Engineering Applications of Artificial Intelligence 119 (2023) 105715

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Generative design of physical objects using modular framework

Nikita O. Starodubcev [a,*], Nikolay O. Nikitin [a], Elizaveta A. Andronova [b], Konstantin G. Gavaza [a], Denis O. Sidorenko [a], Anna V. Kalyuzhnaya [a]

[a] ITMO University, Saint-Petersburg, Russia
[b] Peter the Great St. Petersburg Polytechnic University, Saint-Petersburg, Russia

ARTICLE INFO

Dataset link: https://github.com/ITMO-NSS-team/GEFEST, https://github.com/ITMO-NSS-team/GEFEST-paper-experiments

Keywords:
Generative design
Deep learning
Evolutionary algorithms
Optimization problems

ABSTRACT

In recent years generative design techniques have become firmly established in numerous applied fields, especially in engineering. These methods are crucial for automating the initial stages of the engineering design of various structures, which reduces the amount of routine work. However, existing approaches are limited by the specificity of the problem under consideration. In addition, they do not provide the desired flexibility in choosing a method for a particular problem. To avoid these issues, we proposed a general approach to an arbitrary generative design problem and implemented a novel open-source framework called GEFEST (Generative Evolution For Encoded STructure) on its basis. This approach is based on three general principles: sampling, estimation, and optimization. This ensures the freedom of method adjustment for the solution of the particular generative design problem and therefore enables the construction of the most suitable one. A series of experimental studies was conducted to confirm the effectiveness of the GEFEST framework. It involved synthetic and real-world cases (coastal engineering, microfluidics, thermodynamics, and oil field planning). The flexible structure of GEFEST makes it possible to obtain results that surpass baseline and state-of-the-art solutions: 12% improvement in the coastal engineering problem; 9% in microfluidics; 8% in thermodynamics and 7% in oil field planning.

1. Introduction and problem definition

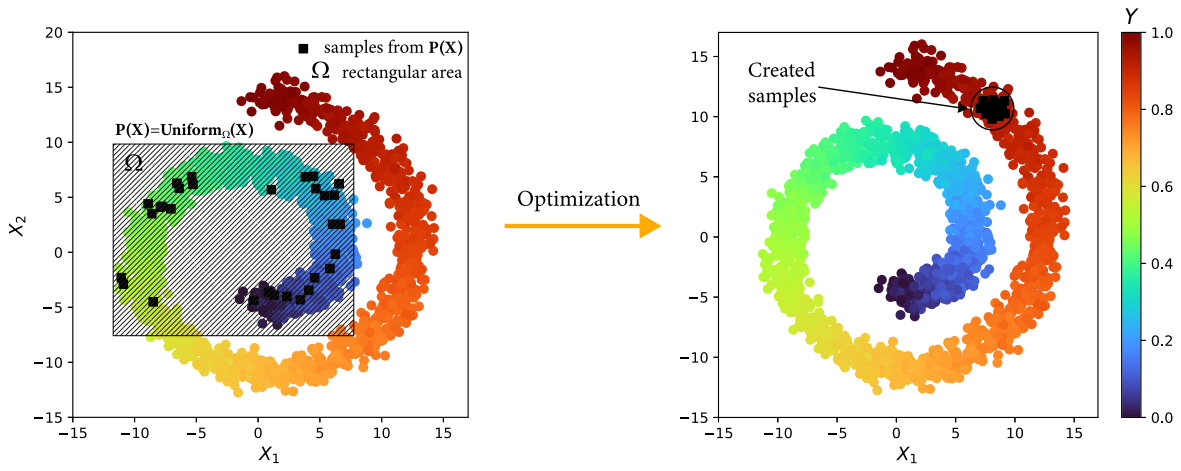
Over the past decades artificial intelligence, machine learning, and optimization methods have become an inevitable part of the solution to engineering design problems (Chen et al., 2020; Steinbuch, 2010; Zheng and Yuan, 2021). These methods demonstrate great potential to improve and simplify tasks usually performed by engineers. The particular complexity of engineering design problems is associated with a large design space originating from a great number of parameters (Danhaive and Mueller, 2021; Harding, 2016). Human efforts are not enough to explore such a high-dimension space. In contrast, computational-based approaches can be examined as efficient tools for given purposes.

The most common computational-based methods for an engineering design problem are generative design (Vajna et al., 2005) and topology optimization (Bendsøe, 1989). In general, the main goal of these methods is the same — to find one or several physical objects whose properties are more preferable than those of the existing ones while taking into account their geometrical and boundary restrictions (Bendsøe and Kikuchi, 1988; Bendsøe, 1989; Vajna et al., 2005). The key difference lies in the way this goal is reached. Topology optimization seeks to enhance performance and reduce the weight of the already existing objects via optimizing material distribution in it (Bendsøe, 1989; Tyflopoulos et al., 2018). Conversely, in the generative design there is no prior knowledge about the initial object, it "generates" structures based on space constraints and design goals only (Vlah et al., 2020). The aforementioned approaches have gained widespread acceptance in various applied fields, for instance, ocean engineering (Tian et al., 2022), mechanical design (Oh et al., 2019), heat and mass transfer (Qian et al., 2022), thermal engineering (Zou et al., 2022). However, well-defined theory fundamentals were established only for topology optimization(Bendsøe and Kikuchi, 1988; Bendsøe, 1989), whereas a strict problem statement in the generative design of physical objects is still missing (Vlah et al., 2020).

In previous works several definitions of generative design were proposed, Shea et al. (2005): "Generative design systems are aimed at creating new design processes that **produce** spatially novel yet efficient and buildable designs through exploitation of current computing and manufacturing capabilities", Kallioras and Lagaros (2020): "Generative Design is the methodology for automatic **creation** of a large number of designs via an iterative algorithmic framework while respecting user-defined criteria and limitations". Nevertheless, the production and creation mechanisms of designs were not explained accurately. Originally, in statistical theory, generative modeling problems have

* Corresponding author.
E-mail addresses: nstarodubtcev@itmo.ru (N.O. Starodubcev), nnikitin@itmo.ru (N.O. Nikitin).

https://doi.org/10.1016/j.engappai.2022.105715
Received 30 August 2022; Received in revised form 3 December 2022; Accepted 4 December 2022
Available online xxxx
0952-1976/



(a) True joint distribution **P(X, Y)** and samples from uniform distribution on the hatched rectangle

(b) True joint distribution **P(X, Y)** and designed samples

Fig. 1. Two-dimensional example $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2)$ for sample production from $\mathbf{P}(\mathbf{X}, \mathbf{Y})$ using a uniform distribution, gradient boosting and genetic algorithm as object distribution, conditional target distribution and optimization method, respectively.

been aimed at the reconstruction of the joint probability distribution $\mathbf{P}(\mathbf{X}, \mathbf{Y})$ on random variables $\mathbf{X}$ (observable variable) and $\mathbf{Y}$ (target variable) (Ng and Jordan, 2001; Jebara, 2012). Generative design is associated with the same problem, but joint distribution is extremely intricate. Special complexity is related to the variables $\mathbf{X}$ and $\mathbf{Y}$ that correspond to a real physical object and its performance, respectively. For instance, heat-generating components of electronic systems and temperature fields (Qian et al., 2022); car wheel and its cost, novelty, compliance (Oh et al., 2019); ship hull form and its strength (Liu et al., 2022). In generative design, objects with the highest performance are of greatest interest, and joint distribution allows one to obtain (produce or create) such physical objects using sampling procedures. Thus, the **production** and **creation** mechanisms of designs lie in sampling from $\mathbf{P}(\mathbf{X}, \mathbf{Y})$, and the generative design problem is focused on the estimation of joint probability distribution on real physical objects and its performance.

However, reconstruction of $\mathbf{P}(\mathbf{X}, \mathbf{Y})$ for real objects is not available as of this day for the following reasons: (1) extremely high dimension of design space; (2) numerous geometrical and boundary restrictions on $\mathbf{X}$ variable; (3) possible continuity of target space which might also be multi-dimensional. Existing approaches enable to obtain only a minority of samples from the joint distribution. All of them include the following stages (we call this a *generative design procedure*):

1. define object distribution $\mathbf{P}(\mathbf{X})$ to sample $\mathbf{X}$;
2. model conditional target distribution $\mathbf{P}(\mathbf{Y}|\mathbf{X})$ to acquire performance of sampled $\mathbf{X}$;
3. solve optimization problem

$$\mathbf{X}^* = \arg\max_{\mathbf{X} \sim \mathbf{P}(\mathbf{X})} \mathbf{Y}(\mathbf{X}) \tag{1}$$

An implementation example of the mentioned procedure used for sample creation from the joint distribution $\mathbf{P}(\mathbf{X}, \mathbf{Y})$ in two-dimensional space is shown in Fig. 1. Uniform distribution, gradient boosting, and a genetic algorithm were used as the object distribution $\mathbf{P}(\mathbf{X})$, the conditional target distribution $\mathbf{P}(\mathbf{Y}|\mathbf{X})$, and the optimization method, respectively. It can be clearly seen that the number of designed $\mathbf{X}$ is low compared to the number of all possible samples from $\mathbf{P}(\mathbf{X}, \mathbf{Y})$. A detailed discussion of the outlined stages is given in Section 2.

In this paper we propose a flexible open-source framework GEFEST (Generative Evolution For Encoded STructures) for the generative design of two-dimensional physical objects from various applied fields. We consider physical objects that can be represented as flat polygons neglecting their internal structure. Our approach is based on the *generative design procedure*. The flexibility of the framework is attained due to the possibility of toolkit construction for a particular applied problem, where the toolkit implies a set of methods for implementing the *generative design procedure*. Since the GEFEST framework offers different approaches for each stage, it is possible to select the most suitable of them for a considered problem and thereby improve the final designs. Moreover, we provide an opportunity to implement custom approaches and modify the GEFEST core. In addition, the framework allows it to operate with physical objects of different natures due to the universal representation of each processed object. Our main contributions are the following:

- Formulation of the general approach to an arbitrary generative design problem.
- Novel generative design framework implemented as an open-source tool. The novelty lies in the individual approach to the solution of each stage in the *generative design procedure* achieved through flexible combination and modification of multiple methods for a particular problem.
- Validation of our framework on several real-world problems and comparison with baselines.

The paper is organized as follows. In Section 2 we consider related works to this paper. In Section 3 we explain our general approach to the generative design problem. In Section 4 we describe the implementation details of the proposed framework. In Section 5 we show real-world applications of the GEFEST framework. Finally, in Sections 6 and 7 we conclude this work with the pros and cons of our framework and future directions of research.

2. Related works

In this section a review of different approaches to the solution of each stage in the *generative design procedure* will be carried out. Moreover, existing generative design frameworks will be discussed.

2.1. Definition of object distribution

The brute force approach involves the selection of standard distributions $\mathbf{U}(\mathbf{X})$, for example, uniform or normal as $\mathbf{P}(\mathbf{X})$ (Nikitin et al., 2021; Mukkavaara and Sandberg, 2020). $\mathbf{U}(\mathbf{X})$ is defined on a variable $\mathbf{X}$ from design space $D = \{\mathbf{X} \in \mathbb{R}^n \,|\, f(\mathbf{X}) = 1\}$, where $f = \{0, 1\}$ is a geometrical and boundary constraint identification function, n is the dimension of the design variable $\mathbf{X}$. The described method is accompanied by several challenges. Firstly, dimension n may vary within

one considered problem, which causes difficulties in further conditional distribution $\mathbf{P}(\mathbf{Y}|\mathbf{X})$ modeling. Secondly, the sampling procedure from $\mathbf{U}(\mathbf{X})$ will be time-consuming if a form of D is non-trivial. In other words, samples $\mathbf{X} \sim \mathbf{U}(\mathbf{X})$ may not satisfy the boundary conditions, therefore it should be rejected by f, and the sampling algorithm should be repeated afterward. Lastly, the diversity of samples can be poor because of the simplicity of the selected standard distribution.

Another rapidly developing class of approaches for $\mathbf{P}(\mathbf{X})$ estimation is based on deep generative neural networks (Oh et al., 2019; Qian et al., 2022; Tan et al., 2020). These are data-driven approaches usually using implicit (or semi-implicit) models. The latter work in black box mode, and are only able to generate samples $\mathbf{X}$. In addition, they commonly require a great amount of data as well as training time. Despite the shortcomings, if the models are well trained, this class will be free from challenges that were inherent in the brute force approach. The vast majority of approaches are based on generative adversarial networks (Goodfellow et al., 2014) and variational autoencoders (Kingma and Welling, 2013).

2.2. Modeling of conditional target distribution

In generative design a well-established approach to the performance estimation (or estimation of the conditional target distribution $\mathbf{P}(\mathbf{Y}|\mathbf{X})$) of objects is numerical modeling. These models are based on equations of mathematical physics that can be solved using physical simulators $\mathbf{Sim}(\mathbf{X})$, for instance, species distribution modeling (Xu et al., 2021a), computational fluid dynamic (Xu et al., 2021a), COMSOL multiphysics (Nikitin et al., 2021), Simulating WAves Nearshore (James et al., 2018). Although such equation-based models provide a highly accurate approximation of performance (i.e. $\mathbf{Sim}(\mathbf{X}) \approx \mathbf{Y}$), this approach is computationally expensive and consequently extremely time-consuming.

Another approach that focuses on solving the mentioned problem of high computational complexity is surrogate models (Chen et al., 2020; Deshpande et al., 2020; Palar et al., 2019; González-Gorbeña et al., 2016). The main idea is to build a lightweight data-driven model $\mathbf{Surr}(\mathbf{X})$ that will approximate outputs of $\mathbf{Sim}(\mathbf{X})$ with reasonable accuracy. A wide range of machine learning and deep learning methods can be used as surrogate models: random forest, kriging, gradient boosting, neural networks, etc.

2.3. Solving optimization problem

After $\mathbf{P}(\mathbf{X})$ and $\mathbf{P}(\mathbf{Y}|\mathbf{X})$ are specified, the last stage of the *generative design procedure*, i.e., optimization problem (1), should be discussed. This is the main step to get samples with the highest performance from joint distribution $\mathbf{P}(\mathbf{X}, \mathbf{Y})$, which is the goal of generative design. Gradient-based methods and biologically inspired algorithms can be considered as tools for this purpose.

In generative design different variations of evolutionary algorithms are the most common approaches in solving optimization problems for real physical objects (Shen et al., 2022; Qian et al., 2022; Nikitin et al., 2021). In addition to generative design, evolutionary algorithms and other metaheuristics methods (Ramezani et al., 2021) are widely used in neural architecture search (Xue et al., 2021b,a), tumor diagnosis (Hu and Razmjooy, 2021; Tian et al., 2021) and other applied fields (Yin and Razmjooy, 2020). The increased attention to emphasized algorithms is caused by the following reasons: (1) gradients of $\mathbf{Sim}(\mathbf{X})$ with respect to $\mathbf{X}$ are difficult to calculate, whereas evolutionary algorithms are gradient-free; (2) evolutionary algorithms can be easily generalized to a multi-criteria optimization problem ($\mathbf{Y} \in \mathbb{R}^k, k > 1$), while for gradient-based methods this operation is more complicated. However, the convergence of such algorithms strongly depends on genetic operators and in some cases can be time-consuming.

With the growth of machine learning technologies, gradient approaches are gaining more and more popularity in generative design (Tan et al., 2020). If the surrogate model ensures a high approximation accuracy, in the optimization problem it is possible to replace $\mathbf{Sim}(\mathbf{X})$ with $\mathbf{Surr}(\mathbf{X})$. Modern optimizers, such as Adam or RMSProp, enable to obtain gradients of the machine learning model $\mathbf{Surr}(\mathbf{X})$ quickly and efficiently.

The major issue for the integration of optimization approaches with generative design is the uncertainty of the model and the underlying physical system itself. First, some physical processes (e.g. thermodynamic mixing, wind waves or turbulent flows) can be considered stochastic rather than deterministic (Zielinski, 1991). Second, the modeling results of both physics-based and data-driven models also contain a significant stochastic component (Palmer and Williams, 2008). For this reason, the generative design results can be very unstable to any disturbances since the underlying distribution $\mathbf{P}(\mathbf{X})$ can be considered multimodal. A similar problem arises in the analysis of non-linear dynamical systems (Xu et al., 2021b). For example, the stability problems are solved in control algorithms of dynamic technical systems (Djordjevic et al., 2022). In the field of generative design, the implementation of optimization techniques also should take the stability issues into account to produce the appropriate solutions. In this case, the population-based methods can be preferred since it allows preserving better diversity of the solutions (Cheng et al., 2018).

2.4. Generative design frameworks

Attempts to develop generative design frameworks have been made for quite some time. It is necessary to highlight the work presented by Singh and Gu (2012). In this paper the authors proposed a framework based on different generative design approaches: genetic algorithms, swarm intelligence, L-systems, cellular automata and shape grammars. Eventually, they came to the conclusion that there is no universal approach to the generative design problem. In other words, a generative design framework needs to be flexible, that is, to provide various approaches to a specific problem.

In recent years the development of generative design frameworks has become more widespread. For example, Mukkavaara and Sandberg (2020) devised a framework for architectural design. In the presented paper researchers tried to develop a generic framework including several generators of designs (genetic algorithm and random sampling). However, this work suffers from the lack of modern deep learning models.

Moreover, the Autodesk generative design framework (Buonamici et al., 2020) deserves special attention as one of the best-known commercial software. This framework provides a flexible approach for the user-specified problem. For obtaining designs numerical analysis of differential equations is performed on external cloud servers. Despite the advantages of the given software, its implementation details and essential features are not discussed.

Increasingly, works with a combination of deep learning networks and traditional approaches are being published. For instance, Oh et al. (2019) presented a framework that consists of topology optimization (Solid Isotropic Material with Penalization) and a generative model (Generative Adversarial Network). This approach demonstrated high diversity and aesthetics of created designs in resolving the two-dimensional wheel problem. Nevertheless, it has not been validated on tasks from different applied fields.

To consider the difference between the GEFEST and other generative design frameworks we propose to compare five, as we believe, most important properties of the tools:

- *Applicability to the design of 3D objects.*
- *Versatility*. The possibility of applicability for tasks from various fields.
- *Flexibility*. The ability of the framework to provide different approaches to a specific problem.

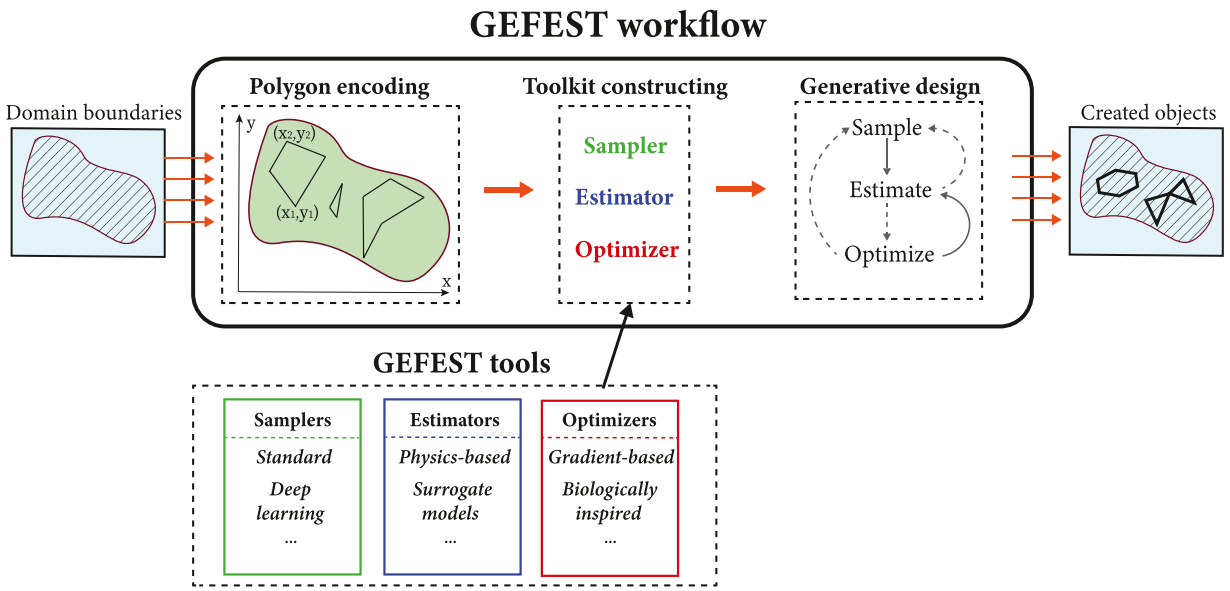


Fig. 2. The GEFEST approach for the generative design of physical objects including polygon encoding, toolkit constructing, and generative design. In the generative design stage hatched and thick lines mean possible and required gates, respectively.

Table 1
Comparison between generative design frameworks using selected properties.

Framework	*Property*				
	3D	Versatility	Flexibility	Openness	Arbitrary shapes
Singh and Gu (2012)	✗	✗	✓	✗	✗
Mukkavaara and Sandberg (2020)	✓	✗	✗	✗	✓
Buonamici et al. (2020)	✓	✓	✗	✗	✓
Oh et al. (2019)	✗	✗	✗	✗	✗
Proposed framework (GEFEST)	✗	✓	✓	✓	✗

- *Openness*. This property includes two points: does the framework provide an opportunity for the user to implement their own modules and is the source code open?
- *Arbitrary shapes*. The property answers the question: is the tool capable of potentially approximating physical objects of any shape?

Based on the analysis of Table 1, we can conclude that **the potential advantages** of the proposed approach against other generative design frameworks are as follows: *flexibility* (compared to Singh and Gu (2012) we also incorporate deep learning algorithms in our framework) and *openness*. In other words, our framework provides the opportunity to consider different approaches to a specific problem and incorporate user-defined modules that implements SOTA approaches. However, **the disadvantages** are that GEFEST cannot be applied to 3D objects and cannot approximate physical objects of arbitrary shapes (as we show in Section 5.1.2)

3. GEFEST approach for generative design

As was mentioned earlier, there is no universal approach to generative design. Each applied problem requires detailed consideration and long-term research. However, the combined application of deep learning, numerical modeling and optimization can be regarded as a general trend in solutions to the generative design problem. With the right combination of different methods from these specified branches, well crafted physical objects can be produced. This is the main idea of the GEFEST approach.

The GEFEST framework is a modular tool for the generative design of two-dimensional physical objects that can be presented as flat polygons without an internal structure. The essential points of the framework architecture are shown in Fig. 2. First of all, the GEFEST receives the boundaries of the considered two-dimensional domain as input. Optimization will be carried out in this domain. Then the workflow including polygon encoding, toolkit constructing, and generative design is performed. In the output, GEFEST generates samples from the joint probability distribution $\mathbf{P}(\mathbf{X}, \mathbf{Y})$.

Our approach is mostly based on three tools:

- **Sampler**. This tool is designed to generate samples $\mathbf{X}$ (we associate this variable with a physical object). In other words, it solves the first stage of the *generative design procedure*.
- **Estimator**. We developed the Estimator to obtain the performance of a physical object, i.e., target variable $\mathbf{Y}$.
- **Optimizer**. The final stage of the *generative design procedure* can be solved by the Optimizer. The latter is aimed at obtaining the most efficient $\mathbf{X}$.

These tools are presented abstractly because we want to provide the desired flexibility in choosing a specific implementation of each tool. Whichever implementation is chosen, each of them will be within the same procedure. Some implementations are listed in Section 3.2.

3.1. Polygon encoding

The first stage of the GEFEST workflow is the polygon encoding. This procedure allows the representation of real physical objects as two-dimensional (flat) polygons. We distinguish two types of structure: opened-form and closed-form, however, the framework operates them in the same manner. An example of encoding is shown in Fig. 3. It is clear that each polygon node is identified by corresponding Cartesian coordinates. Hence, the polygon can be described by a set of points, i.e. $\mathbf{X} = (x_1, y_1, \ldots, x_j, y_j), \mathbf{X} \in \mathbb{R}^{2j}$. If $\mathbf{X}$ satisfies the boundary and geometrical constraints ($f = 1$), it will belong to the design space D. The restrictions are based on the domain boundaries that are obtained as input in GEFEST. A set of different $\mathbf{X}$ corresponding to all

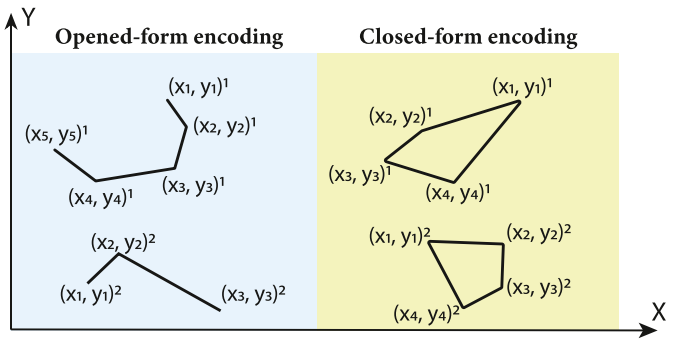


Fig. 3. Opened-form and closed-form polygon encoding in Cartesian coordinates.

possible polygons and their combinations defines the design space. It is worth noting that the dimension of this space is high, especially for closed-form polygons.

3.2. Toolkit constructing using GEFEST tools

After the specification of the design space, it is necessary to set a certain approach for each stage of the *generative design procedure*. To accomplish this purpose, we implemented GEFEST tools (*Samplers*, *Estimators*, *Optimizers*) including various computational methods. Semantically, each tool can be divided into two classes: deep learning (Generative Adversarial Networks, Variational Auto Encoders, etc.) and standard (standard statistical distributions) for Samplers; surrogate models (fully connected/convolutional neural networks, kriging, etc.) and physics-based (different physics simulators) for Estimators; biologically-inspired (genetic/evolutionary algorithms, etc.) and gradient-based (Adam, gradient descent, etc.) for Optimizers. Such a variety of approaches provides the opportunity to deal with applied problems of different natures and goals. For example, for problems with opened-form polygons and multi-criteria target (Nikitin et al., 2020) it will be effective to select a standard distribution and multi-objective evolutionary algorithm as the sampler and optimizer, respectively. In cases requiring high diversity of polygons with closed-form, a deep learning sampler is preferable.

3.2.1. GEFEST standard sampler

The principal requirement imposed on the sampler is the computational efficiency of sample generation. We would like to create correct polygons, i.e., without any self-intersections or intersections with other domain elements and out-of-bound parts in an acceptable amount of time. For these purposes, we implemented the GEFEST standard sampler including two stages: generation of the centroid region and generation of points inside this region. We refer our sampler to the standard class because it is based on standard statistical distributions. In a simplified form, the sampling procedure is presented in Algorithm 1 and Fig. 4.

First of all, the centroid is created using a uniform distribution on a rectangle area. This poses the central point of the region called the centroid region. The size of the latter is determined by the radius, which is also a sample from a uniform distribution. However, the support of this new one is different. More precisely, for the radius we considered uniform distribution on the ray $\left(0, \frac{|\Omega|}{n_{poly}}\right]$ instead of the rectangle area Ω. The selection of such an upper bound is motivated by the following idea: when the number of polygons increases, it becomes more difficult to find a correct region with a large radius. To avoid this, the ray can be reduced by n_{poly} times. Finally, when the correct region is created, a polygon can be freely generated inside it. The polygon consists of points sampled from a normal distribution with parameters x and $\frac{r}{3}$, the mean and the variance, respectively (the latter is chosen taking into account the three-sigma rule). Note that in Algorithm 1 we do not adduce details about the stopping criterion (when the while loop takes a good deal of time), postprocessing, and recreating the centroid (in cases when radius creation becomes time-consuming).

Algorithm 1 GEFEST standard sampling

Require: Ω, Σ, N, max ▷ Rectangle area, optimization domain, number of polygons to generate and maximum number of points in polygon
Ensure: S ▷ The GEFEST Structure (set of polygons)
$S = Array()$
while $|S| < N$ **do**
  $n_{poly} = |S| + 1$ ▷ Number of already created polys and additional one to avoid 0
  $x \sim \mathbb{U}(\Omega)$ ▷ Creating centroid
  while x not in Σ **do**
    $x \sim \mathbb{U}(\Omega)$ ▷ Repeat until x is in optimization domain
  $r \sim \mathbb{U}((0, \frac{|\Omega|}{n_{poly}}])$ ▷ Creating radius of centroid region
  while $\vec{xr}$ *is incorrect* **do**
    $r \sim \mathbb{U}((0, \frac{|\Omega|}{n_{poly}}])$ ▷ Repeat if region is incorrect
  $n_{point} = \mathbb{U}_{int}(max)$ ▷ Number of points
  $P = Array()$ ▷ Initialization of polygon
  while $|P| \neq n_{point}$ **do**
    $p \sim \mathbb{N}(x, \frac{r}{3})$ ▷ Creating polygon point
    $P.append(p)$
  $S.append(P)$
return S

3.2.2. Deep learning estimators and samplers

Within the GEFEST framework, the deep learning estimator and sampler work with the images of polygons, not the polygons themselves. This is caused by the various dimension of the latter. As can be seen from Fig. 3, the number of points describing the polygons can be different. It depends on the number of its segments. Thus, in the application of classical machine learning methods, certain difficulties arise (the dimension of the input data varies). To avoid these issues, it is necessary to make the universal polygon parameterization insensitive to the number of its points. In this work we considered three-dimensional matrix parameterization that is invariant to changes in the number of points. In other words, an image is mapped to each polygon or polygon structure. The produced images are binary in which maximum intensity corresponds to the polygon. Such a representation allows the consideration of well-established practical tools, namely convolutional neural networks.

The generalized architecture of the deep learning estimator is shown in Fig. 5. The estimator passes an input binary matrix through the convolutional backbone and prediction layers to approximate the target. As options for the backbone, various widely-used convolutional architectures (Khan et al., 2020) can be listed: VGG, ResNet, UNet, AlexNet. The choice is dictated by the complexity of the considered problem and the amount of training data. Fully connected networks are usually used as prediction layers.

A deep learning sampler works the opposite way, it tries to create a realistic output binary matrix as shown in Fig. 6. The main purpose of the sampler is to create a plausible binary matrix from noise. The generator should produce images with the correct geometric form of polygons. The most common approaches to the construction of, for example, a convolutional generator are Variational Auto Encoder, Generative Adversarial Networks, Normalizing Flows, etc. The final step is the transformation from matrix parameterization to Cartesian coordinates encoding. This can be done using classical computer vision algorithms detecting edges (Canny, 1986).

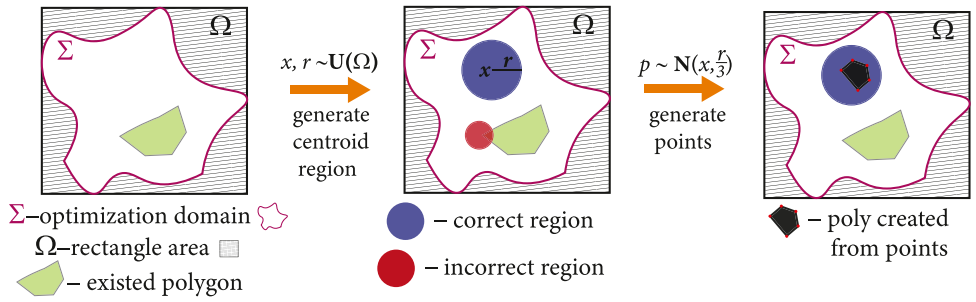


Fig. 4. Visualization of a single step of the GEFEST standard sampler. Our sampler operates in two main stages: centroid generation and points generation inside the centroid region. Here $\mathbf{U}(\Omega)$– uniform distribution of x, r on the rectangle area; x, r – centroid and radius of the centroid region, respectively; $\mathbf{N}(x, \frac{r}{3})$– normal distribution, where $x, \frac{r}{3}$ — its mean and variance, respectively.

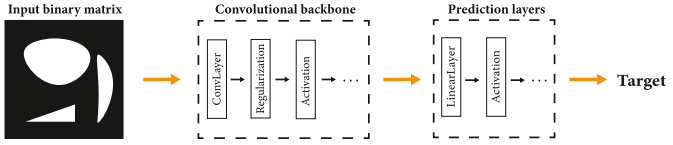


Fig. 5. The generalized architecture of the deep learning estimator takes the image of polygons as input.

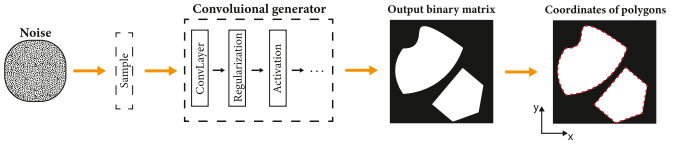


Fig. 6. The generalized architecture of the deep learning sampler in GEFEST. It tries to produce a sample from the noise distribution using a convolutional generator.

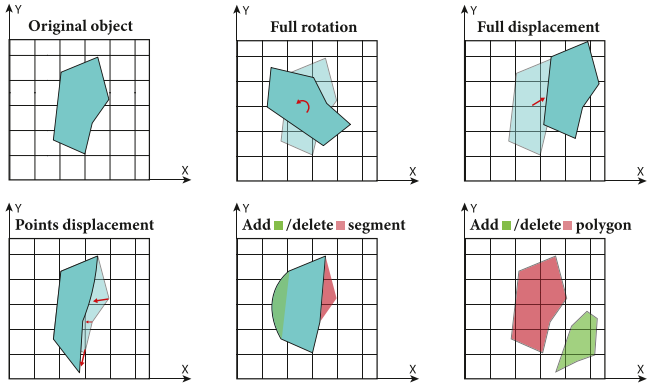


Fig. 7. Geometrical transformations over polygons: rotation, displacement, adding, removal.

3.2.3. Evolutionary core

The optimization step, particularly the biologically inspired one, deserves special attention. It is known that the convergence rate of every evolutionary algorithm strongly depends on the genetic operators (Song et al., 2021). For effective convergence, the latter must be implemented taking into account the semantics of the problem being solved. In the case of two-dimensional polygons, it is natural to consider geometric transformations shown in Fig. 7 as mutation operators. We realized the following transformations over polygons: rotation of the center of mass by a certain angle; polygon and points displacement; adding/deleting segments and polygons. Moreover, we used the crossover operator shown in Fig. 8. These genetic operators constitute the evolutionary core of GEFEST. More precisely, every evolutionary algorithm used in the framework utilizes this evolutionary core.

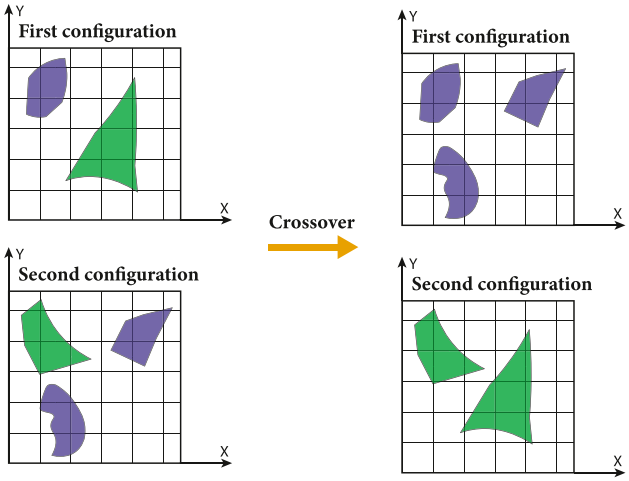


Fig. 8. Example of a crossover operator. In the left figures, we present configurations before transformation, and in the right figures — after.

3.3. Generative design

The fundamental stage of the GEFEST workflow is the generative design, depicted as the last step in Fig. 2 and in Algorithm 2. This procedure is based on three principles: sampling, estimation, and optimization, which can be combined in different ways. The *traditional approach* includes lines 1, 3, 4, and 6 in Algorithm 2. It performs single sample operation, whereas estimation and optimization are repeated until the stopping criterion is reached. Therefore, low exploration and high exploitation rates are inherent in this process. However, in

Algorithm 2 GEFEST generative design

Require: $P = (S, E, O)$ ▷ User-defined toolkit
Ensure: D_{fin} ▷ Final designed objects
1: $D_{curr} \leftarrow S$.sample() ▷ Initial designs
2: **while** *stopCriteria* **do**
3: $PF \leftarrow E$.estimate(D_{curr}) ▷ Design performance
4: $D_{curr} \leftarrow E$.select(PF, D_{curr}) ▷ Selecting k best objects
5: **if** O.required **then**
6: $D_{curr} \leftarrow O$.optimize(D_{curr}, PF)
7: **if** S.required **then**
8: $D_{sample} \leftarrow S$.sample()
9: $D_{curr} \leftarrow D_{sample} \cup D_{curr}$ ▷ Combination
10: **else** ▷ Skip optimization
11: $D_{sample} \leftarrow S$.sample()
12: $D_{curr} \leftarrow D_{sample} \cup D_{curr}$
13: $D_{fin} \leftarrow D_{curr}$
14: **return** D_{fin}

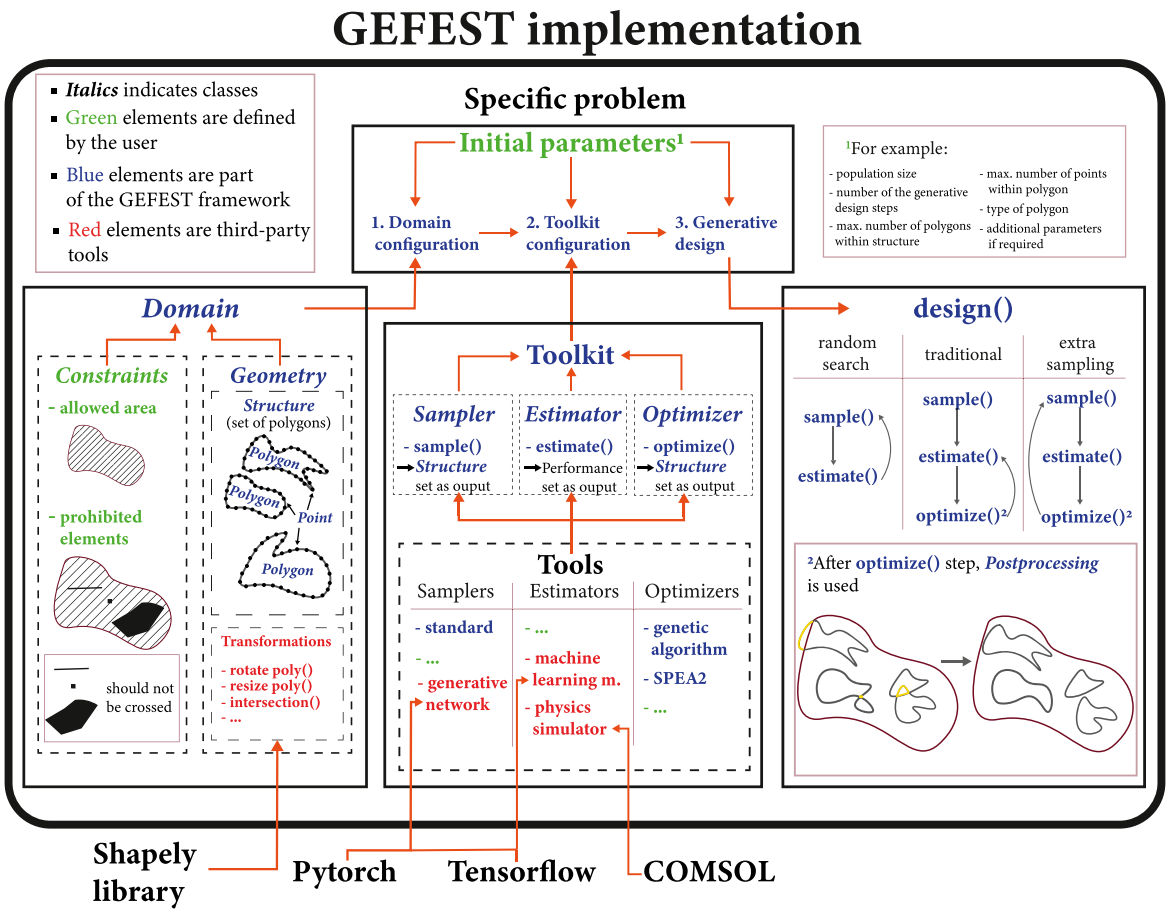


Fig. 9. The scheme of the GEFEST framework implementation. The specific problem can be solved by completing three main steps: domain and toolkit configuration and design.

some problems, it is necessary to increase the exploration rate. This can be done by integration of lines 8 and 9, i.e., by executing the addition sample operation at each stage of the loop, we call this *extra sampling*. As an alternative, it is possible to skip the optimization phase completely (lines 1, 3, 4, 11, and 12). Such a method, characterized by a great exploration rate, is called *random search*.

4. Open-source software framework

We provide our framework GEFEST as an open-source tool for the solution of a user-specified generative design problem. The architecture of the framework is presented in Fig. 9. Certain problems may be solved by adjusting the following main blocks: domain, toolkit, and design. It is worth noting that access to them must be carried out in the presented order. This stems from the fact that subsequent blocks require elements configured at the previous stages. Furthermore, to make user interaction available, we divided GEFEST elements into three groups: user-defined, internal and external.

4.1. Domain

First of all, it is necessary to configure the *Domain* class, which is responsible for the whole information about the geometry of the problem. In order to do this, the user should define *Constraints*, namely the allowed area and prohibited elements. The first of them determines the domain in which optimization will take place. The second one is responsible for fixed elements within the domain that should not be intersected by the generated polygons.

In addition to the *Constraints*, we set the *Geometry* class, which consists of *Structure* and transformations. *Structure* is necessary for the creation of abstraction over real objects in accordance with the following hierarchy scheme: *Structure* → *Polygon* → *Point*. Lastly, to realize the geometrical transformations over polygons (for instance, rotation, resizing and etc.) we provide access to the external methods from the Shapely library (Gillies et al., 2007).

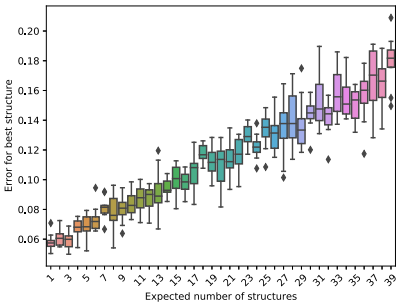
4.2. Toolkit

The next stage, after the *Domain* specification, is toolkit configuration. Note that this is the core part affecting the performance of created objects. The toolkit comprises three classes: *Sampler*, *Estimator*, and *Optimizer* realizing corresponding abstract methods (sample(), estimate(). and optimize()). Such an abstraction is necessary to define the general behavior of objects, which are inserted in the GEFEST tools. In the latter block, we implemented our own objects and integrated external elements from machine learning frameworks and physics-based applications. In addition, we provided access to custom tools.

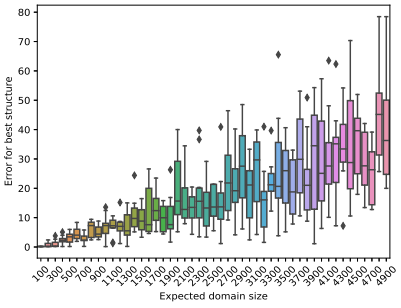
It should be pointed out that the main requirement imposed on tools is consistency. In other words, the inputs and outputs of class methods should have the same type. For example, if the sample() method delivers a *Structure* array, then the input of the estimate() method should have the same type. Initially, all methods operate with the *Structure* array, however other options are available (for example, array of images).

4.3. Design

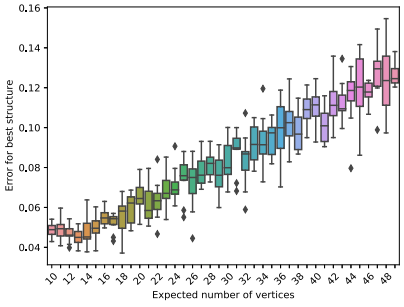
The final step is a generative design based on Algorithm 2. Here we offer three possible options (random search, traditional and extra sampling methods), as was discussed earlier. We just emphasize that after the optimize() step, generated structures undergo *Postprocessing* by which defective polygons (self-intersected, out-of-domain and etc.) are corrected.



(a) Variable number of objects in optimal solution



(c) Variable size of the search domain



(b) Variable number of vertices in optimal solution

Fig. 10. The dependence of solution search error from (a) number of objects in the optimal solution; (b) number of vertices in the optimal solution; (c) size of the search domain.

Table 2
Summary of all experimental studies including the main goal.

Section	Type	Main goal
5.1	Synthetic	Reveal the applicability of the GEFEST framework
5.2	Coastal engineering	Demonstrate the benefits of combining different estimators
5.3	Microfluidics	Compare deep learning and standard samplers
5.4	Heat Sources	Illustrate how to perform generative design with an only dataset
5.5	Oil field	Show how to apply GEFEST to a specific subproblem

5. Experimental studies

This section considers the application of the GEFEST framework to physical objects of different natures. The main purpose of the experiments lay in the demonstration of framework versatility and flexibility by addressing real-world problems from different specific areas using the same GEFEST approach (Fig. 2). It is necessary to highlight that we cannot apply other generative design frameworks discussed in Section 2.4 because they are not open source. However, we implemented state-of-the-art approaches in specific areas as a part of the framework and involved them in the experiments. Also, we analyzed existing results for the considered problems and proposed our own baselines. A summary of all experimental studies is presented in Table 2.

It should be noted that the computational complexity and working time strongly depend on the chosen methods. For example, if neural networks are selected as an estimator or sampler, then a GPU is usually required for training and utilization. Also, some optimization algorithms can work slower than others, e.g. SPEA2 works slower than Differential Evolution. Thus, the computational complexity may vary depending on the chosen sampler (generative neural networks are calculated faster than classical distributions, but require training); estimator (physics-based simulators require much more time for calculation than deep learning algorithms), and optimizer (metaheuristic algorithms are usually slower than gradient-based methods).

All subsequent experiments were conducted using a Windows 2008 Server with 32 core units and DGX 1 NVIDIA Cluster with Tesla V100.

5.1. Synthetic cases

5.1.1. Applicability of GEFEST

It is necessary to analyze the applicability of GEFEST to problems with different properties before getting down to real-world cases. To achieve this purpose we conducted several synthetic experiments in which we varied different properties to get the optimal solution: number of polygons, number of vertices, and domain size. The main purpose of these experiments is to investigate the relationship between the complexity of the expected optimal solution and the efficiency of the search. The hypothesis is that there is a linear relationship for GEFEST since it implements a generalized approach that is not specific to some sub-classes of the generative design problem.

In this section, we considered the following GEFEST tools:

- **Sampler**: standard approach (GEFEST Standard Sampler)
- **Estimator**: synthetic estimator based on the distance between the obtained and reference solution.
- **Optimizer**: biologically-inspired method (Genetic Algorithm).

Variable number of objects

The first experiment was devoted to an analysis of GEFEST's applicability for reproducing configurations that consist of various numbers of polygons (from 1 to 30). A boxplot that describes the effectiveness of GEFEST for this task is presented in Fig. 10(a).

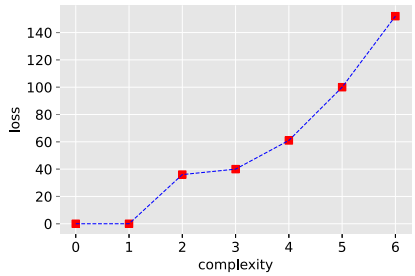


Fig. 11. Behavior of the loss function (estimator values) with the increasing complexity of the figure to be approximated.

Variable number of vertices

The main difference in the second experiment was the variability of the number of vertices in a single polygon (from 10 to 100) instead of the number of polygons. Obtained results are presented in Fig. 10(b).

Variable domain size

Another factor that can influence the effectiveness of the proposed approach is the size of the domain that represents the two-dimensional search space. Obtained results are presented in Fig. 10(c).

As can be seen, the restoration error's dependence on the configuration's complexity can be considered near-linear for synthetic cases. It empirically confirms the formulated hypothesis. We can conclude that the proposed approach has no bias towards any property of the problem under consideration.

5.1.2. Biasness of GEFEST

In previous experiments, we concluded that GEFEST could potentially be applied to problems with different properties (domain sizes, number of structures, and vertices). However, it is also necessary to understand the bias of GEFEST to the type of structures that need to be generated. For this purpose, we considered the problem of figure approximation with various complexity. We set the same tools as in the previous section except for the estimator. As the latter, we considered the maximum length of the relative complement of two polygons. The results are presented in Figs. 11 and 12. As can be seen from the results, our framework can easily handle right-form figures (examples 1, 2, 4 in Fig. 12) and their combinations (examples 3 and 5 in Fig. 12). However, if the polygon is not derived from the right one, the framework cannot approximate its irregularities (examples 6 and 7 in Fig. 12) even with a small number of points.

So, it is possible to conclude that the proposed approach with the GEFEST standard sampler can be applied to structures of regular shape and their derivatives. However, for complex shapes, the standard sampler can be changed to a deep learning sampler, as we showed in Section 5.3.

5.1.3. Sensitivity to hyperparameters choice

Hyperparameters are one of the most important tuning values and their configuration can make a serious effect on the final result. So, in this part, we aim to analyze the sensitivity of GEFEST to the hyperparameters choice.

In GEFEST we distinguish the following main hyperparameters: population size, crossover, and mutation rates. To analyze the sensitivity we approximated two circles (example four in Fig. 12) each of which contains 15 points. As the estimator we choose an area-to-length ratio, it is well known that for a circle this ratio is the largest among other figures. The number of design steps was equal to 200 and each run was repeated five times. The results are presented in Fig. 13.

As it can be seen from Fig. 13(a) the results are slightly worse for high crossover rates (0.8 and 0.9) and low mutation rates (0.2, 0.1) compared to other values. This is due to the fact that mutations are more effective than crossover for polygons in our realization. So, we recommend giving preference to mutations in your experiments, for instance, 0.3 and 0.7 for crossover and mutation rate, respectively. In Fig. 13(b) we can see that for a small population size (5 individuals) the result is the worst. For the other values, the error is approximately the same. To give recommendations on the choice of the population size, we considered the dependency between the time to run 200 steps and the population size. It is presented in Fig. 14.

As we can see, the dependency can be considered linear. So, our recommendation is to choose medium values for the population size (for example 50–150). Because at high values, the post-processing time of polygons becomes significant and at low values, the results are worse.

5.2. Coastal engineering

In this subsection, we consider a real-world design problem from the coastal engineering field. This task is dedicated to protecting critical objects (targets) in the water area from natural phenomena (sea waves). Breakwater structures are being developed for these purposes. The main goal is to find a configuration of breakwaters (opened-form polygons in terms of the GEFEST encoding), which minimizes the wave heights at significant points. The cost of breakwaters should also be taken into account. More details about this problem can be found in existing works (Xu et al., 2021a; Nikitin et al., 2020).

In Fig. 15 configuration of our breakwaters design problem is shown. First of all, we specified bathymetry (water depth at each point of the water area), as well as the direction and speed of the wind. In this particular case, we set two land areas, fairways, and three targets. These objects are fixed, whereas the position of breakwaters (red polygon in Fig. 15) should be optimized to ensure maximum protection of critical objects from sea waves. In addition, a crucial constraint is imposed: protecting constructions should not cross fixed objects. Breakwater length, meanwhile, is of great importance, since it directly affects the cost. Therefore, the target variable is two-dimensional $\mathbf{Y} \in \mathbb{R}^2$: the first component is responsible for the sum of wave heights, and the second one — is for the cost of breakwaters. And the third step of the *generative design procedure* is a multi-objective optimization problem with constraints (see Appendix A.1 for formal basics). Worth noting that in this case, we consider the optimization problem (1) in terms of minimization. As a baseline solution, we choose a configuration of breakwaters specially created by ourselves. It is shown in Fig. 15(b). Also, we determined a low value of the breakwater protection coefficient. It means that wave height at the point will increase quite quickly when moving away from the breakwater.

In this problem statement, we considered the following GEFEST tools:

- **Sampler**: standard approach (GEFEST Standard Sampler (GSS))
- **Estimator**: physics-based simulator (Simulating WAves Nearshore) and deep learning (Convolutional Neural Network (CNN)),
- **Optimizer**: biologically-inspired methods (differential evolution (DE) and SPEA2, details about the last algorithm are provided in Appendix A.2).

The main purpose of this example is to demonstrate how physics-based and deep learning estimators can be combined to enhance the obtained solution. Moreover, we aim to show the opportunity to utilize different optimizers. Thus, we constructed the following toolkits based on the mentioned GEFEST tools: (1) GSS + SWAN + DE; (2) GSS + SWAN + SPEA2; (3) GSS + CNN/SWAN + SPEA2, and compared them. The second toolkit implements an existing approach from the work (Nikitin et al., 2020). It represents the optimization strategy that is widely used in state-of-the-art works (Elchahal et al., 2013), so we incorporated it into GEFEST. All toolkits used Algorithm 2 in a *traditional manner*, except for the third one, in which we added *extra sampling* to increase the exploration rate. Algorithm 2 was repeated three times for each

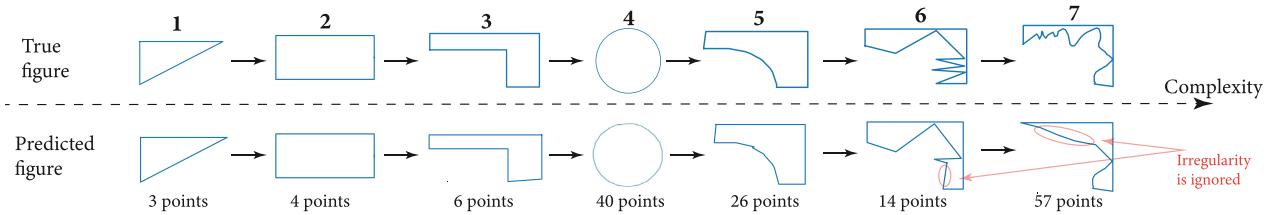
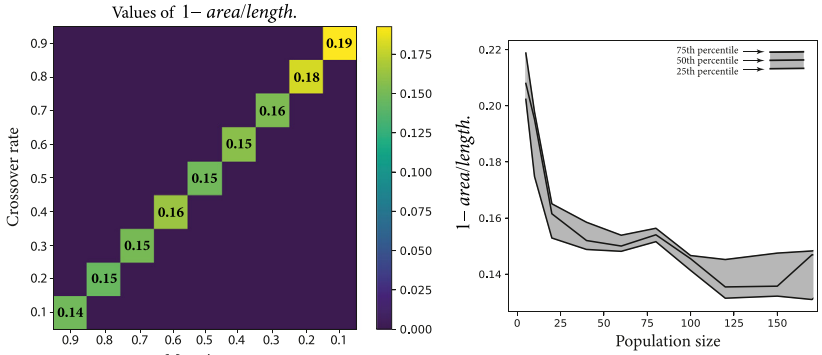


Fig. 12. Real figures that should be approximated (top row) and the GEFEST prediction (bottom row). The framework easily copes with right-form figures (examples 1, 2, 3, 4, 5). However, it cannot approximate the irregularities (examples 6, 7).



(a) We ran experiment five times and calculated average values.

(b) We ran experiment five times and calculated percentiles.

Fig. 13. The dependency of the area-to-length ratio on the crossover and mutation rates (a) and on the population size (b). The lower values are better.

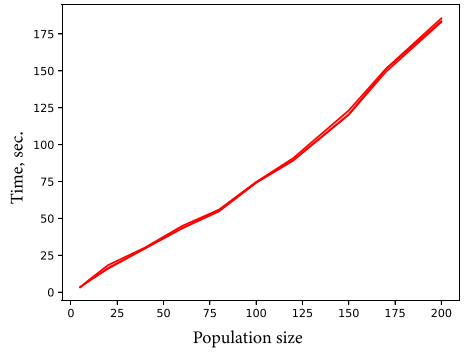


Fig. 14. Time (in seconds) to run 200 steps of the generative design for different population sizes.

Algorithm 3 Combination of physics based and deep learning estimators

Require: $DL, PB, sample$ ▷ Deep learning, physics-based estimators and sample for estimation

Ensure: pf ▷ Performance of sample

1: $pf = DL.estimate(sample)$ ▷ Calculating performance of sample using deep learning model

2: **if** $pf < threshold$ **then**

3: $pf = PB.estimate(sample)$ ▷ Recalculating performance using physics-based model if the sample is close to the minimum

4: **return** pf

toolkit with a time limit of 10 h for one run. Also, we set the population size to 30 and the archive size to 15.

The third toolkit needs to be discussed in more detail. The most time-consuming procedure among the considered GEFEST tools is the physics-based SWAN model estimation of the wave heights. In order to reduce the number of SWAN calls we included an additional deep learning estimator, which is noticeably more lightweight. On the other side, the deep learning estimator is less accurate. Nevertheless, high accuracy is required only for solutions close to the minimum, whereas in other cases a rough approximation is sufficient. More formally, such a combination is described in Algorithm 3. This procedure allows the execution of more optimization steps dew to fewer calls to the physics-based model. The parameter *threshold* provides an opportunity to skip unsuccessful samples, in our case, it is equal to 6.0.

The deep learning estimator chosen for this problem is based on the generalized architecture (Fig. 5). To collect data for its training, Algorithm 2 with the second toolkit (GSS + SWAN + SPEA2) was in progress for 3 h. As a result, about 700 labeled examples were obtained. After the deep learning estimator was prepared, about 6.7 h were left for generative design with the third toolkit. The details about the architecture and training procedures are in Appendix B.1.

The results of the experiments are shown in Table 3 and Fig. 16. We decided to use hypervolume as a metric for comparison because it is the main measure in multi-objective optimization problems (Zitzler et al., 2001, 2004). For each toolkit, we calculated the 25th, 50th, and 75th percentiles of the latter based on three runs. Moreover, in Table 3 the minimum wave heights among three runs are shown. As can be clearly seen from the results, the toolkit with the deep learning estimator shows superior performance. This outcome can be explained by the greater number of generative design steps (140 against 60 and 80 for non-deep learning estimator toolkits). The integration of the deep learning estimator enables the application of the physics-based model only for important breakwaters and provides more time for further optimization steps. This fact allows for the inclusion of *extra sampling*. Furthermore, one out of three of our approaches surpasses the baseline solution.

In Fig. 17 some created samples for each toolkit are presented. As can be observed, the only approach permitting generating the breakwater close to small land includes the deep learning estimator. Furthermore, it excels in sample diversity. Such outcomes are related to the higher exploration rate of the third toolkit compared to others. As was mentioned above, it was achieved by the use of *extra sampling*.

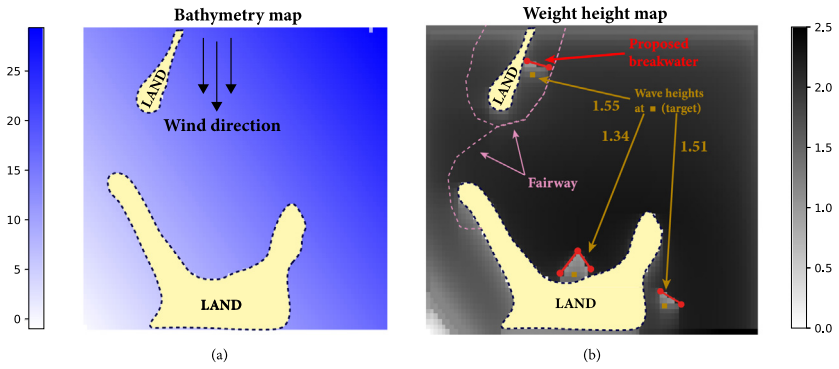


Fig. 15. The problem statement for the generative design of coastal breakwaters. Bathymetry increases from the lower left corner (0 meters depth) to the upper right (25 meters depth); wave height depends on the position of the breakwaters, the maximum value is 2.5. In the right figure the baseline solution is presented.

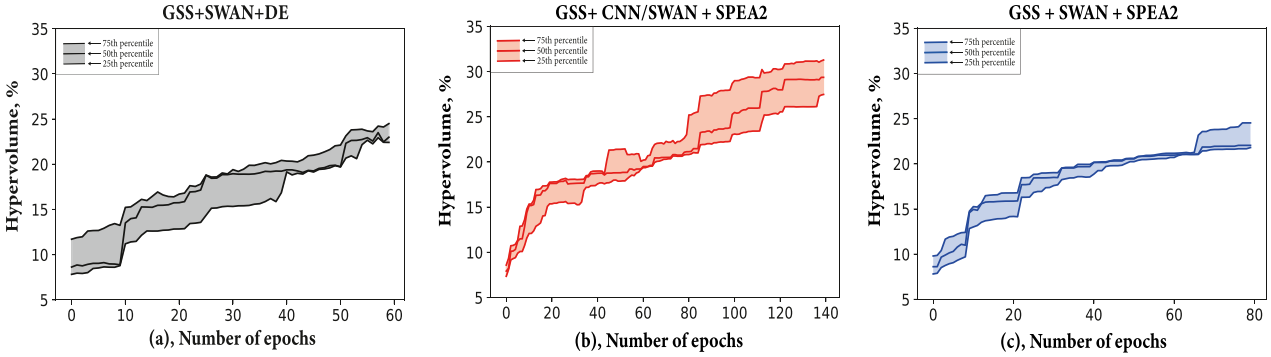


Fig. 16. Hypervolume of the population at each step (epoch) of the generative design. The hypervolume was calculated based on three runs, and after the 25th, 50th, 75th percentiles were calculated.

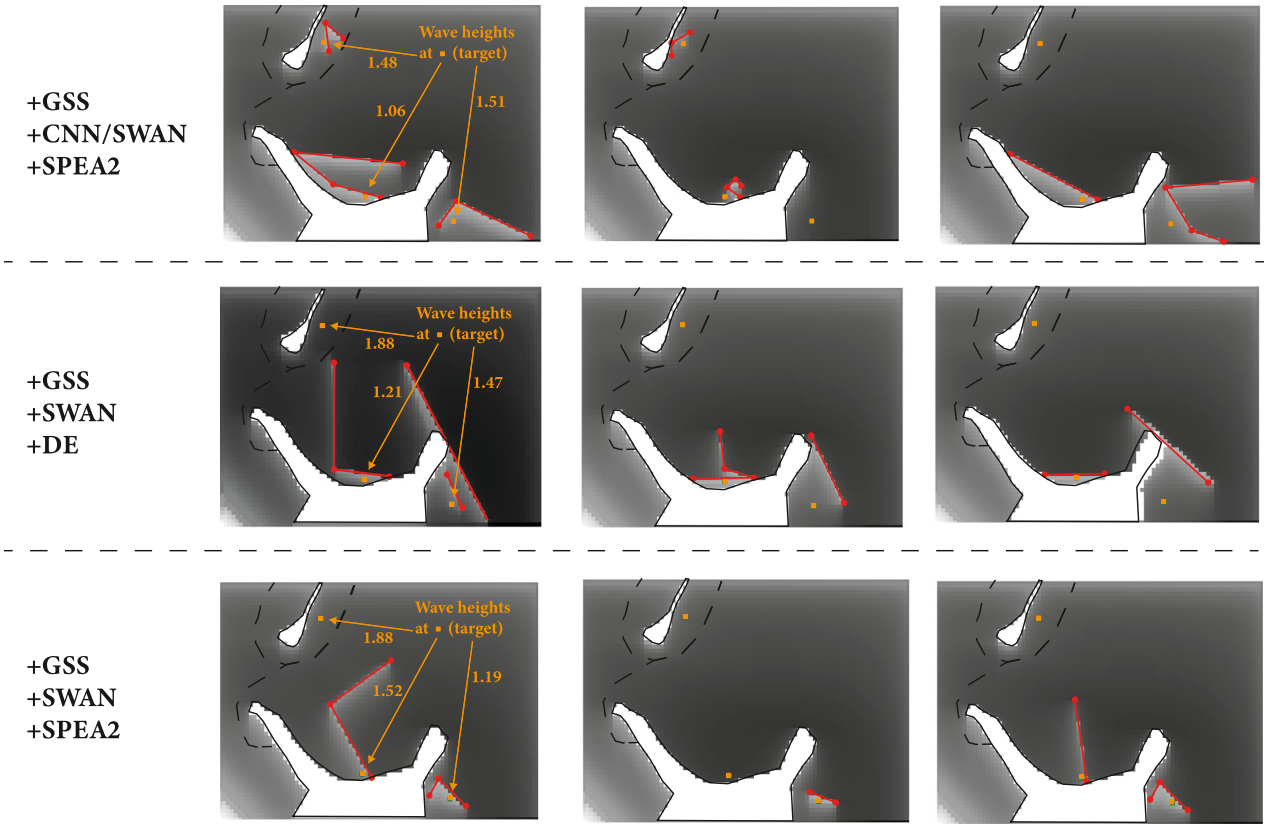


Fig. 17. Visualization of some created samples for the breakwaters design problem. Three samples for each toolkit are demonstrated. The left figures in each row show the best-found samples with corresponding wave heights for each target.

Table 3
Comparison between different toolkits. The hypervolume is calculated relative to the maximum possible. Here we present the hypervolume at the final step of the generative design. Wave heights are the sum of wave heights at all targets. In the table arrow ↑ reflects the larger the better rule, for ↓ the opposite is true.

Toolkit	Hypervolume ↑ (%), percentile			Wave heights ↓ (m)
	25th	50th	75th	
GSS+SWAN+DE	22.42	22.99	24.49	4.56
GSS+SWAN/CNN+SPEA2	**27.47**	**29.36**	**31.28**	**4.05**
GSS+SWAN+SPEA2 (Nikitin et al., 2021)	21.80	22.04	24.53	4.59
Baseline	-	-	-	4.41

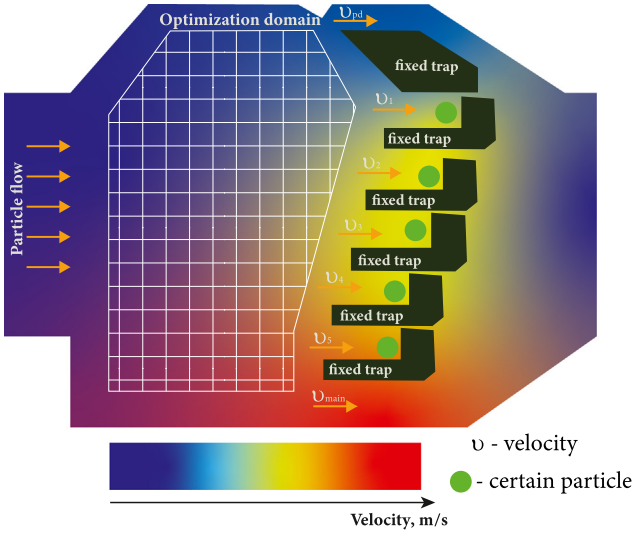


Fig. 18. The problem statement for the generative design of microfluidic devices. A flow of particles passes through this device. Barriers (closed-form polygon in GEFEST term) can be located inside the optimization domain.

5.3. Microfluidics

A microfluidic device is a system with a size of about hundreds of micrometers, permeated with several microchannels, the fluid flow through which is investigated. One of the most prominent applications of microfluidics is studying the behaviors of single red blood cells for further biological analysis, disease diagnostics and etc. Conventionally, only certain particles need to be analyzed and thus they should be separated from the unwanted flow components. For these purposes, hydrodynamic traps are used. The faster the flow passes between these traps, the higher the trapping probability becomes. For more details, refer to the works (Grigorev et al., 2022; Man et al., 2020).

The general problem statement is shown in Fig. 18. This task can be formulated as the construction of barriers that can be represented as closed-form polygons (in terms of GEFEST). The main goal is to create polygons inside the optimization domain to maximize the velocity of particles through fixed traps (1–5). The increase in the flow rate enhances the capture probability of certain particles by fixed traps. In addition, the reduction of the velocity through the main and pressure-dropping (PD) channels facilitates achieving the primary goal. Hence, the target variable can be written as follows:

$$\mathbf{Y} = \frac{\sum_{i=1}^{5} \upsilon_i}{\upsilon_{\text{main}} + \upsilon_{\text{pd}}} \tag{2}$$

In this case, the optimization problem includes only boundary restrictions without constraints caused by fixed objects within the domain (as it was in the previous section).

For the generative design of the hydrodynamic cell traps, we utilized the following GEFEST tools:

- **Sampler**: standard approach (GEFEST Standard Sampler) and deep learning (Generative Neural Network (GNN)),
- **Estimator**: physics-based simulator (COMSOL Multiphysics),
- **Optimizer**: biologically-inspired method (Genetic Algorithm).

Table 4
Comparison between final target variable for considered toolkits. Here we presented the best value of the target variable among individuals of the population in the last epoch. We ran the experiment three times and calculated the 25th, 50th, and 75th percentiles. In the table arrow ↑ reflects the larger the better rule.

Toolkit	Target variable ↑			Best target ↑
	25th	50th	75th	
GSS + COMSOL + GA	0.333	**0.347**	**0.354**	**0.361**
GNN + COMSOL + GA	**0.333**	0.336	0.337	0.339
GA (Grigorev et al., 2022)	-	-	-	0.329

For the closed-form polygon encoding, the application of the deep generative model is more reasonable than in the case of the opened-form polygons. This fact is associated with greater variability of closed structures.

The main goal of this study is to reveal the benefits of the deep learning sampler compared to the standard sampler. To this end, we built the following toolkits: (1) GSS + COMSOL + GA; (2) GNN + COMSOL + GA. In order to show the influence of samplers on the generative design results, we used Algorithm 2 in *extra sampling* mode. As in the previous section, the calculation was repeated three times with a time limit of 10 h for each. The population size was set to 40. As a baseline solution we took the result from the paper (Grigorev et al., 2022), which can be considered as state-of-the-art in microfluidics. In short, the approach of Grigorev et al. (2022) is based on a genetic algorithm that starts optimization with an expert solution. Thus, we further denote it as GA.

For the preparation of the deep learning sampler, we collected 100 000 training objects using the standard GEFEST sampler, which took nearly 1.5 h. In addition, the training in the deep generative model required about 30 min. Moreover, beyond the main experiment, we compared different generative neural networks with regard to sample diversity and quality. Details are presented in Appendix B.2. Based on the comparison, for this problem, we chose the Adversarial Auto Encoder as a deep learning sampler.

The results of the experiments are shown in Table 4 and Fig. 19.

As can be seen from Table 4, the toolkit based on the GEFEST standard sampler performs slightly better. However, the difference between toolkits is negligible. Moreover, both of our approaches surpass the state-of-the-art solution. Actually, it is necessary to highlight another significant fact. As depicted in Fig. 19, the deep learning sampler-based toolkit enables the creation of higher target variable samples at the initial steps of the generative design. This suggests that the deep learning sampler produces more beneficial primary objects, that is, allows to get the generative design process to start from better samples. In addition, in Fig. 20 several samples created by both methods are presented. As can be observed, the deep learning based approach generates more diverse and unconventional samples. Creation of such objects using the standard sampler would be a cumbersome and lengthy procedure.

Besides, we compared the sampling time for both approaches. The results are demonstrated in Table 5.

It can be clearly seen that the deep learning sampler works approximately four times faster than the standard method. Worth noting that

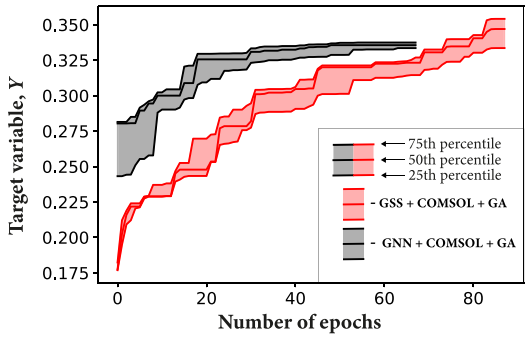


Fig. 19. Dependence of target variable on the number of the generative design step for two considered toolkits. The generative design with the normal distribution-based toolkit takes a greater number of epochs due to the training time of the deep learning sampler which we took into account.

Table 5
Sampling time of 50, 500, and 1000 objects for deep learning and the standard sampler. Each time measurement was repeated 10 times.

Sampler	Time (s.) for sampling		
	50	500	1000
Deep learning	**0.56 ± 0.02**	**6.2 ± 0.3**	**13.2 ± 0.2**
Standard	2.49 ± 0.17	25.3 ± 0.6	50.6 ± 1.1

the most time-consuming operation in deep learning sampling is the GEFEST polygon encoding, that is, a transformation from the image to the Cartesian coordinate set.

In Fig. 21 we demonstrated the best objects found using two toolkits. As can be seen, obtained structures closely resemble each other. In both cases, the optimization converges to easy-form polygons. However, for the other problems, the opposite may be required.

5.4. Heat-source systems

In this part, we demonstrate the capabilities of the GEFEST framework as a tool for dealing with already prepared datasets. Note that the corresponding dataset for the generative design field can be hardly found in open access. Moreover, researchers usually investigate their own specific problems and therefore conventional benchmarks are scarce. Here, we considered an open dataset from the related field that is engaged in the heat-source systems investigation (Chen et al., 2021a).

Heat-source systems are part of an electronic microdevice (micro- or nanometers-sized) that poses a source of heat and therefore temperature field. The control over the temperature distribution within the microdevice (usually called heat management) plays a significant role in practical applications. For instance, real-time knowledge of temperature distribution allows avoiding technical failures, in particular caused by exceeding the critical temperature, thereby lengthening the life of the electronic device. However, the distribution across the entire device is commonly unknown. But instead, we have the temperature of the monitoring points at our disposal. Thus, the problem of temperature field reconstruction within the microdevice often attracts the attention of researchers. For more details, refer to the existing works (Chen et al., 2020, 2021a).

The chosen dataset consists of 10 000 examples, each representing two images. We demonstrate one instance in Fig. 22. The left image provides the selected heat-source system inside the electronic microdevice. In this dataset, heat-sources are divided into three types according to their shape (circle, rectangle and capsule). Their number remains constant and equals 10, aside from the rare cases when it is reduced by one. Also, each source generates heat evenly, that is, the value is the same within the component. The right image is a temperature field produced by the given combination of heat-sources. Note that adiabatic conditions are applied to the boundaries of the device, except for one point, in which the heat sink is located. The temperature of the latter is a constant quantity equal to 298 K.

In most existing works, authors examined the reconstruction of the temperature field using a set of heat-sources (Sun et al., 2022; Chen et al., 2020, 2021b). However, we formulated another problem more specific to the generative design. Our goal is the production of a heat-source system that insures a minimum average temperature within the device. Thus, the target variable had the following form:

$$\mathbf{Y} = \frac{1}{M}\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} T_{ij}, \tag{3}$$

where M, N — the number of grid points, T_{ij} — the temperature at a certain point. It is worth noting that in this experiment the optimization problem was considered in terms of minimization. As a baseline solution, we chose an example from the dataset with the minimum target value.

For the solution to the mentioned problem, we selected the following GEFEST tools:

- **Sampler**: deep learning (Generative Neural Network),
- **Estimator**: deep learning (Convolutional Neural Network),
- **Optimizer**: -.

Data-driven methods were chosen because we limited ourselves to the dataset only in this case. More precisely, the data generators (that produce new heat-sources) and the physics simulators (that accurately estimate the temperature field for new objects) were left beyond the scope of the described experiment. Thus, the constructed toolkit (GNN + CNN) is completely based on deep learning models. Since in the toolkit the optimizer is absent, Algorithm 2 was run in *random search* manner. Nevertheless, the deep learning based toolkit can be expanded by including a certain optimizer, but this option will be discussed later.

As in the previous section, the Adversarial Auto Encoder was selected as a deep learning sampler. We trained this model on images of the heat components without taking into account its temperature field. The deep learning estimator learned to approximate the average temperature within the device from the image of heat-sources. Some details are presented in Appendix B.3. Note that the number of iterations for Algorithm 2 was set to 10 000.

Before proceeding to the results of the generative design, it is necessary to compare the existing samples and the samples created by the deep learning model, presented in Fig. 23. As can be observed, the generative model has acquired the ability to generalize. In other words, in addition to existing objects, the deep learning sampler generates objects that were not in the original dataset. Consequently, the number of the heat-sources can vary. Such a generalization may produce new unseen samples, which possess a significant value in the generative design.

The results of the generative design are presented in Figs. 24 and 25. As can be seen from the comparison between the minimum temperature in the dataset and the minimum value obtained during generative design, we found the configuration of heat-sources reduced the average temperature by 25 degrees. This advantageous configuration is depicted in Fig. 25. Note that the found object was not present in the original dataset.

5.5. Oil field planning

In this part, we consider the problem of the optimal location of wells and roads in an oil field. The location of wells is the most important stage of field development. So, here we aim to maximize the production of the field. In modern works (Minton, 2012; Tukur et al., 2019; Jesmani et al., 2020) real limitations in the development of fields are not considered. More precisely, it does not pay attention to various geographical objects (lakes, swamps, or rivers) that make it difficult to build wells and roads. In such a way we consider a more general and realistic formulation of the problem. Our optimization task is to find

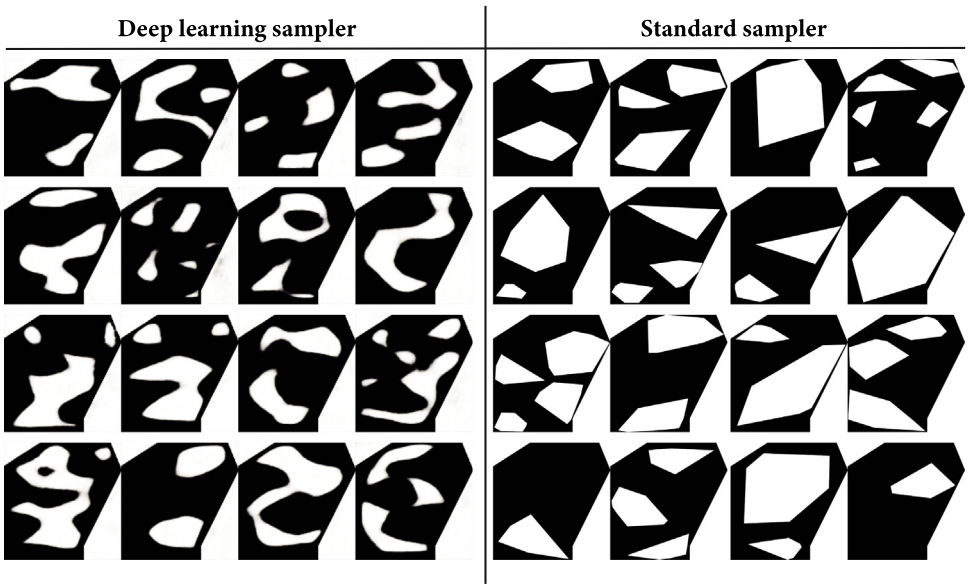


Fig. 20. Several objects created using deep learning and the standard sampler for the microfluidic generative design problem. An adversarial autoencoder was used as a deep learning sampler.

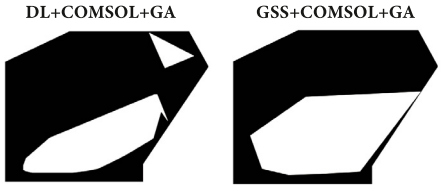


Fig. 21. The best objects found by two toolkits in three runs.

the optimal location of wells and roads taking into account inaccessible areas that represent geographical objects. Thus, the joint optimization of wells and roads is being considered in the presence of a set of areas that either prohibit the construction of roads and wells or allow it to be done with a significant penalty. An example of a field with a road, an inaccessible area, and three wells is shown in Fig. 26.

The main goal of this study is to illustrate how to apply the GEFEST framework only to a subproblem. In other words, GEFEST should solve one part of the whole task, and another tool should solve the rest. For this purpose, we used a cooperative algorithm that can be divided into two parts:

- an algorithm for optimizing the location of wells considering roads and inaccessible areas as fixed;
- an algorithm for optimizing roads considering the location of wells and inaccessible areas as fixed.

As the first algorithm, various approaches developed on the basis of the GA and PSO algorithms were used. At this stage, the basic algorithms have been modified to consider the structures of wells and deposits and also to take into account inaccessible points and roads during optimization. As the second one, the GEFEST was used. For both parts of the joint algorithm, special objective functions were used. More precisely, these functions have the following form:

1. When optimizing the location of wells, the NPV function is used as the target function, which reflects the economic benefit from the developed field (Minton, 2012):

$$NPV = \sum_{t=1}^{T} \frac{CF_t}{(1+r)^t} - C^{capex} - r_{road} \times e_{dist},$$

where

- r is the percentage of profit or discount rate;
- T is the number of periods;
- CF_t is profit in the period t, which is equal to the difference between revenue and expenses in this period. The costs of operating the well are constant, and the profit depends on the volume of oil that was produced in a given period of time in accordance with the physics-based simulator;
- C^{capex} is the cost of well development work. This value takes into account the costs of well construction at the field. The cost depends on the length and gradient of the well;
- r_{road} is the coefficient of the cost of one road cell;
- e_{dist} is total distance from wells to road.

A mathematical model was used as a geosimulator that uniformly pumps oil based on only a part of the oil in a certain volume. This model was used for calculating CF_t. The synthetic deposit *SPE2*[1] was used as test data.

2. A function linearly dependent on the length of the necessary roads is considered as an objective function for road optimization:

$$NPV_{road} = r_{road} \times (len_{road} + e_{dist}),$$

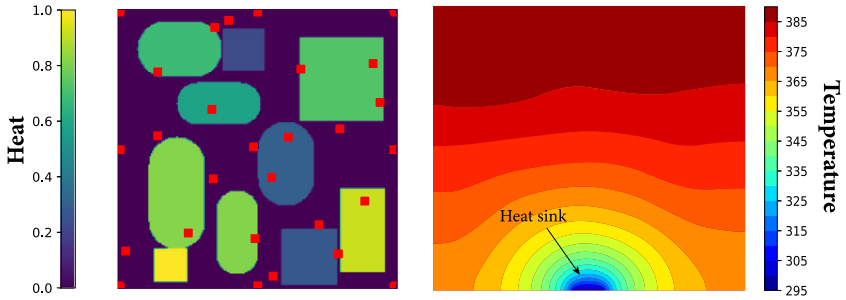
where len_{road} is the length of the roads built;

The cooperative approach to the optimization of wells and roads in the field consists of the periodic exchange of information between the well optimization algorithm and the road optimization algorithm with the transfer of information about the current optimal locations.

Take a closer look at using the GEFEST framework to solve the problem described in this paragraph. Besides the optimization problem mentioned above, this tool was used for the generation of inaccessible areas. In terms of GEFEST, encoding roads are opened-form polygons with fixed beginnings and ends, inaccessible areas are closed-form polygons. For the optimization problem, we configured the GEFEST toolkit based on the following tools:

- **Sampler**: standard approach (GEFEST Standard Sampler),
- **Estimator**: synthetic approach (NPV_{road} function),
- **Optimizer**: biologically-inspired method (Genetic Algorithm).

[1] https://www.spe.org/web/csp/datasets/set02.htm#case2a



(a) Heat components in the form of cirlcle, rectangle and capsule. Monitoring points shown as ■

(b) Temperature field generated by the heat components system.

Fig. 22. One example from the heat-sources dataset. Heat components (a) and corresponding temperature field (b).

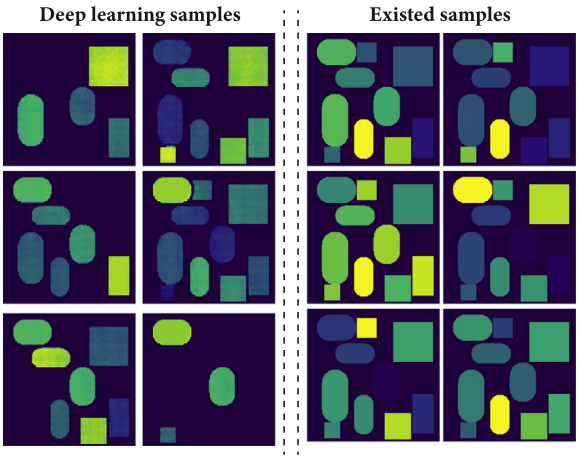


Fig. 23. Visualization of several existing samples from the dataset and samples generated by the deep learning model.

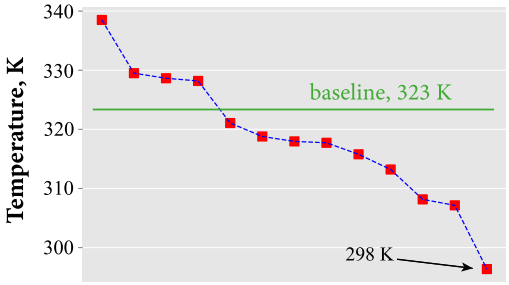
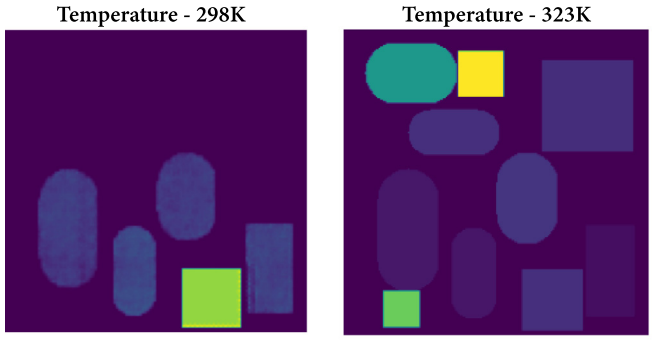


Fig. 24. The average temperature of the best samples found during generative design.



(a) GEFEST solution. The best found sample providing the minimum average temperature. It was not presented in the original dataset.

(b) Baseline. Object with the minimal average temperature in the original dataset.

Fig. 25. Visual representation of the (a) GEFEST solution and (b) baseline solution for heat-sources.

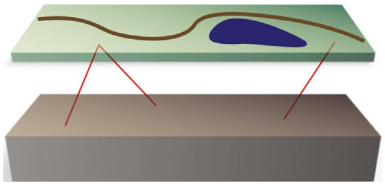


Fig. 26. Example of a real field model (wells are colored in red, the road in brown, and the lake in blue).

For the generation of inaccessible areas, we used the GEFEST Standard Sampler.

In Fig. 27 an example of the location of objects on the surface of the deposit is shown. In this case, a field with five wells and an inaccessible area filling 2% of the field surface is presented. The blue line shows the road, the yellow dots represent the optimized location of five wells on the surface of the field and the red dots represent 8 inaccessible points where it is impossible to build wells and roads.

Investigating this task, no developed or published solutions were found. To evaluate the effectiveness of the realized joint algorithm for wells and roads optimization, taking into account inaccessible areas, a naive approach was chosen, which is based on the following steps:

1. Optimization of the location of wells without considering roads.
2. Building a road through optimized well locations without optimizing roads.

The presented algorithm was used as a baseline for evaluating the effectiveness of the joint approach. Unlike the cooperative approach, this naive one does not imply optimization of the road. To compare the baseline and our approach, the following metric was built:

$$NPV_{joint} = NPV - NPV_{road} - r_{road} \times e_{dist},$$

This function allows for taking into account the economic benefits of developed wells and the costs of necessary roads.

In Tables 6 and 7 results for different sizes of the inaccessible area and various road cost coefficients are presented. These tables show the values of the coefficient K:

$$K = \left(\frac{NPV_{joint}^{cooperative}}{NPV_{joint}^{naive}} - 1 \right) * 100\%. \quad (4)$$

As can be seen from the presented results, the cooperative approach is more effective than the naive approach in all cases.

For convergence of the joint algorithm, 200 iterations of the algorithm were used to optimize the location of roads. An example of an averaged convergence curve is shown in Fig. 28. We presented the

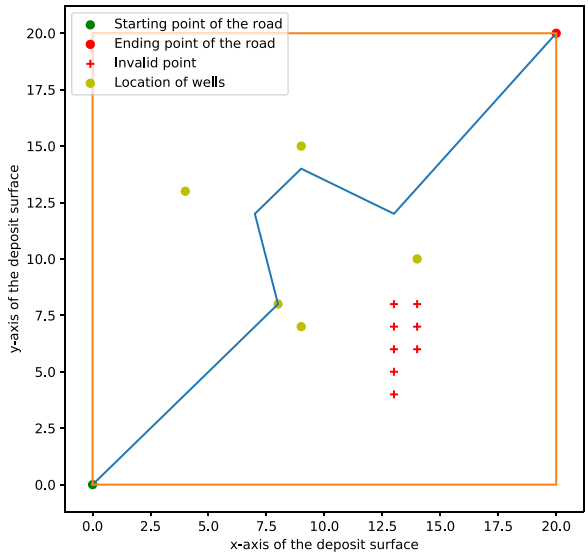


Fig. 27. An example of the location of roads, oil wells, and inaccessible points on the surface of the field.

Table 6
Evaluation of the effectiveness of the cooperative and naive approaches depending on the size of the inaccessible areas.

Number of wells	Size of inaccessible areas			
	0%	2%	4%	6%
3 wells	4.13%	6.41%	7.35%	10.93%
5 wells	4.88%	5.16%	5.96%	8.77%
7 wells	4.56%	6.05%	6.31%	9.19%

Table 7
Evaluation of the effectiveness of the cooperative and naive approaches depending on the road cost coefficient.

Number of wells	Road cost coefficient			
	500	1000	3000	4000
3 wells	4.14%	6.41%	11.95%	18.09%
5 wells	2.77%	4.17%	7.19%	11.84%
7 wells	2.84%	4.96%	9.67%	13.91%

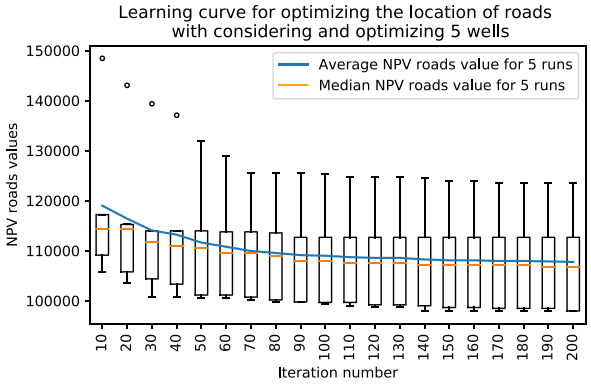


Fig. 28. An example of the convergence curve of the objective function for roads when optimizing the location of 5 wells.

dependency of NPV_{road} on the iteration number in the case of location optimization of 5 wells with an inaccessible area occupying 2% of the deposit.

So, the developed cooperative approach with the GEFEST framework proved to be consistently more effective than the naive approach. Also with the growth of the road price, the efficiency (K-coefficient) of the cooperative algorithm increases compared to the naive algorithm.

6. Discussions

In the experimental studies, we demonstrated the flexibility of the GEFEST approach, which can be applied to various practical problems. In addition, we revealed the benefits of some tools that can be valuable in addressing unexamined real-world problems. Moreover, we showed the opportunity of GEFEST to generate novel objects when the problem has a limited dataset.

In the coastal engineering problem, the combination of a physics-based model and convolutional neural network in a single approach led to the results surpassing those provided by standard methods. Furthermore, we obtained a hard-to-reach extremum, that is, the structure of polygons covering all targets by increasing the exploration rate (*extra sampling* procedure). Such techniques can be easily applied to different problems taking into account that the deep learning estimator should be sufficiently trained.

In the microfluidic problem, we showed the contribution of the deep learning sampler. The latter allows the creation of higher-performance objects at the initial steps of the generative design. In addition, the generative network can produce not merely regular samples but also diverse and unusual objects in contrast to the standard sampler. It is worth noting that these properties depend on the *inductive bias* of the utilized generative neural network. More precisely, it may be the case that a generative model creates samples analogous to those contained in the training set, i.e. reproducing them without any distinguishing features. The choice of a particular model is generally conditioned by the specific problem under consideration. Anyway, despite the appearance of the samples, the inference of the deep learning sampler is faster than for the standard, as was shown earlier. Thus, the deep learning sampler is preferable, if a large set of objects is needed.

In the heat-source systems problem, we demonstrated that the generative design can also be performed using only a prepared dataset. In this case, we used Algorithm 2 in *random search* manner, that is, without the optimization step. However, another option exists: it is possible to integrate a gradient-based optimizer in the toolkit. In this case, the gradient of the deep learning estimator is calculated with respect to the input object. Then, the input updates by gradient descent step. Nevertheless, such a procedure can only be applied to neural networks well-trained on huge datasets.

Finally, in the oil field design problem, we illustrated that our framework can be implemented as part of the solution to the whole problem. Considering such a situation can be useful for users who want to apply our framework only to a specific subproblem of their task.

6.1. Limitations

The primary limitation of our framework is the impossibility of its application for three-dimensional objects that are of the greatest interest in practical fields.

Furthermore, the GEFEST standard sampler is only suitable for the production of arbitrary polygonal samples. More precisely, user-defined shapes (e.g. circles, rectangles, ellipses) are infeasible. In such cases, it is necessary to utilize other generators or train generative networks on prepared datasets.

Finally, in our approach we considered physical objects neglecting their internal structure. These limitations can be crucial in some generative design problems.

6.2. Future work

Future work focuses on extensions of our framework to three-dimensional problems and other types of physical objects. Further, it would be useful to consider dimensionality reduction methods because of the redundant dimension of polygon structure images. Finally, it is essential to explore gradient-based algorithms as a part of the generative design concept.

Table 8
The summary of all experimental studies, including quantitative results. In the table, the arrow ↑ (↓) reflects the larger (smaller) the better rule. *SOTA means state-of-the-art approach or result. **result or approach was not presented earlier and we take our own baseline. ***we calculate this value as the average among values of Tables 7 and 6. In square brackets, we indicated the improvements achieved by GEFEST in percentages. In parentheses, we note the specific approach by which the value was achieved. The following abbreviations are used: GSS — GEFEST Standard Sampler; GA — Genetic Algorithm; GNN — Generative Neural Network; CNN — Convolutional Neural Network.

Applied field	Coastal engineering (Section 5.2)	Microfluidics (Section 5.3)	Heat Sources (Section 5.4)	Oil field (Section 5.5)
Values	*Metric*			
	Wave heights ↓, m	Relative velocity ↑, (2)	Average temperature ↓, K, (3)	NPV coefficient ↑, %, (4)
*SOTA**	4.59 (Nikitin et al., 2020) (GSS+SWAN+SPEA2)	0.329 (Grigorev et al., 2022) (GA)	323 (**)	0 (**)
GEFEST	**4.05** [+12%] (GSS+SWAN/CNN+SPEA2)	**0.361** [+9%] (GSS + COMSOL + GA)	**298** [+8%] (GNN + CNN)	+7.39*** (GSS + GA)

7. Conclusions

In this paper, we propose a novel open-source framework for the generative design of two-dimensional physical objects. The developed approach is based on three general principles: sampling, estimation, and optimization. These elements constitute the core of the solution to every generative design problem that can be applied to various real-world tasks.

We demonstrated the relevance and flexibility of our approach by addressing different applied tasks from ocean engineering, microfluidics, heat-source systems, and oil field planning. Furthermore, it was shown that the modification of the general approach ensures superior performance over baselines. In Table 8 we present the final quantity comparison between the GEFEST and SOTA/baseline results. As it can be seen GEFEST gives 12% improvements in the coastal engineering problem; 9% in microfluidics; 8% in heat-sources and 7% in oil field planning.

Finally, we revealed the benefits of some GEFEST tools inspired by state-of-the-art solutions. For instance, in the coastal engineering problem, the deep learning estimator can be combined with the physics-based simulator to skip less important objects; in the microfluidic problem, the deep learning sampler can create more diverse and higher-performance objects than the standard method. And, as we believe, these findings can provide objects with refined performance in other generative design problems.

CRediT authorship contribution statement

Nikita O. Starodubcev: Conceptualization, Methodology, Software, Writing – original draft. **Nikolay O. Nikitin:** Software, Visualization, Writing – review & editing. **Elizaveta A. Andronova:** Conceptualization, Methodology, Writing – review & editing. **Konstantin G. Gavaza:** Data analysis. **Denis O. Sidorenko:** Software, Visualization. **Anna V. Kalyuzhnaya:** Supervision, Project administration.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: This research is financially supported by The Russian Scientific Foundation, Agreement #22-71-00094.

Code and data availability

The software implementation of all the described methods and algorithms is available as a part of the GEFEST framework in the open repository https://github.com/ITMO-NSS-team/GEFEST. The code and data for experimental studies are available at https://github.com/ITMO-NSS-team/GEFEST-paper-experiments.

Acknowledgment

This research is financially supported by The Russian Scientific Foundation, Agreement #22-71-00094.

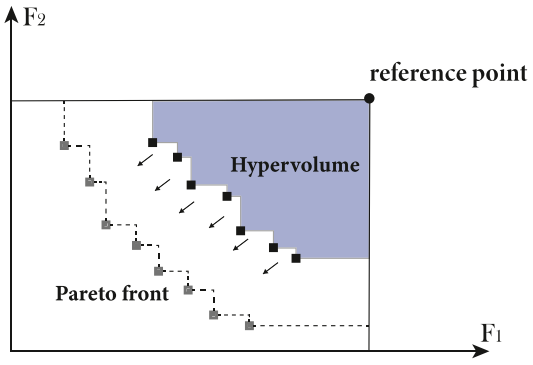


Fig. A.29. Hypervolume definition. As the algorithm converges, the Pareto front tends to the lower left corner, increasing the hypervolume.

Appendix A. Multi-objective optimization problem

Here basic concepts of a multi-objective optimization problem with constraints are discussed. Moreover, some details about the SPEA2 algorithm are presented.

A.1. Basic concepts

We consider a multi-objective optimization problem with constraints, which can be formulated as follows:

$$\begin{aligned} &\min_{x\in X} F(x) \\ &\text{s.t. } \mathbf{g}(x) = 0 \\ &\quad\ \mathbf{s}(x) \geq 1 \end{aligned} \tag{A.1}$$

where $\mathbf{F} : X \to \mathbb{R}^m, m \geq 2$ — is a multi-criteria function; $\mathbf{g}(x) = 0, \mathbf{g}(x) \geq 1$ — are constraints that are required to be satisfied. Usually, there is no solution that minimizes all criteria of $\mathbf{F}$ simultaneously. In such cases, the Pareto front is considered. This is a set of all Pareto efficient points in the functional space, more formally (Zitzler et al., 2001):

Definition 1 (*Pareto Front*)**.** Let $\mathbf{F} : \mathbb{R}^m \to \mathbb{R}^n$ is a vector function with a set of values $\mathbb{Y} = \{y \in \mathbb{R}^n : y = \mathbf{F}(x), x \in \mathbb{R}^m\}$. The Pareto front is a set $\mathbb{P}(\mathbb{Y}) = \{y \in \mathbb{Y} : \forall y' \neq y \in \mathbb{Y}\ y \succ y'\}$.

In the definition sign "$\succ$" means Pareto domination.

Definition 2 (*Pareto Domination*)**.** Let $y^1, y^2 \in \mathbb{R}^m$, y^1 Pareto dominates y^2 $(y_1 \succ y_2) \iff \forall i = 1 \dots m\ y_i^1 \leq y_i^2$ and $\exists\ j = 1 \dots m : y_j^1 < y_j^2$.

The main measure of convergence of a multi-objective optimization algorithm is a hypervolume, which can be defined as the area between the Pareto front and the reference point as shown in Fig. A.29. As the algorithm converges, the Pareto front aspires to the left bottom corner (in case of a minimization problem), thus hypervolume should increase.

A.2. SPEA2 algorithm

The Strength Pareto Evolutionary Algorithm 2 (SPEA2) is an evolutionary-based algorithm for approximating the Pareto front. In SPEA2 two types of populations are considered: the archive **A** and the population **P**. The archive contains individuals not dominated by any other. In other words, an archive is necessary to preserve elitism. The population **P** allows bringing new individuals via genetic transformation (mutation, selection, crossover). A core of the SPEA2 is a fitness calculation based on raw and density functions:

$$\mathbf{F}(I) = \mathbf{R}(I) + \mathbf{D}(I), \tag{A.2}$$

where I — is the individual from the population, $\mathbf{R}, \mathbf{D}$ — are the raw and density functions, which are defined as follows:

$$\begin{aligned} \mathbf{R}(I) &= \sum_{I' \in \mathbf{A} \cup \mathbf{P}} \left[I' \succ I \right] \cdot \mathbf{S}(I'), \\ \mathbf{S}(I') &= \#\ \{ I \mid I \in \mathbf{A} \cup \mathbf{P} : I' \succ I \}. \end{aligned} \tag{A.3}$$

$$\mathbf{D}(I) = \frac{1}{d_I^k + 2}. \tag{A.4}$$

In (A.3) $S(I')$ is the strength, which defines the number of individuals dominated by I', in (A.4) d_I^k is the distance from **I**-th individual to its k-th neighbor in the functional space. For non-dominated solutions, $\mathbf{R}(I) = 0$, whereas the density function is necessary to increase the diversity of the population.

The main loop of SPEA2 is shown in Algorithm 4, (Zitzler et al., 2004).

Algorithm 4 SPEA2

Require: M, N, T ▷ Population, archive size and maximum number of steps
Ensure: A ▷ Archive population
1: *Random initialization*
2: *Fitness calculation* ▷ Assigning fitness to each individual from $\mathbf{P} \cup \mathbf{A}$
3: *Environmental selection* ▷ Filling the archive with not dominated solutions
4: *Termination* ▷ If stopping criterion is satisfied then return **A**
5: *Mating selection* ▷ Perform selection operator on $\mathbf{P} \cup \mathbf{A}$
6: *Variation* ▷ Apply crossover and mutation operators to the selected population

Appendix B. Deep learning models

Here architectures and training processes of the deep learning models used in the experimental studies are discussed.

B.1. Coastal engineering estimator

The deep learning estimator takes the image of the breakwaters as input corresponding to Fig. 5. The estimator consists of three convolutional layers with $L2$ regularization, one GlobalMaxPooling layer, and two fully-connected layers. The total number of parameters is equal to 372 449.

For the evaluation of the convolutional neural network, we used a validation set created outside the stage of generative design. The results of the deep learning estimator approximation of wave heights are shown in Table B.9 and Fig. B.30.

As can be seen from Fig. B.30, some predictions of the deep model are prone to be overestimated. However, in our problem high accuracy of prediction is not required.

Table B.9
Losses of the deep learning estimator and sizes for training, testing, and validation datasets.

Dataset	MAE	MAPE	Size
train	0.07	1.34	705
test	0.08	1.42	79
validation	0.20	3.69	2376

Table B.10
Frechet Inception Distance for different deep learning samplers in the microfluidic generative design problem. We used 10 000 samples to evaluate the FID.

	Adversarial Auto Encoder	Variational Auto Encoder	Normalizing flows	Variational GAN
FID	**277**	308	305	407

B.2. Microfluidic deep learning sampler

To train the deep learning sampler, we collected 100 000 examples using a standard GEFEST sampler, some of which are shown in Fig. 20. Produced objects pose right-form polygons without self-intersection, intersection with other structures, and out-of-bound parts. The number of polygons within the domain varied from 1 to 7. It is worth noting that the GEFEST standard sampler has no restrictions on the number of polygons. However, in the case of a large number of the latter, such a straightforward procedure will be computationally expensive.

On the gathered dataset we trained and compared several deep generative models: Variational Auto Encoder (Kingma and Welling, 2013), Adversarial Auto Encoder (Makhzani et al., 2015), Variational Normalizing flows (Rezende and Mohamed, 2015) and Variational Generative Adversarial Network (Larsen et al., 2016). In the inference (or sample creation) mode all mentioned deep learning samplers are based on the architecture (Fig. 6). For these models, we constructed the same backbone: six convolutional layers with batch normalization and ReLU activation with the exception of the last one where the tanh function was used. The total number of parameters was generally about 4M, the accurate value depends on the specific model.

The key criteria in the selection of a generative neural network are diversity, quality of samples, and speed of inference. The latter turned out to be equal for outlined models due to an identical sampling procedure (Fig. 6). The diversity and quality of samples can be estimated using Frechet Inception Distance. The results are presented in Table B.10. It is evident that the best performance was shown by the Adversarial Auto Encoder. The AAE-produced samples are demonstrated in Fig. 17. Based on calculated FID values, we decided on AAE as a deep learning sampler in the microfluidic problem. Therefore, we give a more detailed description of this method below

Adversarial Auto Encoder (Makhzani et al., 2015) is a generative model that combines two of the most well-known generative models: GAN (Goodfellow et al., 2014) and VAE (Kingma and Welling, 2013). The main idea is to replace the regularization term in the VAE, i.e. Kullback–Leibler divergence between the posterior distribution, $p(z|x)$ (where z — the latent variable and x — the sample from the dataset), and prior distribution, $p(z)$ (this is usually taken as a normal distribution), to adversarial loss from GAN. In simple words, instead of KL-divergence, we have a discriminator that should distinguish the real latent variable, $z \sim p(z)$, and latent variable from the posterior, $z \sim p(z|x)$. This modification allows choosing more diverse types of posterior distributions than in the classical VAE.

B.3. Deep learning estimator for heat-source systems

We compared several deep learning models that predict the average temperature in a device. The results are presented in Table B.11. Despite the fact that the DenseNet demonstrates the best performance in terms of MAE loss, we decided to choose the EfficientNet model (Tan

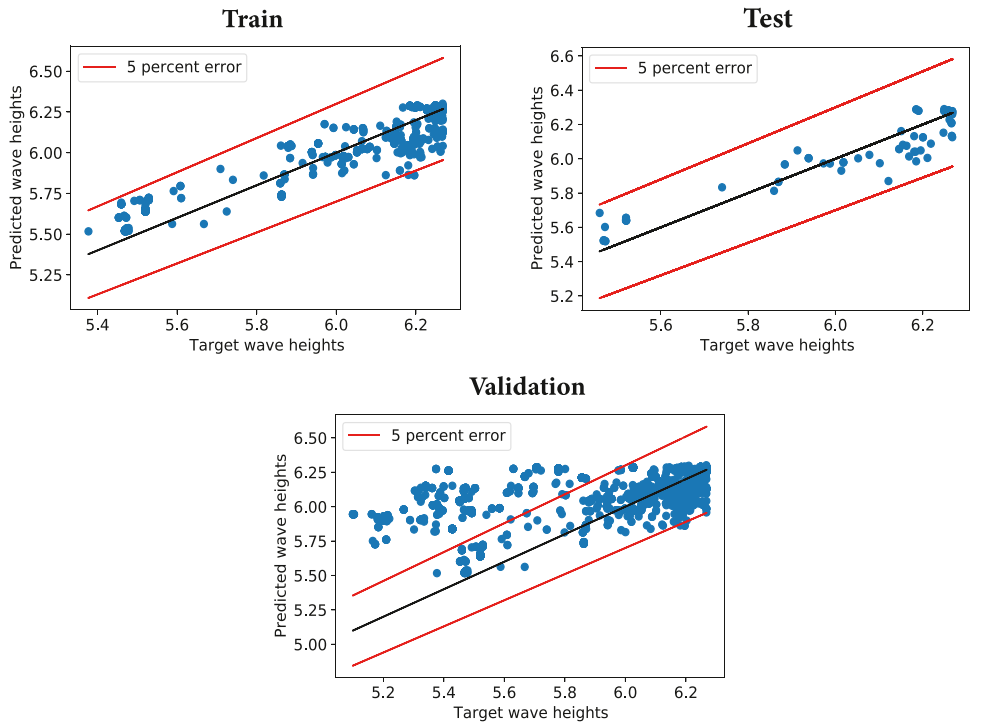


Fig. B.30. Correlation plots between predicted wave heights and simulated wave heights for training, test, and validation samples.

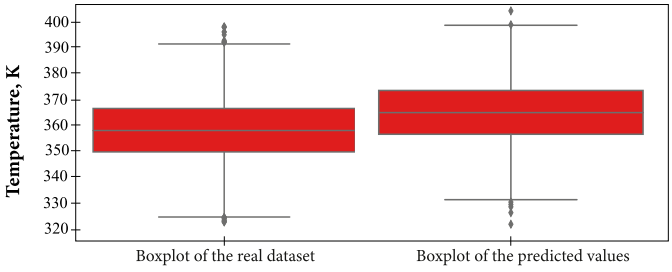


Fig. B.31. Boxplots for the dataset values and predictions of EfficientNetB0.

Table B.11
Losses of various deep learning models in the problem of heat sources. We divided the initial dataset into a train (80%) and test (20%) datasets. The number of the training epoch was equal to 10.

Model	Train MAE	Test MAE	N. params, M.
ResNet18	5.09	4.70	11.8
EffNetB0	5.03	4.63	7.2
VGG16	5.07	4.54	138
MobileNetV2	4.85	4.72	3.4
DenseNet	4.58	**4.37**	25.6

and Le, 2019) as a deep learning estimator. The main reason is that it has a small number of parameters and provides fast inference mode.

To compare statistics between the real dataset and predictions of the EfficientNet model, we created two boxplots (see Fig. B.31).

References

Bendsøe, M.P., 1989. Optimal shape design as a material distribution problem. Struct. Optim. 1 (4), 193–202.

Bendsøe, M.P., Kikuchi, N., 1988. Generating optimal topologies in structural design using a homogenization method. Comput. Methods Appl. Mech. Engrg. 71 (2), 197–224.

Buonamici, F., Carfagni, M., Furferi, R., Volpe, Y., Governi, L., 2020. Generative design: an explorative study. Comput.-Aided Des. Appl. 18 (1), 144–155.

Canny, J., 1986. A computational approach to edge detection. IEEE Trans. Pattern Anal. Mach. Intell. (6), 679–698.

Chen, X., Chen, X., Zhou, W., Zhang, J., Yao, W., 2020. The heat source layout optimization using deep learning surrogate modeling. Struct. Multidiscip. Optim. 62 (6), 3127–3148.

Chen, X., Gong, Z., Zhao, X., Zhou, W., Yao, W., 2021a. A machine learning modelling benchmark for temperature field reconstruction of heat-source systems. arXiv preprint arXiv:2108.08298.

Chen, X., Zhao, X., Gong, Z., Zhang, J., Zhou, W., Chen, X., Yao, W., 2021b. A deep neural network surrogate modeling benchmark for temperature field prediction of heat source layout. Sci. China Phys. Mech. Astron. 64 (11), 1–30.

Cheng, S., Chen, J., Qin, Q., Shi, Y., 2018. Population diversity of particle swarm optimisation algorithms for solving multimodal optimisation problems. Int. J. Comput. Sci. Eng. 17 (1), 69–79.

Danhaive, R., Mueller, C.T., 2021. Design subspace learning: Structural design space exploration using performance-conditioned generative modeling. Autom. Constr. 127, 103664.

Deshpande, A., Patavardhan, P., Estrela, V.V., Razmjooy, N., 2020. Deep learning as an alternative to super-resolution imaging in UAV systems. Imaging Sens. Unmanned Aircr. Syst. 2, 9.

Djordjevic, V., Stojanovic, V., Tao, H., Song, X., He, S., Gao, W., 2022. Data-driven control of hydraulic servo actuator based on adaptive dynamic programming. Discrete Contin. Dyn. Syst. Ser. S 15 (7), 1633.

Elchahal, G., Younes, R., Lafon, P., 2013. Optimization of coastal structures: Application on detached breakwaters in ports. Ocean Eng. 63, 35–43.

Gillies, S., et al., 2007. Shapely: manipulation and analysis of geometric objects. URL https://github.com/shapely/shapely.

González-Gorbeña, E., Qassim, R.Y., Rosman, P.C., 2016. Optimisation of hydrokinetic turbine array layouts via surrogate modelling. Renew. Energy 93, 45–57.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial nets. Adv. Neural Inf. Process. Syst. 27.

Grigorev, G.V., Nikitin, N.O., Hvatov, A., Kalyuzhnaya, A.V., Lebedev, A.V., Wang, X., Qian, X., Maksimov, G.V., Lin, L., 2022. Single red blood cell hydrodynamic traps via the generative design. Micromachines 13 (3), 367.

Harding, J., 2016. Dimensionality reduction for parametric design exploration. Adv. Archit. Geom. 274–286.

Hu, A., Razmjooy, N., 2021. Brain tumor diagnosis based on metaheuristics and deep learning. Int. J. Imaging Syst. Technol. 31 (2), 657–669.

James, S.C., Zhang, Y., O'Donncha, F., 2018. A machine learning framework to forecast wave conditions. Coast. Eng. 137, 1–10.

Jebara, T., 2012. Machine Learning: Discriminative and Generative. vol. 755, Springer Science & Business Media.

Jesmani, M., Jafarpour, B., Bellout, M.C., Foss, B., 2020. A reduced random sampling strategy for fast robust well placement optimization. J. Pet. Sci. Eng. 184, 106414.

Kallioras, N.A., Lagaros, N.D., 2020. DzAIN: Deep learning based generative design. Procedia Manuf. 44, 591–598.

Khan, A., Sohail, A., Zahoora, U., Qureshi, A.S., 2020. A survey of the recent architectures of deep convolutional neural networks. Artif. Intell. Rev. 53 (8), 5455–5516.

Kingma, D.P., Welling, M., 2013. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114.

Larsen, A.B.L., Sønderby, S.K., Larochelle, H., Winther, O., 2016. Autoencoding beyond pixels using a learned similarity metric. In: International Conference on Machine Learning. PMLR, pp. 1558–1566.

Liu, X., Zhao, W., Wan, D., 2022. Multi-fidelity Co-Kriging surrogate model for ship hull form optimization. Ocean Eng. 243, 110239.

Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., Frey, B., 2015. Adversarial autoencoders. arXiv preprint arXiv:1511.05644.

Man, Y., Kucukal, E., An, R., Watson, Q.D., Bosch, J., Zimmerman, P.A., Little, J.A., Gurkan, U.A., 2020. Microfluidic assessment of red blood cell mediated microvascular occlusion. Lab. Chip 20 (12), 2086–2099.

Minton, J., 2012. A comparison of common methods for optimal well placement. Research rep., University of Auckland.

Mukkavaara, J., Sandberg, M., 2020. Architectural design exploration using generative design: framework development and case study of a residential block. Buildings 10 (11), 201.

Ng, A., Jordan, M., 2001. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. Adv. Neural Inf. Process. Syst. 14.

Nikitin, N.O., Hvatov, A., Polonskaia, I.S., Kalyuzhnaya, A.V., Grigorev, G.V., Wang, X., Qian, X., 2021. Generative design of microfluidic channel geometry using evolutionary approach. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 59–60.

Nikitin, N.O., Polonskaia, I.S., Kalyuzhnaya, A.V., Boukhanovsky, A.V., 2020. The multi-objective optimisation of breakwaters using evolutionary approach. arXiv preprint arXiv:2004.03010.

Oh, S., Jung, Y., Kim, S., Lee, I., Kang, N., 2019. Deep generative design: Integration of topology optimization and generative models. J. Mech. Des. 141 (11).

Palar, P.S., Liem, R.P., Zuhal, L.R., Shimoyama, K., 2019. On the use of surrogate models in engineering design optimization and exploration: The key issues. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 1592–1602.

Palmer, T., Williams, P.D., 2008. Introduction. Stochastic physics and climate modelling.

Qian, C., Tan, R.K., Ye, W., 2022. An adaptive artificial neural network-based generative design method for layout designs. Int. J. Heat Mass Transfer 184, 122313.

Ramezani, M., Bahmanyar, D., Razmjooy, N., 2021. A new improved model of marine predator algorithm for optimization problems. Arab. J. Sci. Eng. 46 (9), 8803–8826.

Rezende, D., Mohamed, S., 2015. Variational inference with normalizing flows. In: International Conference on Machine Learning. PMLR, pp. 1530–1538.

Shea, K., Aish, R., Gourtovaia, M., 2005. Towards integrated performance-driven generative design tools. Autom. Constr. 14 (2), 253–264.

Shen, Q., Vahdatikhaki, F., Voordijk, H., van der Gucht, J., van der Meer, L., 2022. Metamodel-based generative design of wind turbine foundations. Autom. Constr. 138, 104233.

Singh, V., Gu, N., 2012. Towards an integrated generative design framework. Des. Stud. 33 (2), 185–207.

Song, S., Jin, H., Yang, Q., 2021. Performance analysis of different operators in genetic algorithm for solving continuous and discrete optimization problems. In: Filipe, J., Smialek, M., Brodsky, A., Hammoudi, S. (Eds.), Proceedings of the 23rd International Conference on Enterprise Information Systems, ICEIS 2021, Online Streaming, April 26-28, 2021, Volume 1. SCITEPRESS, pp. 536–547. http://dx.doi.org/10.5220/0010494005360547.

Steinbuch, R., 2010. Successful application of evolutionary algorithms in engineering design. J. Bionic Eng. 7, S199–S211.

Sun, J., Zheng, X., Yao, W., Zhang, X., Zhou, W., 2022. Heat source layout optimization using automatic deep learning surrogate model and multimodal neighborhood search algorithm. arXiv preprint arXiv:2205.07812.

Tan, M., Le, Q., 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In: International Conference on Machine Learning. PMLR, pp. 6105–6114.

Tan, R.K., Zhang, N.L., Ye, W., 2020. A deep learning–based method for the design of microstructural materials. Struct. Multidiscip. Optim. 61 (4), 1417–1438.

Tian, X., Sun, X., Liu, G., Deng, W., Wang, H., Li, Z., Li, D., 2022. Optimization design of the jacket support structure for offshore wind turbine using topology optimization method. Ocean Eng. 243, 110084.

Tian, Q., Wu, Y., Ren, X., Razmjooy, N., 2021. A new optimized sequential method for lung tumor diagnosis based on deep learning and converged search and rescue algorithm. Biomed. Signal Process. Control 68, 102761.

Tukur, A.D., Nzerem, P., Nsan, N., Okafor, I.S., Gimba, A., Ogolo, O., Oluwaseun, A., Andrew, O., 2019. Well placement optimization using simulated annealing and genetic algorithm. In: SPE Nigeria Annual International Conference and Exhibition. OnePetro.

Tyflopoulos, E., Tollnes, F.D., Steinert, M., Olsen, A., et al., 2018. State of the art of generative design and topology optimization and potential research needs. In: DS 91: Proceedings of NordDesign 2018, Linköping, Sweden, 14th–17th August 2018.

Vajna, S., Clement, S., Jordan, A., Bercsey, T., 2005. The autogenetic design theory: an evolutionary view of the design process. J. Eng. Des. 16 (4), 423–440.

Vlah, D., Žavbi, R., Vukašinović, N., 2020. Evaluation of topology optimization and generative design tools as support for conceptual design. In: Proceedings of the Design Society: DESIGN Conference. 1, Cambridge University Press, pp. 451–460.

Xu, Y., Cai, Y., Sun, T., Tan, Q., Sun, J., Peng, J., et al., 2021a. Ecological preservation based multi-objective optimization of coastal seawall engineering structures. J. Clean. Prod. 296, 126515.

Xu, Z., Li, X., Stojanovic, V., 2021b. Exponential stability of nonlinear state-dependent delayed impulsive systems with applications. Nonlinear Anal. Hybrid Syst. 42, 101088.

Xue, Y., Jiang, P., Neri, F., Liang, J., 2021a. A multi-objective evolutionary approach based on graph-in-graph for neural architecture search of convolutional neural networks. Int. J. Neural Syst. 31 (09), 2150035.

Xue, Y., Wang, Y., Liang, J., Slowik, A., 2021b. A self-adaptive mutation neural architecture search algorithm based on blocks. IEEE Comput. Intell. Mag. 16 (3), 67–78.

Yin, Z., Razmjooy, N., 2020. PEMFC identification using deep learning developed by improved deer hunting optimization algorithm. Int. J. Power Energy Syst. 40 (2), 189–203.

Zheng, H., Yuan, P.F., 2021. A generative architectural and urban design method through artificial neural networks. Build. Environ. 205, 108178.

Zielinski, P.A., 1991. On the meaning of randomness in stochastic environmental models. Water Resour. Res. 27 (7), 1607–1611.

Zitzler, E., Laumanns, M., Bleuler, S., 2004. A tutorial on evolutionary multiobjective optimization. Metaheuristics Multiobjective Optimisat. 3–37.

Zitzler, E., Laumanns, M., Thiele, L., 2001. SPEA2: Improving the strength Pareto evolutionary algorithm. TIK-Rep. 103.

Zou, A., Chuan, R., Qian, F., Zhang, W., Wang, Q., Zhao, C., 2022. Topology optimization for a water-cooled heat sink in micro-electronics based on Pareto frontier. Appl. Therm. Eng. 207, 118128.