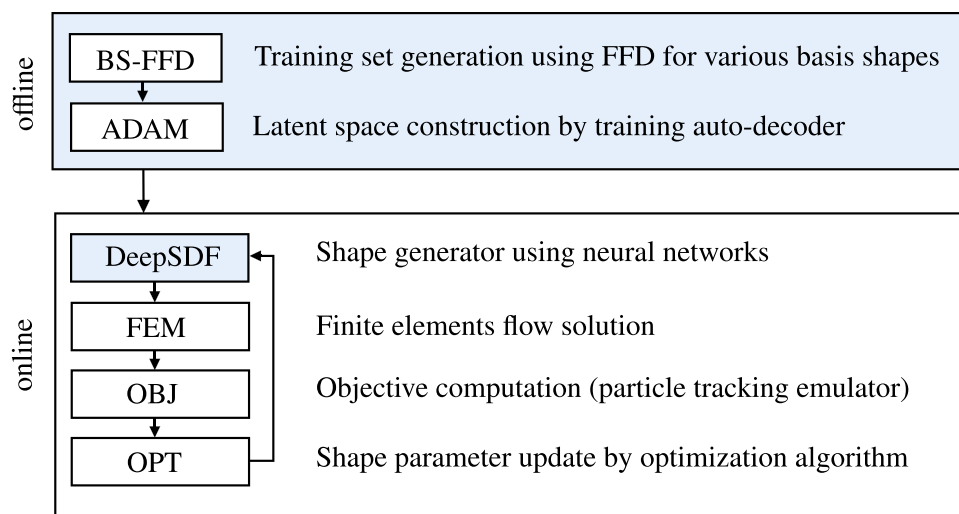


Fig. 7 Pipeline with building blocks of the proposed computational framework. The process is split into two parts: a one-time computationally intensive training part and the actual optimization, including the quick filter evaluation. To create a training set, FFD is applied to a set of basis shapes. Subsequently, we train the network using the ADAM optimizer, which concludes the *offline* phase. During optimization (i.e., the *online* phase), first, a new shape is created from the

neural net. Then, a new computational mesh is created around this shape, and based on FEM simulations, the new design's mixing is assessed. Depending on the objective value, the optimization loop is re-initiated using altered latent variables. Building blocks that are modified compared to the general, geometry-kernel-based approach (cf. Figure 1) are highlighted in blue

present the mixing shapes that could be obtained using our shape-optimization approach.

5.1 Latent space dimension

One of the most important choices is the target dimension of the embedding space l . In all established filtering mechanisms like radial basis functions, free-form deformation, CAD-based approaches, and even mesh-based methods, the practitioner has to balance improved flexibility against the computational demand. Despite a potentially more compact and dense embedding with neural networks, this is still of relevance and manifests itself in the dimension of the chosen latent space. Previous works utilized only a very small number of optimization variables. Elgeti et al. vary between only one and two parameters [12]. Other works by the authors, however, showed that also for six design variables good results are obtained [16]. To obtain a competitively small number of optimization variables, we investigate embedding spaces of dimension 4, 8, and 16, respectively, and compare against a free-form-deformation approach using nine variables.

Even though the latent space, as discussed in Sect. 3.1, in general, obtained lacks an intuitive interpretation, we are still interested in evaluating the quality of the learned embedding space. We do so in three different ways which we present in the following: (1) we show a data reduction technique that allows us to visually investigate the latent space; (2) we apply an interpolation between the latent

representation of two training shapes and compare with the expected result; (3) we apply shape arithmetics, i.e., we isolate a specific modification of a basis shape and impose it onto another basis shape to inspect whether or not features are also recognized by the latent space.

(1) For the visualization of the high-dimensional latent space, a dimension reduction technique is required. An intuitive choice might be principal component analysis (PCA), but PCA tries to primarily preserve global structures and thus data points which are far apart in the high-dimensional data will also be drawn far apart in the 2D plot. Conversely, the correlation between similar points is often lost. This loss of correlation in similar data is problematic, since we aim to investigate whether—from a human's perspective—similar shapes are represented by similar latent code. The problem of loss in local correlation is, however, alleviated by t-SNE [50]. Using t-SNE, we plot each training shape's obtained latent code and—due to the preservation of local similarities—similar latent code will form clusters in the scatter plot. These clusters can then be sampled to verify that the latent code clusters resemble similar shapes. t-SNE plots for all three latent dimensions—4, 8, and 16—are shown in Fig. 8.

Figure 8 shows how an increased latent dimension leads to increased classification performance of the neural net. Specifically, the four chosen basis shapes are clustered with their respective modifications more and more densely as the latent dimension increases. This improved classification performance indicates that the neural net was able to learn the

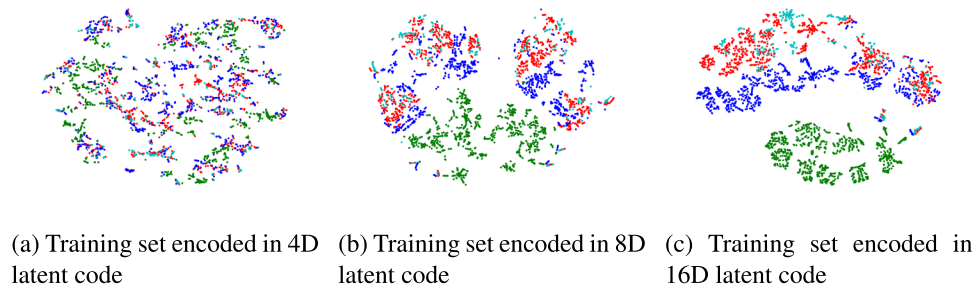


Fig. 8 **a** Training set encoded in 4D latent code. **b** Training set encoded in 8D latent code. **c** Training set encoded in 16D latent code. t-SNE plots obtained using different latent space dimensions. Increased latent dimension resembles in increased classification per-

formance of the neural net. Each color corresponds to one base training shape: green corresponds a triangular base, dark blue is the cube, red is the hexahedron, and light blue is a tessellated version of the cylinder (colour figure online)

similarities between similar shapes properly for the case of 8 and 16 dimensions.

(2) In addition to comparing clusters of similar shapes in physical and latent space, we also investigate how well the latent space is suited to represent shapes that have not been included in the training set. We do so by *interpolation* between two shapes. Figure 9 shows the obtained results for all three latent spaces.

Consistent to the observed lack in classification ability of the four-dimensional latent space, Fig. 9a shows that interpolation between shapes yields unsatisfactory results. In particular, shape defects are observed. This might be a result of the fact that the twisted cube is not at all well represented in the latent space as seen in the rightmost figure. However, both the 8- and the 16-dimensional latent space show a visually smooth transition between the regular and the twisted cube shape.

(3) The above two analyses investigated the overall classification ability of the neural net and the suitability to represent intermediate shapes. A final test is given by applying *shape arithmetic*. Using arithmetic operations applied to the latent code, we extract an exemplary feature—here a stretching along the center plane—by taking the component-wise difference of a stretched and a regular cube. This difference represents center-plane expansion and can then be applied to any other basis shape—here the undeformed hexahedron. Figure 10 shows the resulting shapes. Again, the four-dimensional latent space performs significantly worse, since the basis shapes are not represented in detail. Contrary to the interpolation case, the 16-dimensional latent space now shows better results than the 8-dimensional case.

All three investigations, t-SNE plots, interpolation, and arithmetic, indicate that the four-dimensional latent space fails in producing a suitable latent representation. It should

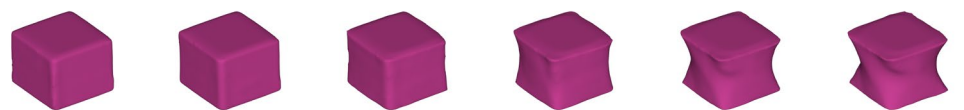
Fig. 9 **a** 4D shape interpolation revealing artifacts in the reconstructed shapes, i.e., bad quality of the latent representation. **b** 8D shape interpolation with satisfactory results. **c** 16D shape interpolation, which brings only slight improvement in shape representation compared to eight-dimensional latent representation. Shape interpolation using different latent dimensions. An interpolated shape is obtained using $z_{\text{interp}} = z_a + \frac{z_b - z_a}{N+1}n$ with z_a and z_b denoting the latent code between shapes a —here the undeformed cube—and b —here the twisted cube. With $N = 20$, the shown examples represent $n \in [1, 3, 7, 13, 17, 20]$



(a) 4D shape interpolation revealing artifacts in the reconstructed shapes, i.e. bad quality of the latent representation.



(b) 8D shape interpolation with satisfactory results.



(c) 16D shape interpolation, which brings only slight improvement in shape representation compared to eight-dimensional latent representation.

Fig. 10 **a** 4D shape arithmetics with significant representation errors, especially for the hexahedron and the final shape. **b** 8D shape arithmetic with improved representation compared to 4D latent code but still yielding slightly imprecise results. **c** 16D shape arithmetic showing perfect resemblance of all training shape and also a clean resulting shape. Shape arithmetics for different latent dimensions. A linear thickening in the center plane is imposed on a hexagonal base body by evaluation of the latent code as $z_{E4_{thick}} - z_{E4} + z_{E6}$, where $z_{E4_{thick}}$, z_{E4} , and z_{E6} denote the latent codes of the thickened cube, the regular cube, and the regular hexahedron, respectively



(a) 4D shape arithmetics with significant representation errors, especially for the hexahedron and the final shape.



(b) 8D shape arithmetic with improved representation compared to 4D latent code but still yielding slightly imprecise results.



(c) 16D shape arithmetic showing perfect resemblance of all training shape and also a clean resulting shape.

Table 1 Different optimization algorithms and latent space dimensions compared by best objective value and contrasted to a nine-dimensional FFD **1b**

	4	8	16	(FFD)
SOGA	-0.0726	-0.0710	-0.0750	-
DIRECT	-0.0645	-0.0738	-0.0769	-0.0422

Smaller values correspond to better results using the aforementioned objective formulation

be noted though that in view of the doubled number of optimization variables, the attainable gains in using 16 latent variables compared to 8 appear unattractively small.

5.2 Optimization results

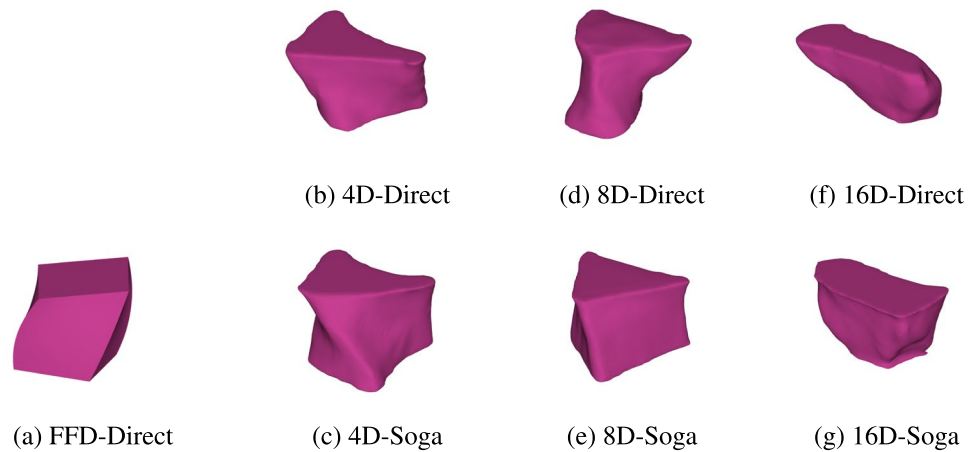
To study the effects of the novel shape-parameterization technique, we compare configurations that vary in latent space dimensions and optimization algorithms, as shown in Table 1. I furthermore, we require all generated shapes to have the exact same volume as the undeformed rhombic mixing element utilized in the spline-based optimization (cf. Sect. 4.1). We choose such scaling to avoid convergence towards merely enlarged shapes that yield good objective values but do not deliver helpful insights. Table 1 lists the obtained results, and Table 2 gives insights into the corresponding computational effort.

The obtained best shapes are shown in Fig. 11.

Table 2 Different optimization algorithms and latent space dimensions compared by the final iteration count, the obtained objective value, and the number of total iterations; contrasted to a nine-dimensional (FFD)

# Iteration(s)	4		8		16		(FFD)	
	Optimal	Total	Optimal	Total	Optimal	Total	Optimal	Total
SOGA	768	1000	752	1000	534	1000	-	-
DIRECT	96	113	129	143	138	149	16	67

Fig. 11 **a** FFD-Direct, **b** 4D-Direct, **c** 4D-Soga, **d** 8D-Direct, **e** 8D-Soga, **f** 16D-Direct, and **g** 16D-Soga. Optimization results obtained for all different latent codes and optimization algorithms compared to an existing (FFD)-based shape optimization



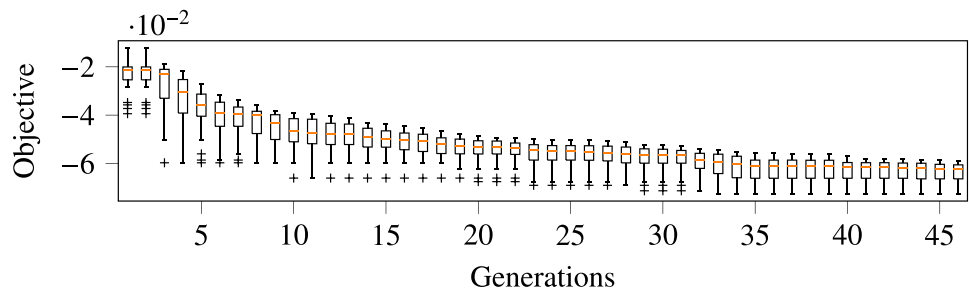
Comparing the optimized geometries shows interesting results from a plastics processing point of view. On the one hand, the triangular shape and a mixing element that widens towards the top appear advantageous. One should note, however, that these deformations do not correspond to a general optimum for plastics engineering but are merely the best possible deformations within the range permitted by the training set. Choosing an even more diverse training set is expected to yield even further improved shapes.

More relevant for this study (with a focus on neural nets as shape parameterizations) is the comparison of convergence, the achieved mixing, and the difference and similarities in the results. Table 1 shows that for the chosen shape optimization problem, the DIRECT algorithm has no disadvantages compared to SOGA and converges reliably.

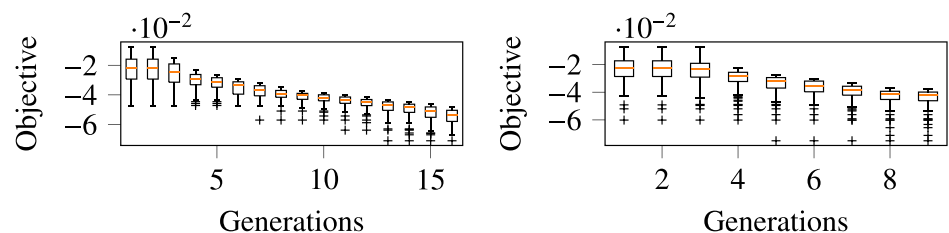
Simultaneously, the shape parameterization's dimensionality appears to influence the optimization because the four- and eight-dimensional neural networks lead to optimized triangles. In contrast, the 16-dimensional case renders the top-expanded quadrilateral optimal. Common to all results is a skewed and slightly twisted geometry.

The objective value per generation of the SOGA algorithm is shown in Fig. 12. For all dimensions, each generation tends to evolve towards better objective values. We investigate if improvements in objective values can be related to certain trends in terms of geometric features. Within the latent space, features are expressed as multi-dimensional vectors, as opposed to single-dimensional values; thus, latent code arithmetics [24, 41], for example, are often used to explore learned features (cf. Figure 10). Furthermore, the

Fig. 12 **a** 4D latent code with 40 populations per generation. **b** 8D latent code with 80 populations per generation. **c** 16D latent code with 160 populations per generation. Comparison of objective value per population among different latent dimensions from SOGA algorithm



(a) 4D latent code with 40 populations per generation



(b) 8D latent code with 80 populations per generation (c) 16D latent code with 160 populations per generation

latent code alone does not provide information about the associated geometric features. To determine the corresponding features, we begin by searching for the nearest neighbor in the training set based on Euclidean distance. From the nearest neighbor, we can then quantify applied features, i.e., deformations. The mean values of the approximated features are shown in Fig. 13. It shows that diverse features, even with less significance, were visited at the beginning of the iterations. Across all generations, relatively high magnitudes of “rotate”, “rotate top”, and “shrink along y”, as well as a low magnitude of “expand top”, consistently appear in all the results. The four-dimensional latent space also includes consistent application of “shrink top”. Interestingly, all of the consistently applied features are reflected in Fig. 11.

The behavior of objective values for DIRECT is not separately investigated, as the objective values and geometric features can highly fluctuate during partitioned sampling of the latent space. Consequently, DIRECT’s iteration history does not yield significant insight regarding the deformation trend.

A noticeable difference between the spline-based and neural-net-based shape optimization is that the neural-net-based

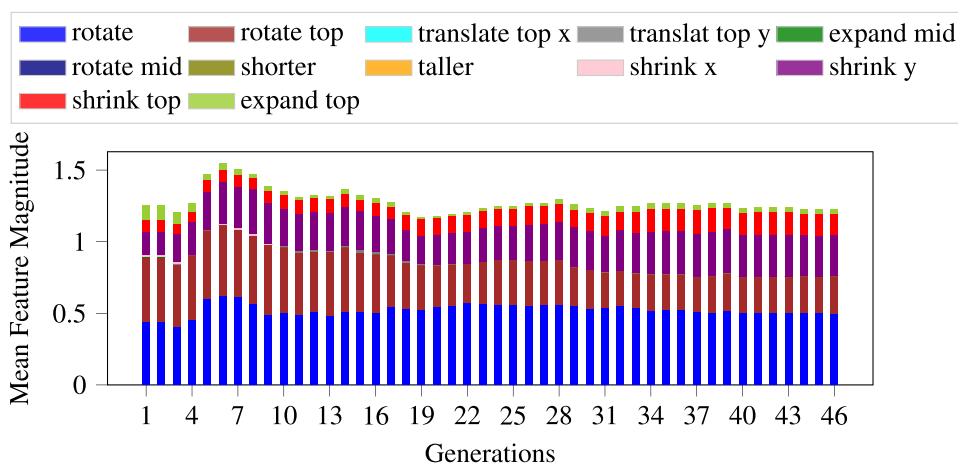
shape parameterization encodes several shapes, of which multiple may mix the melt equally well. Because of this, from the practitioner’s point of view, it does make sense to not only look at the best result but rather compare numerous equally optimal designs and derive design rules from that comparison. Figure 14 shows such a comparison and reveals one advantage of evolutionary algorithms.

While the DIRECT algorithm converges locally and, therefore, the ten best designs are geometrically similar, the generative nature of SOGA allows the practitioner to identify possibly equally well-working designs (cf. Figures 14f and g) amongst which the most economical option may be chosen. Such a choice allows one to account for further restrictions regarding screw cleaning, manufacturability, and others.

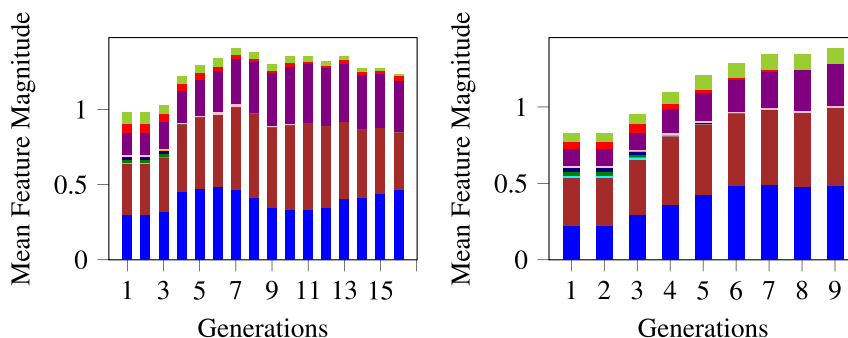
6 Discussion and outlook

In this work, we studied the applicability of generative models as shape parameterizations. We chose numerical shape optimization of dynamic mixing elements as a use

Fig. 13 a 4D latent code with 40 populations per generation. b 8D latent code with 80 populations per generation. c 16D latent code with 160 populations per generation. Comparison of mean value of approximated applied deformation per population among different latent dimensions from SOGA algorithm

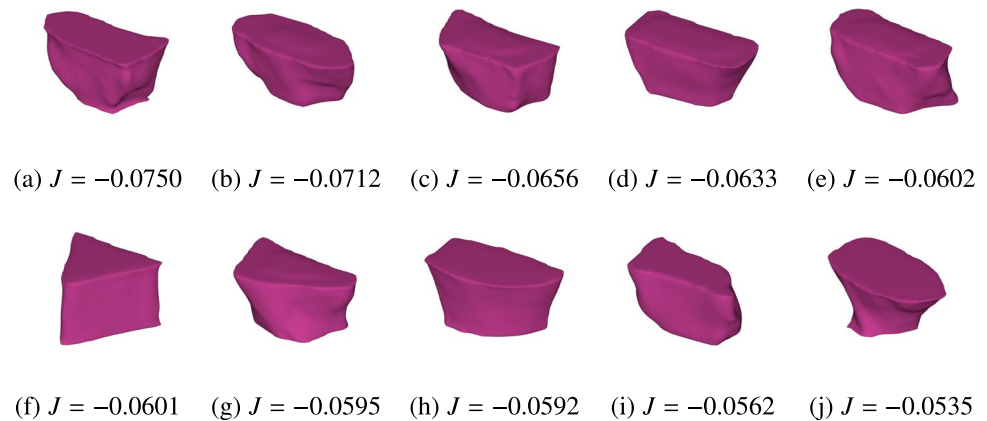


(a) 4D latent code with 40 populations per generation



(b) 8D latent code with 80 populations per generation (c) 16D latent code with 160 populations per generation

Fig. 14 **a** $J = -0.0750$; **b** $J = -0.0712$; **c** $J = -0.0656$; **d** $J = -0.0633$; **e** $J = -0.0602$; **f** $J = -0.0601$; **g** $J = -0.0595$; **h** $J = -0.0592$; **i** $J = -0.0562$; **j** $J = -0.0535$. Ten best shapes obtained from 16D SOGA optimization. Except for the sixth-best shape (h), all shapes feature an expanded top, similar orientation, and appear widened in y direction (i.e., perpendicular to the main flow direction)



case. The developed shape parameterization's fundamental principle is to exploit neural nets' ability to construct a dimension reduction onto a feature-dense, low-dimensional latent space. It should be noted that other interpolation methods may be able to construct similar interpolation spaces. However, this work demonstrates neural networks' exceptional generalization power yields excellent shape parameterization, which particularly allows interpolation between shapes of varying topologies.

First, the nature of this low-dimensional space is studied by t-SNE-plots. These plots give visual evidence that the generative models create *smooth* shape parameterizations that enable one to use classical, heuristic optimization algorithms. Comparing genetic to such heuristic algorithms, Table 2 reveals that the SOGA algorithm required significantly more iterations (i.e., simulations). Additionally, Table 1 shows that in the studied examples, this additional computational effort is not reflected proportionally by improved mixing. One may expect that the SOGA algorithm's random nature may be better suited to explore the hardly interpretable latent space. However, the results suggest a smoothness of the learned parameterization that renders deterministic methods like DIRECT equally well suited for optimization in the latent space.

In addition, to the general applicability of generative models, we study the influence of different latent dimensions. While the actual optimization results appear pleasing, Figs. 9 and 10 suggest that very compressed (i.e., four-dimensional) latent spaces may not be used for optimization purposes. Analogously, no direct preference between the 8- and 16-dimensional results can be drawn from the optimization results. Similarly, Fig. 10 indicates that higher dimensional latent spaces yield more precise shape encoding, which seems generally preferable. Since the overall number of iterations until convergence of the optimization problem is comparable, the 16-dimensional parameterization might be chosen over the 8-dimensional variant.

As intended, a fundamental improvement over established low-dimensional shape parameterizations is that the new approach covers a much broader design area in a single optimization. Since its fundamental concept is to encode diverse shapes, optimizations lead to numerous, nearly equally optimal shapes. Consequently, this novel approach extends on the existing methods in that it allows the practitioner to *derive* design features that enhance mixing most and for a wide range of basis shapes. Therefore, rather than creating complex shape parameterizations, the crucial step towards optimal design reduces to the creative definition of a training set.

A significant challenge in using neural-net-based shape parameterization is proper control of the output shapes' size. This work implements a volume constraint to avoid simple size maximization of the mixing elements. However, a reformulated objective, such as penalizing pressure loss, may circumvent such adverse designs. Alternatively, a scale factor may be added as an additional optimization variable. Both size control and efficient training set generation may be topics of further studies.

Given the presented results, utilizing the feature-rich latent representations and their immense generalization power has a significant potential to improve established industrial designs.

Finally, the presented framework can be extended by adopting neural-network-based simulators, as recent works [51–55] have shown promising results. Using neural-network-based simulators, the evaluations of objective functions during the online phase become mere forward pass(es) of neural nets that replace costly numerical simulations. Combining the presented parameterization and aforementioned simulators, one could also benefit from the differentiability of neural networks to acquire gradients, which naturally opens the door to other gradient-based optimization algorithms. It is worth noting that if the objective function is formulated as part of the network, automatic differentiation can be leveraged to its full potential, allowing for the

efficient computation of derivative values with respect to design parameters. Both neural-network-based simulators and development of suitable objective functions will be investigated in the future work.

Acknowledgements The German Research Foundation (DFG) fundings under the DFG grant “Automated design and optimization of dynamic mixing and shear elements for single-screw extruders” and priority program 2231 “Efficient cooling, lubrication and transportation—coupled mechanical and fluid-dynamical simulation methods for efficient production processes (FLUSIMPRO)”—under Project No. 439919057 are gratefully acknowledged. Implementation was done on the HPC cluster provided by IT Center at RWTH Aachen. Simulations were performed with computing resources granted by RWTH Aachen University under Project Nos. jara0185 and thes0735.

Funding Open access funding provided by TU Wien (TUW).

Data availability Not applicable.

Declarations

Conflict of interest The authors have no conflict of interest related to this manuscript.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Celik O, Erb T, Bonten C (2017) Mischgüte in Einschneckenextrudern vorhersagen. *Kunststoffe* 71:175–177
- Gale M (2009) Mixing in single screw extrusion. Handbook series. Smithers Rapra Publishing, Shrewsbury
- Roland W, Marschik C, Miethlinger J, Chung C (2019) Mixing study on different pineapple mixer designs—simulation results 1. In: SPE-ANTEC Technical Papers
- Sun X, Spalding MA, Womer TW, Uzelac N (2017) Design optimization of maddock mixers for single-screw extrusion using numerical simulation. In: Proceedings of the 75th Annual Technical Conference of the Society of Plastic Engineers (ANTEC)
- Campbell GA, Spalding MA (2021) Analyzing and troubleshooting single-screw extruders. In: Analyzing and troubleshooting single-screw extruders (second edition), Second edition edn., pp 1–19. Hanser, Munich. https://doi.org/10.3139/9781569907856_fm
- Marschik C, Osswald T, Roland W, Albrecht H, Skrabala O, Miethlinger J (2018) Numerical analysis of mixing in block-head mixing screws. *Polym Eng Sci*. <https://doi.org/10.1002/pen.24968>
- Böhme G (1981) Strömungsmechanik Nicht-newtonscher Fluide, 1st ed. 1981 edn. Leitfäden der angewandten Mathematik und Mechanik - Teubner Studienbücher. Vieweg+Teubner Verlag, Wiesbaden
- Kim SJ, Kwon TH (1996) Enhancement of mixing performance of single-screw extrusion processes via chaotic flows: Part i. basic concepts and experimental study. *Adv Polym Technol* 15(1):41–54. [https://doi.org/10.1002/\(SICI\)1098-2329\(199621\)15:1<41::AID-ADV4>3.0.CO;2-K](https://doi.org/10.1002/(SICI)1098-2329(199621)15:1<41::AID-ADV4>3.0.CO;2-K)
- Wong AC-Y, Lam Y, Wong ACM (2009) Quantification of dynamic mixing performance of single screws of different configurations by visualization and image analysis. *Adv Polym Technol* 28(1):1–15. <https://doi.org/10.1002/adv.20142>
- Kim SJ, Kwon TH (1996) Enhancement of mixing performance of single-screw extrusion processes via chaotic flows: part ii. numerical study. *Adv Polym Technol* 15(1):55–69. [https://doi.org/10.1002/\(SICI\)1098-2329\(199621\)15:1<55::AID-ADV5>3.0.CO;2-J](https://doi.org/10.1002/(SICI)1098-2329(199621)15:1<55::AID-ADV5>3.0.CO;2-J)
- Domingues N, Gaspar-Cunha A, Covas J (2012) A quantitative approach to assess the mixing ability of single-screw extruders for polymer extrusion. *J Polym Eng*. <https://doi.org/10.1515/polyeng-2012-0501>
- Elgeti S, Probst M, Windeck C, Behr M, Michaeli W, Hopmann C (2012) Numerical shape optimization as an approach to extrusion die design. *Finite Elem Anal Des* 61:35–43. <https://doi.org/10.1016/j.finel.2012.06.008>
- Siegbert R, Behr M, Elgeti S (2016) Die swell as an objective in the design of polymer extrusion dies. *AIP Conf Proc* 1769(1):140003. <https://doi.org/10.1063/1.4963540>
- Gaspar-Cunha A, Covas J (2001) The design of extrusion screws: an optimization approach. *Int Polym Process*. <https://doi.org/10.3139/217.1652>
- Potente H, Többen WH (2002) Improved design of shearing sections with new calculation models based on 3d finite-element simulations. *Macromol Mater Eng* 287(11):808–814. <https://doi.org/10.1002/mame.200290010>
- Hube S, Behr M, Elgeti S, Schön M, Sasse J, Hopmann C (2022) Numerical design of distributive mixing elements. *Finite Elem Anal Des* 204:103733. <https://doi.org/10.1016/j.finel.2022.103733>
- Botsch M, Kobbelt L (2004) An intuitive framework for real-time freeform modeling. *ACM Trans Graph* 23(3):630–634. <https://doi.org/10.1145/1015706.1015772>
- Farin G (2002) 5 - the bernstein form of a bézier curve. In: Farin G (ed) Curves and Surfaces for CAGD (Fifth Edition), Fifth edition edn. The Morgan Kaufmann Series in Computer Graphics, pp 57–79. Morgan Kaufmann, San Francisco. <https://doi.org/10.1016/B978-155860737-8/50005-3>
- Piegl L, Tiller W (1996) The NURBS book. Monographs in visual communication. Springer, Berlin
- Sederberg TW, Parry SR (1986) Free-form deformation of solid geometric models. *SIGGRAPH Comput Graph* 20(4):151–160. <https://doi.org/10.1145/15886.15903>
- Hojjat M, Stavropoulou E, Bletzinger K-U (2014) The vertex morphing method for node-based shape optimization. *Comput Methods Appl Mech Eng* 268:494–513. <https://doi.org/10.1016/j.cma.2013.10.015>
- Johnson J, Alahi A, Li F (2016) Perceptual losses for real-time style transfer and super-resolution. *CoRR* arXiv:1603.08155
- Friedrich T, Aulig N, Menzel S (2018) On the potential and challenges of neural style transfer for three-dimensional shape data. In: *EngOpt* 2018. Springer
- Liu S, Giles CL, Ororbia A (2018) Learning a hierarchical latent-variable model of 3d shapes. In: 2018 International Conference on 3D Vision, 3DV 2018, Verona, Italy, September 5–8, 2018, pp 542–551. IEEE Computer Society, your home. <https://doi.org/10.1109/3DV.2018.00068>

25. Wu J, Zhang C, Xue T, Freeman WT, Tenenbaum JB (2016) Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In: Proceedings of the 30th International Conference on Neural Information Processing Systems. NIPS'16, pp 82–90. Curran Associates Inc., Red Hook, NY, USA
26. Wu Z, Song S, Khosla A, Yu F, Zhang L, Tang X, Xiao J (2015) 3d shapenets: a deep representation for volumetric shapes. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 1912–1920. <https://doi.org/10.1109/CVPR.2015.7298801>
27. Charles RQ, Su H, Kaichun M, Guibas LJ (2017) Pointnet: deep learning on point sets for 3d classification and segmentation. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 77–85. <https://doi.org/10.1109/CVPR.2017.16>
28. Ramasinghe S, Khan S, Barnes N, Gould S (2020) Spectral-gans for high-resolution 3d point-cloud generation. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 8169–8176. <https://doi.org/10.1109/IROS45743.2020.9341265>
29. Yang Y, Feng C, Shen Y, Tian D (2018) Foldingnet: point cloud auto-encoder via deep grid deformation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)
30. Bagautdinov T, Wu C, Saragih J, Fua P, Sheikh Y (2018) Modeling facial geometry using compositional vaes. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 3877–3886
31. Tan Q, Gao L, Lai Y-K, Xia S (2018) Variational autoencoders for deforming 3d mesh models, pp 5841–5850. <https://doi.org/10.1109/CVPR.2018.00612>
32. Groueix T, Fisher M, Kim VG, Russell BC, Aubry M (2018) A papier-mache approach to learning 3d surface generation. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 216–224. <https://doi.org/10.1109/CVPR.2018.00030>
33. Park JJ, Florence P, Straub J, Newcombe R, Lovegrove S (2019) DeepSDF: learning continuous signed distance functions for shape representation. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)
34. Chen Z, Zhang H (2019) Learning implicit fields for generative shape modeling. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)
35. Mescheder L, Oechsle M, Niemeyer M, Nowozin S, Geiger A (2019) Occupancy networks: learning 3d reconstruction in function space. In: Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)
36. Lassila T, Rozza G (2010) Parametric free-form shape design with pde models and reduced basis method. *Comput Methods Appl Mech Eng* 199(23):1583–1592. <https://doi.org/10.1016/j.cma.2010.01.007>
37. Doersch C (2016) Tutorial on Variational Autoencoders. arXiv preprint. [arXiv:1606.05908](https://arxiv.org/abs/1606.05908)
38. Kingma DP, Welling M (2013) Auto-Encoding Variational Bayes. arXiv preprint. [arXiv:1312.6114](https://arxiv.org/abs/1312.6114)
39. Tan Q, Gao L, Lai Y, Xia S (2018) Variational autoencoders for deforming 3d mesh models. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 5841–5850. <https://doi.org/10.1109/CVPR.2018.00612>
40. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. In: Ghahramani Z, Welling M, Cortes C, Lawrence N, Weinberger KQ (eds) *Advances in neural information processing systems*, vol 27. Curran Associates Inc, New York, pp 2672–2680
41. Wu J, Zhang C, Xue T, Freeman B, Tenenbaum J (2016) Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In: Lee D, Sugiyama M, Luxburg U, Guyon I, Garnett R (eds) *Advances in neural information processing systems*, vol 29. Curran Associates Inc, New York, pp 82–90
42. Dawson-Haggerty et al.: trimesh. <https://trimsh.org/>
43. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv preprint. [arXiv:1412.6980](https://arxiv.org/abs/1412.6980)
44. Lorensen WE, Cline HE (1987) Marching cubes: a high resolution 3d surface construction algorithm. *SIGGRAPH Comput Graph* 21(4):163–169. <https://doi.org/10.1145/37402.37422>
45. Lewiner T, Lopes H, Vieira AW, Tavares G (2003) Efficient implementation of marching cubes' cases with topological guarantees. *J Graph Tools* 8:2003
46. Si H (2015) Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Trans Math Softw.* <https://doi.org/10.1145/2629697>
47. Adams BM, Eldred MS, Geraci G, Hooper RW, JD, J, Maupin KA, Monschke JA, Rushdi AA, Stephens JA, Swiler LP, Wildey TM (July 2014; updated May 2019) Dakota, A multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: version 6.10 user's manual
48. Jones D, Perttunen C, Stuckman B (1993) Lipschitzian optimisation without the lipschitz constant. *J Optim Theory Appl* 79:157–181. <https://doi.org/10.1007/BF00941892>
49. Eddy J, Lewis K (2001) Effective generation of pareto sets using genetic programming. *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. Volume 2B: 27th Design Automation Conference, pp 783–791. <https://doi.org/10.1115/DETC2001/DAC-21094>
50. Maaten LJP, Hinton GE (2008) Visualizing high-dimensional data using t-sne. *J Mach Learn Res* 9(11):2579–2605
51. Li A, Zhang YJ (2023) Isogeometric analysis-based physics-informed graph neural network for studying traffic jam in neurons. *Comput Methods Appl Mech Eng* 403:115757. <https://doi.org/10.1016/j.cma.2022.115757>
52. Mallik W, Farvolden N, Jelovica J, Jaiman RK (2022) Deep convolutional neural network for shape optimization using level-set approach. arXiv preprint. [arXiv:2201.06210](https://arxiv.org/abs/2201.06210)
53. Remelli E, Lukoianov A, Richter S, Guillard B, Bagautdinov T, Baque P, Fua P (2020) MeshSDF: differentiable iso-surface extraction. In: Larochelle H, Ranzato M, Hadsell R, Balcan MF, Lin H (eds) *Advances in Neural Information Processing Systems*, vol 33. Curran Associates Inc, New York, pp 22468–22478 https://proceedings.neurips.cc/paper_files/paper/2020/file/fe40fb944ee700392ed51bfe84dd4e3d-Paper.pdf
54. Shukla K, Oommen V, Peyvan A, Penwarden M, Bravo L, Ghoshal A, Kirby RM, Karniadakis GE (2023) Deep neural operators can serve as accurate surrogates for shape optimization: a case study for airfoils. arXiv preprint. [arXiv:2302.00807](https://arxiv.org/abs/2302.00807)
55. Woldseth RV, Aage N, Bærentzen JA, Sigmund O (2022) On the use of artificial neural networks in topology optimisation. *Struct Multidiscip Optim* 65(10):294. <https://doi.org/10.1007/s00158-022-03347-1>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.