Deep Generative Design

Deep reinforcement learning for performance-based design assistance

https://github.com/JairLemmens/Thesis/

By: Jair Lemmens 4645448

Date: 19/01/2024

Supervisors: Dr. Charalampos Andriotis, Dr. Michela Turrin

Delegate of the Board of Examiners: Dr. Olindo Caso

Special thanks to: Casper van Engelenburg, Hans Hoogenboom

Contents

1. Introduction.

In the face of ever-growing challenges concerning climate change and increased urbanization, it is imperative to foster sustainable design practices. Architects play a pivotal role play in the sustainable development of the built environment, shaping spaces that minimize environmental impact while enhancing human well-being. However, due to the escalating complexity of design requirements, designers are becoming underequipped to solve today's design challenges. Solving such problems involves balancing multiple different co-dependent factors, such as spatial configuration, daylight satisfaction and embodied carbon. Currently this is addressed through inter-disciplinary collaboration where different parties focus on their own specialisation. As a result, a significant gap exists between design and building performance analysis leading to underperforming and expensive solutions.

An artificial intelligence-powered platform could aid in the design process by bridging the gap between different disciplines. Such a platform could ensure that all aspects of a project are considered holistically, reducing the risk of conflicting decisions that cause lacking performance and unnecessary expenses. Simply consolidating information, as done with building information modelling (BIM), is not sufficient. Instead, the platform must provide performance-driven recommendations, offering deeper insights when conflicts between different disciplines arise.

An emerging field that is aiming to address this need is that of deep generative design. Unfortunately, the real-life application of thus far proposed methodologies has been limited, reasons for this include but are not limited to the following; First, the environment of design is a relatively data sparse one, making it difficult to train supervised learning-based solutions. Second, conventional deep generative models rely on the replication of existing products through supervised learning based on examples, leading to a creativity gap. A derivative of existing solutions is not always desirable in design. Third, the development of performance aware deep generative models is challenging due to the (computational) cost of numerical simulation and real-life evaluation (Regenwetter, Heyrani Nobari, & Ahmed, 2022).

The data format used in this work will be that of floorplans. As a visual representation of a building's layout, floorplans can be widely understood by professionals across relevant fields. Additionally, they provide a detailed spatial description from which critical details such as: room dimensions, wall and window placement, floor spans, composition, and circulation can be extracted.

To address the previously mentioned challenges, this work seeks to explore the following questions:

How can deep learning be used to assist designers in creating performance-informed floorplans?

How can a collaborative, performance aware deep learning system overcome data scarcity and the creativity gap?

How can the associated computational costs be alleviated through ensemble models?

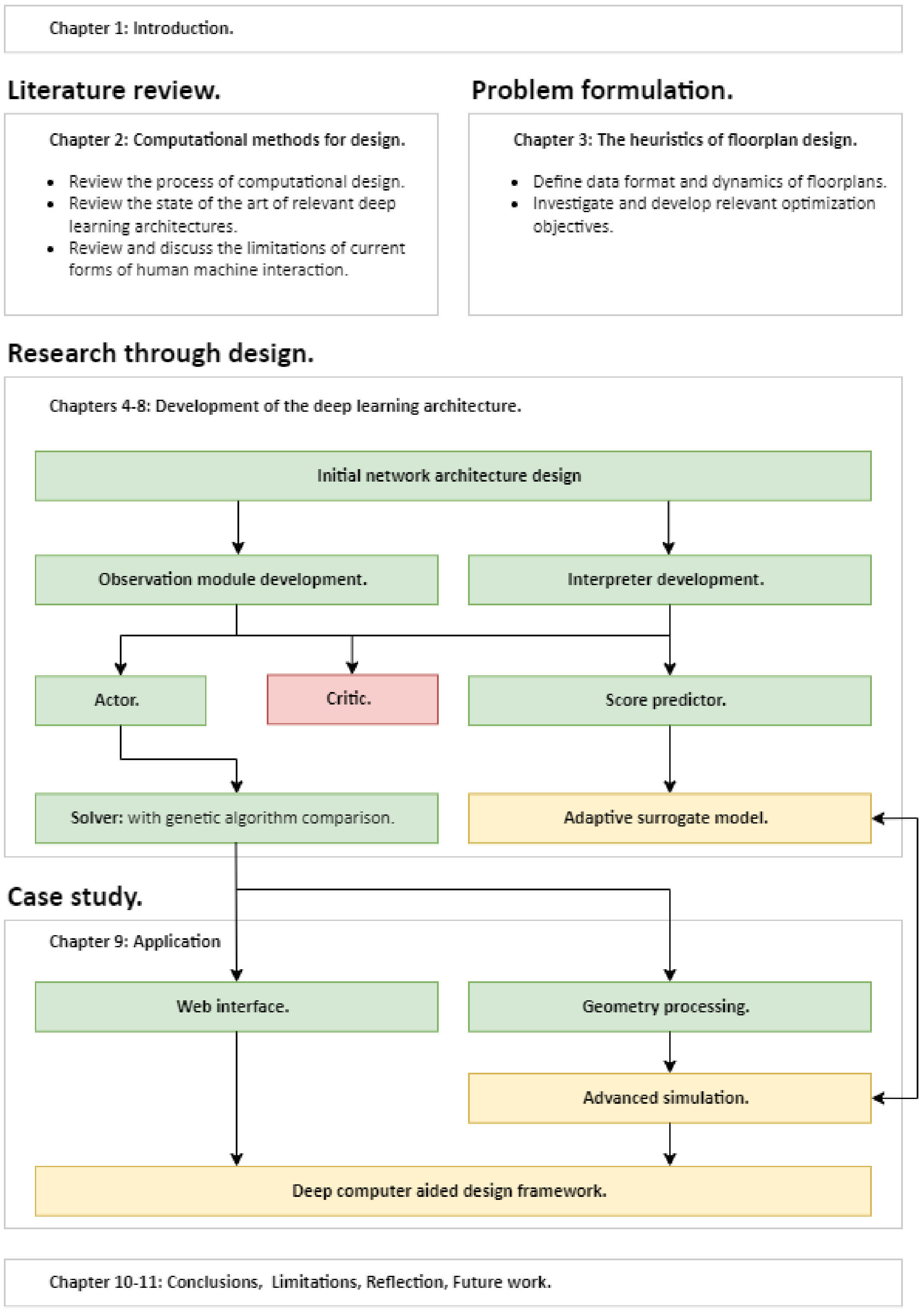
1.1 Objectives.

This work aims to develop a framework through which deep learning methods can be applied to create floorplans informed by performance-based criteria and user guidance. Since functionality is heavily dependent on site-conditions and requirements laid out by the client, the relationship between form and function may have to be uncovered dynamically. To achieve this, inspiration will be drawn from the field of reinforcement learning which allows the training of a neural network without the use of training data.

Due to time constraints the performance criteria will be limited to reference similarity and daylight satisfaction. However, the ability to accommodate arbitrary future criteria is a highly desirable characteristic of the envisioned approach.

1.2 Methodology.

The methodologies used in this work can be seen in figure 1.



Chapter 1: Introduction.
Literature review.
Problem formulation.
Chapter 2: Computational methods for design.
Review the process of computational design.
Review the state of the art of relevant deep learning architectures.
Review and discuss the limitations of current forms of human machine interaction.
Chapter 3: The heuristics of floorplan design.
Define data format and dynamics of floorplans.
Investigate and develop relevant optimization objectives.
Research through design.
Chapters 4-8: Development of the deep learning architecture.
Initial network architecture design
Observation module development.
Interpreter development.
Actor.
Critic.
Score predictor.
Solver: with genetic algorithm comparison.
Adaptive surrogate model.
Case study.
Chapter 9: Application
Web interface.
Geometry processing.
Advanced simulation.
Deep computer aided design framework.
Chapter 10-11: Conclusions, Limitations, Reflection, Future work.

FIG 1 METHODOLOGICAL OVERVIEW, GREEN: COMPLETED, RED: CANCELLED, YELLOW: FUTURE WORK.

1.2.1 Literature review.

The literature review has been conducted to understand the current state of the art in the field of computational design. While the fundamentals such as design representation and multi objective optimization will be discussed, the focus will be deep learning and its applications within engineering design. The review will be concluded with a short overview of the findings.

The literature will be selected based on relevance, year of publication, and quality of the source. The search will be conducted on multiple academic databases such as google scholar using keywords such as “generative design”, “deep learning”, “timeseries analysis”. After reading the abstract and conclusions of the papers to determine its relevance, ChatGPT was used to locate the desired details in the works.

1.2.2 Research through design.

Within this part of the work the deep learning modules will be developed and described. Deep learning architectures, through their complexity, are often quite unpredictable. To compensate a process of iterative prototyping will be used to guide further research. Parts of the model will be constructed trained and compared to alternatives to determine the best performing combination. It is also worth mentioning that various model variants were constructed trained and tested that did not impress enough te be included in the report, some of these can be found in the appendix.

1.2.2 Case study.

During the case study the developed model will be tested by applying it to a lifelike problem, with the goal of uncovering any design oversights. The case study will consist of a design exercise in which a small building will be automatically designed directly from a project brief and context model.

1.3 Relevance.

Using the developed tool, designers will be enabled to investigate the effect of design modifications on the technical performance of their designs. The reinforcement learning approach, allows design representations to develop dynamically, reducing the bias embedded in explicitly programmed parametric models. The combination of these factors may improve design performance while reducing development time saving cost. This is beneficial to both the environmental and financial sustainability of the built environment. The flexibility of the system allows it to be adapted to different fields as well increasing potential societal impact.

Academically this work is interesting for its utilization of the attention mechanism for information transfer between parallel states in a reinforcement learning setting. This methodology is shown to be capable of being applied both to Markov Decision Processes and adaptive surrogate models. Additionally, the application of proximal policy optimization in a greedy fashion is an approach for which there is limited precedent.

For researchers looking to apply deep learning to generative design, this work describes a system architecture that can correlate performance with geometry dynamically. Based on these correlations, samples may be modified such that their assigned performance is improved. This is interesting since two of the four main limitations preventing the real-life application of deep learning to engineering problems, are resolved. These limitations are data scarcity and an inability to cross the creativity gap.

2. Computational methods for design.

Design is an iterative process that leads designers to satisfactory variants. Traditionally this iteration is performed manually, however it may also be automated. Potential benefits of automation are reduced design time and improved performance (Martins & Ning, 2021). To enable automation the design problem must first be formalized through the specification of design variables, objectives, and constraints. Once this is achieved, various optimization algorithms may be used to find design variables that maximize objectives while satisfying constraints.

2.1 Limitations of design representation.

As mentioned, the first step of automation is the specification of design variables. The traditional approach for determining these variables involves manually programming an explicit design representation. However, since floorplans are highly complex and varied the process of determining a satisfactory representation is challenging. Additionally, such a representation would likely not cover the whole design space.

An alternative approach is to apply neural networks to automatically extract abstract representations from reference data. This does however not come without its own limitations such as a reliance on large quantities of training data. Another result of such a data driven approach is that these algorithms are only able to mimic or interpolate between preexisting designs contained in its training data, this does not allow for novel solutions (Regenwetter, Heyrani Nobari, & Ahmed, 2022). Nevertheless, solutions for floorplan generation based on deep learning do exist. An example of this is HouseGAN which can generate building layouts while being conditioned on a graph (Nauata, Chang, Cheng, Mori, & Furukawa, 2020). This work utilizes a generative adversarial network to generate a mask for each room, these masks are then postprocessed by fitting an axis aligned rectangle. The method of combining multiple masks to form a floorplan allows this approach to drastically increase sample diversity compared to a combined approach.

2.2 Design objectives and constraints.

To identify the optimal design relevant objectives and constraints must be defined. An objective is a scalar value that is to be minimized or maximized through the optimization process (Martins & Ning, 2021). Successful design optimization relies heavily on the choice of these objectives since they must accurately represent the true design goals. Constraints are a subcategory of objectives that represent a condition that must be satisfied, unlike regular objectives they are represented in a Boolean format instead of a scalar. These provide an extra challenge especially for deep generative approaches since the probabilistic nature of these algorithms may lead them to generate completely invalid designs with various and unpredictable failure conditions (Regenwetter, Heyrani Nobari, & Ahmed, 2022). Constraints also introduce the risk of over constraining a problem, especially when multiple constraints are superimposed.

2.3 Multi objective optimization.

In many cases it may be necessary to combine multiple different objectives. The most straightforward approach is to assign weights to each objective, add them together and treat them as one composite objective. However, it is not always clear how these weights should be distributed. Additionally, the dynamics of the trade-offs may not be obvious, the ability to compare these trade-offs after the optimization is therefore desirable.

One way in which points can be defined as optimal without having to define fixed weights is by applying the concept of pareto optimality (Martins & Ning, 2021). For a solution to be pareto optimal there should not exist another solution which outperforms it in all objectives, this is called a non-dominated solution. These non-dominated solutions together are called the Pareto front which represents the set of optimal trade-offs. From these, designers can choose solutions that best align with their preferences and priorities, facilitating informed decision making without predefined fixed weights.

Generating these optimal solutions requires an optimization methodology. A popular method is the evolutionary algorithm, specifically NSGA-II (Deb, Pratap, & Agarwal, 2002; Martins & Ning,2021). It sorts solutions using the pareto optimality while maintaining solution diversity using crowding distance. It also incorporates elitism, ensuring the best solutions are carried forward to subsequent generations, thereby preventing the loss of high-quality solutions. The algorithm works by initializing a population, evaluating fitness, performing non-dominated sorting, calculating crowding distances, selecting parents based on fitness, and finally applying gene crossover and mutation to generate offspring. The same procedure is repeated on the offspring until satisfactory performance has been achieved.

Performing a multi criteria optimization using a gradient based technique is more challenging. In the absence of a clear weight for the parameters the direction of the gradient vectors becomes unclear. One way to address this is by utilizing a technique called normal boundary intersection (Martins & Ning, 2021). It works by maximizing the distance between a solution projected onto a reference plane along its normal vector as seen in figure 2. The reference plane is created by first generating the optimal solutions for isolated criteria, these solutions are then used to as anchors for the plane.

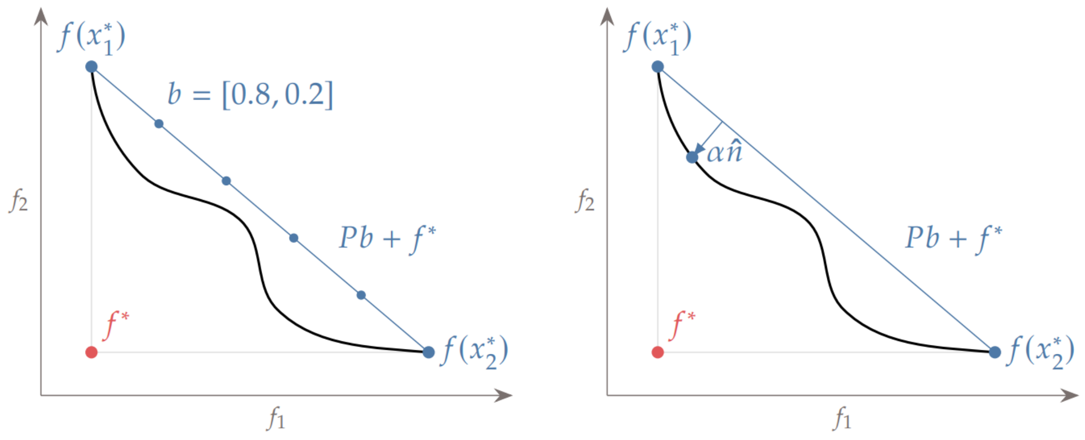


FIG 2 NORMAL BOUNDARY INTERSECTION (Martins & Ning, 2021).

2.4 Deep Learning.

As mentioned previously, deep learning may be used to automatically extract abstract design representations from reference data, to facilitate design optimization. But what is deep learning? Deep learning represents a subset of machine learning algorithms that aim to model high-level abstractions by using architectures composed of multiple nonlinear transformations. The main advantage of deep learning as opposed to traditional machine learning, lies in its ability to automatically discover the representations needed for tasks such as classification and representation learning, without the need for manual feature engineering. (LeCun, Bengio, & Hinton, 2015). What makes this possible is, as the name implies, the increased depth of the networks. Over the last decade, deep learning has advanced various fields including speech recognition, visual object recognition, and many others.

The most common form of deep learning is supervised learning, where a neural network is trained on a labelled dataset. During training, the weights and biases of the artificial neurons are tuned through backpropagation in such a way that the labels in the dataset can be replicated from the input data

(LeCun, Bengio, & Hinton, 2015). When properly trained, the neural network can then be used to predict labels associated with datapoints that are similar to, but not included in the training data, this is called generalization. Label prediction is however not the only task that can be learned by deep neural nets, other applications include feature extraction, image generation, and natural language processing. Additionally, un-supervised learning or reinforcement learning are also viable options to train neural networks, depending on the available data and envisioned task these techniques may be more suitable than supervised learning.

2.4.1 Convolutional neural networks.

A subclass of neural networks that may be particularly relevant to this project are convolutional neural networks (CNN). The reason for its relevance is its applicability to spatial data such as images which are a suitable medium for floorplans. CNNs utilize convolution filters to the extract the spatial structure of the input data. By reducing the size of the data between layers the information can be progressively more compressed until only an abstract representation remains this is called an encoder network (LeCun, Bengio, & Hinton, 2015). The reverse can also be achieved by increasing the size of the data in between the layers, this is called a decoder.

2.4.2 Recurrent neural networks.

Another significant subclass is the recurrent neural network (RNN), which is very suitable for sequential data, such as text and timeseries. RNNs, and their variants like Long Short-Term Memory (LSTM) networks, can capture temporal dependencies, making them ideal for tasks like language processing and timeseries analysis (Hochreiter & Schmidhuber, 1997). They work network by altering an internal state at each timestep while ensuring constant error flow to improve stability. The internal state is a dense representation of the previous timesteps. This makes it very useful in problems like time series prediction and reinforcement learning problems in which information of past states is needed (Staudemeyer & Morris, 2019).

2.4.3 Attention.

Recently, the attention mechanism has become indispensable from modern deep learning architectures. First introduced to address some of the limitations of the previously mentioned RNN, the attention mechanism can handle dependencies between datapoints dynamically, without being impacted by the distance of data within the input and output sequences (Vaswani, et al., 2017). It works by mapping a query and a key-value pair to an output, which consists of a weighted sum of the different value vectors. When query key and value are linear projections of the same data this is called self-attention, when they originate from different data points this is called cross-attention.

The attention mechanism is often combined with a pointwise feedforward network into a transformer block. This architecture forms the foundation of many cutting-edge deep learning architectures such as GPT-3 and the video synthesis system Phenaki (Brown, et al., 2022; Villegas, et al., 2022). Since the transformer architecture is translationally invariant it is not able to account for the order in which the data is presented. To account for this a positional encoding is often added to the input to provide additional information about the location of each item in the sequence.

2.4.4 Deep learning in generative design.

As previously mentioned, deep learning may be applied to automatically extract design representations. However, this is not the only use for deep learning in generative design. Neural networks such as feedforward and generative adversarial networks as well as auto encoders have already been applied in a variety of design problems. Applications, include structural optimization, materials design and shape synthesis (Regenwetter, Heyrani Nobari, & Ahmed, 2022). Typically, the networks are used in one of three ways. They may be trained to generate designs directly using auxiliary losses that model the functional performance of the design. Alternatively, a model may repeatedly be refined by training it on progressively updated datasets or by using the model's outputs to generate new training data. Finally, the latent of a generative model may be optimized, this involves adjusting the latent variables to find configurations that result in desirable generated outputs.

The application of these methodologies in design practice has been so far limited due to four main reasons. First, the environment of design is a relatively data sparse one, making it difficult to train supervised learning-based solutions. Second, conventional deep generative models rely on the replication of existing products through supervised learning based on examples, leading to a creativity gap. A derivative of existing solutions is not always desirable in design. Third, the development of performance aware deep generative models is challenging due to the (computational) cost of numerical simulation and real-life evaluation. Fourth, solutions so far have not represented designs in sufficient detail for them to be manufactured, nor is the manufacturability guaranteed (Regenwetter, Heyrani Nobari, & Ahmed, 2022).

The data scarcity and lack of novelty have a significant impact especially on solutions that rely on supervised learning where the models must extract the underlying distribution of the solution space from training data. These problems may be overcome using reinforcement learning, in which an agent learns an optimal policy by interacting with an environment. Unfortunately, this exacerbates the problem of expensive numerical evaluations since it relies solely on meaningful reward signals which are usually determined through simulation.

These reward signals may be replicated using a surrogate model. A particularly interesting approach to these is the ensemble model, which consists of multiple copies of the same surrogate model (Clavera, et al., 2018). By utilizing multiple models, the regions within the model that are too inaccurate may be identified. If the models agree then the approximation is likely accurate if not, a real sample can be created using the original method and used to further train the ensemble models.

2.5 Human machine interaction.

Design in a multidisciplinary field such as architecture involves a substantial amount of collaboration. Collaboration involves the sharing of ideas, means, and responsibilities in order to neutralize the weaknesses of the collaborating parties (Chiu, 2002). This idea may also be extended to non-human collaborators such as computational tools or artificial intelligence that can provide insights to the designer. However, to collaborate, the parties should first be able to exchange information through communication. Information exchange with a computational tool typically requires extensive knowledge of the software. Additionally, many currently available simulation tools require a detailed description of the envisioned design. These factors undermine effective design since the act of designing requires a fast and intuitive method of information exchange without which the required experimentation is impossible (Van Dooren, Boshuizen, Van Merrienboer, Asselbergs, & Van Dorst, 2013). This lack of intuitive information exchange is a limitation of traditional human-machine systems that are designed for basic command-response interactions (Ren, Chen, & Qiu, 2023).

Although modern deep learning algorithms enable more complex multi-modal interactions, making collaboration with them more intuitive, transparency and interpretability remain significant challenges. These issues limit the application of deep learning in collaborative decision-making environments (Ren, Chen, & Qiu, 2023). The complexity of tackling such a problem on top of the more technical requirements of this work may prevent a satisfactory solution. It will however be noted that fast results and intuitive control are of great importance.

2.6 Summary of findings.

The field of computational design aims to (partially) automate solving design problems. Creating a computational solution often involves specifying design variables, objectives, and constraints. Based on these optimization techniques such as the genetic algorithm or gradient descent, can be applied to generate well performing designs. Traditionally, determining the design variables involved manually programming an explicit design representation. While this approach works well for simple problems, it can result in an unsatisfactory representation of the design space for more complex cases.

Deep learning can be used to automatically extract design representations from data. However, the probabilistic nature of these techniques, may lead to the generation of invalid designs with unpredictable failure conditions. Additionally, the reliance on training data makes supervised learning-based techniques susceptible to data scarcity and unable to create novel designs solutions. These issues may be resolved through the application of reinforcement learning, although this exacerbates the problem of expensive numerical evaluations.

3. The heuristics of floorplan design.

To tackle the challenge of automated floorplan design, the dynamics of the process must first be defined. Important to note is that this is a personal interpretation that lays the foundation for how the sequential process will be defined later. This interpretation affects the functionality of the system and therefore introduces bias.

A floorplan can be understood as a conglomeration of spaces that are linked together in a fixed way. The amount, requirements, and connectivity of these spaces are at the discretion of the architect as an extension of the client. This implies that any system that hopes to be used in practice should give consistent control over these aspects to the user. Two main types of space can be described: the open and the closed space. The boundary condition of these spaces can be extrapolated from these two concepts. For example, if an open space is next to a closed space these two should be separated using a wall containing a door that allows for movement between these two spaces, a partial connection. Unlike the wall between an open and closed space, the wall between two closed spaces should not contain a door. This is because there is no need to move directly between two private rooms such as bedrooms. The last case is the boundary between two open spaces which will be assumed to not contain a wall, a full connection. Using these rules, a map of walls and a graph of an arbitrary floorplan can be constructed as seen in figure 3.

From these floorplan drawings, various indicators such as reference similarity, daylight and carbon footprint can be determined. The methodology of extracting these will be discussed next, starting with reference similarity.

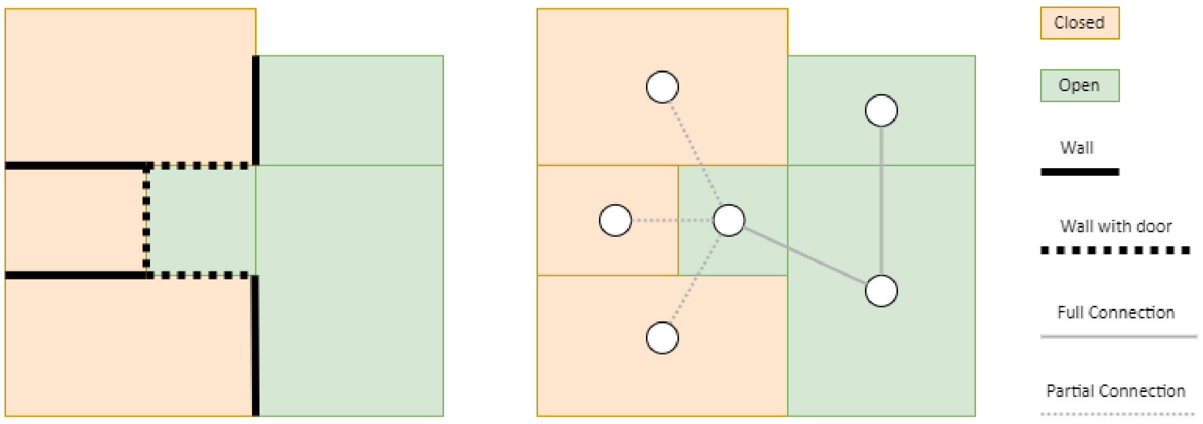


FIG 3 FLOORPLAN WALLS AND GRAPH.

3.1 Reference similarity.

As mentioned previously, the useability of the floorplan generation relies heavily on the controllability of the requirements of spaces and the connectivity between them. To control these aspects, they will first have to be measured, therefore the methodology for achieving this will be discussed in the following section. This will be done by applying the method to a sample from the Swiss Dwelling Dataset (Matthias Standfest, 2022) as shown in figure 4.

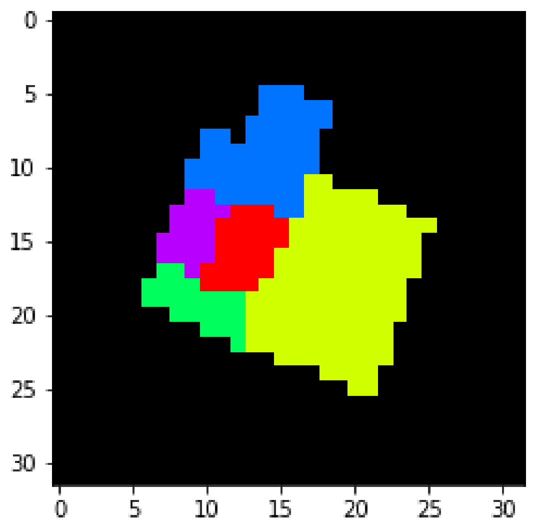


FIG 4 UNPROCESSED SAMPLE SWISS DWELLING DATASET DISPLAYED USING 32X32 PIXELS (Matthias Standfest, 2022).

As seen in figure 4, the sample contains five different spaces each assigned their own channel, thus leading to an array of shape *5x32x32*. In samples containing more spaces, the number of channels would increase to match. It is important to note that, in this case, unlike the usual three, as found in RGB images, there are five channels. To display these channels each one is assigned a colour sampled from the plt.cm.get_cmap('hsv',*N*) function from matplotlib where *N* represents the total amount of channels. In the case of five channels this results in the following colours: red, yellow, green, blue, and purple. The colours are then assigned to the different rooms in this same order room one: red, room two: yellow etc. The composition of this sample can be represented in the form of a composition matrix with size *N*x*N*. Such a matrix can be seen in figure 5. Here rooms that are adjacent to each other are marked with a one, and rooms that are not adjacent are marked with a zero. An observation is that every room is always connected to itself, therefore the ones on the diagonal are redundant. They are replaced with the number of pixels present in each channel. Another observation is that the matrix is symmetric along this same diagonal, therefore half of these values could also be replaced by more useful information this is however not done.

	🟥	🟨	🟩	🟦	🟪
🟥	1	1	1	1	1
🟨	1	1	1	1	0
🟩	1	1	1	0	1
🟦	1	1	0	1	1
🟪	1	0	1	1	1

	🟥	🟨	🟩	🟦	🟪
🟥	36	1	1	1	1
🟨	1	117	1	1	0
🟩	1	1	31	0	1
🟦	1	1	0	55	1
🟪	1	0	1	1	20

FIG 5 COMPOSITION MATRIX EXTRACTED FROM FIG 4 WITHOUT AND WITH NUMBER OF PIXELS ON DIAGONAL.

Although it is easy to see which rooms are connected, an efficient way of determining this computationally is not as straightforward. The methodology settled upon consists of the steps detailed below. A visualization of these steps can be found in figure 6, note that in the example the convolution is shown in only two locations however in the algorithm this is done with stride equals one.

1 Create the composition matrix of shape *N*x*N* containing zeros.

2 Apply to the sample a depth wise 3x3 convolution fully padded with ones. The output of this operation will be flattened and referred to as stamps.

3 For each channel *N* multiply all other channels with the clipped value of channel *N*, such that when the value of channel *N* equals zero, all other channels will also be zero.

4 Sum the result maintaining the channel dimension, thereby getting the number of pixels per channel within stamps also containing channel *N*.

5 Clip the result between zero and one and set the relevant row of the composition matrix equal to this clipped result.

6 Repeat steps 3-5 for each channel.

7 Set the diagonal of the composition matrix equal to the number of pixels per channel.

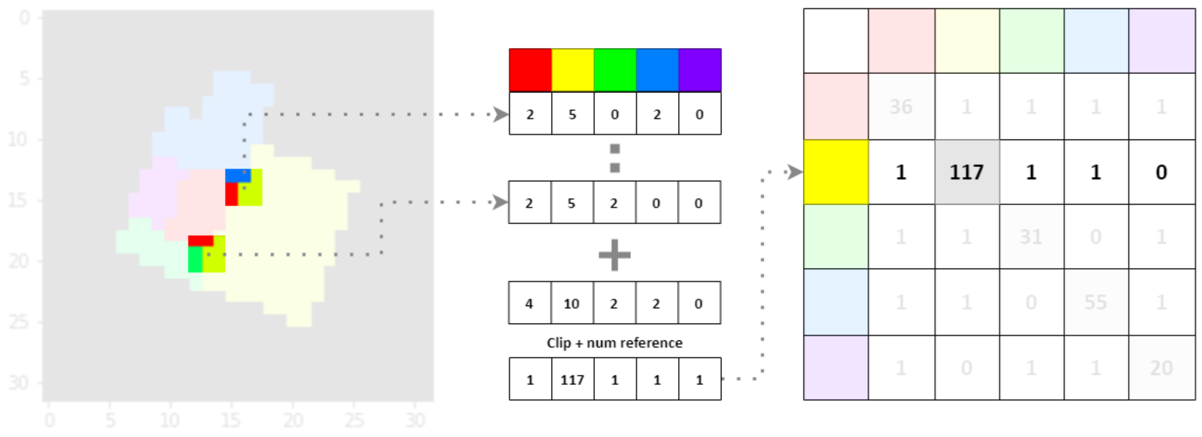
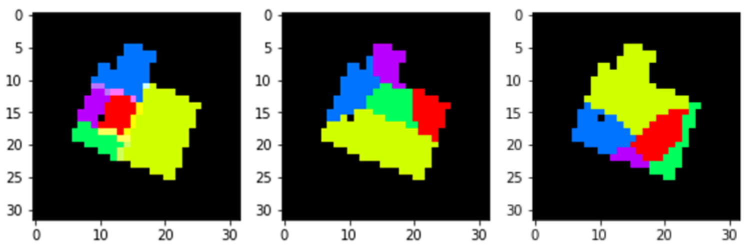


FIG 6 ADJACENCY MATRIX EXTRACTION PROCEDURE SHOWING THE CONVOLUTION ON TWO LOCATIONS.

By constructing such a matrix for multiple floorplans, their adjacency score can be computed by simply taking the mean absolute error between the trilled matrices. A trilled matrix is one where the diagonal and everything to its top right is replaced by zero. In figure 7, the reference floorplan is compared to a generated sample containing the same number of rooms.



	🟥	🟨	🟩	🟦	🟪
🟥	36	1	1	1	1
🟨	1	117	1	1	0
🟩	1	1	31	0	1
🟦	1	1	0	55	1
🟪	1	0	1	1	20

Reference

	🟥	🟨	🟩	🟦	🟪
🟥	36	1	1	0	0
🟨	1	117	1	1	0
🟩	1	1	31	1	1
🟦	0	1	1	55	1
🟪	0	0	1	1	20

70%

	🟥	🟨	🟩	🟦	🟪
🟥	36	1	1	1	1
🟨	1	117	1	1	0
🟩	1	1	31	0	1
🟦	1	1	0	55	1
🟪	1	0	1	1	20

100%

FIG 7 SIMILARITY SCORES FOR DIFFERENT PLANS.

Apart from the adjacency score, an area similarity score can also be computed from this matrix. However, taking the mean absolute error would lead to scaling issues. For example, if a room has an area error of 2m^2, this becomes more significant the smaller the room. Additionally, a 2m^2 room that becomes 1m^2 is significantly worse than that same room becoming 3m^2. To account for this the following formula will be applied $score_{area} = \frac{2}{e^{steepness*error} + e^{-skew*steepness*error}}$, where steepness and skew are parameters that can be tuned depending on user preference. This function is plotted in figure 8 using some arbitrary parameters.

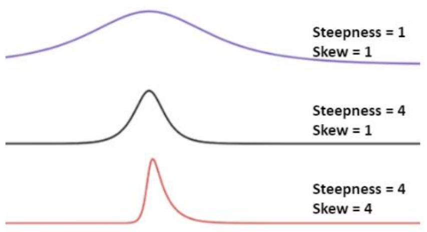


FIG 8 AREA SCORE WITH ARBITRARY STEEPNESS AND SKEW PARAMETERS.

3.1.1 Facilitating control.

The similarity between graphs and areas facilitates the evaluation of designs against the given requirements. For instance, a design brief might specify the need for six rooms: a living room with a 20m² area, two bedrooms each with a 15m² area, a kitchen covering 10m², and a hallway spanning 5m². These spaces, along with their spatial connections and attributes, can be depicted graphically by the user, as illustrated in figure 9. This graphical representation can then be translated into the composition matrix discussed earlier and used for evaluation.

There is one complication, bedroom 1 and 2 may very well be allowed to be adjacent even though they are not linked in the graph. However, if these rooms are adjacent this would be punished using the mean absolute error. This is addressed by using the mean clipped error instead of the mean absolute error. Thus, any adjacencies not present in the reference will lead to an error of minus one, which will then be clipped to zero, eliminating the punishment.

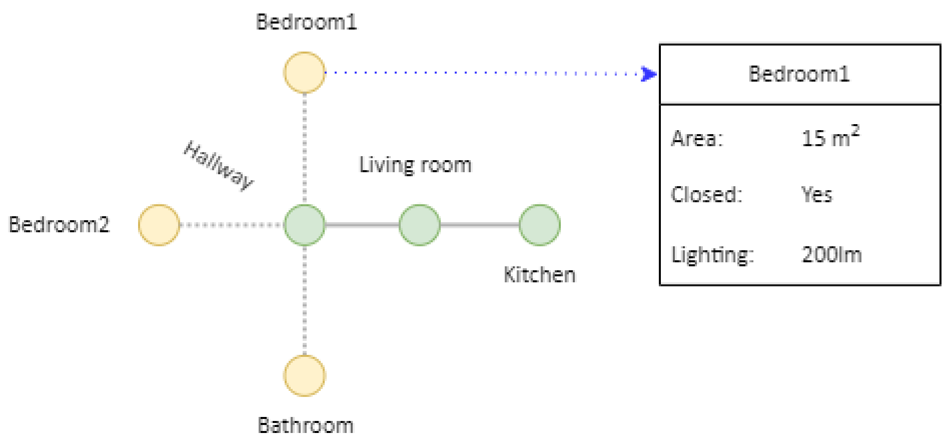


FIG 9: GRAPH AS PRODUCED BY THE USER.

There will be one additional control mechanism, a manual override in the form of a modified Jaccard index. The architect will be able to make a colour coded sketch which will control specific channels by adding the following criterium:

$$score_{similarity} = \frac{Drawing_{architect} \cap Drawing_{machine}}{Drawing_{architect}}$$

Through this similarity score, the system is incentivized to include what the user has drawn into its own design, it may however exceed it to meet other requirements. This effectively kickstarts an iterative collaboration between user and machine. Interesting to note is that these criteria have all been implemented using batched tensor operations in PyTorch allowing them to be executed in parallel, optionally on a CUDA device, thus the time required to compute the scores is negligible. If one is to add additional criteria on top of this it may be wise to use these control criteria as a filter, thus the more demanding computations can be reserved only for comparing samples that at least satisfy the design brief.

3.2 Daylighting.

The daylight score settled upon is based on the exemplary performance criterium embedded in BREAAM-NL which outlines the minimum percentage of a space that has at least three percent of the exterior light levels, these percentages differ by function and can be found in figure 10.

Function	Minimum average daylight factor by function	Minimum percentage of the floor area that satisfies this condition.
Office	3%	80%
Teaching	3%	80%
Shopping	3%	35%
Living space	3%	80%
Visiting space	3%	50%
Meeting spaces	3%	
Daycare	3%	80%
other	3%	35%

FIG 10: TABLE DAYLIGHT CONDITIONS (BREAAM-NL, 2024)

Based on these regulations a score can be calculated using the following equation using the fraction required in this formula is taken from the right most column of fig 8. This will yield a score of zero to one approaching the required percentage and one above it.

$$score_{daylight} = clip(\frac{sum\left(where\left(\frac{light_{pixel}}{light_{exterior}} \geq 0.03, 1, 0\right)\right)}{fraction\_required * total_{pixels}}, 0{,}1)$$

Required to fill in this formula is the light level per pixel which can be determined in various ways with different levels of complexity/accuracy. Within the context of this work, it was chosen to use a method based on projecting geometry onto the floor plane, it therefore does not include any bounce light. Additionally, the facades are assumed to be fully permeable by light, this assumption is made based on the following premise: the light levels inside of the building can be lowered by making alterations to the façade, however, it can never be raised. Therefore, this method gives a lower bound estimate, optionally all interior light levels can be scaled down to create a more conservative solution if this is desirable.

3.2.1 Calculating light levels.

To calculate the light levels, the geometry blocking the light must be defined. This geometry can be divided into three categories: context which doesn't depend on the design, floor slabs which depend on massing, and walls which depend on floorplan segmentation. The context geometry is the simplest to obtain since the meshes can be imported from publicly available sources such as 3D-Bag or alternatively modelled approximately in an external program. The floor slabs can be assumed to have the same shape as the floorplan and can therefore be found by extracting the contours. To obtain the wall geometry a few convolutional kernels are needed; these can be found in figure 11.

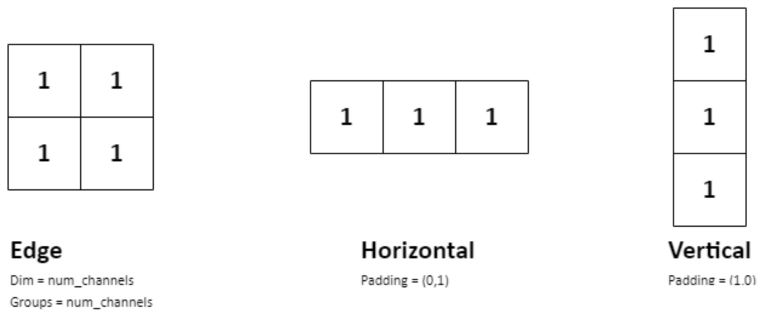


FIG 11: CONVOLUTIONAL KERNELS FOR WALL EXTRACTION NO BIAS ALL UNNAMED VARIABLES ARE DEFAULT.

Using these kernels the walls can be generated using the following procedure:

1 Apply the edge kernel to the floorplan.
2 Sort the channels and select the second largest one.
3 Clip the results between zero and one to obtain the edge map as seen in fig 12
4 Apply the horizontal kernel to the edge map and check where the result equals two to find the end points of horizontal lines as seen in fig 10. Repeat for vertical lines.

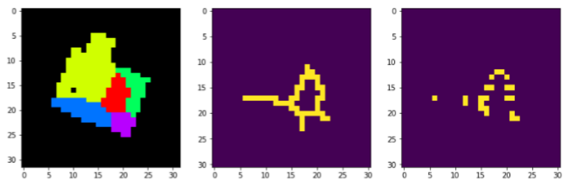


FIG 12: SAMPLE, EDGE MAP AND HORIZONTAL END POINTS.

5 Points are added to a list, since a line cannot end before it begins it can be stated that all even indices including zero are the beginnings of lines and all odd indices are the endings of lines. Using this the walls can be constructed by creating a quad between $point_{begin}$ at z_{floor}, $point_{end}$ at z_{floor}, $point_{end}$ at $z_{ceiling}$, and $point_{begin}$ at $z_{ceiling}$ for each line.

These three types of geometry: context, floorslabs, and walls can then be combined to generate a simple 3d model of the building as shown in figure 13. Notice that the geometry is significantly smoother than in the images, this is due to postprocessing of the geometry.

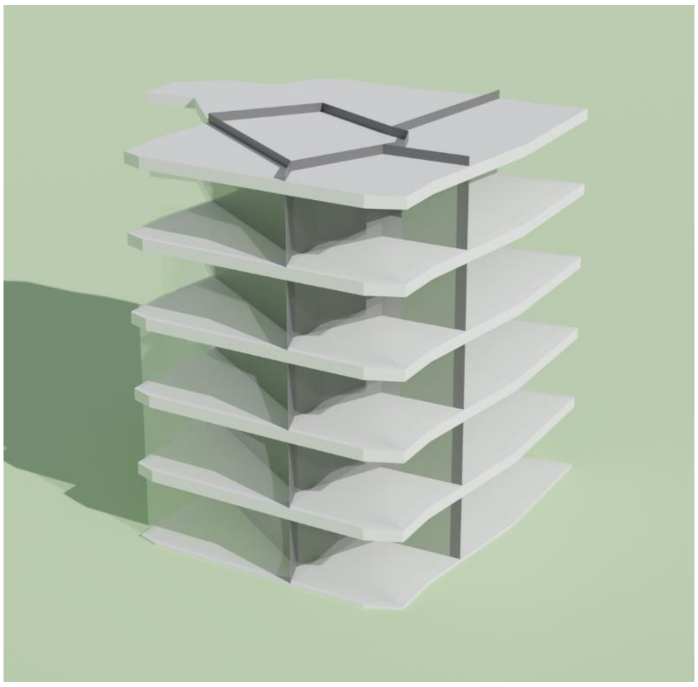


FIG 13: SIMPLE 3D MODEL AS GENERATED FROM SAMPLE DRAWN USING BLENDER.

Using this model, the light levels can be calculated. Typically for the verification of designs, a dedicated light simulation software such as Radiance is used. However, since a low execution time is the priority for this project, another, more approximate method was devised. This method is as follows:

1. Generate a sky dome from a suitable EPW file.
2. Construct a *shadow* array of shape (145, imsize, imsize) padded with zeroes, where 145 is the number of patches in the skydome, and the imsize is the size of the image to which the shadows are to be drawn.
3. For each of the 145 patches project the geometry onto the floor plane along the vector pointing from the centre of the sky patch to the origin and draw the result to the corresponding layer in the *shadow* array using a value of one.
4. For each timestep to be considered, multiply the layers in the *shadow* array with the intensities of the corresponding sky patches.
5. Sum across the layers to obtain the shadows.
6. subtract this from the total light intensity to obtain the light levels on a pixel basis.
7. Mask with the floorplan.

An example of a result can be seen in figure 14, here the drawn shadows are the areas in which the light level does not satisfy the three percent threshold from the criteria. This way of displaying the result is especially helpful because it allows the user to immediately see which areas need adjustment. Surrounding buildings and other obstructions can be included by constructing an additional shadow matrix containing the shadow cast by these objects. The total shadow can then be determined by taking the maximum shadow value between the context and design shadow matrices.

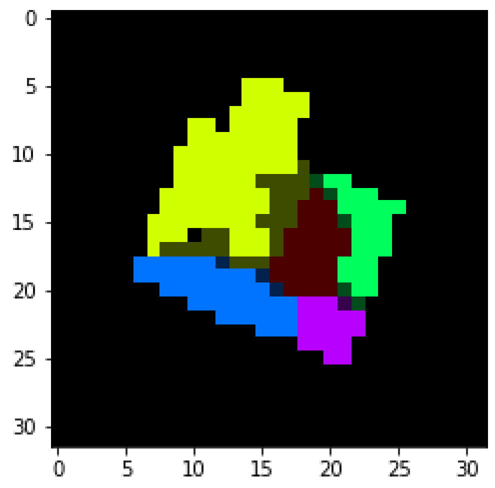


FIG 14: LIGHT CALCULATION RESULTS DISCRETIZED AROUND THREE PERCENT.

3.3 Floor spans and embodied carbon

The span of floors has a significant impact on the embodied carbon of a building. The relationship between span and embodied carbon for various types of concrete slabs is shown in figure 15. This relationship will be used to determine the estimated embodied carbon of a design based on the span of its floorslabs. The associated score will be the inverse of the estimated carbon footprint per square meter. The span of a floor will be estimated using an algorithm which procedurally erodes the space until there is nothing left, with the number of steps required being half of the span. The result of this can be seen in figure 16. This is obviously not very accurate for rooms with complex shapes. However, this inaccuracy is deemed acceptable since it still demonstrates the ability of the agent to account for such criteria. For real life application the criteria could be replaced by more accurate ones since this would not fundamentally change how the model operates.

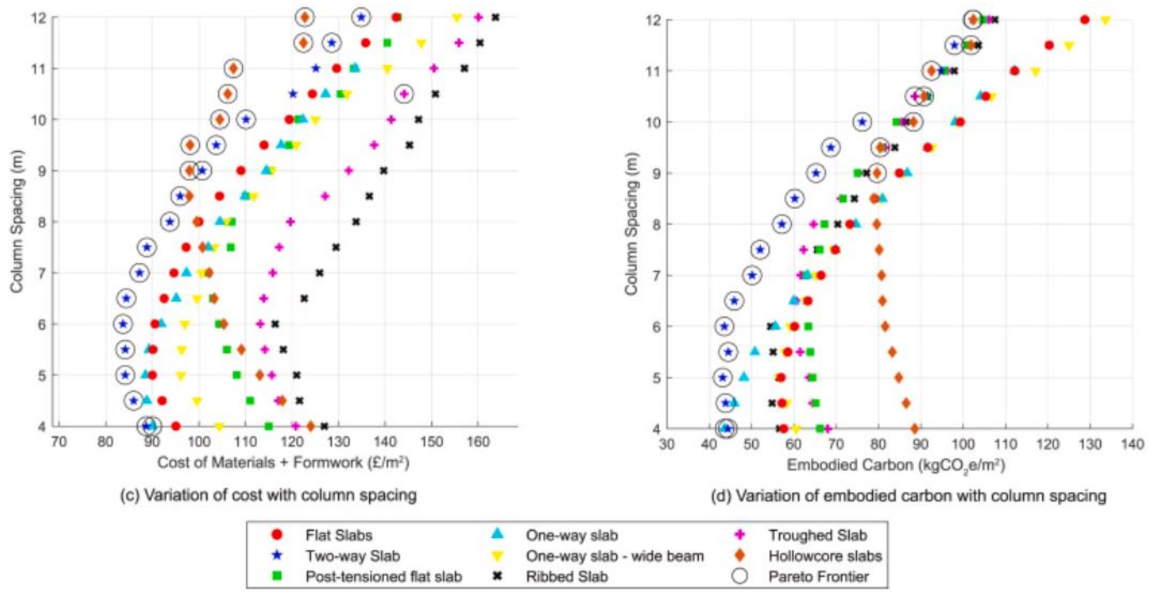


FIG 15 MATERIAL COST AND EMBODIED CARBON AS A FUNCTION COLUMN SPACING, (Yayasinghe, Orr, Ibell, & Boshoff, 2021).

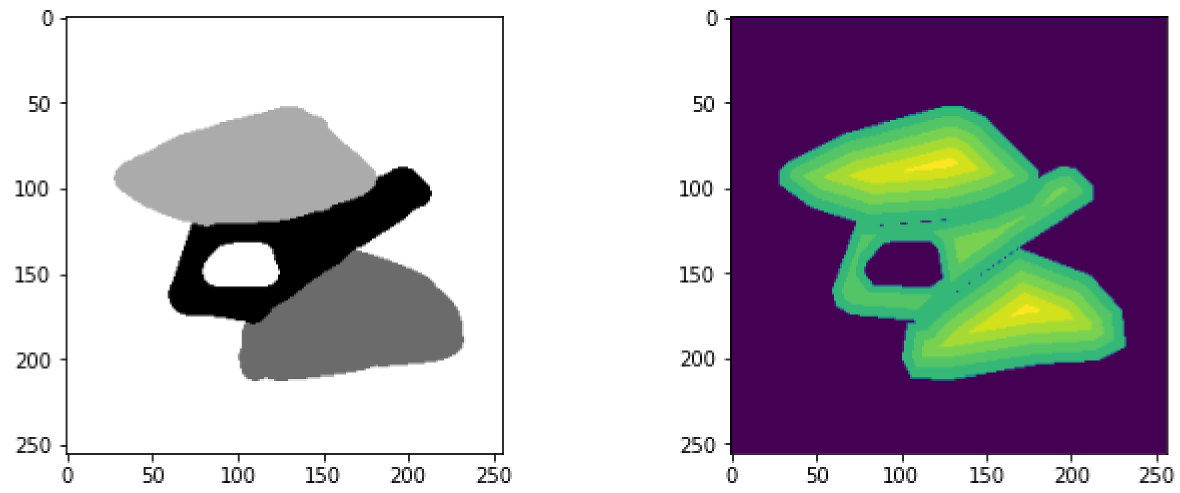


FIG 16: SPANS OF ARBITRARY FLOORSLABS.

3.4 Dropped criteria.

Originally it was planned to include an operational energy score, however, work on this has been suspended. The reason for this is that the assumptions that will be outlined in the next section would, based on a discussion with Martin Tenpierik, have led to very inaccurate results not worth optimizing for. A potential solution would be the introduction of dynamic energy calculations and a more detailed façade modelling workflow. This, while interesting, would have taken up time that was better spent on the development of the solvers discussed in this report. This is compounded by the speculation that the time required the execute these calculations would likely have made it unsuitable for use in a generative approach. This could be addressed using a surrogate model which would make a wonderful thesis subject for a future student.

The original proposal was to estimate the energy use of proposed designs and a simple static energy calculation will be made based on three assumptions. Firstly, the window to wall ratio will be assumed to be a function of the climate and orientation of the façade. To account for this, the intensities of the irradiance obtained from climate data will be scaled based on the direction of the vectors associated with them. The exact relationship between orientation and window to wall ratio leading to maximum energy savings will have to be determined empirically. This scaling is assumed to yield a reasonable approximation of the effect caused by the window to wall ratio, some inaccuracy is considered acceptable since it can be addressed later in the design process. Secondly, the irradiance entering the building envelope will be assumed to be the total irradiance from which the projections of obstructions, such as walls and floorslabs onto the floor, are subtracted. Thirdly, the Rc values for façade (Rc4.7), roof (Rc6.3), and foundation (Rc3.7) are assumed to be complying exactly to *bijna energie neutraal gebouw* (BENG) regulations (van Oranje-Nassau, 2023).

4. Why reinforcement learning?

Many techniques that have thus far been applied to tackle problems in the field of deep generative design rely on the use of (conditional) generative adversarial networks, variational autoencoders or diffusion models that are trained in a supervised fashion using training data (Regenwetter, Heyrani Nobari, & Ahmed, 2022). Although these are data driven techniques, it is possible to apply these same frameworks in a performance driven way. This is often done by manipulating the latent space of the models, which can be achieved through gradient descent as shown in previous work (Pavlidou, 2022; Sterrenberg, 2023). There is also an alternative to gradient descent that could be used for non-differentiable objective functions which is the genetic algorithm (Deb, Pratap, & Agarwal, 2002). While these techniques have been effective at solving a large variety of problems, they cannot be applied effectively when data is sparse and the solution space complex. Due to the nature of the supervised learning frameworks on which they rely, a large quantity of training data is required to ensure full coverage of the solution space.

An alternative approach would be to apply reinforcement learning (RL), in which an agent learns an optimal policy by interacting with an environment. In order to apply this technique, the problem and its potential solution have to be formulated in a sequential process in which an agent takes an *action* which causes the environment to transition from *state*$_t$ to *state*$_{t+1}$ for which the agent receives a *reward.* This interaction is typically formulated in the form of a Markov Decision Process (S,A,P_a,R_a) where S is the state, A is the action, P_a is the probability that action A will cause state S_t to transition to S_{t+1}, and R_a is the (expected) reward given to the agent for causing transition S_t -> S_{t+1} through action A. The benefit of this approach is that it can be applied to any problem on the condition that the environment and rewards can be modelled, even if there is no relevant training data available.

An example of how this can be applied to image-based problems, such as floorplan generation, can be found in the paper: *Learning to Paint with Model-based Deep Reinforcement Learning* (Huang, Heng, & Zhou, 2019). In this paper the authors describe how an agent is taught to recreate images using strokes. This methodology is very different from typical data generation techniques, such as variational autoencoders, which learn to represent the solution space in a latent vector using convolutional neural networks. This difference enables application to data scarce problems. While the variational encoder must learn the full distribution of the solution space from training data, the stroke-based technique can generate any sample that can be created using a sequence of strokes without relying on training data. Additionally, this is not limited to strokes but can be generalized as: the agent can create any sample that can be created through a sequence of actions. Therefore, the same technique could be applied to different data representations, for example graphs. This is only valid on the condition that a set of actions can be described that will enable the full manipulation of the solution space.

4.1 Policy optimization algorithms

Solving any problem using RL involves one or multiple agent(s) learning a *policy* that maximizes the cumulative *reward*. The *policy* is a function that determines the *action* to be taken based on the current *state*. This policy can be learned using different algorithms such as Proximal Policy Optimization (PPO), Deep Deterministic Policy Gradient (DDPG), Soft Actor Critic (SAC), and many more. There are no set-in stone guidelines through which one can pick the most suitable algorithm, however each has specific (dis)advantages which will be discussed in the next section.

4.1.1 PPO.

Proximal Policy Optimization (PPO) is a model free, on policy actor-critic algorithm that can be used for both continuous and discrete action spaces. It updates the policy by maximizing a surrogate objective function which aims to increase the probability of advantageous actions occurring under the new policy. The training is stabilized by clipping the objective function to prevent a change in the policy that is irresponsibly fast. The main advantages of PPO are its simplicity and applicability to very general settings, such as when the actor and critic share parts of their network architecture, and the fact that it is quite sample efficient compared to other on policy methods (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017).

4.1.2 DDPG.

Deep Deterministic Policy Gradient (DDPG) is a model free, off policy actor-critic algorithm that can be applied to problems with both continuous states and actions. It applies the actor critic approach used in Deterministic Policy Gradient (DPG) and adds neural network function approximators to account for a continuous state/action space as well as a replay buffer to enable batching. Both of these changes are taken from Deep Q learning (DQN) thereby adding the deep to Deep Deterministic Policy Gradient. Two additional mechanisms are the introduction of target networks and action noise. The target networks act to improve the training stability by making copies of the actor and critic networks.This copy is called the target network, which is used to sample the environment while being slowly updated to match the weights of the original network. The action noise is implemented to facilitate exploration. The noise is simply added to the generated actions before sampling the environment and is determined using the Ornstein-Uhlenbeck process which includes inertia. The off-policy nature of the algorithm gives it improved sample efficiency when compared to PPO (Lillicrap, et al., 2015).

4.1.3 SAC.

Soft Actor Critic (SAC) is in many ways similar to DDPG, as both are off-policy algorithms that use the actor critic concept to solve problems with continuous states and actions. The main difference lays in the way in which the two algorithms facilitate exploration. Unlike DDPG which adds noise to the actions, SAC adds a measure of entropy to the objective function which directly rewards exploration. The influence of the entropy on the objective function is often regulated to target a specific amount of exploration, this influence is called the temperature. If the entropy of the policy is too high the temperature is adjusted down to reward a more deterministic policy while the temperature is increased when entropy is too low rewarding a more stochastic policy (Haarnoja, et al., 2018). SAC is claimed to outperform DDPG especially in complex environments, due to its more effective exploration.

4.2 Timeseries processing

To address the problem of floorplan generation the agent must be able to correlate spatial features to the assigned performance criteria. Since the criteria and underlying conditions can be changed between episodes the agent will have to uncover this relationship by exploring it within one episode, this necessitates a form of timeseries processing. There are two main ways in which memory is implemented in machine learning problems: the Long- Short-Term Memory (LSTM) and the Transformer (Li, et al., 2023). The most popular choice is the LSTM, which is a form of recurrent neural network that alters an internal state at each timestep while ensuring constant error flow to improve stability. The internal state can be used to create a dense representation of a timeseries. This makes it very useful in problems like time series prediction and reinforcement learning problems in which information of past states is needed (Staudemeyer & Morris, 2019).

An alternative to the LSTM is the Transformer, which instead of modifying an internal state, considers all previous states and correlates them with each other using the attention mechanism (Li, et al., 2023). Transformers are state of the art when it comes to time series problems and form the backbone of well-known systems such the language model Generative Pre-training Transformer (GPT) (Brown, et al., 2022) and video synthesis model Phenaki (Villegas, et al., 2022). Although the Transformer is superior to the LSTM in encoding long dependencies, it can suffer from poor data efficiency making it expensive to train, especially in on policy reinforcement learning systems (Li, et al., 2023).

4.3 Collaboration

The environment that was described in a previous chapter relies on a certain amount of collaboration between different agents. Such collaboration is achieved if the agents can reach the Nash equilibrium point, which is the state in which none of the agents have an incentive to deviate from the current state. In this special fully collaborative case, agents typically share a *common reward* function, such a setup is referred to as a Multi-agent Markov Decision Process (MMDP) (Zhang, Yang, & Basar, 2021). This is a relatively simple case where single agent RL algorithms can be applied with minimal modification. A slightly different version uses a *team-average reward*, this allows different agents to use different reward functions. The implementation of this can also be done using single agent RL algorithms but may necessitate the use of a communication protocol between agents. A particularly interesting implementation of communication is one where the agents will not only take an action but also broadcast a message at each timestep. The messages from each agent can then be compressed into a fixed size vector using for example a recurrent neural network and broadcast to all agents in the next timestep (Zhang, Yang, & Basar, 2021).

5. Designing the model.

In this section a preliminary outline of the model will be laid out. It is worth noting that the exact composition is most likely going to be changed during the implementation phase of the project. The design of the model must account for various aspects of the environment. Firstly, the model should observe the state of the environment. Secondly, the model should correlate performance with geometry dynamically. Finally, communication between agents should be possible to facilitate collaboration. Inspiration for this setup has been taken from AlphaStar (Vinyals, et al., 2019) and Learning to Paint With Model-based Deep Reinforcement Learning (Huang, Heng, & Zhou, 2019). A diagrammatic overview of the task that the agent should perform can be seen in figure 17.

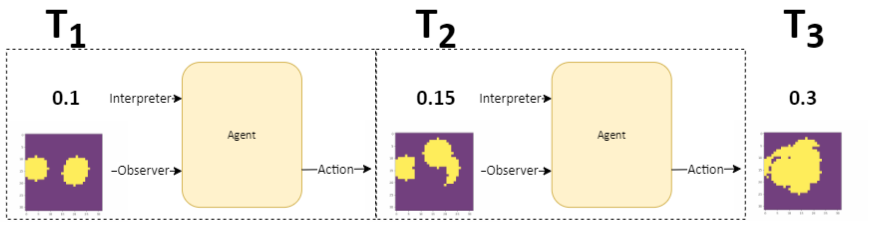


FIG 17: AGENT TASK OVERVIEW.

Representing an image in a compressed latent vector is a problem for which a robust solution is known. Image compression is usually achieved using a cascade of convolutional layers that progressively compress the image further and further until a target size is reached; such a setup is called a convolutional encoder. This encoder can either be pretrained using a Variational Auto Encoder (VAE) or trained simultaneously with the rest of the model. Since the implementation of a VAE is relatively straightforward, the encoder will be trained separately since this may increase convergence speed due to a reduction in trainable parameters.

The communication and evaluation vectors will be concatenated together with the latent vector produced by the convolutional encoder. This concatenated vector can then be processed in a way that includes information from previous timesteps. There are two methods of doing this, which have been discussed previously. The conservative approach would be to utilize a LSTM which is a popular choice for similar applications. The alternative approach would be to use a Transformer which has been shown to outperform the LSTM in time series related problems, such as natural language processing. This approach is however not as mature within the field of reinforcement learning, which may cause complications. Additionally, the Transformer is known to require more training than a LSTM.

An interesting aspect of this setup is that the three model heads: the Messenger, Actor and Critic share a considerable portion of their architecture. This makes the application of the PPO training algorithm very tempting since this is inherently compatible with shared components between Actor and Critic. A complete overview of the network can be seen in figure 18.

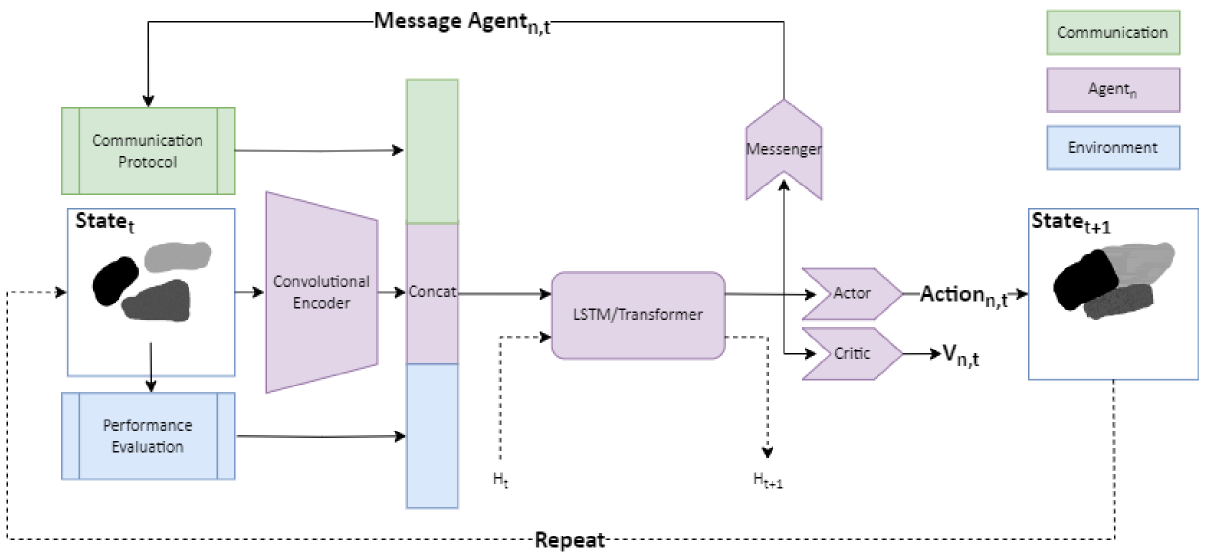


FIG 18: MODEL CONCEPT OVERVIEW.

5.1 Training concept

Training a model of this complexity will no doubt present a challenge. Throughout the construction of the model elements will be pre-trained separately as much as possible. This will be done both to reduce instabilities and to verify that the different elements have been implemented and tuned correctly. An interesting methodology could be to treat the convolutional encoder and LSTM/Transformer as a world model in a similar fashion to how this is done in DreamerV3 (Hafner, Pasukonis, Ba, & Lillicrap, 2023) as seen in figure 19.

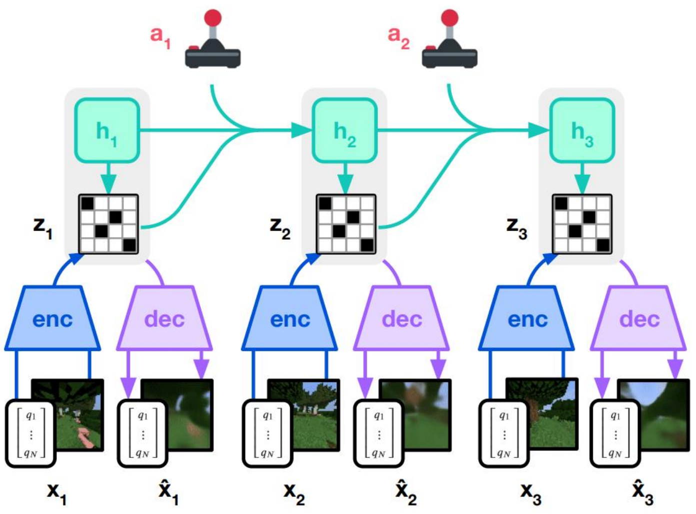


FIG 19: WORLD MODEL TRAINING (Hafner, Pasukonis, Ba, & Lillicrap, 2023).

By treating the world model as a separate module, it can be trained in a supervised manner. In Dreamer, the training procedure works as follows: the world state s_t is encoded into latent representation Z_t, this is fed into a recurrent neural network (RNN) alongside action a_t. Using these two vectors the RNN must predict Z_{t+1}. The result of this prediction can of course be verified by sampling Z_{t+1} using samples collected in a replay buffer. The actor and critic networks can then be trained based on simulated trajectories using the world model. In DreamerV3 the gradients are not shared, through which the training is stabilized significantly. This autoregressive approach used for learning the world dynamics could of course be applied to both LSTM and Transformer-based solutions. In the case of a transformer, this would be done by masking, as done in large language models such as GPT (Brown, et al., 2022). The benefit of using a transformer would in this case be that it has complete access to all previous states and not just their representation within the LSTM hidden state. It is worth mentioning that it may not be necessary to predict future states based on actions but instead predict another metric contained in a future timestep, such as the score. This is possible because the dynamics of the interaction with the environment are consistent. This means that modifying an image in a certain way will always lead to the same result, only the associated score may be different.

5.2 Differentiability

The ability to fully differentiate and train the model in an end-to-end fashion may be beneficial to model performance and convergence (Clavera, et al., 2019;Huang, Heng, & Zhou, 2019). To this end, the agent will manipulate the canvas using a neural renderer, which replicates the behaviour of a deterministic painting tool. A neural renderer can be trained by randomly sampling actions through traditional painting software and then using a convolutional decoder to imitate the results.

The performance metrics may be replicated using a method called ensemble models. An ensemble model consists of multiple copies of the same surrogate model (Clavera, et al., 2018). The reason for using multiple models is that this allows the identification of regions within the model that are too inaccurate. If the models agree then the approximation is probably accurate and if not, a real sample can be created using the original method and used to further train the ensemble models. This does not only enable end-to-end training, but also efficient use of computational resources. The models can be trained in a decentralized way on different machines using supervised learning based on centrally stored experience replay data. It is worth noting that backpropagating through complex world models may cause exploding or vanishing gradients (Clavera, et al., 2018).

6. Developing the model.

Now that a general architecture of the model has been established, the development can begin. The model can be split in four different modules: observation, interpretation, action, and communication. These modules can partially be developed separately, which is helpful since the reduction in complexity allows issues to be tracked down and addressed faster. Bear in mind that the following subchapters are presented categorically and not in chronological order since the various modules were developed concurrently.

6.1 Observation

Observation in the context of this model refers to the extraction of feature vectors from the images that make up the environment. There are two main categories of networks that can be applied to this problem: convolutional neural networks (CNN) and vision transformers (ViT). The CNN used to be the default approach for this. However, recently ViTs have been outperforming them both in accuracy and execution speed (Dosovitskiy, et al., 2020). This is nevertheless not without caveats since unlike the CNN, the ViT does not have any inductive bias towards neighbouring pixels, nor does it possess translational equivariance. These characteristics cause the ViT architecture to have to learn all spatial relationships from scratch, which significantly increases the required amount of training samples.

There is another aspect to consider, although ViTs outperform CNNs on datasets such as CIFAR and ImageNet, the complexity of floorplans lies in the composition and not as much in the high level of detail that can be found in natural images. Therefore, it is entirely possible that the difference in performance between the two architectures is negligible in this context. Combined with the previously mentioned increase in required training samples, there is a case to be made that a CNN is more suitable in this case. This however does not mean that a ViT based observer is not feasible for this application.

6.1.1 CNN design

The specific CNN architecture that will be implemented is the ConvNeXt model (Liu, et al., 2022). The only alteration is that a batchnorm will be used instead of the layernorm because the layernorm caused instabilities. An interesting characteristic of this block is its use of depth wise convolutions followed by pointwise convolutions. Although this is unusual and leads to some accuracy loss, it is much faster. The lost accuracy is recovered through the application of an inverted bottleneck, in which the data is first scaled up before it is scaled down. The final convolutional block can be seen in fig 20. After a variable number of repeating blocks, the size of the sample will be reduced or increased by a factor two and the number of dimensions N changed. This size reduction will be done using a convolutional layer with size and stride two. The increase will be achieved by a transposed convolutional layer with size and stride two. The full network architecture can be described as a list of depths and dims, which refer to the number of repeats per layer and the dimensionality of those layers respectively. For example, an encoder with depths = [3,1,1] and dims = [4,16,32] will have 3 repeating blocks of dimension 4, one block of dimension 16, and one block of dimension 32. For a decoder the lists are simply reversed.

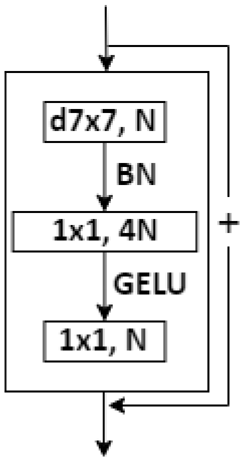


FIG 20: CONVOLUTIONAL BLOCK WHERE N REPRESENTS THE DIMENSIONALITY BASED ON (Liu, et al., 2022).

Using these parts, an autoencoder can be constructed. This may however not be suitable for this application because the latent space of an autoencoder is typically not smooth or evenly distributed. Thus, datapoints close together in latent space may represent very different samples. This property limits the interpretability of the latent vectors, and interpretation is exactly what they would be used for. This can be addressed by regularising the latent space through the introduction of the Kullback-Leibler (KL) divergence to the loss function creating the variational autoencoder (Kingma & Welling, 2013). Instead of producing the latent vectors directly, the encoder will instead give the means μ and variances σ of a multivariate normal distribution which will be used to sample from the latent space. These distributions will be subject to the KL-divergence to enforce a normally distributed latent space.

However, there is an alternative approach called the vector quantised variational autoencoder (VQ-VAE). This model has specifically been designed for extracting meaningful representations in the form of discrete latent vectors using the application of a codebook (van den Oord, Vinyals, & Kavukcuoglu, 2017). The codebook contains a finite number of vectors to which the encoder outputs are mapped. After sampling, the encoder outputs are moved towards the nearest codebook vector through the commitment loss. Simultaneously, the nearest codebook vector is moved towards the encoder output using the VQ-Loss. This method is especially interesting for this application because instead of passing the latent vector directly to the interpreter, the codebook index representing it can be passed instead. This allows the interpreter to have its own equally sized codebook containing different vectors that are more suitable for the relevant architecture.

6.1.2 VAE vs VQ-VAE

Both the VAE and VQ-VAE may function well for this problem. Therefore, they will be compared on their ability to assist the interpreter at making score predictions, their architectures are shown in fig 21. During this test, the interpreter will remain unchanged and will consist of a masked transformer with a hidden size of sixteen and four layers, this transformer is combined with a score predictor block which consist of three fully connected layers with a GELU activation function and dimensions: 32, 128, and 1. The interpreter design will be further discussed in chapter 5.2.

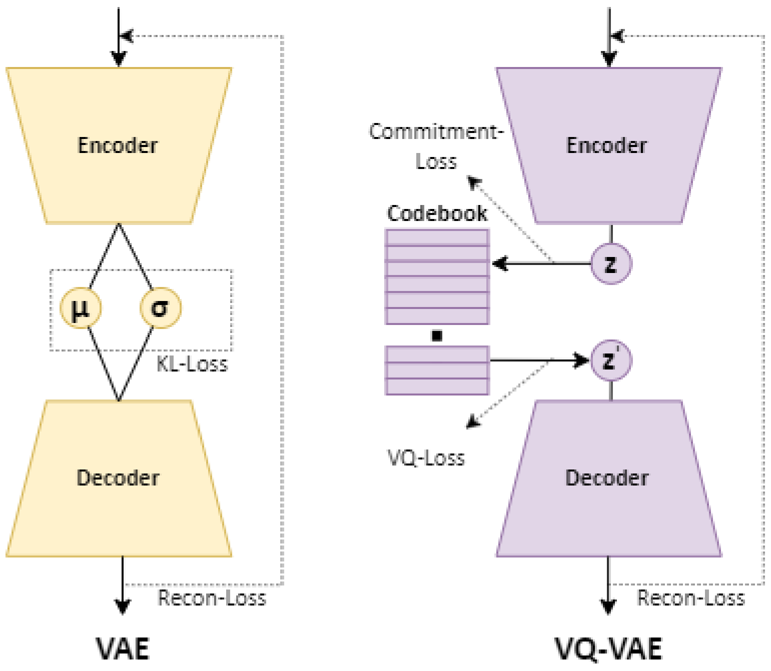


FIG 21: VAE AND VQ-VAE (Kingma & Welling, 2013) (van den Oord, Vinyals, & Kavukcuoglu, 2017).

The test works as follows:

1. An episode containing fifty samples is created in a stochastic fashion and compared to a reference sample using the Jaccard index to obtain a score.
2. All samples are fed through the VAE or VQ-VAE to obtain a latent vector of size sixteen. Bear in mind that the VQ-VAE, instead of the latent, will provide sixteen indices which correspond to vectors of size one in another codebook attached to the transformer.
3. To these latent vectors is added a score embedding, which contains the score projected to a vector of size sixteen using a fully connected network with dimensions one and sixteen. Thus, creating a series of in this case fifty inputs of size sixteen for the transformer.
4. The transformer produces fifty vectors in an autoregressive manner, such that vector T_n is conditioned on inputs $T_{\leq n}$. Thus, output vector T_2 is only conditioned on input T_1 and T_2 not T_3.
5. The output from the transformer is then concatenated to the shifted unaltered encoder latent vectors, such that the transformer output of T_n one is concatenated to the encoder output of T_{n+1}.
6. This is fed through the score predictor to predict the score of sample T_{n+1}.

After training the transformer to predict the scores of T_{n+1}, the predictions of the models based on the VAE and VQVAE can be compared, this can be seen in fig 22. In this graph the real scores per sample are displayed in orange while the predictions are shown in blue. A perfectly performing algorithm would produce a blue line that perfectly aligns with the orange one. Although both variants perform reasonably well the VQ-VAE performs better with a mean error of ±10% compared to the VAEs mean error of ±20%. These values are rounded significantly because there is a fair amount of variation between episodes especially for the VAE. On the contrary, the VQ-VAE is very consistent between different episodes, which can be seen in figure 23. It is worth noting that the error rapidly reduces as a function of the number of samples available to the transformer, this can be seen in figure 24. The VQ-VAE will be used as the observer for the remainder of the project.

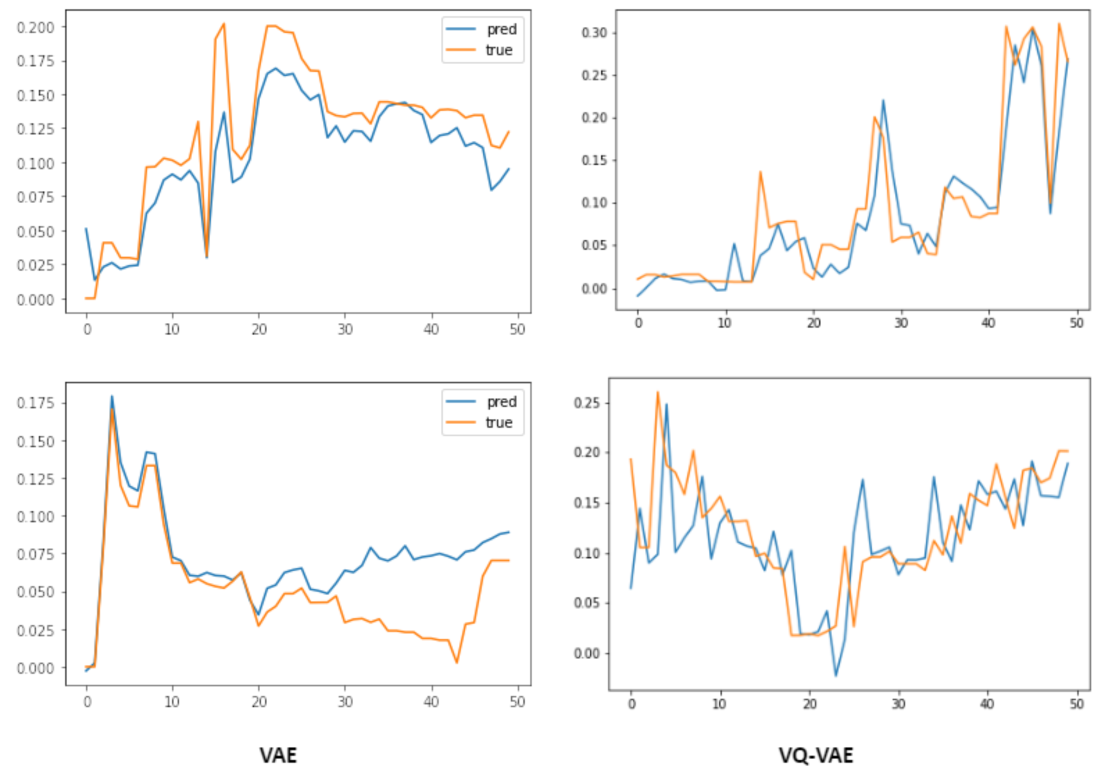


FIG 22: VAE VS VQ-VAE IN SCORE PREDICTION.

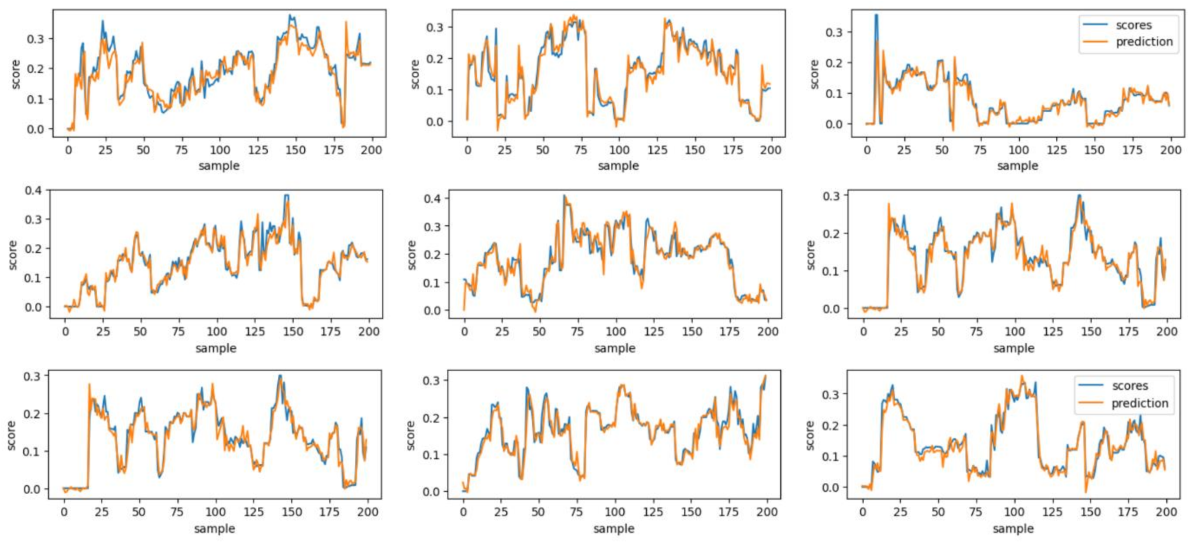


FIG 23: VQ-VAE PREDICTIONS FOR NINE DIFFERENT RANDOM EPISODES.

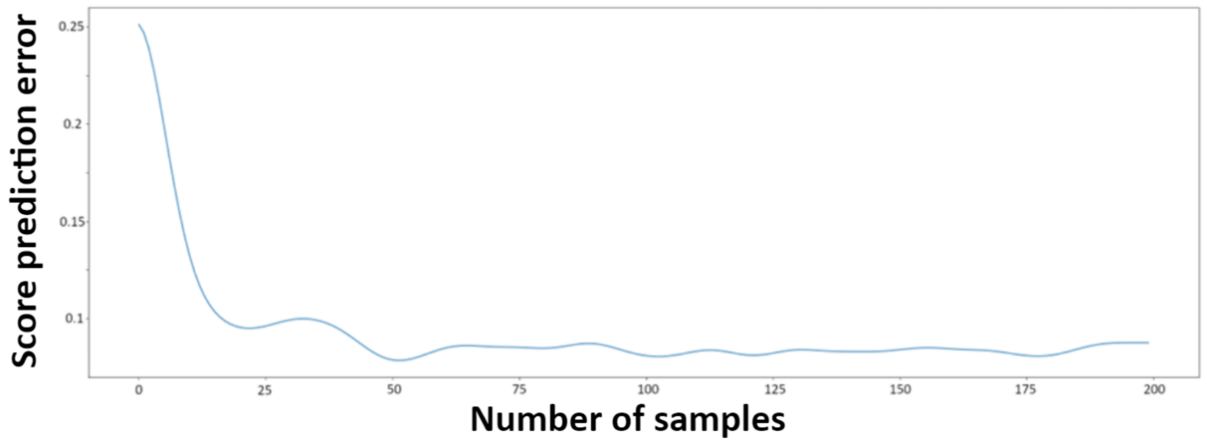


FIG 24: MEAN PREDICTION ERROR REDUCING AS A FUNCTION OF SAMPLE COUNT USING VQ-VAE.

6.2 Interpretation

For the interpretation of timeseries there are, as previously mentioned, two suitable system architectures: the long-short term memory (LSTM), and the multi headed self-attention Transformer. Although both systems can be applied to the same problem, they function in a fundamentally different way, a diagrammatic overview of how they can be used to interpret timeseries can be seen in figure 25. The LSTM keeps track of previous timesteps by modifying a hidden state, this hidden state is iteratively updated throughout every episode. The Transformer on the other hand has access to all steps simultaneously and will extract relevant information from each step using the self-attention mechanism. A small adjustment must be made to stop it from accessing future steps. This is achieved by setting the attention scores for all future steps to zero using a triangular mask, in the same way as done in GPT (Brown, et al., 2022).

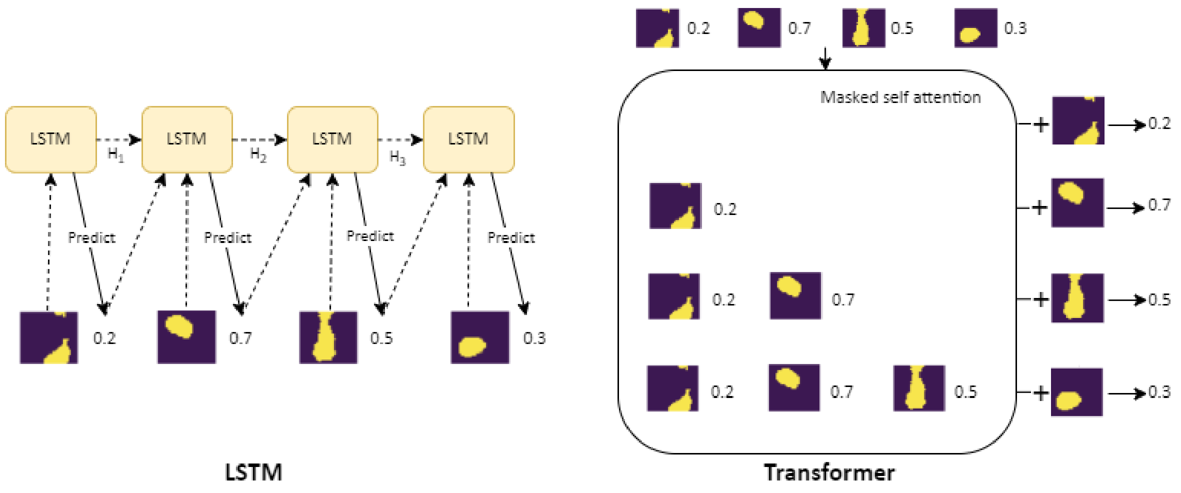


FIG 25: LSTM AND TRANSFORMER ARCHITECTURES FOR SCORE PREDICTION.

6.2.1 LSTM vs Transformer

Based on literature, the expectation is that the Transformer model will significantly outperform the LSTM (Brown, et al., 2022; Li, et al., 2023). However, the increased training data requirements usually associated with transformers may make it too difficult train in a reinforcement learning setting. To compare the models the test conducted in chapter 5.1.2 will be repeated, this time however, the observer will not be changed. The observer used is the VAE used in the test of chapter 5.1.2. The idea behind using this test as a proxy for the performance of the system in a generative setting is as follows: if the interpreter can predict the scores, then it must be able to extract the relationship between shape and score. Assuming this is the case, it should be able to use this insight into this relationship to choose an appropriate action.

The LSTM used will be one with dimension sixteen and one layer, the transformer on the other hand will have a hidden size of sixteen and four layers. The interpreters are yet again combined with a score predictor block which consist of three fully connected layers with dimensions: 32, 128, and 1. The results of this test can be seen in figure 26. The LSTM has a mean error of ±30% while the transformer does noticeably better with a mean error of ±20%, again these values are rounded significantly because there is a fair amount of variation between episodes. Additionally, the transformer is much more consistent, which was further improved by replacing the VAE with a VQ-VAE as discussed previously. The real surprise is that the transformer was faster to train as well. This is most likely the result of being able to use the hardware more efficiently since the transformer can predict all timesteps simultaneously while the LSTM must look at the steps one at a time to obtain the new hidden states. As a result of the higher accuracy and reduced training time, the Transformer will be used as the interpreter for the remainder of the project.

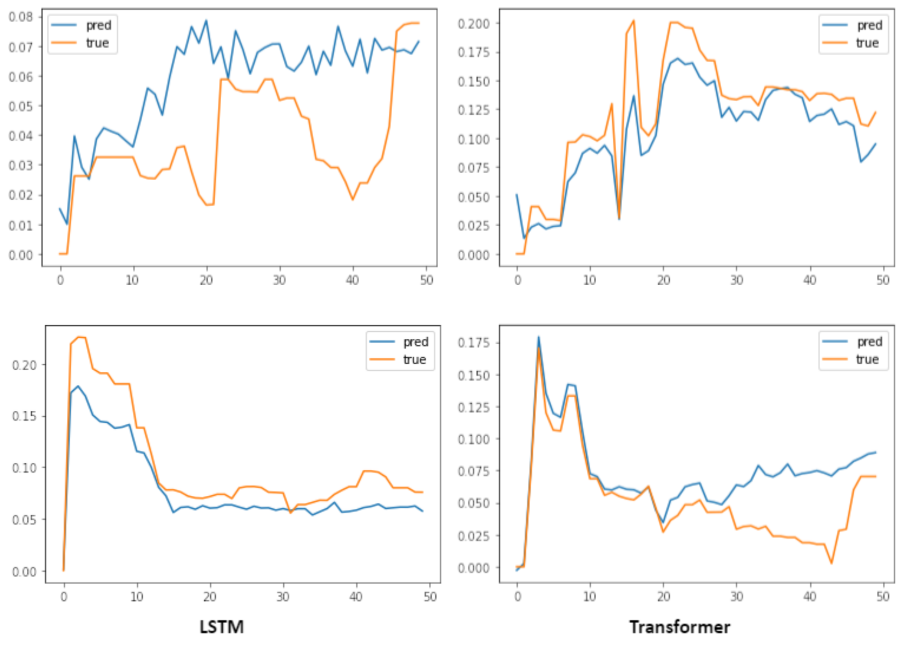


FIG 26: LSTM AND TRANSFORMER IN SCORE PREDICTION, SAMPLE ON X, SCORE ON Y

6.2.2 Network layers and hidden size

The interpreter network has a variable number of layers and size. Therefore, to ensure optimal performance the influence of these parameters will have to be investigated. To do this the previously used score predicting test is performed repeatedly with varying sizes and depths to see which combinations perform best. First the number of transformer layers, the tests were run with four layers. This number was chosen since this is the number of layers used in the causal transformers for the video generation algorithm Phenaki (Villegas, et al., 2022). This seemed like a good starting point since the use case is somewhat comparable to this one, namely, a sequence of images is being represented in a compressed format. To further investigate the influence of the number of layers, a handful of numbers in a similar range were chosen, specifically 2, 5, 8 and 11, the results of this test can be seen in figure 27. To make sure that the difference between the layers is purely coincidental, the models were trained twice. The model containing five layers performed best, however it is still possible that models with three to seven layers still perform better. Therefore, another trial was run using number of layers: 4, 5, and 6, these results can be seen in figure 28. In these results there is no clear pattern to be recognized, indicating that the difference between the models is negligible. As a result, four layers will be used in the future since this will execute faster with no apparent loss of accuracy. The same procedure was used to identify the ideal hidden size of the transformer. However, any of the sizes that were feasible for the convolutional encoder, namely sixteen and above, seemed to perform equally well. The hidden dimension of the transformer will thus be set equal to the output dimension of the convolutional encoder.

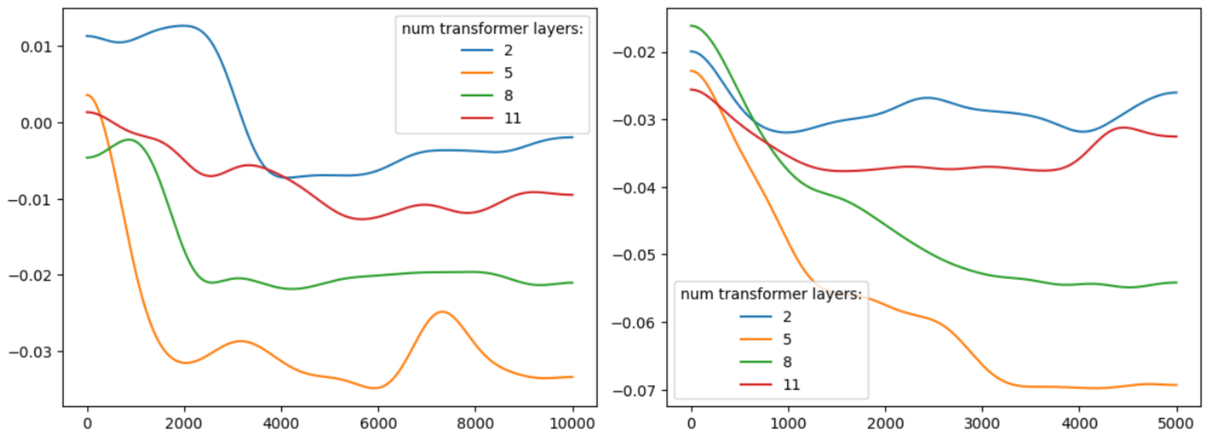


FIG 27: TRANSFORMER TRAINING WITH VARYING NUMBER OF LAYERS, NUMBER OF STEPS ON X, RELATIVE ERROR ON Y.

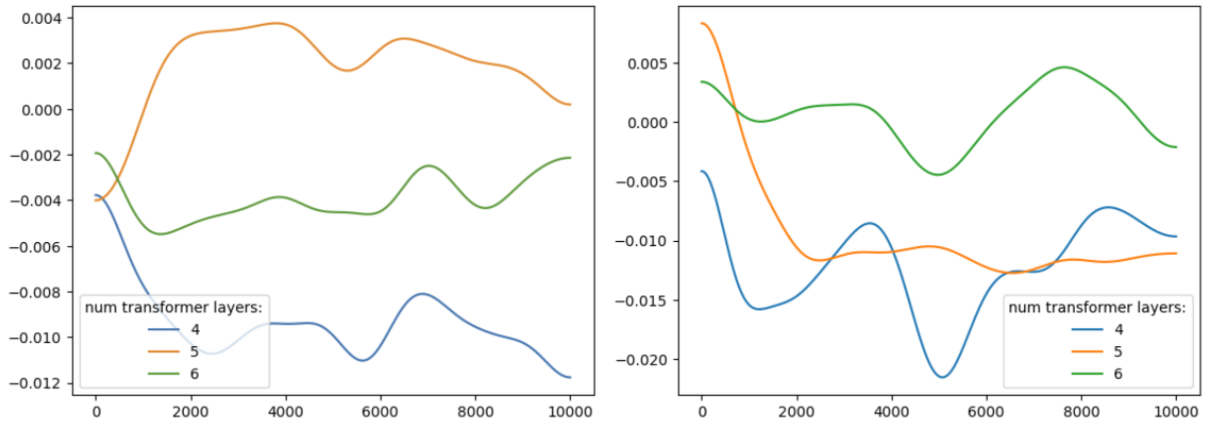


FIG 28: TRANSFORMER TRAINING WITH VARYING NUMBER OF LAYERS, NUMBER OF STEPS ON X, RELATIVE ERROR ON Y.

6.3 Action

Now that the observer and interpreter architectures have been established, they can be added to the agent network shown in fig 29. The modules used for testing the observer and interpreter are coloured purple, this group of modules will be referred to as the world model. It is possible and potentially beneficial to train these modules in a supervised manner on data from previous episodes saved to a buffer. This will enable the batch size to be increased since sampling the environment would otherwise probably be the bottleneck. Additionally, the use of a replay buffer prevents overfitting the world model to the states common in the current policy which would increase training instability.

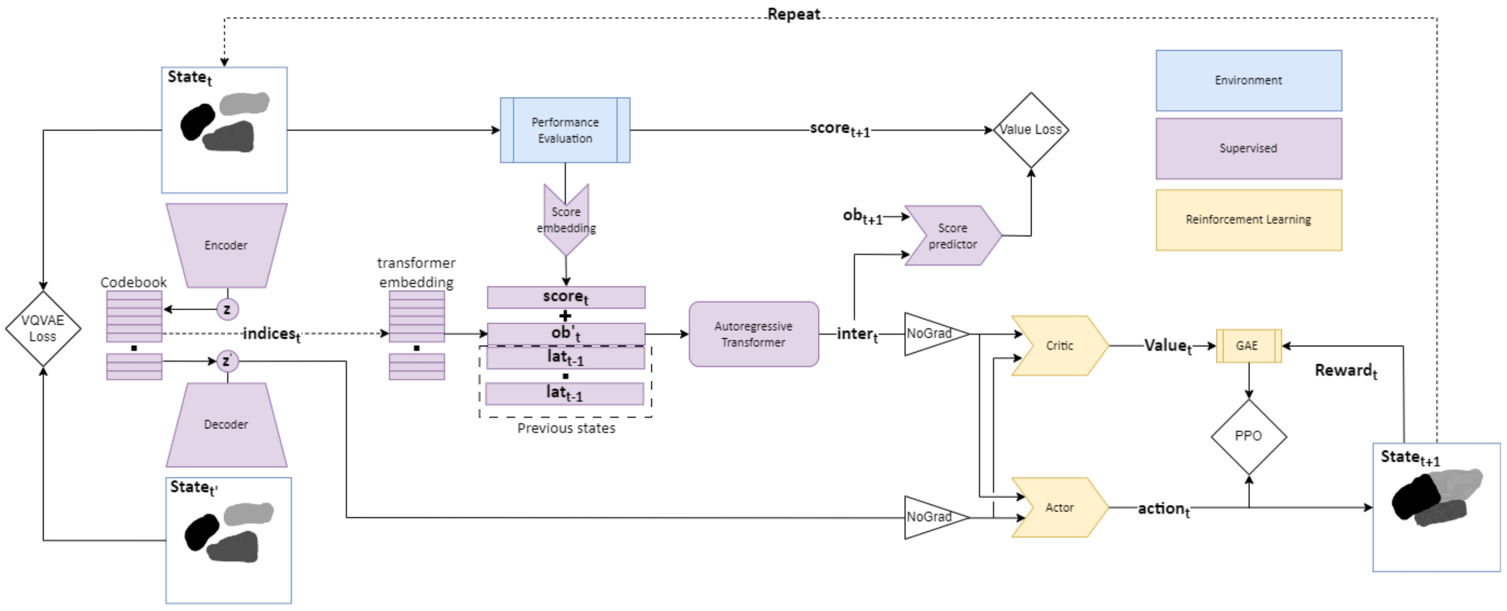


FIG 29: AGENT NETWORK WITH VQ-VAE, TRANSFORMER AND PPO IMPLEMENTATION.

Proximal Policy Optimization (PPO) was chosen as the policy optimisation method for three main reasons: Firstly, PPO is significantly easier to implement compared to DDPG and SAC. Secondly, the reduced sample efficiency of PPO is mitigated through the use of a world model. Thirdly, the improved exploration provided by SAC can be replicated through a more stochastic policy. To implement PPO, three components must be added to the network: an actor, critic, and generalized advantage estimator. Both the actor and the critic will initially consist of three repeated skipped multi-layer perceptron blocks containing three layers with dimensions [32,128,32] projected down to dimensions four and one respectively. Their input will be the transformer output concatenated together with the encoder output.

The actor is tasked with modifying the samples in state s_t such that the scores assigned to them in state s_{t+1} are increased. This modification is done by taking action *a*, which represents adding or removing a circle with normalized position [*x,y*] and normalized radius *r*, there is a fourth variable, boolean *d,* indicating whether the circle is to be added or subtracted, this process can be seen in figure 30. The action can thus be described as a vector of size four containing: [x,y,r,d]. The modification will be carried out by a neural renderer consisting of a decoder with the same general architecture of the observer. The input dimension will however be three, representing the x,y, and r of the action vector size, d will set the value assigned to the mask created by the neural renderer. Through the use of this neural renderer the modification is both differentiable and able to be multi-processed efficiently, decreasing training and inference time.

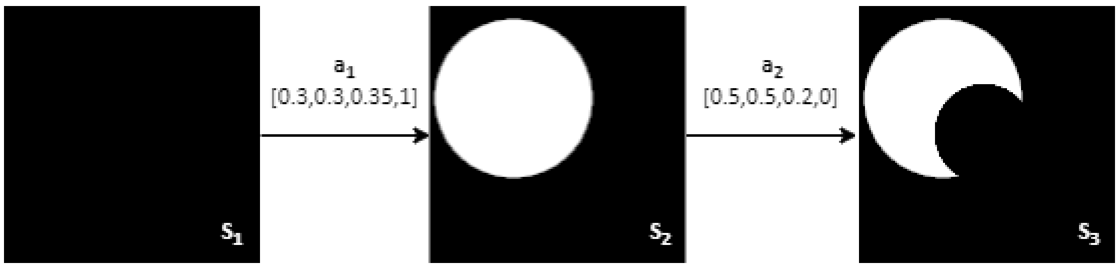


FIG 30: AGENT ACTIONS STARTING FROM EMPTY CANVAS.

6.3.1 Proximal Policy Optimization (PPO)

Since PPO is a policy gradient method, it optimizes not the actions themselves but their probabilities. Therefore, the action outputs are used as the mean of a multivariate normal distribution with hyperparameter variance σ. To obtain the final actions these distributions are sampled, and the log probability of the resulting actions saved. The actor network is then optimized by increasing the log probability of advantageous actions, while minimizing the log probability of disadvantageous actions. This is done by performing several gradient descent steps on the policy ratio multiplied by the advantage. The policy ratio can be calculated by exponentiating the difference between the new log probabilities θ and the old log probabilities θ_{old}. This is mathematically equivalent to dividing the non-log probabilities. To stabilize the training and prevent the policy from changing too much in one step, the policy ratio is clipped by between hyperparameter -ε and ε, initially set to 0.2. Additionally, the optimization is stopped when the difference between the new and old policy becomes greater than hyperparameter α, initially 0.05. Combined, this gives us the agent loss:

$$L_{actor} = -\min(\exp(\theta - \theta_{old})\,\hat{A}_{(s,a)}, clip(\exp(\theta - \theta_{old})\,, 1 - \varepsilon, 1 + \varepsilon)\hat{A}_{(s,a)})$$

Where *s* is the current state, *a* is the action taken and $\hat{A}_{(s,a)}$, is the advantage of action *a* taken in state *s*. Notice the minus sign in front of the equation, this is added because gradient descent is used instead of gradient ascent which is used in the original paper (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017).

Attentive readers may have noticed that $\hat{A}_{(s,a)}$ is not yet known, this is why a critic network and advantage estimator are needed. The critic is trained to predict value V based on the current state, the value is calculated through the summation of the current reward and discounted future reward. The critic is trained to minimize the loss function is:

$$L_{critic} = (V_{pred} - V)^2$$

The advantage can be determined by calculating the difference between the true value and the predicted value in a process named generalized advantage estimation as described in (Schulman, Moritz, Levine, Jordan, & Abbeel, 2015). If the true value is greater than the predicted value, the advantage is positive, implying that the agent performs better than expected. By multiplying the policy gradient with this advantage, the gradients will be pointing towards an increased probability for positive advantage, and towards decreased probability when negative, thereby achieving the desired behaviour.

There is one last term added to the loss function, which in (Lillicrap, et al., 2015) is a bonus score for increased entropy, encouraging exploration. This has however been replaced by a mean squared error loss between the entropy and a target entropy since the unbounded bonus caused the entropy to increase rapidly in testing. These three losses are combined to calculate the complete loss function.

The critic loss and entropy loss are scaled by hyperparameters c_1 and c_2, initially set to 0.5 and 0.001 respectively. The full loss is as follows:

$$L_{agent} = -\min\left(\exp(\theta - \theta_{old})\,\hat{A}_{(s,a)}, clip(\exp(\theta - \theta_{old})\,, 1 - \varepsilon, 1 + \varepsilon)\hat{A}_{(s,a)}\right)$$

$$+c_1\left(V_{pred} - V\right)^2$$

$$+c_2(entropy_{target} - entropy)^2$$

6.3.2 Training procedure

As established, the goal of the training is to teach the agent to manipulate the images in order to optimize for an unknown scoring criterion. To make this compatible with reinforcement learning, a reward function must be defined. To keep things as simple as possible the reward function will simply be reward = $score_t - score_{t-1.}$ Bear in mind that the world model in the form of the observer, interpreter, and score predictor is simultaneously fine-tuned in a supervised manner to better represent the datapoints generated by new episodes. To increase the speed this world model training is executed on a separate GPU, its network parameters are copied to the agent after every completed PPO cycle. This also implies that the PPO algorithm only adjusts the parameters of the relatively simple actor and critic networks.

Considering training time is limited, the process will use multiple workers to sample multiple environments in parallel. Through this method, the training procedure will be stabilized by using the mean gradients between multiple environments. Additionally, this will increase VRAM utilization on the GPU increasing efficiency, thus hopefully reducing overall training time. Each environment will give a similarity score using the intersection over union (IoU) compared to a reference image, this reference will be different for the environments and sampled randomly from the MNIST and later Swiss Dwelling datasets.

Comparing samples with a random image as the criterium may seem strange, there is however an idea behind this. If one is to optimize a design for a criterium such as daylighting, the optimal design will be some seemingly arbitrary shape. Since shapes that are similar to this optimal shape will likely also have high scores, solving the problem becomes somewhat comparable to increasing the IoU between the current sample and this unknown optimal shape. Therefore, a model trained on solving this IoU based problem will likely also do well on other criteriums. The benefit being that the IoU is very fast to compute, reducing training time. This is especially useful considering the final network architecture is not yet known, and many experiments are likely required. On a final version it is of course possible to train the network on a more problem specific criterium. In figure 31, various randomly selected samples created by a trained agent can be seen. The network parameters used in previous chapters were not changed. The Adam optimizer with a learning rate of 1e-4 which seemed to work reasonably well. The original intent was to further tweak the (hyper) parameters to improve performance. In the end a system overhaul was done instead, based on new insights gained from these initial experiments.

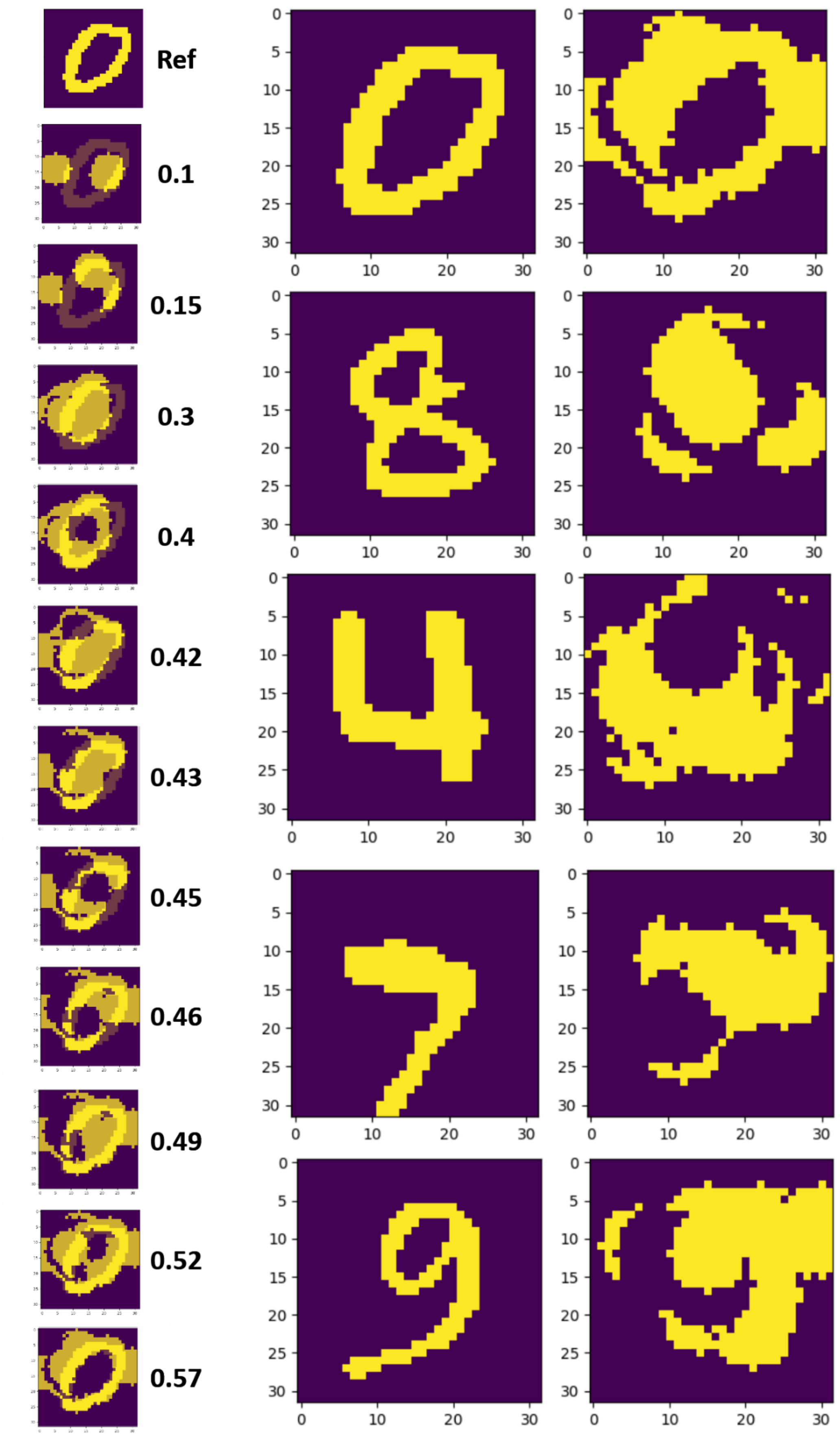


FIG 31: LEFT: SAMPLE THROUGHOUT TIMESTEPS WITH IOU SCORES, RIGHT: VARIOUS SAMPLES CREATED BY AGENT.

6.4 Communication

The concept for dealing with multiple channels is to encode the different channels independently, and then use self-attention to share information between the different embeddings. This method is much simpler than the communication protocol previously envisioned and seems viable since the transformer-based architecture turned out to be much faster to train than initially estimated. A diagram of how this would work can be seen in figure 32. Developing a training procedure is significantly more challenging than the single channel version. The main reason for this is the inhomogeneous nature of the data, different compositions have a different number of channels, which makes parallelization very difficult. Additionally, the fitness score is more complicated to compute for this case slowing down training significantly. Due to these reasons the performance of this version is not yet satisfactory and will not be discussed further.

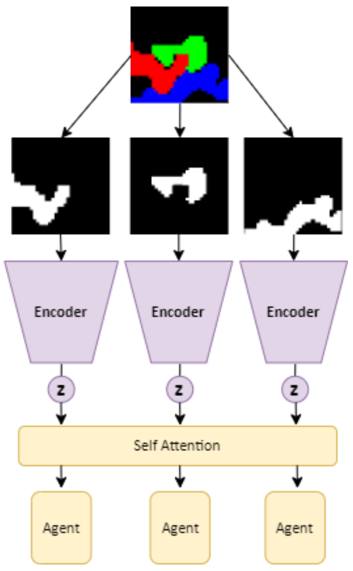


FIG 32: COMMUNICATION PROTOCOL FOR MULTIPLE CHANNEL OPTIMIZATION.

7. Insights, alterations, and tuning.

Although the results from the previous chapter look promising, they are not yet satisfactory. The development of the model has led to several insights that call into question the suitability of the current interaction between agent and environment. By reflecting upon these questions, the design may be improved. In the following chapter, these questions will be discussed, and alterations proposed.

7.1 Interpreter performance.

The rapid decrease of the score prediction error as seen in figure 33, demonstrates the ability of the VQ-VAE/transformer-based model to effectively decode the relationship between shapes and their performance based on relatively few examples. However, the fact that it takes approximately 25-50 samples to perform optimally, implies that the first few actions are ill informed. This may be alleviated by padding the memory of the interpreter with samples obtained through a different method. One option is to generate random samples using the observer decoder, these could however be further improved by employing a genetic algorithm in the latent space.

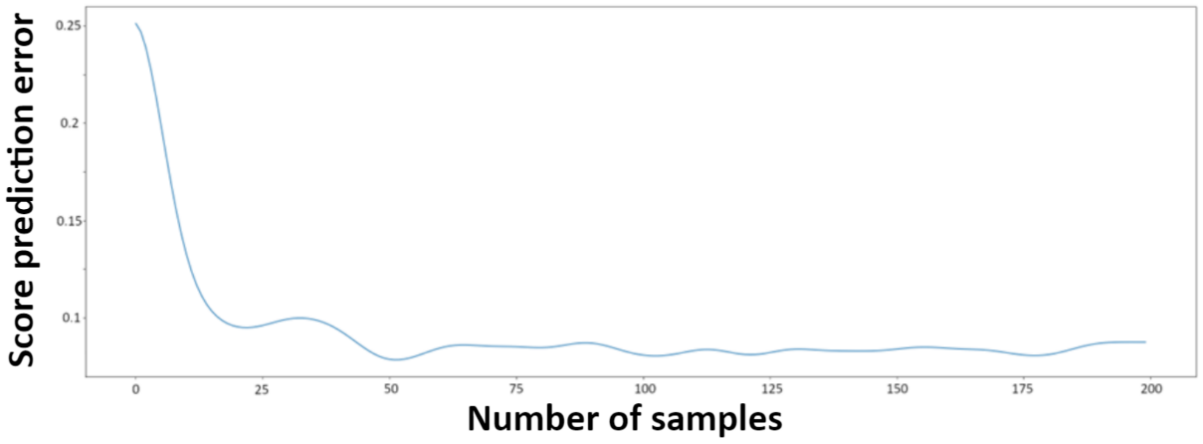


FIG 33: VQ-VAE MEAN PREDICTION ERROR REDUCING AS A FUNCTION OF SAMPLE COUNT.

7.1.1 Genetic algorithm.

Although the use of a genetic algorithm was previously dismissed in chapter 2 due to its reliance on an autoencoder that fully represents the solution space, it may yet prove useful. The data scarcity preventing this representation can now be circumvented by training the decoder based on samples generated by the agent. In fact, since this is currently how the observer is trained, the decoder can simply be taken from the existing VQ-VAE which is currently being used for training its encoder. Another option is to use the neural renderer which generates simpler samples and only takes three parameters. Which one of these to use depends on the exact application.

The genetic algorithm will be loosely based on NSGA-II (Deb, Pratap, & Agarwal, 2002). It will be implemented using PyTorch such that it can be executed on the same device as the rest of the algorithm. The algorithm consists of three phases: selection, crossover, and mutation. A diagrammatic overview of how these are implemented can be found in figure 34. One notable aspect is that the non-dominated selection is only done when there is more than one objective, otherwise the solutions are sorted by score.

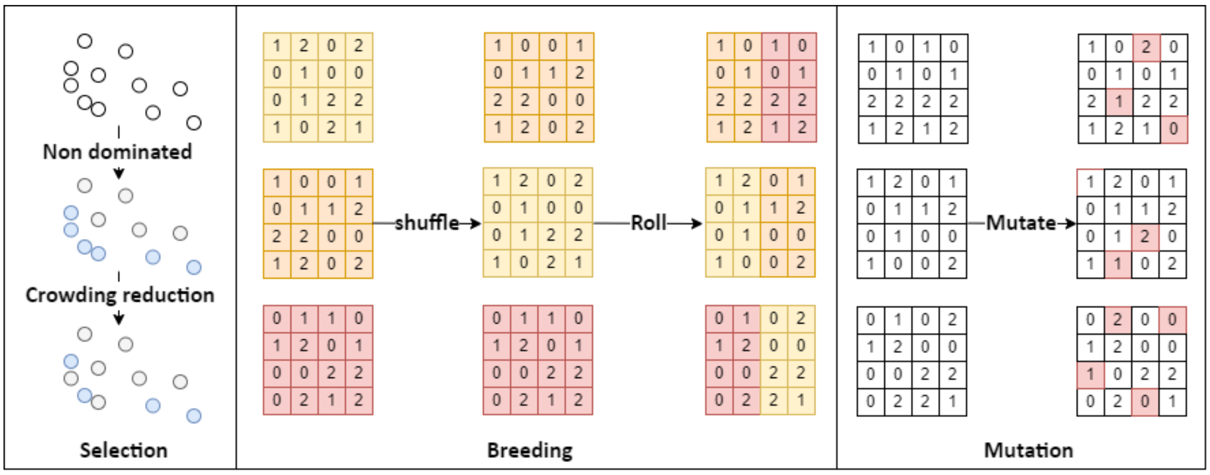


FIG 34: GENETIC ALGORITHM OVERVIEW.

Although the intended application for the genetic algorithm was to generate padding samples for the interpreter memory, the methodology performs surprising well even for generating results directly. This is showcased in figure 35, where optimization results based on two criteria: compactness (polsby popper index), and daylight satisfaction can be seen. This level of performance is only possible because the VQ-VAE can generate a wide range of spatially coherent samples. To capitalize on this surprising performance, it will be used as a placeholder for the multiple channel case until the transformer outperforms it.

In the pareto front visible in figure 35, a significant number of samples have a score of almost zero. This is the result of multiplying both compactness and daylight satisfaction with the area score, to guarantee fixed area. This is quite inefficient since compared to the area score, the daylight score is quite expensive to compute. Therefore, it may be helpful to introduce a cutoff value for the area score and not consider any unsatisfactory samples for further analysis.

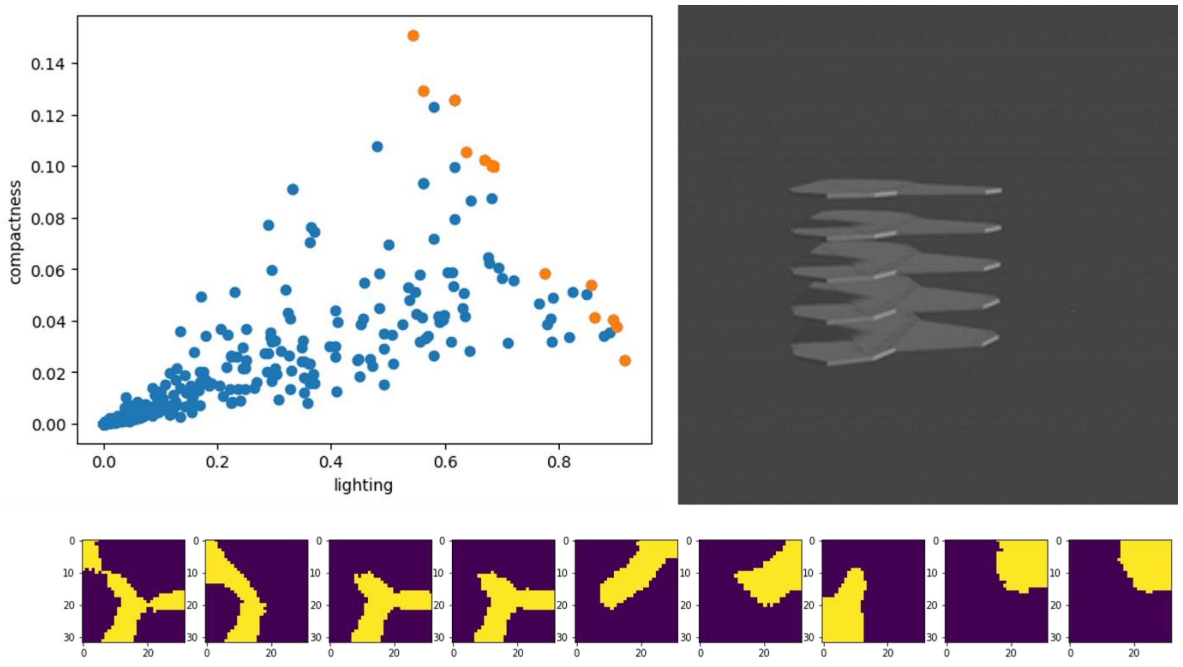


FIG 35: RESULTS OF COMPACTNESS VS DAYLIGHT SATISFACTION OPTIMIZATION WITH FIXED AREA. TOP LEFT: PARETO FRONT, TOP RIGHT: 3D REPRESENTATION OF THIRD SAMPLE, BOTTOM: SAMPLES ASSOCIATED WITH PARETO FRONT, INCREASING COMPACTNESS TO THE RIGHT.

7.2 Critic bottleneck.

The performance of the agent seems to be bottlenecked by the accuracy of the critic network as seen in figure 36. The reason for this is most likely a combination of an increasing diversity of actions taken by the actor as it learns, and a large variety of reward trajectories caused by ever changing reference images. Perhaps an important question must be asked, is the critic necessary in this setting?

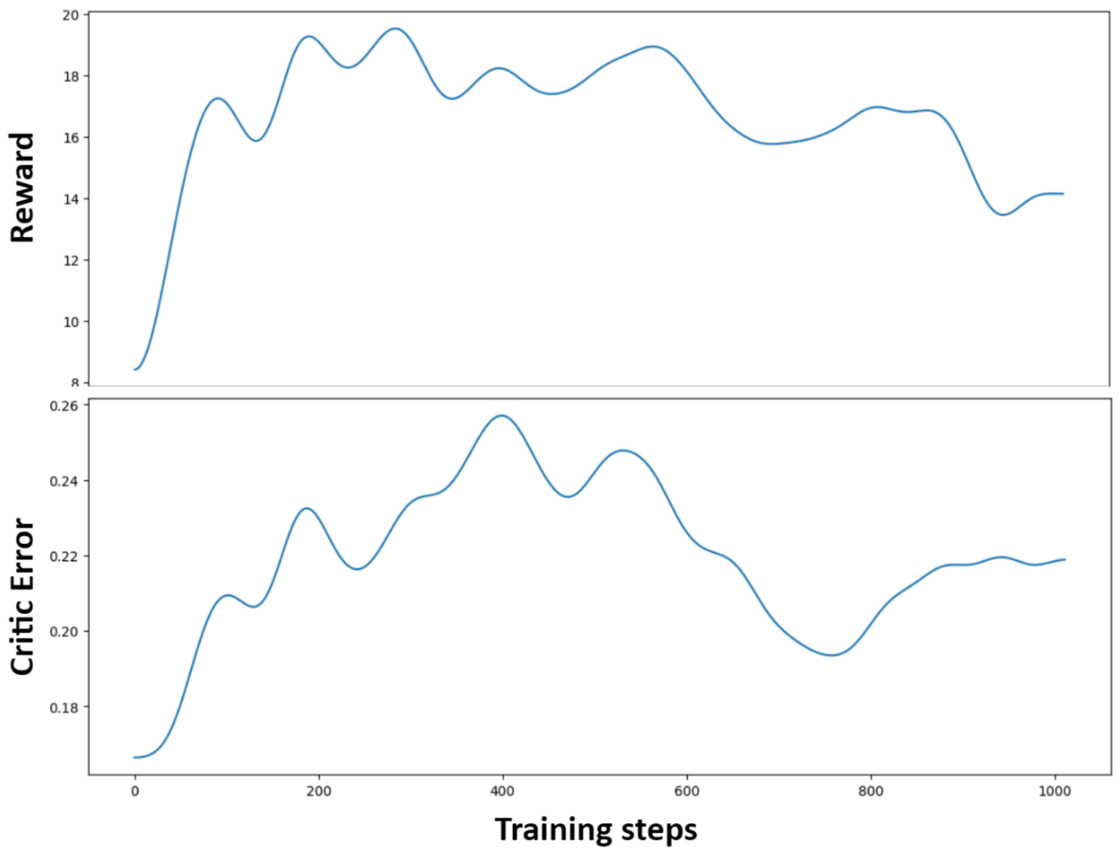


FIG 36: ACTOR REWARD AND CRITIC ERROR DURING TRAINING.

The goal of the critic is to predict the current and discounted future rewards that result from being in a certain state. This is done such that the actor is also criticized for future negative effects associated with taking certain actions. However, in this setting the effects of actions do not change throughout an episode. Additionally, all possible future states remain available regardless of what action is chosen since actions can be undone without consequence. Therefore, an argument can be made that the potential negative effect of actions on future rewards is negligible, and that the critic can thus be omitted entirely. The advantage of an action would then simply be replaced by the current reward, advantage = $\text{score}_t - \text{score}_{t-1}$. This could be described as a special case of reinforcement learning where the discount factor for the future rewards is zero.

The removal of the time dependency enables another significant change to the network. Currently only the last vector produced by the interpreter, the one representing the present, is used to inform an action. However, with the time dependence being eliminated it becomes possible to act on all states both “past” and “present” since these become meaningless without time dependency. This also allows the triangular mask to be removed allowing full information flow between all samples. The new network architecture with these changes implemented can be seen in figure 37. Notice that the state does not contain a single sample anymore, it now contains a variable number of samples that are manipulated simultaneously. Since the samples provide context to each other, the memory can also be removed, significantly reducing the model complexity. Additionally, since the transformer must no longer inform both the actor and critic, its training stability is increased. The increased stability allows the transformer and its score embedding to be trained through PPO without the score prediction as a surrogate objective. This more direct approach will likely increase model performance.

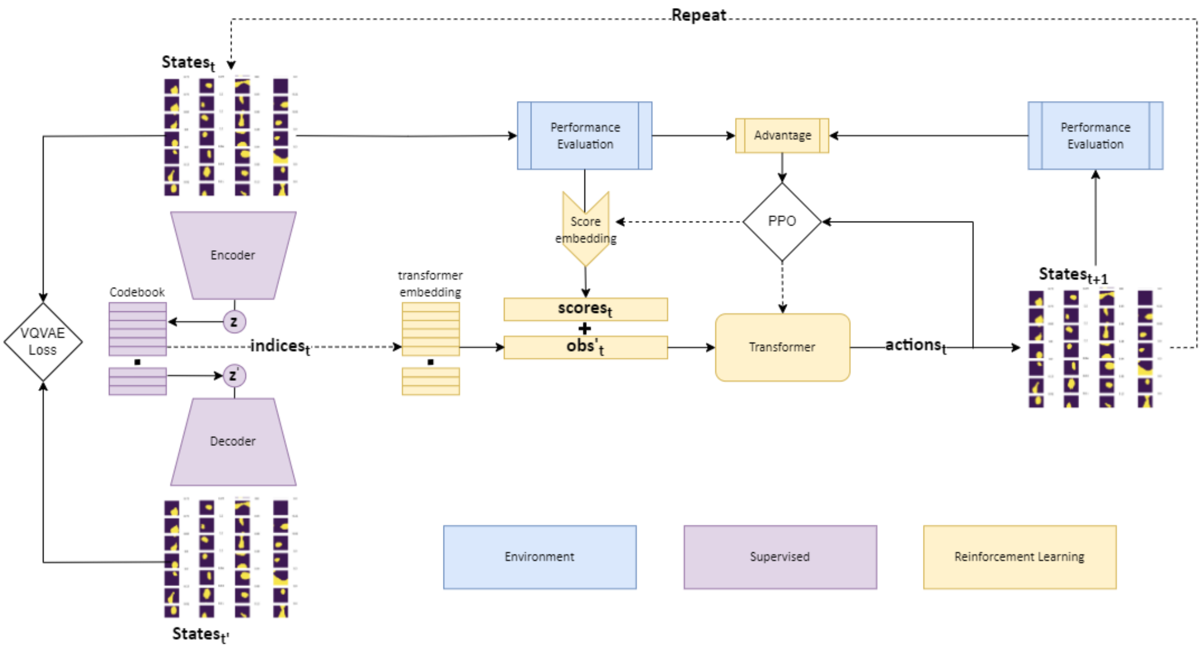


FIG 37: MODIFIED NETWORK WITHOUT CRITIC AND PARALLEL ACTIONS.

7.2.1 Network training alterations.

With this new network architecture, the training algorithm must also be adjusted. For starters with the critic removed, the loss function becomes:

$$L_{agent} = -\min\left(\exp(\theta - \theta_{old})\,\hat{A}_{(s,a)}, clip(\exp(\theta - \theta_{old}), 1 - \varepsilon, 1 + \varepsilon)\hat{A}_{(s,a)}\right)$$
$$+c_1(entropy_{target} - entropy)^2$$

Although there are still multiple environments being sampled simultaneously, each environment now also produces multiple actions. This can simply be flattened to form a batch of size num_environments*samples_per_environment. To scale training either the number of environments, or the number of samples within one environment can be increased. The difference in scaling strategy is important because, even though both methods generate the same amount of training data, the transformer execution time scales quadratically with the number of samples, running more environments on the other hand increases the execution time linearly. Thus, scaling the number of samples is less efficient comparatively especially because the performance of the model does not seem to increase when more than fifty samples are used as seen in figure 33. Considering the model requires about 100 megabytes of VRAM per environment when using 64 samples, it is easy to see how this method can generate large batch sizes. On an A40 GPU this would allow for 480 parallel environments, bringing the theoretical maximum batch size for the PPO to 30720. This will likely help improve training stability further since transformers tend to perform well with large batch sizes. GPT-3 for example uses batch sizes ranging from 0.5 million to 3.2 million, depending on the model size (Brown, et al., 2022).

Another change that can be made is related to the input samples for each training step. One could imagine that during the first step of generating a solution the samples have a high variance since they are randomly generated. After a few steps of refinement however, the concurrent samples will become more like the optimum design but also each other. During training it is possible to create this effect by generating entire episode, this may however be inefficient. The reason being that the system will not be exposed to relatively good samples during early training since it first must learn to create these. An alternative is to start from the target image and add an increasing degree of distortion inspired by diffusion models (Ho, Jain, & Abbeel, 2020). A major difference to how this is implemented here is that the noise is added to the latent space instead of to the pixels directly, this will increase the "resolution" of the distortions which is more suitable for this problem.

7.3 Prioritizing positive advantage.

Another trick that seemed to have significantly improved performance is to prioritize positive advantages. What this means is that actions that lead to improved scores will be encouraged a lot while actions leading to reduced scores are only punished a little bit. This is achieved by applying a Leaky ReLU to the advantages, the effect that this has on model performance is quite significant as can be seen in figure 38. The best performance was found with a leaky ReLU slope of 0.2-0.25.

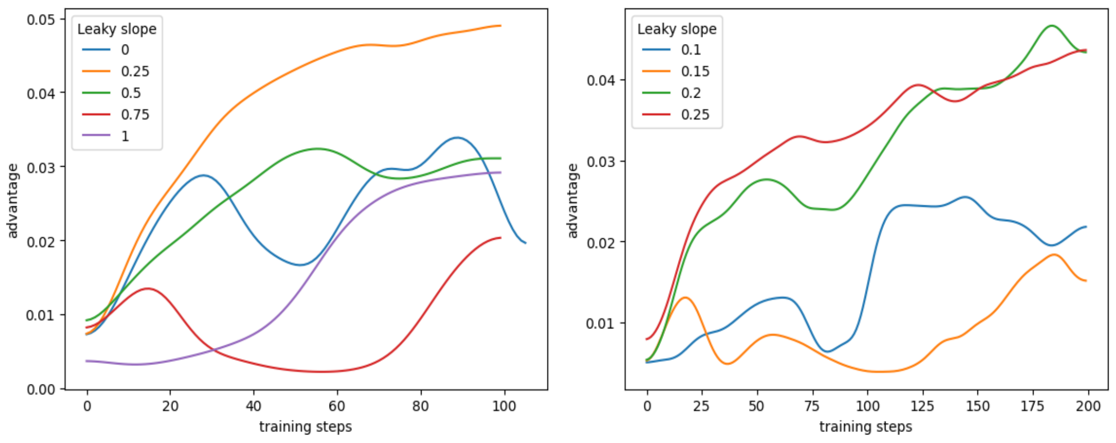


FIG 38: AVERAGE IMPROVEMENT WITHOUT AND WITH LEAKY RELU ON ADVANTAGE.

7.4 Stabilizing effect of variance *σ*.

Since PPO samples actions from a normal distribution, hyperparameter *σ* is introduced to control the variance of this distribution. This hyperparameter has a significant effect on the stability of the training. The reason for this is most likely that if *σ* is set high enough, all actions are possible even under policies that exhibit almost no variation. Combined with the prioritization of positive advantages this allows the model to converge to advantageous actions even if they are very unlikely under the current policy. The effect this has on the training can be seen in figure 39. The best setting was found to be 0.15, this seemed to provide a nice balance between exploration and exploitation ability.

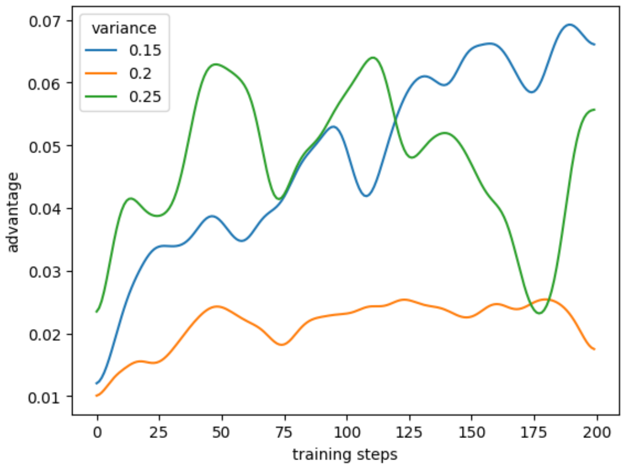


FIG 39: AVERAGE IMPROVEMENT WITH DIFFERENT SETTINGS FOR THE VARIANCE.

7.5 Policy collapse, and non-stationary Adam.

During training policy collapse was quite a common occurrence. The exact cause of this is unknown, however it became more common with increasing network size. A potential relief for this phenomenon is to use equals β2 in the hyperparameters for the Adam optimiser, this is referred to as non-stationary Adam (Shibhansh, Qingfeng, & Rupam, 2023). These parameters control the speed with which the running averages for the first and second moments of the gradients are updated. Another alternative is to switch the Adam optimiser for stochastic gradient descent (SGD) since this does not use the same momentum-based technique. Although both methods effectively reduced the frequency of policy collapse the use of SGD significantly slowed down convergence while the modified Adam did not. Hyperparameters β1 = 0.95 and β2 = 0.95 seemed to work best as can be seen in figure 40. Pay attention to is the spread in the network performance after 100 training steps. Although the training for β1 = 0.95 and β2 = 0.95 is noticeably more stable there is still some variation between different training runs.

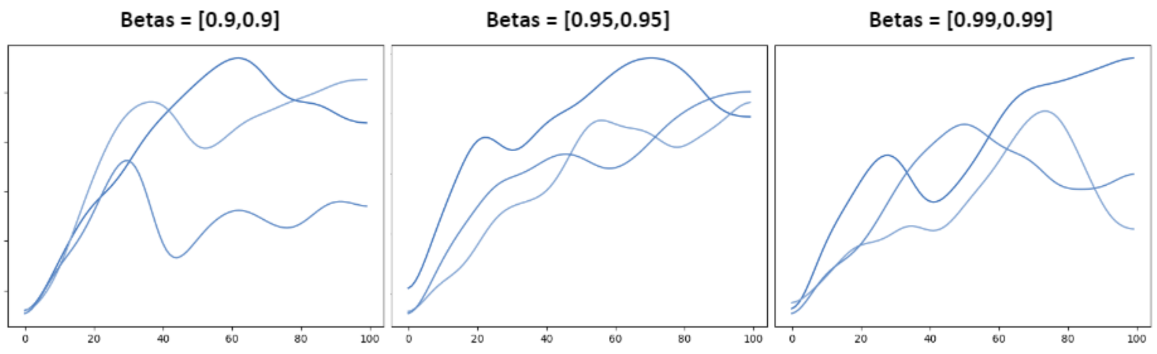


FIG 40: THREE DIFFERENT TRAINING SESSIONS FOR IDENTICAL NETWORKS UNDER THREE DIFFERENT BETAS.

8. Results.

In the previous chapters, the network has been designed, developed, and tuned. Here the resulting network will be discussed. The training parameters are listed in figure 41. Note that the sizes of the different networks are very modest, this was done to speed up experimentation the whole system could be trained in approximately 15 minutes on a RTX 4090 GPU. When improved performance is desirable, the networks could be scaled up drastically. The performance improvements would be significant especially for the VQVAE since the current compression quite lossy as seen in figure 42. This limits the ability of the network to observe small differences between states.

VQVAE-Dims	[1,32,16,2]
VQVAE-Depths	[1,1,1,1]
Codebook size	16
Transformer dim	32
Transformer layers	4
Actor MLP layers	1
PPO clip	0.15
Poolsize	64
Batch size	256
Leniency	0.2
Variance of action distribution	0.15
Betas	[0.95,0.95]
Lr	1e-3

FIG 41: TRAINING SETTINGS FOR EVALUATED VERSION.

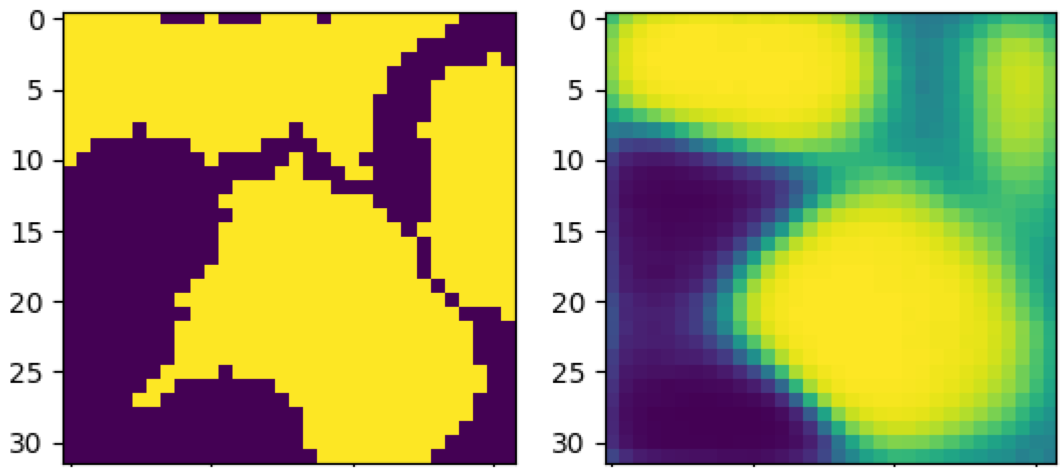


FIG 42: VQ-VAE SAMPLE AND RECONSTRUCTION.

8.1 Network operation.

To understand how the trained network operates, the steps undertaken within one cycle will be discussed. The whole procedure will be displayed visually in the following figures, additional text explanations will be added to the figure description.

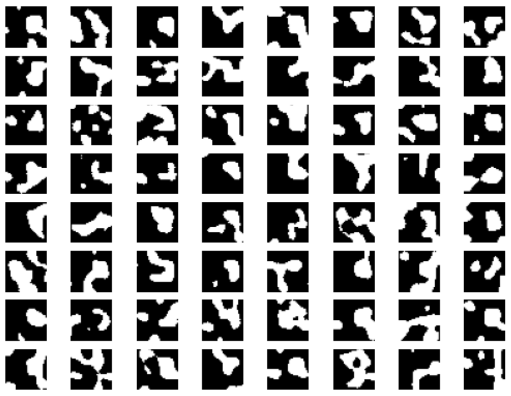


FIG 43: STEP 1: 64 CONCURRENT SAMPLES ARE RANDOMLY GENERATED IN THE VQ-VAE, EACH SAMPLE IS REPRESENTED AS A SEQUENCE OF SIXTEEN DISCRETE VARIABLES. FOR DEMONSTRATION PURPOSES THE OPERATIONS WILL BE DISPLAYED ON THE SAMPLES DIRECTLY, IN REALITY THESE OCCUR IN LATENT SPACE. SAMPLES ARE ONLY GENERATED RANDOMLY IN THE FIRST ITERATION, IN SUBSEQUENT STEPS THE LATENTS WILL BE GENERATED BY PRESENTING THE VQ-VAE ENCODER WITH THE RESULTING SAMPLES FROM THE PREVIOUS ITERATION.

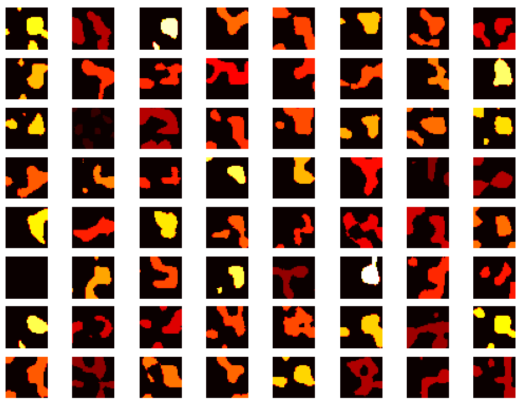


FIG 44: STEP 2: TO THE SAMPLES IS ADDED A SCORE EMBEDDING REPRESENTING THE FITNESS OF THE SAMPLES, IN THIS CASE THE FITNESS IS DETERMINED THROUGH CALCULATING THE IOU BETWEEN THE SAMPLE AND AN ARBITRARY REFERENCE SHAPE. WHITER SAMPLES HAVE A HIGHER SCORE. AS AN EXERCISE TRY TO IMAGINE WHAT THE REFERENCE SHAPE LOOKS LIKE BY LOOKING AT WHICH SAMPLES HAVE HIGHER SCORES. THIS IS COMPARABLE TO HOW THE NETWORK DETERMINES ITS NEXT MOVE.

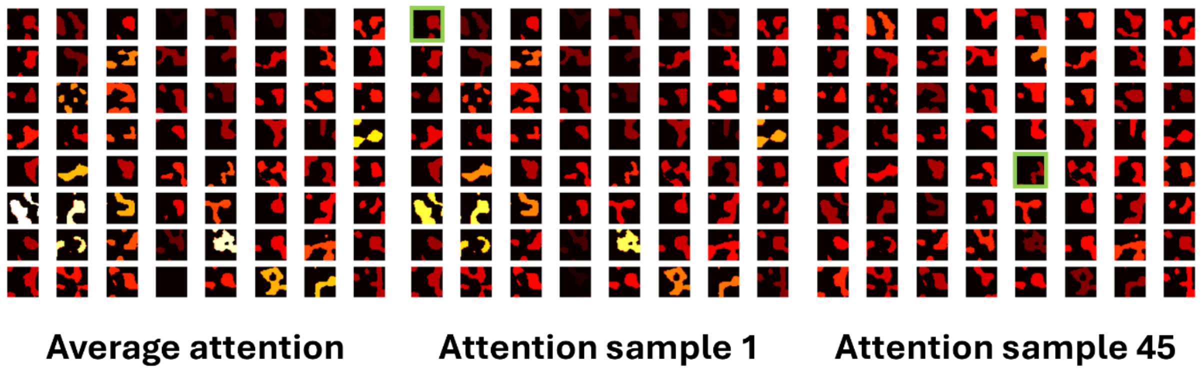


FIG 45: STEP 3: THE SAMPLES WITH THEIR SCORE EMBEDDINGS ARE FED TO THE TRANSFORMER WHICH COMPARES THE SAMPLES TO DETERMINE WHICH ACTIONS SHOULD BE TAKEN. IN THE FIGURE THE ATTENTION MAPS CAN BE SEEN, NOTICE THAT THE ATTENTION MAPS FOR SAMPLE 1 AND 45 ARE DRASTICALLY DIFFERENT. THIS INDICATES THAT DIFFERENT ACTIONS

WITHIN ONE SAMPLE POOL ARE INFORMED BY DIFFERENT INFORMATION EXTRACTED FROM THE SAMPLES. COMPARE THE AVERAGE ATTENTION TO THE SCORES, WHICH SAMPLES GAIN MORE ATTENTION?

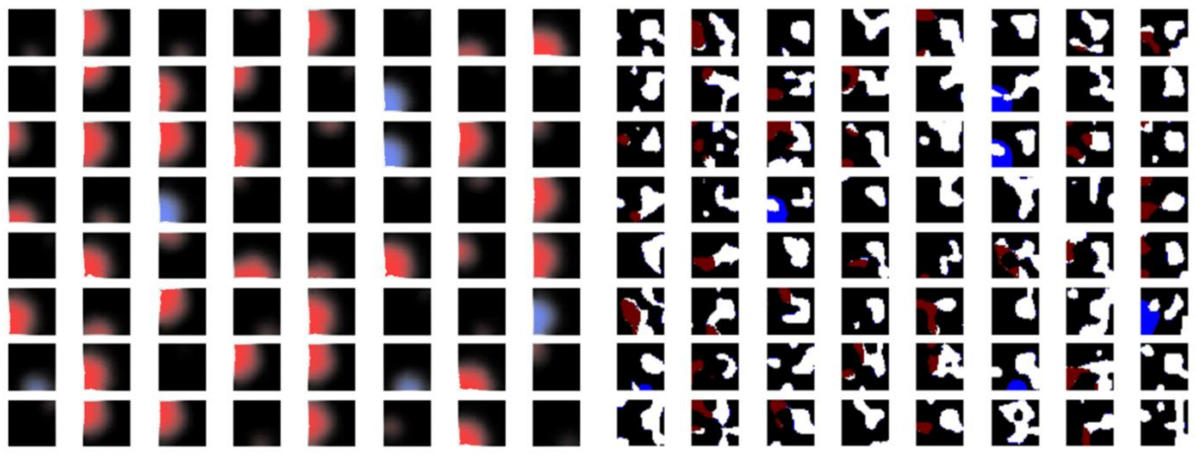


FIG 46: STEP 4: BASED ON THE INFORMATION EXTRACTED FROM THE SAMPLES ALTERATIONS ARE MADE, ON THE LEFT THE PROPOSED CHANGES CAN BE SEEN, RED FOR ERASE AND BLUE FOR PAINT. ON THE RIGHT THE RESULTS OF THESE ALTERATIONS CAN BE SEEN.

8.2 Comparison to evolutionary algorithm.

Now that that a functional solver has been developed, it can be compared to the previously discussed genetic algorithm. The method of comparison is as follows: 256 pools of 64 samples each will be generated by randomly sampling the VQ-VAE decoder. These samples will then be assigned a score based on their similarity to a pool specific reference. For the transformer-based solution these scores will be fed into the score embedding module, while for the genetic algorithm they will represent the fitness. Both solvers will then make ten alterations to improve the samples. The average improvement is then computed. The performance of the two methods is not vastly different, 13 and 9 percent for the transformer and genetic algorithm respectively. However, there are some relevant comments to be made. First, the genetic algorithm can generate these samples only through the VQ-VAE that was trained by the transformer. Second, the transformer cannot yet reach the same performance on multi-channel samples due to time constraints, while the genetic algorithm can. Third, both algorithms are limited in performance due to the lossy compression of the relatively small VQ-VAE.

9. Application.

Now that the analysis methods, the genetic and the transformer-based solvers have been established, they can be applied to our design problem. Although the samples considered so far have only had one channel, this can be amended by stacking multiple images together and performing some light post processing. In the case of floorplans, the images are segmentation maps. In these maps, the different channels may not overlap, otherwise two rooms could be overlapping, which is not allowed. The procedure for processing layers into non overlapping segmentations will be outlined next.

9.1 Sample processing.

What will follow is a step-by-step walkthrough of the image postprocessing. Each step is explained visually with further explanation provided in the figure description. With the genetic algorithm it was found that the last step can be skipped when the neural renderer is used instead of the VQ-VAE decoder, since this inherently leads to more coherent shapes. One limitation of doing this is that the achievable complexity of the shapes is reduced. This is not necessarily a downside in the case of floorplans since this tends to lead to simpler room layouts. For the generation of the building footprint this does not allow for a satisfactory degree of variation limiting the maximum achievable performance.

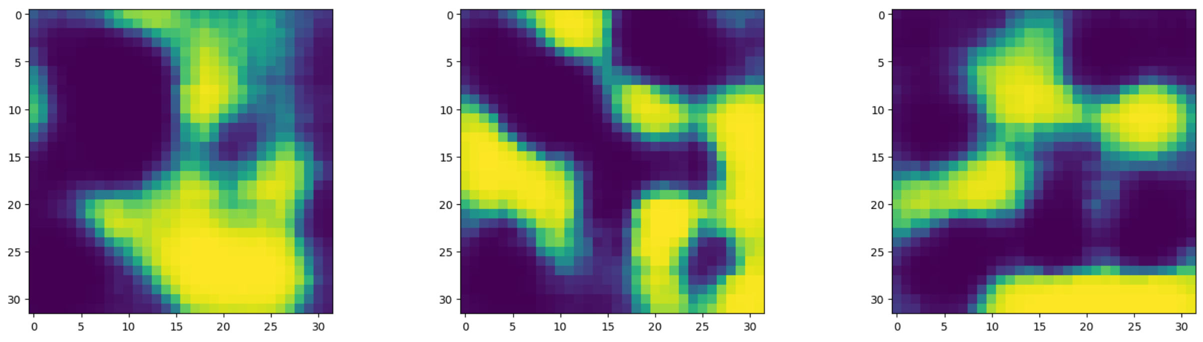


FIG 47: ONE IMAGE PER ROOM IS CREATED; THESE GIVE AN INDICATION OF WHERE EACH ROOM WOULD IDEALLY BE.

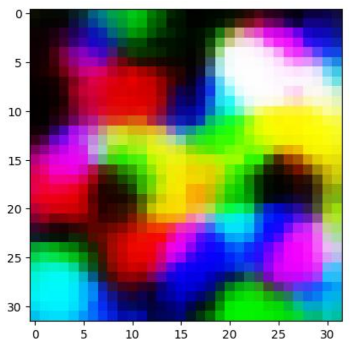


FIG 48: THE IMAGES ARE CONCATENATED TOGETHER, IN THIS CASE CREATING ONE IMAGE OF DIMENSIONS [32,32,3].

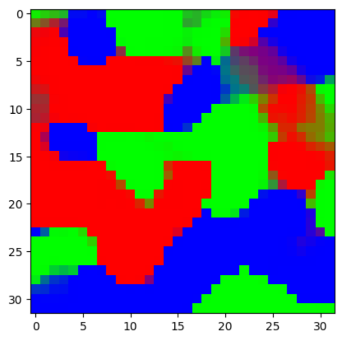


FIG 49: THE VALUES ARE MULTIPLIED BY A FACTOR 100 BEFORE A SOFTMAX FILTER IS APPLIED, THIS HAS THE EFFECT OF AGGRESSIVELY PRIORITIZING ONLY ONE CHANNEL PER PIXEL.

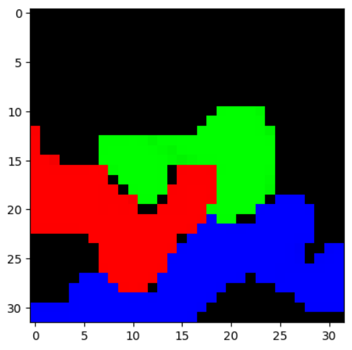


FIG 50: THE LARGEST AREAS ARE KEPT AND THE REST DISCARDED.

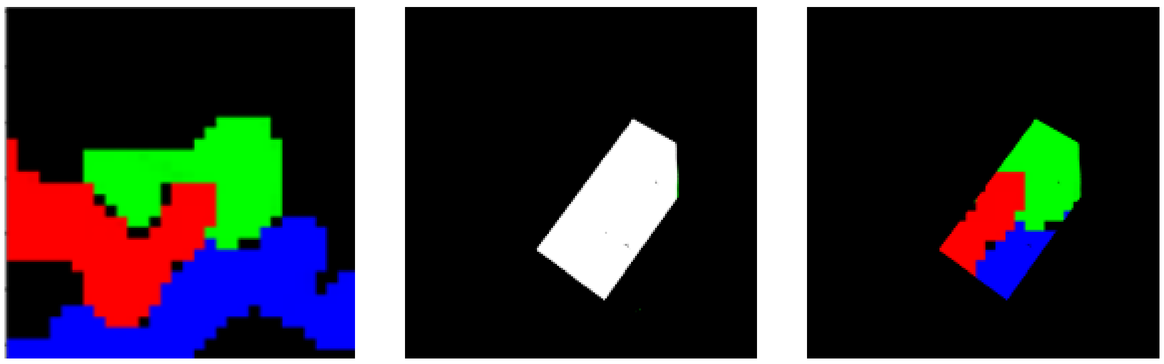


FIG 51: OPTIONALLY A MASK CAN BE APPLIED TO CONDITION THE RESULT ON A GIVEN SHAPE. THIS MASK CAN OFCOURSE ALSO BE GENERATED AUTOMATICALLY IF THIS IS DESIRABLE.

Once these samples are processed the criteria outlined in chapter 2 can be used to grade, rank, and inform the drawings. Once satisfactory performance has been achieved the results may be displayed to the user through a user interface.

9.2 Multi-story building.

To construct a multi-story building, the same methodology of generating multiple samples can be used, in this case multiple samples represent different levels. Since the light levels on one floorplan is typically not impacted by the walls of a floorplan on another level, the segmentation can be performed after the massing. This does however not account for the fact that some building massings are more receptive to design appropriate segmentations than others. Due to the high speed of the current workflow, it is possible to assign scores to massings based on the optimized wall placement within these massings. This would however still lead to an increase in computing time which is not desirable. An alternative approach would be to generate pseudo random samples and assign scores both with and without walls. These samples could then be used to train another model to predict the score difference between these two scores, conditioned on the massing. For now, this aspect is neglected and the scores for the massing are assigned without accounting for walls.

9.2.1 Post processing geometry.

To better communicate the design variations, it would be ideal to construct 3D reconstructions of the buildings as seen in figure 52. A model like this can be constructed using a simple parametric model informed by the floorplan samples. In this case Blender was used, which is very suitable for this application for various reasons. Firstly, it has a node-based geometry modelling system similar to grasshopper, which is convenient. Secondly, it can be used from python without the use of the interface since it is available as a python package. Thirdly, it is open source and python based, thus a future version of the interface developed in this work can be built on top of it with relative ease.

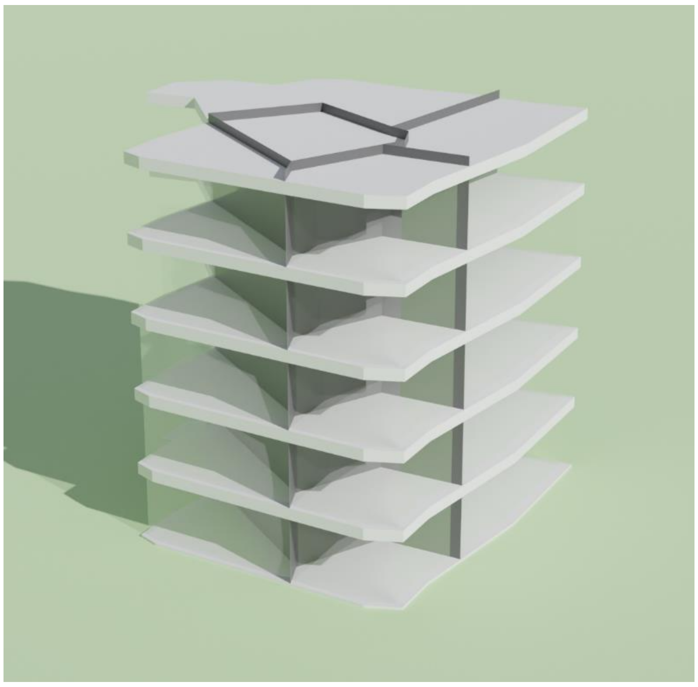


FIG 52: SIMPLE 3D MODEL AS GENERATED FROM SAMPLE DRAWN USING BLENDER.

Additional geometric information may be assigned to the various room types, for example: a bathroom will typically only have windows above eye level, while a living space may have a window across the entire height of the façade. Combined with daylight requirements the approximate composition of the façade can be determined. An example of how this could look can be seen in figure 53, where a bathroom and living room façade can be seen for two different window-to-wall rations.

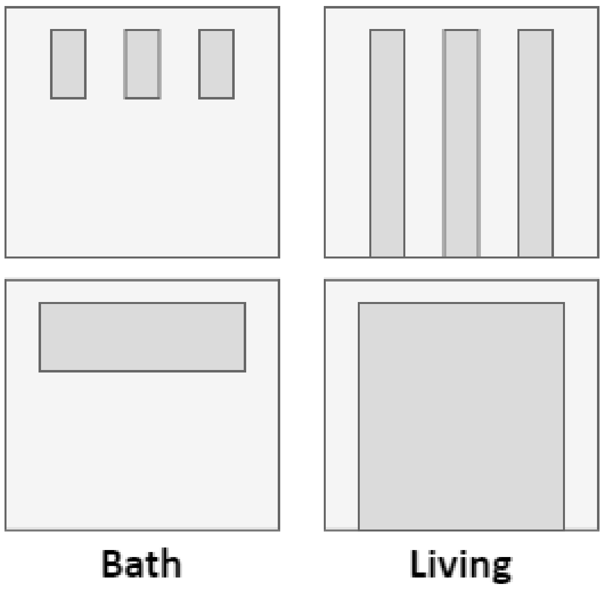


FIG 53: FAÇADE COMPOSITIONS FOR DIFFERENT FUNCTIONS AND WINDOW-TO-WALL RATIOS.

9.3 Interface.

All that remains is to allow the user to use the tool in an intuitive way, this is achieved through the implementation of a graphical user interface using Gradio. The interface should ideally, allow the communication of all relevant data such as climate conditions, context, and relevant building requirements. So far it is possible to mask the general layout and specific rooms conditioned on areas and connectivity matrices as seen in figure 54. The interface will also provide more insight into the underlying criteria such as the daylighting and embodied carbon. This system is directly linked to a geometry processing workflow that converts the drawings to three dimensional representations.

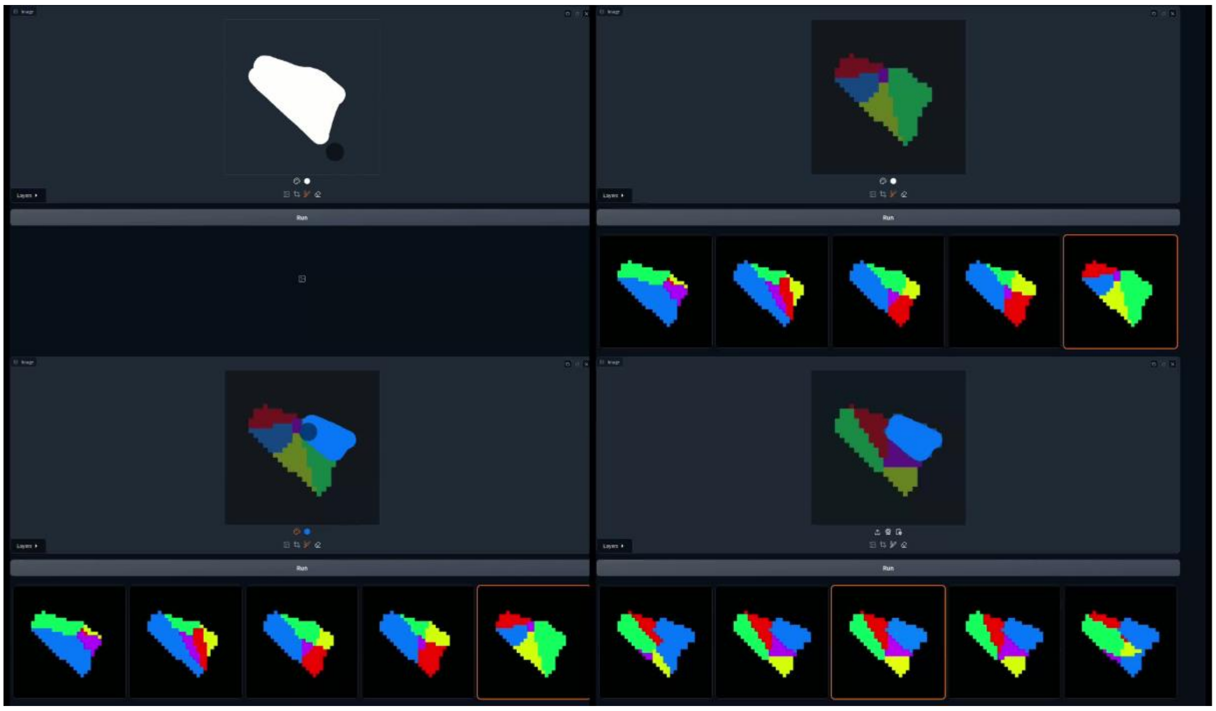


FIG 54: TOP LEFT: MASK FOR THE BUILDING, TOP RIGHT: GENERATED SAMPLES, BOTTOM LEFT: MASK FOR SPECIFIC ROOM, BOTTOM RIGHT: GENERATED SAMPLES CONDITIONED ON SPECIFIC ROOM MASK.

For designs with many floors or a complex shape it may be beneficial the generate the masks per floor based on a 3d model. A workflow to do this has been developed using Blender3d and can be seen in figure 55. A mesh-based approach was chosen for its simplicity, flexibility and ease of use. While modelling, the algorithm continuously creates slices from the building volume representing floor slabs. These can then be assigned zoning and coupled to the deep learning framework which will segment the floorplans and provide insight into the technical performance. Additionally, it may also be used to estimate more financially relevant information such as the estimated maximum number of apartments that can be realized within the provided shape. It may also be interesting to implement a mechanism through which the deep learning framework may modify a 3d model like this. Currently, the 3d model can either be constructed by the agent or the user but not simultaneously.

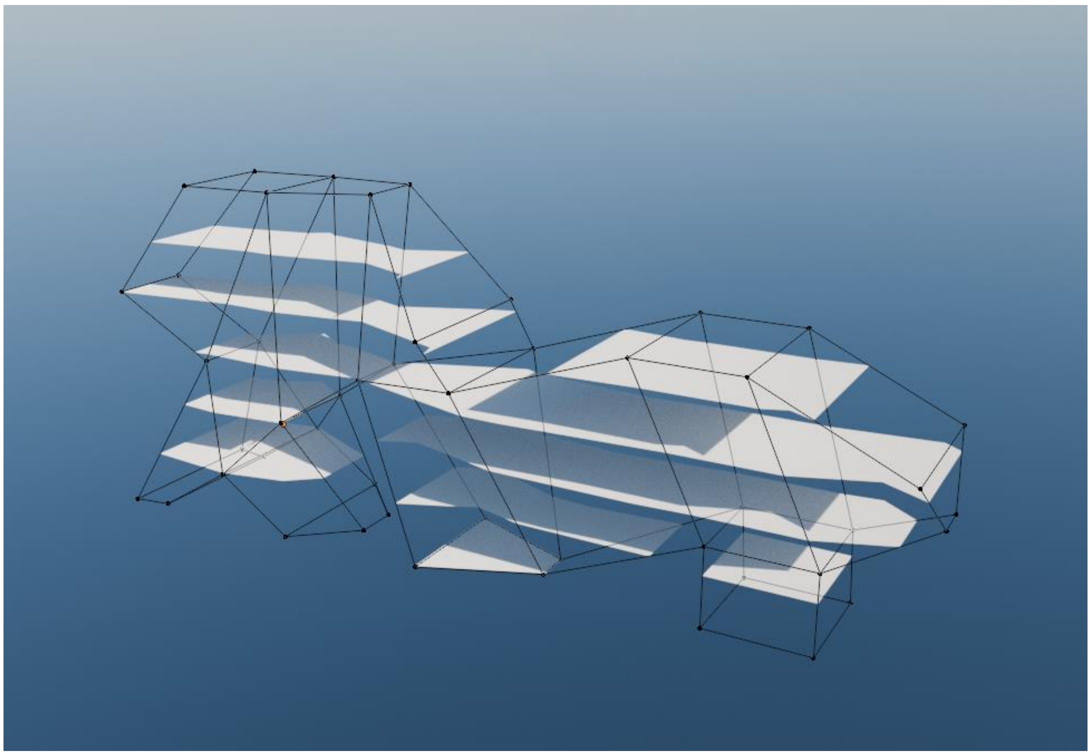


FIG 55: FLOOR SLAB MASKS AS GENERATED FROM AN ARBITRARY MESH DRAWN IN BLENDER3D.

9.4 Case study.

As a test of the workflow, a single-family home will be designed based on a realistic brief and location. The location will contain arbitrary buildings and vegetation as well as some sloped terrain to cast more complex shadows onto the lot. The climate will be that of Amsterdam, since this data has already been processed during the development of the lighting algorithm. A 3D view can be seen in figure 56.

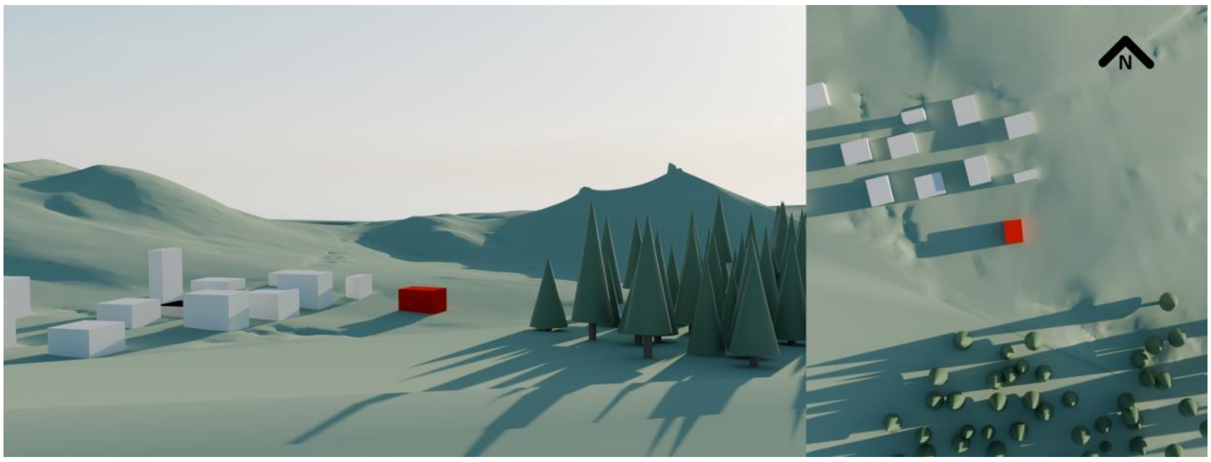


FIG 56: CONTEXT MODEL FOR CASE STUDY, MAXIMUM BUILDING VOLUME IN RED.

In order to showcase the flexibility of the developed approach, the building geometry on which the floorplans are conditioned will be automatically generated as well. Below the effect of various constraints are shown, further explanation provided in the figure description. Each sample contains a yellow area which represents the interior floorspace and a white area which represents an overhang of the roof. Additionally, there a small second story generated with each of these variants. These are not shown to improve readability.

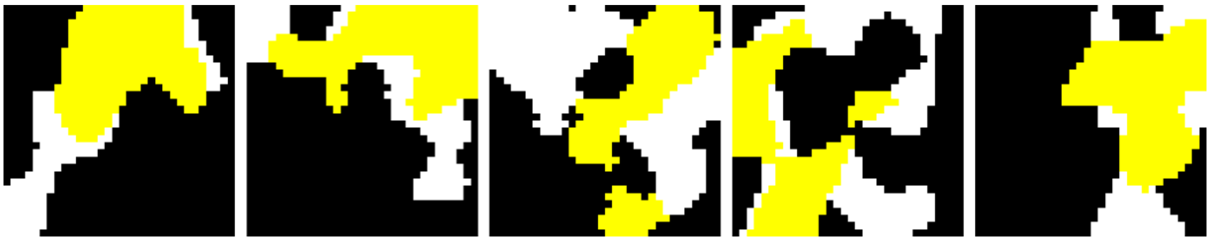


FIG 57: FIVE SAMPLES WITH A BY FIXED AREA CONSTRAINT. NOTICE THAT ALTHOUGH THE COVERAGE OF THE YELLOW IS REASONABLY CONSISTENT, THE SAMPLES HAVE VERY IRREGULAR SHAPES.

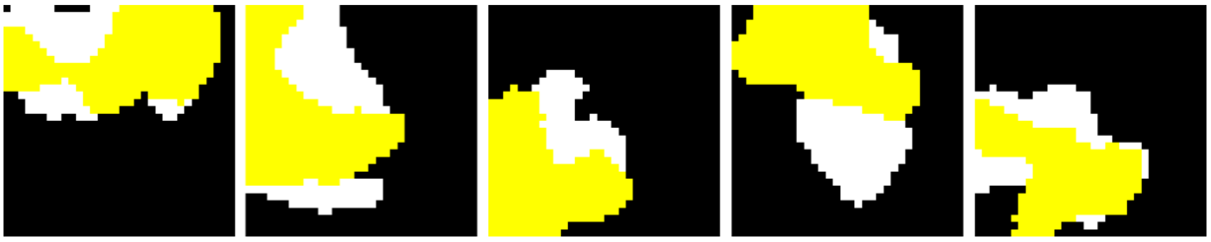


FIG 58: FIVE SAMPLES WITH AN ADDITIONAL COMPACTNESS CONSTRAINT. COMPACTNESS IS CALCULATED USING THE POLSBY-POPPER INDEX.

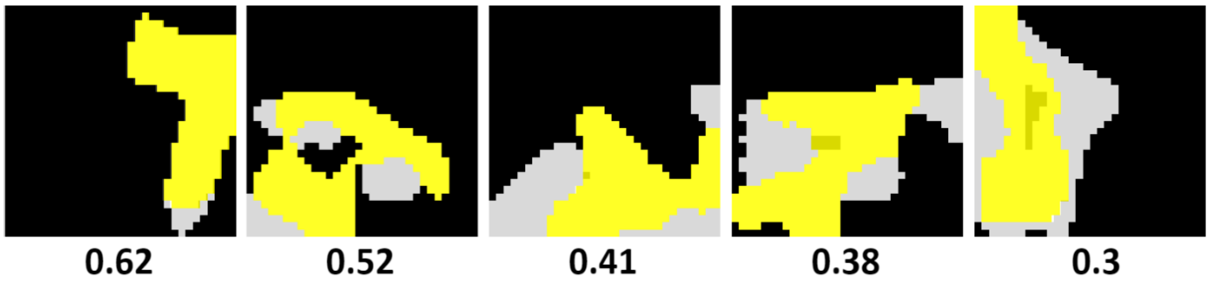
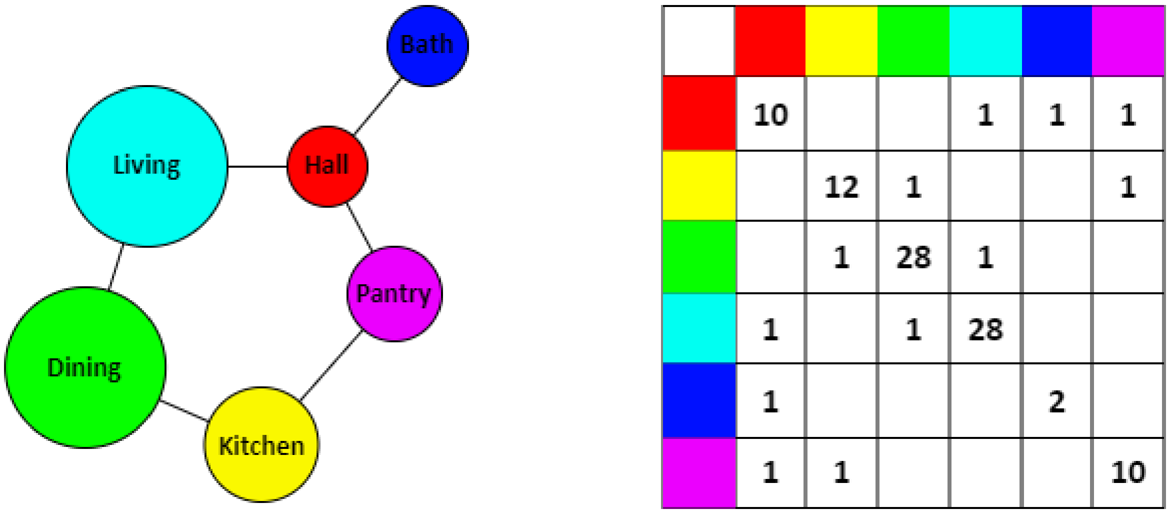


FIG 59: FIVE PARETO OPTIMAL SOLUTIONS IN THE TRADE-OFF MOST FLOOR AREA WITH LIGHT LEVEL > 3% ON THE LEFT AND LEAST DIRECT RADIATION ON THE RIGHT. THE REGION OF THE FLOORPLAN WHICH IS NOT PROPERLY LIT IS DARKENED YELLOW. THE FRACTION OF THE RADIATION THAT END UP ON THE INTERIOR IS WRITTEN BELOW. FOR THE AUTOMATED DESIGN SAMPLE 5 WILL BE TAKEN SINCE THE DAYLIGHT SATISFACTION IS HIGH AT HALF THE HEAT GAIN.

Based on this mask a floorplan may be generated, the desirable layout as well as its representative matrix can be seen in fig 60. Using this graph and the mask generated in the previous step a floorplan may be created. Note that for demonstration purposes the collaborative element of the system will not be used, instead one of the five samples presented will be picked. These samples can be seen in fig 61. The same procedure is then repeated with the second floor which only has three rooms.



	🟥	🟨	🟩	🩵	🟦	🟪
🟥	10			1	1	1
🟨		12	1			1
🟩		1	28	1		
🩵	1		1	28		
🟦	1				2	
🟪	1	1				10

FIG 60: DESIGN GRAPH AND MATRIX.

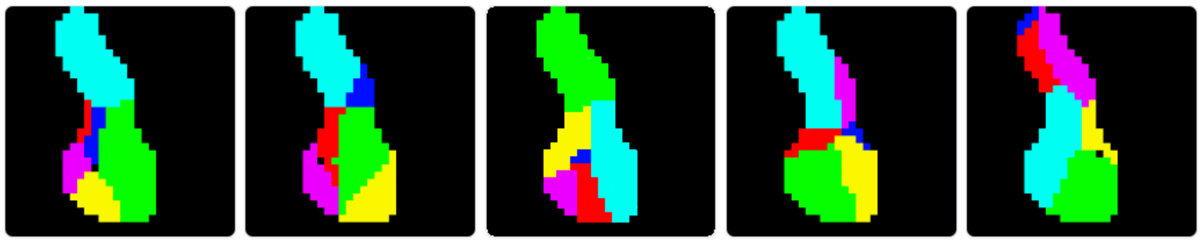


FIG 61: FLOORPLAN LAYOUTS.

Using the floorplans and roof shapes a 3D model can be constructed, for demonstration purposes floorplan number three will be used. The façade composition will be assigned based on the function in the floorplan behind it. The living, dining, and bedroom will have a full glass façade. The kitchen and pantry will have windows between 80cm and 160cm and the bathroom a window above 180cm. The resulting 3D model and its location in the context can be seen in fig 62.

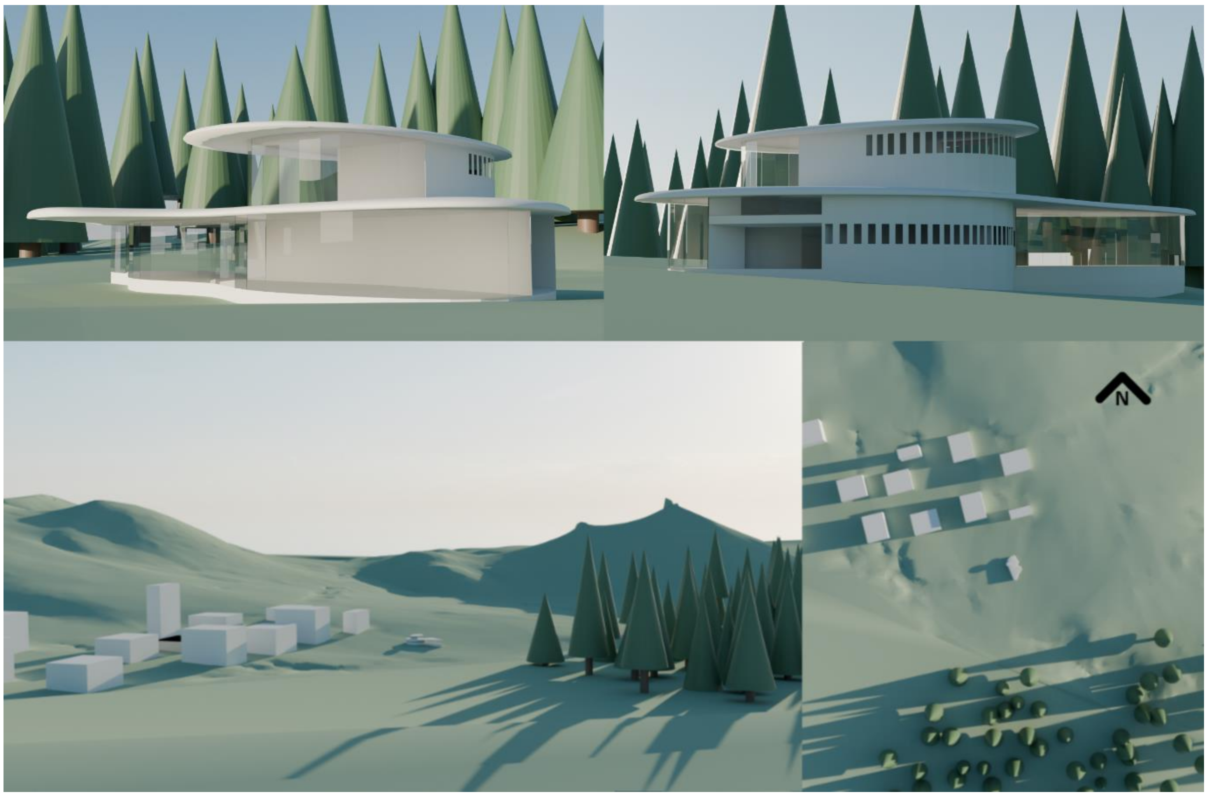


FIG 62: FINAL DESIGN AS PICKED FROM FIVE GENERATED OPTIONS.

10. Discussion.

In this work a deep learning algorithm was established that can correlate performance with geometry dynamically and use this insight to inform modifications. Through this method computational design optimization may be performed without the specification of design variables even if there is no training data available. This does come with the limitation that this only possible if a set of actions can be described that will enable the full manipulation of the solution space. Interestingly, this technique could be applied to different data representations such as graph since the model operates in latent space.

Alternative data modes can be facilitated by using a different VQ-VAE architecture that is suitable for the envisioned data. By locking the codebook and mapping the different data formats representing the same information to the same vector, the system could be made indifferent to data modality. For example, the index to the latent vector representing the word "cat" could be the trained to be identical to the one representing the image of a cat. This may make collaboration with the system more intuitive through enabling multi modal communication.

The probabilistic nature of the algorithm may cause complex and unpredictable sample failure states, which are traditionally fixed through careful shaping of the reward function. However, this is only possible if the failure can be quantified, which is not always feasible. To account for this, mechanisms are implemented that enable the user to meaningfully control the agent through visual means such as graph similarity and masking. A limitation of this is that a process driven by this technique may never be able to be fully automated. Additionally, the reliance on meaningful reward signals exacerbates the problem of expensive numerical evaluations since it relies solely on meaningful reward signals. Therefore, the development of an accompanying surrogate model is an absolute necessity.

The algorithms used for performance evaluation, specifically the daylighting and the carbon footprint, are oversimplified. This was done to speed up development and training of the agent, which is the main topic of this project. The simplified performance evaluations may also oversell model performance, more complicated relationships between the samples and their scores may be more difficult to predict. Even with the simplified evaluations, the complexity of the model and its instability during training limited the level of performance that could have been achieved. Since not all avenues of improvement have been exhaustively considered, it is very likely a future version will perform significantly better. Additionally, the multi-channel application of the network has for this same reason not been properly tuned. Therefore, the genetic algorithm is used to explore the VQ-VAE decoder latent space as a placeholder when applied to multi-channel problems.

An aspect of the work that has not received as much attention as intended is the ensemble approach for surrogate modelling. However, the performance of the score prediction algorithm used to evaluate the observer and interpreter models was surprising. Therefore, it may be possible to construct a surrogate model that can adapt itself to specific problem environments. It was shown that after being shown fifty samples the transformer could predict the score of new samples with an error margin below ten percent. Knowing this it may be very interesting to experiment with a model as shown in figure 63. In this model, the developed interpreter module may be used to extract information from reference simulation results. These results will then be sent to a cross-attention transformer to transfer the information to a new sample to predict its corresponding simulation results. Leveraging the concept of model ensembles, it can be argued that when multiple independently trained transformers produce similar predictions, these predictions are likely to be accurate. Therefore, when results are similar, they can be treated as if they were actual simulation outcomes. If the results differ significantly, a new simulation can be conducted, and their results added to the references.

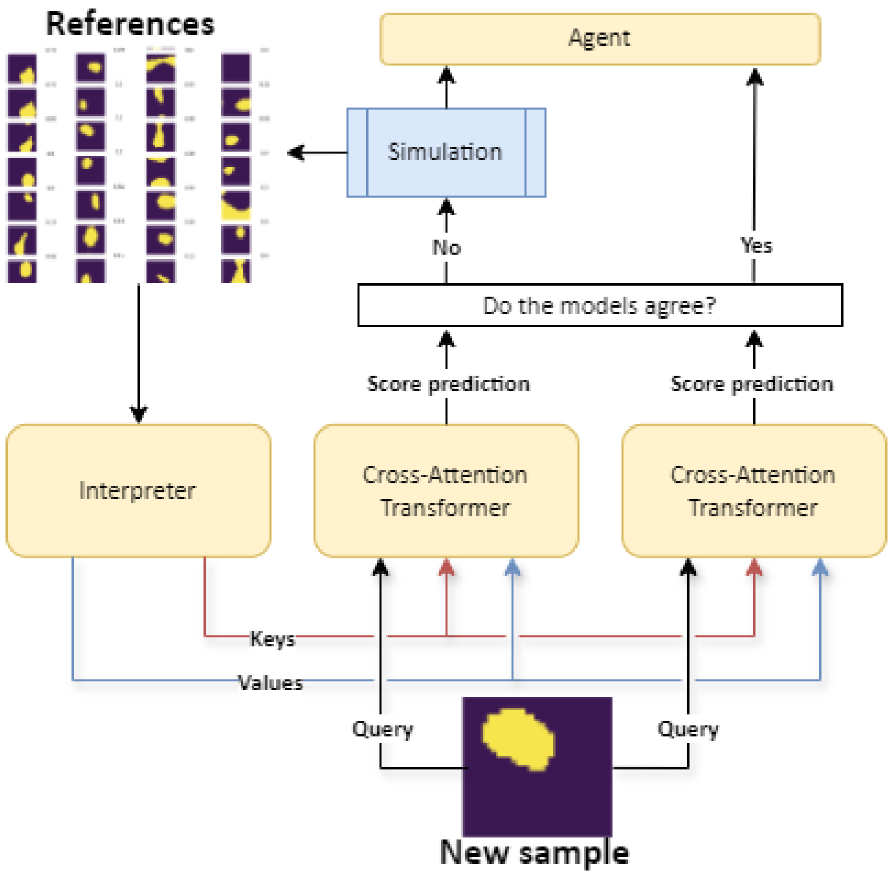


FIG 63: ADAPTIVE SURROGATE MODEL BASED ON ENSEMBLES.

The question could be asked, is the developed methodology still reinforcement learning. The proximal policy optimization is vital since this allows the method to work when the reward function is not differentiable, which for both performance indicators is not the case. Additionally, the solution still relies on solving a Markov Decision process. The agent takes an action which causes the environment to transition from state$_t$ to state$_{t+1}$ for which the agent receives a reward. However, typically the policy tries to maximize the reward and the discounted future reward, in this case there is no future reward. Therefore this solution could be categorized as a special case of reinforcement learning where the discount factor is zero.

11. Conclusion

In this report a methodology of generating floorplans using deep reinforcement learning has been developed to answer three main questions, their answers are summarised below:

11.1 How can deep reinforcement learning (DRL) agents be used to assist designers in creating performance-informed floorplans?

It was shown that the agent can correlate performance with geometry dynamically. Dynamic implies that the correlations are not learned during training. Instead, the agent learns to extract them based on a limited number of examples, diminishing returns were already visible after showing more than fifty of these examples. Based on these correlations, the agent can inform its own actions in order to modify the examples such that their assigned performance is improved.

By allowing the user to include meaningful controls such as graph similarity and masking, the agent can be effectively guided towards desirable solutions in a visual way. This also allows the compliance with user preference or desired aesthetics, criteria for which there are no objective measurement.

11.2 How can a collaborative, performance aware DRL agent overcome data scarcity and the creativity gap?

The agent uses actions to alter samples and through this method builds its own 'dataset' of samples which is used to train the observer model. Because of this approach no preexisting dataset is required at all, therefore data scarcity is not a problem. This lack of a dataset also means that there is no creativity gap. Since no distribution of training data is learned, all solutions are essentially novel. The bias in the model can only come through the nature of the actions and the performance evaluation.

An unintended byproduct of the model is its observer. This module, implemented as a VQ-VAE will learn to represent an ever-increasing amount of data throughout the use of the model. As a result, when trained, this VQ-VAE decoder can also be used in other algorithms to generate data. An example of this is the genetic algorithm used in this work, the application of this method is only possible because the agent trained the required decoder. Interestingly, this genetic algorithm currently outperforms the agent which trained it, this will however likely change.

11.3 Can the challenges associated with computational cost be alleviated using ensemble models?

The score predictor network has been shown to be able to predict scores for unobserved samples based on a limited number of examples. Although this work has not yet yielded a fully developed surrogate model based on this technique, a potential avenue for this has been discussed. It should be reiterated that a way to reduce the computational demands for meaningful reward signals is vital.

12. Reflection.

The reflection will include four main topics: Firstly, the academic and societal relevance of the project. Secondly, the research methodology and process. Thirdly, personal development. Fourthly, future work.

12.1 Relevance and Impact.

In this work, a transferable framework has been developed that facilitates the integration of various engineering and design disciplines. Central to this framework is an agent that can interrogate various aspects related to sustainability and architectural design in a collaborative way. The highly adaptive nature and independence from training data allows for the integration of a wide range of measurable design characteristics beyond the ones demonstrated in this work. This integration forms the essence of the building technology master which aims to bridge the gap between designers and engineers.

This work also perfectly aligns with the goal of Deep Generative Design which is to deliver design solutions that exceed the limits of traditional design methodologies. The agent has demonstrated an arguably superhuman ability to correlate geometry with performance indicators and has to a lesser extend been able to use its understanding of this correlation to inform design decisions. When improved further, this technique may lead to a paradigm shift in the field of design informatics since it drastically increases the speed and intuitive controllability that current techniques lack.

This is not only academically interesting but may also have an effect on industry and society. Due to the escalating complexity of design, designers are becoming underequipped to solve today's design challenges. This results in various problems including long design processes, suboptimal energy performance and an unnecessarily high carbon footprint. A new generation of design tools revolving around human-machine collaboration may have the ability to alleviate these problems. This would enable engineers and designers to exercise their craft more effectively and reduce the environmental and financial costs of humanities building stock.

12.2 Research methodology.

Although I was already quite familiar with deep learning, a significant amount of time was spent reading literature to identify suitable system architectures. This research was quite focussed since the goal was quite clear, I specifically wanted to develop a methodology that would be independent of data availability. The reason for this was that work performed by previous students showed that data scarcity hindered the applicability. Additionally, I had encountered this problem myself during CORE and Computation Intelligence.

The literature research got me interested in model-based reinforcement learning approaches such as DreamerV3 which have a very high degree of complexity. This was both very insightful and time consuming since it resulted in refining the interaction between multiple independent neural networks, which is very challenging even on powerful hardware. Later when I started to get a better grip on the theory and problem specific requirements, I was able to simplify the system architecture significantly. This was only possible by continuously switching between small experiments and literature research.

One aspect that suffered from this was the performance evaluation, which was originally intended to be a significant part of the project. However, with some encouragement from my supervisors I was convinced to focus on the framework itself. The reasoning being that more sophisticated performance evaluations can be added in the future without requiring fundamental changes to the system architecture. Therefore, the current project of should be regarded as a proof of concept instead of a production ready implementation. Another effect of the high degree of novelty and complexity is a certain lack of refinement. This is logical since having more variables inevitably leads to less time available per variable. Unfortunately, this reduced the performance, which makes the true potential difficult to estimate.

12.3 Personal development.

During this project I have learned a lot about the design of sophisticated deep learning methodologies which has been a very rewarding process. It quickly became obvious that timelines for the development of neural networks of this complexity are highly unpredictable. The main culprit for this is the disconnect between theory and practice. There may be approximate directions in the form of comparable work in literature. However, the only way to guarantee that a network architecture works for an untested application is to implement and tune it, which is very time consuming. Additionally, if the design turns out to not work as expected it is immediately back to the drawing board.

The biggest lessons I have learned are however not related to deep learning but to time management and task prioritisation. I became quite good at judging relative importance and expected timelines of tasks. This became increasingly important when the development of the agent started to lag behind the intended schedule due to the previously discussed unpredictability. These situations would often lead to questions such as, what do I really need right now? or what exactly am I trying to achieve? And these questions most often led to the biggest changes, such as the removal of the critic network or the implementation of the genetic algorithm to get the desired outcomes in a different way.

12.4 Future development

This workflow can be divided into four main themes: analysis, generation, interaction, and processing. Were this report focused mostly on generation and interaction, the connection to future analysis workflows is vital. An interesting avenue for development would be to create an application programming interface (API) to convert the data generated using this technique to and from formats compatible with programs such as EnergyPlus, Radiance, and Rhinoceros. This would not only allow design automation informed by more accurate analysis but also facilitate the development of future surrogate models that are compatible with this technique and each other. Another direction would be a processing workflow to turn the generated data into a design that is buildable. This could be thought of as a parametric model where some parameters are replaced by the geometries generated using the workflow discussed here. This also plays into the analysis, since more detailed models would lead to more accurate performance estimates.

The most promising direction however is to experiment both on the existing framework and the adaptive surrogate model as proposed in the discussion. I am convinced that with more work both architectures may become very useful assets to engineers in various fields.

References.

BREAAM-NL. (2024, 04 29). *credit: daglichttoetreding-10*. Retrieved from BREEAM-NL: https://richtlijn.breeam.nl/credit/daglichttoetreding-10

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., & Dhariwal, P. (2022). Language Models are Few-Shot Learners. *Advances in neural information processing systems*.

Chiu, M.-L. (2002). An organizational view of design communication in design collaboration.

Clavera, I., Rothfuss, J., Schulman, J., Fujita, Y., Asfour, T., & Abbeel, P. (2018). Model-Based Reinforcement Learning via Meta-Policy Optimization. *Conference on Robot Learning*.

Deb, K., Pratap, A., & Agarwal, S. (2002). A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION.*

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., . . . Houlsby, N. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv preprint*.

Du, T., Jansen, S., Turrin, M., & van den Dobbelsteen, A. (2020). Effects of Architectural Space Layouts on Energy Performance: A Review. *Sustainability*.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., . . . Levine, S. (2018). Soft Actor-Critic Algorithms and Applications. *arXiv preprint*.

Hafner, D., Pasukonis, J., Ba, J., & Lillicrap, T. (2023). Mastering Diverse Domains through World Models. *arXiv preprint*.

Ho, J., Jain, A., & Abbeel, P. (2020). Denoising Diffusion Probabilistic Models. *Advances in neural information processing systems.*

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*.

Huang, Z., Heng, W., & Zhou, S. (2019). Learning to Paint With Model-based Deep Reinforcement Learning. *Proceedings of the IEEE/CVF international conference on computer vision*.

Kingma, D., & Welling, M. (2013). Auto-Encoding Variational Bayes. *arXiv preprint*.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature 521*, 436-444.

Li, W., Luo, H., Lin, Z., Zhang, C., Lu, Z., & Ye, D. (2023). A Survey on Transformers in Reinforcement Learning.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., . . . Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., . . . Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv*.

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., & Xie, S. (2022). A ConvNet for the 2020s. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, (pp. 11976-11986).

Martins, M., & Ning, A. (2021). *Engineering Design Optimization.* Michigan.

Matthias Standfest, M. F.-L. (2022). Swiss Dwellings: A large dataset of apartment models including aggregated geolocation-based simulation results covering viewshed, natural light, traffic noise, centrality and geometric analysis. *Zenodo*.

Nauata, N., Chang, K.-H., Cheng, C.-Y., Mori, G., & Furukawa, Y. (2020). House-GAN: Relational Generative Adversarial. *Computer Vision–ECCV 2020.*

Pavlidou, S. (2022). *Deep Generative Design: A Deep Learning Framework for Optimized Shell Structures.* Delft: TU Delft.

Regenwetter, L., Heyrani Nobari, A., & Ahmed, F. (2022). Deep Generative Models in Engineering Design: A Review. *Journal of Mechanical Design*.

Ren, M., Chen, N., & Qiu, H. (2023). Human-machine Collaborative Decision-making: An Evolutionary Roadmap Based on Cognitive Intelligence. *International Journal of Social Robotics*.

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015). High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv preprint*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms.

Shibhansh, D., Qingfeng, L., & Rupam, M. (2023). Overcoming policy collapse in deep reinforcement learning.

Staudemeyer, R. C., & Morris, E. R. (2019). Understanding LSTM -- a tutorial into Long Short-Term Memory Recurrent Neural Networks.

Sterrenberg, A. (2023). *Deep Generative Design: A Deep Learning Framework for Optimized Spatial Truss Structures with Stock Constraints.* Delft: TU Delft.

van den Oord, A., Vinyals, O., & Kavukcuoglu, K. (2017). Neural Discrete Representation Learning. *Advances in neural information processing systems*.

Van Dooren, E., Boshuizen, E., Van Merrienboer, E., Asselbergs, M., & Van Dorst, M. (2013). Making explicit in design education: generic elements in the design process.

van Oranje-Nassau, W. A. (2023, 09 07). *Bouwbesluit 2012.* Retrieved from Rijksoverheid: rijksoverheid.bouwbesluit.com

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . Polosukhin, I. (2017). Attention Is All You Need.

Villegas, R., Babaeizadeh, M., Kindermans, P.-J., Moraldo, H., Zhang, H., Taghi Saffar, M., . . . Erhan, D. (2022). Phenaki: Variable Length Video Generation From Open Domain Textual Description. *International Conference on Learning Representations.*

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., & Chung, J. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature 575*.

Yayasinghe, A., Orr, J., Ibell, T., & Boshoff, W. P. (2021). Comparing the embodied carbon and cost of concrete floor solutions. *Journal of Cleaner Production*.

Zhang, K., Yang, Z., & Basar, T. (2021). Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms. *Handbook of reinforcement learning and control*.

Appendix.

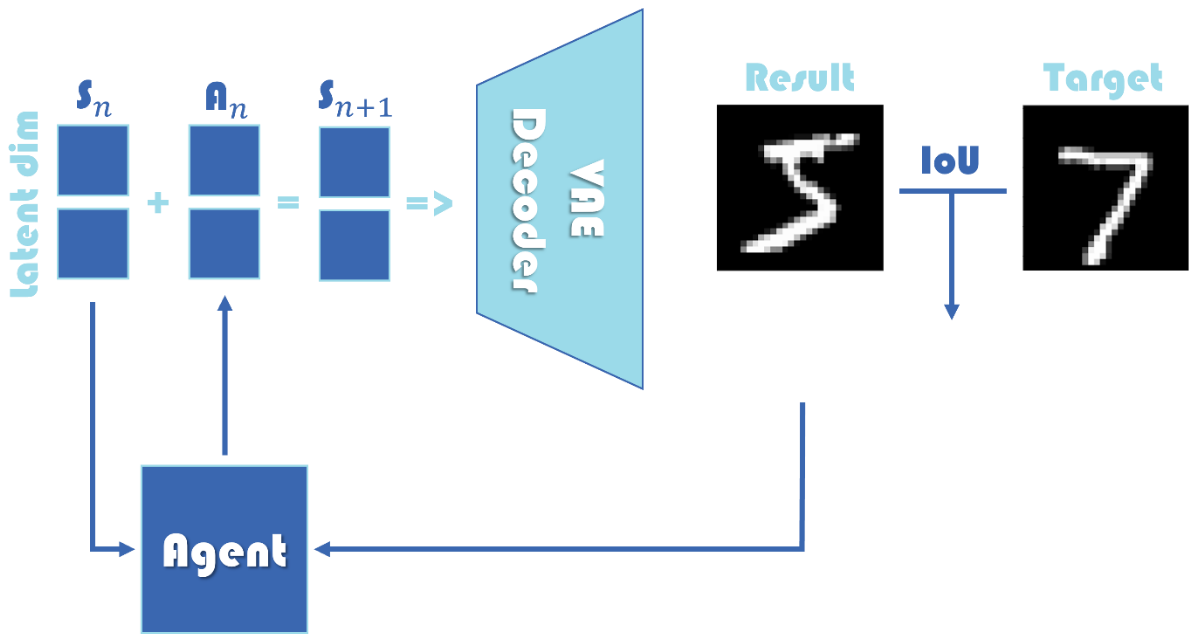


FIG 57: NETWORK DESIGN VARIANT, DROPPED BECAUSE THE EXPRESSIVENESS OF THE DECODER WAS FOUND TO BE INSUFFICIENT.

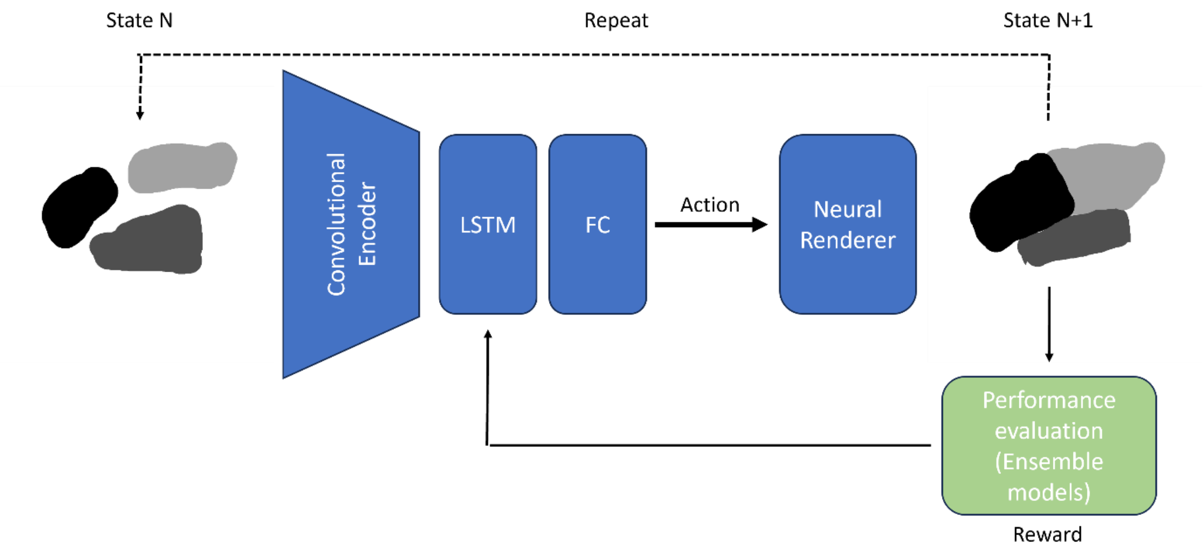


FIG 58: NETWORK DESIGN VARIANT, FURTHER UPGRADED SINCE THERE WAS NO GOOD WAY TO TRAIN THE ENCODER IN THIS SETUP.