Predicting semantic building information (BIM) with Recurrent Neural Networks

B. Mete & J. Bielski & PhD C. Langenhan & Prof. F. Petzold
Technical University of Munich, Germany

V. Eisenstadt & Prof. K.D. Althoff
German Research Center for artificial intelligence (DFKI) / University of Hildesheim

ABSTRACT: Recent advances in technology established artificial intelligence (AI) as a crucial domain of computer science for both industry and research, but also for everyday life. However, while computer-aided architectural design (CAAD) and digital semantic building models (BIM) became essential aspects of the contemporary architectural design process, AI cannot be seen as a leading supportive computational method due to its absence in the established design software and the challenging acquisition of proper data. An option to acquire rich design data, for example in the form of time slices and relations of atomic design steps, is the reproduction of design protocol studies (Lawson, 2004). However, this data is still unstructured and requires a framework for pre-processing and training artificial neural networks (ANN).

In this paper, we present our research on BIM and AI, dedicated to autocompletion of design steps for architectural design, based on the methods of the 'metis' projects. Autocompletion is achieved through the suggestion of further design steps to improve the quality and speed of the design process of the early design stages. It is inspired by other autocompletion methods that have been applied for data-driven decision-making.

Assuming the position of Lawson (Ibid.), we propose an approach for a recurrent neural network (RNN) model to predict future design steps through sequential learning. Thus, we propose a model based on cognitive sequences of the architectural design process as relational sequences (Lawson, 2004), using sketch data quantified through custom labelling via an open-source tool assigning the respective design phase (Lawson, 2004; Laseau, 2000). We adapt to the idiosyncrasies of the user by identifying the current cognitive processes to predict further mental activities and thus, future design steps.

1 INTRODUCTION

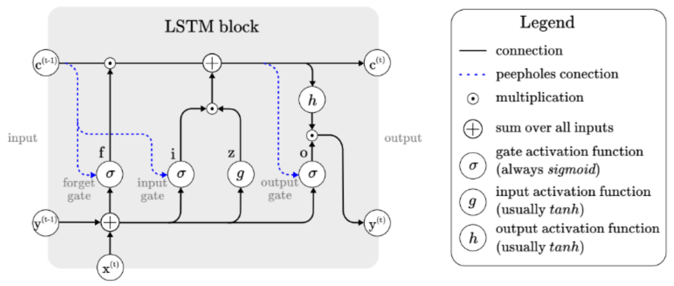
Within the last decades, the scientific field of artificial intelligence has been a rapidly growing field of research. Its methods have been further applied in various capacities and varieties in other areas, both for professional operators, e.g., recognition of abnormalities in X-Ray images, and casual everyday users, e.g., text auto-completion of digital keyboards on smartphones. AI methods have also been applied within the architecture, engineering and construction (AEC) industry, mostly focusing on optimization during the later stages of a construction process, e.g., financial, temporal and performance (Abioye et al. 2021). However, the architectural design process, especially the early stages, are rather untouched because of its complexity. The architectural design process contains a lot of meaningful information of geometrical and semantic nature. Semantic building models (BIM) predominantly formalize the result of a design process. In contrast, we are tracking the steps of the design process that lead to the final design result.

In this paper, we propose a novel Deep Learning (DL) approach as a design auto-completion method, to assist the architect during the design decision making process. The overall goal of the methods developed within the 'metis' projects is to predict the current and the future design phase, based on design process segmentation using design phases by Laseau (2000), Lawson (2005) and Barelkowski (2013), which is an extended version of the *Analysis, Synthesis and Evaluation* (ASE) model. Accessing the most recent changes in the design phase and the current status of the design phase, made tangible through the art of hand-drawn sketching (Lawson, 2004; Suwa & Tversky, 1997), the model is able to suggest possible continuations of the design direction. Furthermore, we aim to build an auto-completion pipeline that operates in real-time and suggest design steps to the architect, in which the user has an option to either reject or accept the proposition.

2 BACKGROUND AND RELATED WORK

AI assists humans in various domains of both professional and daily life. Having an intelligent assistant that will aid the architect during designing, is first

propounded by Negroponte (1973), which can both predict and suggest new ideas based on architectural design knowledge. First immediate obstacle for such approaches is to collect reproducible data. In order to overcome this drawback, sketch protocol studies have been used to trace the architect's way of thinking during a design process (Suwa & Tversky, 1997). As previous studies only produce qualitative data, a prototype tool was implemented and employed. It enables the development of quantitative results from retrospective sketch protocol studies, including custom categories for manual assigning (Bielski et al., 2022) Thus, both custom and common parameters, e.g., timing and pen pressure on a digital drawing board, can be included for analysing the design process through the art of sketching.
Furthermore, in order to process such information in the context of Machine Learning (ML), the data should be quantifiable, and categorizable. Lawson (2004) proposes both temporal and relational segments for classifying sketch protocols, while introducing the subclasses *Analysis, Synthesis, Evaluation and Communication* for defining the design phases. In addition to that, Laseau (2000) further partitions the *Synthesis* into two different subclasses as the *Exploration* and *Discovery*, while Barelkowski (2013) focuses on dividing the *Analysis* into *Knowing* and *Understanding* for a more distinguished look on the involved knowledge management. This also results in the separation of the *Evaluation* as a final decision, as well as a tool for creating more information as *Evaluation - (informing) Knowing*. Hence, the temporal categorization of the design decision making process can be enabled through the relational sequences of design phases. The state-of-art solution for such a categorization problem, is to make use of Artificial Neural Networks (ANNs).
Recurrent Neural Networks (RNNs) are a subset of ANNs that include loops, hence it considers several previous input values, while calculating the output. Therefore, the knowledge can persist in the network, allowing the network to come up with predictions within sequential data, such as time sequences. One example of such a learning problem in real life can be predicting stock prices by looking at the previous and the current stock values, or predicting the weather in the following days by having access to the recent weather forecast. In that regard, they distinguish themselves from the ANNs or basic *feed-forward neural networks* through integrating loop connections in order to include data from the past. However, since neural networks rely on *back-propagation* that utilizes *partial derivatives*, having a looped architecture with a long chain, can cause the *gradients* for the learning weights to either drastically increase, or shrink to 0. This phenomenon is called *Vanishing Gradient Problem*. Moreover, being vulnerable to the *Vanishing Gradient Problem*, RNNs are prone to failures while capturing the long-range correlations of sequential data (Hochreiter, 1998).
There are several neural network architectures, which are subsets of the RNNs that is able to overcome the *Vanishing Gradient Problem*, such as Gated Recurrent Units (GRU) (Cho et al., 2014) and Long-Short Term Memory (Hochreiter, 1997). However, LSTMs are much more widely used in the state-of-art networks, which facilitates the development and the maintenance of the project for further improvements. LSTMs include specific *gated cells*, illustrated in the Figure 1, that allow to store and/or remove parts of the previous information, which enables the model to improve the handling of the long-term dependencies within the data. Hence, LSTM architecture provides a more robust learning scheme for sequential data.


LSTM block
input
output
forget gate
input gate
output gate
f
i
z
o
Legend
connection
peepholes conection
multiplication
sum over all inputs
gate activation function (always *sigmoid*)
input activation function (usually *tanh*)
output activation function (usually *tanh*)
Figure 1. A vanilla LSTM cell that includes three different kinds of gates (Van Houdt et al., 2020).

3 APPROACH

In this section, we present the approach for our supervised learning pipeline. It includes the dataset, the pre-processing and augmentation of this data, the details of the proposed RNN architecture based on the cascaded sequential learning method and finally a learning criterion, resulting in an implemented custom loss function.

3.1 *Dataset*

The dataset consists of five different design processes made by architects that have been quantified through our open-source sketch protocol analyser tool (Bielski et al., 2022). Each design process data consists of a feature vector, a design phase, and a specific timestamp. All data instances have varying numbers of timestamps, ranging between *4.000* and *18.000* which spans across 15 minutes. Each data instance that corresponds to a specific timestamp includes a *feature vector*, and the design phase attached to it, which we will refer to as the label. The labels are unique, and have a value among our seven design phases (i.e. *Analysis-Knowing, Analysis-Understanding, Synthesis-Exploration, Synthesis-Discovery, Evaluation, Evaluation- Knowing, Communication*) that represent a more distinguished version of the phases (see Figure 2) of the common design model ASE (Analysis, Synthesis, Evaluation (Lawson, 2005). The feature vector consists of distinct information related to the design process for each timestamp. Namely, these information parameters are the pen pressure and geometric coordinates of the pen, gathered from a WACOM tablet used as a digital drawing board during the sketch protocol study, and the sketched elements (e.g. 'symbol', 'line') and objects (e.g. 'door', 'wall') that are present in the sketch at the respective timestamp.

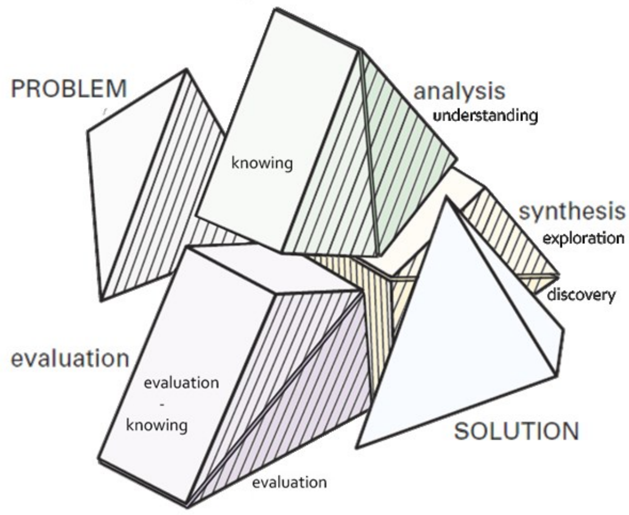
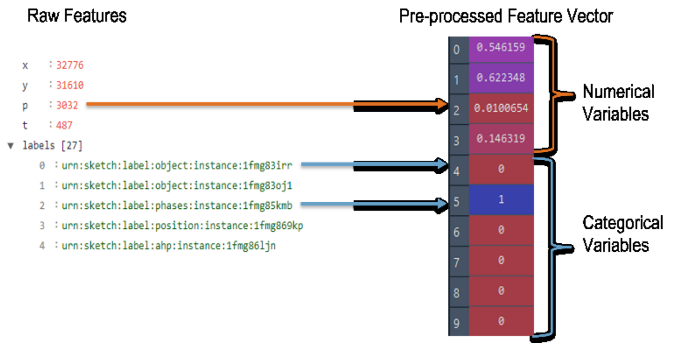


Figure 2. The design process as an extended version of the ASE model (Lawson, 2005).

An important step within our approach is the data preprocessing. Even though the data is quantified through the sketch protocol analyzer, there are still categorical values both in the feature vector and the labels that have to be encoded as numeric values in order for the Artificial Neural Network (ANN) to work with the data. As mentioned before, there are seven design phases (see above) for the labels, while each timestamp is being labeled by only one of them. In order to map that textual information, we are using a categorical data embedding technique called '*Dummy Variables'* (Draper & Smith, 1998). In our case*, dummy variables* create a vector with a length of 7 units, where each unique design phase category is attached to a unique vector element for every timestamp. The resulting vector is a *unit vector*, where only the value that corresponds to the current design phase is *1*, whilst all the remaining vector elements stay as *0*. Therefore, after the embedding, instead of having categorical values, the design phase is represented with seven unique *unit vectors*. For encoding the feature vector, a similar approach is being used, called 'Multi-Hot-Encoding', which is a generalized version of *dummy variables*. This approach is being used, since each categorical feature can appear more than once in the data, therefore it requires an integer variable rather than a binary variable.

The different features have varying ranges for their values, which can cause the network to be influenced more by the numerically large values. To be specific, while the encoded features have small integer values, continuous variables like the pen pressure can take values up to a million, which might prioritize the pen pressure value during the learning process. Thus, a final *normalization* routine must be performed. For this purpose, the last part of the pre-processing step is the L1 normalization, which maps the values of every feature between *0* and *1*. This results in changing the feature dataset into a common scale, without deforming the numeric relation between the values since it is a linear map. An exemplary conversion between raw data and the processed features is shown at Figure 3.



```
x   : 32776
y   : 31610
p   : 3032
t   : 487
▼ labels [27]
   0 : urn:sketch:label:object:instance:1fmg83irr
   1 : urn:sketch:label:object:instance:1fmg83oj1
   2 : urn:sketch:label:phases:instance:1fmg85kmb
   3 : urn:sketch:label:position:instance:1fmg869kp
   4 : urn:sketch:label:ahp:instance:1fmg86ljn
```

0	0.546159
1	0.622348
2	0.0100654
3	0.146319
4	0
5	1
6	0
7	0
8	0
9	0

Figure 3. An exemplary representation of the feature mapping. The values on the left shows the unprocessed features with both numeric and non-numeric values. The figure on the right, is the processed and the final version of the feature vector.

3.2 *Model*

As explained in the section 3.1 the training data includes *features* that are extracted from the design process data, and each timestamp is labelled with a unique design phase. Our model proposes a sequential prediction scheme, using a fixed number of timestamps as an input, and aims to predict the current and the next design phases as the output.

In order to overcome the shortcomings of common RNNs, our cascaded model consists of a chain of *"processing blocks",* which consists of an LSTM cell, followed by a *fully connected layer* and a *Sigmoid* activation function. Each *processing block* coincides with the features in one timestamp, and they produce an output for their corresponding timestamps. The overall model comprises a chain of *processing blocks*, and the length of this chain is defined with the parameter *"processing window size"*. The *processing window size* parameter is crucial, since we require a long enough chain that the model can capture the correlation between the features well, to predict the next design phase, but short enough that the training is *tractable* and *feasible*. The proposed processing window size value of this work is *50*. The LSTM cells accept a total number of *41* features and produce *21* output values. The *fully connected layers* accept *21* features and produce *7* output values. Those *7* output values are indicated with the parameter *output size*, and it refers to the probabilities for the predictions of all *7* design phase labels. Simply, the largest probability value is selected to be the model's prediction. The reason why a *fully connected layer* is added on top of an LSTM layer is to even capture extra correlations that the LSTM model itself fails to detect. The overall scheme of the learning model, consisting of the chain of '*processing blocks'*, is represented at Figure 4.

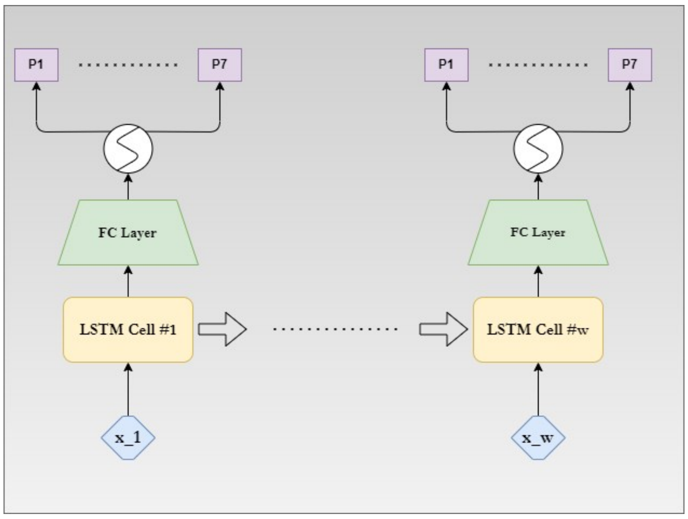


Figure 4. The model as a chain of *processing blocks*: Blue shapes represent the feature vector for each timestamp fed into the LSTM cells, yellow blocks represent an individual LSTM cell, green blocks the fully connected layers, white circle the S*igmoid* activation function, and finally the purple rectangles the probabilities attached for each design phase for each timestamp, while numerically the largest value eventually becomes the prediction of the model, and *w* represents the processing window length.

3.3 *Learning Criteria*

Each individual data set per sketch data contains more than 10000 timestamps, but only a handful of design phase changes. The biggest learning obstacle is to not *overfit* the model through the exceeding timestamps with rare to no changes. *Overfitting* is a common problem in ML (Ying, 2019), which arises when the network specifically learns the training data instead of the solution to a general and much wider problem. Along with achieving low accuracy during the evaluation, *an overfitted network* can be thought of as a memorizing *model*, instead of a *learning model*, therefore it should be avoided for a general ML problem. Instead, the most important functionality of the model is to be able to capture the internal dynamics of the periods where there is a design phase with better accuracy. Thus, in order to mitigate and optimize the learning process, we implemented a loss function as a learning constraint. The reason for this is, since a large portion of the design process data continues to stay at the same design phase for a long period of time, and only at several instances, a design phase change can be observed. This fact makes the intervals that have a design phase change more crucial in the learning stage, since the architect's thought process is most likely to stay the same, when the architect is still in the same design phase.

Therefore, instead of using the binary cross-entropy (BCE) which can be seen in Equation 1, we are proposing a custom loss function that augments the BCE loss function. Our loss function penalizes the input sequences that include a phase change, with a large penalty term that can be hyper-tuned.

$$BCE = -\sum_{i}^{N} y_i \log \hat{y} \tag{1}$$

$$CLF = \begin{cases} -\sum_{\mathrm{w}}\sum_{i}^{N} y_i \log \hat{y}, & \text{if } \mathrm{w} \notin P \\ \lambda * \left(-\sum_{\mathrm{w}}\sum_{i}^{N} y_i \log \hat{y}\right), & if\ w\ \in P \end{cases} \tag{2}$$

The custom loss function we propose, can be seen in the Equation 2. In the equation *P* denotes the set that includes the intervals, in which there is a design phase change. Therefore, our loss function, calculates the loss just like the BCE, if the processed batch is not in an interval where there is a design phase change, but it penalizes the term with another parameter when the batch is in an interval with a design phase change.

4 EVALUATION

The model has been implemented using the TensorFlow (Abadi et al., 2016) framework and trained the final model with 10 epochs, and 3.500 steps per epoch. The optimizer being used is the Adam (Kingma & Ba, 2014) with default TensorFlow learning rate of *0.001*. The convergence of the model can be seen from the loss graph in Figure 5.

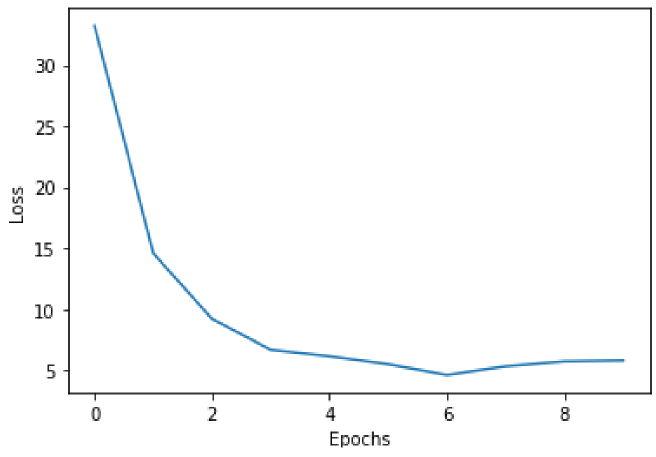


Figure 5. The loss function with 10 epochs.

Having access to a limited number of design process data requires a pertinent evaluation method. We have selected the k-fold cross validation for our training and evaluation subroutine, where the data is split into training, validation and test data. Furthermore, there is no established or best practices evaluation method for temporal data, since the whole sequence is needed for the network to learn the pattern across the data. Hence, splitting the data instance into two sub-arrays as train and test sets can cause a significant loss of information. In order to remedy the shortcoming of splitting the data into train and test sets for temporal data, there are several techniques proposed, for example successively enlarging both the train and the test data, across different epochs (Cerqueira, Torgo & Mozetič, 2020) However, due to the limited amount of data, we applied another evaluation method, in which the design processes from different architects are used for both training and evaluating the data without separating them into training and test sketches. Several time intervals, that include a design phase change, are selected from these sketches as *evaluation intervals*. Hence, in the evaluation, we examine if the model can capture the pattern, since preferably the crucial time intervals are used.

Our evaluation method consists of creating various numbers of intervals in different sketches, which will then be separated arbitrarily as training and test sets. The important difference is that the intervals are selected among the time sequences which include a design phase change. That way it can be examined whether the model can capture the required pattern or not, since preferably the crucial time intervals are used while calculating the accuracy, as it was explained in Section 3.3

Using the described approach, the accuracy is calculated additionally to the prediction results of the design phases for the entire sketch data.

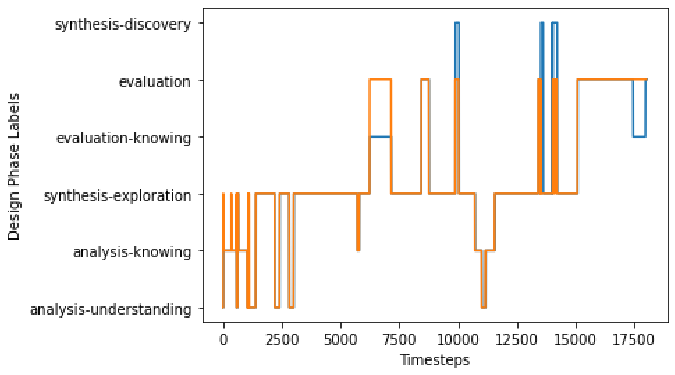


Figure 6. Exemplary evaluation graph: orange lines indicate the prediction values for the whole sketch data, while blue lines represent the ground truth values for the same sketch.

Figure 6 illustrates the prediction values through two lines for all the data within one sketch. While the blue line represents the ground truth values for the respective sketch that indicate the actual design phases of the architect's design process, the orange line shows the design phase predictions of our model. In this categorical line graph, the *y-axis* depicts possible design phases, while the *x-axis* shows the temporal progress with timestamps. As it can be seen in Figure 6, the model predicts the design phases with relatively high accuracy, i.e., 94%. Only a few errors occurred due to the similar patterns in transitioning between design phases. For instance, one repeating error was the model's inability to capture the design phase *Synthesis-Discovery* and instead it mistook said phase as the *Evaluation*. This shows that the transition between the *Synthesis-Exploration* and the *Synthesis-Discovery* has similar dynamics to the transition between the *Synthesis-Exploration* and the *Evaluation*.

Finally, Figure 7 highlights the effect of our custom loss function on evaluation and training success. For both testing and training, our custom function has improved the accuracy of the predictions significantly, compared to the models that have been trained with the same characteristics, but without the BCE loss function. Therefore, the deep learning method, along with the custom loss function, achieves successful results for predicting the current and next design phase of a given design process.

To sum up, even though, the model continues to occasionally mistake design phase changes due to similar patterns, specifically *Synthesis-Exploration* to *Synthesis-Discovery*, and *Synthesis-Exploration* to *Evaluation*, the results show a high accuracy for the prediction of both the current and following design phase. By implementing a custom loss function, which emphasizes the use of temporal intervals for both training and evaluating, we improved the accuracy for both types of predictions by 6-8 percent.

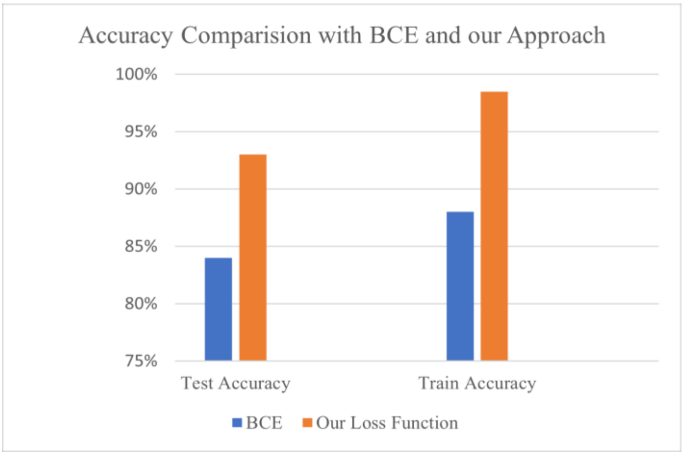


Figure 7. Comparison of the evaluation results, using the BCE loss function, and our proposed custom loss function.

5 CONCLUSION AND FUTURE WORK

The results of our model for both prediction and evaluation are visibly accurate with a percentage of 94 at the end. The implementation of the custom loss function improved the accuracy by 6-8 percent, compared to accuracy of the models trained without the BCE loss function. Thus, we contribute a successful approach for predicting the current and next design phase, based on the categorisation by Laseau (2000), Lawson (2004; 2005) and Barelkowski (2013), using an RNN trained with quantified design process data, to the research field. This novel approach includes the workflow for pre-processing of the design process data, quantified with the sketch analyser tool, the LSTM model architecture and finally, interventions to improve accuracy. Consequently, this novel approach is transferable for predicting custom temporal parameters of various nature of design process data, e.g., design intentions (Lawson, 2004), assigned as custom labels within the protocol analyser tool.

However, the low number of design process data remains a major limitation for training the model, but far and foremost for evaluating the model's behaviour for projecting more general results and outlook. Since recruiting a large number of participants and preparing the dataset with manual labelling proves to be too resource-inefficient and cumbersome, Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) can be employed in the future. GANs are generative models and, trained with enough information, can be used to create novel and virtual data. That way, the data retrieval can be automated and fasten the process of dataset preparation.

Finally, the 'metis' projects aim to ultimately suggest the next design step to the user (e.g., 'outlining parcel') during the sketching process to support the architectural design decision making. Thus, we plan to extend the current approach for predicting and suggesting new design phases for further values, such as design intentions and design steps. The individual RNN models for each value type will be connected in a cascading series from the largest segmentation, the design phases, to the smallest, the design step.

6 ACKNOWLEDGMENTS

We want to thank the DFG for funding the 'metis' projects, as well as the study participants for offering their insight, feedback and design knowledge, as well as their valuable time and sketches.

REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). {TensorFlow}: A System for {Large-Scale} Machine Learning. In 12th USENIX symposium on operating systems design and implementation (OSDI 16) (pp. 265-283).

Abioye, S. O., Oyedele, L. O., Akanbi, L., Ajayi, A., Delgado, J. M. D., Bilal, M., ... & Ahmed, A. (2021). Artificial intelligence in the construction industry: A review of present status, opportunities and future challenges. *Journal of Building Engineering*, *44*, 103299.

Barelkowski, R. (2013). Designing more by knowing less, Verbeke, J., Pak, B., In Proceedings of the Conference 'Knowing (by) designing' at LUCA, Sint-Lucas School of Architecture Brussels, 22-23 May 2013 (pp. 522-531). Ghent, Brussels.

Bielski, J., Langenhan, C., Ziegler, C., Eisenstadt, V., Dengel, A., and Althoff, K.D. 2022. Quantifying the Intangible - A tool for retrospective protocol studies of sketching during the early conceptual de-sign of architecture. In *International Conference of the Association for Computer-Aided Architectural Design Research in Asia* (pp. 403-411). Association for Computer-Aided Architectural Design Research in Asia.

Cerqueira, V., Torgo, L., & Mozetič, I. (2020). Evaluating time series forecasting models: An empirical study on performance estimation methods. Machine Learning, 109(11), 1997-2028.

Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. arXiv preprint arXiv:1409.1259.

Draper, N. R., & Smith, H. (1998). "Dummy" variables. Applied regression analysis, 299-325.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., & Bengio, Y. (2014). Generative adversarial nets. Advances in neural information processing systems, 27.

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 6(02), 107-116.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Laseau, P. (2000). *Graphic thinking for architects and designers*. John Wiley & Sons.

Lawson, B. (2004). What Designers Know. Boston, MA: Elsevier/Architectural Press.

Lawson, B. (2005). How Designers Think. 4th edition, Routledge. ISBN 9780080454979.

Nicholas Negroponte. The Architecture Machine: Toward a More Human Environment. The MIT Press, Jan. 1973. isbn: 9780262368063. doi: 10.7551/mitpress/8269. 001.0001. url: https://doi.org/10.7551/mitpress/8269.001.0001.

Suwa, M and Tversky, B 1997, 'What do architects and students perceive in their design sketches? A protocol analysis', Design Studies, 18(4), pp. 385-403

Van Houdt, G., Mosquera, C., & Nápoles, G. (2020). A review on the long short-term memory model. Artificial Intelligence Review, 53(8), 5929-5955.

Ying, X. (2019, February). An overview of overfitting and its solutions. In Journal of Physics: Conference Series (Vol. 1168, No. 2, p. 022022). IOP Publishing.