Interactive Design Support for Architecture Projects During Early Phases Based on Recurrent Neural Networks

Johannes Bayer(✉), Syed Saqib Bukhari, and Andreas Dengel

German Research Center for Artificial Intelligence,
Trippstadter Strasse 122, 67663 Kaiserslautern, Germany
{johannes.bayer,saqib.bukhari,andreas.dengel}@dfki.de
https://www.dfki.de

Abstract. In the beginning of an architectural project, abstract design decisions have to be made according to the purpose of the later building. Based on these decisions, a rough floor plan layout is drafted (and subsequently redrafted in successively more refined versions). This entire process can be considered an iterative design algorithm, in which high-level ideas and requirements are transformed into a specific building description.

Nowadays, this process is usually carried out in a manual and labor-intensive manner. More precisely, concepts are usually drafted on semi-transparent paper with pencils so that a when a new sheet of paper is put on an existing one, the old concept may serve as a template for the next step in the design iteration.

In this paper, we present a semi-automatic approach to assist the developer by proposing suggestions for solving individual design steps automatically. These suggested designs can be modified between two successive automatic design steps, hence the developer remains in control of the overall design process. In the presented approach, floor plans are represented by graph structures and the developer's behavior is modeled as a sequence of graph modifications. Based on these sequences we trained a recurrent neural network-based predictor that is used to generate the design suggestions. We assess the performance of our system in order to show its general applicability.

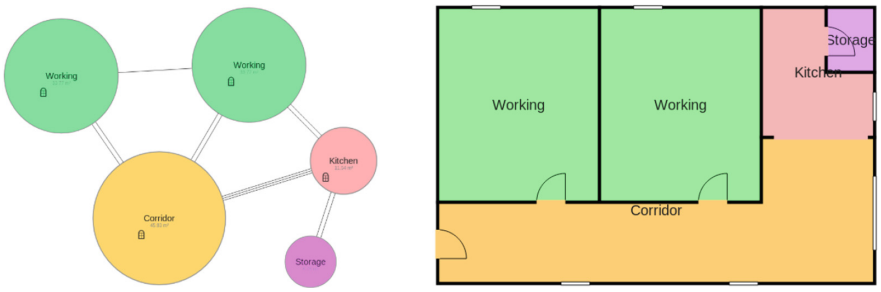
The paper at hand is a extended version of our ICPRAM 2018 conference paper [1], in which we address the different aspects of our proposed algorithm, challenges we faced during our research as well as intended work flow in greater detail.

Keywords: Interactive design support · Early phase support
Architecture project · LSTM · Archistant

M. De Marsico et al. (Eds.): ICPRAM 2018, LNCS 11351, pp. 27–43, 2019.
https://doi.org/10.1007/978-3-030-05499-1_2

1 Introduction

In many architectural projects, the development of a building can be considered to be following a top-down strategy. In the entire life-cycle of a building (formulation of an initial concept of a building to its demolition), the early design phase deals with deriving a first and rough floor plan layout given an abstract idea. Such a concept may be purpose-driven (e.g. 'apartment building for 12 average families with children in a mid-sized city') or may contain stylistic wishes ('Tuscan Mansion') as well as semiotic statements ('The temple should reflect the openness of our religion'). Given that such a high-level description usually comes from a customer or contractor, this initial concept can also be considered a requirement to the project and therefore contains rather punctual yet specific constrains ('The house should have two bathrooms as well as three sleeping rooms and the living room should be next to the kitchen'). Generally, only a rough idea is given, hence most of the final floor plan's aspects remain vague.



(a) Room Graph **(b)** Floor Plan Layout

Fig. 1. Illustration of the room schedule work flow (a → b).

Buildings can be described using graphs structures. In such a modeling, individual rooms are conveniently described as nodes while the connections between them (e.g. walls, doors) are described as edges. Using such a description, all properties of a floor plan layout must consequently be expressed as properties of nodes and properties of edges. The surrounding walls of a room for example can be denoted as a polygon that is a property of the node which represents the room.

Given a graph-based description of a building, the early-phase design task in architecture translates into an iterative graph manipulation algorithm (see Fig. 1). This technique is already existing as a traditional and manual working method in architecture, the so-called *room schedule* (also referred to as *architectural program*). From the algorithmic viewpoint, the building project's concept (or requirements) translate to structural constrains of the building's graph at the start of the algorithm: A fixed amount of rooms translates to a fixed amount of nodes in the graph. Adjacency requirements ('The kitchen must be next to

the living room') translate to predetermined edges in the graph. Defined room aspects translate to fixed properties of the corresponding nodes. The task of such an algorithm is to complete the missing graph properties.

The entire process described to far is nowadays usually carried out manually and informally. This usually involves semi-transparent sketching paper that is being written on with pencils. Algorithmically, each sheet of semitransparent paper represents one iteration (or corresponds to a set of graph modifications). When putting on a new sheet of semi-transparent paper on top of the old one, the old sketch serves as a template for the next, refining design iteration. While a fair amount of this task is highly creative, many actions remain rather repetitive and monotone for the architectural developer. Usually, this process makes use of previously existing projects since architectural projects often work with references. Hence, graph structures from former projects are assumed to be at least partly copied or resembled in newer ones.

In this paper, a semi-automated approach for drafting architectural sketches is described. In Sect. 2, the existing base technologies and concepts on which the approach of this paper is based are outlined. In Sect. 3, the employed representation of floor plans for recurrent neural networks is introduced. Section 4 describes the central machine learning approach, i.e. how models are trained and how floor plan drafts are extended as well as completed using the trained models. Some of the problems encountered during the design of the system as well as how trade-offs attempt to address them are also described here. After that, in Sect. 5 the integration of the approach into an existing sketching software is outlined. The approach is evaluated in Sect. 6, where the results of an automatically conducted performance evaluation on a set of floor plans is presented. Examples of real-value outputs of a trained models are provided to illustrate the use of the integrated system. Section 7 concludes this paper by giving an outlook on possible future research directions.

2 Related Work

2.1 The Long-Short Term Memory

Long-Short Term Memories [7,9] are a class of recurrent artificial neural networks. During each time step, they are supplied with an input vector of arbitrary (but fixed) length, while they emit an output vector of a size equivalent to their amount of cells. Stacked with an MLP, they may also return a vector of also arbitrary (but fixed) size. As vector sequence processing units, LSTMs possess different key properties like the ability to transform data in various ways and storing information for an arbitrary length of time steps. The components of their input vectors have to be normalized to a certain the interval (often $[0, 1]$). Likewise, their output values are limited to a certain interval (often $[0, 1]$). The experiments described in this paper have been conducted using the OCRopus LSTM implementation [4].

2.2 The Architectural Design Support Tool Archistant

Archistant [11] is an experimental system for supporting architectural developers during early design phases. Its key feature is the search of floor plans similar to an entered sketch. It consists of a front-end, the Archistant WebUI, and a modular back-end, in which floor plans are processed between the individual modules via the dedicated AGraphML format.

Archistant WebUI. The Archistant WebUI (see Fig. 2, formerly known as Metis WebUI [2]) is the graphical user interface of the system. It mainly consists of a sketch editor for floor plans (originally the created sketches just served as search requests to the core system). The workflow intended by the Archistant WebUI follows the traditional room schedule working method. Every aspect of a room may be specified as abstract or specific as intended by the user and the degree of abstractness may be altered by the user during his work. This

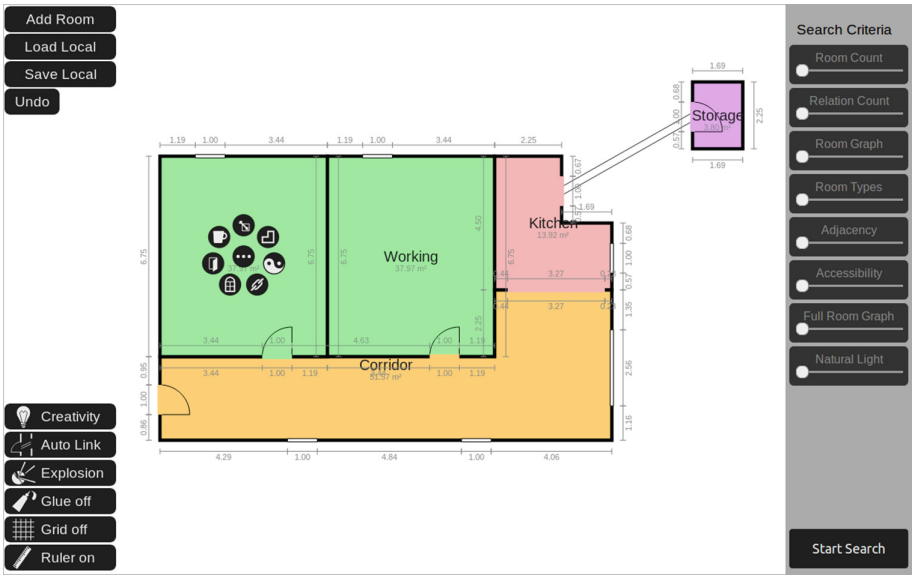


Fig. 2. Screenshot of the Archistant WebUI.

Table 1. Edge types in Archistant.

Type	Description	Visualization (WebUI)
Wall	Rooms share a uninterrupted wall only	1 Continuous Line
Door	Rooms connected by door	2 Continuous Lines
Entrance	Rooms connected via a reinforced door	2 Dashed Lines
Passage	Rooms connected by a simple discontinuity in a wall	3 Continuous Lines

continuous refinement allows for a top-down work process, in which a high-level building description is transformed into a specific floor plan. The Archistant WebUI comes as a web application and therefore runs inside a HTML5-supporting web browser.

AGraphML. AGraphML [10] is Archistant's exchange format for floor plan concepts. AGraphML itself is a specification of GraphML [3]. It follows the convention of the paper at hand, in which rooms are modeled as nodes and their connections are modeled as edges. The AGraphML specification therefore mainly consists of the definitions of node and edge attributes (see Table 1).

3 Encoding Floor Plans for Recurrent Neural Network Processing

This section outlines the representation of floor plans which is used to make them processable by recurrent neural networks. In the current status of our work, we restrict ourselves to a limited set of floor plan attributes that are incorporated into this representation: Room functions, connections between rooms, room layouts (i.e. a polygon which is representing a rooms surrounding walls), and information whether or not natural light is available in a room or not (which roughly equals to whether or not a room is equipped with at least one window).

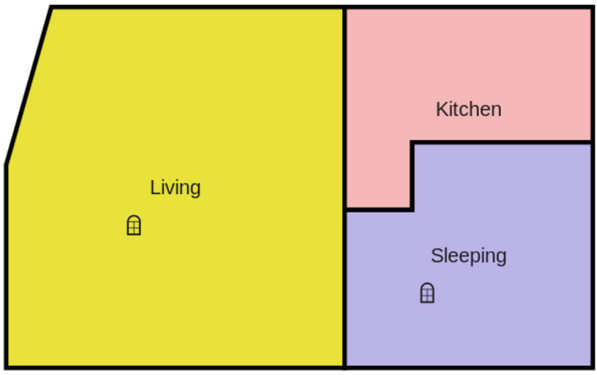


Fig. 3. Rendered image of a sample floor (from [1]). Window symbols indicate access to natural light. The detail level shown in this image equals the information contained in the neural network representation.

3.1 Requirements

In order to be processable by recurrent neural networks, floor plans need to be described as a sequence of feature vectors (all feature vectors must have the same length). There are several requirements to this sequence representation: Both sequence length and vector size should be as small as possible in

order to minimize learning and inference execution times. The vectors should be easy to interpret by automated means. Finally, the actual information should be organized into separated chunks of small vector sequences and these should be separated by data-less so-called *control vectors* (their purpose will be explained later). Most important, the information flow in the vector sequence should mimic the actual workflow of a user who develops a floor plan. Consequently, abstract information should precede specific information, i.e. declaration of all rooms along with their room functions should be before the declaration of the actual room layouts.

3.2 Blocks

A complete floor plan description in the chosen representation consists of 3 consecutive blocks:

1. Room Function Declarations.
2. Room Connections.
3. Room Geometry Layouts.

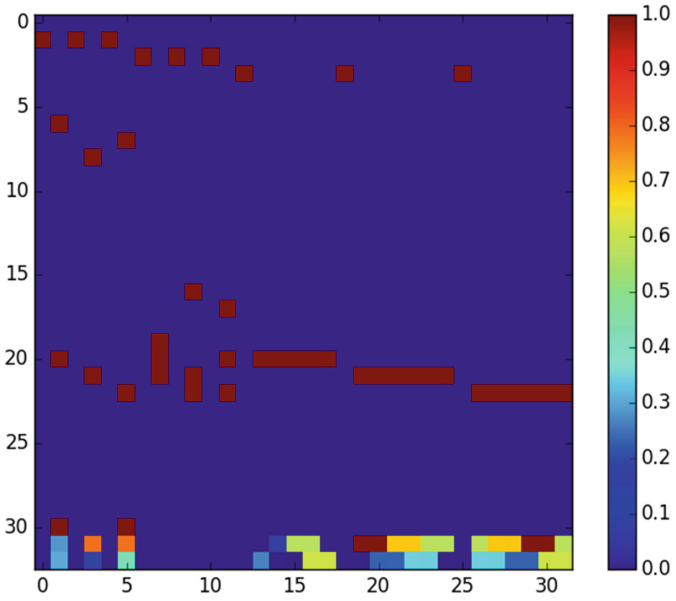


Fig. 4. Feature vector sequence of the same sample floor encoding (from [1]). Every feature vector occupies one column. The encoding consists of three blocks. The first block ranges from column 0 to 5 and defines the rooms along with IDs (row 20–29), the room function, the room's center position (row 31 and 32), and whether or not a room has access to natural light (row 30). In Block 2 (column 6–12) connections between the rooms are defined. In block 3 (column 13–31) the polygons of the room surrounding walls are described.

Each block consists of a number of tags of the same kind (i.e. there is one tag type for each block). Each tag is represented by a number of vectors. An example for a rendered floor plan along with its representation as a sequence of vectors can be seen in Figs. 3 and 4 respectively.

3.3 The Feature Vector

The feature vector is considered to be structured into several channels (see Fig. 5 for the relation between channels and actual feature vector components):

- The blank channel indicates that no information are present (used to indicate start and ending of floor plans or to signal the LSTM to become active)
- The control channel indicates that a new tag begins and what type the new tag is
- The room type channel (room types are called room functions in architecture)
- The connection type channel
- The room ID channel is used to declare or reference an individual room
- A second ID channel is used for connection declaration
- has Window property

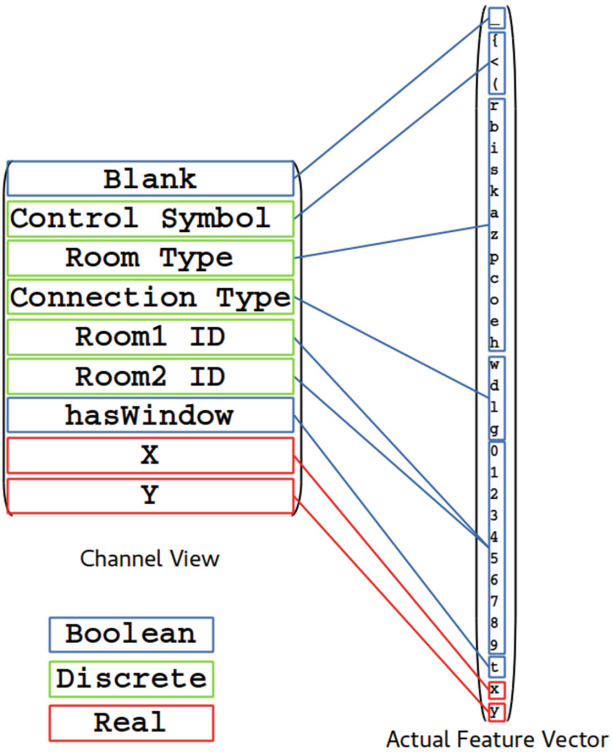


Fig. 5. Structure of the feature vector (from [1]).

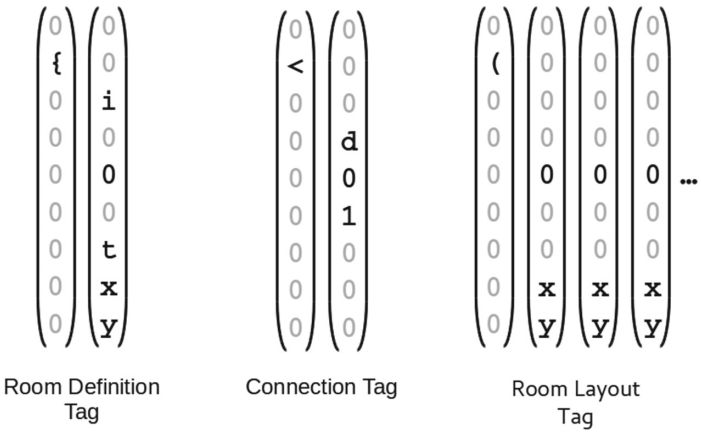


Fig. 6. The different tag types in feature vectors representation (channel view, from [1]). Left: Definition of a Living (`i`) Room with ID (`1`) with a Window (`t`) with a center at position (`x`, `y`). Middle: Definition of a door connection (`d`) between rooms `0` and `1`. Right: Definition of the polygon layout of room `0`.

– X Ordinate of a Point
– Y Ordinate of a Point

Generally, LSTMs allow for real-valued vector components as input and output. However, only a 2D-Point's X and Y ordinates (which are both normalized to $[0, 1]$) actually make use of this capability. All other components are modeled as boolean variables (0.0 for `false`, 1.0 for `true`). In a former version of the floor plan representation, 1-Hot Encoding has been considered. However, this approach is rather inefficient compared to the finally chosen one (the final, tighter encoding occupies around 88 less space than our experimental 1-Hot encoding of the same information). So in order to minimize the sequence length, each feature vector may contain several information and the ID of each room is represented as a dedicated channel. As an disadvantage, this finally chosen, tight encoding only allows for a fixed upper limit of rooms that has to be determined before training (10 rooms in the paper at hand).

3.4 Tags

Tags are the atomic units of floor plan features. At the same time, they can be considered graph modifications carried out by an entity (like the user) to develop a floor plan. Every tag is represented by a set of successive feature vectors. All tags start with a so-called control vector, that only serves the purpose of indicating the tag's type. Figure 6 illustrates how the different tag types are made up from feature vectors. Currently, there are three different types of tags:

Room Definition Tags. Each tag of this type defines a room by assigning it an ID along with a room type, a flag indicating whether or not the room has a window, and the position of the room's center. This tag type always occupies 2 feature vectors.

Connection Tags. A Connection Tag declares a connection between two rooms. It consists of the references between the two connection partners as well as the connection type. Since all room IDs are represented in the feature vector by dedicated components, two room IDs can be represented in one feature vector. The supported connection types are taken from the definition of Archistant. This tag type always occupies 2 feature vectors.

Room Layout Tags. These tags define a polygon surrounding walls around a room by describing the individual corners of the room's walls successively. A room layout tag occupies $p+1$ feature vectors, where p is the number of corners of the room.

3.5 Room and Connection Order

The order of rooms and connections in the floor plan representation underlies a trade-off: a well defined order of rooms and connections only allows for exactly one representation of the same floor plan. By allowing for any arbitrary order of rooms and connections, the actual user behavior is better approximated and there are multiple representations of the same floor plan (many samples can be created from one single floor plan). However, when considering elements to be given in a random fashion, an LSTM that should predict them is difficult to train. Since a random order of room definitions and connections adds an unpredictable noise to the LSTM, the order or rooms and connections is defined as follows:

Room Order. The order in which the rooms are given in the feature vector sequence is determined by the center position of the room within the floor plan. A room which center has a smaller X ordinate appears before a room with a greater X center ordinate. In case of the centers of two rooms share the same X ordinate, the room with the smaller Y ordinate precedes the other room (top to bottom, left to right). The order of rooms is the same for block 1 and block 3.

Connection Order. The order of connections in the feature vector sequence is determined by the order of the rooms. At this point, the connection graph is considered to be directed and that the source room IDs are always smaller than target room IDs. If two connections have different source room IDs, the connection with the lower source room ID will come before the one with the higher source room ID. If two connections have the same source room ID, the connection with the lower target room ID will precede the connection with higher target room ID.

4 Proposed Mechanism of Autocompletion of Floor Plans Using LSTM

In this section, the proposed algorithm for expanding and completing floor plans is presented. More precisely, the modus operandi, in which existing parts of floor plans are given to the (LSTM) model and new floor plan parts are retrieved is outlined.

4.1 Input and Output Sequences

The structure of the model's input vector is identical to the structure of its output vector and therefore both are referred to as feature vectors. Consequently and as a general working principle, existing parts of a floor plan are used as input to the model while new floor plan parts are retrieved from the model's output. Two different approaches that implement such a behavior are examined here: *block generation sequencers* and *vector prediction sequencers*.

Block Generation Sequencers. Block generation sequencers follow a simple pattern: The first n blocks are given to the model's input. Simultaneously, the model's output is simply a series of blank vectors (the blank component is 1.0, while all other components remain 0.0). Afterwards, a sequence of blank vectors is used as input while the $n+1th$ block is emitted by the model (eventually finished by a blank vector). As a result, there has to be one model trained for each block that should be predicted. Consequently, multiple models are used to support the user during the entire work flow. Additionally, a model for supporting the first design step cannot be created using this kind of sequencer.

Vector Prediction Sequencers. Vector prediction sequencers aim to predict the n-th vector of a sequence given the first $n-1$ sequence vectors. These sequencers are trained by using a concatenation of a blank vector with the full feature vector sequence of a floor plan as the input. As an output, the full feature vector sequence of the same floor plan is used, but this time followed by a blank vector.

4.2 Preparation of Database

In order to make maximum use of the limited set of floor plans available in AGraphML, some preprocessing is applied to generate the final training set as well as test set (see Fig. 7). The validation set is omitted here since the amount of floor plans available was very limited and previous experiments on a similar DB indicated that overfitting is not a serious issue in the given situation. First of all, the original sample set is split into two disjoint subsets. Each floor plan is now converted into b different samples (we refer to this process as blow-up and to b as blow-up factor). A sample is derived from a floor plan by rotating all point

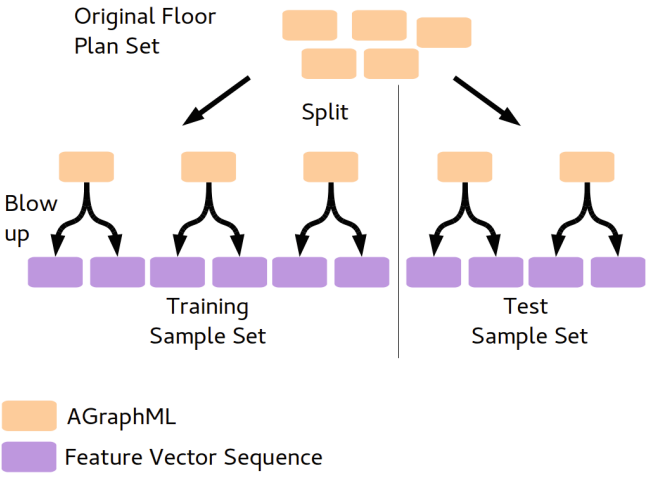


Fig. 7. Sample preparation (from [1]).

of the floor plan (centers, corner points) by an random vector and then create the feature vector sequence as described so far. During this step, the center and corner points of the floor plan are also normalized to the $[0, 1]^2$ space. Because of the applied rotation, the order of rooms and connections differs within the samples generated from one single floor plan.

4.3 Extension of Floor Plans

The two different sequencing approaches need different strategies to generate new floor plan aspects, as outlined below:

Block Generation Sequencers. In this approach, a new block is generated by feeding a concatenation of previous blocks with a sequence of blank vectors into the model and reading the predicted block from the model's output. The blank vector sequence length must be larger that the expected length of the predicted block. This is can be done by determining the upper limit of predicted block length in the training database. A more sophisticated approach is to make use knowledge on the input sketch (e.g. the room count of the input sketch).

Vector Prediction Sequencers. Following a metaphor by Alex Graves, in which sequence predictors used for sequence generation are compared to a person dreaming (both are iteratively treating their own output as new input [8]), this structures works like a dreaming person who occasionally gets inspiration from outside and who combines the information from outside with its flow of dreaming. Because of that, this technique is referred to as the *shallowDream* structure here (see Fig. 8).

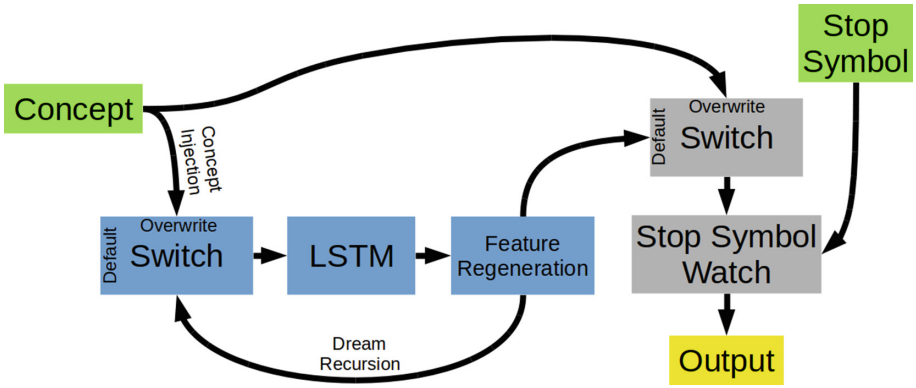


Fig. 8. The shallowDream structure (from [1]). The inputs are marked green. Components of the LSTM recursion are marked blue. (Color figure online)

Basically, this structure operates in two different phases. During the first phase, the already existing floor plan parts as provided or accepted by the user (also referred to as *concept*) are fed into the model. During this phase of *concept injection*, all outputs of the model are ignored (the concept is kept as it is in the feature vector sequence, i.e. existing requirements or design iterations are fully sustained). After the concept has been injected completely, the model takes over both the generation of the structure's output and also serves as its own input. This phase is also referred to as *generation phase*. The process is terminated when a predefined stop symbol occurs in the feature vector sequence.

Using the shallowDream structure, it is possible to implement multiple different functions by simply altering the concept and the stop symbol. For example, in order to predict room connections, the a concatenation of block 1 and the control vector of a block 2 tag (connection tag) is used as concept and the control vector of a room layout tag is used as stop symbol. The control vector at the end of a concept is used to instruct the model to generate the favored tag type (and hence to start the new block).

Even after intensive training, the output produced by the model only approximates the intended feature vectors. An error at any time step influences the outputs of all subsequent time steps due to the recursive characteristics of the shallowDream structure. This effect aggravates over time, therefore feature vectors have to be regenerated during the generation phase. For that purpose, three different strategies are proposed:

No Regeneration. In this primitive approach, the current feature vector is simply reinserted without any modifications into the models input.

Vector-Based Regeneration. This strategy solely utilizes knowledge about the feature vector's structure. Generally, all boolean components are recovered

by mapping them to 1.0 or 0.0 based on which the component is closer to and the real-valued components remain unaltered.

Sequence-Based Regeneration. In this approach, a state machine is keeping track of the current block and tag the sequence is in (thereby utilizing knowledge about the sequence structure). Based on that information a vector is regenerated by calculating the most likely, possible vector. In order for the state machine to transit between different states, only selected components are evaluated against a threshold. Different components (and combinations of them) might be used in different blocks.

5 Integration of the Proposed Mechanism into Archistant

This paper is restricted to the following two functions:

- Room Connection Generation. Given a set of rooms (each room is described by a center position coordinate and room function) connections are generated between them, turning a set of rooms into a room graph.
- Room Layout Generation. Given a room Graph, layouts for each room a layout (i.e. a polygon describing its surrounding walls) is generated.

For the sake of simplicity, a single button is added to the WebUI only, which we labeled "Creativity". Based on the current state of the user's work, the different functions are selected automatically.

6 Experiments

LSTMs are trained based on the two described sequencing approaches. In all cases, a training database with 200 entries, a test database of 40 entries, a blowup factor of 30, 500 LSTM cells and a learning rate of 0.01 are used.

6.1 Quantitative Analysis

In order to compare the performances of the different approaches, the room connection generation is calculated on the room definitions of the test set of floor plans and compared with these floor plan's actual connections. As a metric, the amount of wrong connections is divided by the amount of actual connections. A connection is considered to be completely wrong (error 1.0), if the predicted connection does not actually exist in the ground truth. It is considered partly wrong (error 0.5), if the connection type in ground truth is different. The final evaluation value is the arithmetic average over all floor plans in the test set. The results are shown in Table 2. It is emphasized that floor plan generation is a creative task and that there is not necessarily one solution to a given problem, i.e. the used error calculation metrics do only hint the actual performance of the approaches (an error of 0% appears unrealistic to accomplish given that also the floor plans in the database are only one way to solve the problem).

Table 2. Performance comparison of the individual approaches for the connection generation task on the test set (from [1]).

Approach	Error
Block generation sequencer	65.78%
Vector prediction sequencer	66.08%

6.2 Qualitative Analysis

The performance of the shallowDream structure is shown exemplary for two scenarios. For the room connection generation, four rooms are given (Fig. 10) as a concept. As illustrated in Fig. 9, the regeneration strategy influences the output. Figure 11 depicts a rendered version of the output produced by the sequence based regeneration.

In order to illustrate the performance of the room layout generation function, a different starting situation is used (see Fig. 12), the result is depicted in Fig. 13.

7 Future Work

We have shown the general viability of our approach (i.e. the output of the trained models resembles the intended syntax in a quality sufficient for our inferencing algorithm to produce results). Nevertheless, a lot of floor plan aspects are not yet covered in the existing models. The actual position of doors and windows are needs to be included into the models as well as support for multi-storey buildings. Apart from that, the performance of the existing models is still fairly limited. At the moment, there is only one phase of concept injection followed by a generation phase. By allowing for multiple alternating phases of generation and concept injection, even more functions could be realized. Apart from that, better metrics have to be found to assess a models performance. As of now, our approach only supports a strict order of design phases. A more flexible approach that allows for a more fluid transition between design steps would be useful in future.

In order to both allow for better comparison to similar approaches and to improve the performance of our system, the presented approach can be applied to a standard database [5] and existing algorithms [6] can be used to increase the sample size of the sample database.

Many of the mentioned problems could be avoided by a completely different sequencing strategy: Instead of employing a single neural network that manipulates the entire room graph, multiple neural network instances could be used instead. In such a scenario, one LSTM could be responsible for a single room. Such a group of LSTMs could communicate the results of individual design steps among each other. Not only would this approach allow for shorter sequences, it would also render most of the room ID assignment as well as room and connection ordering redundant. The problem of a predetermined, fixed upper limit of

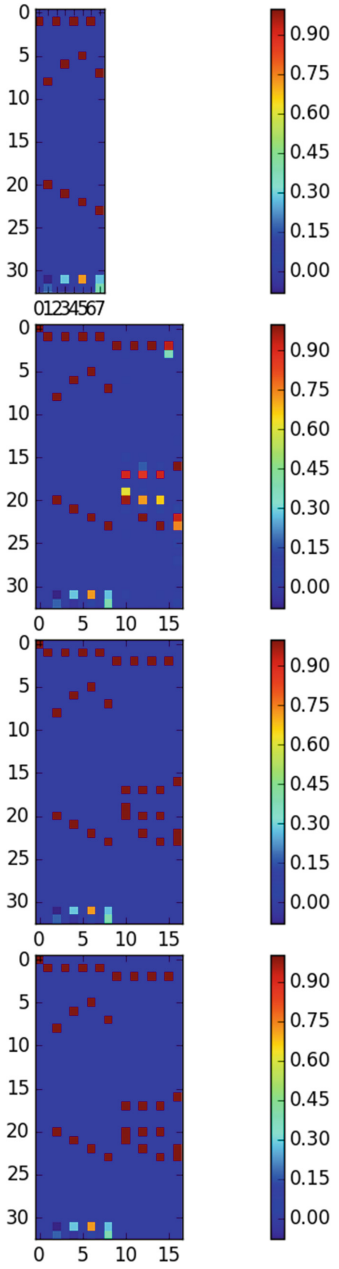


Fig. 9. The regeneration technique in shallowDream influences the generated feature vector sequences (from [1]). From Top to Bottom: 1. The concept. 2. No Regeneration. 3. Vector-Based Regeneration 4. Sequence-Based Regeneration.

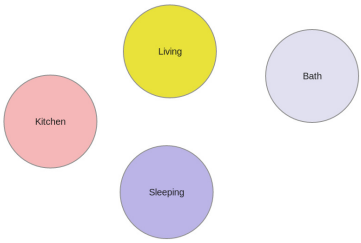


Fig. 10. Input given to the shallowDream structure (from [1]).

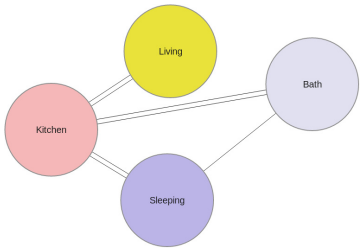


Fig. 11. Output obtained from the shallowDream structure (from [1]).

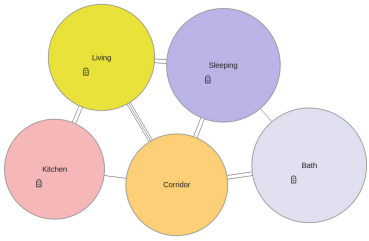


Fig. 12. Input given to the shallowDream structure (from [1]).

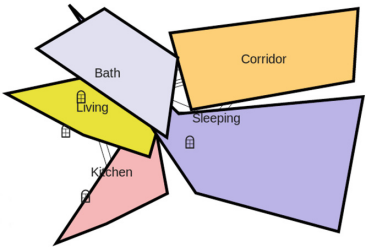


Fig. 13. Output obtained from the shallowDream structure (from [1]).

rooms would therefore also be mitigated tremendously in such a setting. Likewise, one of the assumed reasons for the low training performance of the models is the amount of information that the LSTMs had to store. This problem would also be mitigated in a distributed approach.

Apart from that, our approach can be used as a general template for machine learning of user behavior, given that the data structure manipulated by the user can be described as a graph. Consequently, a more general implementation of a graph-based machine learning framework can be build using our approach.

Acknowledgment. This work was partly funded by Deutsche Forschungs-Gemeinschaft.

References

1. Bayer, J., Bukhari, S., Dengel, A.: Interactive LSTM-based design support in a sketching tool for the architectural domain. In: 7th International Conference on Pattern Recognition Applications and Methods, Funchal (2018)
2. Bayer, J., et al.: Migrating the classical pen-and-paper based conceptual sketching of architecture plans towards computer tools - prototype design and evaluation. In: Lamiroy, B., Dueire Lins, R. (eds.) GREC 2015. LNCS, vol. 9657, pp. 47–59. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-52159-6_4
3. Brandes, U., Eiglsperger, M., Lerner, J., Pich, C.: Graph Markup Language (GraphML). In: Tamassia, R. (ed.) Handbook of Graph Drawing and Visualization, vol. 20007, pp. 517–541. CRC Press, Boca Raton (2013)
4. Breuel, T.M.: The OCRopus open source OCR system. In: Electronic Imaging 2008, p. 68150F. International Society for Optics and Photonics (2008)
5. de las Heras, L.-P., Terrades, O.R., Robles, S., Sánchez, G.: CVC-FP and SGT: a new database for structural floor plan analysis and its groundtruthing tool. Int. J. Doc. Anal. Recognit. (IJDAR) **18**(1), 15–30 (2015)
6. Delalandre, M., Pridmore, T., Valveny, E., Locteau, H., Trupin, E.: Building synthetic graphical documents for performance evaluation. In: Liu, W., Lladós, J., Ogier, J.-M. (eds.) GREC 2007. LNCS, vol. 5046, pp. 288–298. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88188-9_27
7. Gers, F.A., Schmidhuber, J., Cummins, F.: Learning to forget: continual prediction with LSTM. Neural Comput. **12**(10), 2451–2471 (2000)
8. Graves, A.: Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850 (2013)
9. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
10. Langenhan, C.: Datenmanagement in der Architektur. Dissertation, Technische Universität München, Müchen (2017)
11. Sabri, Q.U., Bayer, J., Ayzenshtadt, V., Bukhari, S.S., Althoff, K.-D., Dengel, A.: Semantic pattern-based retrieval of architectural floor plans with case-based and graph-based searching techniques and their evaluation and visualization (2017)