Computer-Aided Design 189 (2025) 103945

Contents lists available at ScienceDirect

Computer-Aided Design

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

Research Paper

AIXD: AI-eXtended Design Toolbox for data-driven and inverse design

Alessandro Maissen [a,1], Aleksandra Anna Apolinarska [b,1], Sophia V. Kuhn [c,1], Luis Salamanca [a,1,*], Michael A. Kraus [c,d], Konstantinos Tatsis [a], Gonzalo Casas [b], Rafael Bischof [a], Romana Rust [b], Walter Kaufmann [c], Fernando Pérez-Cruz [a], Matthias Kohler [b]

[a] Swiss Data Science Center, ETH Zürich, Zürich, Andreasturm, Andreasstrasse 5, 8092, Switzerland
[b] Gramazio Kohler Research, ETH Zürich, Zürich, HIB E43, Stefano-Franscini-Platz 1, 8093, Switzerland
[c] kfmResearch, Institut für Baustatik und Konstruktion, ETH Zürich, Zürich, Stefano-Franscini-Platz 5, 8093, Switzerland
[d] Professur Baustatik, Institut für Statik und Konstruktion, TU Darmstadt, Darmstadt, Franziska-Braun-Straße 3, 64287, Germany

ARTICLE INFO

MSC:
68T01
68U07
62P30
68N01

Keywords:
Artificial intelligence
Generative design
Parametric modeling
Architecture
Civil engineering
Computer-aided design

ABSTRACT

Design processes, in many disciplines like architecture, civil engineering or mechanical engineering, involve navigating large, high-dimensional and heterogeneous data. While AI-driven approaches like inverse design and surrogate modeling can enhance design exploration, their adoption is hindered by complex workflows and the need for coding and machine learning expertise. To address this, we introduce AI-eXtended Design (AIXD): a low-code, open-source toolbox that integrates AI into computational design. AIXD simplifies handling of mixed data types, as well as the analysis, training, and deployment of machine learning models for inverse design, surrogate modeling, and sensitivity analysis, enabling domain experts to rapidly explore diverse solutions with minimal coding. In this paper, we show the functionalities of the toolbox, and we demonstrate AIXD's capabilities in architectural and engineering design applications, showing how it accelerates performance evaluation, generates high-performing alternatives, and improves design understanding by delivering new insights. By bridging AI and design practice, AIXD lowers the entry barrier to data-driven methods, making AI-extended design more accessible and efficient.

1. Introduction

1.1. Motivation

In construction and manufacturing industries like architecture, civil engineering, product design, mechanical engineering, design is oftentimes a complex and time-consuming process. The design team needs to negotiate several competing objectives, adjust many parameters, observe multiple constraints and satisfy criteria that are often difficult to quantify. Conventionally, architects and engineers rely on their experience and iterative refinement to craft a working solution [1,2]. Computer-aided design methods provide practical means to assist the design. For example, parametric modeling is popularly used to create and evaluate many design variants. However, finding satisfactory designs by adjusting design parameters may be tedious. In some cases, sensitivity analysis can provide insights about the contribution of individual parameters to the design outcome and thus inform the refinement of the design parameters [2]. Nevertheless, this process only allows to refine solutions in the vicinity of the sought-after designs. Additionally, while optimization is often used to find the best solutions, it is inherently constrained by the definition of the objective function, which may not fully capture the complete set of design goals. Consequently, optimization facilitates solution finding, but does not necessarily yield optimal solutions for the entire scope of the design problem, nor does it contribute to a better understanding of a complex design problem [3]. All these approaches often deliver a small set of possible answers, discarding many other valuable solutions and leaving most of the usually large and complex design space unexplored. Also, in many design problems, a single pass through the parametric model can be time-consuming due to computationally expensive simulations, which limits the number of design candidates affordable for consideration by the designer or for evaluation in optimization or search procedures. Surrogate modeling is an approach to alleviate this limitation by creating simpler and faster models that approximate the original design model [4]. Finally, recent machine and deep learning approaches allow for novel ways to cope with inverse design as a paradigm in the Architecture, Engineering and Construction (AEC) in-

* Corresponding author.
E-mail address: luis.salamanca@sdsc.ethz.ch (L. Salamanca).
[1] Authors contributed equally.

https://doi.org/10.1016/j.cad.2025.103945
Received 21 May 2025; Received in revised form 21 July 2025; Accepted 10 August 2025
Available online 28 August 2025

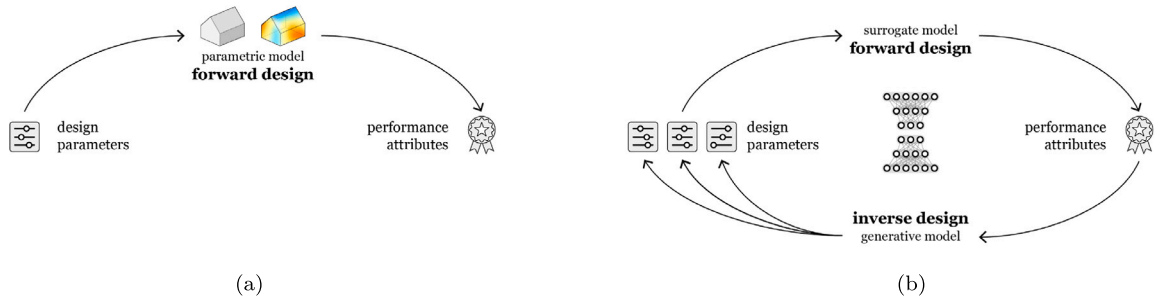


Fig. 1. Schematic representation of a parametric modeling paradigm (a) and of a forward and inverse machine learning models (b).

dustry, which facilitates design exploration and discovery of (multiple) candidate solutions that match the specified target requirements.

AIXD, the software tool presented in this paper, extends the parametric design approach in multiple ways. It helps to augment design exploration, expedite the discovery of interesting solutions, and enhance the designer's understanding of the design problem.

1.2. Forward and inverse design

In many disciplines and projects, design tasks can be formulated in terms of input parameters that can be varied and outputs by which the design is assessed. In the context of this work, design tasks are viewed as a mapping between input variables, which we call *design parameters*, and output (target) variables, which we call *performance attributes*. Parametric design models often naturally represent such a mapping. They encapsulate the design process into a sequence of procedures (step-by-step modeling of the geometry, running a simulation, evaluation, etc.) based on some input (numerical variables, setout geometry, etc.) and producing some output (the resulting 3D geometric model, computed performance values etc.). A mapping in this direction (from design parameters to performance attributes) is often referred to as the *forward model*. In design, this mapping is typically (analytically) intractable and not bijective, i.e., there may exist different designs with the same performance, and hence it cannot be easily inverted (see Fig. 1).

In contrast to forward design, an *inverse model* maps in the opposite direction than the parametric model. In an inverse design paradigm, one can instead specify values for the required performance attributes and obtain multiple design instances that fulfill them. This is particularly valuable for design exploration, when the design space is large and valid solutions are scattered in the design parameter space. Presented with multiple and diverse solutions that are equivalent in terms of the requested performance metrics, the user can further refine their design choices based on other criteria that are not captured by the model — be it because they are not quantifiable (e.g. aesthetics, construction heritage), are easier to assess by a human than to express algorithmically (e.g. durability, functionality), are costly to compute (e.g. entail expensive Finite Element Modeling, FEM, or Computer Fluid Dynamics, CFD, simulations) or are not integrated computationally in the model (e.g. constructability).

Machine learning methods have proven capable of learning the inverse mapping from data. Applications of machine learning for inverse design are intensely studied in many research fields such as pharmaceutical [5,6] and material sciences [7–9] to discover compounds with desired structures and properties, as well as structural, architectural [10] and mechanical engineering [11,12] to find valid design alternatives.

1.3. Machine learning for design problems

Forward and inverse design problems can be approximated by multiple types of neural networks depending on the nature of the design task and its representation in terms of inputs and outputs. In our context, forward design is akin to a regression or classification problem and can be tackled with discriminative methods that predict target values given some input, e.g. a multilayer perception (MLP). Inverse design, in turn, requires a generative model (e.g. a generative adversarial network, diffusion model, etc.), conditioned on the given performance values. In this work, we rely on conditional autoencoder architectures in which the encoder, in addition of obtaining a latent representation of the input, learns a surrogate model of the parametric model; and the decoder acts as the generative model for the inverse mapping.

Despite the availability of high-level libraries like Keras that are based on low-level libraries such as Tensorflow or PyTorch, deploying and training neural network architectures remains challenging for users unfamiliar with deep learning. Furthermore, for machine learning methods to truly complement and benefit the design process, the entire pipeline – which includes dataset acquisition, model setup, training, and deployment – needs to be well integrated into the design workflow and its computational design toolchain.

Even though many designs that can be formulated as parametric problems can greatly benefit from AI, the adoption of these methodologies remains limited in the aforementioned domains and industries.

Therefore, to bridge the gap between AI and design practice, we developed AIXD: a low-code toolbox that simplifies training and deploying AI models within computational design workflows. AIXD provides high-level functions for inverse design and surrogate modeling, tools for data exploration, to get insights about the design problem, data handling, and many more.

2. AIXD Toolbox

2.1. Introduction

AIXD is an open-source toolkit for data-driven forward and inverse design and is tailored to design-related domains like architecture, civil engineering, product design and mechanical engineering. With few high-level commands, designers can harness their parametric models to generate project-specific datasets, or import data generated externally. This data is used then to train custom machine learning models, and finally to deploy them for the desired downstream tasks like design exploration, sensitivity analysis or data visualization. It follows a simple few-steps workflow, generalized from [1,2,13], which can be re-used for different projects with little adjustment.

The main features of AIXD are:

- **Inverse design** using autoencoder models to generate design instances that meet requested performance constraints, to expedite design exploration.
- **Surrogate modeling** using neural networks for forward design, rapid evaluation and optimization.
- **Sensitivity analysis** obtained by leveraging gradients of the trained model to identify key variables, support data-driven decision making and navigate design complexity.

- **Visual data exploration** to get insights from the data about the design task, and to enhance the user's understanding of the design problem.
- The ability to leverage **parametric models** to acquire datasets for training and to provide ground truth data.
- Intuitive handling of **data** and **variables** reflecting the design logic (as design parameters and performance attributes).
- Simple and **low-code** workflow to easily train, evaluate and deploy (project-tailored) models.
- **Versatility**, as it is applicable to many design problems across diverse domains.

The toolkit has been developed for domain experts, including designers, architects, engineers who do not have extensive expertise in machine learning and requires only basic skills in Python programming. Furthermore, AIXD comes with detailed documentation, including tutorials and examples, helping advanced users to build more specialized solutions and extend the functionalities of the toolbox: https://aixd.ethz.ch. With AIXD, we aim to bridge the current gap between CAD, physical simulators and powerful machine learning (ML) solutions for AI-assisted design, and facilitate their usage by bringing it closer to practitioners and fostering its application in real-life projects.

The AIXD is available under an MIT license here: https://gitlab.renkulab.io/ai-augmented-design/aixd

2.2. Workflow

AIXD presents a simple workflow, depicted in Fig. 2, that consists of the following stages:

1. **Problem definition**: Define all the design parameters and performance attributes that describe the characteristics of the design problem (e.g., as defined in the parametric model).
2. **Data collection**: Import an existing dataset or harness the parametric model to generate a collection of design samples.
3. **ML model**: Set up and automatically build an ML-model tailored to the design problem, and train it.
4. **Design exploration**: Leverage the trained model for accelerated forward design, inverse design or sensitivity analysis.
5. **Visualization**: Throughout the workflow, use different visualization tools to explore the data, assess the ML model's training and evaluate generation results.

The following section presents the key functionalities from the point of view of the user. Section 3 provides further details on the methods and the implementation.

2.3. Problem definition and data collection

2.3.1. Variables

In AIXD, every project (design task) is defined as a mapping between the input variables called *design parameters* (DP), and the output variables called *performance attributes* (PA), and which typically correspond to the inputs and outputs of the parametric design model, respectively.

Variables in AIXD are represented by `DataObjects`, which can be declared given a type, name, domain and dimensionality. In all further steps in a project, variables can be referred by the given variable names in an intuitive, high-level manner. AIXD offers four data types: real-valued numbers (as `DataReal` class), discrete numbers (integers, as `DataInt`), boolean (true/false, as `DataBool`) or categorical (labels, as `DataCategorical`).

Designs often entail many different representations such as NURBS geometry, meshes, images, point clouds etc. In CAD software, parametric models oftentimes use such high-level objects as inputs and outputs – for example, referenced setout geometry as input, resulting geometry as output, simulation results formatted in software-specific data structures as performance metrics. For use with AIXD, such objects need to be represented or parameterized by (lists of) numerical and categorical data types – for example, a curve as the coordinates of its control points, a mesh as its vertices and faces. Also, the variables must preserve the data structure in terms of data types and dimensions throughout the project and across the dataset.[2]

2.3.2. Data structure

Variables (`DataObjects`) are grouped into the `DataBlocks` to streamline data handling, transformations, pre- and post-processing. `DesignParameters` and `PerformanceAttributes` are two pre-defined data blocks that bundle variables for tasks related to the dataset. In later stages, another two pre-defined data blocks (`InputML` and `OutputML`) bundle variables for tasks related to the machine learning models.

Finally, a `Dataset` object encapsulates the schema and the settings of the project, and manages all data-related tasks in subsequent steps (see example in Code 1).

```
# Definition of Data Objects
x1 = DataCategorical(name="x1", dim=1, domain=["a","b","c"]),
x2 = DataReal(name="x2", dim=3, domain=Interval(−5, 5))]
y1 = DataBool(name="y1", dim=2)

# Grouping of Data Objects into Data Blocks
dp = DesignParameters(name="design_par", dobj_list=[x1, x2]])
pa = PerformanceAttributes(name="perf_att", dobj_list=[y1]])

# Creation of Dataset instance
dataset = Dataset(name="toy_example", design_par=dp, perf_attributes=pa)
```

Code Listing 1: Definition of parametric model and `Dataset` creation.

2.3.3. Data collection

AIXD separates the dataset into the definition of a `Dataset` object, and into data samples stored in a pre-defined folder structure and sharded into multiple files for scalability purposes.

Already existing data can be easily imported from an external source (e.g., from a CSV file, a dataframe object) – AIXD will reformat it and save it into its specific representation and organization. If the parametric design model is available, AIXD can also generate data samples, first by using its in-built methods to sample the space of the design parameters, according to their type and domain, and then by obtaining the corresponding values of the performance attributes from the parametric model hooked up as a callback.[3] In this scenario, the parametric design model serves also as the ground-truth model of the design/data.

2.3.4. Automated data pre- and post-processing

Throughout a project, data undergoes multiple transformations. The toolbox supports standard transformations such as standardization (i.e., z-score normalization), min–max scaling, log and sigmoid transformation and label encoding, and throughout the workflow applies typical defaults based on the data type. Users can also override these defaults for selected variables, and instead specify another of the provided transformations, or extend the base class `DataObjectTransformation` to create a custom transformation effortlessly.

Also fully automated and hidden from the user by AIXD are various "technical" conversions which are inevitably required throughout the

[2] By the nature of the machine learning models used in AIXD, variable-length inputs (e.g. graphs) currently cannot be accommodated.

[3] This workflow is possible if either the parametric model can be called as a Python method, e.g., using a CAD/CAE software's API.

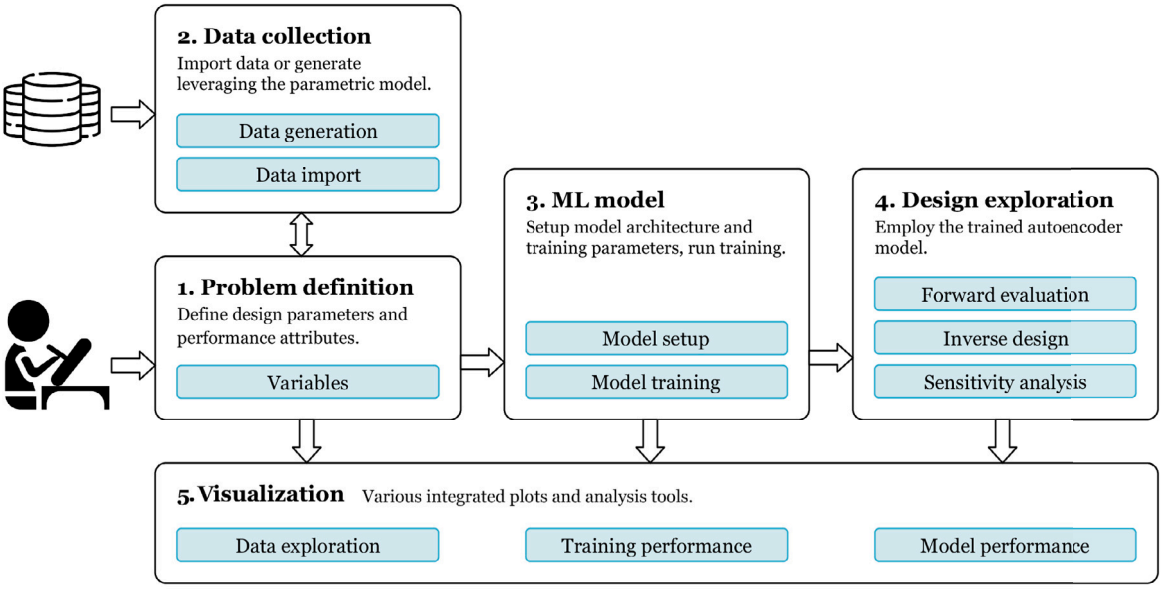


Fig. 2. Overview of the full workflow, with reference to the different steps.

pipeline to comply with different libraries, for example, converting between `numpy` arrays, `pandas` data frames, `torch` tensor formats, and between nested and flattened tables of multi-dimensional data.

2.3.5. Data exploration

Data can be a valuable source of insights about the design problem. For this, AIXD provides a `Plotter` class with many pre-configured figure templates to plot, for example, the correlation matrix or their marginal and joint distributions. These graphics may help the user to understand, for example, which combinations of performance attributes are feasible according to the collected data (see example in Fig. 3).

All visualizations can be easily set up by specifying the names of the selected variables or blocks — the figures are automatically created according to given data types and value ranges, with all necessary data transformations handled internally.

```
from aixd.visualisation.plotter import Plotter
plotter = Plotter(dataset, output="show")
plotter.contours2d(attributes=["x2", "y1"])
```

Code Listing 2: Initialization and example utilization of the plotter object.

2.4. Machine learning model

2.4.1. Automatic setup

The major hurdle for applying machine learning methodologies is creating a suitable model tailored to the design problem at hand. Dimensions and data types of the variables determine the model size and dictate the loss functions used during training. Parametric models, and the design tasks, undergo multiple iterations during a design process which may result in changes to their inputs, outputs and variables involved.

AIXD provides two types of machine learning models: a conditional autoencoder (`CondAEModel`) and a conditional variational autoencoder (`CondVAEModel`). One key feature of these classes is that the model architecture is derived automatically from the project's variables (captured in the `Dataset` class), given just a few high-level settings, making the setup of the machine learning model fast and simple. Also, the data is split automatically into training and validation sets (80:20%), loaded and preprocessed by a `DataModule` built on top of a similar class in PyTorch Lightning. Tedious steps like constructing the network architecture (e.g., layer types, activations), data transformations (e.g., scaling), losses (e.g., mean squared error, cross-entropy) are set up automatically using best-practice defaults or the overrides specified in the variables. The technical and theoretical details of these models are explained in Section 3.1.1.

To set up an autoencoder, the user specifies the input and target variables for the model. Typically these correspond to (a subset of) design parameters on the one side and performance attributes on the other, which become vectors $\mathbf{x}$ and $\mathbf{y}$, respectively, after the necessary transformations (e.g. one-hot encoding) and normalizations. One can also assign certain design parameters to $\mathbf{y}$ in order to be able to constrain the generated design in the request.

The remaining user-defined settings are the number and widths of the hidden layers, the dimension of the latent space and the batch size. They may have a notable effect on the performance of the trained model and require empirical testing to find the optimal configuration. The toolbox's documentation provides rules-of-thumb for the novice users to get started.

```
# Assume you already defined the parametric model and populated
# the dataset with data.
dataset = Dataset(...)

# Create the datamodule
datamodule = DataModule.from_dataset(dataset, batch_size=32)

# Create the ML-Model, here a CAE.
cae = CondAEModel.from_datamodule(datamodule,
                                  layer_widths=[128, 64, 32, 16],
                                  latent_dim=2)

# Train the model
cae.fit(datamodule, max_epochs=10)
```

Code Listing 3: Model setup and training

2.4.2. Model training and optimization

To track the training process, and to compare different runs and model configurations, AIXD logs the dynamics of individual losses

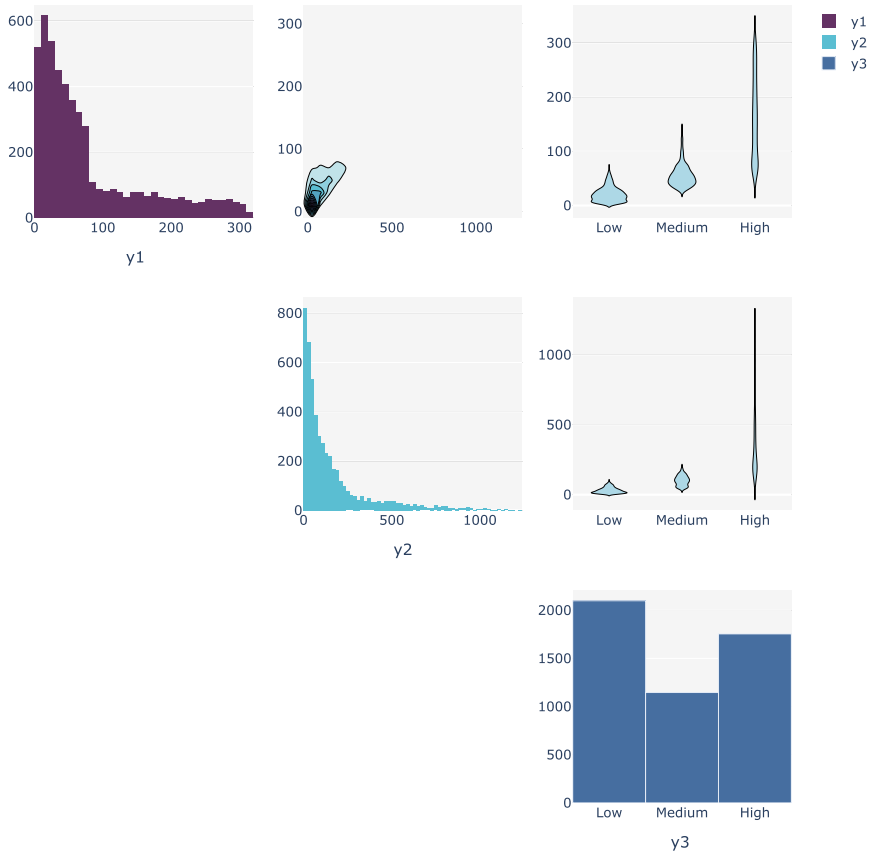


Fig. 3. AIXD Plotter's visualization of the univariate and joint distributions of mixed-type data. Depending on the data type, the suitable plot type is applied automatically. In this toy example, y_1 and y_2 are real-valued variables and y_3 is a categorical variable.

(e.g. validation loss for the reconstruction of **x** and **y**) and, to visualize them, provides integration with the online service Weights &Biases [14]. Checkpointing is provided by the underlying PyTorch Lightning framework.

2.4.3. Evaluation of the trained models

In addition to loss tracking through Weights &Biases, AIXD offers various visualizations to support assessment of the model performance in terms of prediction accuracy or generation quality. This enables the designer to decide if the performance is sufficient to deploy the model for design tasks or if the model's hyperparameters require further tuning.

For example, a scatter plot in Fig. 4(a) compares true vs predicted values of a given performance attribute, revealing the prediction accuracy across its value range. Similarly, in Fig. 4(b) we compare the prediction performance for a categorical value, presenting the results in the form of a confusion matrix. The summary plot in Fig. 4(c) provides an even more concise view and an intuitive depiction of the prediction quality of **y**, helping to assess the encoder's performance as a forward model. The quality of the generator can be evaluated by observing the difference between the requested and predicted values of a performance attribute across its range, and whether this error can be explained by the data density through an overlay with the histogram of the training data, as shown in Fig. 4(d).

Additional guidelines are provided in the documentation, as well as an example where Weights & Biases is used to perform hyperparameter tuning.

2.5. Design exploration

2.5.1. Surrogate models: Accelerated forward evaluation

After training, the encoder can be used as a fast surrogate model of the parametric design model. The user provides a new set of design parameter values, and the model returns the corresponding predicted performance attributes, in the original domain, which can be readily understood and compared to the ground truth output of the parametric model. The toolbox handles the automatic normalization of the input values, and unnormalization of the predicted target values.

2.5.2. Inverse design: Generative AI

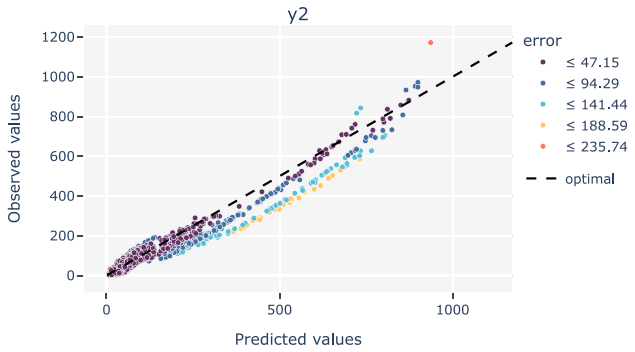
To use the trained model for inverse design, the user requests the desired values of performance attributes and obtains n sets of generated design parameters. The request can be formulated simply by specifying the name(s) of the variable(s) and the desired value(s) - either an exact value or a value range. The request may be underspecified [1], i.e., the user may refrain from providing values for some variables, allowing them to be freely chosen by the model. This may be helpful to explore the design space by focusing on selected aspects in isolation.

In general, the generation process runs swiftly, delivering results in a few seconds, enabling an efficient interaction. The AIXD generator hence works as a "co-pilot" for the designer, boosting the design process efficiency and the more exhaustive exploration of the solution space.

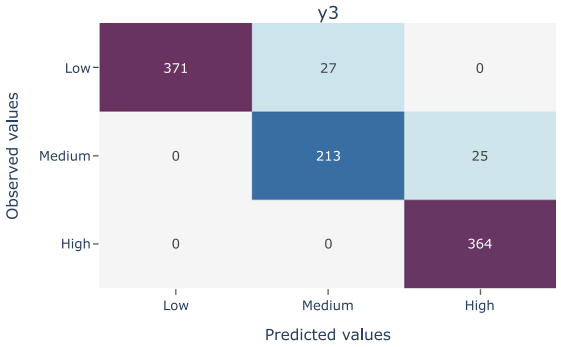
2.5.3. Sensitivity analysis

The sensitivity analysis is offered as an additional tool for understanding the solution space by further providing insights into the relations between design parameters and performance attributes, i.e., the quantification of the effect on **y** of changing **x** via gradients at a specific point in the parameter vector space.

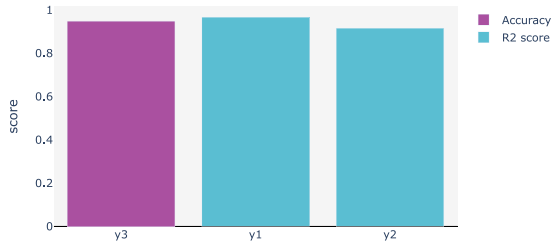
Since neural networks are differentiable, they can be used for sensitivity analysis. Both types of sensitivity analysis, local and global, are performed by leveraging the gradients of the trained model. As such, the forward regressor model is used to calculate the gradients of



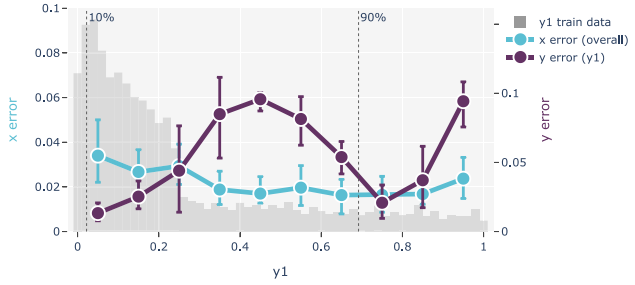
(a) Observed-vs-predicted plot for continuous variables.



(b) Observed-vs-predicted plot for categorical variables.



(c) Summarized scores of the performance attributes.



(d) Evaluation errors for a selected performance attribute.

Fig. 4. Examples of automated plots in AIXD, which help understand the performance of the trained model, demonstration on a toy example from Fig. 3: (a) scatter plot of real vs. predicted values for a selected real-valued performance attribute y_2, (b) confusion matrix for a selected categorical performance attribute y_3, (c) summary of the prediction performance of the model: accuracy score for categorical variables and R2 score for numerical variables, and (d) reconstruction error (over all x) and prediction error for a selected performance attribute y_1, plotted over the histogram of the training set values of y_1.

performance attributes with respect to design parameters through automatic differentiation or finite differences when dealing with discrete variables.

3. Methods

In the following sections, we provide a more thorough description on the methods and techniques implemented, as well as some implementation details that will allow us to understand how AIXD operates under the hood.

3.1. Autoencoder models implemented in AIXD

AIXD adapts a conditional autoencoder (CAE) and a conditional variational autoencoder (CVAE) [15] for forward and inverse design. Both are based on the standard encoder–decoder architecture (cf. Fig. 5), augmenting it through a conditional (target) vector to control the generation process (i.e., inverse design). We will use $\mathbf{x} \in \mathbb{R}^{d_\mathbf{x}}$ to represent the input vector, $\mathbf{y} \in \mathbb{R}^{d_\mathbf{y}}$ to refer to the conditional (target) vector, and $\mathbf{z} \in \mathbb{R}^{d_\mathbf{z}}$ to denote the latent space vector.

Our adaptation of the CAE/CVAE architecture deviates slightly from the standard approach in order to seamlessly accommodate forward and inverse design. Following a similar methodology to [2,16], we introduce an additional path (referred to as a regression head) in the encoder to predict the conditional (target) variables, which is learnt through an additional prediction loss. This enables the encoder to be utilized for forward design purposes. Besides, instead of conditioning both the encoder and the decoder on $\mathbf{y}$, we solely condition the decoder on $\mathbf{y}$ as illustrated in Fig. 5.

Similar to a standard autoencoder (AE) [17], the encoder embeds the input features $\mathbf{x}$ into a latent vector $\mathbf{z}$. Its dimension is a hyperparameter that should typically be smaller than that of the input $\mathbf{x}$ to create a bottleneck, yet large enough to allow the model to encode essential features of the design task. The two models implemented in AIXD primarily differ in how this latent space is encoded and learnt. In particular, both models have terms to reconstruct $\mathbf{x}$ and predict $\mathbf{y}$, but different losses for $\mathbf{z}$. Finally, the decoder takes together the conditional vector $\mathbf{y}$ and the latent vector $\mathbf{z}$ as input and outputs $\hat{\mathbf{x}}$, a reconstruction of the input $\mathbf{x}$. Both, the encoder and decoder are parameterized by multi-layer perceptrons (MLPs) consisting of l blocks. Each block consists of a fully-connected linear layer followed by a non-linear activation function and batch normalization. The number of layers and their widths (i.e., the number of neurons) are user-defined and need to be adjusted to the complexity of the design problem.

The previous descriptions apply to both models implemented in AIXD. In the following sections we describe in more detail their peculiarities.

3.1.1. Conditional autoencoders

For the conditional autoencoder (CAE) implementation, the parameters of the encoder and decoder are optimized jointly through the following objective function:

$$\mathcal{L}_{\text{CAE}} = \underbrace{\lambda_1 \mathcal{L}_{\text{rec}}(\mathbf{x}, \hat{\mathbf{x}})}_{\text{reconstruction loss}} + \underbrace{\lambda_2 \mathcal{L}_{\text{pred}}(\mathbf{y}, \hat{\mathbf{y}})}_{\text{prediction loss}} + \underbrace{\lambda_3 \mathcal{L}_\mathbf{z}(\mathbf{z}) + \lambda_4 \mathcal{L}_{\text{cov}}(\mathbf{y}, \mathbf{z})}_{\text{regularization loss}} \tag{1}$$

Here, $\lambda_i \in \mathbb{R}$ represent hyper-parameters used to balance the different components of the losses. Based on the ideas of regularized autoencoders [18], we augment the standard autoencoder with a regularization loss to improve the generative capabilities of the model. Particularly, the loss $\mathcal{L}_\mathbf{z}(\mathbf{z})$ on the latent space $\mathbf{z}$, is defined as

$$\mathcal{L}_\mathbf{z}(\mathbf{z}) = \bar{\mathbf{z}}^2 + \left(1 - \frac{1}{d_\mathbf{z}} \sum_{i=1}^{d_\mathbf{z}} (z_i - \bar{z})^2\right)^2 \quad \text{with} \quad \bar{\mathbf{z}} = \frac{1}{d_\mathbf{z}} \sum_{i=1}^{d_\mathbf{z}} z_i \tag{2}$$

This penalizes deviations of the values in $\mathbf{z}$ from a mean of zero and a standard deviation of one, resulting in a more compact and bounded latent space leading to more meaningful generations. Similarly to the approach in [2], the second component of the regularization loss, $\mathcal{L}_{\text{cov}}(\mathbf{y}, \mathbf{z})$, aims to enforce decorrelation between latent variables $\mathbf{z}$

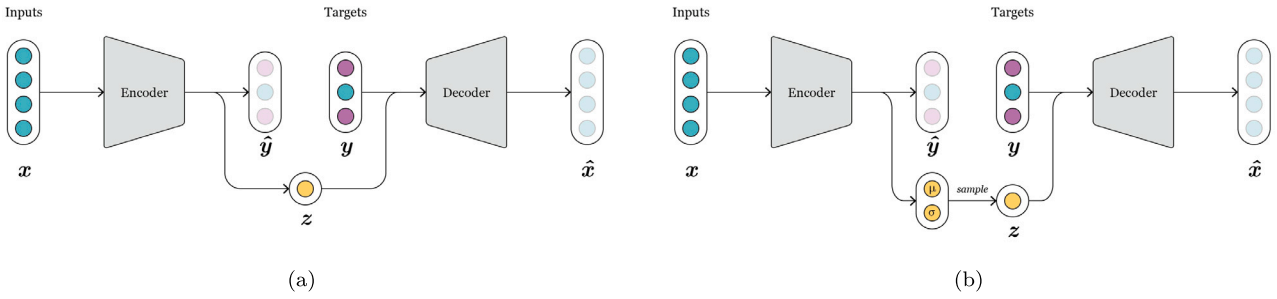


Fig. 5. Modified model architectures with regression head to predict $\mathbf{y}$, for both (a) the conditional autoencoder (CAE) and (b) the conditional variational autoencoder (CVAE).

and conditional variables $\mathbf{y}$ by minimizing their covariance. The imposed constraint helps to learn a disentangled representation, where the encoder is expected to encode all aspects except the performance attributes into the latent space $\mathbf{z}$. This, in turn, improves the controllability of the generation process. Lastly, the reconstruction loss $\mathcal{L}_{\text{rec}}(\mathbf{x}, \hat{\mathbf{x}})$ on the design parameters and the prediction loss on the performance attributes $\mathcal{L}_{\text{pred}}(\mathbf{y}, \hat{\mathbf{y}})$ both encompass the mean squared error for continuous variables and/or the cross-entropy loss for categorical variables.

3.1.2. Conditional variational autoencoder

Additionally we adopt a conditional variational autoencoder (CVAE) architecture [15,19], depicted in Fig. 5(b). This model differs from a CAE in that it incorporates a probabilistic latent space, enabling smooth and continuous representation of the data. The encoder maps the input $\mathbf{x}$ to the mean $\boldsymbol{\mu}_z \in \mathbb{R}^{d_z}$ and variance $\boldsymbol{\sigma}_z^2 \in \mathbb{R}^{d_z}$ parameterizing the multivariate Gaussian distribution of the form $\mathbf{z}|\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}_z, \text{diag}(\boldsymbol{\sigma}_z^2))$. This stands in contrast to the CAE, which maps to a specific point within the latent space. Similarly, the decoder uses a sample from the multivariate Gaussian distribution $\mathbf{z}$ along with the conditional variable $\mathbf{y}$ to reconstruct the input $\mathbf{x}$. The remaining components of the CVAE, i.e., the architecture of the parameterized encoder and decoder networks, remain the same. Nonetheless, this results in a slightly different objective function compared to Eq. (1):

$$\begin{aligned}\mathcal{L}_{\text{CVAE}} = &\underbrace{\lambda_1 \mathcal{L}_{\text{rec}}(\mathbf{x}, \hat{\mathbf{x}})}_{\text{reconstruction loss}} + \underbrace{\lambda_2 \mathcal{L}_{\text{KL}}(\mathcal{N}(\boldsymbol{\mu}_z, \mathbf{I}\boldsymbol{\sigma}_z^2) \,\|\, \mathcal{N}(\mathbf{0}, \mathbf{I}))}_{\text{KL-divergence}} \\ &+ \underbrace{\lambda_3 \mathcal{L}_{\text{pred}}(\mathbf{y}, \hat{\mathbf{y}})}_{\text{prediction loss}} + \underbrace{\lambda_4 \mathcal{L}_{\text{cov}}(\mathbf{y}, \mathbf{z})}_{\text{regularization loss}}\end{aligned} \tag{3}$$

The first two terms (i.e., the reconstruction loss and the KL-divergence) of Eq. (3) equal the negative evidence lower bound (ELBO) in the standard formulation of a CVAE. Here, λ_2 acts similarly to the β parameter in β-VAEs, controlling the intensity of regularization. This enables a balance between accurate reconstruction of input data and the maintenance of a smooth, continuous latent space, thereby preventing undesirable sparsity. Similar to the CAE, we augment the loss function by adding a separate prediction loss, $\mathcal{L}_{\text{pred}}(\mathbf{y}, \hat{\mathbf{y}})$, on the target variables $\mathbf{y}$, and the regularization loss favoring disentangled representations, $\mathcal{L}_{\text{cov}}(\mathbf{y}, \mathbf{z})$. Finally, the objective function (Eq. (3)) is optimized by applying the reparameterization trick. For more information on this CVAE model we refer to [2] and for a more in-depth derivation of the ELBO and details regarding its optimization to [15,19].

3.2. Surrogate model

The encoder part of the trained autoencoder model serves as a surrogate model, effectively capturing the complex relationships between the design parameters (input variables) and performance attributes (conditional/target variables) (Fig. 6). The encoder receives one or a set of design, e.g. $\mathbf{x}^*$, and does a forward pass on the regression head.

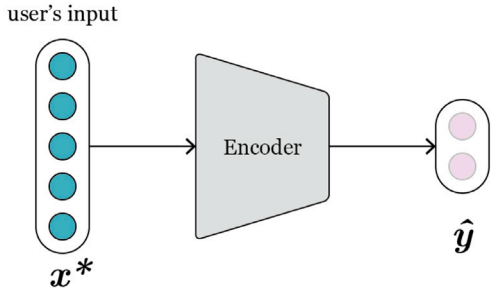


Fig. 6. Forward design/surrogate model.

3.3. Inverse design

The trained decoder acts as an inverse design model (Fig. 7), capable of generating new design parameters $\hat{\mathbf{x}}$ that meet specific performance criteria $\mathbf{y}^*$. Any number of solutions can be obtained by sampling repeatedly the latent space $\mathbf{z}$, and passing these samples along with $\mathbf{y}^*$ to generate the requested number of candidate designs $\hat{\mathbf{x}}$. To quantify the goodness of generation we follow the approach in [1] and compute the so-called *design error*. It leverages the encoder to compute $\hat{\mathbf{y}}$ for the generated design $\hat{\mathbf{x}}$, and then compares them to the desired performance attributes $\mathbf{y}^*$ using the sum of the absolute differences, as shown in the following equation:

$$\mathcal{L}_{\text{design-error}} = \sum_{i=1}^{d_y} |y_i^* - \hat{y}_i(\hat{\mathbf{x}})| \tag{4}$$

If the designer underspecifies the request, i.e., requesting values for just a subset of $\mathbf{y}$, the design error is only computed on the specified variables. This design error can be used to select the best instance of $\hat{\mathbf{x}}$. This mechanism is automatically leveraged in AIXD internally. By default, 10× more samples from $\mathbf{z}$ are sampled than the number of requested designs. Then, only the best 10% are provided as output to the user. The cost of the oversampling is negligible, because all these processes are carried out efficiently by the trained autoencoder.

3.4. Sensitivity analysis

In AIXD we distinguish and implement both local and global sensitivity analysis [2,13]. Local Sensitivity Analysis (LSA) examines the effect of small perturbations in design parameters $\mathbf{x}$ on the performance values $\hat{\mathbf{y}}$, at a baseline point $\mathbf{x}^*$. It is helpful for understanding the immediate neighborhood of a design point, identifying parameters that have the most influence on the performance attributes and selecting the direction of change in the parameter space with the aim of reaching a target performance. The sensitivity is calculated for each performance attribute with respect to all design parameters, thus defining the sensitivity matrix $\mathbf{S} \in \mathbb{R}^{d_y \times d_x}$, where each entry S_{ij} denotes the sensitivity of the ith output with respect to the jth input. Due to the fact that both

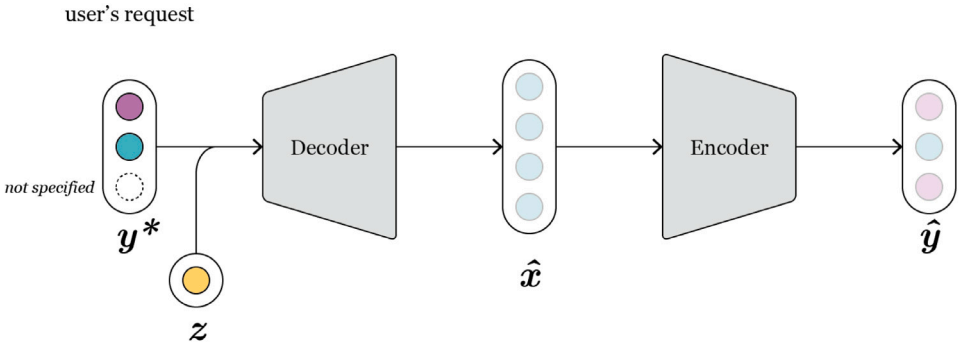


Fig. 7. Inverse design model/generative model.

continuous and discrete variables are supported as model inputs, when dealing with continuous input variables, the sensitivity is quantified using partial derivatives:

$$S_{ij} = \left.\frac{\partial \hat{y}_i(\mathbf{x})}{\partial x_j}\right|_{\mathbf{x}=\mathbf{x}^*} \tag{5}$$

which are calculated through automatic differentiation. On the other hand, the sensitivity of model outputs with respect to discrete, or categorical, model inputs is calculated as the difference:

$$S_{ij} = \hat{y}_i(\mathbf{x}^*) - \hat{y}_i(\mathbf{x}') \tag{6}$$

where $\mathbf{x}^*$ indicates the reference design point and $\mathbf{x}'$ denotes the perturbed design point in which the jth entry of $\mathbf{x}^*$ is switched to an alternative category.

Each entry of the sensitivity matrix $\mathbf{S} \in \mathbb{R}^{d_y \times d_x}$, which is delivered as output from such an analysis, describes the sensitivity of each $\mathbf{y}_i$ with respect to all $\mathbf{x}_j$, thus quantifying the effect of design parameter changes into the performance. In analogy to the functionality of the generator, the sensitivity analysis can be used in order to tweak the design variables in a way that only a subset or some specific attributes of the performance space are affected.

Unlike local sensitivity, Global Sensitivity Analysis (GSA) methods consider the entire range of input parameters and their statistically combined effect on the output. Techniques such as variance-based methods (e.g., Sobol' indices), Monte Carlo simulations, and Latin Hypercube Sampling are employed to provide a comprehensive picture of parameter importance across the entire design space.

3.5. Implementation details of ML model package

The `mlmodel` package encompasses the machine learning aspect of the toolbox and implements all required methods to seamlessly create a datamodule and a model from the dataset. All the required data transformations and manipulations are handled internally through a modular architecture built around heads, as depicted in Fig. 8, chosen based on the defined data types. For instance, categorical data types are encoded with a specialized head handling one-hot encoding, while ordinary numerical variables are by a head implementing a simple fully-connected linear layer. Furthermore, this modular architecture facilitates adding custom heads to process other data modalities, such as images or text.

Another advantage of the modular implementation is that, given that the data handling is decoupled from the machine learning model, new custom models can easily be attached to the parametric model (i.e., the dataset) or vice versa. For instance, a CVAE model with a more expressive latent distribution can readily be added, still leveraging the automatic data handling, freeing the user from this tedious task.

3.5.1. Software architecture

AIXD is implemented in Python using the frameworks PyTorch [20] and PyTorch Lightning [21], hence taking advantage of GPU acceleration for fast model training.

AIXD entails five main subpackages: `data` (modules for definition of data types, domains, encodings, dataset handling, data transformations etc.), `mlmodel` (implementation of the CAE and CVAE models, generator and sensitivity analysis tools), `sampler` (various sampling engines and strategies), `visualization` (plotter and evaluation tools), and `utils` (i.e., tools for logging). Visualization tools are built on top of the Plotly library [22], providing figures with varying interaction capabilities. One of the main design choices we made, was the separation of the ML model and the data handling into a corresponding `data` and `mlmodel` packages.

4. Applications

In this section we present the main application scenarios of AIXD, i.e., for inverse design, surrogate modeling and sensitivity analysis, in three exemplary projects.

4.1. Example: inverse design

In this example, AIXD is used to aid design exploration through inverse design in an architectural design project called Semiramis [1]. Semiramis is a vertical garden structure made of five bowl-shaped planters [23]. The inverse design task is to find the shapes of the planters (their outline curves) to satisfy a given planting surface area and exposure to sunlight and rainfall requested by the landscape architect.

The outline curve of each planter is parameterized by two variables (see Fig. 9): a *constellation* (a categorical variable that indexes 115 different subsets of column points) and *radii* (an 11-dimensional real-valued variable denoting the radii around column points, some of which are implicitly masked by the constellation). The solution space is very large and comprises $115^5 \approx 2 \cdot 10^{10}$ different combinations of constellations, and infinite variations of the radii. The performance attributes consist of three real-valued variables: *surface areas*, *sun occlusion* and *rain occlusion*. We aim to directly invert the parametric model, i.e., the input variables $\mathbf{x}$ of the autoencoder model correspond to the design parameters, while the conditional variables $\mathbf{y}$ consist of the performance attributes. The dimension of $\mathbf{x}$ is $(115 + 11) \cdot 5 = 630$ (after one-hot encoding of the categorical variable), and $\mathbf{y}$ is 3-dimensional.

The dataset consists of data samples obtained from the parametric design model built in Grasshopper/Rhino [24,25]. A single forward pass through the parametric model takes ca. 250 ms per sample,[4] and 24 h to generate around 350 000 samples. With this dataset,

[4] On a commercial laptop with an Intel i7 2.3 GHz CPU with 32 GB RAM.

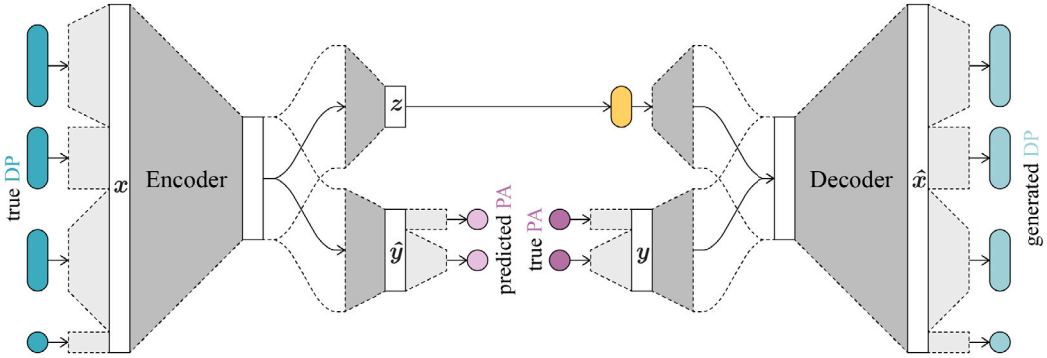


Fig. 8. Schematic of the model architecture. At the interface to and from the model, design parameters (DP) and performance attributes (PA) are provided in original dimensions and formats. **x** and **y** are obtained automatically after necessary transformations (shaded light gray).

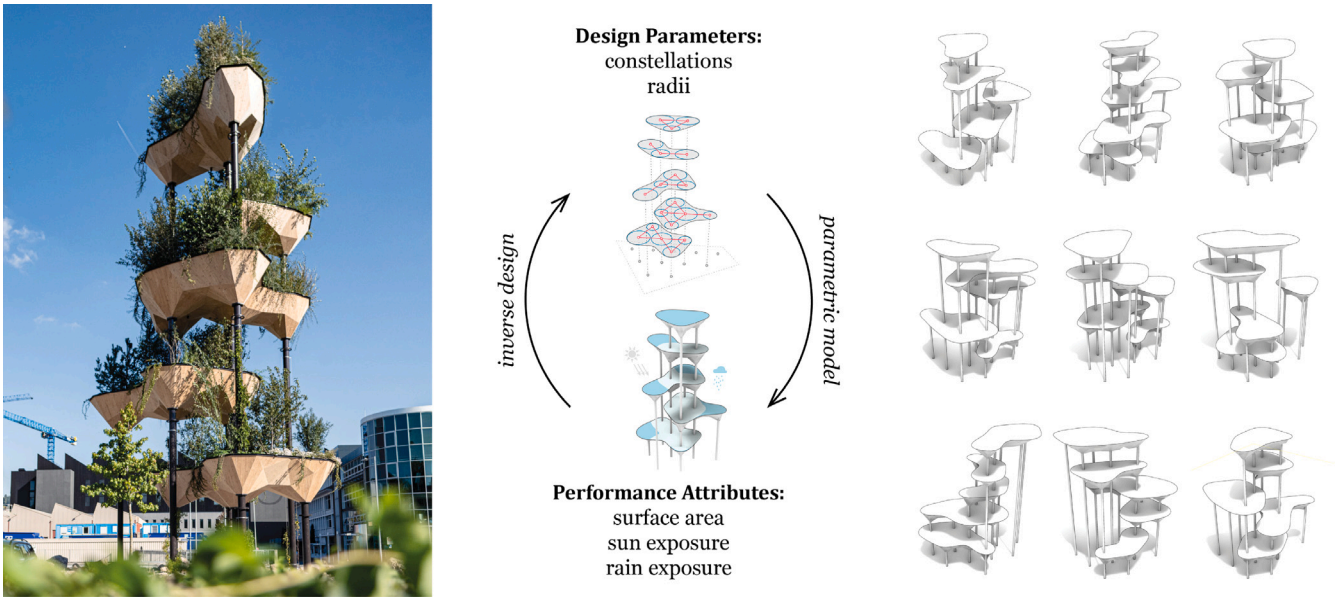


Fig. 9. Inverse design for a vertical garden, from [1]. Left: Completed Semiramis project installed in Zug, Switzerland (2022) (Photo: M. Lyrenmann, © Gramazio Kohler Research). Middle: list of design parameters and performance attributes. Right: Various designs generated with the autoencoder model for the same requested performance attributes.

a well-performing CAE model with four hidden layers with widths [512, 256, 128, 64] and a 30-dimensional latent space can be trained on CPU in minutes.[5] During inference, the trained model returns requested results almost instantly, which facilitates a near-interactive design exploration. The model can generate new designs with good accuracy, for example, within 0.3% error of the requested values for rain and surface variables in the densely populated value ranges (where their joint distribution is the most dense), and 3% error on the verges of the joint distribution. For other quantitative assessments, such as the novelty of the generated designs, we refer to [1].

In conclusion, the model can generate large numbers of accurate and diverse solutions, exceeding what the design team could craft manually in similar time, and allowing the architects and engineers to focus on other, more qualitative selection criteria.

4.2. Example: sensitivity analysis

AIXD can also support the design process and design exploration through its tools for sensitivity analysis. We exemplify it by a structural design project of a pedestrian bridge "Brücke über den Graben", from [2,26]. In this example, in addition to the inverse design capabilities of AIXD, the sensitivity analysis functionalities are demonstrated.

The design subject is a concrete girder bridge that provides passage to pedestrians over a busy street into a park while maintaining as many of the existing trees, ensuring structural safety, and cost-efficiency. The design parameters encode the geometry of the bridge with the heights h and thicknesses t of the girder and pier cross sections, the number of piers n and additional parameters i and w defining the bridge alignment (Fig. 10). The performance attributes are: structural performances (such as internal actions, resistances, displacements, structural and displacement utilizations and eigenfrequencies), boundary condition compliance, and costs, modeled and calculated with a parametric model in Dynamo [27], Revit [28], and Sofistik [29].

After obtaining the dataset of 12 000 random design instances and training an autoencoder model, the sensitivity analysis offers a means of explainability about the model and the design problem. Compared to established sensitivity analysis methods (e.g., leveraging FEM), AIXD is significantly more efficient and can calculate results in near real-time. Local sensitivities can improve the understanding of structural behavior in the neighborhood of a design point. For the pedestrian bridge they can be used to, e.g., identify which design parameters have the most influence on the negative bending moment over the supports M_y of that

[5] Depending on the batch size (e.g., 1024) and the number of epochs (e.g., 10).

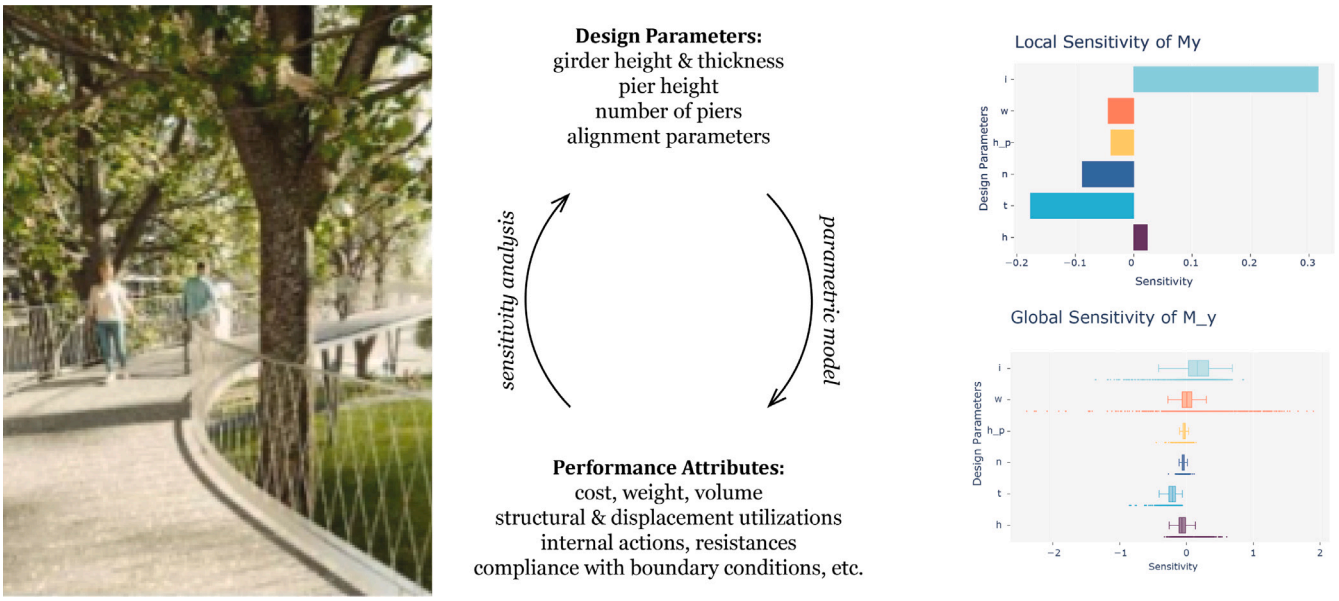


Fig. 10. Sensitivity analysis for design of a pedestrian bridge, from [2]). Left: visualization (© Planning team Basler & Hofmann AG, Nau2 and dgj Landscape). Middle: list of design parameters and performance attributes. Right: Example of graphical plots of sensitivities of the pedestrian bridge cost w.r.t to the design parameters.

specific design: The girder height and thicknesses (see Fig. 10 upper right).

On the other hand, by performing sensitivity analysis on a large set of designs (global sensitivity), the generalizability of local relationships can be evaluated and can further provide a better understanding of the network's decision-making. Thus, global sensitivities enable the derivation of helpful heuristics for design problems. For the pedestrian bridge application, it shows, e.g., how the sensitivities of the bending moment vary over the design space and identify the alignment parameters as the globally most influential design parameters (see Fig. 10 lower right). Further insights harnessing sensitivity analysis for AEC projects can be found in [2,13,30].

In this application example a CVAE was trained using a three-layer architecture with hidden dimensions [512,256,128] and a 2-dimensional latent space. For the trained model, R2 values ranging from 0.6 to 0.9 were achieved for the prediction of key performance attributes such as cost and utilization factors. Inverse design also yielded valid and diverse input configurations that satisfied target performance values, confirming the model's suitability for generative tasks in this setting.

4.3. Example: surrogate modeling

This example illustrates an application of AIXD to obtain a surrogate model for structural assessment of existing reinforced concrete frame bridges, a common type of structures built over the past decades in Switzerland as train track underpasses. Today, many of these bridge structures are approaching the end of their designed lifespan, and surrogate models can enable a swift assessment of their condition to help prioritize maintenance, strengthening or rebuild strategies. Conventional assessment methods require manual modeling of each structure followed by finite element simulations, resulting in hours of work per structure to obtain a preliminary estimate. The surrogate model is not intended to replace this analysis but to offer a rapid initial estimate.

In this study, the dataset consist of bridge geometries and non-linear finite element analysis simulation data. This data set contains over 10 000 frame bridge variations and is based on the data base of existing concrete frame bridges of the Swiss Federal Railways (SBB) with detailed structural assessment calculations gained with a parametric pipeline built with Grasshopper [25], Rhino [24] and Ansys Mechanical APDL [31] utilizing the StrucEngLib plugin [32]. The structural assessment problem is parameterized using design parameters describing the geometry span, plate thicknesses and reinforcement arrangements, as well as material properties and parameters defining the loading conditions. The performance attributes to predict are the code compliance factors. Once trained, the surrogate model provides a structural assessment estimate for an existing frame bridge in seconds, enabling a scalable approach to evaluate large bridge portfolios of large asset owners. Despite the highly non-linear relationship between input parameters and performance attributes, caused by structural element interactions and variable load positions (e.g., track locations), the CAE model is able to accurately capture these dependencies using a deep architecture with hidden layers [256,128,64,32,16]. To prioritize forward prediction over inverse learning, a heavy weight is assigned to the prediction loss term (Eq. (1)). The trained model achieves R^2 scores of 0.8 for bending and 0.6 for shear verification (Ultimate Limit State 2, according to Swiss norms), demonstrating its ability to approximate challenging non-linear behavior effectively. Further details on the underlying data set can be found in the in-depth study [33] (see Fig. 11).

5. Related work

In recent years, machine learning and deep learning techniques have been introduced to many design-related industries such as architecture, civil engineering, mechanical engineering or product design. While promising to transform aspects of the design process such as optimization, surrogate modeling, design exploration and explainability, so far their adoption in practice is rather experimental and explorative. This section gives an overview of the relevant techniques and applications found currently in research and practice.

5.1. Generative design

In the realm of AEC, the term *generative model* requires disambiguation from *generative design* which typically denotes any approach that facilitates the generation of different design variants. Generative design may refer to rule-based algorithms such as (shape) grammars [34,35] or genetic algorithms [36], or in general to the use of parametric models employed to produce various design candidates by performing multiple

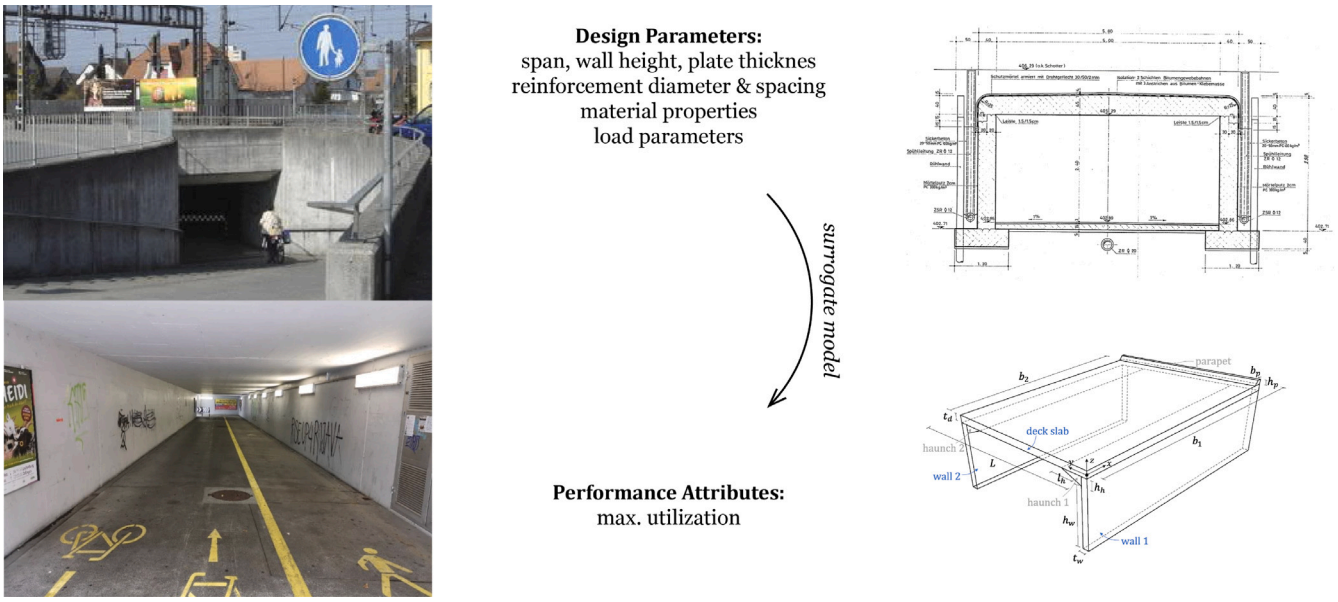


Fig. 11. Surrogate modeling of existing concrete frame bridges, from [33]. Left: Photos of an example concrete frame bridge functioning as a railway underpass in Switzerland. Middle: list of design parameters and performance attributes. Right: As-built plan of a concrete frame bridge example and the parametrization of the bridge type.

forward evaluations, querying the design space randomly, by grid-based sampling or through optimization procedures. Several generative design tools are commercially available today like Colibri [37] and Autodesk's Generative Design tool [38], and are often used in combination with dataset exploration and visualization tools like Design Explorer [39].

5.2. Generative models

By contrast to generative design, a *generative model* is a machine learning algorithm that can generate new content by sampling from an underlying distribution or noise and applying a series of transformations implicitly learned from data. Besides well-known examples in image synthesis and text generation, applications also include synthesis of 2D shapes [40] and 3D assets [41–43]. Popular types of generative models are Generative Adversarial Networks (GANs), (Variational) Autoencoders (VAEs) and Diffusion Models. Also encountered in this context but fundamentally different is Reinforcement Learning, which views the design as an outcome of a sequence of actions.

Generative models are oftentimes used in combination with some conditioning input to "guide" the synthesis and "restrict" the generated output to a particular area of interest, which is equivalent to the problem setting of *inverse design*.

5.3. Inverse design

Inverse design is of interest for many disciplines and industries, and is an active area of research in machine learning and generative models. Prominent applications are drug discovery [6] or material sciences [44] to find compounds or structures with specific sought-after properties or the calibration of model parameters from observation data [45]. A number of examples also show the potential of inverse design for mechanical, structural and architectural engineering and many other design domains. For example, using Variational Autoencoders, [46] presents a method for design exploration of spaceframes by navigating a 2-dimensional latent space and conditioning on a 1-dimensional value representing performance attributes. Similarly, [47] uses a VAE model to generate building blocks with optimal solar gain in a synthetic urban context, and integrates generative modeling with optimization problem-solving. A detailed study of latent space formulations and VAE model architectures for inverse design of network-tied arch bridges and masonry walls is discussed by [48]. Already presented in Section 4, [1] showcases an AI-guided generative design workflow using CAE for design of a vertical garden structure in Switzerland while [2] use CVAE for AI-guided generative design of a pedestrian bridge in Switzerland. Multiple works showcase generation of building floor plans using GANs [49] or diffusion models [50], conditioned on various user inputs such as outer boundary or a room connectivity diagram. These inverse design approaches using generative models have also been compared to more established generative design methodologies (cf. Section 5.1), for instance, [48] presents a comparison between CVAEs and genetic algorithms in engineering design examples.

5.4. Sensitivity analysis

Sensitivity analysis allows us to evaluate how variations in input design parameters impact the output performance attributes. This helps to provide insights into the robustness and reliability of the designs, identify how parameters influence the performance, identify key parameters, understand interactions, and guide the design process. These steps become essential when dealing with high-dimensional models, which are characterized by intricate relations. The sensitivity analysis of parametric models is typically carried out using perturbation [51], analytical gradient-based [52] or variance-based approaches [53]. An alternative to these formulations was based on a modified CVAE for forward and inverse design [2], in which the use of gradient-based sensitivity analysis is introduced as a mean of Explainable AI (XAI), demonstrating the approach on a pedestrian bridge in Switzerland. Within the same context, a method for computing local and global design decision metrics based on CVAE gradients to inform design decision discrete and high-dimensional design spaces was proposed in [13].

5.5. Surrogate modeling

Surrogate models are widely used for the so-called "forward engineering" or "forward design problem". The goal of a surrogate model may be, for example, to simulate an (unavailable) parametric model from historical or observational data such as physical experiments, or to approximate computationally expensive simulations. Surrogate models are thus often used to aid or expedite various downstream tasks in design [54,55] and optimization [56–58]. Among previous

research into the use of ML and DL algorithms for the learning of surrogates, the formulation and methodology of these models are strongly dependent on the use case, ranging from Proper Orthogonal Decomposition (POD) approaches for the identification [59,60] and real-time assessment [61] of engineering systems, to Gaussian Processes (GPs), Polynomial Chaos Expansion (PCE) and Kriging models for adaptive learning schemes [62], as well as to Bayesian model updating of uncertain systems [63]. Further developments include deep Neural Networks (NNs), which have been leveraged for the construction of multi-fidelity surrogates [64], advanced mixture of experts ensemble NNs [65,66], latent representations of the design space by means of CVAE [2,48,67], and the introduction of domain knowledge [68] or physics laws in the learning process, which has been also investigated using Physics-Informed Neural Networks (PINNs) [12,69,70] for linear and nonlinear systems.

5.6. Integration with Computer Aided Design software

Designers, architects and engineers typically use various and often multiple off-the-shelf software for carrying out Computer-Aided Design (CAD) tasks, such as geometric modeling, analysis and simulations. Even though AIXD is a stand-alone Python toolbox that allows leveraging data generated externally, it can benefit from the interaction with CAD software. Through this integration it is possible to enable the implementation of the entire AIXD pipeline in the CAD system, facilitating its usage by domain experts. This can be done through the Python APIs provided by software tools such as Autodesk Revit, Abaqus, or ANSYS. Conversely, other CAD environments do not implement a compatible API and require alternative communication protocols, such as subprocess or a REST interface, where AIXD code is executed on a local server. An example of such implementation can be seen in ARA [71] – a plugin that specifically interfaces AIXD with Grasshopper, a visual programming environment in the 3D modeling software Rhino. ARA allows to run AIXD from within the same high-level graphical user-interface as the parametric model of the design, reducing the implementation-barrier.

5.7. Comparison with other toolboxes and frameworks

As an open-source Python-based framework built on PyTorch, AIXD is uniquely tailored for project-specific inverse design, surrogate modeling, and sensitivity analysis, offering automated mixed-type data handling and integrated visualization for domain experts with minimal Python skills. This contrasts with CAD-integrated frameworks like Autodesk's commercial Generative Design tool in Revit [38] 5.1, as well as LunchBoxML [72], a plugin for commercial products like Rhino's Grasshopper, primarily focused on general ML tasks (regression, clustering, classification) and data management within existing design parameters, both lacking explicit inverse design or dedicated sensitivity analysis. Also, LunchBoxML does not automatically handle categorical variables. Furthermore, while the open-source high-level framework SDV (Synthetic Data Vault) can generate high-quality tabular data, its primary focus is on replicating statistical properties of existing tabular datasets, and it does not inherently possess the direct conditional modeling capabilities necessary for constrained generative design, where the goal is to create novel designs based on user-defined performance targets. Therefore, this positions AIXD as a specialized, low-code, higher-level framework that bridges the gap between powerful but complex low-level ML libraries (e.g., PyTorch, TensorFlow, JAX) and existing high-level libraries (e.g., Scikit-learn, Keras, and Hugging Face Transformers) that often focus solely on supervised/unsupervised models. Crucially, AIXD brings together functionalities for inverse design, surrogate modeling, and sensitivity analysis that might otherwise be scattered across different specialized libraries, consolidating and tailoring them specifically for generative design applications, thereby democratizing advanced AI capabilities for design practitioners.

6. Discussion and conclusions

6.1. Summary, limitations and future work

In this paper, we have introduced AIXD, a low-code, open-source toolkit designed to integrate machine learning techniques into the design process. The main features provided by the toolbox are the following:

- Simplified deployment and training of AI models.
- Tools to carry out inverse design, forward modeling and sensitivity analyses.
- Domain-agnostic methodology applicable to all problems where designs are defined parametrically.
- Intuitive template that helps to apply AIXD capabilities.
- Visualization tools to assist in the exploration and understanding of the design problems.
- Automation of repetitive tasks such as data handling, model setup, and evaluation.
- The possibility for advanced users to implement custom models and fine-tune the configurations.

AIXD offers unique capabilities such as the generation, using inverse design, of multiple design instances meeting specified performance attributes with one single generation call, or the understanding of the trade-offs between competing objectives by leveraging sensitivity analysis. But more importantly, AIXD intends to democratize the use of AI in design, enabling practitioners across various domains to leverage its features.

In order to enhance the toolkit's capabilities and usability, we have identified some limitations of AIXD that we will tackle in future releases[6]:

- Limited encoding options: AIXD is limited to a vectorial and fixed-length representation of inputs and outputs. We intend to devise embedding methods to allow AIXD to handle inputs of different size, as well as implement other data representations such as graphs and images by leveraging graph and convolutional neural networks. This direction is supported and builds on recent advancements in surrogate modeling with graph neural networks [73] and in graph-based generative modeling of structured design spaces [74].
- Model dimensionality and data size: More complex designs may require large amounts of data to achieve valid performance. This can be hindered by computationally expensive parametric models. We plan to incorporate active learning strategies to minimize the number of data samples required, while ensuring the required performances. This is crucial for dealing with long simulation times often encountered in the AEC industry.
- Model configuration and diversity: While AIXD offers default configurations that can deploy performant models, finding the optimal configuration still requires some expertise. Developing heuristic methods and AutoML techniques for the full creation and optimization of machine learning models will help users find the most effective model configurations without requiring extensive expertise. Furthermore, CAE and CVAE architectures, while suitable for a wide range of problems, may not be optimal for all types of design tasks or data distributions. To address this, we plan to extend the toolbox to support a broader range of model backends, including alternative generative approaches (e.g., diffusion models, transformer-based architectures). This will enable users to experiment with different modeling strategies depending on the characteristics of their specific design problem. The addition of further models is easily possible thanks to the modular software architecture of AIXD.

[6] At the time of writing this paper, the available version of AIXD is 1.0.0.

6.2. Conclusions

Computer-aided modeling and simulation software are today standard tools in design-related industries. However, only accelerated forward design and optimization approaches are used to streamline the usage of these design software tools. The adoption of ML/AI approaches is not yet widespread due to the following reasons: the expertise gap, which hinders the integration of these methods into the current design pipeline, and the project specificity, as data is project dependent hence hindering its usability. Furthermore, the usage of design software tools for the bulk generation of data and its application downstream to train ML models is seldomly explored. With AIXD we tackle these limitations. First, by abstracting away all the complexity, we bring ML/AI technologies closer to designers, enabling them to easily deploy models, explore the data, and seamlessly carry out design in different flavors. Then, by integrating the workflow into the standard parametric design pipeline, we overcome the data limitation, as now the parametric model, or data generated through it, can be easily leveraged inside AIXD and used downstream.

AIXD's features enable a broader exploration of the design space, uncovering innovative solutions that might be missed using traditional methods, and enhancing the understanding of complex design problems. It has the potential to increase the efficiency of designs with respect to desired performance goals, including cost efficiency, structural safety, and durability, as well as reducing environmental impact. AIXD is generic and applicable to any parametric problem across diverse domains. This versatility ensures that the toolkit can be adapted to various design challenges. By providing this tool as an open-source package, we hope to encourage users to adopt it and contribute to it. This way, we believe the AIXD toolbox can initiate a widespread adoption of ML methods into the AEC and design communities, and change the paradigm on how design is approached and tackled.

CRediT authorship contribution statement

Alessandro Maissen: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Data curation. **Aleksandra Anna Apolinarska:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Data curation, Conceptualization. **Sophia V. Kuhn:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Methodology, Investigation, Data curation. **Luis Salamanca:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Data curation, Conceptualization. **Michael A. Kraus:** Writing – review & editing, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Conceptualization. **Konstantinos Tatsis:** Writing – review & editing, Writing – original draft, Visualization, Software, Resources, Data curation. **Gonzalo Casas:** Software, Resources, Data curation. **Rafael Bischof:** Visualization, Software, Resources, Methodology, Data curation. **Romana Rust:** Writing – review & editing, Validation, Resources, Methodology, Investigation, Funding acquisition, Data curation, Conceptualization. **Walter Kaufmann:** Writing – review & editing, Project administration, Funding acquisition. **Fernando Pérez-Cruz:** Writing – review & editing, Project administration, Funding acquisition. **Matthias Kohler:** Supervision, Investigation, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The development of AIXD has been supported by research projects C20-09 "AI-augmented Architectural Design (AAAD)" (Gramazio Kohler Research and Swiss Data Science Center, ETH Zurich), C21-08 "Domain-Aware AI-augmented Design of Bridges (DAAAD Bridges)" (kfm Research and Swiss Data Science Center, ETH Zurich), ETH Foundation grant No. 2020-HS-388 (kfm Research), and Design++ Center for Augmented Computational Design in Architecture, Engineering and Construction at ETH Zurich.

Data availability

Source code and all data required to run the examples is already available in the repository.

References

[1] Salamanca L, Apolinarska AA, Pérez-Cruz F, Kohler M. Augmented intelligence for architectural design with conditional autoencoders: Semiramis case study. In: Towards radical regeneration. Cham: Springer International Publishing; 2023, p. 108–21. http://dx.doi.org/10.1007/978-3-031-13249-0_10.

[2] Balmer V, Kuhn SV, Bischof R, Salamanca L, Kaufmann W, Perez-Cruz F, et al. Design space exploration and explanation via conditional variational Autoencoders in meta-model-based conceptual design of pedestrian bridges. Autom Constr 2024;163:105411. http://dx.doi.org/10.1016/j.autcon.2024.105411, URL https://www.sciencedirect.com/science/article/pii/S092658052400147X.

[3] Wortmann T, Costa A, Nannicini G, Schroepfer T. Advantages of surrogate models for architectural design optimization. Artif Intell Eng Des Anal Manuf 2015;29(4):471–81. http://dx.doi.org/10.1017/S0890060415000451.

[4] Forrester AI, Sóbester A, Keane A. Engineering design via surrogate modelling: a practical guide. Chichester, UK: John Wiley & Sons; 2008.

[5] Sanchez-Lengeling B, Aspuru-Guzik A. Inverse molecular design using machine learning: Generative models for matter engineering. Science 2018;361(6400):360–5. http://dx.doi.org/10.1126/science.aat2663, URL https://www.science.org/doi/abs/10.1126/science.aat2663.

[6] Bian Y, Xie X-Q. Generative chemistry: Drug discovery with deep learning generative models. J Mol Model 2021;27:1–18.

[7] Zunger A. Inverse design in search of materials with target functionalities. Nat Rev Chem 2018;2(4):0121.

[8] Qi W, Longfei Z. Inverse design of glass structure with deep graph neural networks. Nat Commun 2021;12(5359).

[9] Naseri P, Hum SV. A generative machine learning-based approach for inverse design of multilayer metasurfaces. IEEE Trans Antennas and Propagation 2021;69(9):5725–39.

[10] Nauata N, Chang K-H, Cheng C-Y, Mori G, Furukawa Y. House-GAN: Relational generative adversarial networks for graph-constrained house layout generation. In: Computer vision–ECCV 2020: 16th European conference, Glasgow, UK, August 23–28, 2020, proceedings, part i 16. Springer; 2020, p. 162–77.

[11] Oh S, Jung Y, Kim S, Lee I, Kang N. Deep generative design: integration of topology optimization and generative models. J Mech Des 2019;141(11):111405.

[12] Haghighat E, Raissi M, Moure A, Gomez H, Juanes R. A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. Comput Methods Appl Mech Engrg 2021;379:113741. http://dx.doi.org/10.1016/j.cma.2021.113741, URL https://www.sciencedirect.com/science/article/pii/S0045782521000773.

[13] Fang D, Kuhn SV, Kaufmann W, Kraus MA, Mueller C. Quantifying the influence of continuous and discrete design decisions using sensitivities. In: Advances in architectural geometry. Stuttgart; 2023, http://dx.doi.org/10.1515/9783111162683-031.

[14] Biewald L. Experiment tracking with weights and biases. 2020, Software available from wandb.com. URL https://www.wandb.com/.

[15] Sohn K, Lee H, Yan X. Learning structured output representation using deep conditional generative models. In: Cortes C, Lawrence N, Lee D, Sugiyama M, Garnett R, editors. In: Advances in neural information processing systems, vol. 28, Curran Associates, Inc.; 2015, URL https://proceedings.neurips.cc/paper_files/paper/2015/file/8d55a249e6baa5c06772297520da2051-Paper.pdf.

[16] Zhao Q, Adeli E, Honnorat N, Leng T, Pohl KM. Variational AutoEncoder for regression: Application to brain aging analysis. In: Shen D, Liu T, Peters TM, Staib LH, Essert C, Zhou S, Yap P-T, Khan A, editors. Medical image computing and computer assisted intervention. Cham: Springer International Publishing; 2019, p. 823–31.

[17] Goodfellow I, Bengio Y, Courville A. Deep learning. MIT Press; 2016, http://www.deeplearningbook.org.

[18] Ghosh P, Sajjadi MSM, Vergari A, Black M, Scholkopf B. From variational to deterministic Autoencoders. In: International conference on learning representations, ICRL. 2020, URL https://openreview.net/forum?id=S1g7tpEYDS.

[19] Kingma DP, Welling M. Auto-encoding variational Bayes. 2022, arXiv:1312.6114.

[20] Ansel J, Yang E, He H, Gimelshein N, Jain A, Voznesensky M, et al. Pytorch 2: Faster machine learning through dynamic Python Bytecode transformation and graph compilation. In: 29th ACM international conference on architectural support for programming languages and operating systems, vol. 2, ACM; 2024, http://dx.doi.org/10.1145/3620665.3640366, URL https://pytorch.org/assets/pytorch2-2.pdf.

[21] Falcon W, The PyTorch Lightning team. Pytorch lightning. 2019, http://dx.doi.org/10.5281/zenodo.3828935, URL https://github.com/Lightning-AI/lightning.

[22] Plotly Technologies Inc. Collaborative data science. 2015, URL https://plot.ly.

[23] Gramazio Kohler Research, ETH Zurich and various project partners. Semiramis. 2021, URL https://gramaziokohler.arch.ethz.ch/web/e/projekte/409.html. https://gramaziokohler.arch.ethz.ch/web/e/projekte/409.htm.

[24] Robert McNeel & Associates. Rhinoceros. 2010-02-19, URL www.rhino3d.com. www.rhino3d.com.

[25] McNeel, David Rutten. Grasshopper. 2010-02-19, URL https://www.rhino3d.com/features/#grasshopper.

[26] Basler & Hofmann AG. Mit künstlicher intelligenz zur optimalen Brücke: Basler & hofmann engagiert sich für forschung der ETH zürich. 2022, URL https://www.baslerhofmann.ch/aktuelles/details/mit-kuenstlicher-intelligenz-zur-optimalen-bruecke-basler-hofmann-engagiert-sich-fuer-forschung-der-eth-zuerich.html. [Accessed 9 July 2024].

[27] Autodesk. Dynamo: Visual programming for design. 2024, URL https://dynamobim.org/. [Accessed 9 July 2024].

[28] Autodesk. Revit: BIM software for architects, engineers, and contractors. 2024, URL https://www.autodesk.com/products/revit/overview. [Accessed 9 July 2024].

[29] Sofistik AG. SoFiSTiK: Structural analysis and design software. 2024, URL https://www.sofistik.de/. [Accessed 9 July 2024].

[30] Fang D, Wang P, Kuhn SV, Kraus MA, Mueller C. Trans-typology design space exploration: Using gradients to inform decision-making in the design of spanning structures. In: Proceedings of the IASS 2024 symposium. Zurich; 2024.

[31] ANSYS Inc. ANSYS mechanical APDL. 2024, Available from: https://www.ansys.com/products/structures/ansys-mechanical-apdl. [Accessed January 2025].

[32] kfmResearch. Strucenglibplugin 0.0.22. 2024, URL https://github.com/kfmResearch-NumericsTeam/StrucEng_Library_Plug_in.

[33] Kuhn SV, Weber M, Binggeli A, Kraus MA, Perez-Cruz F, Kaufmann W. Predictive structural assessment of concrete frame bridges with Bayesian deep learning. 2025, Manuscript submitted for publication.

[34] Ruiz-Montiel M, Boned J, Gavilanes J, Jiménez E, Mandow L, de-la Cruz J-LP. Design with shape grammars and reinforcement learning. Adv Eng Inform 2013;27(2):230–45. http://dx.doi.org/10.1016/j.aei.2012.12.004, URL https://www.sciencedirect.com/science/article/pii/S1474034612001139.

[35] Karin LY, Kraus MA, Chatzi E, Kaufmann W. Grammar-based generation of strut-and-tie models for designing reinforced concrete structures. Comput Struct 2024;305:107549.

[36] Khan S, Awan MJ. A generative design technique for exploring shape variations. Adv Eng Inform 2018;38:712–24. http://dx.doi.org/10.1016/j.aei.2018.10.005, URL https://www.sciencedirect.com/science/article/pii/S1474034618300983.

[37] Thornton Tomasetti CORE studio. Colibri. 2022, URL https://www.food4rhino.com/en/app/colibri. https://www.food4rhino.com/en/app/colibri.

[38] Autodesk. Generative design. 2024, URL https://help.autodesk.com/view/RVT/2024/ENU/?guid=GUID-492527AD-AAB9-4BAA-82AE-9B95B6C3E5FE. https://help.autodesk.com/view/RVT/2024/ENU/?guid=GUID-492527AD-AAB9-4BAA-82AE-9B95B6C3E5FE.

[39] Thornton Tomasetti CORE studio. Design explorer 2. 2019, URL https://tt-acm.github.io/DesignExplorer/. https://tt-acm.github.io/DesignExplorer/.

[40] Wang X, Qian W, Zhao T, Chen H, He L, Sun H, et al. A generative design method of airfoil based on conditional variational autoencoder. Eng Appl Artif Intell 2025;139:109461. http://dx.doi.org/10.1016/j.engappai.2024.109461, URL https://www.sciencedirect.com/science/article/pii/S0952197624016191.

[41] Xie J, Zheng Z, Gao R, Wang W, Zhu S-C, Wu YN. Learning descriptor networks for 3D shape synthesis and analysis. 2018, arXivURL https://arxiv.org/abs/1804.00586.

[42] Gao J, Shen T, Wang Z, Chen W, Yin K, Li D, et al. GET3D: A generative model of high quality 3D textured shapes learned from images. In: Advances in neural information processing systems. 2022.

[43] Xu X, Jayaraman PK, Lambourne JG, Willis KDD, Furukawa Y. Hierarchical neural coding for controllable CAD model generation. 2023, arXiv:2307.00149. URL https://arxiv.org/abs/2307.00149.

[44] Sanchez-Lengeling B, Aspuru-Guzik A. Inverse molecular design using machine learning: Generative models for matter engineering. Science 2018;361(6400):360–5.

[45] Kraus MA, Schuster M, Kuntsche J, Siebert G, Schneider J. Parameter identification methods for visco-and hyperelastic material models. Glas Struct Eng 2017;2(2):147–67.

[46] Danhaive R, Mueller CT. Design subspace learning: Structural design space exploration using performance-conditioned generative modeling. Autom Constr 2021;127:103664. http://dx.doi.org/10.1016/j.autcon.2021.103664, URL https://www.sciencedirect.com/science/article/pii/S0926580521001151.

[47] Ampanavos S, Nourbakhsh M, Cheng CY. Structural design recommendations in the early design phase using machine learning. In: Gerber D, Pantazis E, Bogosian B, Nahmad A, Miltiadis C, editors. Computer-aided architectural design. Design imperatives: The future is now. Springer Singapore; 2022, p. 190–202.

[48] Bucher MJJ, Kraus MA, Rust R, Tang S. Performance-based generative design for parametric modeling of engineering structures using deep conditional generative models. Autom Constr 2023;156:105128. http://dx.doi.org/10.1016/j.autcon.2023.105128, URL https://www.sciencedirect.com/science/article/pii/S0926580523003886.

[49] Chaillou S. ArchiGAN: Artificial intelligence x architecture. In: Yuan PF, Xie M, Leach N, Yao J, Wang X, editors. Architectural intelligence: selected papers from the 1st international conference on computational design and robotic fabrication. Singapore: Springer Nature Singapore; 2020, p. 117–27. http://dx.doi.org/10.1007/978-981-15-6568-7_8.

[50] Shabani MA, Hosseini S, Furukawa Y. HouseDiffusion: Vector floorplan generation via a diffusion model with discrete and continuous denoising. 2022, arXiv:2211.13287. URL https://arxiv.org/abs/2211.13287.

[51] Castillo E, Conejo A, Castillo C, Minguez R, Ortigosa D. Perturbation approach to sensitivity analysis in mathematical programming. J Optim Theory Appl 2006;128:49–74.

[52] Kovacs I, Iosub A, Ţopa M, Buzo A, Pelz G. A gradient-based sensitivity analysis method for complex systems. In: 2019 IEEE 25th international symposium for design and technology in electronic packaging. 2019, p. 333–8.

[53] Andrea S, Paola A, Ivano A, Francesca C, Marco R, Stefano T. Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index. Comput Phys Comm 2010;181(2):259–70.

[54] Jiang J, Chen M, Fan JA. Deep neural networks for the evaluation and design of photonic devices. Nat Rev Mater 2021;6:679–700. http://dx.doi.org/10.1038/s41578-020-00260-1.

[55] Wang L, Janssen P, Ji G. SSIEA: A hybrid evolutionary algorithm for supporting conceptual architectural design. Artif Intell Eng Des Anal Manuf 2020;34(4):pp. 458–476. http://dx.doi.org/10.1017/S0890060420000281.

[56] Jiang P, Zhou Q, Shao X. Surrogate-model-based design and optimization. In: Surrogate model-based engineering design and optimization. Springer Singapore; 2020, p. 135–236.

[57] Peri D. Machine learning algorithms in design optimization. 2022, arXiv:2203.11005.

[58] Wortmann T. Opossum - Introducing and evaluating a model-based optimization tool for Grasshopper. In: Janssen P, Loh P, Raonic A, Schnabel MA, editors. Protocols, flows, and glitches - proceedings of the 22nd CAADRIA conference. CAADRIA; 2017, p. 283–92.

[59] Agathos K, Tatsis KE, Vlachas K, Chatzi E. Parametric reduced order models for output-only vibration-based crack detection in shell structures. Mech Syst Signal Process 2022;162:108051. http://dx.doi.org/10.1016/j.ymssp.2021.108051, URL https://www.sciencedirect.com/science/article/pii/S0888327021004404.

[60] Tatsis KE, Agathos K, Chatzi EN, Dertimanis VK. A hierarchical output-only Bayesian approach for online vibration-based crack detection using parametric reduced-order models. Mech Syst Signal Process 2022;167:108558. http://dx.doi.org/10.1016/j.ymssp.2021.108558, URL https://www.sciencedirect.com/science/article/pii/S0888327021008967.

[61] Mainini L, Willcox K. Surrogate modeling approach to support real-time structural assessment and decision making. AIAA J 2015;53(6):1612–26. http://dx.doi.org/10.2514/1.J053464.

[62] Gaspar B, Teixeira A, Guedes Soares C. Adaptive surrogate model with active refinement combining Kriging and a trust region method. Reliab Eng Syst Saf 2017;165:277–91. http://dx.doi.org/10.1016/j.ress.2017.03.035, URL https://www.sciencedirect.com/science/article/pii/S0951832016301892.

[63] Ramancha MK, Vega MA, Conte JP, Todd MD, Hu Z. Bayesian model updating with finite element vs surrogate models: Application to a miter gate structural system. Eng Struct 2022;272:114901. http://dx.doi.org/10.1016/j.engstruct.2022.114901, URL https://www.sciencedirect.com/science/article/pii/S0141029622009798.

[64] Torzoni M, Manzoni A, Mariani S. A multi-fidelity surrogate model for structural health monitoring exploiting model order reduction and artificial neural networks. Mech Syst Signal Process 2023;197:110376. http://dx.doi.org/10.1016/j.ymssp.2023.110376, URL https://www.sciencedirect.com/science/article/pii/S0888327023002832.

[65] Kraus MA, Bischof R, Kaufmann W, Thoma K. Artificial intelligence-finite element method-hybrids for efficient nonlinear analysis of concrete structures. In: International probabilistic workshop 2022, vol. 36, Czech Techical University; 2022, p. 99–108.

[66] Kraus MA, Bischof R, Riedel H, Schmeiser L, Pauli A, Stelzer I, et al. Strength Lab AI: a mixture-of-experts deep learning approach for limit state analysis and design of monolithic and laminate structures made of glass. Glas Struct Eng 2024;9(3):607–55.

[67] Kraus MA, Müller A, Bischof R, Taras A. Predictive modelling and latent space exploration of steel profile overstrength factors using multi-head autoencoder-regressors. Ce/Papers 2023;6(3–4):836–42.
[68] Bischof R, Sprenger M, Riedel H, Bumann M, Walczok W, Drass M, et al. Temp-AI-estimator: Interior temperature prediction using domain-informed deep learning. Energy Build 2023;297:113425.
[69] Bischof R, Kraus M. Multi-objective loss balancing for physics-informed deep learning. 2021, arXiv preprint arXiv:2110.09813.
[70] Balmer VM, Kaufmann W, Kraus MA. Physics-informed neural networks for nonlinear analysis of reinforced concrete beams. In: International probabilistic workshop. Springer; 2024, p. 271–80.
[71] Apolinarska AA, Casas G, Salamanca L. ARA: Grasshopper plugin for data-driven and inverse design. 2024, URL https://github.com/gramaziokohler/aixd_ara. https://github.com/gramaziokohler/aixd_ara.
[72] Miller N. LunchBox. 2020, URL https://apps.provinggground.io/lunchbox/. https://apps.provinggground.io/lunchbox.
[73] Hadizadeh F, Mallik W, Jaiman RK. A graph neural network surrogate model for multi-objective fluid-acoustic shape optimization. 2024, arXiv preprint arXiv:2412.16817.
[74] Zheng L, Karapiperis K, Kumar S, Kochmann DM. Unifying the design space and optimizing linear and nonlinear truss metamaterials by generative modeling. Nat Commun 2023;14(1):7563.