C. Belém, L. Santos, and A.Leitão, "On the Impact of Machine Learning. Architecture without Architects?" in *CAAD Futures 2019,* Seoul, South Korea

On the Impact of Machine Learning

Architecture without Architects?

Catarina Belém[1], Luis Santos[2], António Leitão[1]

[1] INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
[2] UC Berkeley, Center for the Built Environment, United States of America

Abstract. Architecture has always followed and adopted technological breakthroughs of other areas. As a case in point, in the last decades, the field of computation changed the face of architectural practice. Considering the recent breakthroughs of Machine Learning (ML), it is expectable to see architecture adopting ML-based approaches. However, it is not yet clear how much this adoption will change the architectural practice and in order to forecast this change it is necessary to understand the foundations of ML and its impact in other fields of human activity. This paper discusses important ML techniques and areas where they were successfully applied. Based on those examples, this paper forecast hypothetical uses of ML in the realm of building design. In particular, we examine ML approaches in conceptualization, algorithmization, modeling, and optimization tasks. In the end, we conjecture potential applications of such approaches, suggest future lines of research, and speculate on the future face of the architectural profession.

Keywords: Machine Learning, Algorithmic Design, AI for Building Design

1 Introduction

In the last decades, computational advances changed the way architects design. Computation revolutionized architecture and, nowadays, computational approaches are fully embedded in the architectural practice.

Recently, a new computational revolution is under way. This revolution is being driven by recent breakthroughs in the area of Machine Learning (ML) [1] and it already affected many fields [2], including medicine [3-4], physics [5], and finance [6], among others. ML is a non-symbolic branch of Artificial Intelligence (AI), based on computational statistics and optimization procedures, that explore self-improving learning techniques to solve problems or perform specific tasks. In contrast to symbolic approaches to AI, non-symbolic approaches strive to build computational systems that do not need to be programmed to perform the task. Particularly, ML builds mathematical models of sampled data, known as training data, and adapts its parameters to progressively improve its performance on specific tasks without any human intervention [1, 7].

Across diverse domains, the benefits and successes of ML are contributing not only to automate tiresome and monotonous tasks, but also to aid in decision-making processes. Given the impact ML has had in many diverse areas, it is expected that, sooner than later, it will have a similar impact in architecture. However, at this moment, ML is seldom applied in architecture [8]. Exceptions include optimized control strategies for building systems operations, generation of surrogate models for design optimization, floor plan layout, urban planning, construction modeling, among others [8-13]. Regarding design methods, [8] presents an education framework for teaching ML to architects with the aim of facilitating the deployment of such techniques. Moreover, [13] argues that the adoption of AI and ML techniques will result in more intuitive design tools. Different works have also proposed the application of different ML methods to architecture [14-15], emphasizing how the advances of ML, such as in computer vision, can intersect architecture and improve processes like building design and digital fabrication.

Given the rising success of ML methods and its inevitable use in architecture, a new concern has emerged: will machines replace humans in architectural practice? Will these methods limit the architect's creative control? Will we have architecture without architects?

In this paper, we delve into these questions, identifying and discussing the potential impacts of ML in different design tasks. The following section introduces ML. In section 3, we discuss the benefits and risks of applying ML in current digital architectural practice. Finally, we end by discussing the main takeaways and draw future research paths.

2 Related Work

The study of systems with human cognitive capabilities [16], i.e., capable of understanding, acting, speaking, and thinking, and learning like humans, is not recent. For decades, AI attempted to create these systems, which were often limited by the computational resources or the knowledge instilled to them. These systems were not nearly as successful as the ML systems introduced decades later that achieved human level performance, or even surpassed it, on an increasing number of complex tasks. Among the most reputed superhuman systems, which surpass the performance of any human expert in a specific task, are AlphaGo [17] and DeepStack [18] in the field of games, IBM's Watson [19] in the field of question-answering, and image classification systems [20-21].

In the following sections, we present the fundamental concepts of ML and its potential application in architecture.

Machine Learning Concepts

Machine Learning (ML), also known as Statistical Learning [22], focus on algorithms capable of learning from data. Essentially, they require a training set of data containing examples of past experiences and an algorithm that builds a mathematical model out of the training set samples in order to make predictions or decisions, i.e., to learn without being explicitly programmed to do so. The set of attributes associated to an example are called features. For example, to learn how to predict the house prices, we represent houses in terms of features (e.g., number of rooms, total area, coordinates) and then use those features to correlate to their price, i.e., their output feature. A good ML system (or learner) is one that accurately predicts or decides based on a given training set [22].

The characteristics of the problem to solve and the available data determine the type of ML approach. This paper discusses two fundamental ML learning methods. For more detailed descriptions of other learning tasks, see [23].

Supervised Learning: uses a data training set containing both the inputs and the corresponding outputs, i.e., the associated labels. The outputs are used to guide the learning process. Supervised learning is used to address classification, regression, and ranking problems. An example of a supervised setting is to predict the species of a flower based on their petal attributes (e.g., length and width).

Unsupervised Learning: uses an unlabeled training data set, i.e., a training set that only contains the inputs. In this learning task, the goal is to describe how the data is organized or clustered, and, consequently, to make predictions for unforeseen data points. Because there is no labelled example, it can be difficult to evaluate the performance of an unsupervised learner. Unsupervised learning is often used to address clustering and dimensionality reduction problems. An example is to group flowers in different groups according to their petal attributes (e.g., determine the flowers species by grouping flowers according to their petal length).

Regarding the mathematical models used in ML, the most common ones are Artificial Neural Networks (ANNs), Support Vector Machines (SVMs), Bayesian networks, Radial Basis Function networks (RBFs), Gaussian Processes (GPs), Linear Regressions, Decision Trees, Random Forests, Markov Random Fields, restricted Boltzmann Machines (RBMs), K-means, k-Nearest Neighbours (kNN), Expectation Maximization Clustering, Principal Component Analysis (PCA), among others. Each model works differently, and some are more intelligible than others. For example, for Linear Regression, it is easy to understand the outcomes and the reasoning behind certain predictions, while ANNs are often opaque since it is unclear the correlation of inputs to specific outcomes. The work presented in [1,22] gives a complete description of the previously mentioned ML models, along with their advantages and disadvantages.

Many ML techniques require large amounts of data in order to generalize well, which might be problematic in settings where data is not easily available. To bridge the data gap and spur better performance in such settings, the transfer learning methodology [24] might be a solution. This methodology explores the ability to transfer knowledge between tasks applying relevant knowledge from previous learning experiences, therefore allowing to master new tasks more quickly, more easily, and with less data.

Initially, ML applications focused on the analysis of existing data but, more recently, ML is also being applied to creative tasks using techniques such as Neural Style Transfer [25], DeepDream [26], and Generative Adversarial Networks (GANs) [27-28].

3 The Impact of Machine Learning in Architectural Design

Current research [8-13] already illustrates some ML applications in architecture. We complement those studies by hypothesizing on alternative applications of ML techniques that might impact the architectural practice. Whenever possible, we will trace an analogy of these hypothetical applications with similar successful applications in other fields.

Several of the proposed applications envisage the improvement of existing design tools, by using ML to allow a more efficient and informed design process. In fact, ML shows large potential for enhancing the design process since some of its learning algorithms resemble reasoning processes that are frequently applied by architects [15], such as abductive reasoning, a form of logical inference based on observation and inferral of the most likely explanation for what has been observed [29]. Abductive reasoning in design explains the frequent practice of making representations of the designs before their implementation, i.e., based on past observations and experiences, architects select whatever elements fulfil the initial representation [15].

Although ML will have a broad impact in architectural practice, in the next sections we focus on the following aspects: (1) conceptualization, including conceptual definition, approach, and exploration by the designer, (2) algorithmization, which consists in developing and implementing a computer program capable of representing and instantiating the applied concepts, (3) modeling, which encompasses the tasks of 3D modeling either for visualization or building simulation, and (4) optimization, i.e., the search for high-performance solutions according to different criteria defined beforehand by the design team.

3.1 Conceptualization

Conceptualization tasks are essential at early design stages and consist in the definition of the main ideas, strategies, and spatial and temporal narratives of a specific design. Although there are few structured approaches or guidelines for conceptualization tasks, it is not uncommon for architects to use systematic approaches that can either assume a bottom-up or a top-down structure. In the bottom-up approach, several concepts and strategies are incrementally created and combined to form complex and compound concepts. On the contrary, in the top-down approach, there is a high-level description of the idealized design concepts, such as the desired types of space, or the lighting conditions, among others. This description is then processed in different ways to create the corresponding design. The next subsections will depict how different ML techniques could help designers in conceptualization tasks.

Artificial Neural Networks and Deep Learning

When starting off a project, architects often spend considerable time studying references and related projects in order to envision a design that fulfils the client's needs. As a consequence, architects frequently get absorbed in large amounts of incomplete, complex, and ambiguous information, from which they attempt to infer which concept or image better fits the project's brief. One of the main features of ML is its ability to capture patterns and correlations concealed in the data. This feature can augment the conceptual design process, for example, by retrieving different conceptual ideas according to the project's requirements and context information provided by the architect using a pre-defined vocabulary. The retrieved concepts might be presented in either textual or visual descriptions. Regardless of their representation, such ML use allows the rapid exploration of design concepts and reduces the time spent researching design references and context-based information, such as, the vernacular buildings of a given site.

Regarding textual representations, we can explore the ML techniques typically used in natural language processing tasks, such as, automatic summarization, text simplification, and machine translation [30-32]. By synthesizing the provided input into a summary of the main ideas and by extracting the meaning of those ideas, it might be possible to encode them into design features. Then, ANNs can learn to map the encoded features to conceptual ideas. Another alternative is to represent the information directly using the vocabulary terms and use algorithms, such as ANNs, SVMs, or Naive Bayes classifiers [33-36], to learn the associations between the provided vocabulary terms and the conceptual ideas, or even to use unsupervised learning algorithms, such as, kNN, Hierarchical Agglomerative Clustering, Self-Organizing Maps (SOMs), or Gaussian Mixtures, among others, to learn how input features are related to each other in terms of the outputs. Unfortunately, by using the vocabulary approach, the input size might grow considerably, incurring in the curse of dimensionality [1, 22]. To address this problem, one can try to obtain more data, which can be challenging, or one might apply dimensionality reduction methods, like PCA or Latent Dirichlet Allocation (LDA) [37].

Concerning the production of visual representations, we can use the textual output of the previous system to produce corresponding visual examples. This technique is called text-to-image synthesis and there are examples, particularly in the translation of visual concepts from characters to pixels [38-40] and the retrieval of objects and scenes from videos [41]. Given the speed of recent developments in the fields of ML, we believe that it is only a matter of time until we get "text-to-3D models" synthesis. The utmost difficulty lies in the definition of an accurate representation of 3D models in terms of features that can be exploited by the learning methods of ML. The described system would integrate seamlessly into a typological conceptual design approach. The architect would have to describe the context and the project's requirements, upload the typological description in the design tool, and the system would return suggestions for different conceptual ideas complying to such conditions.

Recommender Systems

A recommender system helps users making decisions tasks by providing suggestions for the most appropriate course of action [42]. These systems use the features of each alternative to rank them and generate customized suggestions. While experienced users do not usually need such suggestions, it can be advantageous for less experienced ones.

We argue that some of the fundamental concepts of recommender systems could also be applied in architecture, for example, to suggest specific design templates (e.g., for truss types, floor plan layouts, or façade designs) during the initial and exploratory stages of the design process. We envision the improvement of conceptualization tasks through the suggestion of design variations extrapolated from a set of previous records. To this end, we can: (1) extrapolate the existing patterns in the design records and, hence, codify these patterns as templates; (2) analyze the interactions between the architect and his design tool, making more informed and accurate suggestions regarding the next developments in the conceptual design; (3) learn the set of operations mostly used and exploit this information to augment the architectural design tool; and (4) advise the architect to avoid design paths that might yield a poor performance in a set of predefined criteria.

Unfortunately, these systems suffer from the cold-start problem [43], which happens when the system has no past information about new users. In this case, online learning techniques can be a useful methodology to follow. Through conversational interfaces, where the system poses questions to capture the architect's intent, thus making more adequate suggestions. Additionally, conversational approaches also enable thoughtful reflections about the design, leading to clearer perspectives about the involved concepts, and to the identification of conceptual gaps [44].

However, the construction of an ML system encompassing the whole architectural community designs' history would require a large infrastructure to efficiently manage the information. Alternatively, one could opt for a simpler ML system that would only be responsible for managing the information with respect to just one user. In this case, the templates gamut would be restricted by the architect's past experiences, potentially compromising variety. To address this limitation, a knowledge-base must be constructed *a priori* to maximize the efficiency and range of template suggestions.

In the architectural field, Gagne et al. [45] applied similar ideas to daylight design, by developing a knowledge-based system that relates performance enhancements with design changes. More recently, Kulcke [46] explored recommender systems' ideas to create a design-bot that iteratively asks questions with the purpose of enhancing self-communication, and, thus, clarifying architect's intents. Both works depend on the knowledge that they were initially programmed with, thus, suffering from contextual limitations, making them unable to learn from new unseen designs or to adapt to new trends. In fact, none of the works exploited the benefits of ML or the hidden patterns lying within each designer's past designs.

As a possible scenario of a recommender system's use in architectural design, consider the design of an art exhibition room. With a conversational interface, the architect would communicate his intention to maximize the daylight performance of the space using skylights. If the system had information about the existing metrics of daylight performance and about art exhibitions lighting requirements, it could ask the architect which daylight metric he would like to optimize and provide him with alternatives of different skylights types and positions.

Despite the advantages, there are also potential risks in integrating these approaches in architecture. Firstly, if we create an omniscient system, i.e., a system for which the knowledge-base includes the history of all users, it might assume that each concept means the same for every user, which might not always be true. Secondly, if we consider a system solely based on a single user past experience, it might end up not providing many ideas, thus becoming less generative. Finally, if there is no user feedback, the system stagnates. To overcome these problems, it is necessary to continuously feed the system with new information.

Style Transfer

Style transfer is a recent advance in computer vision that can benefit the architectural practice. Previous work suggested working with ANNs to apply different design styles to buildings [15], using a technique known as neural style transfer [25, 47]. Neural style transfer use Convolutional Neural Networks (CNNs) to obtain the representation of the style of an input image. The work presented in [25] uses two different source images to generate images that mix content and style representation. By preserving the composition of the original image and combining it with the colors and local structures of the image containing the style to transfer, this approach balances content and style. The CNN is trained in an object recognition setting to derive the necessary neural representations. Other studies explored alternative style transfer applications, such as representing characters in different handwritings or smooth transfer style in videos [48-51].

Recent work discusses the application of these techniques in architecture. In [15] the authors speculate about the existing parallelism between ML approaches and the abductive reasoning process employed by humans. In [52] they also hypothesize having AI and ML generating architectural designs, by emulating the human mental process that articulates non-formal grammars in the development of architectural concepts. The authors attempt to extract rules from pictures using CNNs to then recombine them in order to generate new designs. The main limitation of this approach is that it only considers images of buildings, rather than 3D models. Additionally, the application of CNNs to plain images might not result in different designs, but just in the same images with different textures. In an attempt to emulate the creative process, [52] explored *DeepDream's* [26] generative capabilities to instill imagination into the approach. However, this process does not capture the 3D structures presented in the input images. This might be a result of the past observations/experiences used to train the CNN and we argue that if we trained it with a more accurate 3D representation, it might generate more feasible designs.

In architecture, style transfer techniques can also be very useful to address urban planning and, particularly, mass customization problems [53-54]. By using neural style transfer approaches, one can transfer styles of other buildings or other architects, instantiating stylistic variations. Thus, we can achieve mass customization by generating different houses in the same style, provided that we have (1) a representation of the building style we wish to use, (2) a set of base designs to which we ought to apply different style variations, and a (3) previously trained neural style transfer model.

To explore style transfer techniques in the context of architecture, we can devise at least two approaches which greatly differ in the representation of designs, hence affecting their implementation. The first one uses mere sequences of images to represent designs, whilst the second one uses geometrical properties to represent designs. Considering the first approach, the proposed representation resembles sequences of frames, thus allowing the application of techniques similar to those used in video style transfer [51]. In this approach, the architect provides a set of images that defines a style, as well as the images of the designs to which the style will be applied. Note that this approach might have the same limitations reported in [52] regarding 3D geometry.

The second approach requires more cohesive vector representations of design models, as several ML techniques rely on mathematical representations to capture and generalize data patterns. The first step is finding a tangible design representation, e.g., based on the detailed description of faces, edges, or vertices. The next step is the creation of large training data sets. Similarly, to [25], we envision training an ML model capable of becoming invariant to differences in design. This means that the data set must contain heterogeneous associations between designs, elements, and concepts, so that the ML model can capture the global contents of a base design and to apply locally stylized elements to these designs.

Considering the techniques available nowadays, the composition of multiple different styles might be problematic, as existing techniques do not yet support this feature seamlessly. An alternative is to sequentially apply different style transfer operations, i.e., chaining the output of the first style transfer operation to the input of the second style transfer operation. Moreover, it is important that architects remain critical towards the outputs, for it is possible that an erroneous usage of such system yields unreasonable design variations.

Image-to-design Synthesis

Based on the ideas behind text-to-image synthesis, architects can generate a design model satisfying the formal expression illustrated in a given image. This partially resembles neural style transfer in the sense that there is an image from which to collect intrinsic structures and properties to produce a design. Recent work uses Self-Organizing Maps (SOMs) to analyze 3D models, capturing formal features to then produce new models [55].

An ML model that generates designs from a single picture is similar to object recognition and image segmentation combined with machine translation. The depicted shapes in the picture must be captured, e.g., by CNNs. These have already proven to be very successful in object recognition and instance segmentation tasks. Additionally, it is also important to capture the conceptual ideas and the style patterns represented in the picture, for which we might use style transfer systems.

To avoid generating designs that differ only in their textures, as in [55], we envision a "*shape-to-geometry"* translator to map the captured shapes to appropriate geometrical representations. Then, it is possible to use ML generative abilities to instantiate different designs.

Clustering and Dimensionality Reduction

In earlier design stages, model categorization might enable a faster exploration of the design space. By providing an approximation of similar but disparate design variations, one is able to quickly verify if certain models better fit the desired intentions. Unsupervised learning techniques can be used to explore larger regions of the design space and, according to given criteria, group them in distinct categories. Also, in the case, of high-dimension design spaces, these techniques might be used to reduce their dimensionality and thus enable the visualization of simpler variants of designs and allow a smoother, broader, and quicker exploration.

These techniques can be very useful in bottom-up approaches, by suggesting different variations of what an architect is currently designing, thus locally expanding the architect´s perception of the design solution space. Moreover, assuming the existence of mechanisms capable of creating geometry from textual descriptions, this approach can also benefit top-down design procedures,.

3.2 Algorithmization

Algorithmic approaches to architectural design (i.e., Algorithmic Design (AD) [56]) are becoming increasingly popular among architects [57]. By being algorithmic, ML techniques easily integrate with AD. In this section, we discuss possible integration scenarios.

Program Synthesis, Code Completion

One of the hardest AD tasks is the development of the program that generates the intended design. This task requires not only generic knowledge about algorithms, programming languages, and software engineering but also an understanding of the Application Programming Interfaces (APIs) of the modeling tools being used. In fact, searching for information regarding how to use APIs consumes a considerable fraction of the time designers spend programming. Obviously, this task can be significantly improved if we know an expert that can point us in the correct direction. Unfortunately, such expert is not usually available at all times. More recently, developments on the field of ML seem to suggest the possibility of having a mechanized "*API Expert*".

The dream of having such an assistant helping us developing our programs is not new. Initial AI approaches explored the deductive capabilities of logic-based systems yielding successful results for tasks, such as automatically asserting the correctness of a program, or finding test cases that demonstrated the presence of errors. However, these approaches proved to be incapable of solving the mechanized assistant problem, i.e., the desired intelligent assistant that would help a programmer write, debug, and evolve software [58]. Two developments made this elusive goal much more accessible. Firstly, computers are orders-of-magnitude more powerful now. Secondly, huge amounts of software written by millions of programmers are now publicly accessible in repository sites such as github.com. As a result, for each problem a programmer needs to solve, there is a high probability that another programmer had very similar issues and the implemented solution is available somewhere. To complement this treasure trove of information, there is also the public Question and Answering (Q&A) repositories, such as stackoverflow.com, containing tens of millions of questions and, for each, a multiplicity of answers graded by the community.

All these developments are now being used for program synthesis research [59-61]. ML applications are currently mining code and Q&A repositories, identifying programming patterns, bug fixes, and the best answers for each question. While it is not yet possible to have an automatic programmer capable of inventing a new piece of software, it is now definitely possible to have one that helps us in the development of a program. AD can greatly benefit from such applications, particularly, (1) to predict which API operation best fits a given need [62], (2) to learn algorithms that solve particular problems [63], (3) to synthesize geometry constructions [64], (4) to identify bugs [65-66], and (5) to fix them [67-68]. Despite current limitations, it is expectable that these capabilities will increase in scope and precision and that, in a not very distant future, ML will enable different forms of automatic programming for AD.

Intelligent Domain Specific Languages

Another interesting application of ML in AD is the creation of Domain Specific Languages (DSLs) [69-70]. Contrary to general-purpose languages, these are languages designed to excel at addressing a very narrow field of application [71]. In the case of architecture, it makes sense to have DSLs whose primitives encode knowledge about construction rules and regulations. The creation of such primitives requires the analysis of several building regulations, as well as of previous positive and negative designs, i.e., designs that comply with the regulations and others that do not. By applying ML, we can extract patterns to instantiate heuristics for the DSL described above. The usage of such approach could provide immediate feedback by evidencing whether the current AD program is producing designs that do not comply with regulations. This idea could be further extended with a recommender system, that would suggest how to modify the current program to generate a similar but compliant design solution. As an example, consider a modeling primitive to create a wall. An ML-based system can embed information about the minimum requirements for thermal resistance (U-factor) given the project's location, and, consequently, generate a default wall assembly that complies with those requirements. Moreover, the same system can learn from the dataset to either suggest walls with a high-performing U-factor or provide feedback on different wall-types, their U-factor performance, and the consequent impact in overall building energy performance.

Anticipating Next Intentions and Making Suggestions
ML has been very successful in prediction tasks. One example is auto-completion of text. Gmail, for example, after learning from millions of email texts is able to complete common sentences. We argue that it is just a matter of time until we see the application of similar ideas to AD, particularly in the prediction of programming intentions [72-73]. ML techniques, such as ANNs or SVMs, could predict, based on past coding actions, what fragment of code might be written next. To further boost its potential, this mechanism could be coupled with previews of design variants that result from following the application of the suggested code blocks. Similarly, the same techniques are applicable to modeling scenarios. In this case, a digital assistant trained on a design corpus containing a large set of well-defined examples, suggests possible model completions.

3.3 Modeling

ML has the potential to enhance design activities related to the generation of different types of building models, particularly, analytical models for building performance simulation and 3D models for building visualization and documentation.

Analytical Modeling - Simplification
Recent environmental concerns have led governments to establish strict requirements about indoor environmental quality and the energy efficiency of buildings. Additionally, clients and designers often aspire for designs that can convey the desired spatial and structural idea with the minimal use of resources. As a result, the architectural community started to adopt performance-based approaches. However, building performance evaluation usually demands time-consuming simulations. A common approach to reduce the simulation time relies on the simplification of the model's geometry, e.g., replacing complex curves and surfaces with simplified counterparts with reduced level of detail. However, this approach is also time-consuming for the designer and, thus, it should only be applied when and where it is beneficial, which might not be obvious. Moreover, it is typically the case that only part of a large model needs to be evaluated but it might be difficult to identify which one better captures the performance of the whole. Finally, when the estimated performance does not satisfy the imposed requirements, there is usually no guidance regarding which aspects should change to attain more efficient designs.

There have been attempts to use AI techniques to solve these problems, particularly, rule-based expert systems [44]. Although they have proven to have advantages, these approaches are still confined to their specific context and, as a result, they have problems, e.g., expertise in a single domain, fixed knowledge-base that does not improve over time, and poor performance in face of an unseen model. The major problem of these expert systems is their inability to autonomously learn from external data and, therefore, we foresee that ML methods will be more successful. In a sense, supervised ML techniques resemble expert systems since they also require the *a priori* collection of data. However, the key difference is in how they acquire their knowledge, namely, the former ones are usually based on human-written *if-then* rules and are programmed to act in well-known situations, whereas the latter ones learn through observation of examples and, thus, can more easily adapt to new situations. We argue that by gathering enough data about successful and unsuccessful simplifications that were applied to different models, ML methods can infer patterns and correlations in the collected data, therefore predicting which simplifications to apply in future cases. The proposed simplifications might not be the best possible, but even a smaller simplification might lead to considerable time savings in the context of performance analysis. Similarly, we foresee the use of ML to learn to identify which parts of a model to use for performance analysis.

Although the ideas previously discussed might seem speculative, they have already begun to be explored, namely the segmentation of structural elements such as slabs, columns, beams, among others [74]. Similar ideas have been discussed for the automatic extraction of geometry from satellite images [75], which facilitate urban modeling and planning tasks.

3D Modeling - Production and Labeling

Current literature already addresses ML techniques that enhance the production and labeling of 3D models [74-76]. In this paper, we also mentioned other applications of ML that have the ability to produce them from concepts or even images. In fact, the work presented in [76] explores a similar idea where bayesian networks are used to automatically generate residential floors. After cataloguing 120 architectural programs, the authors trained a network to infer the implicit semantic relations between different spaces, e.g., between a dining room and a kitchen or between a bedroom and a bathroom. After generating the floor plan of a residence, they produce the 3D model based on predefined style templates, encoding the geometric and material properties of every building element. We envisage that style transfer could improve this process by allowing the use of building images instead of predefined templates.

Finally, if one can represent 3D models in terms of their descriptive features, then the same ML techniques used in computer vision to produce photorealistic images [28, 77-78] might be able to infer patterns of 3D modelling. In this case, through generative models like GANs [27], ML might produce feasible building designs [28, 77-78].

3.4 Optimization

Despite being an optional stage in the design workflow, optimization is becoming more common in the architectural practice. This is due to the demand for sustainability and efficient use of resources in the built environment, as well as by the emergence of different ready-to-use toolsets [79].

In order to optimize a design, it is necessary to implement a parametric model of the building. Algorithmic-based approaches [80] are useful to produce that model and the corresponding analytical models used to assess the design's performance, a key factor in the optimization process. Depending on the optimization goals, different analysis tools perform simulations that evaluate a given analytical model. Hence, an optimization algorithm is responsible for generating different values for the design parameters and, based on the performance evaluations results, guide the search towards more satisfactory designs. In this context, black-box optimization algorithms are frequently used, since the performance metrics are not obtained through analytical means, but rather through simulations [82].

Approximating Functions and Dimensionality Reduction

Historically, the first black-box optimization algorithms were deterministic and sequential, iterating the design space, step-by-step, over one or more dimensions. Although these algorithms exhibit good convergence properties, their performance tend to decrease with the increase of the number of decision variables, and they cannot simultaneously optimize multiple objectives [81-82]. Stochastic algorithms address some of these issues. Their randomness, as well as the evolutionary metaphors that some of them use, provides the flexibility to tackle both single and multiple objective optimization problems, producing fairly good results as long as there is enough time to run thousands of evaluations [82]. However, these algorithms quickly become infeasible when dealing with time-consuming evaluation functions, which typically occur in architectural design problems. The time premise now becomes a bottleneck, hindering the convergence of these algorithms and their utility [82].

More recently, an approach based on the creation of approximate models of the costly evaluation functions is gaining popularity [83-84]. By replacing the costly functions by cheaper but valid surrogate models, this approach, called Surrogate-Based Optimization (SBO), reduces the number of costly evaluations, therefore also reducing the total optimization time [84]. The surrogate model is then explored by optimization strategies to find the most promising solutions, which are then evaluated using the original and costly evaluation function. The result of this evaluation is then used to iteratively update the surrogate model.

One problem with this approach is the amount of data required to create a good initial surrogate model. As previously mentioned, ML techniques can generalize functions from large sets of data and, consequently, are particularly suitable for SBO. ANNs, SVMs, Gaussian Processes (GPs) and Radial Basis Functions (RBFs) are examples of such techniques [82-85], some of which are available as ready-to-use tools (e.g., Opossum [83]). To accurately and unbiasedly approximate a function these techniques require sufficient, independent, and identically distributed data. However, given that the architectural practice is often constrained by tight evaluation budgets, it might not be possible to obtain a large dataset. In that case, more complex techniques such as ANNs and SVMs might not be able to successfully generalize/approximate the evaluation function. To overcome the data size restriction, one should opt for Bayesian Optimization techniques, such as GPs, which are built upon probabilities and are shown to work well with smaller data sets. In addition to performance, Bayesian approaches provide clearer results and are usually more intelligible than, for example, ANNs which obscure the correlations among variables in their nonlinear structures [86].

Notwithstanding its impact in SBO, ML has also been useful to improve other aspects of optimization algorithms, particularly the expected increase of their running time with the number of variables and objectives. Depending on the designer's knowledge about the problem, it is not uncommon for the problem description to have redundant variables, either because some of them have little impact on the evaluation function or because there are some hidden correlations among the variables. This redundancy is problematic, since the algorithm might focus on exploring design solutions that do not significantly improve the results. ML techniques can minimize the impact of such redundancy by identifying (1) the most important variables and their hidden correlations, and (2) similar design solutions.

In the first case, by applying ML dimensionality reduction techniques, such as PCA and LDA, we can produce a smaller equivalent problem description that encodes the hidden relations between design variables and excludes variables that are useless to the optimization process. This contributes not only to the overall reduction of the total optimization time, but also to the algorithm's efficiency, since it can now focus on exploiting meaningful and non-redundant parameter configurations.

In the second case, clustering techniques such as K-means, or Gaussian Mixture models, are particularly useful in earlier stages of population-based optimization algorithms, e.g., Genetic Algorithm (GA) or Particle Swarm Optimization (PSO), which involve evaluating more than one candidate solution per iteration. Most of these algorithms generate candidates randomly and sometimes end up exploring nearby solutions which are not representative of the whole solution space. [87] applied K-means to group a set of solutions in distinct clusters, from which only one solution was submitted to the costly evaluation function. Similarly, [88] used the K-means clustering technique to guarantee a distribution of surrogates across the design space.

Improving Optimization

One of the difficulties of performance-based design is the identification and fine-tuning of the best optimization algorithms for a given design problem. Indeed, the No Free Lunch Theorems (NFLTs) [89] state that there are no optimization algorithms that excel in every problem. Instead, there are algorithms that are very effective in some problems and less so in others. One way to improve an algorithm's efficiency for particular problems is to fine-tune its parameters, a process known as Hyperparameter Optimization.

By coupling ML techniques to the optimization process, it is possible to learn which parameter values yield better algorithmic efficiency for a given problem and, thus, adaptively optimize its configuration. To achieve this, it is necessary to gather a dataset comprised of design problems, algorithms' configurations, and their corresponding efficiency. Subsequently, supervised ML techniques, such as ANNs, Bayesian Networks, SVMs, Decision Trees, or Random Forests, can infer hidden relations and try to adjust each algorithm's configurations at runtime. Another ML approach is Ensemble Learning, which is becoming more frequent [87, 90]. This approach uses a set of learning algorithms to obtain better predictive performance than that obtained from each algorithm individually.

Optimization in design often criticized for alienating the architect from the process. Das [12] addressed this problem by combining a GA with a set of rules to build an iterative optimization process such that, in each iteration, the user selects the design variants that satisfy him the most, thus making the subsequent design variants closer to the user's intent. Given a large dataset, one could use ANNs to learn the user preferences and generate ready-to-use design variations for future projects. One other application of these ANNs would be to extend the proposed approach by Das and use this knowledge to predict what would be the design a few iterations from now.

In conclusion, although the present work merely focuses in the application of ML to architecture, the extent of its applications is vast. A more extensive review of the literature will evidence the application of ML to several other fields of architecture, such as, the design of canopies that autonomously react to human presence [91], the development of adaptive façade control systems [92], and the creation of materials with certain properties [93].

4 Conclusion

Given the increasing impact of ML in human activities, it is expectable that it will also affect architecture. In this paper, we described multiple alternatives for the use of ML in building design, particularly, in conceptualization, algorithmization, modeling, and optimization tasks. We also illustrated ML's considerable potential for helping architects in their conceptual endeavors, with applications that range from recommender systems to style transfer and other techniques. Similarly, ML could benefit algorithm development in the context of AD, particularly in code completion, code generation, program synthesis, and domain-specific language design. Modeling tasks can also take advantage of ML approaches, either in the simplification of building simulation models or in the automatic generation of design solutions based on images or textual descriptions of buildings. Considering optimization tasks, ML uses include evaluation function approximation, dimensionality reduction, and hyperparameter optimization.

Unfortunately, ML systems still have several limitations. One is that they require well-balanced training sets, which are not yet easily available in architecture. Indeed, if provided with an unbalanced dataset, ML learners might become biased towards specific patterns and become poor generalizers. Another difficulty is the large amount of data necessary to ensure good performance of any data-driven ML technique. The application of ML in architecture is still in its early beginnings and, naturally, the data sets either do not exist or are immature. Recent research in transfer learning [24] aims to bridge this gap, by exploiting the knowledge previously obtained for similar tasks to diminish the impacts of data deficit.

A final but important drawback of some ML approaches is their limited explanation capabilities, which might prevent architects from understanding the outcomes. In this scenario, architects may take decisions based on information that they do not fully understand. The lack of explainability also troubled legislators, causing them to demand that certain systems, e.g., search engines, recommender systems, and medical diagnosis systems, must provide an explanation of their decisions [94]. Due to the legal implications of having humans blindly taking decisions that were suggested by machines, it is only a matter of time until this law extends to other disciplinary fields, including Architecture.

To wrap-up, we look back to our initial questions: Will machines replace humans in architectural practice? Will these methods limit the architect's creative control? Will we have architecture without architects?

In our opinion, we can address these questions under two perspectives. Firstly, in architecture, the pursuit of beauty has always been paramount in fueling creativity. However, the assessment of beauty is culturally-based and shifts over time, therefore, it is hard to capture by current ML techniques. In this regard, we are still far away from a complete replacement of human architects by architect-bots. Secondly, disruptive changes are always difficult to handle, but history shows that those who master them prevail. In the past, the shift from manual technical drawing to Computer-Aided Design was perceived by some as a vehicle for the destruction of architecture. However, architects adapted, mastered it, and now take full advantage of it. Similarly, rather than distrust ML, architects should embrace it like they did with other paradigm shifts, and then steer it to their advantage.

Acknowledgements. This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with references UID/CEC/50021/2019, PTDC/ART-DAQ/31061/2017, and SFRH/BD/98658/2013.

References

1. Bishop, C. M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc., 2006.
2. Magoulas, G. D.: Machine Learning and Its Applications. (Vol. 2049; G. Paliouras, V. Karkaletsis, & C. D. Spyropoulos, Eds.) (No. September 2015). Berlin, Heidelberg: Springer Berlin Heidelberg, 2001.
3. Magoulas, G. D., & Prentza, A.: Machine Learning in Medical Applications, 300–307, 2001.
4. Deo, R. C.: Machine Learning in Medicine, 2015.
5. Ferreira, D. R.: Applications of Deep Learning to Nuclear Fusion Research, 2018.
6. Bolton, R. J., & Hand, D. J.: Statistical Fraud Detection: A review. Statistical Science, 17(3), 235–255, 2015.
7. Behera, R. N., & Das, K.: A Survey on Machine Learning: Concept, Algorithm and Applications. International Journal of Innovative Research in Computer and Communication Engineering, 5(2), 2017.
8. Khean, N., Fabbri, A., & Haeusler, M. H.: Learning Machine Learning as an Architect, How to?. 2018.
9. Sjoberg, C., Charlotte, U., Beorkrem, C., & Ellinger, J.: Emergent Syntax: Machine Learning for the Curation of Design Solution Space. Proceedings of the 37th ACADIA, 552–561, 2017.
10. Racec, C. E., Budulan, S., Vellido, A., & Polit, U.: Computational Intelligence in architectural and interior design : a state-of-the-art and outlook on the field, 2016
11. Merrell, P., Schkufza, E., & Koltun, V.: Computer-generated residential building layouts. ACM SIGGRAPH Asia 2010, 10, 1, 2010.
12. Das, S.: Interactive Artificial Life Based Systems, Augmenting Design Generation and Evaluation by Embedding Expert A Human Machine dialogue for form finding. Proceedings of the 36th ECAADe, 2018.

13. Cudzik, J., & Radziszewski, K.: Artificial Intelligence Aided Architectural Design, 1, 77–84, 2018.
14. Tamke, M., Nicholas, P., & Zwierzycki, M.: Machine learning for architectural design: Practices and infrastructure. International Journal of Architectural Computing, 16(2), 123–143, 2018.
15. Steinfeld, K.: Dreams May Come. In Acadia 2017, 590–599, 2017.
16. Turing, M.: I.-Computing Machinery and Intelligence. Mind, LIX(236), 433–460, 1950.
17. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. Nature 529, 484, 2016.
18. Moravcík, M., Schmid, M., Burch, N., Lisý, V., Morrill, D., Bard, N., & Bowling, M.: DeepStack: Expert-Level Artificial Intelligence in Heads-Up No-Limit Poker, 1–37, 2017.
19. Ferrucci, D.: Introduction to "This is Watson". IBM Journal of Research and Development 56(3.4), 2012.
20. Simonyan, K., Vedaldi, A., & Zisserman, A.: Deep Inside Convolutional Networks: Visualizing Image Classification Models and Saliency Maps, 1-8, 2013.
21. He, K., Zhang, X., Ren, S., & Sun, J.: Deep Residual Learning for Image Recognition, IEEE Conference on CVPR, 770-778, 2016.
22. Hastie, T., Tibshirani, R., & Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. York, Springer-Verlag New, 2009.
23. Mohri, M., Rostamizadeh, A., & Talwalkar, A.: Foundations of Machine Learning, 2012.
24. Torrey, L., & Shavlik, J.: Transfer Learning, 2009.
25. Gatys, L. A., Ecker, A. S., & Bethge, M.: A Neural Algorithm of Artistic Style, 2015.
26. Mordvintsev, A., Olah, C., & Tyka, M.: Inceptionism. Available at http://googleresearch.blogspot.com/2015/06/inceptionism-going-deeper-into- neural.html, 2015.
27. Goodfellow, I. J., Pouget-abadie, J., Mirza, M., Xu, B., & Warde-farley, D.: Generative Adversarial Nets. In Proceedings of the 27th International Conference on Neural Information Processing Systems, 2014.
28. Ledig, C., Theis, L., Huszar, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A.P., Tejani, A., Totz, J., Wang, Z., & Shi, W: Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. 2017 IEEE CVPR, 105-114, 2017.
29. Claus L. Cramer-Petersen, Bo T. Christensen, Saeema Ahmed-Kristensen: Empirically analysing design reasoning patterns: Abductive-deductive reasoning patterns dominate design idea generation, Design Studies, 60, 39-70, 2019.
30. Neto, J. L., Freitas, A. A., & Kaestner, C. A. A.: Automatic Text Summarization Using a Machine Learning Approach A Review of Text Summarization, 205–215, 2002.
31. Memisevic, R.: On Using Very Large Target Vocabulary for Neural Machine Translation, 2014.
32. Aggarwal, C. C., & Zhai, C. X.: Mining Text Data, 2012.
33. Joachims, T.: Text Categorization with Support Vector Machines : Learning with Many Relevant Features. 2–7, 2005.
34. Sebastiani, F.: Machine Learning in Automated Text Categorization. 34(1), 1–47, 2002.
35. Yang, Y., & Liu, X.:. A re-examination of text categorization methods. 1999.
36. Damerau, F., & Weiss, M.: Automated Learning of Decision Rules for Text Categorization. 12, 1994.
37. Blei, D. M., Ng, A. Y., & Jordan, M. I.: Latent Dirichlet Allocation, 3, 993–1022, 2003.
38. Reed, S., Akata, Z., Yan, X., & Logeswaran, L.: Generative Adversarial Text to Image Synthesis. 2016.

39. Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., & Metaxas, D.: StackGAN : Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks. 5907–5915, 2017.
40. Reed, S., Akata, Z., Mohan, S., Tenka, S., Schiele, B., & Lee, H.: Learning what and where to draw. In NIPS, 2016.
41. Sivic, J, & Zisserman, A.: Video Google: a text retrieval approach to object matching in videos. Proceedings 9th IEEE ICCV, 2, 1470-1479, 2003.
42. Ricci, F., Rokach, L., & Shapira, B.: Recommender Systems Handbook. 2011.
43. Christakopoulou, K., Radlinski, F., & Hofmann, K.: Towards Conversational Recommender Systems. Proceedings of the 22nd ACM International Conference on Knowledge Discovery and Data Mining, 16(3), 815–824, 2016.
44. Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T.:. Evaluating Collaborative Filtering Recommender Systems, 22(1), 2004.
45. Gagne, J. M. L., Andersen, M., & Norford, L. K.: An interactive expert system for daylighting design exploration. Building and Environment, 46(11), 2351–2364, 2011.
46. Kulcke, M.: Design-Bot - Using Half-Automated Qualitative Interviews as Part of Self Communication within the Design Process, 1(Schumacher), 103–108, 2018.
47. Jing, Y., Yang, Y., Feng, Z., Ye, J., & Song, M.: Neural Style Transfer: A Review, 2017.
48. Tenenbaum, J., & Freeman, W.: Separating Style and Content with Bilinear Models. 2000.
49. Elgammal, A., & Lee, C.: Separating Style and Content on a Nonlinear Manifold. 2004.
50. Tenenbaum, J. B., & Freeman, W. T.: Separating Style and Content. 1997.
51. Ruder, M., Dosovitskiy, A., & Brox, T.: Artistic style transfer for videos, 2016.
52. Silvestre, J., & Ikeda, Y.: Artificial imagination of architecture with deep convolutional neural network, 881–890, 2016.
53. Duarte, J., & Caldas, L.: An Urban Grammar For The Medina of Marrakech Design Computing and Cognition. 2006.
54. Duarte, J. P.: Customizing mass housing: a discursive grammar for Siza's Malagueira houses. Architecture, 14(2), 2001.
55. Algeciras-Rodriguez, J.: Stochastic Hybrids: From references to design options through Self-Organizing Maps, 1, 119–128, 2018.
56. Castelo-Branco, R., & Leitão, A. M.: Integrated Algorithmic Design: A single-script approach for multiple design tasks. Proceedings of the 35th ECAADe, 1, 729–738, 2017.
57. Peters, B.: Computation Works: The Building of Algorithmic Thought, Architectural Magazine, Computation Works: The Building of Algorithmic Thought, 222, 8-16, 2013.
58. Shrobe, H., Katz, B., & Davis, R.: Towards a Programmer's Apprentice (Again). Proceedings of the 29th AAAI Conference on Artificial Intelligence, 2015.
59. Feng, Y., Martins, R., Van Geffen, J., Dillig, I., & Chaudhuri, S.: Component-Based Synthesis of Table Consolidation and Transformation Tasks from Examples. In Programming Language Design and Implementation, 422–436, 2017.
60. Feng, Y., Martins, R., Bastani, O., & Dillig, I.: Program Synthesis using Conflict-Driven Learning. In Programming Language Design and Implementation, 2018.
61. Polosukhin, I., & Skidanov, A.: Neural Program Search: Solving Programming Tasks from Description and Examples. 1–11, 2018.
62. Raychev, V., Vechev, M., and Yahav, E.: Code completion with statistical language models. ACM SIGPLAN Notices, 49(6), 2014.
63. Kaiser, L., & Sutskever I.: Neural gpus learn algorithms, 2015.
64. Gulwani, S., Korthikanti, V., & Tiwari, A.: Synthesizing geometry constructions.ACM SIGPLAN Notices, 46(6), 2011.
65. Malhotra, R.: A systematic review of machine learning techniques for software fault prediction. Applied Soft Computing, 27, 504-518, 2015.
66. Pradel, M., & Sen, K.: DeepBugs: A Learning Approach to Name-based Bug Detection. 2018.

67. Le Goues, C., Dewey-Vogt, M., Forrest, S., & Weimer, W.: A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. Software Engineering (ICSE), 34th International Conference on IEEE, 2012.
68. Long, F., & Rinard, M.: Automatic Patch Generation by Learning Correct Code POPL, 2015.
69. Wang, X., Anderson, G., Dillig, I., & Mcmillan, K.: Learning Abstractions for Program Synthesis, 2018.
70. Freitag, D.: Machine Learning for Information Extraction in Informal Domains. Machine Learning, 39, 169-202, 2000.
71. Fowler, M.: Domain-Specific Languages. Addison-Wesley Professional, 2010.
72. Nguyen, A. T., Nguyen, T. T, Nguyen, H. A, Tamrawi, A., Nguyen, H. V., Jafar, A., & Nguyen, T. N.: Graph-based pattern-oriented, context-sensitive source code completion, 34th International Conference on Software Engineering (ICSE), Zurich, 2012, 69-79, 2012.
73. Liu, C., Wang, X., Shin, R., Gonzalez, J. E., & Song, D.: Neural Code Completion. In International Conference for Learning Representations, 1-14, 2017.
74. Yetis, G., Yetkin, O., Moon, K., & Kiliç, O.: A Novel Approach for Classification of Structural Elements in a 3D Model by Supervised Learning, 1, 129-136, 2018.
75. Kobayashi, Y., & Terzidis, K.: Extracting the Geometry of Buildings from Satellite Images: Using Fuzzy Multiple Layer Perceptrons, 1999.
76. Merrell, P., Schkufza, E., & Koltun, V.: Computer-generated residential building layouts. ACM SIGGRAPH Asia 2010 Papers on - SIGGRAPH ASIA '10, 1, 2010.
77. Zhang, H., Xu, T., Li, H., & Aug, C. V.: StackGAN : Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks, 2017.
78. V. d. Oord, A., Kalchbrenner, N., & Kavukcuoglu, K.: Pixel Recurrent Neural Networks, 2016.
79. Cichocka, J. M., Browne, W. N., and Rodriguez, E.: Optimization in the Architectural Practice. Proceedings of the 22nd International Conference of the Association for Computer-Aided Architectural Design Research in Asia, 387-397, 2017
80. Aguiar, R., Cardoso, C., & Leitão, A.:. Algorithmic Design and Analysis Fusing Disciplines, 28–37, 2017.
81. Kolda, T.G., Lewis, R.M., & Torczon, V.: Optimization by Direct Search- New Perspectives on Some Classical and Modern Methods. Society for Industrial and Applied Mathematics 45(3), 385-482, 2003.
82. Wortmann, T., & Nannicini, G.: Black-box optimization methods for architectural design. Proceedings of the 21st International CAADRIA Conference, 177-186, 2016.
83. Wortmann, T.: Opossum. Proceedings of the 22nd International Conference of the Association for Computer-Aided Architectural Design Research in Asia, 283–292, 2017.
84. Koziel, S., Ciaurri, D. E., & Leifsson, L.: Chapter 3 Surrogate-Based Methods. In Computational optimization, methods and algorithms 356, 2011.
85. Díaz-Manríquez, A., Toscano, G., Barron-Zambrano, J., & Tello-Leal, E.: A review of surrogate-assisted multiobjective evolutionary algorithms. Computational Intelligence and Neuroscience, 2016.
86. Bishop, C. M. Bayesian methods for neural networks. Machine Learning, 7(1), 1–11, 1995
87. Jin, Y., & Sendhoff, B.: Reducing Fitness Evaluations Using Clustering Techniques and Neural Network Ensembles, 688–699, 2004.
88. Isaacs, A., Ray, T., & Smith, W.: An evolutionary algorithm with spatially distributed surrogates for multiobjective optimization. Proceedings of the 3rd ACAL, 257–268, 2007
89. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1(1), 1997.
90. Brown, G., Wyatt, J.L., Tiňo, P.: Managing diversity in regression ensembles. The Journal of Machine Learning Research 6, 1621–1650, 2005.

91. Beesley, P., Bouron, G., Gorbet, R., Nicholas, P., & Zwierzycki, M.: Hybrid Sentient Canopy. 2016.
92. Smith, S., & Lasch, C.: Machine Learning Integration for Adaptive Building Envelopes. ACADIA, 2016.
93. Luo, D., Wang, J., & Xu, W.: Applied Automatic Machine Learning Process for Material Computation. 1, 109–118, 2018.
94. GoodMan, B., & Flaxman, S.: European Union regulations on algorithmic decision-making and a right to explanation. 1-9, 2016.